Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 131 (2018) 641–646

Procedia Computer Science

www.elsevier.com/locate/procedia

8th International Congress of Information and Communication Technology (ICICT-2018)

Real-time Frame Buffer Implementation based on External Memory using FPGA

Dogancan Davutoglu[a]*, Nerhun Yildiz[b], Umut Engin Ayten[c], Vedat Tavsanoglu[d]

[a]*Department of Electrical & Electronics Engineering, Istanbul Kultur University, Istanbul 34191, Turkey*
[b]*Department of Media Engineering, Arm Limited, Leicester LE12 5RD, United Kingdom*
[c]*Department of Electronics & Communication Engineering, Yildiz Technical University, Istanbul 34220, Turkey*
[d]*Department of Electrical and Electronics Engineering, Isik University, Istanbul 34980, Turkey*

Abstract

In this paper, design of a real-time video frame buffer with an external memory interface is proposed. In addition, simulation and implementation processes of the design is described. The mentioned system is able to buffer video signals up to 1920×1080 full-HD resolution at 60 Hz frame rate. The memory interface is designed based on an external SDRAM memory and supports burst read/write operations. Input video resolution, video buffer size on memory and burst size of the memory interface are user defined and can be configured.

Keywords: Real-time; field programmable gate arrays; memory interface; video frame buffer

1. Introduction

In digital video processing, necessity of capturing video frames is increasing in many real-time applications; e.g., spatiotemporal filtering, differentiation on video frames and video processing using convolutional or cellular neural networks. In order to realize capturing process in real-time, a frame buffer architecture is needed as it is mentioned in references [1, 2, 3]. However, video frame buffer architecture requires high bandwidth, related to video signal. In other words, capturing video frame pixel data and using them in following processes causes a bottleneck if the video

* Corresponding author. Tel.: +90-555-436-12-82
*E-mail address:*d.davutoglu@iku.edu.tr

10.1016/j.procs.2018.04.307

resolution gets larger. With the proposed design in this paper, this problem can be solved and the need of frame buffer support can be fulfilled based on an external memory unit [4].

Mainly, the scope of this paper is to generate a flexible infrastructure for general use in video frame buffering process. In this way, it is being planned to use this video frame buffer in spatiotemporal filtering [5] and high order differentiation of video frames for extraction of moving object or background from video source. Additionally, implementation results on Altera Stratix IV GX FPGA kit using on-board external DDR3 SDRAM and simulation results are given in this paper. Video resolutions up to 1920×1080 pixels, various memory burst sizes and different frame delay configurations are successfully tested to prove flexibility of the architecture.

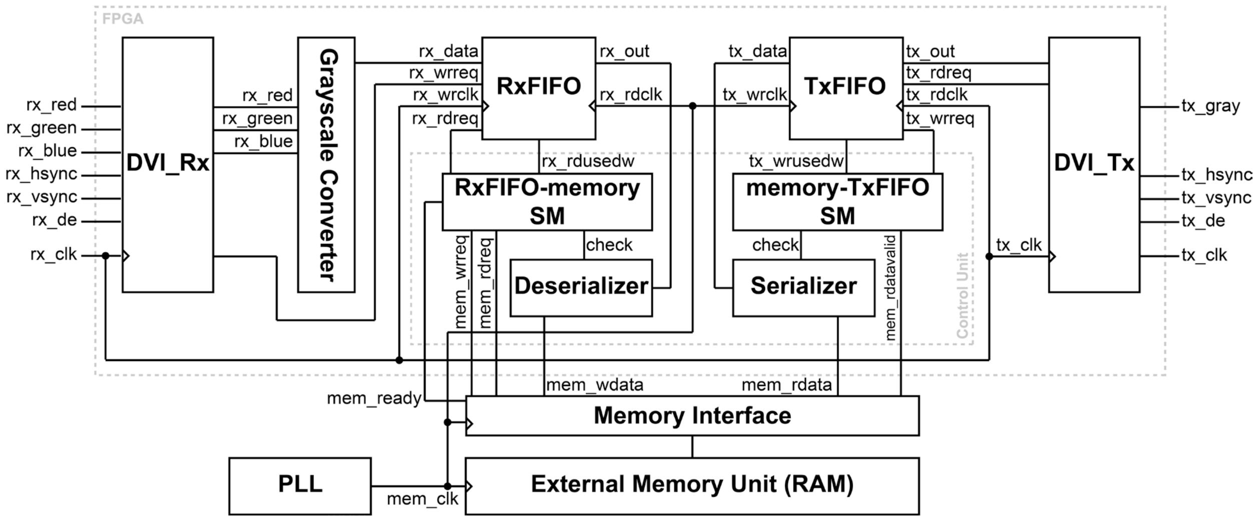

Fig. 1. Generalized block diagram of proposed architecture.

