FPGA based Compressive Sensing Framework for Video Compression on Edge Devices

Abin Bassam Ayub
Department of Mathematics
Indian Institute of Science
Bangalore, India
abinbassam@iisc.ac.in

Pallab Kumar Nath
Department of Electronics and Systems Engineering
Indian Institute of Science
Bangalore, India
pallabkumar@iisc.ac.in

Venkat Rangan
tinyVision.ai Inc.
San Diego, California
venkat@tinyvision.ai

Chetan Singh Thakur
Department of Electronics and Systems Engineering
Indian Institute of Science
Bangalore, India
csthakur@iisc.ac.in

***Abstract*— Compressive Sensing (CS) is an emerging signal processing methodology, which compresses the signal being acquired at the time of sensing. As it relies on the sparsity of the signals, CS allows us to sample the signal at a rate much below the Nyquist sampling rate. Recent advancements in CS made us able to capture compressed videos and images on a sensory level in the camera. As CS cameras are still in the research phase, it is not commercially available. In this paper, we propose a simple FPGA based hardware architecture which makes conventional cameras to produce the compressive sensed video. Here the proposed hardware architecture emulates the sensing methodology of pixel-wise coded exposure imaging for 13× compression, which can be modified based on applications. Our framework does not require any changes to the conventional image sensor as it feeds off the video from the same sensor, this is a huge advantage over previous works in CS video compression. Our implementation results show that the proposed method is effective and easily adaptable to any conventional camera. The proposed framework can be deployed on edge devices with low-cost conventional camera to reduce the communication data rate and help in saving the significant power.**

Keywords— compressive sensing, FPGA, video compression, Pixel-wise Coded Exposure imaging

I. Introduction

Compressive Sensing (CS) is a novel sensing methodology where the signal is compressed at the time of sampling/sensing. It requires the signal to sparsely representable. This sensing technique has seen a lot of advancement in recent years as most natural signals can be represented sparsely (e.g. images in DCT domain). It has been shown that compressive sensing can be used effectively for temporal video compression [1,2].

CS video compression has been gaining a lot of attention lately as it can achieve compression higher than that of the conventional methods. Previous works in CS imaging systems have either used optical image apparatus to pre-process (modulation) the video before passing it to the image sensor [1,2] or replaced the conventional image sensor with a sensor based on a CMOS architecture [3,4,5] to produce the compressed sensed video at the sensory level. But these CS cameras are still in the research phase and it is not yet available commercially. As the compression is done at the sensory level, it can be implemented by modifying the control unit of the sensor [1,2,3]. This makes it a perfect candidate for edge devices with very low-cost cameras where we have very limited on-device storage and low data transmission bandwidth.

In this paper, we propose an FPGA based hardware architecture (Figure 1) that makes conventional cameras to emulate CS cameras. Our framework is helpful in building low-cost imaging system with the capability of the CS camera. However, it requires simple post-processing of the captured video from the sensor, which can be easily implemented using an FPGA as discussed in Section III.

Proposed method consists of 2 parts: Sensing (Encoding) and Reconstruction (Decoding). For sampling, we emulate Pixel-wise Coded Exposure (PCE) imaging [1,2,3]. Here each image-pixel is independently exposed to only a specific amount of time and the start time of that exposure is decided by a randomly generated sensing matrix. This has shown to be an efficient way to capture temporal compressed videos [1,2,3].

