FPGACam: A FPGA based efficient camera interfacing architecture for real time video processing

Sayantam Sarkar1✉ | Satish S. Bhairannawar2 | Raja K.B.3

1Department of Electronics and Communication Engineering, Vijaya Vittala Institute of Technology, Bangalore, Karnataka, India
2Department of Electronics and Communication Engineering, Shri Dharmasthala Manjunatheshwara College of Engineering and Technology, Dharwad, Karnataka, India
3Department of Electronics and Communication Engineering, University Visvesvaraya College of Engineering, Bangalore, Karnataka, India

Correspondence
Sayantam Sarkar, Department of Electronics and Communication Engineering, Vijaya Vittala Institute of Technology, Bangalore, Karnataka-560077, India.
Email: sayantam.61@gmail.com

Abstract
In most of the real time video processing applications, cameras are used to capture live video with embedded systems/Field Programmable Gate Arrays (FPGAs) to process and convert it into the suitable format supported by display devices. In such cases, the interface between the camera and display device plays a vital role with respect to the quality of the captured and displayed video, respectively. In this paper, we propose an efficient FPGA-based low cost Complementary Metal Oxide Semiconductor (CMOS) camera interfacing architecture for live video streaming and processing applications. The novelty of our work is the design of optimised architectures for Controllers, Converters, and several interfacing blocks to extract and process the video frames in real time efficiently. The flexibility of parallelism has been exploited in the design for Image Capture and Video Graphics Array (VGA) Generator blocks. The Display Data Channel Conversion block required for VGA to High Definition Multimedia Interface Conversion has been modified to suit our objective by using optimised Finite State Machine and Transition Minimised Differential Signalling Encoder through the use of simple logic architectures, respectively. The hardware utilization of the entire architecture is compared with the existing one which shows that the proposed architecture requires nearly 44% less hardware resources than the existing one.

1 | INTRODUCTION

Vision is one of the most prominent senses present in the humans [1] due to which real time vision based system or a part of the system are commonly used in various real-time applications. In general, any real time video/image processing technique can be split into four main processing blocks, namely Sensor, Memory, Processing Unit, and Display [1, 2].

(i). Sensor: It is used to capture video sequences from the external environment and transform it into corresponding electrical signals suitable for further processing.

(ii). Memory: It is internal RAM where the captured video sequences are stored temporarily for further processing.

(iii). Processing Unit: In this section, the required processing algorithm/architecture is implemented. This block accepts data from Memory.

(iv). Display: This block accepts the processed data and converts it into the required format supported by the display device.

In this paper, we propose a new Very Large Scale Integrated Circuit architecture to interface low cost Complementary Metal Oxide Semiconductor (CMOS) camera and display device with processing elements to FPGA board efficiently and also to display the video directly using a display device, such as monitor or TV. The entire architecture is optimised to get lower hardware utilizations without affecting the architectural accuracy which is implemented using Vivado 2018.3 tool where the coding is performed by using the standard Very High-Speed Integrated Circuit Hardware Description Language (VHDL) language [3]. This architecture is synthesized and tested in real time using Digilent NexysVideo (xc7a200t-1bg484c) FPGA board [4] and Zybo Z7-10 (xc7z010-1clg400c) FPGA Board [5] separately where NexysVideo is of a medium level FPGA and Zybo Z7-10 is of a low level FPGA. The level of FPGA is generally defined.
by both the cost and hardware complexity, such as logic densities, internal memory, and DSP blocks etc.

1.1 Contributions

The novel concepts of this paper are listed as follows:

(i). The Camera Controller block is optimised by the Modified Serial Camera Control Bus (SCCB) Conversion block at the architectural level.

(ii). The Image Capture and Video Graphics Array (VGA) Generator blocks are optimised using parallel architecture.

(iii). The design complexity of different Colour Plane Conversion blocks are minimized at the architectural level using adders and shifters.

(iv). The VGA to High Definition Multimedia Interface (HDMI) Conversion block is optimised using modified Display Data Channel (DDC) Conversion and TMDS Encoder blocks, respectively, where the optimised Finite State Machine (FSM) is used to modify the DDC Conversion and the Transition Minimized Differential Signalling (TMDS) Encoder is optimised using Comparator and Addition blocks.

2 RELATED WORKS

Normally, embedded systems with hardware/software co-simulation techniques are widely used to implement video processing systems due to the ease of implement. Said et al. [6] proposed a video interface technique where the Xilinx EDK tool is used to interface a Micro-Blaze embedded processor and the architecture is implemented using the embedded C-language onto the processor. This system uses a Micron MT9V022 VGA camera to capture videos at the rate of 60 frames per second (fps) which is then displayed using standard DVI interfaces. This architecture requires a large amount of hardware resources and the overall operating speed of the architecture is low. A similar technique for capturing a video is presented by Abdaoui et al. [7] which is implemented on a Virtex-5 FPGA with a co-processor to control the overall operation. This approach increases the area requirements as well as decreases overall frequencies. Biren and Berry [8] presented a new camera interfacing architecture which is implemented on an Altera Cyclone-III FPGA using different IP cores provided by an FPGA manufacturer. The use of IP cores increases the total hardware utilization of the architecture. Along with camera interfaces, many processing algorithms are implemented for real time video processing. Stereo vision based video rectification is presented by Maldeniya et al. [9], where a dual camera is used to capture stereo images and are interfaced with a Spartan-3E FPGA through the 100Base-T protocol using the embedded processor present in Xilinx EDK tool. Similarly real time motion tracking is presented by Mosqueron et al. [10] where motion is detected from the real time video captured by a camera through FPGA using embedded processing. The disadvantage of this architecture is the less overall frame rate. It is necessary to understand sensor design to interface a camera properly with any of the processing elements. Zhao et al. [11] present a 64 x 64 array image sensor architecture which is designed using the UMC 0.18 μm technology which has different user defined operating modes depending upon applications. The circuit used to read the sensor captures the rows present in the array sequentially and generates an analog voltage which is then digitized by an on-chip Analog to Digital Converter. This architecture is able to produce images of 64 x 64 resolution at 100 fps.

3 PROPOSED ARCHITECTURE

The Digilent NexysVideo [4] and Zybo Z7-10 [5] FPGA Board is used separately to interface the OmniVision OV7670 [12] and OV9655 [13] cameras which further use the two wire SCCB interface [14] for initializing their internal resistors to generate specific user defined video formats. The proposed architecture used to interface camera with FPGA is shown in Figure 1. In the proposed architecture, different blocks need different clock frequencies (i.e. clk1, clk2, and clk3, respectively) to generate proper output video sequences for capture and display. So, using the Phase Locked Loop/MMCE IP core [15] present in a Xilinx Vivado [16] tool as a clock generator, the architecture generates different clock frequencies accurately for proper operations.

