A FPGA DESIGN FOR HIGH SPEED FEATURE EXTRACTION FROM A COMPRESSED MEASUREMENT STREAM

Dustin Richmond, Ryan Kastner

Computer Science and Engineering
University of California, San Diego
{drichmond, kastner}@eng.ucsd.edu

Ali Irturk, John McGarry

Cognex Corporation
San Diego, California
{Ali.Irturk, John.McGarry}@cognex.com

ABSTRACT

A common type of triangulation-based active 3D scanner outputs sets of surface coordinates, called profiles, by extracting the salient features of 2D images formed from an object illuminated by a narrow plane of light. Because a conventional 2D image must be digitized and processed for each profile, current systems do not always provide adequate speed and resolution to meet application demands. To address this challenge, a special purpose image sensor is being developed. Using Compressive Sensing, this sensor will be able to digitize compressed measurements of highly structured images, such as those formed in active 3D scanning, at a rate that would represent the conventional equivalent of 50G pixels/second. It is a significant challenge to process such a high-speed data stream at rates approaching real-time. Therefore, we present a single-chip FPGA design for the extraction of surface profiles from a compressed image stream originating from a 1024 by 768 pixel array at a rate of 14K images per second.

1. INTRODUCTION

Triangulation-based active 3D scanners, or profile scanners, are commonly found in manufacturing environments providing data for volumetric measurement, error-proofing inspection, robot guidance and statistical quality control. In addition, with the emergence of low-cost 3D printing we envision a growing demand for systems that can capture information required to duplicate existing shapes, and even for control-system feedback to 3D printers themselves. Current systems suffer from a significant performance limitation; they are relatively slow compared to widely used 2D line-scan systems. The goal of this project is to create a system for profile-scanning that is capable of operating at speeds and resolutions comparable to conventional 2D line-scan technology, typically in excess of 10K lines/second.

Profile scanners use a structured light source and an image sensor to capture information about the intersection of a physical object and a narrow plane of illumination. Each image produced by the camera is processed to extract a profile,

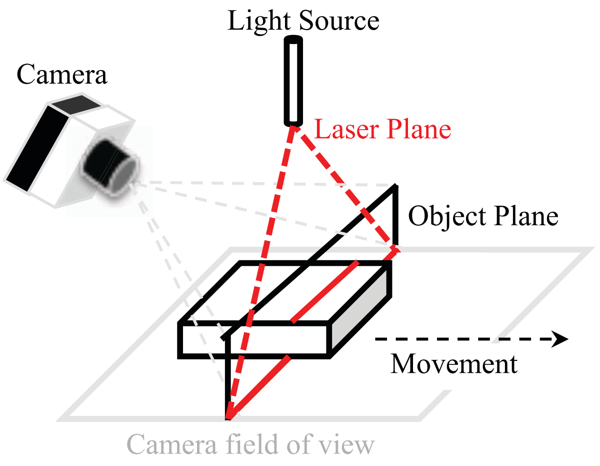

Fig. 1. A scanner system for capturing depth information of an object illuminated by a thin plane of light.

δ, a vector of elements representing the intersection of the laser line with the object in columns of the pixel array. An example scanner system is shown in Fig. 1, and an example image of the object plane is shown in Fig. 2. Fig. 3 contains a graphical representation of several consecutive profiles.

The optical system is configured so that the object plane of the camera is coincident with the plane of illumination. This is accomplished by orienting the plane of the image sensor relative to the lens in accordance with the Scheimpflug principle[1]. Associations are collected between the image and object coordinate planes of the focused optical system, and used to solve for camera calibration coefficients. Image coordinates associated with illuminated points of the scene are transformed with these camera calibration coefficients to form the measurements in the object plane. The third dimension is generated by moving the object in a direction substantially perpendicular to the plane of illumination while capturing profiles at regular intervals. Therefore, the resolution along the axis of travel is determined by the speed of the surface relative to the frame-rate of the image sensor.

