ES Final Project: Blackjack Counter
Project Report

Lennart Schulze
ls3932

Joseph Han
jh4632

Michael Ozymy
mjo2156

Contents

1 Introduction

1.1 Abstract

We propose to use the combined software-hardware nature and extensive peripheral connectivity of FPGAs to develop a computer-vision-based card counting mechanism for the game blackjack. To this end, we connect the Field Programmable Gate Arrays (FPGA) to a camera that observes the playing field. We then implement a computer vision model, that is a Convolutional Neural Network (CNN), on the Hard Processor System (HPS) to recognize the state of the game by detecting the type of cards.

1.2 Motivation

- FPGAs improve throughput and parallelize communications with peripherals that require many repetitions and limited protocols. HPS can handle complicated and low-throughput control flows.
- A combination of the FPGA and the HPS to achieve card-recognition demonstrates efficient low-level hardware interfacing and flexible, powerful library support from the Linux system.

1.3 Objectives

- Interface an OV7670 camera through GPIO ports to capture images of playing cards.
- Control the SDRAM to enable storage of the image stream captured by the camera.
- Interface the VGA to display the images stored in SDRAM.
- Configure the FPGA to bridge data with HPS through the Avalon Bus, allowing for efficient processing of the image data.
- Use a forward pass CNN algorithm to classify the cards in the image data.
- Apply the classified card data to improve the odds of winning at playing blackjack.

1.4 Work Flow and Useful Resources

We followed the following milestones to engineer our system.

1. **Implementation of CNN and standardization of image processing between training and expected test data and select hardware for I/O**
   - Implement CNN in python.
   - Training and pre-processing: used a pre-trained model as a prototype.
   - Obtained the OV7670 camera breakout chip for image capture and a VGA display for viewfinding and debugging.
2. **Interfacing with camera and VGA hardware, configuration of embedded FPGA memory and SDRAM**
   - Physical connections of the camera module to the GPIO port on the FPGA according to the physical GPIO pinout available online at `https://dev.to/lambdamamba/using-the-gpio-for-external-inputs-and-outputs-for-fancy-fpga-projects-2697` and verified the pinout by generating clock frequencies, logic 1s and 0s on specific GPIO pins and measuring the waveforms on an oscilloscope.
   - Double checked .tcl file mapping of SystemVerilog wires to physical pins on the FPGA.
   - Adapted Xilinx implementation of camera, SDRAM and VGA interfaces from `https://github.com/AngeloJacobo/FPGA_OV7670_Camera_Interface` to work with Intel-ALtera IP Cores for clock frequency generation using Phase Locked Loop (*PLL*) and phase difference generation using *ALTDDIO* IP cores.
3. **Hardware-software interface using the Avalon bus, training and forward pass**
   - Created hardware responder named "img_reader" to Avalon in SystemVerilog. Used Platform Designer to generate img_reader conduit. Generated device tree files.
   - Memory-mapped img_reader device, created ioctl function to handle data transfer from Avalon bus to kernel space and then to user space. Used File I/O in Linux to store captured image data
4. **Integration of CNN with hardware-software pipeline and captured image data**

- Verified saved binary image data using `https://rawpixels.net/`, trained CNN on 24 captured images for each of the 52 poker cards.
- Configured the 16.04 version of Ubuntu based on `https://pcplanet.ca/how-to-install-python39-on-ubuntu/` to run python3 to execute the CNN on the HPS.

2 Algorithms

The algorithms consist of convolutional neural network to classify images from the camera. Cropping, down-sizing, and filter-based methods are used to prepare a single image for classification by the CNN on the HPS.

Only the prediction of new data points on the trained CNN model (forward pass) will be performed on the SoC. The training itself (backward pass and gradient descent) of the model happened on a cloud-hosted GPU previously.

2.1 Convolution

- Matrix convolution describes the operation of transforming a given input matrix into an output matrix by sliding a smaller matrix with fixed components, referred to as kernel, over the image and applying a matrix element-wise multiplication between the window of the input and the kernel matrix, which both have the same size.
- At each step, an element-wise multiplication is performed and the resulting matrix is reduced to a single value via adding all the values, which is assigned as the value of the output matrix at that position.
- This is repeated over the entire input matrix by beginning in the top left most window, computing the value for the top left element in the output matrix, and then moving down in an Z shape to the bottom right most window, that is for each row right until the end of the row followed by moving down one row.
- The size of the output matrix is thus determined by the kernel. Padding around the input matrix can be applied to enforce desired output sizes.
- The stride determines how fast the window moves, that is by how many columns it moves right or rows down per iteration, where the standard is one.

Formal definition:

The element of the output matrix O with indices x, y is defined as:

$$O[x,y] = (I * K)[x,y] = \sum_{-N \le i \le N} \sum_{-M \le j \le M} I[x-i, y-j]K[i,j] \tag{1}$$

Where I is the input matrix and K is the convolution kernel with shape (N, M), per convention (note, however, the symmetry of the convolution operation).

Algorithm:

A simplified algorithm disregarding padding and stride looks like:

```
Def convolve(I, K):
    s = size(K)
    O = [][]\\
    For x in I:
        For y in I[x]:
            O[x,y] = sum_elements(multiply_elementwise(I[x:x+s, y:y+s], K))
    return O

\\\\
```

2.2 Data normalization

Data normalization refers to the operation of mapping the range of given data into a new range. We normalize the received images from the RGB888 space, with values ranging from 0 to 255 inclusive into the range -1 to 1 inclusive to prevent the issue of vanishing or exploding gradients [cite!!!] Specifically, we apply max normalization:

$$x_{new} = \frac{x_{old}}{\max'_x x'} \tag{2}$$

Followed by z normalization:

$$x_{new} = \frac{x_{old} - \mu}{\sigma} \tag{3}$$

where we define $\mu = \sigma = 0.5$.

2.3 Convolutional Neural Network

- A CNN is a special type of an artificial neural network, which is a universal function approximator.
- Three types of layers can usually be found in a CNN:
  - Fully-connected layers, where each unit in the current layer is connected to every unit in the next layer via a weighted sum of the current units with trainable weights.
  - Convolutional layers, where the units of the current layer are convolved with a trainable kernel matrix.
  - Pooling layers, where the dimensionality of the vector of the current layer is reduced in the next layer, for instance via taking the mean of the pixel values in a given window that slides over the image.
- The layers are connected via non-linear activation functions, such as Relu.
- The output is generated by applying the layers and activations in order starting from the input.
- Depending on the task, the last activation function is chosen.

In our case, the input is a flattened vector corresponding to pixel color values and the output is one of the 52 distinct cards in blackjack. This represents a multiclass classification problem, for which reason the last activation function is chosen to produce a probability distribution over the 52 cards, from which

Formal definition:

A CNN F, acting on the input vector x can be described as:

$$F : X \rightarrow Y \tag{4}$$

$$F(x) = (f_n \circ ... \circ f_2 \circ f_1)(x) \tag{5}$$

Where $Y = [0, 1]^{52}$ the 52-dimensional one-hot encoded vector representing the probability of the class of the card; $X = \{0, ..., 255\}^{(W*H*3)}$, the flattened vector of pixel of dimension H*W*3 one pixel is a RGB value represented as triplet (R,G,B) in which each channel can assume a value in the range $[0, 255]$; and $f_i \in$ {linear, convolution, pool, activation}.

A linear activation is:

$$f(x) = Wx + b \tag{6}$$

A typical activation for intermediate layers is Relu: ntel.com/

$$f(x) = max(0, x) \tag{7}$$

A typical activation for the output layer for multi-class classifier CNN's is soft max:

$$f(x)[i] = \frac{exp(x[i])}{\sum_{j \in [|x|]} exp(x[j])} \tag{8}$$

Algorithm:

A simplified, forward-only CNN with given weights w can be described as

```
Def cnn(x,w):
  Layer_defs = "linear":..., "cnn":..., "avg_pool":..., "relu":..., "softmax":...
  Layers = [linear, relu, avg_pool, cnn, avg_pool,
    linear, relu, ..., softmax] : definition of cnn architecture
  y = x
  For i in 0,...,length(layers):
    y = layer_defs[layers[i]](y, w[i])
  return y
```

3 System Overview

3.1 Block Diagram

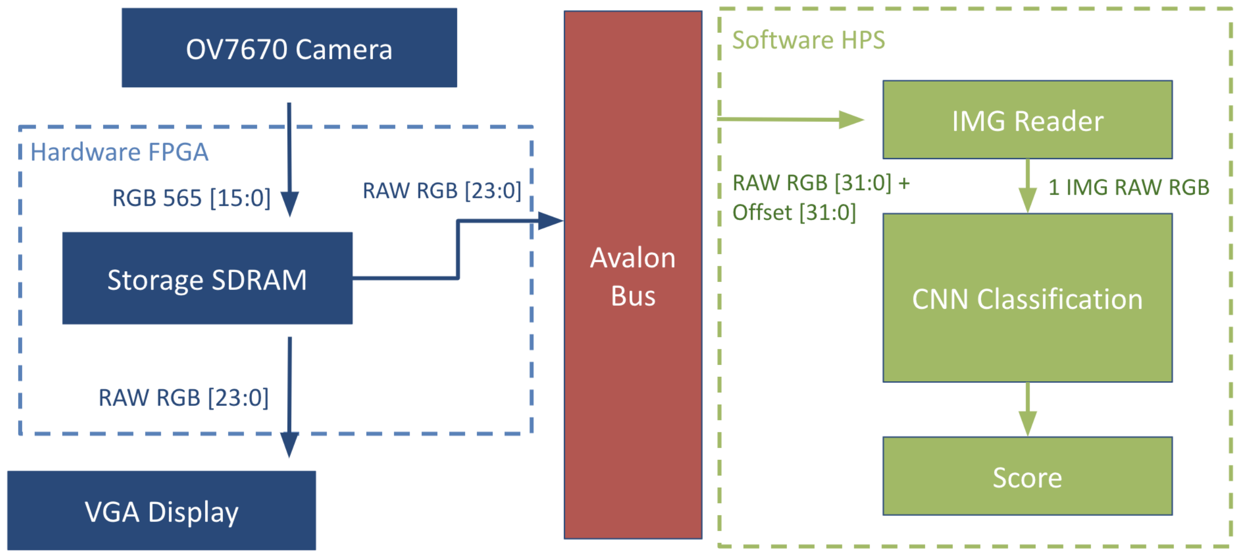


Figure 1: The path of every image with listed transfer sizes. Comes from the Camera Peripheral through the FPGA hardware and SDRAM across the Avalon Bus to the CNN Classification on the HPS.

- The data produced by the camera flows through several systems on the FPGA and HPS.
- The diagram in Figure 1 demonstrates a high level architecture that every byte of data from the camera traverses.
- The data first starts as 16-bit RGB565 format from the Camera into the GPIO pins.
- The data is stored into the 64MB of SDRAM on the FPGA.
- To display the captured image data the VGA display normalizes the data into RAW RGB 24bits and passes to the VGA Display.
- The same normalized data is passed to the HPS over the Avalon Bus.
- On the software side the img_reader driver uses ioread32() to parse an entire image 32 bits at a time as well as an offset to correct tearing to the CNN Classification.
- The CNN model runs a python script to classify the card in the image.
- Finally the classification is factored into the current count calculations and the score returns the count.

3.2 Hardware Map

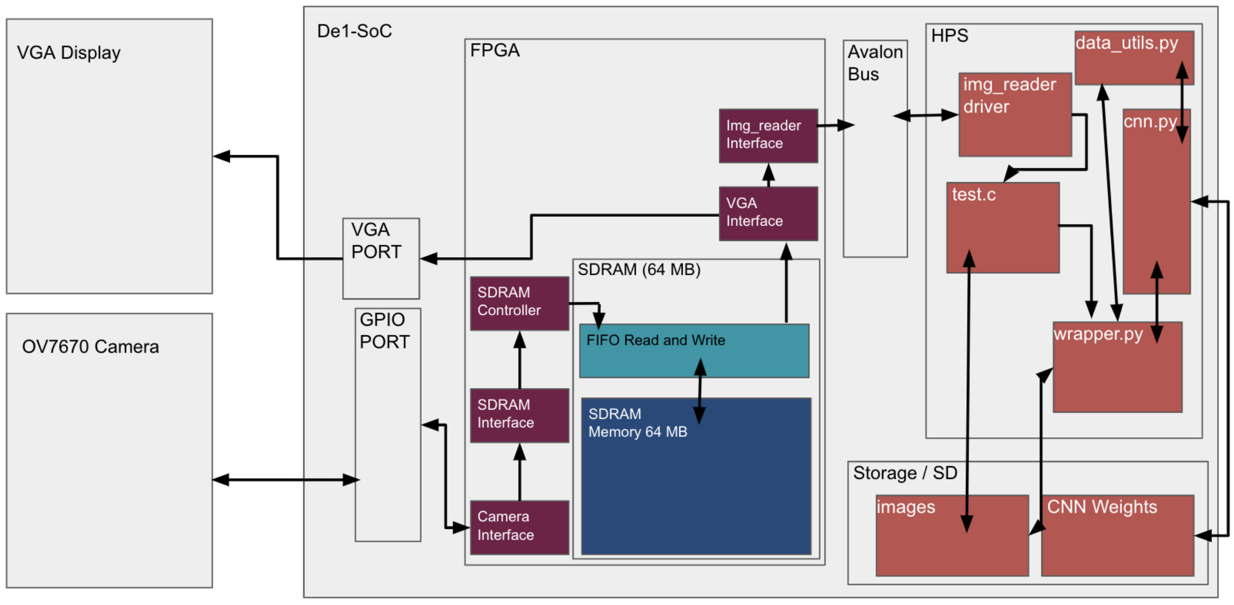


Figure 2: More detailed data path and directional connection specifications. Every major interface and connection that the data stream from the camera traverses are specified.

- The implementation of the data flow described in Figure 1 requires a number of complex interfaces that handle each individual set of wires and calculations. Figure 2 gives a more detailed physical representation of the data that includes every interface.
- First, the camera interface sets the SCCB protocol and interacts with the physical wire assignments of the GPIO ports.
- The Camera Interface module then passes the data to 3 SDRAM components: a SDRAM Interface, SDRAM Controller, and Asynchronous FIFO Buffer for both reading from and writing to the SDRAM.
- These components use burst memory access to maintain a complete image in the SDRAM despite the differences in clock frequency (24MHz camera to 25MHz VGA Display).
- Using the read functionality of the Buffer, the VGA Interface reads the images from the SDRAM and passes it to both the VGA port for Display and the IMG Reader interface.
- The img_reader interface acts as the responder to the Avalon Bus between the HPS and FPGA.
- By communicating with the IMG Reader driver on software, the images are sent in 32-bit sections across the Avalon Bus.
- On the HPS side, the driver passes the data to the user space file named test.
- To store the images and run the CNN, the user space program passes the complete RAW image file to both a wrapper function and SD memory.
- The wrapper function uses the image data from the user space as well as a set of pre-trained weights from SD memory to run the CNN classification and return a score.

3.3 File Hierarchy

- Figure 3 demonstrates the implemented system structure.
- The top most layer (soc_system_top.sv) maps all major data flow between FPGA and HPS as well as the paths to the physical wire assignments of the VGA and GPIO ports. It instantiates teh top_module.
- The Hardware side second layer, top_module.v, handles all secondary data follow that prepares the images to be sent over the Avalon Bus. For example, after each read from the camera the RGB 565 data flows through the top_module.v file to be stored by the SDRAM interface and subsidiary modules.
- For the HPS second layer, a quartus generated system file called soc_system.v handles all data passed to the Avalon Bus from soc_system_top.sv.

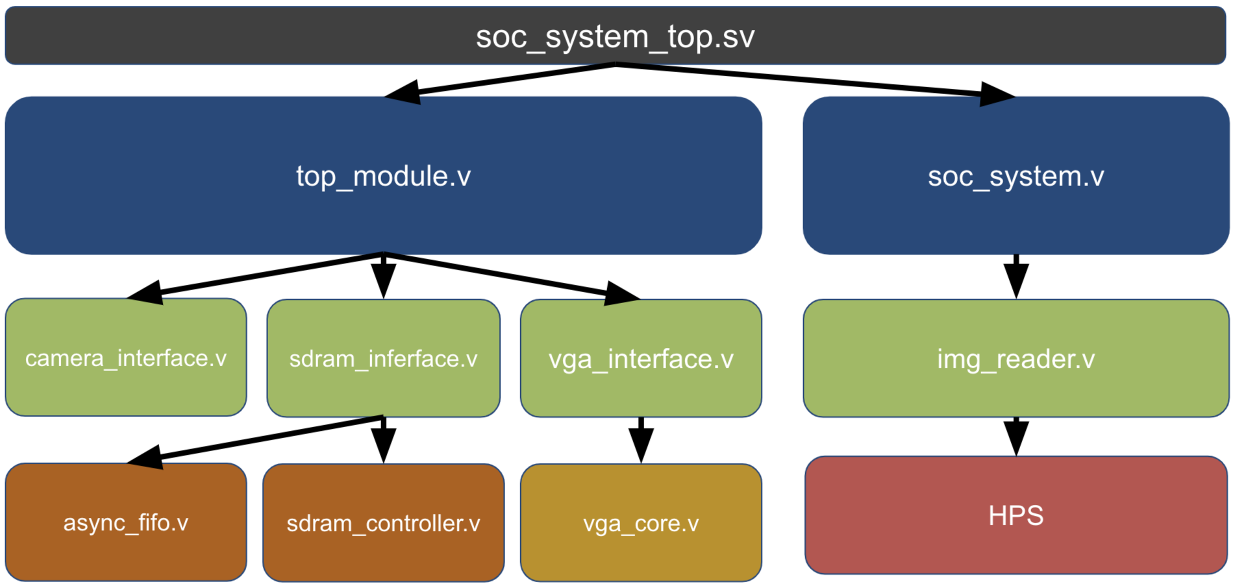


Figure 3: Hardware file hierarchy. All data flow in on the FPGA side of our implementation flow through these files. Each arrow represent a bidirectional communication path that each interface can communicate data through.

- On both hardware and software the third layer contains the primary interface modules that do the majority of the protocol management.
- Lastly, the fourth layer contains all controllers that contain added modules for more complex interfacing. On the HPS, this comprises the majority of the files including the CNN function and user space program.
- Not all branches contain a fourth layer, more simple interfaces such as the camera interface contain all required modules in the third layer.