Initially the camera is configured by the Optimised Controller block to generate a video sequence of 640 × 480 resolution at 30 fps using the two wired SCCB interfacing [14] protocol. The generated video is of the RGB565 [17] format, and for the proper synchronization between the camera and designed blocks, FPGA should start accepting pixel values from the camera serially through the PMOD Ports [18] after the completion of the camera configuration through register setting. The Optimised Image Capture block converts the camera output into the proper serial format using corresponding synchronization signals (VSYNC and HSYNC) of the camera which is then converted to the RGB444 [17] format by Optimised RGB565 to RGB444 Conversion block. This converted pixel values are then temporarily stored into Memory [19] block which helps in synchronizing pixel the values between different blocks operating at different frequencies [20]. After writing one row into the Memory, the Optimised VGA Signal Generator block reads the stored pixel data through Optimised RGB444 to RGB565 Conversion block and then converts this data into the corresponding VGA signal with proper synchronization signals (i.e. vsync and hsync), respectively. Now, depending upon the Mode switch value, color or gray pixel format is selected by both the Optimised Color to Gray Conversion and MUX blocks which are then used to generate the corresponding HDMI format by the Optimised VGA to HDMI Conversion block. The VGA and HDMI signals are connected to the input of the Port Selection block which selects one particular display formatted signal from the input depending upon the port select value which is connected to the display device such as monitor/TV for the purpose of display.
In this architecture, both VGA and HDMI signals are used, as FPGA boards support the VGA port for displaying while others support the HDMI port. As a result with some small user defined modification, our proposed architecture can be implemented on any kind of FPGA boards.

3.1 | Optimised Controller

The Optimised Controller unit is used to program the internal registers of the camera depending upon the corresponding datasheets [12, 13] to set the proper operating mode. For this implementation, an image size of 640 x 480 is considered at 30 fps whose pixel values are represented by the RGB565 format [17]. The architecture of the Optimised Controller is shown in Figure 2 which consists of the Register Sets and Modified SCCB Format Generation blocks.

3.1.1 | Register sets

The specifications of the generated video streams of both cameras (OV7670 and OV9655) such as video size, color specifications, and frame rates, can be set by the designer depending upon the application requirements with the help of the corresponding datasheets [12, 13]. To generate a good quality video, it is essential to assign the proper values at corresponding address in proper sequences, and any mistakes in assigning this register address and values will certainly affect the quality of the generated video streams. So, correct register address and values are stored into the Register Sets block which is then used by the architecture to specify the correct video format.

3.1.2 | Modified SCCB format generation

The OmniVision cameras uses the SCCB interfacing [14] protocol to access its internal register to define video specifications generated by the camera. The abbreviation of the SCCB is Serial Camera Control Bus is the simplified version of Philips Inter-Integrated Circuit (I2C) [21] protocol.

The Modified SCCB Format Generation block fetches the address and data from the Register Sets block and convert these into the corresponding SCCB format values, which is
normally using a serialized data transfer technique. The *pseudo-code* is used to implement this block is given in Algorithm 1. From Algorithm 1, it can be seen that the whole block is designed using *Counters, Comparators, and basic logical components*. This algorithm is designed with the aim to optimize hardware parameters without affecting functionality. Extra signals (*send and taken*) are considered for proper synchronization between two corresponding blocks. Similarly, the signal *id* is used to define the operating mode of the camera registers (i.e., read or write) by considering a specific value depending upon corresponding camera datasheet [12, 13].

### Algorithm 1 Modified SCCB Format Conversion

| Inputs: data_in, send, id; |
| Variables: divider, busy; |
| Outputs: SIOC, SIOD, addr, taken; |

```plaintext
if [{(busy(11 : 0) = 2)\&(busy(20 : 19) = 2)}\&(busy(29 : 28) = 2)] then
  SIOC = 0;
else
  SIOC = data_in(31);
endif;
if {(clk) rising edge} then
  taken = 0;
if {busy(31) = 0} then
  SIOC = 0;
  if {send = 1} then
    busy = (2^{33} - 1); taken = 1;
    addr = (addr + 1);
    data_temp = {4, id, 132, data_in, 1};
  else
    if {divider = 256} then
      busy = {busy(30 : 0), 0}; divider = 0;
      data_temp = {data_temp(30 : 0), 1};
    else
      divider = (divider + 1);
  endif;
else
  SIOC = 1;
endif;
if [{(busy(31 : 29) = 7)\&(busy(2 : 1) = 3)] then
  SIOD = 1;
elseif [{(busy(30 : 29) = 0)\&(busy(2 : 0) = 0)} then
  SIOD = 1;
elseif [{(busy(31 : 29) = 6)\&(busy(2 : 0) = 0)} then
  if {divider(7 : 6) = 0} then
    SIOD = 0;
  else
    SIOD = 1;
  endif;
else
  SIOD = {divider(7) \& divider(9)};
endif;
```

Most of the existing two wire *I²C* or SCCB interface architecture use the FSM model [22] or interdependent counters [23] to perform serialization which increases the design complexity and hardware requirements.

### 3.2 Optimised image capture

For proper synchronization between different pixels present in video frames, the pixel data must be captured depending on various synchronous signals that is, *PCLK, HREF*, and *VSYNC*, respectively, from the camera. The main task of this block is to store pixel data into *Memory* using these synchronous signals through proper address (wr_addr) along with pixel values (data_out) and write enable (wr) signal. The block diagram of the Optimised Image Capture block is shown in Figure 3.

The proposed architecture captures the pixel values form the camera and stores it into the *Memory* block depending on *VSYNC* and *HREF* signals which are checked by *Comparator_1* and *Comparator_2*, respectively. Both blocks generate the intermediate reset signal, *rst_temp* and enable signal, *en*, respectively which are then used to reset the entire architecture and to stay at previous values. The *rst_temp* and *en* signals are made high by *Comparator_1* and *Comparator_2* only when *HREF = 1* and *VSYNC = 0*. At this situation, one pixel value is sent by the camera periodically at each non-overlapping second *PCLK* pulse. To store the pixel values, the feedback architecture is made by interconnecting the *AND Gate, Merger_1, DFF_1*, and *Concatenation* blocks which are able to track second *PCLK* at valid conditions, and by making *wr = 1*, they force the *Memory* block into write mode. Simultaneously *DFF_2, Merger_2* and *DFF_3* are used to merge incoming data from the camera at each rising edge *PCLK* to generate video pixel values in RGB565 format [17]. Also the blocks namely *Counter* and *DFF_4* are used to generate the accurate write address values *wr_addr* to store the pixel values in correct predefined locations. This architecture uses basic gates to reduce hardware complexity and logic utilizations [3].