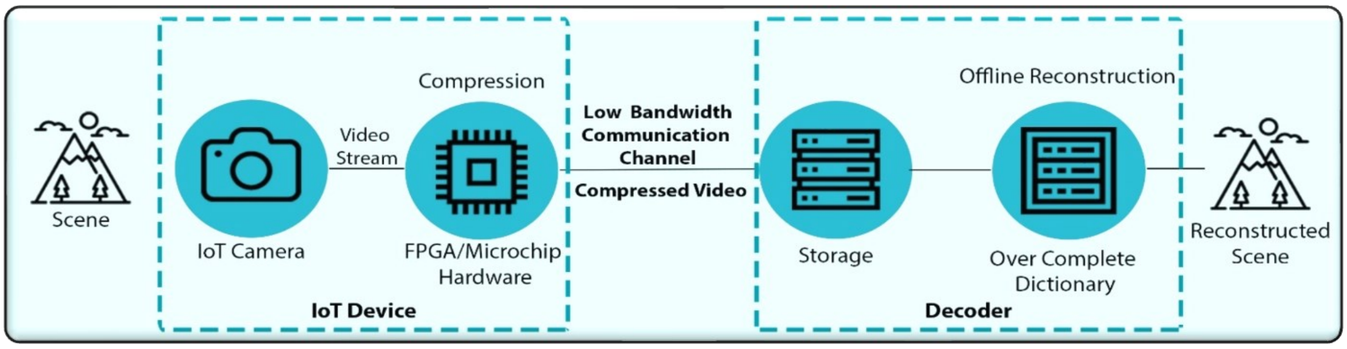

Fig. 1. Our video compression framework for edge devices

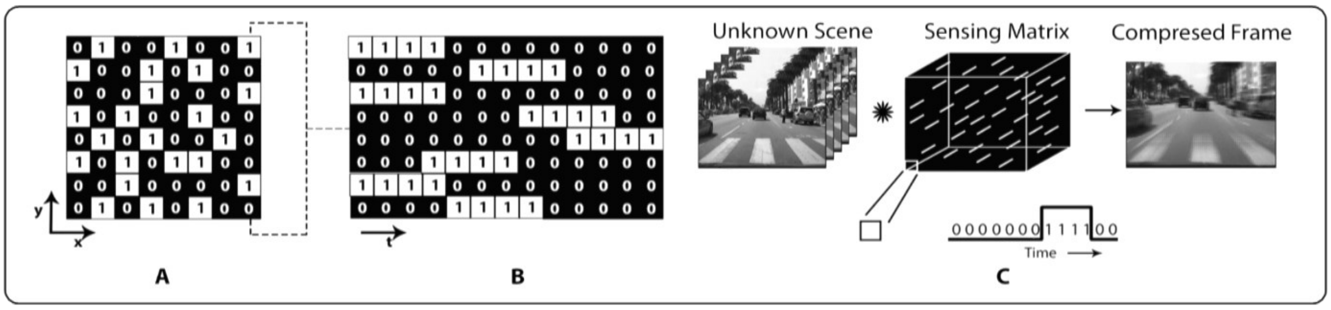

Fig 2. **A**) First frame of Sensing Matrix. **B**)Time axis of the last column of Sensing Matrix in **A**. **C**) Compression of video using the Sensing matrix (T_b= 4).

For reconstruction, we learned an over-complete dictionary (discussed in Section IV) to represent video patches sparsely and we used common sparse approximation techniques for reconstruction of the video patches. We did a weighted average of these patches to reconstruct the whole video. We used experiments and simulations to show the feasibility and efficiency of our proposed framework.

II. Overview of our Approach

Let $\boldsymbol{V}(\boldsymbol{M} \times \boldsymbol{N} \times \boldsymbol{T})$ denotes the video frames from the camera. $\boldsymbol{M}, \boldsymbol{N}$ corresponds to spatial resolution and $\boldsymbol{T}$ is temporal resolution as shown in Fig. 2. To achieve '$\boldsymbol{T}$' times compression in the Temporal Domain, we encode $\boldsymbol{V}$ into a single image of size $\boldsymbol{M} \times \boldsymbol{N}$ using *PCE* imaging as follows.

A. Sensing (Encoding)

For Sensing we emulate *PCE*, where each pixel is exposed at a random time for a 'single-on' fixed duration $\boldsymbol{T_b}$ **(**Bump Size**)** within the Frames $\boldsymbol{T}$. So, we get a single frame at the end of '$\boldsymbol{T}$-Frames' *(*$\boldsymbol{I}$), which has necessary information for the reconstruction of $\boldsymbol{V}$. To achieve this process, we generate a Sensing Matrix with random exposure for each pixel $\boldsymbol{S}(m,n,f) \in \{0,1\}$, where $\boldsymbol{S}$ is 1 for the exposed pixel and 0 for the rest. For each pixel, there is only 1 bump of fixed length ($\boldsymbol{T_b}$). Starting position of bumps is randomly generated using modified uniform distribution (discussed in **Algorithm-1**) to give equal exposure to all the frames in video $\boldsymbol{V}$**.** We sense the video by passing it through the sensing matrix and then averaging over the exposed pixels.
As it is generated randomly, this can be easily implemented in hardware using Linear Feedback Shift Register (LFSR) [9] based architecture (discussed in Section III). The whole process can be represented by the following equation.