2. Architecture of the System

The focus of the design is to create an architecture that captures video frames and recalls them in subsequent processes. In this case, it must be considered that input and output video frame signals are regular and uninterrupted streams. Despite video stream, signal timing of SDRAM memory interface is irregular due to necessity of a refresh operation in certain time interval. Memory refresh operation causes memory interface to pause for several clock cycles. Thus, additional memory management for input and output video frame data is required to regulate video stream in order to process video frame data in real-time. More specifically, duration for processing a frame of full-HD resolution video signal (1920×1080 pixels @60 Hz) is only 16.67 milliseconds. For instance, if bit depth of the video signal is M and bus width of memory interface is N, required number of read and write processes is $(1920×1080×2×M)/N$ per duration between two consecutive frames.

Overview of the proposed architecture is given in Fig. 1. At the input and output stage, two FIFOs are used to store small amount of video data to queue them. Size of stored video data in FIFOs depends on burst size and data width of memory interface. On the other hand, queuing method prevents missing video frame data during pause period of memory interface. Captured video frame data are stored on external memory unit. DDR3 SDRAM is used as an external physical memory unit during prototyping process. In order to access external physical memory, Altera provided memory interface is used as in reference [6]. This interface overcomes refreshing, pre-charge and calibration of external physical memory unit.

In the core of the architecture, control unit handles external memory interface signals and checks data amount inside FIFOs to prioritize read and write operations.

2.1. FIFOs

In Fig. 2, generalized block diagram of Altera provided FIFOs is given. FIFOs have user defined data width and data depth which can be configured using generic parameters before synthesis [7]. Write and read controls of FIFO component are handled by *wrreq* and *rdreq* input signals. Asynchronous reset signal *aclr* can be used to clear all stored data inside FIFOs. Write clock signal of *RxFIFO* and read clock signal of *TxFIFO*, namely *rx_wrclk* and *tx_rdclk* are synchronized with video pixel clock signal. On the memory side, clock rate of external memory interface is higher than video pixel clock rate as it is supposed to be. Hence, FIFOs are generated as dual-clock type to control both memory interface and video signals with different clock rates. Stored frame pixel data inside FIFOs are observable through *wrusedw* and *rdusedw* output signals. *wrusedw* and *rdusedw* signals are updated on rising edges of *wrclk* and *rdclk* inputs, respectively.

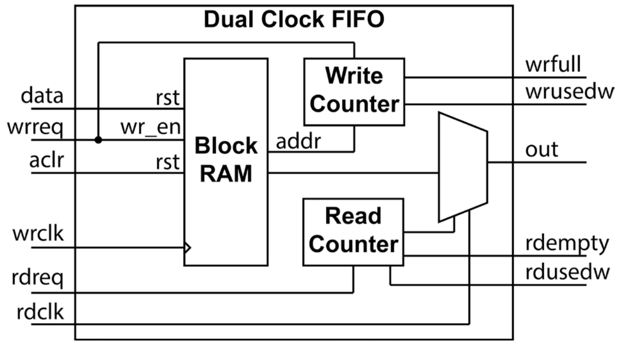

Fig. 2. Generalized block diagram for dual clock FIFO.

2.2. Memory Interface

Instead of designing a new memory interface from scratch, built-in vendor specific external memory interface is more convenient to integrate with the design. In this way, many control signals and operations are handled by external memory interface such as bank, row and column accesses in memory, clock and write enable signals, *dq* and *dqs* signals, pre-charge, refresh, calibration and initialization operations. Eventually, on the front-end of memory interface block, number of control signals are reduced to a few number of basic signals; i.e., write & read request, address, data and burst size signals [8]. In order to prevent encountering overflowed or empty FIFO problem, memory interface clock rate must be selected at least two times higher than video pixel clock rate; in other words, in one cycle of incoming pixel data, a read cycle and a write cycle must be completed.

2.3. Proposed Control Logic Unit

Major contribution to this work is made with control unit design, which bridges FIFOs, memory interface and physical memory.

During buffering, it is crucial to capture all video data from video source without interruption. Similarly, captured video data must feed video output device constantly. In ideal case, *RxFIFO* must be almost empty and *TxFIFO* must be almost full to accept video data and make video output ready for any time.