### 3.3 Colour plane conversions

The *OV7670* [12] and *OV9655* [13] cameras support *RGB565, RGB555, RGB444, Raw Bayer and Processed Bayer* colour formats, but the *RGB565 format* [17] is considered to generate video frames from these cameras [12, 13]. This is because it is nearer to the format of *RGB888* color-space which generally used by most of the modern display devices. As a result, at the time of the conversion (i.e. *RGB565* to *RGB888* format) by the display devices, generated errors will be very less. To store the frame pixels temporarily in the *Memory* unit using this format requires a high amount of memory which is not present in most of the low and medium range FPGAs. So, the *RGB565 format* is converted into *RGB444 format* to reduce the memory requirements by almost 25% of the original requirements. But the *RGB444*
format is not suitable for producing good quality color pixels and not suitable for many applications. So, it is necessary to convert the stored RGB444 format into the corresponding RGB565 format.

3.3.1 Optimised RGB565 to RGB444 conversion

The conversion equation [1] is given in Equation (1) as follows:

\[
\begin{bmatrix}
X_R \\
X_G \\
X_B
\end{bmatrix} = \begin{bmatrix}
\frac{15}{31} \times x_R \\
\frac{15}{63} \times x_G \\
\frac{15}{31} \times x_B
\end{bmatrix} \quad (1)
\]

where, \(x_R \rightarrow\) Red Pixel values in RGB565 format, \(x_G \rightarrow\) Green Pixel values in RGB565 format, \(x_B \rightarrow\) Blue Pixel values in RGB565 format. \(X_R \rightarrow\) Red Pixel values in RGB444 format, \(X_G \rightarrow\) Green Pixel values in RGB444 format, \(X_B \rightarrow\) Blue Pixel values in RGB444 format.

From Equation (1), it can be seen that multipliers and dividers are required for implementation which increases hardware requirements [24]. To overcome this problem, multipliers and dividers are replaced by the corresponding shifter and adder blocks [20] which are given in Equation (2) as follows:

\[
\begin{bmatrix}
X_R \\
X_G \\
X_B
\end{bmatrix} \approx \begin{bmatrix}
(LS_2 + LS_3 + LS_4 + LS_5 + LS_6) \times x_R \\
(LS_2) \times x_G \\
(LS_2 + LS_3 + LS_4 + LS_5 + LS_6) \times x_B
\end{bmatrix} \quad (2)
\]

where, \(LS_n \rightarrow\) Left Shift by Position \(n\).

The architecture used to implement this conversion is shown in Figure 4 where \(Q\) format [25] is used to preserve the data accuracy by considering a fractional part for the intermediate stage, but the input and output signals are in normal binary format, respectively. The Concatenation blocks present in the input side separate the different color components (red, green and blue) from the pixel value and then the required number of 0s are padded to the MSB side to make all three color planes 16-bit signals. Now using shifters and adders, the intermediate values are generated which are then concatenated to generate the individual color planes in the RGB444 format. These planes are then merged by the Merger block to generate the corresponding RGB444 formatted pixel value.
3.3.2 Optimised RGB444 to RGB565 conversion

The equation used to convert the stored RGB444 format into corresponding RGB565 format [1] is given in Equation (3).

\[
\begin{bmatrix}
X_R \\
X_G \\
X_B 
\end{bmatrix} = \begin{bmatrix}
31 \\
15 \\
63 \\
15 \\
31 \\
15 
\end{bmatrix} \times \begin{bmatrix}
x_R \\
x_G \\
x_B 
\end{bmatrix}
\]

(3)

Hardware utilization which is mainly due to multipliers, is reduced by replacing it with shifters and adders [20]. As a result, the Equation (3) is modified into Equation (4):

\[
\begin{bmatrix}
X_R \\
X_G \\
X_B 
\end{bmatrix} \approx \begin{bmatrix}
(RS_1 + LS_1 + LS_6) \times x_R \\
(RS_2 + LS_1 + LS_5 + LS_6) \times x_G \\
(RS_1 + LS_5 + LS_6) \times x_B 
\end{bmatrix}
\]

(4)

where, \(RS_n \rightarrow \text{Right Shift by Position } n\).

The hardware architecture of the optimised RGB444 to RGB565 Conversion block is designed from Equation (4) similar to the way the optimised RGB565 to RGB444 Conversion block is designed from the Equation (2).

3.4 Optimised VGA Signal Generator

The converted Pixel values form the Optimised RGB444 to RGB565 Conversion block is used by the Optimised VGA Signal Generator block to generate the corresponding VGA signal. The process of reading the pixel values from the Memory block starts after storing one frame of the video sequence. This avoids the synchronization problem between Memory and the remaining processing blocks.

The CMOS camera [12, 13] is programmed to generate videos of 640 × 480 resolution. So, the locally generated VGA signals must have the same resolution for regenerating the same video. Standard parameters [26] are used to generate VGA signals of 640 × 480 resolution. The VGA signal with 25.175 MHz frequency is used to generate a local VGA signal to maintain data comparability with camera modules. The proposed architecture is used to generate the local VGA signal given in Figure 5 where the constants values [27] are calculated using Equation (5) to Equation (12).

\[hRez = \text{Horizontal Visible Area.}\]

(5)

\[vRez = \text{Vertical Visible Area.}\]

(6)

\[hMaxCount = \text{Horizontal Line.}\]

(7)

\[bStartSync = hRez + \text{Horizontal Front Porch.}\]

(8)

\[bEndSync = hStartSync + \text{Horizontal Synchronization.}\]

(9)

\[vStartSync = vRez + \text{Vertical Front Porch.}\]

(10)

\[vEndSync = vStartSync + \text{Vertical Synchronization.}\]

(11)

\[vMaxCount = vEndSync + \text{Vertical Back Porch.}\]

(12)

The main task of this block is to generate an address for Memory to read the pixel values and convert those into the corresponding VGA format with required synchronization signals as VGA_blank, VGA_hsync, and VGA_vsync. Comparator_1, Counter_1, Comparator_2, and Counter_2 are used to generate horizontal and vertical timing signals, namely \(bCounter\) and \(vCounter\), respectively, to generate the proper VGA format. Comparator_1 is used to compare the output of Counter_1 with \(hMaxCount \cdot 1\). If \((hMaxCount \cdot 1) > bCounter\), then Comparator_1 activates Counter_1 through the rst signal. Comparator_2 is used to detect the time when \(bCounter = 0\) and activates Counter_2 at that time. Also, Comparator_3 checks whether the output of Counter_2 is \((vMaxCount \cdot 1) > vCounter\) or not, and depending on this condition at each \(bCounter = 0\), the value of \(vCounter\) is incremented by 1. These two intermediate values (\(bCounter\) and \(vCounter\)) are then compared with \(vRez\) and \(hRez\) values through Comparator_4, Comparator_5, NOT Gate, and DFF_1 to generate the VGA_blank signal. Also the output of Comparator_5 is used by Counter_3 to generate the read address for the Memory unit and DFF_2 to send the read data at the rising edge of clock signal. Comparator_6 and Comparator_7 generate an intermediate signal depending on bStartSync and bEndSync values, respectively, which is then ANDed using the AND Gate and passes as the VGA_hsync signal through DFF_3. Similarly, the signal generated by Comparator_8 and Comparator_9 is ANDed and passed through DFF_4 to generate the VGA_vsync signal.