978-1-4799-0004-6/13/$31.00 ©2013 IEEE

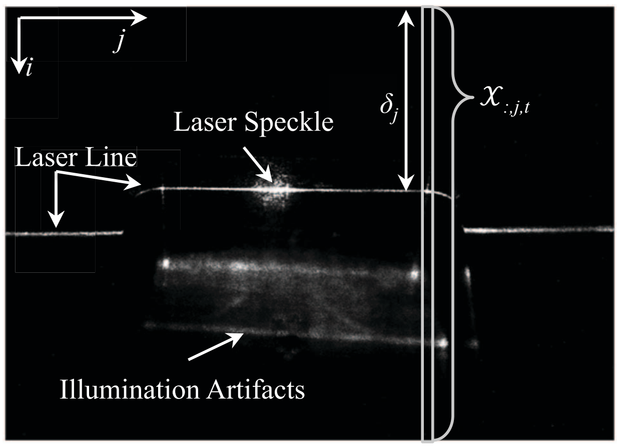

Fig. 2. Grayscale representation of an image signal $\mathcal{X}$ at time t, created by the intersection of a thin plane of illumination and a metallic object. The laser line features are formed by direct illumination of the object, and other features are the result of laser speckle or indirect illumination.

Some applications require a 3D scanning system that can accurately profile the surfaces of large fast moving objects. Such systems must capture and process large images at high speed to generate the hundreds or thousands of profiles necessary for sub-millimeter spatial resolution. CMOS sensor advances allow conventional image sensors to obtain framerates of 3K to 5K frames per second (FPS) for medium sized (512 by 512 pixels) bitmap images, with some speeds approaching 1B FPS for small (32 by 32 pixels) images [2]. However, in large image sensors, transferring the image information stored in the analog domain of the pixel array to the digital domain of the image processor presents a significant problem due to restrictions on the image sampling rate caused by analog-to-digital conversion speed and off-chip transmission bandwidth.

To address these challenges we apply signal processing principles from Compressive Sensing (CS)[3]. CS is a relatively new technique for sampling sparse signals at sub-Nyquist rates. To date, CS has been used in a variety of sparse-signal applications [4][5][6].

A recently developed image sensor combines the multi-bit analog-to-digital measurement and compression steps of the conventional data pipeline into a single stage that forms the coefficients of a compressed measurement as part of the sampling process. It applies certain principles of an extension of CS, called 1-bit CS [7] [8], to enable high frame-rate transmission of a certain class of low information content image signals for off-sensor processing. We believe that this image sensor can generate useful measurements of laser-illuminated images at a rate that represents the conventional equivalent of 50G pixels/second. Although this solves

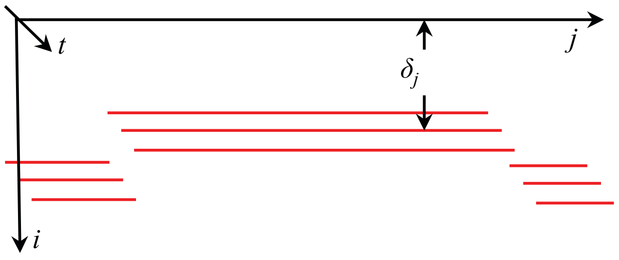

Fig. 3. Graphical representation of profile information δ extracted over three frame times.

the problem of ultra high-speed digital information capture and transmission, it creates a second, equally challenging problem, i.e. real-time processing of the output data stream to extract profile information embedded in the compressed measurements.

In this paper, we present a solution to the problem of high speed feature extraction from the measurement stream of a special purpose compressive image sensor. This work demonstrates that it is possible to exploit the principles of CS to improve the throughput of a class of image processing problems characterized by extremely low information content image streams.

Major contributions of this paper:

- A method for extracting features from a compressed measurement stream that has been formed by a special purpose compressive sensor capturing images of an object surface moving perpendicularly through a narrow plane of illumination.
- A single-chip FPGA design for the real-time extraction of surface profiles from a compressed measurement stream operating at 14K, 1024 by 768 pixel images per second.
- A working demonstration using MATLAB, a Xilinx VC 707 evaluation board, and a PCI Express 2.0 link.

We begin with a brief overview of CS as it relates to our work in section 2.1, and follow with a description of our optimizations in section 2.2. These optimizations adapt CS using specific knowledge of the 3D profile scanning application domain. Section 3 describes our FPGA architecture, and section 4 reports the area, timing, and throughput results for this architecture. Finally, we conclude in section 5.

1.1. Notation

Throughout this paper, 2-dimensional arrays of matrices are denoted by bold uppercase letters (e.g. $\mathbf{\Phi}$), 1 dimensional arrays of matrices are denoted by script (e.g. $\mathcal{X}$). Uppercase letters are use to represent a single matrix (e.g. Φ), and corresponding lowercase letters (e.g. ϕ and x) may be used to represent one vector. We use MATLAB style notation to refer to rows, columns, and various two dimensional slices of higher dimensional arrays (e.g. $\mathbf{\Phi}_{:,:,h,k}$ is the (h, k) matrix element of $\mathbf{\Phi}$). Unless otherwise defined, italicized alphabetic characters denote scalars (e.g. M and N_1).