4 Hardware Design

4.1 Camera Interface

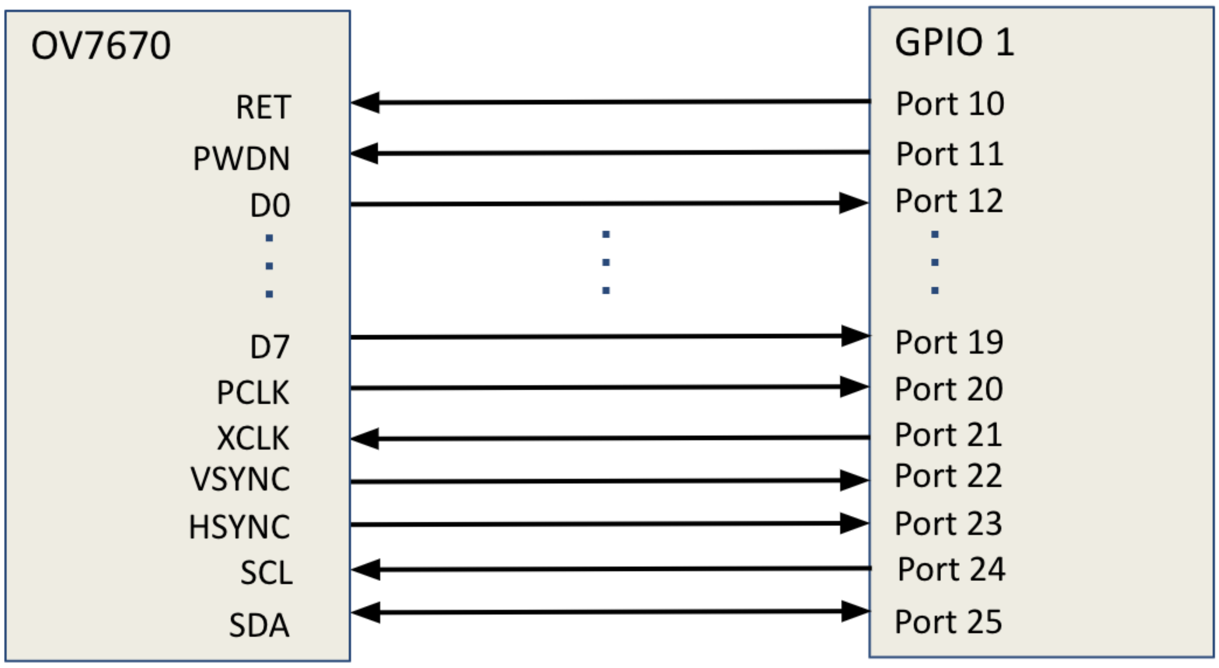


Figure 4: IO Control flow handled in the camera interface. The pin out specifies the physical GPIO pin out that connects the OV7670 camera.

Inputs:

- VCC (Power Supply Voltage): This is the input for the power supply voltage 3.3V, used to power the OV7670 camera module.
- XCLK (Pixel Clock): The XCLK input is used to provide the pixel clock signal, which synchronizes the output data from the camera module.
- PWDN (Power Down): This input is used to control the power state of the camera module. Pulling it low (ground) powers down the camera module.
- SDA (Serial Data): SDA is the input line for the serial data used in the I²C communication protocol.
- SCL (Serial Clock): The SCL input is the clock line used in the I²C communication protocol.

Outputs:

- D0-D7 (Data Bits 0-7): These outputs represent the pixel data output from the OV7670 camera module. Each data bit corresponds to a specific pixel value or color component.

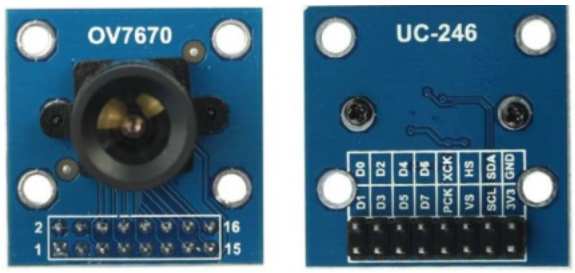
Camera Left	GPIO NUM	GPIO NUM	Camera Right
GND	GND	3.3V	PWER
SDA	25	24	SCL
HS	23	22	VS
XCLK	21	20	PLK
D6	19	18	D7
D4	17	16	D5
D2	15	14	D3
D0	13	12	D1
PWDN	11	10	RET

Table 1: Physical GPIO and OV7670 Camera Mapping.

- HREF (Horizontal Reference): This output indicates the start and end of each horizontal line of the image.
- VSYNC (Vertical Synchronization): VSYNC is the output that indicates the start and end of each frame or field in the image.
- PCLK (Pixel Clock): The PCLK output provides the pixel clock signal that synchronizes the output data from the camera module.

Camera to FPGA 2x20 GPIO Interface

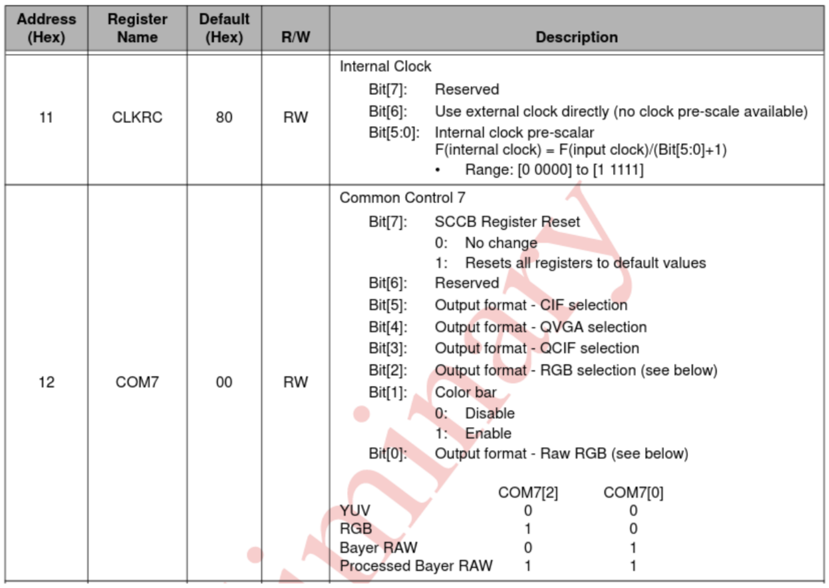
OV7670 Camera Breakout Board. The version in this project includes two additional pins for powerdown and reset



4.1.1 Protocol

SCCB is a variant of the I2C protocol which commonly uses pull up resistors to optimize control flow but this implementation has modified SCCB to exclude them. SCL and SDA are the two main pins for data control. SCL or Serial Clock synchronizes the FPGA and OV7670 to ensure consistent correct communication. The SDA or Serial Data pin signifies the end of each transmission as well as read or write state of the bus.

The modified version of SCCB uses a state machine with 3 primary states start, transmission, and stop. The start state initializes the bus and configures each register on the OV7670. Most notability the 12th register is configured to transmit video feed in RGB 565 format instead of the default YUV format. The transmission state is started on with a request for data from the FPGA and covers tranismision of a specified size and number of data. Lastly the stop state ends the transmission and waits for another request of data.



Address (Hex)	Register Name	Default (Hex)	R/W	Description
11	CLKRC	80	RW	Internal Clock Bit[7]: Reserved Bit[6]: Use external clock directly (no clock pre-scale available) Bit[5:0]: Internal clock pre-scalar F(internal clock) = F(input clock)/(Bit[5:0]+1) • Range: [0 0000] to [1 1111]
12	COM7	00	RW	Common Control 7 Bit[7]: SCCB Register Reset 0: No change 1: Resets all registers to default values Bit[6]: Reserved Bit[5]: Output format - CIF selection Bit[4]: Output format - QVGA selection Bit[3]: Output format - QCIF selection Bit[2]: Output format - RGB selection (see below) Bit[1]: Color bar 0: Disable 1: Enable Bit[0]: Output format - Raw RGB (see below) COM7[2] COM7[0] YUV 0 0 RGB 1 0 Bayer RAW 0 1 Processed Bayer RAW 1 1

4.1.2 Bandwidth

The OV7670 camera is designed to transmit a 640 x 480 resolution at 30 fps. The camera is rated to function at a maximum clock frequency of 24MHz. All data is passed out the 8 data pins. For this project RGB 565 is the output

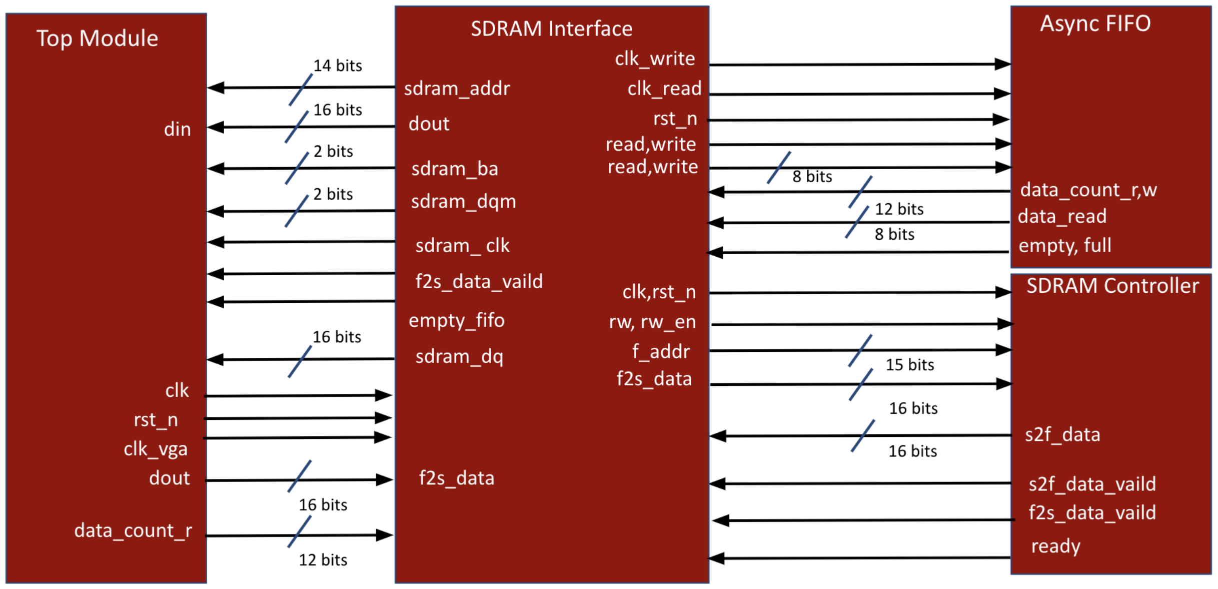


Figure 5: IO Control for the storage of data from the OV7670. The data stream from the camera is represented through the din variable from the top module. The SDRAM interface uses a SDRAM Controller and ASYNC FIFO to handle the storage of data in the 64MB SDRAM on the FPGA.

configuration format from the 8 data pins. RGB 565 is a 16 bit encoding that passes 5 bits of for the red value, 6 bits for the green, and 5 bits for the blue.

4.2 SD-RAM Interface

The SDRAM has the interface below. Outputs are connected to the off-chip SDRAM on the FPGA.

- **Inputs:**
  - pressed: a signal indicating a pause on writing to the SDRAM to get a still image in the SDRAM memory when transferring one frame of image to the HPS.
  - clk,rst_n: input clock of 100MHz and reset.
  - clk_vga, rd_en: VGA clock and read enable bit that enables read from the ASYNC FIFO.
  - data_count_r, data_count_w: counts of the number of data in the ASYNC FIFO buffers between the SDRAM and the VGA, and between the camera and the SDRAM, respectively.
  - f2s_data: 16-bit data from the ASYNC FIFO buffer written to by the camera module to the SDRAM
  - s2f_data: 16-bit data from the SDRAM to the ASYN FIFO that is read by the VGA interface

- **Outputs:**
  - s2f_data_valid, f2s_data_valid: flags for burst-mode memory transfer
  - ready: status register for when the SDRAM is ready for the next read or write
  - s_clk: SDRAM clock, same frequency as the input clock
  - s_cke: clock-enable bit, always high
  - s_ras_n, s_cas_n: row and column addressing strobe used to activate row/column addressing
  - s_we_n, s_cs_n: write enable and chipselect
  - s_addr: 13-bit SDRAM address
  - s_ba: Bank address
  - LDQM , HDQM: low-byte and high-byte masks, always low
- **Inout:**
  - s_dq: 16-bit data from/to SDRAM

4.2.1 Memory Usage

600 pages of 512 words are used, which correspond to the number of bytes in an image. Burst mode reading/writing from/to the SDRAM happens when the ASYNC FIFO for the camera is filled to 512 words, or when the ASYNC FIFO for the VGA has fewer than 250 words left.

4.3 ASYNC FIFO

An ASYNC FIFO is implemented between the camera module and the SDRAM and has the following characteristics:

- Read clock frequency: 100MHz
- Write clock frequency: 100MHz
- Read data: 16-bit camera data
- Write data: 16-bit data to the SDRAM
- Size: 1024 bits

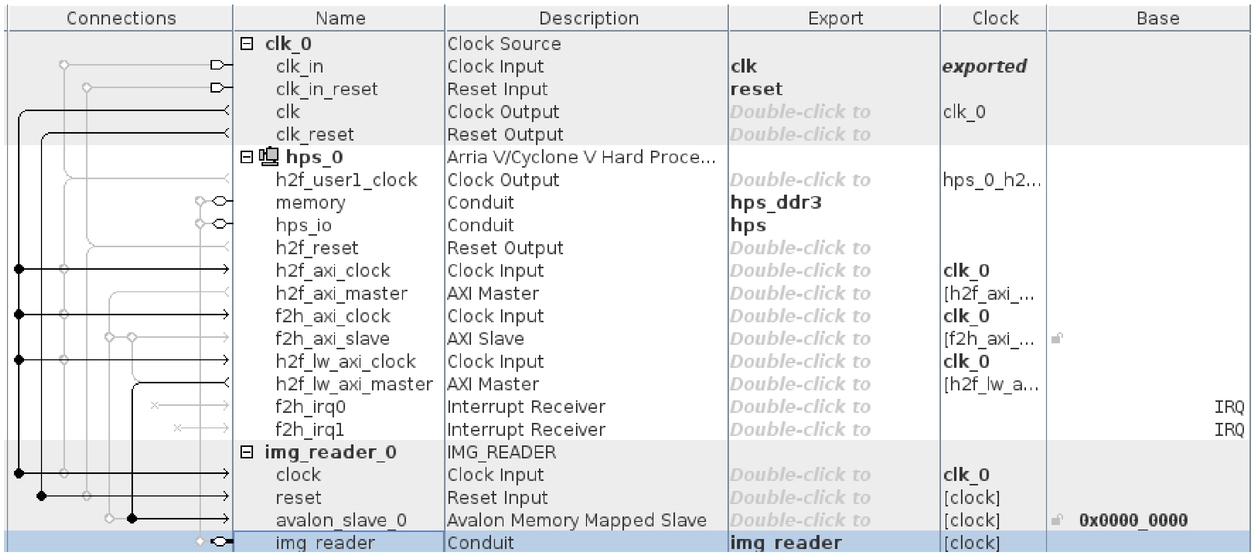
Note that the fact that both the input and output clocks are at 100MHz might appear confusing. The reason behind that is a Finite State Machine (FSM) is implemented in the camera interface and runs on a clock frequency of 100MHz. Only during the rising clock edges where all 16 bits from the camera is captured, the write enable bit is asserted high and a write to the FIFO buffer is completed. Therefore, the camera module is not writing to the ASYN FIFO at 100MHz but only when a pixel data becomes available, which is about 12MHz.

An ASYNC FIFO is implemented between the camera between the SDRAM and the VGA has the following characteristics:

- Read clock frequency: 25MHz
- Write clock frequency: 100MHz
- Read data: 16-bit to the top module where it gets normalized to RGB24-bit data that gets wired to VGA_R, VGA_B,VGA_B outputs
- Write data: 16-bit data from the SDRAM
- Size: 1024 bits

4.4 Avalon Bus

The HPS is the avalon master and a hardware module named img_reader is the avalon slave. Software on the HPS initiates all transactions. The slower LW-AXI bus is used.



Connections	Name	Description	Export	Clock	Base
	⊟ **clk_0**	Clock Source			
	clk_in	Clock Input	**clk**	***exported***	
	clk_in_reset	Reset Input	**reset**		
	clk	Clock Output	*Double-click to*	clk_0	
	clk_reset	Reset Output	*Double-click to*		
	⊟ **hps_0**	Arria V/Cyclone V Hard Proce...			
	h2f_user1_clock	Clock Output	*Double-click to*	hps_0_h2...	
	memory	Conduit	**hps_ddr3**		
	hps_io	Conduit	**hps**		
	h2f_reset	Reset Output	*Double-click to*		
	h2f_axi_clock	Clock Input	*Double-click to*	**clk_0**	
	h2f_axi_master	AXI Master	*Double-click to*	[h2f_axi_...	
	f2h_axi_clock	Clock Input	*Double-click to*	**clk_0**	
	f2h_axi_slave	AXI Slave	*Double-click to*	[f2h_axi_...	
	h2f_lw_axi_clock	Clock Input	*Double-click to*	**clk_0**	
	h2f_lw_axi_master	AXI Master	*Double-click to*	[h2f_lw_a...	
	f2h_irq0	Interrupt Receiver	*Double-click to*		IRQ
	f2h_irq1	Interrupt Receiver	*Double-click to*		IRQ
	⊟ **img_reader_0**	IMG_READER			
	clock	Clock Input	*Double-click to*	**clk_0**	
	reset	Reset Input	*Double-click to*	[clock]	
	avalon_slave_0	Avalon Memory Mapped Slave	*Double-click to*	[clock]	**0x0000_0000**
	img_reader	Conduit	**img_reader**	[clock]	

Figure 6: System contents and Avalon bus selection

4.4.1 Avalon bus responder

An avalon bus responder named "img_reader" is implemented. It works as follows:

- On the rising clock edge of the clock signal from the Avalon master, chipselect and read is asserted high by the master indicating an ongoing transaction.
- 24-bit-wide RGB wires are connected to the VGA interface where the RGB signals change on every VGA clock cycle. A 32-bit register named "offset_to_zeros" records with a reset value of 0 records the number of pixels read before vsync goes low indicating the start of a new frame. This records the offset of the starting RGB values from bottom-right of the screen.
- The master also selects an address, where an address of 0 accesses the RGB wires and an address of 1 accesses the offset register. The master sends an address of 0 for 640x480 clock cycles, during which the responder sends in a complete frame. It then sends an address of 1 for 1 clock cycle, and the responder sends the offset_to_zeros value.

- **Inputs:**
  - read, write, chipselect : master-asserted signals for toggling between modes and initiating transaction
  - writedata : 32-bit data from HPS
  - address : HPS pipeline
  - VGA RGB : 24-bit data from camera.
- **Outputs:**
  - readdata : 32-bit data to HPS. In case of RGB values, the least significant byte will be padded with 0.

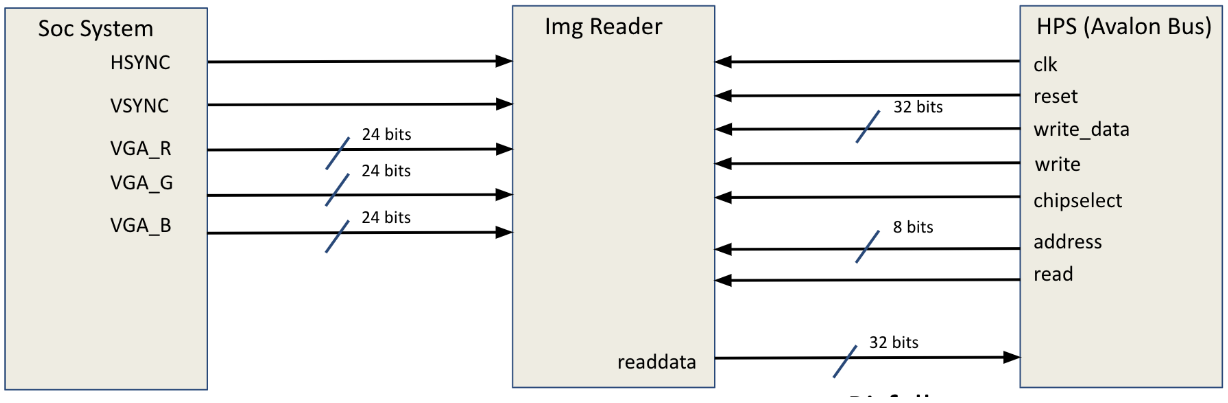


Figure 7: IO Control interface on the edge of FPGA and HPS over the Avalon Bus. The pin outs represent the connections between the hardware file hierarchy and HPS.