In Fig 3., state diagrams of state machines inside control logic unit is given. There are two state machines in the basis of control logic unit that are *RxFIFO-memory state machine* and *memory-TxFIFO state machine*, respectively. There must be enough data on memory before read request from memory occurs. Therefore, after memory initialization, caching process begins. Then, *RxFIFO-memory state machine* checks data amount inside both FIFOs. If data amount inside *RxFIFO* is enough for reading process, *RxFIFO-memory state machine* decides to read data from *RxFIFO* and write it on memory. Otherwise, *RxFIFO-memory state machine* checks whether there is enough space in *TxFIFO* or not. If there is, *RxFIFO-memory state machine* starts reading process from memory. However, read from memory process is handled by *memory-TxFIFO state machine*. These two state machines work parallel to each other. In addition, if data width of memory interface is higher than video pixel data width, packing and serialization operations are required before writing on memory and after reading from memory, respectively. Likewise, this requirement is satisfied with packing and serializing blocks in the design.

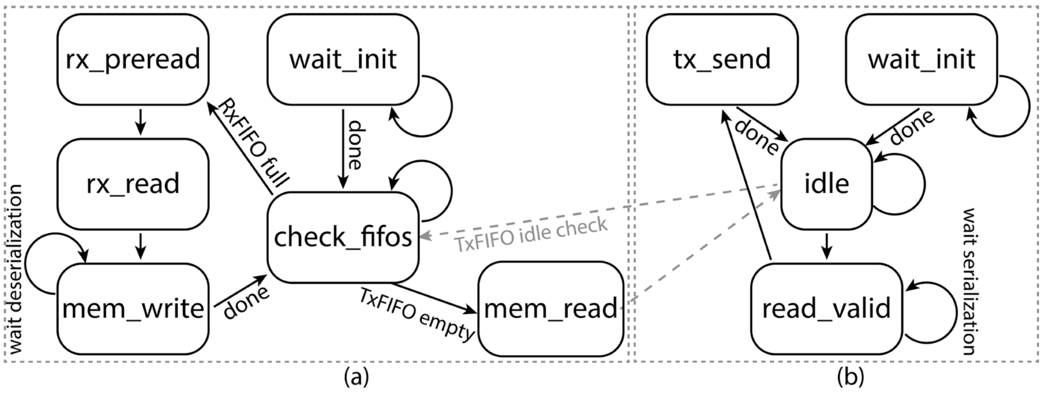

Fig. 3. State diagrams of state machines inside control unit. (a) RxFIFO-memory state machine (b) memory-TxFIFO state machine.

2.4. Video Input & Output Block

At the top layer of the design, there are video input & output units, which are designed to capture video data from a video source and send buffered video data to target device. Block diagram of video input & output units is given in Fig. 4. Most important task of video input & output units is satisfying timing requirements for video stream.

Firstly, all video processing operations are carried out in grayscale. In order to convert pixel color values to grayscale, averaging method is used which simply divides the summation of pixel color values by number of pixel colors ((*red* + *green* + *blue*) / 3). As a result, data width of pixels reduced to gain speed and simplify the design. Input block is synchronized with input video pixel clock, data enable (*de*), horizontal synchronization (*hsync*) and vertical synchronization (*vsync*) signals. If data enable signal is active, capturing starts on *RxFIFO* block. Despite input block, output video signals are generated apart from input video signals. In this way, video stream starts independently of input video signals after buffering process ends. In order to match video source with video output block, specifications for video output block namely video resolution, polarity and blanking information must be configured before synthesis. Additionally, beside choice of DVI on prototype, other video standards such as VGA, HDMI and DisplayPort are also supported.