2. COMPRESSIVE SENSING

2.1. Background

CS is a relatively new signal acquisition and compression technique that has been the subject of much research interest since it was initially described in the early 2000's [3]. In general, CS provides a mathematical framework for capturing sparse or compressible signals at a sampling rate lower than the Nyquist sampling rate. A complete introduction to CS and its extension, 1-bit CS, is beyond the scope of this paper. Instead we refer the reader to one of the many papers that provide an introduction to this field of research [3, 7, 9, 10]. However, in the interest of making subsequent discussion somewhat self contained we provide some background that relates directly to the present discussion.

First, let the image signal captured by a column of our image sensor be represented by $x \in \mathbb{R}^N$ and let the digital measurement of the signal be represented by $y \in \{-1, 1\}^M$. In addition, suppose we know that x is K-sparse, i.e. it only has K nonzero values or it can be adequately represented by K coefficients in some linear basis. Now, let x_1 and x_2 represent any of two signal vectors $x_1 \neq x_2$ and let $y_1 = sign(\Phi x_1)$ and $y_2 = sign(\Phi x_2)$ be their respective measurements using the measurement matrix Φ. Then, $y = sign(\Phi x)$, is a Binary ϵ-Stable Embedding of order K for the K-sparse signal x if the normalized vector angle between any two signals is equal to the normalized Hamming distance between their measurements, within some tolerance ϵ [7]. This relationship can be expressed as follows.

$$d_{ang}(x_1, x_2) - \epsilon \leq d_{ham}(y_1, y_2) \leq d_{ang}(x_1, x_2) + \epsilon \quad (1)$$

In 1-bit compressive sensing it has been shown that if Φ consists of IID (Independent and Identically Distributed) random variables drawn from certain known random distributions and $\epsilon > 0$, then $y = sign(\Phi x)$ is a Binary ϵ-Stable Embedding with probability $P_r > 1 - \rho$ for:

$$M \geq \frac{2}{\epsilon^2}(Klog(N) + 2Klog(\frac{50}{\epsilon}) + log(\frac{2}{\rho})) \qquad (2)$$

Assuming constant error tolerance, Eq. 2 implies that it is the information content of the signal, as represented by the sparseness K, not the signal dimension N, that is the dominate factor determining M, the number of bits necessary to encode the signal x in the measurement y.

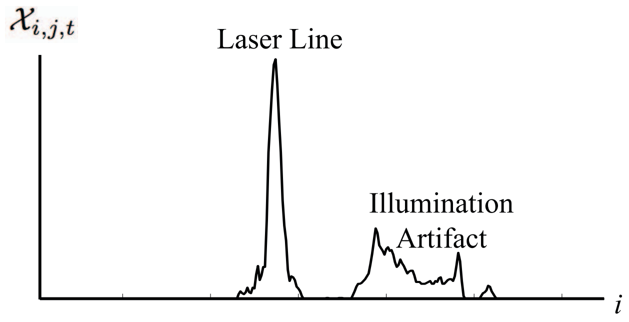

Fig. 4. Graph of the $\mathcal{X}_{:,j,t}$ column from Fig. 2

Methods to recover the signal x from the compressed measurement y depend on the a priori knowledge that the signal of interest is sufficiently sparse and uniquely represented in the measurement vector, i.e. there is a Binary ϵ-Stable Embedding. General signal recovery methods have been widely researched [8, 11, 12, 13]. These methods, typically, involve a search for a maximally sparse signal $\hat{x}$ that satisfies $\|\Phi\hat{x} - y\|_2 \leq \tau$, where τ is the noise tolerance. Iterative signal recovery methods are not practical to implement for our application, given the overall dimension of signal space and our throughput objectives. It may also be noted that complete signal recovery is not possible because the $sign(.)$ function discards the scale information, in effect constraining any reconstruction of x to a spherical sub-manifold in $\mathbb{R}^N$.

Fortunately, complete recovery of the original image signal $\mathcal{X}$ is not necessary or even desirable in our application since it decreases the maximum throughput of our system. We observe that, given certain application specific optimizations, we can model the signal of interest in such a way to make it nearly $K = 1$ block-sparse. This allows a computationally efficient search for signal model parameters representing a filtered signal $\hat{Z}$ that satisfies the test of consistent reconstruction.