$$\boldsymbol{I}(m,n) = \frac{\sum_{f=1}^{F} \boldsymbol{S}(m,n,f) * \boldsymbol{V}(m,n,f)}{\boldsymbol{T_b}} \quad (1)$$

B. Reconstruction (Decoding)

Now we have the sensing matrix ($\boldsymbol{S}$) and the compressed image ($\boldsymbol{I}$). Equation (1) can be written in matrix form as $\boldsymbol{I} = \boldsymbol{SV}$, here we have an under-determined linear system as our observation $\boldsymbol{I}$ has a smaller number of elements than the unknown $\boldsymbol{V}$. This can be solved if $\boldsymbol{V}$ has a sparse representation in a transformed domain (Learned-Dictionary-$\boldsymbol{D}$) [5,6,8]. i.e., $\boldsymbol{V}$ can be represented as:

$$\boldsymbol{V} = \boldsymbol{D\alpha} = \boldsymbol{D_1\alpha_1} + \boldsymbol{D_2\alpha_2} + \dots + \boldsymbol{D_k\alpha_k} \quad (2)$$

where $\boldsymbol{\alpha}$ is the sparse coefficient vector (i.e., most elements $\boldsymbol{\alpha_i}$s of $\boldsymbol{\alpha}$ are zero) and $\boldsymbol{D_i}$ s are the atoms/elements of $\boldsymbol{D}$.

So, our objective function becomes:

$$\min_{\boldsymbol{\alpha}} \|\boldsymbol{I} - \boldsymbol{SD\alpha}\|_2^2 \text{ Subject to } \|\boldsymbol{\alpha}\|_0 < P \quad (3)$$

Where $\|\boldsymbol{\alpha}\|_0$ is the $\mathbf{L_0}$ norm, which is just a count of non-zero elements in $\boldsymbol{\alpha}$. The count P determines the sparsity of the reconstructed frames. Equation (3) can be solved using Sparse Approximation Algorithms such as Orthogonal Matching Pursuit (OMP) as discussed in Section IV.

III. Hardware Architecture

The proposed CS architecture has of two parts: Sensing matrix generation and Compressed frame generation.

A. Sensing matrix Generation

The proposed architecture can easily be modified to achieve any compression ratio by changing the dimension of the sensing matrix. For ease of understanding, an architecture that can generate a sensing matrix to achieve 13X compression of the input video sequences is presented here. In the proposed architecture, an 8×8 sensing matrix is generated for each frame. Hence the size of the sensing matrix for video compression is 8×8×13. The 8×8 sensing matrix for an input frame raster scans the pixels in a non-overlapping manner. The element of a matrix is either "0" or "1". The value "1" indicates that the pixel value of that location for the current frame contributes to compute pixel value of the same location in the compressed frame. The LFSR based sensing matrix generating algorithm is as follows:

Algorithm 1: LFSR based Sensing Matrix generation

```
Initialize sensing matrix (S) with zeros;
for m = 1 to no of row in sensing matrix do
for n = 1 to no of column in sensing matrix do
index = 1 to 13;                 //valid index generated from LFSR
If (index <5) then
initial frame index = 1;
else if (index > 9) then
initial frame index = 10;
else
initial frame index = index;
end if
for k = 0 to 3 do
frame index = initial frame index + k;
S (m, n, f) = 1;          // frame index=f
end for k;
end for n;
end for m;
```