4.5 VGA Interface

- **Inputs:**
  - `clk`: Clock signal.
  - `rst_n`: Active-low reset signal.
  - `empty_fifo`: Input signal indicating if the FIFO is empty.
  - `din`: 16-bit input data.
- **Outputs:**
  - `clk_vga`: VGA clock output.
  - `rd_en`: Read enable signal.
  - `vga_out_r`: 5-bit red color component output.
  - `vga_out_g`: 6-bit green color component output.
  - `vga_out_b`: 5-bit blue color component output.
  - `vga_out_vs`: Vertical synchronization output.
  - `vga_out_hs`: Horizontal synchronization output.

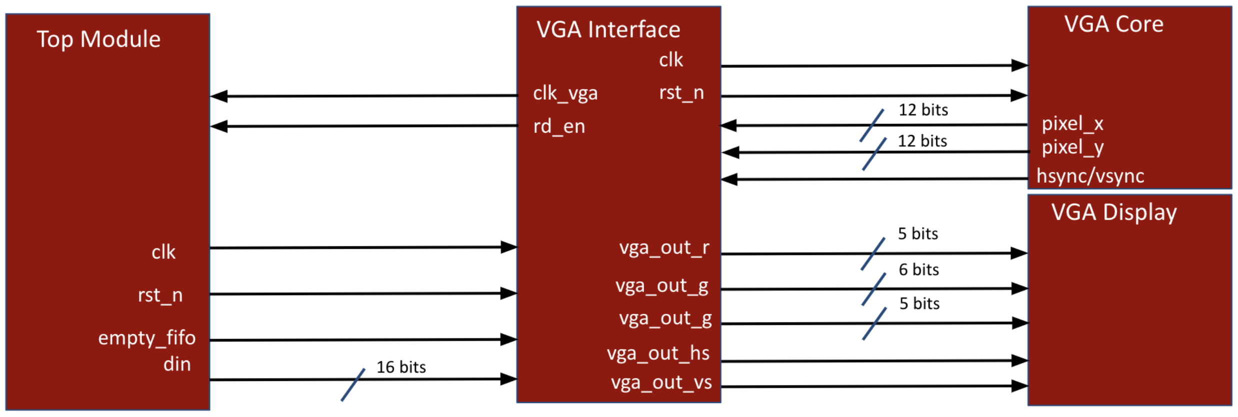


Figure 8: IO Control interface on the FPGA between top_module and vga_interface as well as its subsidiary dependent module vga_core.

4.5.1 VGA Core

The VGA interface controls the data flow from the SDRAM and ASYNC FIFO to output to the 640 x 480 60 fps display (25MHz). Because the camera is set to 30 fps and operates at 24MHz this interface controls the hsync and vsync to display a clean image. The driver hsync and vsync is in the sub module in the VGA interface file called VGA core. This module monitors the status of the FIFO to determine when to increment hsync and vsync to match the 60 fps rating of the display. To accomplish this VGA core also tracks the x and y coordinate of the current pixel location and passes them to the VGA interface.

4.5.2 RGB Normalization

The camera is configured to transmit 16 bits of RGB 565 for each pixel, however the display requires 24 bit RAW RGB data to display an image. In the top level file before the data is set to the VGA port and then to the display it is normalized to be spread across 3 Bytes of data. In this process color correction occurs to account for the uneven amount of information between R,G,and B values.

5 Software Design

5.1 CNN

The CNN is implemented in python. We use the pytorch industry-standard library to implement, train and run our CNN and we use torchvision and numpy to preprocess our data. The image data is written into one text file per image. Accordingly, there exist two files/modules: `data_utils.py` and `cnn.py`. `data_utils.py` fulfills three purpses: It reads the training and test data, it preprocesses it, and it distributes them via the Dataset and dataloader classes. Regarding the first two parts, for each image in the training or test set, it reads the image data from a text file and produces a three dimensional array, where the third dimension is the color. Then, the array is normalized and cropped and downsampled into the target shape of 100x100x3. This is turned into a tensor with rotated dimensions of 3x100x100, which is required since the CNN extends the number of channels through multiple kernels. To facilitate the training of the CNN, we implement a torch DataSet class that at any time holds the filenames, labels, and indices of the training set. The labels are generated through manual filename and location logic. We implement a standard train-test split of roughly 80% train. Then, we instantiate dataloader classes on our Dataset instance. They mitigate the fact that the entire data of the dataset is too large to be held in memory, but required for training. FOr this reason, they discretize into manually batch-sized chunks, which get read sequentially during training. The dataset implements a required getter function that based on an data point id, carries out the above loading and preprocessing steps, and sends the data of this batch to the CNN.

For our purposes and the specific quality received from the camera module, we produced our own dataset. Each of the 52 different playing cards is recorded with approx. 25 images, which get randomized in order for training.

`cnn.py` fulfills the purpose of first, instantiating the CNN, second, training it using training data batches from the dataloader, and third, testing the performance. Regarding the first, we create the model by concatenating predefined pytorch layer instances, such as Conv2d, Linear, and Maxpool. We then define the forward pass of the network to be the sequential execution of each layer, i.e. $x_i = layer_i(x_{i-1})$. In total, we use 3 convolutional layers with kernels between 3x3 and 5x5, 2 max-pooling layers with kernels of 2x2, and 3 linear layers. We flatten the three-dimensional tensors after the last convolution into one-dimensional tensors required for linear layers. To introduce the non-linearity that renders neural networks powerful, each convolution and linear layer except the last is followed by the Relu activation. Regarding the training, iteratively request raining batches from the dataloader for a number of epochs, that is times the entire training set has been visited. At each batch, the loss is comnputed as cross-entropy loss between the produced 53-d vector and the one-hot encoded target 53-d vector in which the target class has entry 1, and every other entry is zero. We then take the gradient of each layer's parameters with respect to the loss, and update these parameters in an step-sized step into the opposite direction. We use batch size 64 and step size $5e - 5$.

Regarding the testing, we run a forward pass on the entire test set and compute the mean accuracy of predicted classes.

5.2 FPGA to HPS Driver

We talk to the FPGA image reader component via a kernelspace driver and a userspace program accessing it.

- The img_reader hardware component is made visible to the kernel through the .dtb file generated and the memory of the device mapped into accessible virtual memory using an _iomap() call.

- To meet our extended output of one R,G,B,0x00 value at a time, we implemented ioctl() using `ioread32()` and `iowrite32()` to write to and read from the memory on the device. Data read from the device is then copied to the userspace using copy_to_user().

- Userspace uses FILE I/O to store the image in both ASCII RGB format and in raw binary formats. The former can be parsed by python and the latter can be rendered on online raw image file viewers to check validity.

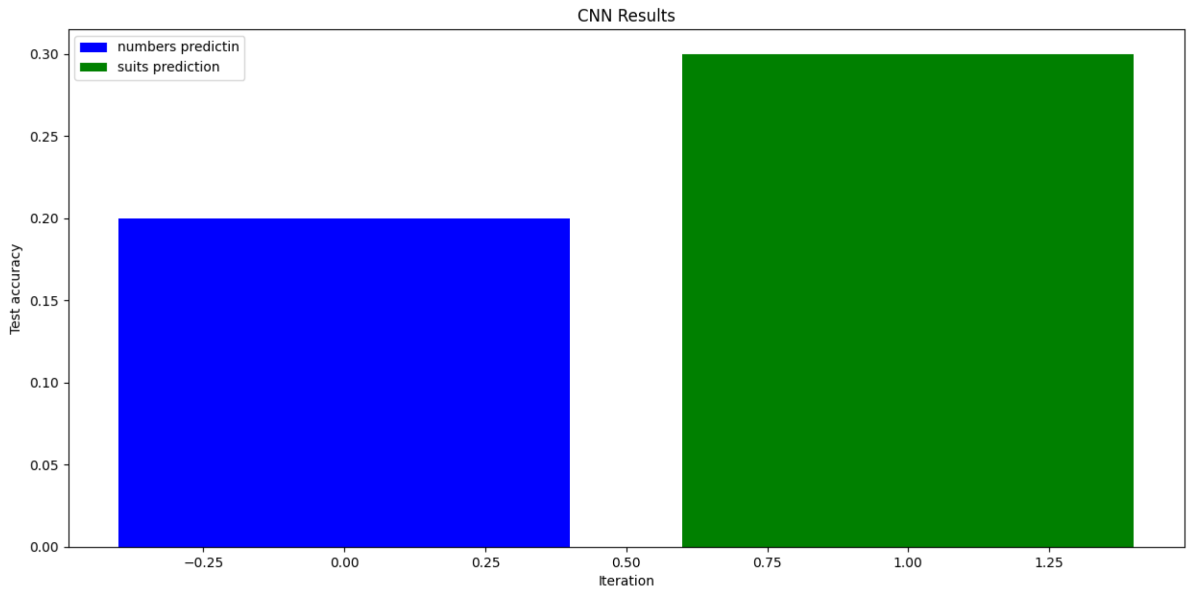
- Compared to lab3, we solved the challenge of reading and writing entire arrays from and to the FPGA, which presented a difficulty regarding their size of 640x480. Since the ioctl defintion in kernel has a fixed required signature, we cast the long argument to ioctl as a pointer to an array. We create this array in userspace via malloc and of the same size in kernel space via kmalloc. We then populate or read these arrays and copy the pointers between the two spaces. While eventually only used for reading, we validated that we can write large arrays to the FPGA as well.

5.3 Executable wrapper

The entire dataflow of an image from camera to FPGA to software driver to CNN is initiated and controlled by one overarching python wrapper file. We use the ctypes library to execute C libraries from the python file, to which end we compiled our C drivers as a library. The wrapper first executes the img reader component C driver, which triggers the image capture on the hardware side. Then the captured image is stored temporarily and cnn.py is executed to read the image and classify it. Then, wrapper prints the result on the HPS.

6 Results

From the systems point of view, we implemented our architecture successfully: The camera modules sends correct RGB data to the FPGA, and the VGA display verifies that the camera works. The throughput of the FPGA-HPS averages around 1 image frame transfer per second. In addition to our working system, our experimental results from our CNN training process shows a prediction accuracy of around 30% for predicting the suit of the cards and up to 20% for predicting the number of the cards, each as separate task with separate training process. While not at maximum, these performances are considerable because they perform better than their random baseline (25% and 7.6%, respectively). Furthermore, the input from the camera comes wit a custom noise, making the prediction harder and the implemented network is constrained in size and complexity to allow for pseudo real-time classification. In light of this, we achieved a viable proof of concept for further expansion.



7 Discussion

While achieving our objectives, we encountered many challenges and bottlenecks during our implementation of this project. The integration of so many systems across the De1-Soc FPGA was extremely challenging and required precise movement of data thorugh perfect verilog syntax.

7.1 Difficulties

- Interface of O7670 took longer than expected. To understand the correct pin out and GPIO configuration we had to test and measure each pin which added a significant amount of time.
- SCCB requires precise register configurations. We faced multiple challenges with trying to understand the SCCB protocol and register configurations required to stream data from the camera. To try and debug this system we interface a VGA Display at the same time. The lack of human interpret able feedback made this difficult.
- SDRAM Controller is VERY Complex. Ended up using an online implementation because it was too complex.
- Avalon bus configuration must be specific to receive the correct data. In order to receive larger sections of data in a more efficient way we reconfigured the read operation to work on 32 bits concurrently. This lead to a large bug that corrupted 2 SD cards and several days to fix.

7.2 Bottlenecks

- The OV7670 has a capped output fps of 30. VGA operates at 60 fps so the throughput is restricted to the limitation of the camera.
- The size of data transfer between FPGA and HPS through ioread/write is limited over the Avalon Bus.
- CNN weights require large RAM storage to implement on hardware.
- Transferring data across multiple interfaces has costs so to increase efficiency the image needs to be down scaled from 640 x 480 to something more manageable.
- Clock speeds of peripherals and internal FPGA hardware are fixed and slower than software on HPS.

7.3 Group Contributions

7.3.1 Lennart Schulze : ls3932

- Designed the CNN model and implemented the final model in python.
- Created the training set comprised of images from the OV7670 camera.
- Trained the CNN on several sets of data including the images from the OV7670.
- Designed the IMG reader hardware component.
- Designed and implemented the IMG Reader FPGA-to-HPS software device driver and userspace program.
- Connected the C and python software components into one executable wrapper.

7.3.2 Joseph Han : jh4632

- Implemented the integration of the SCCB protocol and interface of the OV7670 camera over the GPIO ports.
- Integrated the SDRAM interface to control the FIFO and SDRAM for a single, still image.
- Improved software device driver to achieve a reasonable data throughput
- Designed the FILE I/O operations to store and validate captured image data
- Helped integrate the VGA display with the data from the camera.
- Designed the file structure and hierarchy.
- Helped train the model on images from the OV7670 camera.

7.3.3 Michael Ozymy: mjo2156

- Helped integrate the Interface of the OV7670 camera with the GPIO ports.
- Designed and ran test benches for the SCCB, IMG Reader, and VGA Display protocols.
- Helped integrate the Interface of the VGA display with the data from the camera.
- Created the Platform Designer component for the FPGA to HPS connection.
- Created the presentation.
- Created all graphics.

7.4 Advice

- Making sure hardware and hardware-software connections work before any substantial coding in software or SystemVerilog. In section 1.4, steps 2 and 3 should be prioritized.
- Implement own controller for block RAM addressing and time multiplexing. Using the VGA controller to dictate timing of data transfer to the HPS causes black, padding pixels to be transmitted as part of the VGA protocol, which degrades image quality.
- Be aware of the challenges associated with asynchronous clocks and storing to the SDRAM.It is EXTREMELY complex.
- Use established code for non custom sections. We spent a large portion of time trying to re-invent the wheel when correct implementations are available if given a sufficient amount of search.

8 Appendix

8.1 Black Jack

The focus of our project is to give the player an edge in the popular casino game Blackjack. All casino games are known for dis-proportionally favoring the house to increase profits. However, Blackjack offers players the closest to even odds if played correctly. If players are able to play optimal strategies while keeping a count of the cards being played the odds shift to favor the player. Blackjack is played between a single player and dealer. The goal of the game is to hold a combination of cards that totals 21. All cards maintain their presented values with each face card also being of value 10. The four Aces in the deck can be counted as either a 1 or 11.

The act of counting means the player keeps a mental count of the type of cards shown during the game. By knowing what cards have passed the player can estimate the likelihood of the next card being high or low. Therefore if the count estimates that high cards are likely the player chooses to bet more as the odds of their score being higher are increased. Counting works as follows, a single integer count is kept and altered on the unveiling of any new card. Starting at 0 if any card below the value 7 is shown the count is decreased 1, if any card between the values 7 and 9 (inclusive) the count remains the same, and if any card with value 10 or above is show the count is increased by 1. Based on the state of the count the player can infer the state of the game. Using the CNN to classify snapshots of the playing field in a fixed interval such as every 10 seconds, we keep record of which cards have been played. We then count the score of the cards, using the "Hi-Lo" card counting algorithm laid out in [2].

8.2 Acknowledgements

Sources [1], [2], [3], [4], [5], [6], [7], [8]

[1] Albawi, S., Mohammed, T. A., \& Al-Zawi, S. (2017, August). Understanding of a convolutional neural network. In 2017 international conference on engineering and technology (ICET) (pp. 1-6). Ieee.

[2] How to count cards in blackjack and bring down the House. Blackjack Apprenticeship. (2022, June 1). https://www.blackjackapprenticeship.com/how-to-count-cards/

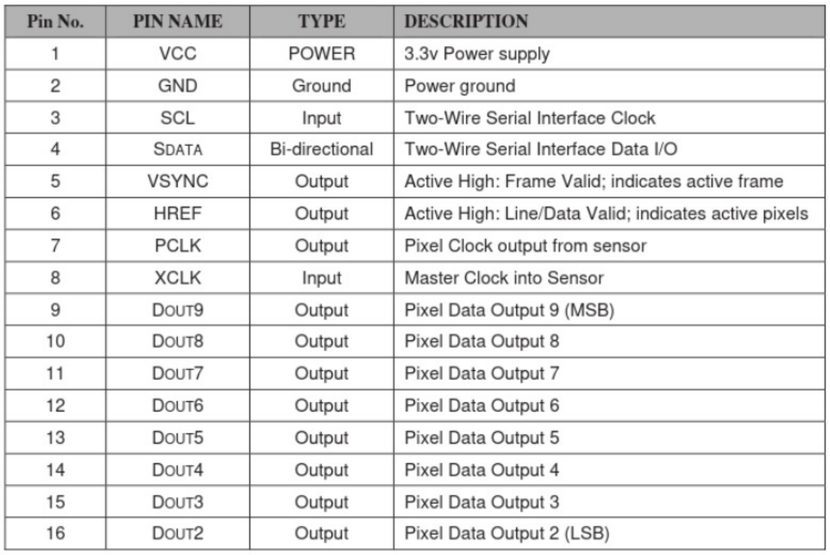
[3] OV7670 CMOS camera module REVB DS - OpenHacks. (n.d.-c). https://www.openhacks.com/uploadsproductos/ov7670_cmos_camera_module_revc_ds.pdf

[4] OV7670 CMOS camera module REVB DS - OpenHacks. (n.d.-c). https://www.openhacks.com/uploadsproductos/ov7670_cmos_camera_module_revc_ds.pdf

[5] OV7670 CMOS camera module REVB DS - OpenHacks. (n.d.-c). https://www.openhacks.com/uploadsproductos/ov7670_cmos_camera_module_revc_ds.pdf
[6] Contents. (n.d.-a). http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/DE1-SoC_User_manual.pdf
[7] OMNI ISION PRELIMINARY DATASHEET advanced information. (n.d.-b). http://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf
[8] angelo76. (n.d.). Security camera #4: Interfacing with ov7670 camera. element14 Community. https://community.element14.com/challenges-projects/design-challenges/summer-of-fpga/b/blog/posts/security-camera-3-interfacing-with-ov7670-camera

8.3 Reference Tables

Pinout of Camera Module:



Pin No.	PIN NAME	TYPE	DESCRIPTION
1	VCC	POWER	3.3v Power supply
2	GND	Ground	Power ground
3	SCL	Input	Two-Wire Serial Interface Clock
4	SDATA	Bi-directional	Two-Wire Serial Interface Data I/O
5	VSYNC	Output	Active High: Frame Valid; indicates active frame
6	HREF	Output	Active High: Line/Data Valid; indicates active pixels
7	PCLK	Output	Pixel Clock output from sensor
8	XCLK	Input	Master Clock into Sensor
9	DOUT9	Output	Pixel Data Output 9 (MSB)
10	DOUT8	Output	Pixel Data Output 8
11	DOUT7	Output	Pixel Data Output 7
12	DOUT6	Output	Pixel Data Output 6
13	DOUT5	Output	Pixel Data Output 5
14	DOUT4	Output	Pixel Data Output 4
15	DOUT3	Output	Pixel Data Output 3
16	DOUT2	Output	Pixel Data Output 2 (LSB)