2.2. Application Specific Optimizations

As shown in Fig. 2, the image of a scene illuminated by a laser line projector ideally consists of a set of thick bright line segments superimposed on a predominately black background. The thickness of these line segments are not always uniform; as the scanned surface changes its optical properties the apparent thickness of the laser line can vary by several pixels. Therefore, the center of the laser line is the best estimate of the location of the object-illumination intersection since it is invariant to surface changes. However, in practice, finding the center of the laser line from the maximum pixel intensity is problematic in the presence of typical nonlinear effects such as laser speckle, which can shift the maxima by several pixels. Moreover, laser light reflected,

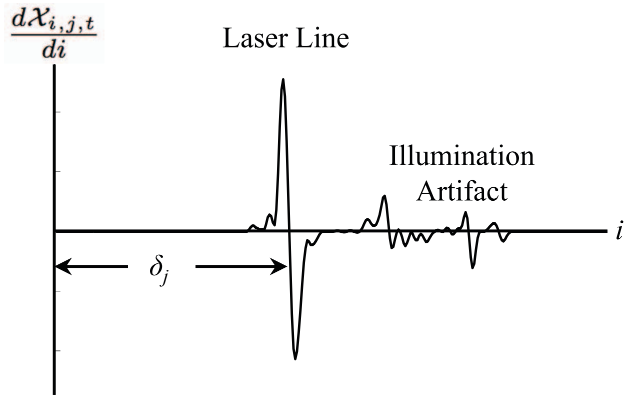

Fig. 5. Graph of the first derivative of column $\mathcal{X}_{:,j,t}$ in Fig. 4. δ_j is measured from the midpoint between minima and maxima in the image.

scattered, and/or diffused by the surfaces in the scene may interfere with line detection by illuminating other parts of the object surface that are not in the plane of the laser but are in the field-of-view of the camera. Features of this type are called illumination artifacts. Fig 2 and Fig. 4 show typical characteristics of illumination artifacts.

To mitigate the effect of noise and illumination artifacts, we modify the frequency response of the measurement matrix to attenuate spatial frequencies associated with sensor noise and illumination artifacts. Therefore, we form measurement matrix Φ by multiplying a spatial filtering matrix Ψ by a random basis matrix Θ. In practice the Ψ matrix may be derived from a convolution kernel having finite support. We construct the random basis matrix Θ and the spatial filtering matrix Ψ in a way that guarantees all elements of $\Phi = \Psi\Theta$, belong to the set $\{-1, 0, 1\}$, which is consistent with the computational capabilities of our image sensor. We use a spatial filtering matrix that approximates the first derivative with respect to rows. By forming the measurement matrix as the product of the random basis Θ and the spatial filter Ψ, the image sensor encodes the first derivative of the columns of the image, like shown in Fig. 5.

The advantages of this methodology are threefold. First, filtering attenuates the influence of illumination artifacts and noise. Second, the first derivative of the image signal is typically substantially sparser than the original image signal, allowing the useful information of the laser line to be encoded in fewer bits. These characteristics are shown in the difference between Fig 4 and Fig. 5. Finally, the center of the laser line, and thus the position of the plane of illumination is better approximated from the coordinates of the top and bottom edge of the bright line segment, as represented by the maxima and minima in Fig. 5.

It is also advantageous to perform filtering in the other two dimensions, i.e. in the horizontal, or j dimension of the image, and in the temporal dimension of the data stream. Consider that each point on the scanned surface is associated with a unique image column signal $\mathcal{X}_{:,j,t}$, where j is the image column index at time t in the sequence. Since object surfaces are generally piecewise smooth and continuous over small 3×3 neighborhoods, all 9 image signal columns associated with the neighborhood are, with high probability, closely correlated with each other. Given this, it is clear that local neighborhood averaging, which represents a low-pass filter, would be advantageous, functioning to attenuate the high spatial frequencies present in image sensor noise while leaving the signal information associated with object surface coordinates substantially unchanged. Moreover, there is a substantial computational advantage to be gained through compressed measurement of the neighborhood average.