This architecture requires less hardware resources than the existing [28] due to the use of basic logic elements in an optimised way.

3.5 Optimised Color to Gray Conversion

In many video processing applications, color images are converted into grey scale to reduce the complexity of processing. The equation for this conversion is derived using the averaging method [1] and is given in Equation (13) as follows:

\[I_{\text{Gray}} = \left\{ \frac{1}{3} \times (R + G + B) \right\}\]

(13)
where, $I_{\text{Gray}} \rightarrow$ Intensity value of Grey Pixel, $R \rightarrow$ Red Pixel values of the image, $G \rightarrow$ Green Pixel values of the image, $B \rightarrow$ Blue Pixel values of the image.

Equation (13) is modified into Equation (14) to generate efficient hardware architectures [20].

$$I_{\text{Gray}} \approx \{(L_S + L_S + L_S) \times (R + G + B)\}. \quad (14)$$

The hardware architecture of the Optimised Color to Gray Conversion block is designed from Equation (14) similar to the way RGB format Conversion blocks are designed from their respective modified equations.

### 3.6 Optimised VGA to HDMI conversion

Most commonly VGA, HDMI, DVI etc., interfacing standards are used by display devices, such as TV, monitor, etc. Among these standards, HDMI standard is the most commonly used in newly manufactured display devices due to its support of uncompressed high quality audio and video interfaces through a single cable [29].

The generalized architecture of the VGA to HDMI Conversion block [30] is used to build the Optimised VGA to HDMI Conversion block by optimizing some of its internal block at the architectural level where the HDMI 1.4 standard [31] is considered suitable for transmitting videos of $640 \times 480$ resolution over channel (wire). The proposed Optimised VGA to HDMI Conversion block consists of Modified TMDS Encoder, Modified DDC Format Generations, Serialiser, and Extended Display Identification Data (EDID) ROM units. The EDID [32] is used to store the supporting display related information defined by the HDMI 1.4 standard [31], which is then converted into the DDC [33] format by the Modified DDC Format Generations block. Further, the pixel values are encoded through the Modified TMDS Encoder to encode with the help of HSYNC and VSYNC signals and then converted into a serial format by the Serialiser block [34].

#### 3.6.1 EDID ROM

In any HDMI protocol, the operational characteristics of the video, such as resolution, frame rate and version etc., must be exchanged between source and sink at the beginning for proper synchronization. These values are normally constant for a specific resolution [32]. The EDID values corresponding to $640 \times 480$ resolution are stored into the EDID ROM block in the proper order for further use. The standard file structure for EDID is considered for this implementation. The EDID versions of 1.3 and above uses a total 256 bytes to define the corresponding EDID structure. In such cases the Extension Flag is defined by a total of 128 bytes. For our implementation, CEA-861 Standards [33] are used to define the Extension Flag field.

#### 3.6.2 Modified DDC Format Conversion

The HDMI standard exchanges EDID ROM data between its Source and Sink using the DDC Protocol [33] which is normally
a standard serial signaling protocol. This is almost similar to the Philips I²C [21] protocol which consists of three wires, such as Serial Data (SDA), Serial Clock (SCL), and high logic value (+5 V). The DDC format [33] uses Inter-Integrated I²C [21] specifications to transfer the TMDS encoded data. To design the proposed DDC Protocol, NXP UM10204 I²C bus specifications [35] for single master buses are considered. Those specifications mainly give the details of Start, Stop, and Acknowledgement signals for proper communication. To implement this, the FSM model is used, which is given in Figure 6. The proposed state machine uses 8-bit data and addressing bits. Upon start-up, the state machine immediately enters into the Ready state and stays in the same state until Send = 1 and Restart = 0 which makes the machine go to the Start state which generates proper start conditions for data transfer. Now the machine goes to the Address state which fetches 8-bit address from the EDID ROM and serialize it using the PISO architecture [36] which is then tracked by the bit_counter variable. The machine stays in this state until bit_counter = 0. When bit_counter = 0, the machine reaches the Slave ACK1 state where it waits for the acknowledgment from the slave device. If acknowledgement is received, then it performs similar operation using data from EDID ROM or else starts sending address values once again. Once Slave ACK2 is received by the sender, the address value is incremented by ‘1’ and goes back to the Start state again. This way when Slave ACK2 is received for the address_max value then the machine enters the Stop state and stays in that state until the process is restarted when Restart = 1 which forces the machine to go to the Ready state and perform the entire operation once again.

3.6.3 | Modified TMDS encoder

Transition Minimized Differential Signaling is abbreviated as TMDS and is used to encode data at a very high speed for various video interfaces. It uses a form of 8b/10b encoding [37] technique which reduces electromagnetic interferences to
provide faster signal transmission with reduced noise [38]. The TMDS encoder encode the input data and send it serially at a high speed which minimizes transitions by retaining high frequency transitions for clock recovery. This process keeps the minimum number of 1s and 0s in the line nearly equal to improve the noise margin. The algorithm is used to implement Modified TMDS Encoding is given in Algorithm-2. The Modified TMDS encoder unit is designed using basic gates and flip-flops with simpler interconnections between them to produce optimized hardware architecture than the existing [39].

Algorithm 2 Modified TMDS Encoder

```plaintext
Inputs: data, clk, c, blank;
Variables: ones, word, word_inv, disparity, bias, xored, xnored;
Output: encoded;

xored(0) = xnored(0) = data(0); xored(8) = 1; xnored(8) = 0; disparity = (12 + ones);
ones = Total number of 1's present in input data;
for (i = 1; i ≤ 7; i++)
  xored(i) = {data(i) ⊕ xored(i - 1)};
xnored(i) = {data(i) ⊗ xnored(i - 1)};
end for;
if {(ones > 4) | (ones = 4 & data(0) = 0)}
  word = xnored; word_inv = not(xnored);
else
  word = xored; word_inv = not(xored);
end if;
if {(clk)_rising_edge} then
  if (blank = 0) then
    if (c = 0) then
      encoded = 852;
    elseif (c = 1) then
      encoded = 171;
    elseif (c = 2) then
      encoded = 340;
    else
      encoded = 683;
    end if;
  else
    if {(bias = 0) | (disparity = 0)} then
      if {word(8) = 0} then
        encoded = {1, word(7 : 0)};
bias = {bias + disparity};
      else
        encoded = {2, word(7 : 0)};
bias = {bias - disparity};
      end if;
    elseif {[bias(3) ∨ disparity(3)] = 1}
      encoded = {1, word(8), word_inv(7 :
```

4 | FPGA IMPLEMENTATION

The proposed architecture is coded using the standard VHDL language [3], synthesized and implemented on Digilent NexysVideo (xc7a200t-1bg484c) [4] and Zynq Z7-10 (xc7z010-1clg400c) [5] FPGA board separately through the bit-file generated by the Xilinx Vivado 2018.3 tool by assigning the ports in a .xdc file. The generated schematic diagram of the proposed architecture after post-implementation step is shown in Figure 7 for the NexysVideo FPGA board.