DE1-SOC 2x20 GPIO Pinout:

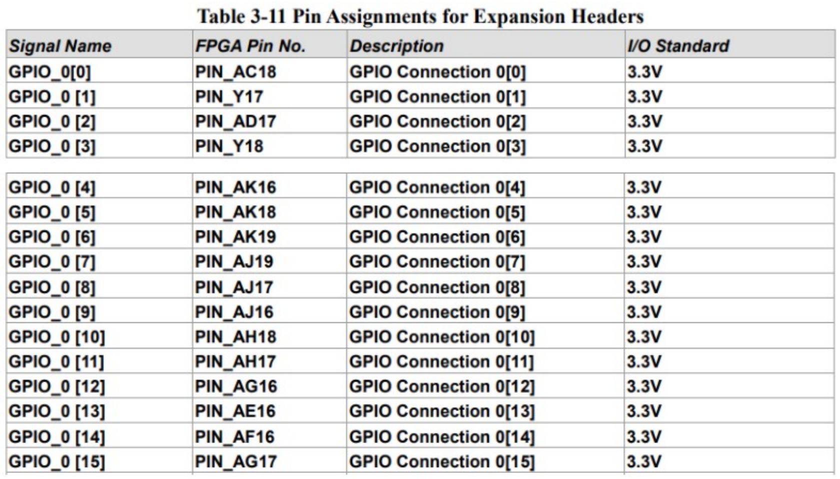


Table 3-11 Pin Assignments for Expansion Headers

Signal Name	FPGA Pin No.	Description	I/O Standard
GPIO_0[0]	PIN_AC18	GPIO Connection 0[0]	3.3V
GPIO_0 [1]	PIN_Y17	GPIO Connection 0[1]	3.3V
GPIO_0 [2]	PIN_AD17	GPIO Connection 0[2]	3.3V
GPIO_0 [3]	PIN_Y18	GPIO Connection 0[3]	3.3V
GPIO_0 [4]	PIN_AK16	GPIO Connection 0[4]	3.3V
GPIO_0 [5]	PIN_AK18	GPIO Connection 0[5]	3.3V
GPIO_0 [6]	PIN_AK19	GPIO Connection 0[6]	3.3V
GPIO_0 [7]	PIN_AJ19	GPIO Connection 0[7]	3.3V
GPIO_0 [8]	PIN_AJ17	GPIO Connection 0[8]	3.3V
GPIO_0 [9]	PIN_AJ16	GPIO Connection 0[9]	3.3V
GPIO_0 [10]	PIN_AH18	GPIO Connection 0[10]	3.3V
GPIO_0 [11]	PIN_AH17	GPIO Connection 0[11]	3.3V
GPIO_0 [12]	PIN_AG16	GPIO Connection 0[12]	3.3V
GPIO_0 [13]	PIN_AE16	GPIO Connection 0[13]	3.3V
GPIO_0 [14]	PIN_AF16	GPIO Connection 0[14]	3.3V
GPIO_0 [15]	PIN_AG17	GPIO Connection 0[15]	3.3V

Configuration of camera module

Output format configuration:

Using GPIO_0 [3] (PIN_Y18) on the FPGA, select address 12 and set its value to 04 to select RGB output format

8.4 Code listing

8.4.1 soc_system_top.sv

```
// ==================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ==================================================================
//
// Modified 2019 by Stephen A. Edwards
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ===============================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
module soc_system_top(

 ///////// ADC /////////
 inout         ADC_CS_N,
 output        ADC_DIN,
 input         ADC_DOUT,
 output        ADC_SCLK,

 ///////// AUD /////////
 input         AUD_ADCDAT,
 inout         AUD_ADCLRCK,
 inout         AUD_BCLK,
 output        AUD_DACDAT,
 inout         AUD_DACLRCK,
 output        AUD_XCK,

 ///////// CLOCK2 /////////
 input         CLOCK2_50,

 ///////// CLOCK3 /////////
 input         CLOCK3_50,

 ///////// CLOCK4 /////////
```

```
input         CLOCK4_50,

///////// CLOCK /////////
input         CLOCK_50,

///////// DRAM /////////
output [12:0] DRAM_ADDR,
output [1:0]  DRAM_BA,
output        DRAM_CAS_N,
output        DRAM_CKE,
output        DRAM_CLK,
output        DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output        DRAM_LDQM,
output        DRAM_RAS_N,
output        DRAM_UDQM,
output        DRAM_WE_N,

///////// FAN /////////
output        FAN_CTRL,

///////// FPGA /////////
output        FPGA_I2C_SCLK,
inout         FPGA_I2C_SDAT,

///////// GPIO /////////
inout [35:0]  GPIO_0,
inout [35:0]  GPIO_1,

///////// HEX0 /////////
output [6:0]  HEX0,

///////// HEX1 /////////
output [6:0]  HEX1,

///////// HEX2 /////////
output [6:0]  HEX2,

///////// HEX3 /////////
output [6:0]  HEX3,

///////// HEX4 /////////
output [6:0]  HEX4,

///////// HEX5 /////////
output [6:0]  HEX5,

///////// HPS /////////
inout         HPS_CONV_USB_N,
output [14:0] HPS_DDR3_ADDR,
output [2:0]  HPS_DDR3_BA,
output        HPS_DDR3_CAS_N,
output        HPS_DDR3_CKE,
output        HPS_DDR3_CK_N,
output        HPS_DDR3_CK_P,
output        HPS_DDR3_CS_N,
output [3:0]  HPS_DDR3_DM,
inout [31:0]  HPS_DDR3_DQ,
inout [3:0]   HPS_DDR3_DQS_N,
inout [3:0]   HPS_DDR3_DQS_P,
output        HPS_DDR3_ODT,
output        HPS_DDR3_RAS_N,
output        HPS_DDR3_RESET_N,
input         HPS_DDR3_RZQ,
output        HPS_DDR3_WE_N,
output        HPS_ENET_GTX_CLK,
inout         HPS_ENET_INT_N,
output        HPS_ENET_MDC,
inout         HPS_ENET_MDIO,
input         HPS_ENET_RX_CLK,
input [3:0]   HPS_ENET_RX_DATA,
input         HPS_ENET_RX_DV,
output [3:0]  HPS_ENET_TX_DATA,
output        HPS_ENET_TX_EN,
inout         HPS_GSENSOR_INT,
inout         HPS_I2C1_SCLK,
inout         HPS_I2C1_SDAT,
inout         HPS_I2C2_SCLK,
inout         HPS_I2C2_SDAT,
inout         HPS_I2C_CONTROL,
inout         HPS_KEY,
inout         HPS_LED,
inout         HPS_LTC_GPIO,
output        HPS_SD_CLK,
inout         HPS_SD_CMD,
inout [3:0]   HPS_SD_DATA,
output        HPS_SPIM_CLK,
input         HPS_SPIM_MISO,
output        HPS_SPIM_MOSI,
inout         HPS_SPIM_SS,
input         HPS_UART_RX,
output        HPS_UART_TX,
input         HPS_USB_CLKOUT,
inout [7:0]   HPS_USB_DATA,
input         HPS_USB_DIR,
input         HPS_USB_NXT,
output        HPS_USB_STP,

///////// IRDA /////////
input         IRDA_RXD,
output        IRDA_TXD,

///////// KEY /////////
input [3:0]   KEY,

///////// LEDR /////////
output [9:0]  LEDR,

///////// PS2 /////////
inout         PS2_CLK,
inout         PS2_CLK2,
inout         PS2_DAT,
inout         PS2_DAT2,

///////// SW /////////
input [9:0]   SW,

///////// TD /////////
```

```
 input         TD_CLK27,
 input [7:0]   TD_DATA,
 input         TD_HS,
 output        TD_RESET_N,
 input         TD_VS,


 ///////// VGA /////////
 output [7:0]  VGA_B,
 output        VGA_BLANK_N,
 output        VGA_CLK,
 output [7:0]  VGA_G,
 output        VGA_HS,
 output [7:0]  VGA_R,
 output        VGA_SYNC_N,
 output        VGA_VS
);


   soc_system soc_system0(
     .clk_clk                      ( CLOCK_50 ),
     .reset_reset_n                ( 1'b1 ),

     .hps_ddr3_mem_a               ( HPS_DDR3_ADDR ),
     .hps_ddr3_mem_ba              ( HPS_DDR3_BA ),
     .hps_ddr3_mem_ck              ( HPS_DDR3_CK_P ),
     .hps_ddr3_mem_ck_n            ( HPS_DDR3_CK_N ),
     .hps_ddr3_mem_cke             ( HPS_DDR3_CKE ),
     .hps_ddr3_mem_cs_n            ( HPS_DDR3_CS_N ),
     .hps_ddr3_mem_ras_n           ( HPS_DDR3_RAS_N ),
     .hps_ddr3_mem_cas_n           ( HPS_DDR3_CAS_N ),
     .hps_ddr3_mem_we_n            ( HPS_DDR3_WE_N ),
     .hps_ddr3_mem_reset_n         ( HPS_DDR3_RESET_N ),
     .hps_ddr3_mem_dq              ( HPS_DDR3_DQ ),
     .hps_ddr3_mem_dqs             ( HPS_DDR3_DQS_P ),
     .hps_ddr3_mem_dqs_n           ( HPS_DDR3_DQS_N ),
     .hps_ddr3_mem_odt             ( HPS_DDR3_ODT ),
     .hps_ddr3_mem_dm              ( HPS_DDR3_DM ),
     .hps_ddr3_oct_rzqin           ( HPS_DDR3_RZQ ),

     .hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
     .hps_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
     .hps_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
     .hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
     .hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
     .hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
     .hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
     .hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
     .hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
     .hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
     .hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
     .hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
     .hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
     .hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),

     .hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD           ),
     .hps_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]       ),
     .hps_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]       ),
     .hps_hps_io_sdio_inst_CLK     ( HPS_SD_CLK           ),
     .hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]      ),
     .hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]      ),

     .hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]     ),
     .hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]     ),
     .hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]     ),
     .hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]     ),
     .hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]     ),
     .hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]     ),
     .hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]     ),
     .hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]     ),
     .hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT      ),
     .hps_hps_io_usb1_inst_STP     ( HPS_USB_STP         ),
     .hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR         ),
     .hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT         ),

     .hps_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
     .hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
     .hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
     .hps_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),

     .hps_hps_io_uart0_inst_RX     ( HPS_UART_RX     ),
     .hps_hps_io_uart0_inst_TX     ( HPS_UART_TX     ),

     .hps_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT     ),
     .hps_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK     ),

     .hps_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT     ),
     .hps_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK     ),

     .hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
     .hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
     .hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

     .hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
     .hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
     .hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
     .hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),

     //.vga_r (VGA_R),
     //.vga_g (VGA_G),
     //.vga_b (VGA_B),
     //.vga_clk (VGA_CLK),
     //.vga_hs (VGA_HS),
     //.vga_vs (VGA_VS),
     //.vga_blank_n (VGA_BLANK_N),
     //.vga_sync_n (VGA_SYNC_N),

     // HPS input
     .img_reader_hsync(VGA_HS),
     .img_reader_vga_r(VGA_R),
     .img_reader_vga_g(VGA_G),
     .img_reader_vga_b(VGA_B),
     .img_reader_vsync(VGA_VS),
     .img_reader_get_img(get_img)
  );


    top_module top(
.pressed(pressed),
.clk ( CLOCK_50 ),
.rst_n(SW[0]),
.key( KEY ), //key[1:0] for brightness control , key[3:2] for contrast
 control
//camera pinouts
.cmos_pclk(GPIO_1[20]), // in
.cmos_href(GPIO_1[23]), // in
.cmos_vsync(GPIO_1[22]), // in
.cmos_db({GPIO_1[18], GPIO_1[19], GPIO_1[16], GPIO_1[17], GPIO_1[14],
 GPIO_1[15], GPIO_1[12], GPIO_1[13]}), // in
.cmos_sda(GPIO_1[25]), //inout
.cmos_scl(GPIO_1[24]), //inout
.cmos_rst_n(GPIO_1[10]), //out
.cmos_pwdn(GPIO_1[11]), //out
.cmos_xclk(GPIO_1[21]), //out
//Debugging
.led(), //out
//controller to sdram
.sdram_clk(DRAM_CLK),//out
.sdram_cke(DRAM_CKE), //out
.sdram_cs_n(DRAM_CS_N), //out
.sdram_ras_n(DRAM_RAS_N), //out.
.sdram_cas_n(DRAM_CAS_N), //out
.sdram_we_n(DRAM_WE_N), //out
.sdram_addr(DRAM_ADDR),//out
.sdram_ba(DRAM_BA), //out
.sdram_dqm({DRAM_UDQM, DRAM_LDQM}), //out // Check for endianness
.sdram_dq(DRAM_DQ),//out // Check for endianness
//VGA output
.clk_vga(VGA_CLK),
.vga_out_r(vga_r),//out
.vga_out_g(vga_g),//out
.vga_out_b(vga_b),//out
.vga_out_vs(VGA_VS),//out
.vga_out_hs(VGA_HS), //out
.vga_blank_n(VGA_BLANK_N) //out

    );

  logic [25:0]    count;
   always_ff @(posedge CLOCK_50)
     begin
count <= count+1;
     end
   assign slowclk = count[25];
   assign LEDR[1] = slowclk;
   always_ff @(posedge slowclk) begin
      LEDR[9:5] <= vga_r;
   end
   // DEBUG
   //assign VGA_R = 8'b00011100;
   //assign VGA_G = 8'b00010010;
   //assign VGA_B = 8'b11000000;
  ////////////////
   logic pressed;
   wire cmos_pclk;
   wire get_img;
   wire cmos_href;
   wire cmos_vsync;
   wire [7:0] cmos_db;
   wire cmos_sda;
   wire cmos_scl;
   wire cmos_rst_n;
   wire cmos_pwdn;
   wire cmos_xclk;
   wire [4:0] vga_r;
   wire [5:0] vga_g;
   wire [4:0] vga_b;
   //assign pressed = SW[1];
   assign pressed = get_img;
   assign VGA_R = vga_r*255/31 + 8'd0;
   assign VGA_G = vga_g*255/64 + 8'd0;
   assign VGA_B = vga_b*255/31 + 8'd0;
   assign cmos_pclk = GPIO_1[20];
   assign cmos_href = GPIO_1[23]; // horizontal sync signal from camera
   assign cmos_vsync = GPIO_1[22]; // vertical sync signal from camera
   assign cmos_db = {GPIO_1[18], GPIO_1[19], GPIO_1[16], GPIO_1[17],
   GPIO_1[14], GPIO_1[15], GPIO_1[12], GPIO_1[13]};
   // assign cmos_sda = GPIO_1[25];
   // assign cmos_scl = GPIO_1[24];
   assign cmos_rst_n = GPIO_1[10];
   assign cmos_pwdn = GPIO_1[11];
   assign cmos_xclk = GPIO_1[21];

   //assign VGA_SYNC_N = 1'b0;


   // The following quiet the "no driver" warnings for output
   // pins and should be removed if you use any of these peripherals

   assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
   assign ADC_DIN = SW[0];
   assign ADC_SCLK = SW[0];

   assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;
   assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
   assign AUD_DACDAT = SW[0];
   assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
   assign AUD_XCK = SW[0];

   /*
   assign DRAM_ADDR = { 13{ SW[0] } };
   assign DRAM_BA = { 2{ SW[0] } };
   assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : 16'bZ;
   assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
   DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };
   */
   assign FAN_CTRL = SW[0];

   assign FPGA_I2C_SCLK = SW[0];
   assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;

   assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : 36'bZ;
   //assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : 36'bZ;

   assign HEX0 = { 7{ SW[1] } };
   assign HEX1 = { 7{ SW[2] } };
```

```
   assign HEX2 = { 7{ SW[3] } };
   assign HEX3 = { 7{ SW[4] } };
   assign HEX4 = { 7{ SW[5] } };
   assign HEX5 = { 7{ SW[6] } };

   assign IRDA_TXD = SW[0];

   assign LEDR[0] = SW[0];

   assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
   assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

   assign TD_RESET_N = SW[0];

   /*
   assign {VGA_R, VGA_G, VGA_B} = { 24{ SW[0] } };
   assign {VGA_BLANK_N, VGA_CLK,
   VGA_HS, VGA_SYNC_N, VGA_VS} = { 5{ SW[0] } };*/

endmodule
```

8.4.2 top_module.v

```
‘timescale 1ns / 1ps

   module top_module(
input wire pressed,
input wire clk,rst_n,
input wire[3:0] key, //key[1:0] for brightness control , key[3:2]
 for contrast control
//camera pinouts
input wire cmos_pclk,cmos_href,cmos_vsync,
input wire[7:0] cmos_db,
inout cmos_sda,cmos_scl,
output wire cmos_rst_n, cmos_pwdn, cmos_xclk,
//Debugging
output[3:0] led,
//controller to sdram
output wire sdram_clk,
output wire sdram_cke,
output wire sdram_cs_n, sdram_ras_n, sdram_cas_n, sdram_we_n,
output wire[12:0] sdram_addr,
output wire[1:0] sdram_ba,
output wire[1:0] sdram_dqm,
inout[15:0] sdram_dq,
//VGA output
        output wire clk_vga,
output wire[4:0] vga_out_r,
output wire[5:0] vga_out_g,
output wire[4:0] vga_out_b,
output wire vga_out_vs,vga_out_hs, vga_blank_n
    );

 wire f2s_data_valid;
 wire[9:0] data_count_r;
 wire[15:0] dout,din;
 wire clk_sdram;
 wire empty_fifo;
 wire state;
 wire rd_en;


camera_interface m0 //control logic for retrieving data from camera,
 storing data to asyn_fifo, and  sending data to sdram
(
.clk(clk),
.clk_100(clk_sdram),
.rst_n(rst_n),
.key(key),
//asyn_fifo IO
.rd_en(f2s_data_valid),
.data_count_r(data_count_r),
.dout(dout),
//camera pinouts
.cmos_pclk(cmos_pclk),
.cmos_href(cmos_href),
.cmos_vsync(cmos_vsync),
.cmos_db(cmos_db),
.cmos_sda(cmos_sda),
.cmos_scl(cmos_scl),
.cmos_rst_n(cmos_rst_n),
.cmos_pwdn(cmos_pwdn),
.cmos_xclk(cmos_xclk),
//Debugging
.led(led)
    );

 sdram_interface m1 //control logic for writing the pixel-data from camera to
  sdram and reading pixel-data from sdram to vga
 (
.pressed(pressed),
.clk(clk_sdram),
.rst_n(rst_n),
//asyn_fifo IO
.clk_vga(clk_vga),
.rd_en(rd_en),
.data_count_r(data_count_r),
.f2s_data(dout),
.f2s_data_valid(f2s_data_valid),
.empty_fifo(empty_fifo),
.dout(din),
//controller to sdram
.sdram_cke(sdram_cke),
.sdram_cs_n(sdram_cs_n),
.sdram_ras_n(sdram_ras_n),
.sdram_cas_n(sdram_cas_n),
.sdram_we_n(sdram_we_n),
.sdram_addr(sdram_addr),
.sdram_ba(sdram_ba),
.sdram_dqm(sdram_dqm),
.sdram_dq(sdram_dq)
    );

 vga_interface m2 //control logic for retrieving data from sdram, storing data
  to asyn_fifo, and sending data to vga
 (
.clk(clk),
.rst_n(rst_n),
//asyn_fifo IO
.empty_fifo(empty_fifo),
.din(din),
.clk_vga(clk_vga),
.rd_en(rd_en),
//VGA output
.vga_out_r(vga_out_r),
.vga_out_g(vga_out_g),
.vga_out_b(vga_out_b),
.vga_out_vs(vga_out_vs),
.vga_out_hs(vga_out_hs),
.vga_blank_n(vga_blank_n)
    );



phase_diff_180 p1(
.datain_h(1'b0),
.datain_l(1'b1),
.outclock(clk_sdram),
.dataout(sdram_clk));

clk_100 clk_100_MHz(
.refclk(clk),   //  refclk.clk
.rst(RESET),      //   reset.reset
.outclk_0(clk_sdram), // outclk0.clk
.locked(LOCKED)    //  locked.export
);
/*
//ERROR APPEARS IF ODDR2 IS ROUTED INSIDE THE FPGA INSTEAD OF BEING DIRECTLY
 CONNECTED TO OUTPUT (so we bring this outside)
 ODDR2#(.DDR_ALIGNMENT("NONE"), .INIT(1'b0),.SRTYPE("SYNC")) oddr2_primitive
 (
.D0(1'b0),
.D1(1'b1),
.C0(clk_sdram),
.C1(~clk_sdram),
.CE(1'b1),
.R(1'b0),
.S(1'b0),
.Q(sdram_clk)
);



//SDRAM clock
dcm_165MHz m3
   (// Clock in ports
    .clk(clk),      // IN
    // Clock out ports
    .clk_sdram(clk_sdram),     // OUT
    // Status and control signals
    .RESET(RESET),// IN
    .LOCKED(LOCKED));      // OUT
*/
```

```
endmodule
```