If 3×3 neighborhood averaging is performed in image signal space, i.e. on the sensor, then measurement of the 3×3 average signals can be stated as:

$$\mathcal{Y}_{:,j,t} = sign(\sum_{k=1}^{3}\sum_{h=1}^{3} \Phi\ \mathcal{X}_{:,(j+h-2),(t+k-2)}) \tag{3}$$

Although theoretically possible, it is generally infeasible to perform the spatiotemporal image averaging calculation of Eq. (3) in the analog domain of an image sensor, since it would involve complicated analog signal storage an processing means that complicate the design of the pixel circuit, thereby creating an undesirable tradeoff with respect to image sensor resolution and pixel light sensitivity. However, if we decompose Φ by rows into a 3×3 array, $\boldsymbol{A}$, of 9 independent terms from Φ, each with m non-zero rows, where $m = \frac{M}{9}$ and $\boldsymbol{A} \in \{-1, 0, 1\}^{M \times N_1 \times 3 \times 3}$:

$$\begin{aligned}
\boldsymbol{A}_{:,:,1,1} &= \begin{pmatrix} {\Phi_{0m+1:1m,:}}^T & 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix}^T \\
\boldsymbol{A}_{:,:,2,1} &= \begin{pmatrix} 0 & {\Phi_{1m+1:2m,:}}^T & 0 & 0 & 0 & \dots & 0 \end{pmatrix}^T \\
\boldsymbol{A}_{:,:,3,1} &= \begin{pmatrix} 0 & 0 & {\Phi_{2m+1:3m,:}}^T & 0 & 0 & \dots & 0 \end{pmatrix}^T \\
&\vdots \\
\boldsymbol{A}_{:,:,3,3} &= \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 & {\Phi_{8m+1:9m,:}}^T \end{pmatrix}^T
\end{aligned}$$

We observe that the measurement of the average signal can be approximated by the sum of the signs of partial measurements performed on the spatiotemporal neighborhood under certain conditions, as stated below:

$$\mathcal{Y}_{:,j,t} \approx \sum_{k=1}^{3}\sum_{h=1}^{3} sign(\boldsymbol{A}_{:,:,h,k}\ \mathcal{X}_{:,(j+h-2),(t+k-2)}) \tag{4}$$

In words, given a sufficient number of samples M, the summation of signs of partial measurements over a neighborhood is approximately the same as the signs of measurement of the neighborhood average.

In our image sensor we dispense with the needless multiplication of empty row vectors in Eq. (4) by folding Φ into a

3×3 array of matrices that replaces $\boldsymbol{A}$. This forms the four-dimensional array $\boldsymbol{\Phi} = fold(\Phi) \in \{-1, 0, 1\}^{m \times N_1 \times 3 \times 3}$.

On the sensor, the pixel array is divided into three interleaved column control sets, allowing three different sampling matrices from $\boldsymbol{\Phi}$ to be applied simultaneously to the image signal. Additionally, the image sampling matrices are changed as a function of the frame time t. In effect, a total of nine separate sampling matrices are used over three frame times to form interleaved partial measurement vectors $\mathcal{Y}^I$ according to:

$$\mathcal{Y}^I_{:,j,t} = sign(\boldsymbol{\Phi}_{:,:,(j \bmod 3+1),(t \bmod 3+1)}\ \mathcal{X}_{:,j,t}) \quad (5)$$

The output of our image sensor, as defined in Eq. (5), consists of one m bit binary vector for each column of the image on every frame time t.

As in other CS work [8], the first approximation of the filtered image column vector $\hat{Z}_{:,j}$ can be formed from the product of the transpose of the random basis Θ and the concatenation of measurement vectors over the 3×3 spatiotemporal neighborhood, as in Eq. (6) below.

$$\hat{Z}_{:,j} = \Theta^T \begin{pmatrix} \mathcal{Y}^I_{:,j-1,t-1} \\ \mathcal{Y}^I_{:,j+0,t-1} \\ \mathcal{Y}^I_{:,j+1,t-1} \\ \vdots \\ \mathcal{Y}^I_{:,j+1,t+1} \end{pmatrix} \quad (6)$$

However, if we fold the $M \times N$ matrix Θ by rows to form $\boldsymbol{\Theta} = fold(\Theta) \in \{-1, 0, 1\}^{m \times N_1 \times 3 \times 3}$ a 3×3 array of $m \times N_1$ matrices,

Then Eq. (6) can be restated as follows:

$$\hat{Z}_{:,j} = \sum_{k=1}^{3} \sum_{h=1}^{3} \boldsymbol{\Theta}_{:,:,h,k}{}^T\ \mathcal{Y}^I_{:,(j+h-2),(t+k-2)} \quad (7)$$

As we will show in the following hardware description, the first summation operation of Eq. (7) can be efficiently performed as a neighborhood-averaging operation on partial sums of $\hat{Z}_{:,j}$. In practice we have observed that, using a surprisingly small number of samples, it is possible to capture a digital approximation of a filtered image signal of sufficient quality to be useful for purpose of estimating the signal model parameters of interest, i.e. the offset coordinates of the illumination plane as it appears in our image.