The hardware utilizations of the proposed interfacing architecture after post-implementation stage including most of the internal blocks are given in Table 1. The hardware utilizations and power requirements of the entire camera interfacing architecture is lesser than the sum of all the individual components present internally in the architecture for both the cases, respectively, which is mainly due to the use of the Balanced Synthesis and Optimizations [16] operation present in the Xilinx Vivado tool.

5 | REAL TIME IMPLEMENTATION

The image of the experimental product setup using the proposed camera interfacing architecture is shown in the Figure 8 where the OV7670 camera is mounted on a stand for better focusing and is connected to a Digilent NexysVideo FPGA board through PMOD ports using general purpose jumper wires. The architecture programs the camera to the correct mode and accepts video sequences which are processed and sent to the available HDMI/VGA port to display through the connected display device.

It is also possible to convert this architecture into a real time product which requires replacing the general purpose jumper wires used in Figure 8 with a custom designed Printed Circuit Board (PCB). The architecture of the custom designed PCB is shown in Figure 9 which is a simple two layered PCB used to provide proper connection between the camera and FPGA module.

6 | PERFORMANCE EVALUATION

The performance of the proposed architecture is compared with various existing architectures or techniques with respect to data accuracy, board comparability, cost, and hardware utilizations to check the superiority of the proposed architecture.
FIGURE 6  Proposed finite state machine model for modified Display Data Channel format conversion
(a) RTL Schematic

![RTL Schematic Image]

(b) FPGA Schematic

![FPGA Schematic Image]

**FIGURE 7** Generated schematics of the proposed architecture on NexysVideo FPGA board

**TABLE 1** Hardware utilizations of proposed camera interfacing architecture

| Parameters       | SCCB interface | Camera controller | Image capture | RGB565 to RGB444 | VGA generator | RGB444 to RGB565 | RGB to grey | DDC interface | TMDS encoder | Total module |
|------------------|----------------|-------------------|---------------|------------------|---------------|------------------|-------------|---------------|--------------|--------------|
| FPGA             | Artix-7 (xc7a200t-1sbg484c) |                  |               |                  |               |                  |             |               |              |              |
| Slice LUTs       | 50             | 81                | 53            | 5                | 39            | 6                | 44          | 7             | 41           | 419          |
| Slice registers  | 73             | 76                | 60            | 32               | 55            | 39               | 26          | 25            | 14           | 376          |
| Flip-flops       | 73             | 76                | 60            | 32               | 55            | 39               | 26          | 25            | 14           | 376          |
| Slices           | 22             | 30                | 38            | 28               | 21            | 30               | 16          | 10            | 11           | 221          |
| LUTs as logic    | 50             | 81                | 53            | 5                | 39            | 6                | 44          | 7             | 41           | 419          |
| BRAMs            | 0              | 0                 | 0             | 0                | 0             | 0                | 0           | 0             | 0            | 60           |
| Memory (KB)      | —              | —                 | —             | —                | —             | —                | —           | —             | —            | —            |
| Memory (KB)      | —              | —                 | —             | —                | —             | —                | —           | —             | —            | —            |
| Negative slack\(^a\) (ns) | —              | —                 | —             | —                | —             | —                | —           | —             | —            | 3.556        |
| Hold slack\(^a\) (ns) | —              | —                 | —             | —                | —             | —                | —           | —             | —            | 0.129        |
| Pulse width slack\(^a\) (ns) | —              | —                 | —             | —                | —             | —                | —           | —             | —            | 3.000        |
| Latency (ms)     | —              | —                 | —             | —                | —             | —                | —           | —             | —            | 0.0256       |
| Total power (W)  | —              | —                 | —             | —                | —             | —                | —           | —             | —            | 0.405        |
| FPGA             | Zynq-7 (x7z010-1clg400c) |                  |               |                  |               |                  |             |               |              |              |
| Slice LUTs       | 54             | 87                | 60            | 8                | 48            | 10               | 49          | 11            | 46           | 434          |
| Slice registers  | 79             | 83                | 66            | 36               | 61            | 43               | 31          | 27            | 22           | 484          |
| Flip-flops       | 79             | 83                | 66            | 36               | 61            | 43               | 31          | 27            | 22           | 484          |
| Slices           | 25             | 37                | 49            | 30               | 28            | 39               | 20          | 16            | 19           | 283          |
| LUTs as logic    | 54             | 87                | 60            | 8                | 48            | 10               | 49          | 10            | 46           | 434          |
| BRAMs            | 0              | 0                 | 0             | 0                | 0             | 0                | 0           | 0             | 0            | 60           |
| Memory (KB)      | —              | —                 | —             | —                | —             | —                | —           | —             | —            | —            |
| Negative slack\(^a\) (ns) | —              | —                 | —             | —                | —             | —                | —           | —             | —            | 3.516        |
| Hold slack\(^a\) (ns) | —              | —                 | —             | —                | —             | —                | —           | —             | —            | 0.117        |
| Pulse width slack\(^a\) (ns) | —              | —                 | —             | —                | —             | —                | —           | —             | —            | 2.293        |
| Latency (ms)     | —              | —                 | —             | —                | —             | —                | —           | —             | —            | 0.0256       |
| Total power (W)  | —              | —                 | —             | —                | —             | —                | —           | —             | —            | 0.361        |

Abbreviations: DDC, Display Data Channel; SCCB, Serial Camera Control Bus; TMDS, Transition Minimized Differential Signalling; VGA, Video Graphics Array.

\(^a\)For worst case scenario.
The $|\text{Error}|$ is calculated for both of the conversion processes (i.e., RGB565 to RGB444 and RGB444 to RGB565 respectively) with the corresponding full range of RGB values of different color planes where only integer values are considered. The $|\text{Error}|$ occurred in both of the conversion processes is always lesser than ‘1’ due to the use of $Q\text{format}$ for intermediate calculations. As a result, when both conversion blocks are cascaded to get double conversion, due to design issues the $|\text{Error}|$ introduced at the final result must also be equal to or less than ‘1’. These small errors are introduced due to $truncation\ errors$ $[3, 24]$ that occurred in binary arithmetic calculations.

### 6.2 Board compatibility and approximated costs

The FPGA manufactured by $\text{Xilinx}$ and $\text{Altera}$ are most commonly used by the PCB manufacturing companies to make FPGA boards $[41]$. As a result, these board manufacturers supply many extra $addon\ components$ as accessories to some specific FPGA boards to perform selected operations, such as video and audio processing. Along with these accessories, interfacing program is also supplied by the manufacturer in terms of IP Cores. Due to various issues such as $compatibility$, $availability$ and $easier$ $implementation$ etc., in most of the cases those IP Cores are developed by embedded programing with the help of supporting embedded platforms which are supported by high level FPGA boards only. On the other hand, the proposed architecture is designed through optimised architectures coded using the standard $\text{VHDL\ language}$ $[3]$ with $minimal$ $numbers$ $of$ $IP$ $Cores$ $[15, 19, 34]$ and connected to the FPGA board through two $PMOD$ $ports$ $[18]$ which helps the architecture to be implemented on most of the available FPGA boards.