8.4.3 camera_interface.v

```
‘timescale 1ns / 1ps

  module camera_interface(
input wire clk,clk_100,rst_n,
input wire[3:0] key, //key[1:0] for brightness control , key[3:2] for contrast
 control
//asyn_fifo IO
input wire rd_en,
output wire[9:0] data_count_r,
output wire[15:0] dout,
//camera pinouts
input wire cmos_pclk,cmos_href,cmos_vsync,
input wire[7:0] cmos_db,
inout cmos_sda,cmos_scl, //i2c comm wires
output wire cmos_rst_n, cmos_pwdn, cmos_xclk,
//Debugging
output wire[3:0] led
    );
 //FSM state declarations
 localparam idle=0,
start_sccb=1,
write_address=2,
write_data=3,
digest_loop=4,
delay=5,
vsync_fedge=6,
byte1=7,
byte2=8,
fifo_write=9,
stopping=10;

 localparam wait_init=0,
sccb_idle=1,
sccb_address=2,
sccb_data=3,
sccb_stop=4;

 localparam MSG_INDEX=77; //number of the last index to be digested by SCCB



 reg[3:0] state_q=0,state_d;
 reg[2:0] sccb_state_q=0,sccb_state_d;
 reg[7:0] addr_q,addr_d;
 reg[7:0] data_q,data_d;
 reg[7:0] brightness_q,brightness_d;
 reg[7:0] contrast_q,contrast_d;
 reg start,stop;
 reg[7:0] wr_data;
 wire rd_tick;
 wire[1:0] ack;
 wire[7:0] rd_data;
 wire[3:0] state;
 reg[3:0] led_q=0,led_d;
 reg[27:0] delay_q=0,delay_d;
 reg start_delay_q=0,start_delay_d;
 reg delay_finish;
 reg[15:0] message[250:0];
 reg[7:0] message_index_q=0,message_index_d;
 reg[15:0] pixel_q,pixel_d;
 reg wr_en;
 wire full;
 wire key0_tick,key1_tick,key2_tick,key3_tick;

 //buffer for all inputs coming from the camera
 reg pclk_1,pclk_2,href_1,href_2,vsync_1,vsync_2;


 initial begin //collection of all adddresses and values to be written in the camera
//{address,data}
 message[0]=16'h12_80;  //reset all register to default values
 message[1]=16'h12_04;  //set output format to RGB
 message[2]=16'h15_20;  //pclk will not toggle during horizontal blank
 message[3]=16'h40_d0; //RGB565

// These are values scalped from https://github.com/jonlwowski012/OV7670_NEXYS4
 _Verilog/blob/master/ov7670_registers_verilog.v
    message[4]= 16'h12_04; // COM7,     set RGB color output
    message[5]= 16'h11_80; // CLKRC     internal PLL matches input clock
    message[6]= 16'h0C_00; // COM3,     default settings
    message[7]= 16'h3E_00; // COM14,    no scaling, normal pclock
    message[8]= 16'h04_00; // COM1,     disable CCIR656
    message[9]= 16'h40_d0; //COM15,     RGB565, full output range
    message[10]= 16'h3a_04; //TSLB       set correct output data sequence (magic)
 message[11]= 16'h14_18; //COM9       MAX AGC value x4 0001_1000
    message[12]= 16'h4F_B3; //MTX1       all of these are magical matrix coefficients
    message[13]= 16'h50_B3; //MTX2
    message[14]= 16'h51_00; //MTX3
    message[15]= 16'h52_3d; //MTX4
    message[16]= 16'h53_A7; //MTX5
    message[17]= 16'h54_E4; //MTX6
    message[18]= 16'h58_9E; //MTXS
    message[19]= 16'h3D_C0; //COM13      sets gamma enable, does not preserve
    reserved bits, may be wrong?
    message[20]= 16'h17_14; //HSTART     start high 8 bits
    message[21]= 16'h18_02; //HSTOP      stop high 8 bits //these kill the odd
    colored line
    message[22]= 16'h32_80; //HREF       edge offset
    message[23]= 16'h19_03; //VSTART     start high 8 bits
    message[24]= 16'h1A_7B; //VSTOP      stop high 8 bits
    message[25]= 16'h03_0A; //VREF       vsync edge offset
    message[26]= 16'h0F_41; //COM6       reset timings
    message[27]= 16'h1E_00; //MVFP       disable mirror / flip //might have magic
    value of 03
    message[28]= 16'h33_0B; //CHLF       //magic value from the internet
    message[29]= 16'h3C_78; //COM12      no HREF when VSYNC low
    message[30]= 16'h69_00; //GFIX       fix gain control
    message[31]= 16'h74_00; //REG74      Digital gain control
    message[32]= 16'hB0_84; //RSVD       magic value from the internet *required*
    for good color
    message[33]= 16'hB1_0c; //ABLC1
    message[34]= 16'hB2_0e; //RSVD       more magic internet values
    message[35]= 16'hB3_80; //THL_ST
    //begin mystery scaling numbers
    message[36]= 16'h70_3a;
    message[37]= 16'h71_35;
    message[38]= 16'h72_11;
    message[39]= 16'h73_f0;
    message[40]= 16'ha2_02;
    //gamma curve values
    message[41]= 16'h7a_20;
    message[42]= 16'h7b_10;
    message[43]= 16'h7c_1e;
    message[44]= 16'h7d_35;
    message[45]= 16'h7e_5a;
    message[46]= 16'h7f_69;
    message[47]= 16'h80_76;
    message[48]= 16'h81_80;
    message[49]= 16'h82_88;
    message[50]= 16'h83_8f;
    message[51]= 16'h84_96;
    message[52]= 16'h85_a3;
    message[53]= 16'h86_af;
    message[54]= 16'h87_c4;
    message[55]= 16'h88_d7;
    message[56]= 16'h89_e8;
    //AGC and AEC
    message[57]= 16'h13_e0; //COM8, disable AGC / AEC
    message[58]= 16'h00_00; //set gain reg to 0 for AGC
    message[59]= 16'h10_00; //set ARCJ reg to 0
    message[60]= 16'h0d_40; //magic reserved bit for COM4
    message[61]= 16'h14_18; //COM9, 4x gain + magic bit
    message[62]= 16'ha5_05; // BD50MAX
    message[63]= 16'hab_07; //DB60MAX
    message[64]= 16'h24_95; //AGC upper limit
    message[65]= 16'h25_33; //AGC lower limit
    message[66]= 16'h26_e3; //AGC/AEC fast mode op region
    message[67]= 16'h9f_78; //HAECC1
    message[68]= 16'ha0_68; //HAECC2
    message[69]= 16'ha1_03; //magic
    message[70]= 16'ha6_d8; //HAECC3
    message[71]= 16'ha7_d8; //HAECC4
    message[72]= 16'ha8_f0; //HAECC5
    message[73]= 16'ha9_90; //HAECC6
    message[74]= 16'haa_94; //HAECC7
    message[75]= 16'h13_e5; //COM8, enable AGC / AEC
 message[76]= 16'h1E_23; //Mirror Image
 message[77]= 16'h69_06; //gain of RGB(manually adjusted)
  end

 //register operations
 always @(posedge clk_100,negedge rst_n) begin
if(!rst_n) begin
state_q<=0;
led_q<=0;
delay_q<=0;
start_delay_q<=0;
message_index_q<=0;
pixel_q<=0;

sccb_state_q<=0;
addr_q<=0;
```

```
data_q<=0;
brightness_q<=0;
contrast_q<=0;
end
else begin
state_q<=state_d;
led_q<=led_d;
delay_q<=delay_d;
start_delay_q<=start_delay_d;
message_index_q<=message_index_d;
pclk_1<=cmos_pclk;
pclk_2<=pclk_1;
href_1<=cmos_href;
href_2<=href_1;
vsync_1<=cmos_vsync;
vsync_2<=vsync_1;
pixel_q<=pixel_d;

sccb_state_q<=sccb_state_d;
addr_q<=addr_d;
data_q<=data_d;
brightness_q<=brightness_d;
contrast_q<=contrast_d;
end
 end


 //FSM next-state logics
 always @* begin
state_d=state_q;
led_d=led_q;
start=0;
stop=0;
wr_data=0;
start_delay_d=start_delay_q;
delay_d=delay_q;
delay_finish=0;
message_index_d=message_index_q;
pixel_d=pixel_q;
wr_en=0;

sccb_state_d=sccb_state_q;
addr_d=addr_q;
data_d=data_q;
brightness_d=brightness_q;
contrast_d=contrast_q;

//delay logic
if(start_delay_q) delay_d=delay_q+1'b1;
if(delay_q[16] && message_index_q!=(MSG_INDEX+1) && (state_q!=start_sccb))
  begin  //delay between SCCB transmissions (0.66ms)
delay_finish=1;
start_delay_d=0;
delay_d=0;
end
else if((delay_q[26] && message_index_q==(MSG_INDEX+1)) || (delay_q[26] &&
  state_q==start_sccb)) begin //delay BEFORE SCCB transmission, AFTER SCCB transmission,
  and BEFORE retrieving pixel data from camera (0.67s)
delay_finish=1;
```

```
start_delay_d=0;
delay_d=0;
end

case(state_q)

////////Begin: Setting register values of the camera via SCCB///////////

  idle:  if(delay_finish) begin //idle for 0.6s to start-up the camera
state_d=start_sccb;
start_delay_d=0;
end
else start_delay_d=1;

start_sccb:  begin   //start of SCCB transmission
start=1;
wr_data=8'h42; //slave address of OV7670 for write
state_d=write_address;
end
 write_address: if(ack==2'b11) begin
wr_data=message[message_index_q][15:8]; //write address
state_d=write_data;
end
 write_data: if(ack==2'b11) begin
wr_data=message[message_index_q][7:0]; //write data
state_d=digest_loop;
end
  digest_loop: if(ack==2'b11) begin //stop sccb transmission
stop=1;
start_delay_d=1;
message_index_d=message_index_q+1'b1;
state_d=delay;
end
  delay: begin
if(message_index_q==(MSG_INDEX+1) && delay_finish) begin
state_d=vsync_fedge; //if all messages are already
        digested, proceed to retrieving camera pixel data
led_d=4'b0110;
end
else if(state==0 && delay_finish) state_d=start_sccb;
       //small delay before next SCCB transmission(if all messages are not yet digested)
end



///////////////Begin: Retrieving Pixel Data from Camera to be Stored to
    SDRAM////////////////

vsync_fedge: if(vsync_1==0 && vsync_2==1) state_d=byte1;
  //vsync falling edge means new frame is incoming
byte1: if(pclk_1==1 && pclk_2==0 && href_1==1 && href_2==1) begin
    //rising edge of pclk means new pixel data(first byte of 16-bit pixel RGB565) is
    available at output
pixel_d[15:8]=cmos_db;
state_d=byte2;
 end
 else if(vsync_1==1 && vsync_2==1) begin
state_d=vsync_fedge;
 end
```

```
byte2: if(pclk_1==1 && pclk_2==0 && href_1==1 && href_2==1)
    begin //rising edge of pclk means new pixel data(second byte of 16-bit pixel RGB565)
    is available at output
pixel_d[7:0]=cmos_db;
state_d=fifo_write;
 end
 else if(vsync_1==1 && vsync_2==1) begin
state_d=vsync_fedge;
 end
 fifo_write: begin //write the 16-bit data to asynchronous fifo to be
   retrieved later by SDRAM
wr_en=1;
state_d=byte1;
if(full) led_d=4'b1001; //debugging led
 end
default: state_d=idle;
endcase

//Logic for increasing/decreasing brightness and contrast via the 4
  keybuttons
case(sccb_state_q)
wait_init: if(state_q==byte1) begin //wait for initial SCCB
   transmission to finish
sccb_state_d=sccb_idle;
addr_d=0;
data_d=0;
brightness_d=8'h00;
contrast_d=8'h40;
  end
sccb_idle: if(state==0) begin //wait for any pushbutton
if(key0_tick) begin//increase brightness
brightness_d=(brightness_q[7]==1)?
         brightness_q-1:brightness_q+1;
if(brightness_q==8'h80) brightness_d=0;
start=1;
wr_data=8'h42; //slave address of OV7670 for write
addr_d=8'h55; //brightness control address
data_d=brightness_d;
sccb_state_d=sccb_address;
led_d=0;
end
if(key1_tick) begin //decrease brightness
brightness_d=(brightness_q[7]==1)?
         brightness_q+1:brightness_q-1;
if(brightness_q==0) brightness_d=8'h80;
start=1;
wr_data=8'h42;
addr_d=8'h55;
data_d=brightness_d;
sccb_state_d=sccb_address;
led_d=0;
end
else if(key2_tick) begin //increase contrast
contrast_d=contrast_q+1;
start=1;
wr_data=8'h42; //slave address of OV7670 for write
addr_d=8'h56; //contrast control address
data_d=contrast_d;
sccb_state_d=sccb_address;
```

```
led_d=0;
end
else if(key3_tick) begin //change contrast
contrast_d=contrast_q-1;
start=1;
wr_data=8'h42;
addr_d=8'h56;
data_d=contrast_d;
sccb_state_d=sccb_address;
led_d=0;
end
  end
sccb_address: if(ack==2'b11) begin
wr_data=addr_q; //write address
sccb_state_d=sccb_data;
end
  sccb_data: if(ack==2'b11) begin
wr_data=data_q; //write databyte
sccb_state_d=sccb_stop;
 end
  sccb_stop: if(ack==2'b11) begin //stop
stop=1;
sccb_state_d=sccb_idle;
led_d=4'b1001;
 end
 default: sccb_state_d=wait_init;
endcase

 end


 assign cmos_pwdn=0;
 assign cmos_rst_n=1;
 assign led=led_q;

 //module instantiations
 i2c_top #(.freq(100_000)) m0
(
.clk(clk_100),
.rst_n(rst_n),
.start(start),
.stop(stop),
.wr_data(wr_data),
.rd_tick(rd_tick),
  //ticks when read data from servant is ready,data will be taken from rd_data
.ack(ack),
  //ack[1] ticks at the ack bit[9th bit],ack[0] asserts when ack bit is ACK,else NACK
.rd_data(rd_data),
.scl(cmos_scl),
.sda(cmos_sda),
.state(state)
    );

clk_24 clk_24_MHz(
.refclk(clk),   //  refclk.clk
.rst(RESET),      //   reset.reset
.outclk_0(cmos_xclk), // outclk0.clk
.locked(LOCKED)    //  locked.export
);
```

```
 /*
 dcm_24MHz m1
   (// Clock in ports
    .clk(clk),      // IN
    // Clock out ports
    .cmos_xclk(cmos_xclk),     // OUT
    // Status and control signals
    .RESET(RESET),// IN
    .LOCKED(LOCKED));      // OUT
 */

/*asyn_fifo m2 //buffer between camera and SDRAM
(
  .rst({!rst_n}), // input rst
  .wr_clk(clk_100), // input wr_clk [CAMERA]
  .rd_clk(clk_100), // input rd_clk [SDRAM]
  .din(pixel_q), // input [15 : 0] din
  .wr_en(wr_en), // input wr_en
  .rd_en(rd_en), // input rd_en
  .dout(dout), // output [15 : 0] dout
  .full(full), // output full
  .empty(), // output empty
  .rd_data_count(rd_data_count), // output [9 : 0] rd_data_count
  .wr_data_count() // output [9 : 0] wr_data_count
);*/
asyn_fifo #(.DATA_WIDTH(16),.FIFO_DEPTH_WIDTH(10)) m2
 //1024x16 FIFO mem
(
.rst_n(rst_n),
.clk_write(clk_100),
.clk_read(clk_100), //clock input from both domains
.write(wr_en),
.read(rd_en),
.data_write(pixel_q), //input FROM write clock domain
.data_read(dout), //output TO read clock domain
.full(full),
.empty(),
  //full=sync to write domain clk , empty=sync to read domain clk
.data_count_r(data_count_r)
  //asserted if fifo is equal or more than than half of its max capacity
    );

debounce_explicit m3
(
.clk(clk_100),
.rst_n(rst_n),
.sw({!key[0]}),
.db_level(),
.db_tick(key0_tick)
    );

debounce_explicit m4
(
.clk(clk_100),
.rst_n(rst_n),
.sw({!key[1]}),
.db_level(),
.db_tick(key1_tick)
```

```
    );

 debounce_explicit m5
(
.clk(clk_100),
.rst_n(rst_n),
.sw({!key[2]}),
.db_level(),
.db_tick(key2_tick)
    );

 debounce_explicit m6
(
.clk(clk_100),
.rst_n(rst_n),
.sw({!key[3]}),
.db_level(),
.db_tick(key3_tick)
    );

endmodule
```