3. FPGA ARCHITECTURE

This section describes the architecture for our FPGA system. Our initial design space exploration was conducted using Vivado HLS. As the design developed, we switched to Verilog for the final design and demo. For our demonstration the current system uses a Xilinx VC707 Evaluation board with a XC7VX485T-2 chip and MATLAB performs the role of the camera. We intend to move this FPGA system onto the camera in the near future.

In our architecture, we make one final optimization that affects $\boldsymbol{\Theta}$. To simplify our logic, we replace the 0's in $\boldsymbol{\Theta}$ with values chosen randomly from $\{-1, 1\}$. This reduces the multiplication in Eq. (7) to Hamming distance calculations performed between binary vectors.

3.1. Input and Output Interface

As described above, our current system uses MATLAB to generate compressed signal batches for processing on the FPGA. These images are transmitted over the PCI Express (PCIe) bus using PCIe hardware available on the VC707 development board and the Reusable Integration Framework for FPGA Accelerators (RIFFA 2.0) [14].

In our demonstration, a set of laser-line images in MATLAB are used to produce a batch of compressed images. These images are exported out of MATLAB by the MEX API and transmitted to the FPGA using RIFFA. In this setup, MATLAB replaces the camera in Fig. 6. In hardware, the RIFFA core provides this data to our FPGA design *Input Interface*. The *Input Interface* loads data into the *Image Buffer* at the head of the FPGA image processing pipeline in Fig. 6.

The *Output Interface* stage transfers a finished profile, δ, to the same RIFFA core. The RIFFA core transmits the profile back to the PC via PCIe bus. Finally, the profile is returned to MATLAB for analysis.

3.2. Memory Read

The *Memory Read* stage is the third stage of the FPGA pipeline. This stage is responsible for producing read addresses for the *Image Buffer* and passing the read data to the *Reconstruction Stage*. The *Memory Read* stage is also responsible for signaling the end of a row, which resets the *Reconstruction Stage* and the end of a frame which generates a single profile, δ, at the *Output Interface*.

3.3. Reconstruction Stage

The *Reconstruction Stage* reconstructs individual rows in $\hat{Z}$ from a set of three consecutive measurements, $\mathcal{Y}^I_{:,:,t-1:t+1}$. This stage is the core of our architecture, and where many of our CS optimizations were targeted. The bulk of our Vivado HLS design exploration focused on this unit and how to efficiently accelerate image decompression and processing, while minimizing routing overhead and area.

Our first design decision was to reconstruct all 1024 pixels in a row simultaneously. This approach has numerous advantages: First, it matches the format of the data from the scanner. Second, this approach minimizes control logic;

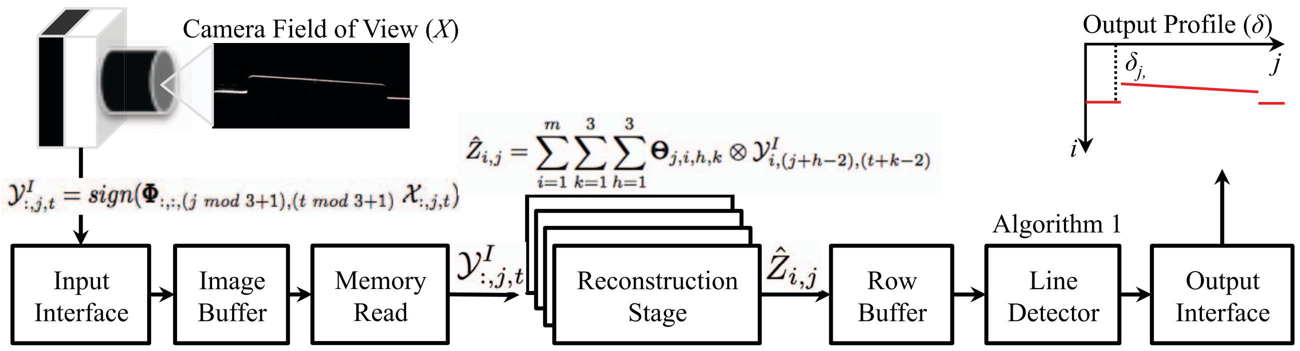
Camera Field of View (X)
Output Profile (δ)
δ_j
j
i
$\hat{Z}_{i,j} = \sum_{i=1}^{m}\sum_{k=1}^{3}\sum_{h=1}^{3} \boldsymbol{\Theta}_{j,i,h,k} \otimes \mathcal{Y}^I_{i,(j+h-2),(t+k-2)}$
$\mathcal{Y}^I_{:,j,t} = sign(\boldsymbol{\Phi}_{:,:,(j \bmod 3+1),(t \bmod 3+1)} \mathcal{X}_{:,j,t})$
Algorithm 1
Input Interface
Image Buffer
Memory Read
$\mathcal{Y}^I_{:,j,t}$
Reconstruction Stage
$\hat{Z}_{i,j}$
Row Buffer
Line Detector
Output Interface