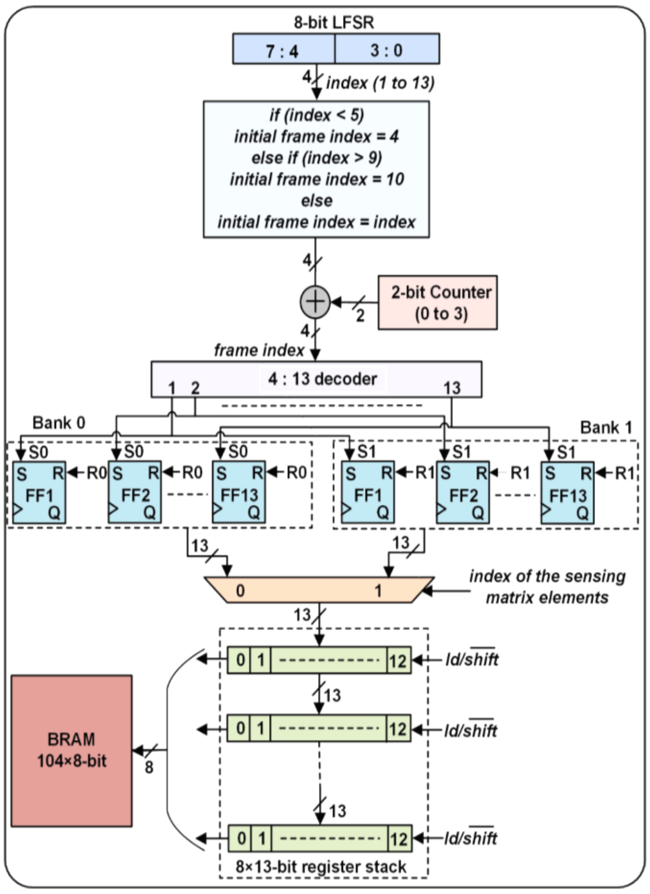

Fig. 3. Architecture for LFSR based Sensing Matrix generation

The VLSI architecture for LFSR based sensing matrix generation algorithm is shown in Fig 3. The upper 4 bits of an 8-bit LFSR is used to generate the initial frame index of the exposed pixel. The 8-bit LFSR is used to increase randomness for choosing initial frame index. For the proposed design, valid indexes are the 1 to 13. If we are using uniform distribution for generating initial frame index our initial and final frames get under exposed. To give equal exposure to all the frames, we are considering the frame index 1 to 4 as 1 and the frame index 10 to 13 as 10. The frame indexes for the 4 exposed pixel locations are generated in four consecutive clock cycles by adding 0 to 3 values to the initial frame index. A 2-bit binary counter is used to generate these values. The frame index is fed to a 4:13 decoder circuit. The outputs of the decoder unit are connected to active high SET (S) inputs of D-flip flops (DFFs). At the positive edge of the clock cycle, the output of DFF is set to “1” depending on the values of S signal. The DFFs are arranged in two banks (bank 0 and bank 1). Each bank consists of 13 DFFs one for each frame. Two banks are worked as ping-pong manner: when one is used for storing the values of the sensing matrix, the other is being initialized to zero using active high RESET (R) signal. Bank 0 and bank 1 are used to store even and odd indexed values of the sensing matrix respectively. The signals S0 and S1 represent signal S for bank 0 and bank 1, respectively. The output of a register bank is selected by multiplexer and stored in 8×13-bit register stack with the help of active high *ld* signal. The register stack become full after processing 8-row elements of the sensing matrix. After that active low *shift* signal left shifts the content of the register

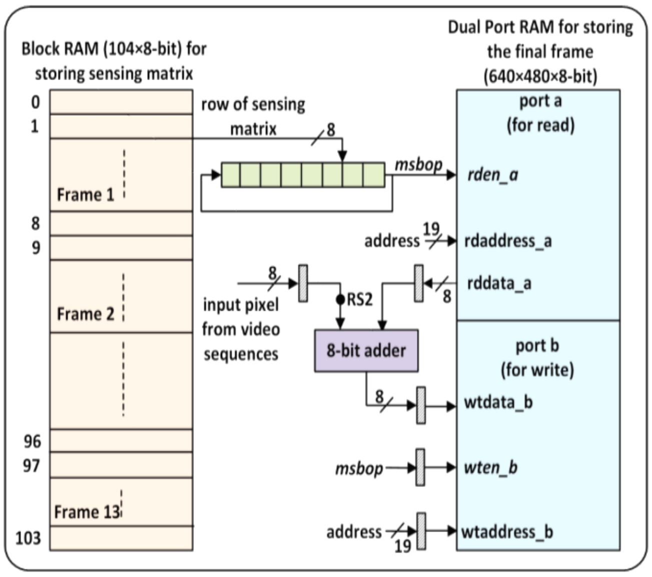

Fig. 4. Architecture for compressed frame generation

stack for 13 clock cycles and generates an 8-bit output in each clock cycle. The outputs are stored in a Block RAM (BRAM) of dimension 104×8-bit. One output corresponds to one row of an 8×8 sensing matrix for one frame. The first output is stored in location 0, second is in location 8, third is in location 16 and so on. After storing all the 13 outputs, the same process repeats for generating the entire sensing matrix.