8.4.4 vga_interface.v

```
‘timescale 1ns / 1ps

module vga_interface(
input wire clk,rst_n,
//asyn_fifo IO
input wire empty_fifo,
input wire[15:0] din,
output wire clk_vga,
output reg rd_en,
//VGA output
output reg[4:0] vga_out_r,
output reg[5:0] vga_out_g,
output reg[4:0] vga_out_b,
output wire vga_out_vs,vga_out_hs, vga_blank_n
    );
 //FSM state declarations
 localparam delay=0,
idle=1,
display=2;

 reg[1:0] state_q,state_d;
 wire[11:0] pixel_x,pixel_y;
 //register operations
 always @(posedge clk_out,negedge rst_n) begin
if(!rst_n) begin
state_q<=delay;
end
else begin
state_q<=state_d;
end
 end

 //FSM next-state logic
 always @* begin
 state_d=state_q;
 rd_en=0;
 vga_out_r=0;
 vga_out_g=0;
 vga_out_b=0;
case(state_q)
  delay: if(pixel_x==1 && pixel_y==1) state_d=idle;
    //delay of one frame(33ms) needed to start up the camera
idle:  if(pixel_x==1 && pixel_y==0 && !empty_fifo) begin
   //wait for pixel-data coming from asyn_fifo
vga_out_r=din[15:11];
vga_out_g=din[10:5];
vga_out_b=din[4:0];
rd_en=1;
state_d=display;
end
display: if(pixel_x>=1 && pixel_x<=640 && pixel_y<480) begin
  //we will continue to read the asyn_fifo as long as current pixel
  coordinate is inside the visible screen(640x480)
vga_out_r=din[15:11];
vga_out_g=din[10:5];
vga_out_b=din[4:0];
rd_en=1;
end
idle: state_d=delay;
endcase
 end



assign clk_vga=clk_out;

//module instantiations
vga_core m0
(
.clk(clk_out), //clock must be 25MHz for 640x480
.rst_n(rst_n),
.hsync(vga_out_hs),
.vsync(vga_out_vs),
.video_on(vga_blank_n),
.pixel_x(pixel_x),
.pixel_y(pixel_y)
);

clk_25 clk_25_MHz(
.refclk(clk),   //  refclk.clk
.rst(RESET),      //   reset.reset
.outclk_0(clk_out), // outclk0.clk
.locked(LOCKED)    //  locked.export
);

/*
 dcm_25MHz m1 //clock for vga(620x480 60fps)
   (// Clock in ports
    .clk(clk),      // IN
    // Clock out ports
    .clk_out(clk_out),     // OUT
    // Status and control signals
    .RESET(RESET),// IN
    .LOCKED(LOCKED));      // OUT
*/


endmodule
```

8.4.5 vga_core.v

```
‘timescale 1ns / 1ps

module vga_core(
input wire clk,rst_n, //clock must be 25MHz for 640x480 30fps
output wire hsync,vsync,
output reg video_on,
output wire[11:0] pixel_x,pixel_y
    ); //650x480 parameters
 localparam HD=640, //Horizontal Display
HR=16, //Right Border
HRet=96, //Horizontal Retrace
HL=48, //Left Border

VD=480, //Vertical Display
VB=10, //Bottom Border
VRet=2, //Vertical Retrace
VT=33; //Top Border
reg[11:0] vctr_q=0,vctr_d; //counter for vertical scan
reg[11:0] hctr_q=0,hctr_d; //counter for vertical scan
reg hsync_q=0,hsync_d;
reg vsync_q=0,vsync_d;
//vctr and hctr register operation
always @(posedge clk,negedge rst_n) begin
if(!rst_n) begin
vctr_q<=0;
hctr_q<=0;
vsync_q<=0;
hsync_q<=0;
end
else begin
vctr_q<=vctr_d;
hctr_q<=hctr_d;
vsync_q<=vsync_d;
hsync_q<=hsync_d;
end
end

always @* begin
vctr_d=vctr_q;
hctr_d=hctr_q;
video_on=0;
hsync_d=1;
vsync_d=1;

if(hctr_q==HD+HR+HRet+HL-1) hctr_d=0; //horizontal counter
else hctr_d=hctr_q+1’b1;

if(vctr_q==VD+VB+VRet+VT-1) vctr_d=0; //vertical counter
else if(hctr_q==HD+HR+HRet+HL-1) vctr_d=vctr_q+1’b1;
```

```
if(hctr_q<HD && vctr_q<VD) video_on=1; //video_on

if( (hctr_d>=HD+HR) && (hctr_d<=HD+HR+HRet-1) ) hsync_d=0; //horizontal sync
if( (vctr_d>=VD+VB) && (vctr_d<=VD+VB+VRet-1) ) vsync_d=0; //vertical sync

end
assign vsync=vsync_q;
assign hsync=hsync_q;
assign pixel_x=hctr_q;
assign pixel_y=vctr_q;

endmodule
```

8.4.6 sdram_interface.v

```
‘timescale 1ns / 1ps

module sdram_interface(
input wire pressed,
input clk,rst_n,
//asyn_fifo IO
input wire clk_vga, rd_en,
input wire[9:0] data_count_r,
input wire[15:0] f2s_data,
output wire f2s_data_valid,
output wire empty_fifo,
output wire[15:0] dout,
//controller to sdram
output wire sdram_clk,
output wire sdram_cke,
output wire sdram_cs_n, sdram_ras_n, sdram_cas_n, sdram_we_n,
output wire[12:0] sdram_addr,
output wire[1:0] sdram_ba,
output wire[1:0] sdram_dqm,
inout[15:0] sdram_dq
    );
 //FSM state declarations
 localparam idle=0,
burst_op=1;

 reg state_q=0,state_d;
 reg[14:0] wr_addr_q=0,wr_addr_d;
 reg[14:0] rd_addr_q=0,rd_addr_d;
 reg rw,rw_en;
 reg[14:0] f_addr;
 wire[15:0] s2f_data;
 wire s2f_data_valid;
 wire ready;
 wire[9:0] data_count_w;


 //register operation
 always @(posedge clk,negedge rst_n) begin
if(!rst_n) begin
state_q<=0;
wr_addr_q<=0;
rd_addr_q<=0;
end
```

```
else begin
state_q<=state_d;
wr_addr_q<=wr_addr_d;
rd_addr_q<=rd_addr_d;
end
 end

 //FSM next-state declarations
 always @* begin
state_d=state_q;
wr_addr_d=wr_addr_q;
rd_addr_d=rd_addr_q;
f_addr=0;
rw=0;
rw_en=0;

case(state_q)
    idle: if(data_count_r>512 && ready) begin
       //wait for the first 512 pixel-data to fill the asyn_fifo then burst-write it to sdram
rw_en=1;
rw=0;
wr_addr_d=1;
f_addr=wr_addr_q;
state_d=burst_op;
 end
burst_op: if(ready) begin //choose whether to read the asyn_fifo
   of camera OR write to asyn_fifo of VGA
if(data_count_r>512 && (!pressed)) begin
        //asyn_fifo of camera is filled to 512 thus we can now burst-write
        (full-page has 512 data) it to SDRAM
rw_en=1;
rw=0;
wr_addr_d=(wr_addr_q==599)? 0:wr_addr_q+1'b1;
         //One frame(640x480) fills the addresses 0-to-599
f_addr=wr_addr_q;
end else if (data_count_r>512 && pressed &&
        (wr_addr_d!=0)) begin
rw_en=1;
rw=0;
wr_addr_d=(wr_addr_q==599)? 0:wr_addr_q+1'b1;
         //One frame(640x480) fills the addresses 0-to-599
f_addr=wr_addr_q;
end else if (data_count_r>512 && pressed &&
        (wr_addr_d==0)) begin
rw_en=1;
rw=0;
wr_addr_d=0; //One frame(640x480) fills the addresses 0-to-599
f_addr=wr_addr_q;
end
else if(data_count_w<250) begin
        //asyn_fifo of VGA has only 250 pixel data left, we will now
        fill it by another 512 pixel data via burst reading the sdram
rw_en=1;
rw=1;
rd_addr_d=(rd_addr_q==599)? 0:rd_addr_q+1'b1;
f_addr=rd_addr_q;
end
 end
 default: state_d=idle;
```

```
endcase
 end

 //module instantiations
 sdram_controller m0(
//fpga to controller
.clk(clk), //clk=165MHz
.rst_n(rst_n),
.rw(rw), // 1:read , 0:write
.rw_en(rw_en), //must be asserted before read/write
.f_addr(f_addr), //14:2=row(13)  ,
  1:0=bank(2) , no need for column address since full page mode will
  always start from zero and end with 511 words
.f2s_data(f2s_data), //fpga-to-sdram data
.s2f_data(s2f_data), //sdram to fpga data
.s2f_data_valid(s2f_data_valid),
  //asserts while  burst-reading(data is available at output UNTIL the next rising edge)
.f2s_data_valid(f2s_data_valid),
  //asserts while burst-writing(data must be available at input BEFORE the next rising edge)
.ready(ready), //"1" if sdram is available for nxt read/write operation
//controller to sdram
.s_clk(sdram_clk),
.s_cke(sdram_cke),
.s_cs_n(sdram_cs_n),
.s_ras_n(sdram_ras_n ),
.s_cas_n(sdram_cas_n),
.s_we_n(sdram_we_n),
.s_addr(sdram_addr),
.s_ba(sdram_ba),
.LDQM(sdram_dqm[0]),
.HDQM(sdram_dqm[1]),
.s_dq(sdram_dq)
);

asyn_fifo #(.DATA_WIDTH(16),.FIFO_DEPTH_WIDTH(10)) m2 //1024x16 FIFO mem
(
.rst_n(rst_n),
.clk_write(clk),
.clk_read(clk_vga),
.write(s2f_data_valid),
.read(rd_en),
.data_write(s2f_data), //input FROM write clock domain
.data_read(dout), //output TO read clock domain
.full(),
.empty(empty_fifo),
  //full=sync to write domain clk , empty=sync to read domain clk
.data_count_w(data_count_w)
    );


endmodule
```

8.4.7 sdram_controller.v

```
‘timescale 1ns / 1ps

module sdram_controller(
//fpga to controller
input wire clk,rst_n,  //clk=165MHz
input wire rw, // 1:read , 0:write
input wire rw_en, //must be asserted before read/write
input wire[14:0] f_addr,
 //14:2=row(13)  , 1:0=bank(2) , no need for column address since full
 page mode will always start from zero and end with 512 words
input wire [15:0] f2s_data,
 //fpga-to-sdram data , write data burst will start at assertion of f2s_data_valid
output wire[15:0] s2f_data,
 //sdram to fpga data
output wire s2f_data_valid,
 //asserts while  burst-reading(data is available at output UNTIL the next rising edge)
output reg f2s_data_valid,
 //asserts while burst-writing(data must be available at input BEFORE the next rising edge)
output reg ready, //"1" if sdram is available for nxt read/write operation

//controller to sdram
output wire s_clk,
output wire s_cke,
 //always high for almost all operations(except for self-refresh w/c I did not use here)
output wire s_cs_n, s_ras_n, s_cas_n, s_we_n, //commands
output wire[12:0] s_addr, //row/colum address bus
output wire[1:0] s_ba, //bank address bus
output wire LDQM , HDQM, //low-byte and high-byte mask (always zero:disabled)
inout[15:0] s_dq //sdram inout/output data
    );

 /*
 sdram_controller m1(
//fpga to controller
.clk(CLK_OUT), //clk=165MHz
.rst_n(rst_n),
.rw(rw), // 1:read , 0:write
.rw_en(rw_en), //must be asserted before read/write
.f_addr(f_addr_q), //14:2=row(13)  , 1:0=bank(2) , no need for column
  address since full page mode will always start from zero and end with 512 words
.f2s_data(f2s_data_q), //fpga-to-sdram data , write data burst will
  start at assertion of f2s_data_valid
.s2f_data(s2f_data), //sdram to fpga data
.s2f_data_valid(s2f_data_valid),  //asserts while
  burst-reading(data is available at output UNTIL the next rising edge)
.f2s_data_valid(f2s_data_valid), //asserts while
  burst-writing(data must be available at input BEFORE the next rising edge)
.ready(ready), //"1" if sdram is available for nxt read/write operation
//controller to sdram
.s_clk(sdram_clk),
.s_cke(sdram_cke),
.s_cs_n(sdram_cs_n),
.s_ras_n(sdram_ras_n ),
.s_cas_n(sdram_cas_n),
.s_we_n(sdram_we_n),
.s_addr(sdram_addr),
.s_ba(sdram_ba),
.LDQM(sdram_dqm[0]),
.HDQM(sdram_dqm[1]),
.s_dq(sdram_dq)
    );
 */
```

```
phase_diff_180 p1(
.datain_h(1'b0),
.datain_l(1'b1),
.outclock(clk),
.dataout(s_clk));
 //s_clock(clk input to sdram) is 180 degrees lagging from main clock
  to solve the hold-setup time requirements of sdram
 /*ODDR2#(.DDR_ALIGNMENT("NONE"), .INIT(1'b0),.SRTYPE("SYNC")) oddr2_primitive
 (
.D0(1'b0), //1'b0
.D1(1'b1 ), //1'b1
.C0(clk),
.C1(~clk),
.CE(1'b1),
.R(1'b0),
.S(1'b0),write
.Q(s_clk)
);*/
//FSM states //initialize
 localparam[3:0]  start=0,
precharge_init=1,
refresh_1=2,
refresh_2=3,
load_mode_reg=4,
//normal operation
idle=5,
read=6,
read_data=7,
write=8,
write_burst=9,
//refresh every 7.81us
refresh=10,

delay=11; //waiting state for any amount of delay needed

//minimum time specs needed(in clks for 165MHz(6ns))
localparam[3:0] t_RP=2, //15ns(precharge)
t_RC=7, //60ns(active to active,ref to ref)
t_MRD=2, //2 clk,(mode register) /2/
t_RCD=2, //15ns (active to read/write)
t_WR=2,
     //2clk delay after writing before manual/auto precharge can start
t_CL=3; //CAS latency(delay of data_out after read command)

//commands {cs_n,ras_n,cas_n,we_n} REFER TO THE DATASHEET: winbond W9825G6KH
localparam[3:0]  cmd_precharge=4'b0010,
  cmd_NOP=4'b1111,
  cmd_activate=4'b0011,
  cmd_write=4'b0100,
  cmd_read=4'b0101,
  cmd_setmode=4'b0000,
  cmd_refresh=4'b0001;

reg[3:0] state_q,state_d; //_q is registered output, _d is input to DFF
reg[3:0] nxt_q,nxt_d; //state after next state
reg[3:0] cmd_q,cmd_d; //{cs_n,ras_n,cas_n,we_n}
reg[15:0] delay_ctr_q,delay_ctr_d;
 //stores delay needed(max is 200us for the initialization sequence)
reg[10:0] refresh_ctr_q=0,refresh_ctr_d;
```

```
reg refresh_flag_q,refresh_flag_d;
reg[9:0] burst_index_q=0,burst_index_d;
 //stores the data left to be burst(512 for full page burst)
reg rw_d,rw_q,rw_en_q,rw_en_d;

//buffer for output for a glitch-free signal
reg[12:0] s_addr_q,s_addr_d;
reg[1:0] s_ba_q,s_ba_d;
reg[15:0] s_dq_q,s_dq_d;
reg tri_q,tri_d;

//buffer for input
reg[14:0] f_addr_q,f_addr_d;
reg[15:0] f2s_data_q,f2s_data_d;
reg[15:0] s2f_data_q,s2f_data_d;
reg s2f_data_valid_q,s2f_data_valid_d;




//register operation
always @(posedge clk,negedge rst_n) begin
if(!rst_n) begin
state_q<=start;
nxt_q<=start;
cmd_q<=cmd_NOP;
delay_ctr_q<=0;
refresh_ctr_q<=0;
s_addr_q<=0;
tri_q<=0;
rw_q<=0;
rw_en_q<=0;

s_ba_q<=0;
s_dq_q<=0;
f_addr_q<=0;
rw_q<=0;
f2s_data_q<=0;
s2f_data_q<=0;
s2f_data_valid_q<=0;
rw_q<=0;
refresh_flag_q<=0;
burst_index_q<=0;
end
else begin
state_q<=state_d;
nxt_q<=nxt_d;
cmd_q<=cmd_d;
delay_ctr_q<=delay_ctr_d;
refresh_ctr_q<=refresh_ctr_d;
s_addr_q<=s_addr_d;
tri_q<=tri_d;
refresh_flag_q<=refresh_flag_d;
burst_index_q<=burst_index_d;

s_ba_q<=s_ba_d;
s_dq_q<=s_dq_d;
f_addr_q<=f_addr_d;
rw_q<=rw_d;
```

```
f2s_data_q<=f2s_data_d;
s2f_data_q<=s2f_data_d;
s2f_data_valid_q<=s2f_data_valid_d;
rw_q<=rw_d;
rw_en_q<=rw_en_d;
end
end


//next-state logics
always @* begin
state_d=state_q;
nxt_d=nxt_q;
cmd_d=cmd_NOP; //always default to No Operation
delay_ctr_d=delay_ctr_q;
ready=0;
s_addr_d=s_addr_q;
s_ba_d=s_ba_q;
s_dq_d=s_dq_q;
f_addr_d=f_addr_q;
rw_d=rw_q;
f2s_data_d=f2s_data_q;
s2f_data_d=s2f_data_q;
tri_d=0;
s2f_data_valid_d=1'b0;
f2s_data_valid=1'b0;
burst_index_d=burst_index_q;
rw_d=rw_q;
rw_en_d=rw_en_q;

//refresh every 7.8us or else data will be lost.
refresh_flag_d=refresh_flag_q;
refresh_ctr_d=refresh_ctr_q+1'b1;
if(refresh_ctr_q==770) begin //7.7 us
refresh_ctr_d=0;
refresh_flag_d=1;
end



case(state_q)
///////////////BEGIN:INITIALIZE///////////////
delay: begin
   //wait here for a delay specified by delay_ctr_q(parameter in time specs)
delay_ctr_d=delay_ctr_q-1'b1;
if(delay_ctr_d==0) state_d=nxt_q;
if(nxt_q==write) tri_d=1;
 end
start: begin //initiliaze after power-up
state_d=delay;
nxt_d=precharge_init;
delay_ctr_d=16'd33_000; //wait for 200us
s_addr_d=0;
s_ba_d=0;
 end
precharge_init: begin //precharge ALL banks (A10 must be high)
state_d=delay;
nxt_d=refresh_1;
delay_ctr_d=t_RP-1;
```

```
cmd_d=cmd_precharge;
s_addr_d[10]=1'b1;
 end
refresh_1: begin
state_d=delay;
nxt_d=refresh_2;
delay_ctr_d=t_RC-1;
cmd_d=cmd_refresh;
  end
refresh_2: begin
state_d=delay;
nxt_d=load_mode_reg;
delay_ctr_d=t_RC-1;
cmd_d=cmd_refresh;
  end
  load_mode_reg: begin
state_d=delay;
nxt_d=idle;
delay_ctr_d=t_MRD-1;
cmd_d=cmd_setmode;
s_addr_d=13'b 000_0_00_011_0_111;
      //{reserved,writemode,reserved,CL,AddressingMode,BurstLength}
s_ba_d=2'b00; //reserved
  end
 ///////////////END:INITIALIZE///////////////

///////////////BEGIN:NORMAL OPERATION///////////////
     idle: begin
ready=rw_en_q? 0:1;
if(rw_en_q) begin //permission granted for r/w operation
state_d=delay;
cmd_d=cmd_activate;
delay_ctr_d=t_RCD-1;
nxt_d=rw_q?read:write;
burst_index_d=0;
rw_en_d=1'b0;
{s_addr_d,s_ba_d}=f_addr_q;//row + bank addr
end
else if(refresh_flag_q || rw_en) begin
      //refresh every 7.7us and BEFORE start of burst read/write operations
state_d=delay;
nxt_d=refresh;
delay_ctr_d=t_RP-1;
cmd_d=cmd_precharge;
       //precharge all banks first before auto-refresh
s_addr_d[10]=1'b1;
refresh_flag_d=0;
if(rw_en) begin
rw_en_d=rw_en;
f_addr_d=f_addr;
rw_d=rw;
end
end
  end
     refresh: begin
 state_d=delay;
 nxt_d=idle;
 delay_ctr_d=t_RC-1;
 cmd_d=cmd_refresh;
```

```
end
read: begin
state_d=delay;
delay_ctr_d=t_CL;
      //not subtracted by one since the sdram is "late" by half a cycle
      so register is one clk after the expected clock latency delay
cmd_d=cmd_read;
s_addr_d=0;
      //what column to activate(in full page mode, column starts at
      LEFTMOST which is zero)
s_ba_d=f_addr_q[1:0]; //what bank to activate
s_addr_d[10]=1'b0; //no auto-precharge for
      full page burst
nxt_d=read_data;
end
 read_data: begin //read data after CAS latency of 3 clk
s2f_data_d=s_dq;
s2f_data_valid_d=1'b1;
burst_index_d=burst_index_q+1;
if(burst_index_q==512) begin
      //if all 512 burst data is already finished, precharge then go back to idle
s2f_data_valid_d=1'b0;
state_d=delay;
nxt_d=idle;
delay_ctr_d=t_RP-1;
cmd_d=cmd_precharge;
end
end
 write: begin
f2s_data_d=f2s_data; //write data
f2s_data_valid=1'b1;
s_addr_d=0;
      //what column to activate(in full page mode, column starts at LEFTMOST
      which is zero)
s_ba_d=f_addr_q[1:0];
s_addr_d[10]=1'b0; //no auto-precharge for full page burst
tri_d=1'b1; //tristate buffer on since we output/write signals
cmd_d=cmd_write;
state_d=write_burst;
burst_index_d=burst_index_q+1;
   end
  write_burst: begin    //write data burst will start at assertion of f2s_data_valid

f2s_data_d=f2s_data; //write data
f2s_data_valid=1'b1;
tri_d=1'b1; //tristate buffer on since we
       output/write signals
burst_index_d=burst_index_q+1;

if(burst_index_q==512) begin
       //if all 512 burst data is already finished, precharge then go back to idle
tri_d=0;
state_d=delay;
f2s_data_valid=1'b0;
nxt_d=idle;
delay_ctr_d=t_RP+t_WR-1;
cmd_d=cmd_precharge;
end
end
```

```
  ///////////////END:NORMAL OPERATION///////////////

default: state_d=start;
endcase




end

//assign the outputs to corresponding buffers
assign s_cs_n=cmd_q[3],
 s_ras_n=cmd_q[2],
 s_cas_n=cmd_q[1],
 s_we_n=cmd_q[0];
assign s_cke=1'b1;
assign LDQM=1'b0,
 HDQM=1'b0;
assign s_addr=s_addr_q;
assign s_ba=s_ba_q;
   assign s_dq=tri_q? f2s_data_q:16'hzzzz;
   //tri-state output,tri=1 for write , tri=0 for read(hi-Z)
assign s2f_data=s2f_data_q;
assign s2f_data_valid=s2f_data_valid_q;


endmodule
```