The board support comparisons of the proposed and existing camera interface architectures are given in Table 2 along with corresponding costs, where the cost of the required peripherals (i.e., stand and connecting PCB respectively) are also considered along with the camera cost. From the Table 2, it can be seen that the camera interfaces $[42–45]$ developed by the corresponding manufactures support only a few number of their boards, whereas the proposed architecture supports any FPGA board with minimum peripherals for interfacing. To prove this, the proposed architecture is implemented on two different level FPGA boards. Moreover, the cost of the entire architecture is very less and mainly depends on the CMOS camera cost. This is due to the absence of extra processing interfaces for the camera module $[42–45]$.

### 6.3 Hardware resources

The utilised hardware resources of most of the subblocks and total module is compared with the corresponding existing technique's hardware resources utilizations to prove that the proposed architecture is better in terms of hardware resource
utilizations as well. To get valid hardware resource comparison, it is essential to use the utilization values of a particular block for the same or similar kind of FPGA board.

### 6.3.1 Camera controller

The comparison of hardware utilizations of the proposed camera controller with the existing controller presented by Xiaokun et al. [46] is compared and given in Table 3. From the table, it can be seen that the proposed camera controller architecture requires less hardware resources than the existing for the same FPGA board (Artix 7). This is because, the proposed camera controller architecture is optimised with the specifications of the OmniVision camera series.

| Parameters | Xiaokun et al. [46] | Proposed architecture |
|------------|----------------------|-----------------------|
| FPGA       | Artix-7              | Artix-7               |
| Slice LUTs | 87                   | 81                    |
| Slice registers | 90 | 76                    |

### 6.3.2 VGA signal generator

The comparison of hardware utilizations of the proposed VGA Signal Generator with the existing VGA Signal Generator is compared and given in Table 4. The VGA Generator architecture presented by Xiaokun et al. [46] is implemented on Artex-7 FPGA. The unoptimised way of using logical components to generate the VGA signal leads to large hardware utilizations than that of the proposed. Similarly, Arun Babu [47] presented a Graphic Controller which was implemented on Virtex 5 FPGA. The use of generalized IP Cores inside the architecture in the unoptimised way requires large hardware resources than that of the proposed. On the other hand, the proposed VGA Generator architecture is optimised with respect to the camera architectural design using basic gates in a very optimised way without using any IP Cores.

### 6.3.3 TMDS encoder

The hardware utilizations of the presented TMDS Encoder is compared with the existing TMDS encoder presented by Roshan and Patil [48] which is shown in Table 5. The TMDS encoder proposed by Roshan and Patil [48] was implemented on Spartan 6 FPGA. The proposed architecture requires very less hardware for the same (Spartan 6) FPGA. This is mainly due to the use of TMDS algorithm which is optimised by using basic gates, comparators, adders, and subtractors.

| Parameters | Roshan and Patil [48] | Proposed architecture |
|------------|------------------------|-----------------------|
| FPGA       | Spartan-6              | Spartan-6             |
| Slice LUTs | 148                    | 52                    |
| Slice registers | 87  | 35                    |
| Flipflops  | 82                     | 35                    |

### Table 2 Comparisons between existing and proposed module FPGA board supports

| Techniques | Supported FPGA boards | Cost ($) |
|------------|-----------------------|----------|
| TDNext [42] | ZedBoard              | 70       |
| Python 1300C [43] | Zedboard and MicroZed | 500      |
| Pcam 5C [44] | Zyo and ZedBoard      | 45       |
| D8M [45] | DE2-115, DE1-SoC and C5G | 80 |
| Proposed | Any FPGA boarda       | 20b      |
|           |                       | 30c      |

*aWith required peripherals.
*bFor OV7670 Camera.
*cFor OV9655 Camera.

### Table 3 Comparisons between existing and proposed camera controller

| Parameters | Xiaokun et al. [46] | Proposed architecture |
|------------|----------------------|-----------------------|
| FPGA       | Artix-7              | Artix-7               |
| Slice LUTs | 87                   | 81                    |
| Slice registers | 90 | 76                    |

### Table 4 Comparisons between existing and proposed VGA signal generator

| Parameters | Xiaokun et al. [46] | Arun Babu [47] | Proposed architecture |
|------------|----------------------|----------------|-----------------------|
| FPGA       | —                    | Virtex-5       | Virtex-5              |
| Slice LUTs | —                    | 241            | 76                    |
| Slice registers | —  | 109             | 97                    |
| Flipflops  | —                    | 109            | 97                    |
| FPGA       | Artix-7              | —              | Artix-7               |
| Slice LUTs | 134                  | —              | 39                    |
| Slice registers | 100 | —              | 55                    |

Abbreviation: VGA, Video Graphics Array.
TABLE 6 Comparisons between existing and proposed camera interface architecture

| Parameters         | Xiaokun et al. [46] | Bhowmik et al. [49] | Zhou et al. [50] | Xilinx IP Core [51] | Honegger et al. [52] | Proposed architecture |
|--------------------|----------------------|----------------------|------------------|----------------------|----------------------|----------------------|
| FPGA               | Artix-7             | —                    | —                | —                    | —                    | Artix-7              |
| Slice LUTs         | 753                 | —                    | —                | —                    | —                    | 14,563               |
| BRAMs              | 106                 | —                    | —                | —                    | —                    | 419                  |
| Memory (KB)        | —                   | —                    | —                | —                    | —                    | 60                   |
| Latency (ms)       | —                   | —                    | —                | —                    | —                    | 3472                 |
| FPGA               | —                   | Zynq-7               | Zynq-7           | Zynq-7               | —                    | Zynq-7               |
| Slice LUTs         | —                   | 7357                 | 39,789           | 30,614               | —                    | 20.0256              |
| Flipflops          | —                   | 8457                 | 43,871           | —                    | —                    | 434                  |

6.3.4 | Total module

The hardware resources comparison of the proposed camera interfacing architecture with similar existing architecture is given in Table 6. An effective CMOS camera interface architecture is presented by Xiaokun et al. [46] which is implemented on the Artix 7 FPGA board and verilog HDL is used for coding. The main reason behind the large hardware utilization than that of the proposed is the use of subblocks in the overall architecture without any optimization. A CPU and FPGA base camera interfacing architecture is presented by Bhowmik et al. [49] which is implemented on the Xilinx Zynq 7000 SoC FPGA board with Vivado tool. The main reason for large hardware requirements by this architecture than existing techniques is the use of generalized IP Cores to implement total architecture without proper optimizations. A smart camera architecture is presented by Zhou et al. [50] which is implemented on the Zynq 7020 FPGA board using the Sum of Absolute Differences (SAD) based Mosaic Algorithm. The use of SAD based Mosaic Algorithm in an unoptimised way increases the hardware requirements. The Xilinx provides IP Cores [51] to interface a specific camera with the Zynq 7000 SoC FPGA board using the ARM CortexA9 present in that board through embedded programing techniques. This increases hardware requirements drastically. A new video processing system is presented by Honegger et al. [52] which uses the FPGA for image acquisition and a mobile CPU for processing. This architecture uses an MT9V034 CMOS camera and an Artix 7 FPGA to acquire video frames. The interfacing architecture is implemented using the embedded technique which increases hardware utilizations. On the other hand, each subblock is designed in an optimised way to get optimised hardware utilizations for the entire architecture. This is achieved by replacing complex logical architectures with the corresponding logical architectures with basic gates.