B. *Compressed frame generation.*

The compressed frame generation process is instantiated after the completion of sensing matrix generation. The architecture for generating the compressed frame is shown in Fig 4. The value of a pixel location in the compressed frame is the average of the four exposed pixels in the video sequences. In the proposed design, the frame size of the video sequences is 640×480. The compressed frame of dimension 640×480×8-bit is stored in Dual-port RAM (DP-RAM) inside the FPGA chip. Port “a” is used for reading and port “b” is used for writing pixel value in the DP-RAM. The row of a sensing matrix is fetched from the BRAM and stored in an 8-bit rotate left register. The MSB (*msbop*) of this register is used as the read enable signal (*rden_a*) of the DP-RAM. In each clock cycle, *rden_a* (active high) signal along with a 19-bit address (rdaddress_a) reads pixel from the DP-RAM for further processing. The high value of *msbop* signal indicates that the pixel value from the current frame of that location contributes to generating the compressed frame. Input pixel is fetching from video sequences (grayscale) in each clock cycle. The input pixel value and the pixel value fetched from the DP-RAM are added together and stored back in the DP-RAM at the same location (wtaddress_b) in the next clock cycle with the help of active high *wten_b* signal. In DP-RAM, the read and write operations in a location are performed only if *msbop* signal is high. The value for a pixel in the DP-RAM is calculated by taking average of four-pixel values from the same location of four consecutive input video frames. To avoid overflow in the pixel values in the compressed frame, input pixel values are divided by four (RS2, right shift by 2) before addition. The next location of the BRAM is fetched after the completion of one row of the input frame. This process is continued until all the frames are scanned and compressed frame is generated. For interfacing with the camera, two input memory blocks (each has dimension of 640×480×8-bit) are associated with the proposed design and operate as ping-pong fashion. When one memory block acquires new frame from the camera, other delivers stored frame to the proposed design as input for processing.

IV. Reconstruction

Video reconstruction from the generated compressed frames (Section III. *B*) is done offline in software. It has been shown that the video can be effectively reconstructed using a learned overcomplete dictionary and OMP [1,2].

A. Dictionary Learning using K-SVD Algorithm

To represent the video as a sparse signal we learned an overcomplete dictionary using the K-SVD algorithm [1,2,5,7,8,10]. We use learned dictionary rather than 3D Discrete Cosine Transform (DCT) or off-the-shelf dictionary (scene-based) as the learned dictionary offers much more versatility in dealing with videos from different scenes. To maximize the scope of our dictionary, we trained it on video patches from different scenes and rotated the patches at different angles. But the videos were about the same frame rate to match our target frame rate for reconstruction. To fix the patch size we couldn't go too low as it makes our reconstruction process more time complex and we couldn't go much higher as it affects sparsity of the video patches. So, we chose patches of dimension 8×8×13 (for 13× compression). We trained the dictionary for 10000 basis elements as it was giving good enough reconstructed results as shown in Fig. 5. With this method, we were able to capture complex edge shifting and rotations in videos, which would have been hard to reproduce with an off-the-shelf dictionary.

B. Sparse Reconstruction using Orthogonal Matching Pursuit

Once we learn the over-complete dictionary, we apply a standard sparse approximation, Orthogonal Matching Pursuit (OMP) to recover the video ($\boldsymbol{V}$) from a single compressed image ($\boldsymbol{I}$). Combining (1) and (2), we get $\boldsymbol{I} = \boldsymbol{SD\alpha}$, where the captured coded image $\boldsymbol{I}$, the sensing matrix $\boldsymbol{S}$, and the over-complete dictionary $\boldsymbol{D}$ are known. By taking $\boldsymbol{\phi} = \boldsymbol{SD}$, we can write (3) as follows:

$$\hat{\alpha} = \min_{\boldsymbol{\alpha}} ||\boldsymbol{I} - \boldsymbol{\phi\alpha}||_2^2 \quad \text{Subject to } ||\boldsymbol{\alpha}||_0 < P \tag{4}$$

We use the (OMP) algorithm to recover a sparse estimate of the vector $\hat{\boldsymbol{\alpha}}$ [4,6]. The reconstructed video is computed as

$$\hat{\boldsymbol{V}} = \boldsymbol{D}\hat{\boldsymbol{\alpha}} \tag{5}$$

We perform the reconstruction for all the 8×8 patches in the image. Since the process of reconstruction is independent of each patch, we can reconstruct the video parallelly.