Fig. 6. Overview of the image sensor and FPGA pipeline for decompressing and processing the sensor output.

the logic for reconstructing each pixel has hardwired inputs from the image buffer and it's respective $\boldsymbol{\Theta}_{:,:,h,k}$ matrix. Vivado HLS demonstrated that producing less than a full row of data introduces far too much overhead, to the point where the added complexity of 3×3 neighborhood convolution becomes unwieldy because additional logic is added to route data from the correct $\boldsymbol{\Theta}_{:,:,h,k}$ partition. These pitfalls are avoided by reconstructing all 1024 pixels from a row in $\hat{Z}$ in parallel, and hard-coding connections to the $\boldsymbol{\Theta}$ and $\mathcal{Y}^I$ memories to minimize overhead.

We begin with the hardware that produces each pixel. Eq. (8) displays the summations necessary to calculate pixel $\hat{Z}_{i,j}$ at row i, column j in $\hat{Z}$.

$$\hat{Z}_{i,j} = \sum_{k=1}^{3}\sum_{h=1}^{3}\sum_{i=1}^{m} \boldsymbol{\Theta}_{j,i,h,k} \otimes \mathcal{Y}^I_{i,(j+h-2),(t+k-2)} \tag{8}$$

In words, the pixel $\hat{Z}_{i,j}$ is equal to the sum of the hamming distances between each of the three $\mathcal{Y}^I_{:,j,t-1:t+1}$ columns in 3 consecutive frame times, and their corresponding measurement column in $\boldsymbol{\Theta}_{i,:,h,k}$. Assuming each XOR-add in the i summation is completed every cycle, in naive hardware this summation will take $3 \times 3 \times m$ cycles.

A simple acceleration approach is to save two previously reconstructed partial sums on DRAM or on-chip BRAM. This reduces the latency by a factor of 3 by eliminating the iterations dedicated to the k summation. However, at 8 bits per pixel and a target rate of 50 Gpixels/second, the rate at which pixels need to be read and stored substantially exceeds DRAM bandwidth. Alternatively, storing the data on BRAM at 8 bits per pixel a 768 row by 1024 column image takes 260 BRAMs per saved image. Saving two previously reconstructed partial sums requires 520 of the total 1030 of 36 Kbit BRAMs available on the chip. While this BRAM-based approach solves the aforementioned bandwidth problem, it clearly does not scale well; routing congestion caused by loading large frames reduces the clock speed, and increasing the image size is nearly impossible. Furthermore, BRAMs for other modules, such as the *RIFFA Interace*, and the *Line Detector* also contend for BRAM resources. Alone, these modules consume a significant fraction of the design's power, and doubling or tripling the power consumption by saving previous frames is undesirable.

Instead, we reconstruct three consecutive images simultaneously from compressed data saved in the *Image Buffer* and combine the images to produce the output $\hat{Z}$. In contrast to saving the previously reconstructed partial sums, this approach only consumes 14 BRAMs per image buffer. The image buffer is replicated 3 times so consecutive images can be accessed simultaneously.

One way to implement this simultaneous reconstruction is to calculate each sub-pixel in the 3×3 neighborhood, and then add the sub-pixels to produce the final output pixel. This is implied in the order of summations in Eq. (8). Implemented correctly, this approach reduces the number of computations by sharing the sub-pixels of overlapping neighborhoods. However, we found that this 9-input add was inefficient in terms of routing resources and chip area because it requires 1024 additional adders. Instead, we calculate the hamming distance of the 3×3 neighborhood before performing the summation over index i. This change is represented by moving the index of summation i to the outermost Σ, and is shown in Eq. (9):

$$\hat{Z}_{i,j} = \sum_{i=1}^{m}\sum_{k=1}^{3}\sum_{h=1}^{3} \boldsymbol{\Theta}_{j,i,h,k} \otimes \mathcal{Y}^I_{i,(j+h-2),(t+k-2)} \tag{9}$$

In hardware, the Hamming distance of a 3×3 neighborhood is equivalent to unrolling the summations of k and h. This is performed in a pipeline, with an initiation interval of 1. The summation of h for a particular $k = k_0 \in \{1, 2, 3\}$ in Eq. 10, is implemented in two 6-input look-up tables (LUT).