8.4.8 async_fifo.v

```
`timescale 1ns / 1ps

module asyn_fifo
#(
parameter DATA_WIDTH=8,
 FIFO_DEPTH_WIDTH=11  //total depth will then be
      2**FIFO_DEPTH_WIDTH
)
(
input wire rst_n,
input wire clk_write,clk_read, //clock input from both domains
input wire write,read,
input wire [DATA_WIDTH-1:0] data_write, //input FROM write clock domain
output wire [DATA_WIDTH-1:0] data_read, //output TO read clock domain
output reg full,empty, //full=sync to write domain clk , empty=sync to
 read domain clk
output reg[FIFO_DEPTH_WIDTH-1:0] data_count_w,data_count_r
 //counts number of data left in fifo memory(sync to either write or read clk)
    );


 /*
 async_fifo #(.DATA_WIDTH(16),.FIFO_DEPTH_WIDTH(10)) m2 //1024x16 FIFO mem
(
.rst_n(rst_n),
.clk_write(),
.clk_read(), //clock input from both domains
.write(),
.read(),
.data_write(), //input FROM write clock domain
.data_read(), //output TO read clock domain
.full(),
.empty(), //full=sync to write domain clk , empty=sync to read domain clk
..data_count_w(),
.data_count_r() //counts number of data left in fifo memory(sync to either
  write or read clk)
    );
 */


 localparam FIFO_DEPTH=2**FIFO_DEPTH_WIDTH;

 initial begin
full=0;
empty=1;
 end


 //////////////////WRITE CLOCK DOMAIN//////////////////////////////
 reg[FIFO_DEPTH_WIDTH:0] w_ptr_q=0; //binary counter for write pointer
 reg[FIFO_DEPTH_WIDTH:0] r_ptr_sync; //binary pointer for read pointer sync
  to write clk
 wire[FIFO_DEPTH_WIDTH:0] w_grey,w_grey_nxt; //grey counter for write pointer
 reg[FIFO_DEPTH_WIDTH:0] r_grey_sync; //grey counter for the
  read pointer synchronized to write clock

 wire we;
 reg[3:0] i; //log_2(FIFO_DEPTH_WIDTH)

 assign w_grey=w_ptr_q^(w_ptr_q>>1); //binary to grey code conversion for
  current write pointer
 assign w_grey_nxt=(w_ptr_q+1'b1)^((w_ptr_q+1'b1)>>1);  //next grey code
 assign we= write && !full;

 //register operation
 always @(posedge clk_write,negedge rst_n) begin
if(!rst_n) begin
w_ptr_q<=0;
full<=0;
end
else begin
if(write && !full) begin //write condition
w_ptr_q<=w_ptr_q+1'b1;
full <= w_grey_nxt == {~r_grey_sync[FIFO_DEPTH_WIDTH:FIFO_DEPTH_WIDTH-1]
    ,r_grey_sync[FIFO_DEPTH_WIDTH-2:0]};
    //algorithm for full logic which can be observed on the grey code table
end
else full <= w_grey == {~r_grey_sync[FIFO_DEPTH_WIDTH:FIFO_DEPTH_WIDTH-1]
   ,r_grey_sync[FIFO_DEPTH_WIDTH-2:0]};

for(i=0;i<=FIFO_DEPTH_WIDTH;i=i+1) r_ptr_sync[i]=^(r_grey_sync>>i);
   //grey code to binary converter
data_count_w <= (w_ptr_q>=r_ptr_sync)? (w_ptr_q-r_ptr_sync):
   (FIFO_DEPTH-r_ptr_sync+w_ptr_q);
   //compares write pointer and sync read pointer to generate data_count
end
 end
```

```
/////////////////////////////////////////////////////////////////////


  ///////////////////READ CLOCK DOMAIN//////////////////////////////
 reg[FIFO_DEPTH_WIDTH:0] r_ptr_q=0; //binary counter for read pointer
 wire[FIFO_DEPTH_WIDTH:0] r_ptr_d;
 reg[FIFO_DEPTH_WIDTH:0] w_ptr_sync; //binary counter for write pointer sync
  to read clk
 reg[FIFO_DEPTH_WIDTH:0] w_grey_sync; //grey counter for the write pointer
  synchronized to read clock
 wire[FIFO_DEPTH_WIDTH:0] r_grey,r_grey_nxt; //grey counter for read pointer


 assign r_grey= r_ptr_q^(r_ptr_q>>1);  //binary to grey code conversion
 assign r_grey_nxt= (r_ptr_q+1'b1)^((r_ptr_q+1'b1)>>1); //next grey code
 assign r_ptr_d= (read && !empty)? r_ptr_q+1'b1:r_ptr_q;

 //register operation
 always @(posedge clk_read,negedge rst_n) begin
if(!rst_n) begin
r_ptr_q<=0;
empty<=1;
end
else begin
r_ptr_q<=r_ptr_d;
if(read && !empty) empty <= r_grey_nxt==w_grey_sync;//empty condition
else empty <= r_grey==w_grey_sync;

for(i=0;i<=FIFO_DEPTH_WIDTH;i=i+1) w_ptr_sync[i]=^(w_grey_sync>>i);
   //grey code to binary converter
data_count_r = (w_ptr_q>=r_ptr_sync)? (w_ptr_q-r_ptr_sync):
   (FIFO_DEPTH-r_ptr_sync+w_ptr_q); //compares read pointer to sync write
   pointer to generate data_count
end
 end
 ////////////////////////////////////////////////////////////////////////


 ////////////////////CLOCK DOMAIN CROSSING/////////////////////////////
 reg[FIFO_DEPTH_WIDTH:0] r_grey_sync_temp;
 reg[FIFO_DEPTH_WIDTH:0] w_grey_sync_temp;
 always @(posedge clk_write) begin //2 D-Flipflops for reduced metastability
  in clock domain crossing from READ DOMAIN to WRITE DOMAIN
r_grey_sync_temp<=r_grey;
r_grey_sync<=r_grey_sync_temp;
 end
 always @(posedge clk_read) begin //2 D-Flipflops for reduced metastability
  in clock domain crossing from WRITE DOMAIN to READ DOMAIN
w_grey_sync_temp<=w_grey;
w_grey_sync<=w_grey_sync_temp;
 end

/////////////////////////////////////////////////////////////////////////



//instantiation of dual port block ram
dual_port_sync #(.ADDR_WIDTH(FIFO_DEPTH_WIDTH) , .DATA_WIDTH(DATA_WIDTH)) m0
```

```
(
.clk_r(clk_read),
.clk_w(clk_write),
.we(we),
.din(data_write),
.addr_a(w_ptr_q[FIFO_DEPTH_WIDTH-1:0]), //write address
.addr_b(r_ptr_d[FIFO_DEPTH_WIDTH-1:0] ), //read address ,addr_b is already
  buffered inside this module so we will use the "_d" ptr to advance the data(not "_q")
.dout(data_read)
);

endmodule



//inference template for dual port block ram
module dual_port_sync
#(
parameter ADDR_WIDTH=11, //2k by 8 dual port synchronous ram(16k block ram)
 DATA_WIDTH=8
)
(
input clk_r,
input clk_w,
input we,
input[DATA_WIDTH-1:0] din,
input[ADDR_WIDTH-1:0] addr_a,addr_b, //addr_a for write, addr_b for read
output[DATA_WIDTH-1:0] dout
);

reg[DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
reg[ADDR_WIDTH-1:0] addr_b_q;

always @(posedge clk_w) begin
if(we) ram[addr_a]<=din;
end
always @(posedge clk_r) begin
addr_b_q<=addr_b;
end
assign dout=ram[addr_b_q];

endmodule
```

8.4.9 img_reader.sv

```
/*
 * custom peripheral
 */

module img_reader(
input logic        clk,
    input logic     reset,
input logic [31:0]  writedata, // must be multiple of 8
input logic     write,
input      chipselect,
input logic [7:0]  address,
input logic      read,
input logic [7:0] VGA_R,
input logic [7:0] VGA_G,
input logic [7:0] VGA_B,
input logic HSYNC,
input logic VSYNC,

output logic [31:0] readdata,
output logic get_img); // must be multiple of 8!

assign readdata = get_img ? {VGA_R, VGA_G, VGA_B, 8'd0} : offset_to_zeros;

logic [31:0] offset_to_zeros = 32'd0;
logic [31:0] counter = 32'd0;
logic reset_next = 1'b0;

always_ff @(posedge clk) begin
// RESET
if (reset_next) begin
counter <= 32'd0;
reset_next <= 1'b0;
end else if (reset) begin
get_img <= 1'd0;
counter <= 32'd0;
offset_to_zeros <= 32'd0;
reset_next <= 1'b0;
end else if (chipselect && read) begin
case(address)
8'b0: begin
get_img <= 1'd1;
//offset_to_zeros <= 32'd0; CHANGED THIS
counter <= counter + 32'd1;
if (!VSYNC) begin
offset_to_zeros <= counter;
//counter <= 32'd5; // CHANGED THIS
reset_next <= 1'b1;
end
end
8'b1: begin
get_img <= 1'd0;
counter <= 32'd0;
end
default: ;
endcase
end else begin
//get_img <= 1'd0; // here::
//counter <= 32'd0;
//offset_to_zeros <= 32'd0;
end
end


endmodule
```

8.4.10 img_reader.h

```
#ifndef _IMG_READER_H
#define _IMG_READER_H
```

```
#include <linux/ioctl.h>


/* ioctls and their arguments */
/* https://stackoverflow.com/questions/22496123/
what-is-the-meaning-of-this-macro-iormy-macig-0-int */
#define MAGIC 'q'
#define IMG_WRITE _IOW(MAGIC, 1, int)
#define IMG_READ  _IOR(MAGIC, 2, int)


#endif
```

8.4.11 img_reader.c

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/delay.h>
#include "img_reader.h"

#define DRIVER_NAME "img_reader"

/* Device registers */
#define ADDR(x) (x+1)  // local helper that maps x (address) to right bit
in target space (returns new address)

struct my_comp {
    struct resource res; /* Resource: our registers */
void __iomem *virtbase; /* Where registers can be accessed in memory */
    int value;
} dev;

// TODO if desired to write images
static void write_value(int val[], int max_addr){
    /* iowrite8(value, adress-to-write-to)*/
    //int addr = 0;
    //int[] val = *val_addr
    /* val_addr is pointer to array */
    //int arr;
    int addr_local = 0;
    //int max_addr = (sizeof(val)*8)/8;  // sizeof gives bytes
    pr_info(" max_addr %d", max_addr);
    pr_info(" val[0] %d", val[0]);
    pr_info(" val[1] t %d", (int) *(val+1));
    pr_info(" val[9] %d", val[9]);
    //iowrite8(val[0], dev.virtbase); // write 8 bits
    for (addr_local = 0; addr_local < max_addr; addr_local = addr_local +1){
        // arr[addr]
       iowrite16(val[addr_local], dev.virtbase + addr_local*2); // write 8 bits
       pr_info("written %d to %d (%d) with size %d", val[addr_local], addr_local,
       dev.virtbase + addr_local, sizeof(val[addr_local]));
    }
    pr_info("Kwrite_value: done iowrite");
};


// cannot return array so will return pointer to array
static int* read_img(int max_reads){
    /* ioread(adress-to-read-from)*/
    //static int out[max_addr-addr]; // doesnt work because dynamic size and static
    (needs static to retain mem addr outside the fucntion)
    //pr_info("trying kmalloc");
    int* out_ptr;
    if ( (out_ptr = kmalloc(sizeof(int)*(max_reads), GFP_USER)) == -1){
        pr_info("ERROR: could not allocate %d bytes in memory\n", sizeof(int)*(max_reads));
        } // dynamic allocation
    int i_read;
    for (i_read = 0; i_read < max_reads; i_read = i_read +1){
        //*(out_ptr+addr_local) = ioread16(dev.virtbase+addr+addr_local);
        *(out_ptr+i_read) = ioread32(dev.virtbase+0); // here;
        ndelay(30000);
        //pr_info("Kread_value: from %d (%d) read %d (%b)", i_read, dev.virtbase,
        *(out_ptr+i_read), *(out_ptr+i_read));
    }
    int offset_zero = ioread32(dev.virtbase+4);
    //*(out_ptr+i_read+1) = offset_zero;
    pr_info("Kread_value: offset zero %d", offset_zero);
    pr_info("\n", offset_zero);
    //pr_info("Kread_value: returning %d", out_ptr);
    return out_ptr;
};

static long img_reader_ioctl(struct file *f, unsigned int cmd, unsigned long val_arg)
{
    //int size = 640;
    int size = 640*480;
    // new array of same size as input
    // changes
    int *arr_ptr = val_arg;
    //int (*a)[10] = 1;
    //int val_local[640];
    //pr_info("iooctl: size = %d, sizeof(val_local) = %d\n", size, sizeof(val_local));


    switch(cmd){
        case IMG_WRITE:;
            /* copy_from_user(to, from, length) */
            /* copy from arg to vla (to dev.virtbase)*/
            //if (copy_from_user(&val_local, (int *) val_arg, sizeof(int)))
            // if (copy_from_user(val_local, (*arr_ptr), sizeof(int)))
            int i = 0;
            /*for (i = 0; i < 10; i = i+1){
                if (copy_from_user(val_local+i, arr_ptr+i, sizeof(val_local)))
                    return -EACCES;
                pr_info("ictl_write val_local[%d]: %d , %d \t arr_ptr %d, %d, %d", i,
                *(val_local+i), val_local[i], *(arr_ptr+i), arr_ptr[i]);
            }*/
            //if (copy_from_user(val_local, arr_ptr, sizeof(val_local)))
            //        return -EACCES;
            pr_info("ictl_write: done copying");
            //write_value(val_local, size);
            pr_info("ioctl_write: done writing");
            break;

        case IMG_READ:;
            int* arr_ptr_local = read_img(size);
            //pr_info("ictl_reading: done reading %d", arr_ptr_local);
            //pr_info("ictl_reading: val_local[0] %d, %d", *(arr_ptr_local), (int) arr_ptr_local[0]);
            //pr_info("\n");
            if (copy_to_user(arr_ptr, arr_ptr_local, sizeof(int) * size)) {
pr_info("copy to user failed");
return -EACCES;
    }
            kfree(arr_ptr_local);
            break;

        default:
            return -EINVAL;
        }

    return 0;
};

/* 1) The operations our device knows how to do (turn ioctl into file operations) */
static const struct file_operations img_reader_fops = {
.owner = THIS_MODULE,
.unlocked_ioctl = img_reader_ioctl
};

/* 2) Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice img_reader_misc_device = {
.minor = MISC_DYNAMIC_MINOR,
.name = DRIVER_NAME,
.fops = &img_reader_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
// 3.1) setup/register device
static int __init img_reader_probe(struct platform_device *pdev)
{
    //int initial = 0;
    int initial[] = {0};
    long initial_ptr = (long) &initial;
int ret;

/* Register ourselves as a misc device: creates /dev/vga_ball */
ret = misc_register(&img_reader_misc_device);

/* Get the address of our registers from the device tree */
// int of_address_to_resource(struct device_node *dev, int index,
 struct resource *r)¶
```

```
ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
if (ret) {
ret = -ENOENT;
goto out_deregister;
}

/* Make sure we can use these registers */
if (request_mem_region(dev.res.start, resource_size(&dev.res),
       DRIVER_NAME) == NULL) {
ret = -EBUSY;
goto out_deregister;
}

/* Arrange access to our registers */
dev.virtbase = of_iomap(pdev->dev.of_node, 0);
if (dev.virtbase == NULL) {
ret = -ENOMEM;
goto out_release_mem_region;
}

/* Set an initial color */
    pr_info("WRITING INITIAL");
    write_value(initial, 1);
    pr_info("DONE WRITING INITIAL");

return 0;

// jumped to from before in case of error
out_release_mem_region:
release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
misc_deregister(&img_reader_misc_device);
return ret;
}

/* Clean-up code: release resources */
// 3.2) remove
static int img_reader_remove(struct platform_device *pdev)
{
iounmap(dev.virtbase);
release_mem_region(dev.res.start, resource_size(&dev.res));
misc_deregister(&img_reader_misc_device);
return 0;
}


/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id img_reader_of_match[] = {
{ .compatible = "csee4840,img_reader-1.0" },
{},
};
MODULE_DEVICE_TABLE(of, img_reader_of_match);
#endif



/* Information for registering ourselves as a "platform" driver */
static struct platform_driver img_reader_driver = {
```

```
.driver = {
.name = DRIVER_NAME,
.owner = THIS_MODULE,
.of_match_table = of_match_ptr(img_reader_of_match),
},
.remove = __exit_p(img_reader_remove),
};