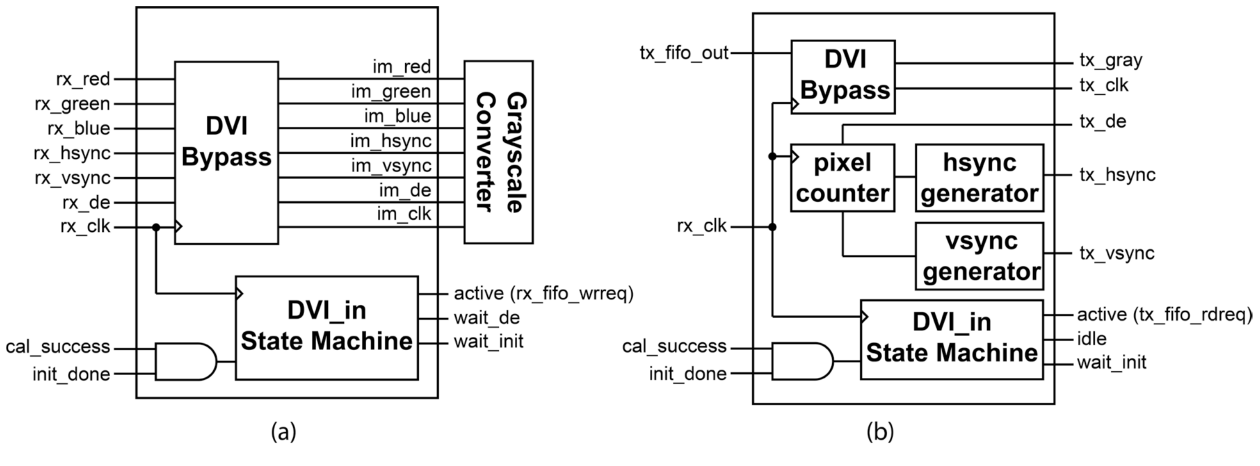

Fig. 4. Block diagram of video input & output units. (a) DVI input block (b) DVI output block.

3. Simulation

Observation of signal timings is the most important part of the simulation. In order to simulate proposed design, ModelSim Altera Edition software was chosen. Additionally, VHDL was preferred in all hardware descriptions of the proposed design.

In Fig. 5, a waveform of write operation is given. In simulation, burst size was selected as ‘4’. Thus, memory address remains same during 4 successful burst operations. In background, memory interface automatically increases memory address before each burst starts. Data (*avl_wdata*) is regularly updated between two write requests because of pixel data packing process. After four writing operations, pixel data were written on addresses between ‘4’ and ‘7’ successfully.

Secondly, a simulation of read operation is given in Fig. 6. After read request signal is asserted, reading operation starts. At the end of reading operation, *avl_rdata_valid* signal states that reading operation is finished. By the way, data can be read via *avl_rdata* signal when *avl_rdata_valid* is asserted.

Moreover, waveform of video signals in simulation is given in Fig. 7. To reduce simulation time and prove basic concepts, video resolution was chosen as 10×1920, i.e., 10 lines of pixels and 1920 pixels in each line. Buffered frame number was selected as ‘1’, so transmitter part of DVI starts to operate after one frame is captured. Generated *hsync* and *vsync* signals are synchronized with their corresponding signals at receiver part of DVI signals.

Finally, input video data and output video data were compared, then it is verified that external memory stores captured video frame data and sends it to DVI output after desired frame delay is achieved.

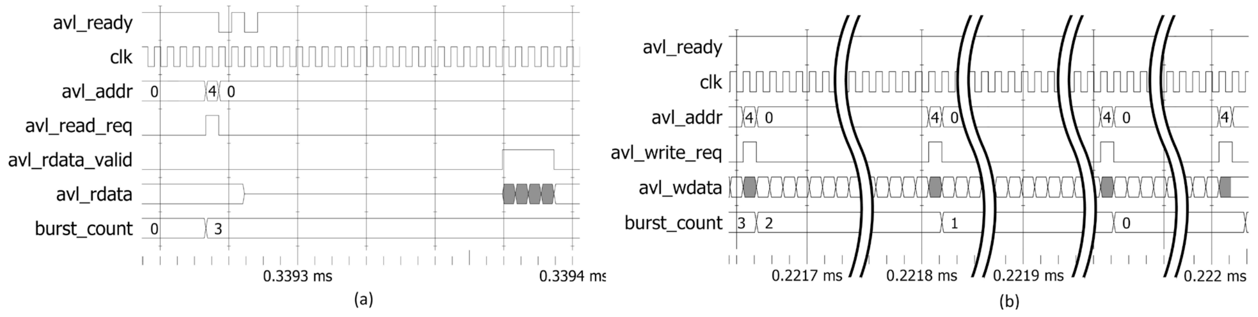

Fig. 5. (a) Simulation result of read operation from ModelSim software; (b) Simulation result of write operation from ModelSim software.