V. Results

A. FPGA implementation

In the hardware, we have configured our FPGA system for 13× compression ratio, but it can be extended for higher compression ratio. The proposed CS architecture is modeled by Verilog HDL and implemented in Xilinx Ultra96 Evaluation Platform (FPGA: XCZU3EG-SBVA484). The implementation results are summarized in Table I. The FPGA resources utilization summary for both the blocks are also shown in Table I. The proposed design can encode 651 frames/second of 640×480 video sequences with the maximum clock frequency of 200 MHz. The sensing matrix generation block consumes 118 LUTs and 169 registers, whereas the compressed frame generation block consumes 100 LUTs and 364 registers. The design consumes 128.5 Block RAM (each 36 Kbit) for storing compressed frame (640×480×8-bit) and sensing matrix (104×8-bit).

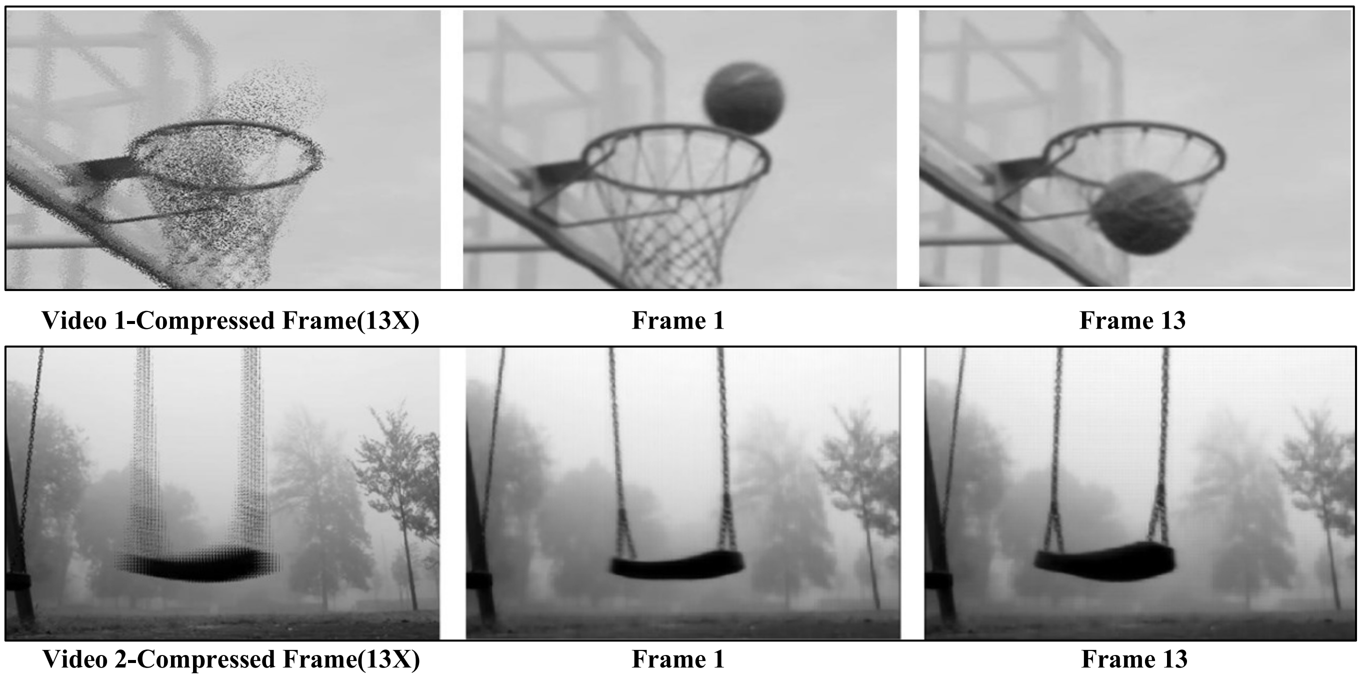

Fig. 5. **Experimental Results**: **First Column:** (13×) FPGA simulation results of the proposed design**. Second and Third Column:** Reconstructed Frame 1 and 13 respectively from a single CS image. **PSNR (video-1) =24.15, PSNR (video-2) =24.36.**

Table I: FPGA (XCZU3EG-SBVA484) implementation results of the proposed design.