7 | CONCLUSION

In this paper, an efficient hardware architecture to interface low cost digital camera with FPGA is proposed which can be used for real time video capturing and processing. To generate efficient interfacing hardware architecture, each subblock (i.e., Controller, Image Capture, Color Plane Conversion, VGA Signal Generator, and VGA to HDMI Conversion) are optimised using different optimizing techniques. Parallel architecture is used to optimise the Image Capture and VGA Signal Generator blocks. Similarly, the use of shifters and adders reduces the hardware utilization of different Color Plane Conversion blocks, respectively. Modified DDC Conversion and TMDS Encoder blocks are used to optimize the VGA to HDMI Conversion where the DDC Conversion block is optimised using the FSM and TMDS Encoder block is modified at the architectural level.

However, at the time of optimization of these architectures, data accuracy was also taken care of by considering sufficient intermediate bit sizes. As a result, the proposed method has less hardware complexity, low cost, and is more accurate compared to existing techniques. In future, an interface of higher resolution camera (HD, 4K etc.) with more frame rates and more sophisticated processing algorithm to generate better quality images will be implemented.

8 | ABBREVIATIONS AND APPENDICES

The abbreviations used in this paper are as follows

DDC ← Display Data Channel; EDID ← Extended Display Identification Data; HDMI ← High Definition Multimedia Interface; I2C ← InterIntegrated Circuit; SCCB ← Serial Camera Control Bus; TMDS ← Transition Minimised Differential Signalling; VGA ← Video Graphics Array.

Similarly, the symbols are used in alright the Algorithms and Figures present in this paper are as follows

○ ← XNOR Operation; ⊕ ← XOR Operation; & ← AND Operation; | ← OR Operation; ⟨⟩/⟨,⟩| ← Concatenation Operation; (x : y) ← Data of Length (x=y+1); (N) ← Bit Present in Nth Position of a Binary Digit.

ORCID
Sayantam Sarkar © https://orcid.org/0000-0002-5763-8692
REFERENCES

1. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, (3rd ed.). Pearson Education, London (2008)

2. Murat Tekalp, A.: Digital Video Processing, (2nd ed.). Prentice Hall, Hoboken (2015)

3. Charls, H. Roth.: Digital System Design using VHDL. Cengage Learning, Boston, (2003)

4. Digilent NexysVideo FPGA Board User Guide. https://reference.digilentinc.com/reference/programmablelogic/nexysvideo/reference manual.pdf. Accessed 26 Mar 2020

5. Digilent Zybo Z7 FPGA Board User Guide. https://reference.digilentinc.com/media/reference/programmablelogic/zyboz7/zyboz7_rm.pdf. Accessed 26 Mar 2020

6. Sid, Y., et al.: Embedded realtime video processing system on FPGA. In: Proceedings of the 5th International Conference on Image and Signal Processing, pp. 85–92. Springer, North Africa (2012). https://doi.org/10.1007/978-3-642-312540_10

7. Abdaoui, A., et al.: Video acquisition between USB 2.0 CMOS camera and embedded FPGA system. In: Proceedings of the 5th International Conference on Signal Processing and Communication Systems, pp. 1–5, USA (2011). https://doi.org/10.1109/ICSPCS.2011.6140863

8. Biren, M., Berry, F.: DreamCam: a modular FPGA-based smart camera architectures. J. Syst. Archit. 26, 519–527 (2014). https://doi.org/10.1016/j.sysarc.2014.01.006

9. Maldenija, B., et al.: Computationally efficient implementation of video rectification in and FPGA for stereo vision applications. In: Proceedings of the 5th IEEE International Conference on Computation and Automation for Sustainability, pp. 219–224, Sri Lanka (2010). https://doi.org/10.1109/ICIAMS.2010.5715663

10. Mosqueron, R., Dubois, J., Paindavoine, M.: Embedded image processing/comparison for high speed CMOS sensor. In: Proceedings of the 14th EURASIP European Signal Processing Conference (EUSIPCO 2006), vol. 2010, pp. 1–17, Florence (2010). https://doi.org/10.1109/EUSIPCO.2010.920603

11. Zhao, B., Zhang, X., Chen, S.: A CMOS image sensor with onchip motion detection and object localization. In: Proceedings of the IEEE International Conference on Custom Integrated Circuits, pp. 1–4, USA (2011). https://doi.org/10.1109/CICC.2011.6055400

12. OmniVision OV7670 Camera Datasheet. https://www.votil.nl/docs/OV7670.pdf. Accessed 26 Mar 2020

13. OmniVision OV9655 Camera Datasheet. http://electricstuff.co.uk/OV9655datasheetannotated.pdf. Accessed 26 Mar 2020

14. OmniVision SCCB Interfacing Datasheet. http://www.4cs.umanitoba.ca/~jacky/Teaching/Courses/74.795/LocalVision/ReadingList/ovsccb.pdf. Accessed 26 Mar 2020

15. Xilinx Clocking Wizard IP Core Datasheet. https://www.xilinx.com/support/documentation/ip_documentation/clk_wiz/v5_5/ps065clkwiz.pdf. Accessed 26 Mar 2020

16. Vivado User Manual. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1_ug910vivadogetstarted.pdf. Accessed 26 Mar 2020

17. Hagara, M., et al.: Grayscale image formats for edge detection and for its FPGA implementation. Microprocessor. Microsyst. 75, 103056 (2020). https://doi.org/10.1016/j.micropro.2020.103056

18. Digilent PMOD Specifications. https://reference.digilentinc.com/reference/pmod_specification.pdf. Accessed 26 Mar 2020

19. Xilinx Block RAM IP Core Datasheet. https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf. Accessed 26 Mar 2020

20. Bhairannawar, S.S., et al.: Implementation of fingerprint based biometric system using optimised 3/5 DWT architecture and modified CORDIC based FFT. Circ. Syst Signal Process. 37(1), 342–366 (2018). https://doi.org/10.1007/s000340170550