/* Called when the module is loaded: set things up */
static int __init img_reader_init(void)
{
pr_info(DRIVER_NAME ": init\n");
return platform_driver_probe(&img_reader_driver, img_reader_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit img_reader_exit(void)
{
platform_driver_unregister(&img_reader_driver);
pr_info(DRIVER_NAME ": exit\n");
}

module_init(img_reader_init);
module_exit(img_reader_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Lennart Schulze");
MODULE_DESCRIPTION("Img reader driver for FPGA to get camera data");
```

8.4.12 test.c

```
/*
 * Userspace program that communicates with the vga_ball device driver
 * through ioctls
 *
 * Stephen A. Edwards
 * Columbia University
 */

#include <stdio.h>
#include "cnn_io.h"
#include "img_reader.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#define _OPEN_SYS_ITOA_EXT

int cnn_fd;
int img_reader_fd;
char path_to_img[] = "./rgb888img";
char path_to_img_bin[] = "./rgb888imgbin";
```

```
// as long as numrows and cols is correct,
//it doesnt matter that last value is offset
void print_image(int *arr, int numrows, int numcols) {
  int *ptr;
  char* rbyte, *gbyte, *bbyte, *lsbyte;
  FILE* fp, *fpbin;
  char buffer[4];
  fp = fopen(path_to_img, "w");
  if (fp == NULL) {
      perror("Failed: ");
      return 1;
  }
  fpbin = fopen(path_to_img_bin, "w");
  if (fpbin == NULL) {
      perror("Failed: ");
      return 1;
  }

  for (int i = 0; i < numrows; i++) {
      for (int j = 0; j < numcols; j++) {
          //if (i % 12 != 0) continue;
          ptr = (int *) arr + i*numcols + j;
          //printf("Image at row %d, col %d:\n", i, j);
          /*lsbyte = (char *) ptr + 0;
          sprintf(buffer, "%d", *lsbyte);
          fputs(buffer, fp);
          fputs(" ", fp);*/
          //printf("Least significant byte is: %s ", buffer);
          rbyte = (char *) ptr + 3;
          sprintf(buffer, "%d", *rbyte);
          fputs(buffer, fp);
          fwrite(rbyte, sizeof(char), 1, fpbin);
          fputs(" ", fp);
          //printf("Red byte is: %s ", buffer);
          gbyte = (char *) ptr + 2;
          sprintf(buffer, "%d", *gbyte);
          fputs(buffer, fp);
          fwrite(gbyte, sizeof(char), 1, fpbin);
          fputs(" ", fp);
          //printf("Green byte is: %s ", buffer);
          bbyte = (char *) ptr + 1;
          sprintf(buffer, "%d", *bbyte);
          fputs(buffer, fp);
          fwrite(bbyte, sizeof(char), 1, fpbin);
          fputs("\n", fp);
          //printf("Blue byte is: %s \n", buffer);
      }
  }

  fclose(fp);
  fclose(fpbin);

}
/* Read and print the background color */
void print_value(void) {
  int value_local;

  /* behavior of ioctl defined in vga_ball.c args(file, cmd, arg) */
  /* this writes the color of the background from &vla */
  if (ioctl(cnn_fd, CNN_READ_VAL, &value_local)) {
      perror("ioctl(CNN_WRITE_VAL) failed");
      goto end;
  }
  printf("%d", value_local);

  end:;
}

// inspired by online resources on array rotating
void reverse(int arr[], int start, int end)
{
  while (start <= end)
  {
    int temp = arr[start];
    arr[start] = arr[end];
    arr[end] = temp;
    start++;
    end--;
  }
}
// Function to Rotate k elements to right
void rotatetoright(int arr[], int num_elem, int k)
{
  // Reverse first n-k elements
  reverse(arr, 0, num_elem - k - 1);
  // Reverse last k elements
  reverse(arr, num_elem - k, num_elem - 1);
  // Reverse whole array
  reverse(arr, 0, num_elem - 1);
}

/* Set the background color */
int* get_value(int mode)
{
  int* value_local = malloc((640 * 480 + 1) * sizeof(int));
  // plus 1 for the offset bit
  char* rbyte, *gbyte, *bbyte, *lsbyte;
  // To check that the least significant byte is always 0
  if (mode==0){ // regular mode
    printf("get_val: READ_VAL");
    if (ioctl(cnn_fd, CNN_READ_VAL, value_local)) {
        perror("ioctl(CNN_READ_VAL) failed");
        return -1;
    }
  } else if (mode==1){ // read image mode
    printf("get_val: READ_IMG");
    if (ioctl(img_reader_fd, IMG_READ, value_local)) {
        perror("ioctl(IMG_READ) failed");
        return -1;
    }
    lsbyte = value_local;
    int offset = *(value_local+640*480);
    printf("pointer is %d\n", offset);
    printf("Least significant byte is: %d\n", *lsbyte);
    rbyte = (char *) value_local + 3;
    printf("Red byte is: %d\n", *rbyte);
    gbyte = (char *) value_local + 2;
    printf("Green byte is: %d\n", *gbyte);
    bbyte = (char *) value_local + 1;
    printf("Blue byte is: %d\n", *bbyte);
    //printf("%d", value_local);

    //memmove(&lsbyte[k+1], &lsbyte[k], (numItems-k-1)*sizeof(double));
    //items[k] = value;
    rotatetoright(lsbyte, 640*480, 500);
  }
  printf("Uget_value: ptr: %d \t ptr[0]: %d\n", value_local, *(value_local));

  return value_local;
};

/* Send value to FPGA */
void set_value(const int *value_local, int target)
{
    switch (target)
    {
    case 0:
        if (ioctl(cnn_fd, CNN_WRITE_VAL, value_local)) {
            perror("ioctl(CNN_WRITE_VAL) failed");
            return;
        }
        break;

    default:
        break;
    }

    printf("\nUwrite_value: written:");
    printf("%d", (int) value_local);
};

int main()
{
    int value_local;
    static const char filename_cnn[] = "/dev/cnn_mem";
    static const char filename_img_reader[] = "/dev/img_reader";
    int numlines=480;
    int size=640;
    int i;  // Loop variable

    int weights_l1 = {0};
    int weights_l2 = {0};
    int weights_l3 = {0};


    printf("CNN Userspace program started\n");

    /*if ( (cnn_fd = open(filename_cnn, O_RDWR)) == -1) {
        fprintf(stderr, "could not open %s\n", filename_cnn);
        return -1;
    }*/
    if ( (img_reader_fd = open(filename_img_reader, O_RDWR)) == -1) {
        fprintf(stderr, "could not open %s\n", filename_img_reader);
        return -1;
    }

    /*int* ptr[numlines];
    for (i = 0; i < numlines; i++) {
      printf("Line number %d\n", i);
      ptr[i] = get_value(1);
    }
    for (i = 0; i < numlines; i++) {
      free(ptr[i]);
    }*/
    usleep(2000000);
    int *ptr = get_value(1);
    print_image(ptr, 480, 640);
    printf("main: got value %d", ptr);
    printf("main: got value %d", ptr[0]);
    free(ptr);

    usleep(400000);
    //}

    printf("CNN Userspace program terminating\n");
    return 0;
}
```

8.4.13 wrapper.py

```
import ctypes
import numpy as np
import os
from data_utils import transform
from cnn import Cnn
import torch

if __name__ == "__main__":
    pass
    mylib = ctypes.CDLL(os.path.join(os.path.dirname(os.path.abspath(__file__)),
    'test.so'))
    cnn = Cnn()
    checkpoint = torch.load("model.pth")
    cnn.load_state_dict(checkpoint['model_state_dict'])
    cnn.eval()
    # for loop begin
    # read from c file
    mylib.main()

    with open(os.path.join(os.path.dirname(os.path.abspath(__file__)),'rgb888img0'),
    'r') as f:
        pixel_values = np.loadtxt(f)
    # postprocess
    in_tensor = transform(pixel_values)

    # CNN
    pred = cnn(in_tensor)
    score = pred
    print("score:", score)

    # send to c file
    #mylib.write_to_hex(score)

    # for loop end
```

8.4.14 cnn.py

```
import numpy as np
import os
from datetime import date

import torch
import torch.nn as nn
import torch.nn.functional as F

from data_utils import PlayingCardsSet

class Cnn(nn.Module):
    def __init__(self) -> None:
        super().__init__()
        #act =
        self.layers = []
        self.layers.append(nn.Conv2d(3, 32, 3, stride=2))
        self.layers.append(nn.ReLU())
        self.layers.append(nn.MaxPool2d(2, 2))
        self.layers.append(nn.Conv2d(32, 128, 3, stride=2))
        self.layers.append(nn.ReLU())
        self.layers.append(nn.MaxPool2d(2, 2))
        self.layers.append(nn.Conv2d(128, 256, 3, padding=1, stride=1))
        self.layers.append(nn.ReLU())
        #self.layers.append(nn.MaxPool2d(2, 2))
        #self.layers.append(nn.Conv2d(128, 256, 3, padding=1))
        #self.layers.append(act)

        self.layers.append(nn.Linear(256*5*5,1000))
        self.layers.append(nn.ReLU())
        #self.layers.append(nn.Linear(8000,1000))
        #self.layers.append(act)
        self.layers.append(nn.Linear(1000,200))
        self.layers.append(nn.ReLU())
        self.layers.append(nn.Linear(200,4))
        self.layers = nn.ModuleList(self.layers)
        print(self)

    def forward(self, input):
        output = input
        first_linear = False
        for i_layer, layer in enumerate(self.layers):
            if first_linear is False and isinstance(layer, nn.Linear):
                output = torch.flatten(output, 1)
                first_linear = True
            output = layer(output)
            #print(output.shape)
            #if not isinstance(layer, nn.MaxPool2d) and i_layer < len(self.layers):
            #    output = torch.nn.functional.relu(output)
        return output

def train(model, src_path, device):
    src_path = src_path#os.path.join(os.path.dirname(os.path.abspath(__file__)),
    "../../Data/PlayingCards")
    datasets = {}
    datasets["train"] = PlayingCardsSet(src_path, "train")
    datasets["test"] = PlayingCardsSet(src_path, "test")
    print([len(dsi) for dsi in datasets.values()])
```

```
dataloaders = {}
batch_size = 16
dataloaders["train"] = torch.utils.data.DataLoader(datasets["train"],
batch_size=batch_size,
                                     shuffle=True, num_workers=0)
dataloaders["test"] = torch.utils.data.DataLoader(datasets["test"],
batch_size=batch_size,
                                     shuffle=False, num_workers=0)
#print(next(iter(dataloaders["train"]))["image"].shape)

loss_criterion = nn.CrossEntropyLoss() # includes softmax on output of
the network
optimizer = torch.optim.Adam(model.parameters(), lr=0.0005) #0.000005

num_epochs = 10
for epoch in range(num_epochs):
    running_loss = 0.0
    running_acc = 0.0
    for i_batch, batch in enumerate(dataloaders["train"]):
        # get batch data
        gt_labels = batch["label"].to(device)
        images = batch["image"].to(device)
        print("gt\t", gt_labels)
        print([batch["filename"][i].split("/")[-2:] for i in
        range(len(batch["filename"]))])

        # reset gradients to zero
        optimizer.zero_grad()

        # forward (get prediction)
        pred = model(images)
        pred_labels = torch.argmax(torch.softmax(pred, -1), -1)
        #print("pred\t", pred_labels)
        # backward (get gradients)
        acc = (pred_labels == gt_labels).detach().double().mean()
        loss = loss_criterion(pred, gt_labels)
        loss.backward()
        # optimize weights according to gradients
        optimizer.step()

        running_loss+=loss.detach()
        running_acc+=acc.detach()
        print(f"epoch {epoch}, batch {i_batch}: loss = {torch.round
        (loss.detach(),decimals=3)},
        acc={torch.round(acc.detach(),decimals=3)}")
        if i_batch % 5 == 4:
            print(f"epoch {epoch}, batch {i_batch}: avg loss per batch =
            {torch.round(running_loss/5.0,
            decimals=3)}, avg acc = {torch.round(running_acc/5.0,
            decimals=3)}")
            running_loss = 0.0
            running_acc=0.0

    if epoch % 1 == 0:
      torch.save({
        'epoch': epoch,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
            'loss': loss,
            }, os.path.join(src_path, f"{date.today()}_model_e{epoch}_b{i_batch}.pth"))

    return model


def test(model, src_path, device):
    src_path = src_path#os.path.join(os.path.dirname(os.path.abspath(__file__)),
    "../../Data/PlayingCards")
    datasets = {}
    datasets["test"] = PlayingCardsSet(src_path, "test")
    print([len(dsi) for dsi in datasets.values()])

    dataloaders = {}
    batch_size = 32
    dataloaders["test"] = torch.utils.data.DataLoader(datasets["test"],
    batch_size=batch_size,
                                         shuffle=False, num_workers=1)
    running_acc = 0.0
    counter = 0
    for i_batch, batch in enumerate(dataloaders["test"]):
        imgs = batch["image"].to(device)
        gt_labels = batch["label"].to(device)
        pred = model(imgs)
        pred_labels = torch.argmax(torch.softmax(pred, -1), -1)
        acc = (pred_labels == gt_labels).double().mean()
        running_acc += acc
        counter += 1
    test_accuracy = running_acc/counter
    print(f"TEST ACCURACY: {test_accuracy}")

if __name__ == "__main__":
    device = "cuda" if torch.cuda.is_available() else "cpu"
    src_path = os.path.join(os.path.dirname(os.path.abspath(__file__))
    ,"../../Data/CustomSet")
    model = Cnn()
    checkpoint = torch.load("model.pth")
    model.load_state_dict(checkpoint['model_state_dict'])
    #train(model, src_path, device)
    #test(model, src_path, device)
```

8.4.15 data_utils.py

```
                import os
import torch
import pandas as pd
#from skimage import io, transform
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
import PIL.Image
import numpy as np

def transform(input:np.ndarray):
    transform = transforms.Compose(
                [transforms.ToTensor(),
                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                transforms.CenterCrop((100,100))])
    image = transform(input)
    return image


class PlayingCardsSet(Dataset):
    def __init__(self, root_dir, ds_type="train", transform=True) -> None:
        super().__init__()

        def get_label_from_filename(fn):
            if len(fn.split(".jpg")[0]) < 6:
                number_map = {str(k):k-2 for k in list(range(2,11))}
                number_map.update({"J":9,"Q":10,"K":11,"A":12})
                color_map = {"C":0,"D":1,"H":2,"S":3}
                number = fn[0] if fn[1] != "0" else "10"
                color = fn[1] if fn[1] != "0" else fn[2]
                label = number_map[number]+color_map[color]*13
            else: # joker
                label = 52
            return label

        def get_label_from_filename_custom_set(fn):
            if len(fn.split("/")[-1]) < 12:
                number_map = {str(k):k-1 for k in list(range(1,14))}
                #number_map.update({"J":9,"Q":10,"K":11,"A":12})
                color_map = {"club":0,"diamond":1,"heart":2,"spade":3}
                card_name = fn.split("/")[-2]
                #print("CARD_NAME: ", card_name)
                if "club" in card_name:
                    color_str = "club"
                elif "diamond" in card_name:
                    color_str = "diamond"
                elif "heart" in card_name:
                    color_str = "heart"
                elif "spade" in card_name:
                    color_str = "spade"
                #number = fn[0] if fn[1] != "0" else "10"
                number = card_name[len(color_str):]
                label = color_map[color_str]#*13+number_map[number]#
            else: # joker
                label = 52
            return label

        self.transform = transform
        self.root_dir = root_dir
        self.img_dir = os.path.join(root_dir, "Images/Images/")
        self.img_dir = os.path.join(root_dir, "")
        self.dirs = sorted(os.listdir(self.img_dir))
        self.dirs = [mydir for mydir in self.dirs if "club" in mydir or
        "diamond" in mydir or "heart"
        in mydir or "spade" in mydir]
        print("self.dirs", self.dirs)
        self.ann_dir = os.path.join(root_dir, "Annotations/Annotations/")
        #self.filenames = sorted(os.listdir(self.img_dir))
        self.filenames = []
        for i, mydir in enumerate(self.dirs):
            to_append = sorted(os.listdir(os.path.join(self.img_dir, mydir)))
            to_append = [os.path.join(self.img_dir, mydir, img_fn)
            for img_fn in to_append
            if "bin" not in img_fn and "9" not in img_fn and "10" not
            in img_fn and "." not in img_fn]
            self.filenames.append(to_append)
        out = []
        for fns in self.filenames:
            out = out + fns
        self.filenames = out
        #print("filenames ",self.filenames)
        #self.labels = [get_label_from_filename(fn) for fn in self.filenames]
        self.labels = [get_label_from_filename_custom_set(fn) for fn in self.filenames]
        print(np.unique(self.labels))
        #print([f'{fn.split("/")[-2:]}\n' for fn in self.filenames])

        # generate train and test
        self.filenames_train = []
        self.len_train = 100
        self.filenames_test = []
        current_label = self.labels[0]
        next_label = self.labels[0]
        i_it = 0
        while i_it < len(self.filenames):
            counter = 0
            current_label = next_label
            while current_label == next_label:
                current_label = next_label
                if counter < self.len_train:
                    self.filenames_train.append(self.filenames[i_it])
                else:
                    self.filenames_test.append(self.filenames[i_it])
                counter += 1
                i_it += 1
                next_label = self.labels[i_it] if i_it < len(self.filenames)
                else "none"
        if ds_type is "train":
            self.filenames = self.filenames_train
        elif ds_type is "test":
            self.filenames = self.filenames_test
        elif ds_type is "all":
            self.filenames = self.filenames
        else:
            raise Exception("unvalid dataset subset type")
        self.labels = [get_label_from_filename_custom_set(fn) for
        fn in self.filenames]

        print(len(self.filenames))
        #index_debug = torch.randint(len(self.filenames), size=(100,)).numpy()
        #print(index_debug)
        #self.filenames = np.asarray(self.filenames)[index_debug]
        #self.labels = np.asarray(self.labels)[index_debug]

        #print(self.filenames)

    def __len__(self):
        return len(self.filenames)

    def __getitem__(self, index):
        if torch.is_tensor(index):
            index = index.tolist()
        file_name = self.filenames[index]
        #print(file_name)
        #image = PIL.Image.open(os.path.join(self.img_dir, file_name)).
        convert("RGB")
        f = None
        with open(file_name, 'r') as f:
            image = np.loadtxt(f) #[h, w, 3]

        image = image.reshape((480,640,3)).astype(np.uint8)
        #print("max, min", np.max(image), np.min(image))

        if self.transform:
            transform = transforms.Compose(
                [transforms.ToTensor(),

                 transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                 transforms.Resize((100,100))])
            image = transform(image)

        if image.size()[0]==1:
            image=image.repeat([3,1,1])

        #print(image.shape, image.max(), image.min())
        sample = {'image': image, 'label': self.labels[index], 'filename': file_name}

        return sample


    def resize_save(self, target_dir):
      if not os.path.exists(target_dir):
        os.mkdir(target_dir)
      transform = transforms.Compose(
                [transforms.ToTensor(),
                 transforms.Resize((512,512)),
                 transforms.ToPILImage()])
      for fn in self.filenames:
        image = PIL.Image.open(os.path.join(self.img_dir, fn)).convert("RGB")
        image = transform(image)
        if not os.path.exists(os.path.join(target_dir, fn)):
          image.save(os.path.join(target_dir, fn))
      print("done saving resized images")

if __name__=="__main__":
    PlayingCardsSet("/Users/lennartschulze/Downloads/Embedded Systems/project/blackjack-counter
    /Data/CustomSet")
```