Architectures	LUTs	Registers	BRAMs (36Kbit)	Clock (MHz)	Dynamic Power (mW)
Sensing Matrix Generation	118	169	0.5	200	339
Compressed Frame Generation	100	364	128		

B. Reconstruction

We verified our framework on different video scenes, two are shown in Fig. 5. Here, we have captured a single CS video frame and 13 frames have been reconstructed from a single image. The reconstructed frames 1 and 13 have been shown in second and third column of the Fig. 5. In the first video, we captured a ball thrown into the basket. Here we captured the fast-paced ball with minimal motion blur. In the second video, we were able to reconstruct the chains on the swings including the gaps in between the chains and we were able to capture the clear movement of swing with less motion blur. The PSNR values of the reconstructed videos generated using our architecture are shown in the description of Fig. 5.

VI. Conclusion

In this paper we proposed a hardware architecture to produce compressive sensed videos in a conventional camera. We implemented the design in FPGA to emulate PCE imaging on a conventional image sensor. Our implementation results show that the proposed compressive sensing framework is efficient and easily adaptable to any conventional camera unlike CMOS based sensors [3,4,5,11]. Hence, making it a perfect candidate for edge devices as the compressed video can be transmitted through low-bandwidth communication channels.

Limitations in our framework comes from the fact that compression rate need to be fixed beforehand. i.e. even for static video, we had to limit ourselves for a pre-fixed compression rate. Future work involves the development of the adaptive compression ratio framework [11], which changes the compression ratio based on the motion in the video.

References

[1] Yasunobu Hitomi, Jinwei Gu, Mohit Gupta, Tomoo Mitsunaga, and Shree K. Nayar, “Video from a single coded exposure photograph using a learned over-complete dictionary,” in *Proceedings of the 2011 International Conference on Computer Vision, ICCV ’11* pages 287–294, Washington, DC, USA, 2011. IEEE Computer Society.

[2] T. Xiong *et al.*, "Spatiotemporal compressed sensing for video compression", *IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Boston, MA, 2017, pp. 289-292, doi: 10.1109/MWSCAS.2017.8052917.

[3] Y. Oike and A. El Gamal, "CMOS Image Sensor With Per-Column ΣΔ ADC and Programmable Compressed Sensing," in *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 318-328, Jan. 2013, doi: 10.1109/JSSC.2012.2214851.

[4] Patrick Llull, Xuejun Liao, Xin Yuan, Jianbo Yang, David Kittle, Lawrence Carin, Guillermo Sapiro, and David J. Brady, "Coded aperture compressive temporal imaging," *Opt. Express* **21**, 10526-10545 (2013)

[5] Jie Zhang, Tao Xiong, Trac Tran, Sang Chin, and Ralph Etienne-Cummings. “Compact all-CMOS spatiotemporal compressive sensing video camera with pixel-wise coded exposure”. *Opt. Express,* 24(8):9013–9024, Apr 2016.

[6] M. Aharon, M. Elad, and A. Bruckstein. “K-svd: An algorithm for designing overcomplete dictionaries for sparse representation.” *IEEE Transactions on Signal Processing*, 54(11):4311–4322, Nov 2006.

[7] R. G. Baraniuk, T. Goldstein, A. C. Sankaranarayanan, C. Studer, A. Veeraraghavan, and M. B. Wakin. “Compressive video sensing: Algorithms, architectures, and applications”. *IEEE Signal Processing Magazine,* 34(1):52–66, Jan 2017.

[8] T. T. Cai and L. Wang, "Orthogonal Matching Pursuit for Sparse Signal Recovery With Noise," in IEEE Transactions on Information Theory, vol. 57, no. 7, pp. 4680-4688, July 2011.

[9] Michael D. Ciletti. 1999. “Modeling, Synthesis, and Rapid Prototyping with the Verilog HDL”. *Prentice-Hall, Inc., Upper Saddle River, NJ, USA.*

[10] E. Candes, J. Romberg, and T. Tao. “Stable signal re-covery from incomplete and inaccurate measurements”. *Communications on Pure and Applied Mathematics,* 59, 2006.

[11] J. Zhang et al., "A Closed-Loop, All-Electronic Pixel-Wise Adaptive Imaging System for High Dynamic Range Videography," in *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 67, no. 6, pp. 1803-1814, June 2020, doi: 10.1109/TCSI.2020.2973396.