21. Fenger, C., Paret, D.: The I2C Bus: From Theory to Practice, (Har/Dskt edn.) Wiley-Blackwell, Hoboken (1997)

22. Hu, Z.W.: I2C protocol design for reusability. In: Proceedings of the 3rd IEEE International Symposium on Information Processing, pp. 83–86, China (2010). https://doi.org/10.1109/ISIP.2010.51

23. Bharath, K.B., Kumaraswamy, K.V., Swan, R.K.: Design of arbitrated I2C protocol with DO254 compliance. In: Proceedings of the IEEE International Conference on Emerging Technological Trends, pp. 1–5, India (2016). https://doi.org/10.1109/ICETTE.2016.7873672

24. Weste, N., Eshraghian, K.: Principles of CMOS VLSI Design: A System Perspective, (2nd ed.). Addison-Wesley Publishing, Boston (2002)

25. Singh, A., Srinivasan, S.: Digital Signal Processing Implementations: Using DSP Microprocessors–With Examples from TMS320C54x. Nelson Engineering, Boston (2003)

26. WANG, H.B., et al.: VGA display driver design based on FPGA. In: Proceedings of the 18th IEEE/ACIS International Conference on Computer and Information Science (ICIS), 530–535, China (2019). https://doi.org/10.1109/ICIS46139.2019.8904166

27. Shi, D., Ye, X.: Design of VGA display system based on CPLD and SRAM. In: Proceedings of the Third International Conference on Intelligent System Design and Engineering Applications, pp. 579–582, China (2013). https://doi.org/10.1109/ISDEA.2012.141

28. Wang, G., Guan, Y., Zhang, Y.: Designing of VGA character string display module based on FPGA. In: Proceedings of the IEEE International Symposium on Intelligent Ubiquitous Computing and Education, pp. 1–5, China (2009). https://doi.org/10.1109/IUCE.2009.12

29. Venuti, S., Chase, J., Iwami, L.: Highdefinition multimedia interface (HDMI). In: Handbook of Visual Display Technology, pp. 1–13. Springer (2015). https://doi.org/10.1007/978-3-642-35477-3_72

30. Kubiak, I., Przybysz, A.: DVI (HDMI) and display port digital video interfaces in electromagnetic crosstalk process. In: Proceedings of the International Symposium on Electromagnetic Compatibility EMC EUROPE, pp. 388–393, Spain (2019). https://doi.org/10.1109/EMC.Europe.2019.8872097

31. HDMI 1.4 Specifications. https://www.hDMI.org/manufacturer/hDMI_1_4/. Accessed 26 Mar 2020

32. Video Electronics Standards Association Datasheet for DDC: Enhanced Extended Display Identification Data Standard. Release A, Revision 2 (2006)

33. Video Electronics Standards Association Datasheet for DDC: Enhanced Display Data Channel (EDDC) Standard Version 1.2 (2007)

34. Xilinx LVDS IP Core. https://www.xilinx.com/support/documentation/application_notes/xapp1315lvdsourcesyncsderivestocolcmultiplication.pdf. Accessed 26 Mar 2020

35. NXP UM10204 Bus Specifications. https://www.nxp.com/docs/en/userguide/UM10204.pdf. Accessed 26 Mar 2020

36. Sobhan Bluian, M.A., et al.: Low Power D Flip-Flop Serial In/Parallel Out Based Shift Register. In: Proceedings of the IEEE International Conference on Advances in Electrical, Electronic and Systems Engineering, pp. 180–184. IEEE, Putrajaya (2016). https://doi.org/10.1109/ICAES.2016.7888034

37. Park, J., et al.: A novel stochastic model-based eyediagram estimation method for 8B/10B and TMDScoded highspeed channels. IEEE Trans. Electromagn. Compat. 60(No. 5), 1510–1519 (2018). https://doi.org/10.1109/1 EMC.2018.8360868

38. Singh, A., Srinivasan, S.: Efficient FPGA architecture of optimised Haar wavelet transform for image and video processing applications. Multidimens. Syst. Signal Process. 32, 821–844 (2021). https://doi.org/10.1007/s110402007594

39. Wolf, W.: FPGA Based System Design. Pearson Education, London (2004)

40. Avnet TDNext Camera Datasheet. http://zedboard.org/sites/default/files/product_briefs/5338PBASPMODTDI14GV1d.pdf. Accessed 26 Mar 2020
43. Avnet Python 1300C Camera Datasheet. https://www.avnet.com/opasdata/d120001/medias/docus/187/PB_AESCAMONP13000CGV2ProductBrief.pdf. Accessed 26 Mar 2020

44. Digilent Pcam5C Camera Datasheet. https://store.digilentinc.com/pcam5Sc5mpfixedfocuscolorcameramodule. Accessed 26 Mar 2020

45. Terasic D8M Camera Datasheet. https://www.intel.com/content/dam/alterawww/global/en_US/portal/dsn/42/docusdmblk425008051605255d8mgpiousermanual.pdf. Accessed 26 Mar 2020

46. Yang, X., Zhang, Y., Wu, L.: A scalable image/video processing platform with open source design and verification environment. In: Proceedings of the 20th IEEE International Symposium on Quality Electronic Design (ISQED), pp. 110–117, USA 6–7 March 2019. https://doi.org/10.1109/ISQED.2019.8697816

47. Babu, A.: FPGA based Graphics controller. Int. J. Eng. Res. Technol. 3(8), 1–6 (2015). 22780181

48. Kumar Chate, R., Patil, S.S.: Serial transmission of video signal using TMDS encoder and VHDL implementation. Int. J. Adv. Res. Electr. Electron. Instrum. Eng. 7(7), 3119–3124 (2018). https://doi.org/10.15680/IJIRE.2018.0606051

49. Bhowmik, D., et al: Power efficient dataflow design for a heterogeneous smart camera architecture. In: Proceedings of the IEEE International Conference on Design and Architectures for Signal and Image Processing, pp. 1–6, Germany (2017). https://doi.org/10.1109/DASIP.2017.8122128

50. Zhou, W., et al.: Realtime implementation of panoramic mosaic camera based on FPGA. In: Proceedings of the IEEE International Conference on Real Time Computing and Robotics, pp. 204–309, Cambodia (2016). https://doi.org/10.1109/RCAR.2016.7784026

51. Xilinx Camera IP Core. https://www.xilinx.com/support/documentation/application_notes/sapp7941080p60camera.pdf. Accessed 26 Mar 2020

52. Honegger, D., Oleynikova, H., Pollefeys, M.: Realtime and low latency embedded computer vision hardware based on a combination of FPGA and mobile CPU. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4930–4935, USA (2014). https://doi.org/10.1109/IROS.2014.6943263

How to cite this article: Sarkar S, Bhairannawar SS, K.B. R. FPGA Cam: A FPGA based efficient camera interfacing architecture for real time video processing. IET Circuits Devices Syst. 2021;1–16. https://doi.org/10.1049/cds2.12074