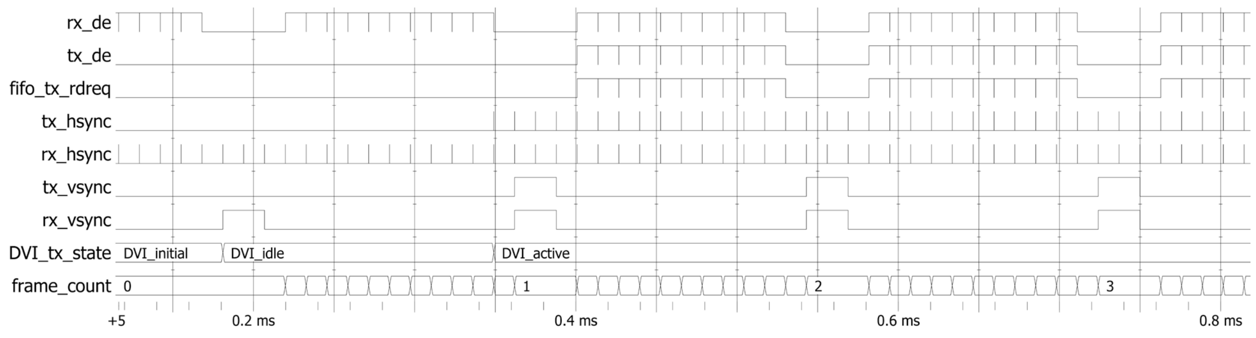

Fig. 6. Waveform of DVI signals from simulation.

4. Implementation

After simulation, the proposed design is implemented on Altera Stratix IV GX FPGA development board [9]. For the memory part of the proposed design, DDR3 SDRAM is preferred and then, memory interface tested at 400, 533, 667 MHz memory clock rates. All tested clock rates were sufficient to process video frames up to full-HD resolution. In order to achieve these clock rates for memory interface, PLL was used to multiply clock frequency of oscillators.

In implementation, pixel data width was 8 bits. Different data widths were used for external memory; i.e., 16 and 64 bits. Different burst sizes were successfully tested for write and read operations; e.g., ‘1’, ‘2’ for x16 memory interface, ‘1’, ‘2’, ‘4’, ‘8’ for x64 memory interface.

5. Conclusion and Future Work

In conclusion, proposed architecture of the design was explained, simulated and implemented on FPGA device with different video resolutions and different memory data widths. Using proposed real-time video frame buffer, it is possible to design video processors that require an external memory to capture video data and use them in subsequent processes. More specifically, mentioned design can be used as an infrastructure for differentiation or temporal filtering on video frames. Furthermore, this design can be a solution for the use of external memory in convolutional or cellular neural network designs.

Lastly, proposed design is tested up to full-HD resolution due to resolution limitation of the video daughter card interface which is connected to FPGA kits. Despite that, it is feasible to operate this frame buffer at higher resolutions such as 4K and 8K using appropriate video interface device.

References

[1] Yildiz N, Cesur E, Kayaer K, Tavsanoglu V and Alpay M, "Architecture of a Fully Pipelined Real-Time Cellular Neural Network Emulator," in *IEEE Transactions on Circuits and Systems I*: Regular Papers, vol. 62, no. 1, pp. 130-138, Jan. 2015.

[2] Yildiz N, Cesur E and Tavsanoglu V, "On the way to a Third Generation Real-Time Cellular Neural Network Processor," *CNNA 2016; 15th International Workshop on Cellular Nanoscale Networks and their Applications*, Dresden, Germany, 2016, pp. 1-2.

[3] Yildiz N, Cesur E and Tavsanoglu V, "Design of a third generation Real-Time Cellular Neural Network emulator," 2014 *14th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA)*, Notre Dame, IN, 2014, pp. 1-2.

[4] Graham A, Jody D: IP solves the increasing challenge of implementing an interface to off-chip DDR SDRAM. *International Engineering Consortium - DesignCon 2007*, 2007, pp 578-601

[5] Yildiz N, Cesur E and Tavsanoglu V, "A Discussion on Spatiotemporal Filtering on a Third Generation Real-Time Cellular Neural Network Processor" *CNNA 2016; 15th International Workshop on Cellular Nanoscale Networks and their Applications*, Dresden, Germany, 2016, pp. 1-2.

[6] Wang B, Du J, Bi X and Tian X, "High bandwidth memory interface design based on DDR3 SDRAM and FPGA," 2015 *International SoC Design Conference (ISOCC)*, Gyungju, 2015, pp. 253-254. doi: 10.1109/ISOCC.2015.7401743

[7] SCFIFO and DCFIFO IP Cores User Guide on https://www.altera.com.

[8] External Memory Interface Handbook Volume 3: Reference Material on https://www.altera.com.

[9] Stratix IV GX FPGA Development Board Reference Manual on https://www.altera.com