$$\sum_{h=1}^{3} \boldsymbol{\Theta}_{j,i,h,k_0} \otimes \mathcal{Y}^I_{i,(j+h-2),(t+k_0-2)} \tag{10}$$

The subsequent summation of k adds together the per-frame Hamming distances. The k summation is implemented us-

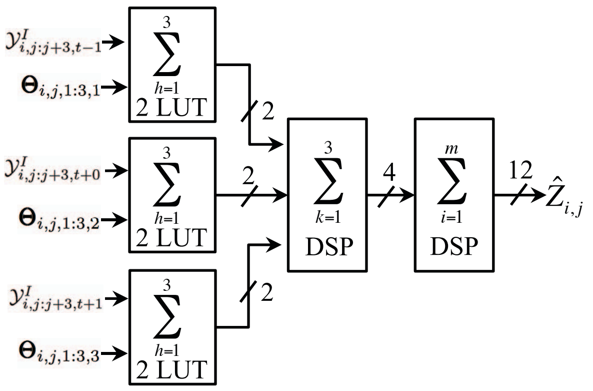

Fig. 7. Pixel unit implementing Eq. (9). The summation of h is implemented in two 6-input LUTs. The subsequent k and i stages are implemented in DSP48E1 chains.

ing a DSP48E1 slice and summation of i which is also implemented using a DSP48E1. Each of the aforementioned DSP's is configured in 4 SIMD fashion, executing 4 independent 12-bit additions so that each DSP calculates 4 independent output pixels. The connection between the k summation and i summation is made using optimized DSP-to-DSP routing logic. This hardware is called the *pixel unit*, and is shown in Fig. 7. This approach reduces the latency of a single pixel by a factor of 9, to m cycles.

Finally, we further accelerate the *Reconstruction Stage* by simultaneously reconstructing 4 rows in $\hat{Z}$. This replication is shown in Fig. 6. The reconstruction stage is followed by the row buffer stage. This stage is responsible for taking 4 rows of data from the multiplier and feeding it one row at a time into the line detector. It is implemented as a parallel-load/serial-out shift register.

3.4. Line Detector

Extracting the profile δ for the current reconstructed image uses 1D column-wise line detection, repeated for all image columns to find individual δ_j values. As stated in section 2.2, the column $\hat{Z}_{:,j}$ is the reconstructed first derivative of $\mathcal{X}_{:,j,t}$ by the construction of the Φ. An ideal first derivative column is in Fig. 5.

A common approach for line detection is using filters. We have found that filters are adequate, but particularly noise prone and cannot adapt to changes in line-width. Each filter generates many candidate locations near the center of the minima/maxima in Fig. 5. In the worst case, none of these candidates can be clearly marked as center of the line in the column. This problem is further compounded as the line width changes in response to surface changes. To address this, we have implemented a width-invariant line detector, described in Algorithm 1. The value of W in the algorithm is set by the operator, and logically represents the maximum distance between any minima and maxima pair. This algorithm operates on a column $\hat{Z}_{:,j}$ with output δ_j.

Algorithm 1: Width invariant line detector

```
Input: $\hat{Z}_{:,j}$, buffer[W] initialized to 0
1:  for all $row$ in $\hat{Z}_{:,j}$ do
2:    $buffer[row\%W]$ = $\hat{Z}_{row,j}$
3:    $current\_pixel = buffer[W/2]$
4:    if $current\_pixel == max(buffer)$ then
5:      $max\_pixel = current\_pixel$
6:      $max\_pixel\_row = row$
7:    else if $current\_pixel == min(buffer)$ then
8:      $current\_\Delta = max\_pixel - current\_pixel$
9:      if $(current\_\Delta \leq max\_\Delta)$ then
10:       $max\_\delta_j = current\_\Delta$
11:       $\delta_j = row + max\_row$
12:     end if
13:   end if
14: end for
```

This algorithm is designed to recognize the maxima followed by minima characteristic created by taking the first derivative. The core of this algorithm is the if statement on lines 4-7. These lines detect a local maxima within W pixels, inhibiting any false maxima. The if statement on line 7 captures the subsequent minima. If the maxima and minima occur within W pixels, this detector will find and report that result.

This algorithm has several benefits. First and foremost it is invariant with respect to the width of the line. If the width of the line, and thus relative location of the minima and maxima are changed by the characteristics of the surface, this filter will adapt because it does not depend on a fixed number of taps. Instead, the value of W can be set such that the minima value will always be reached before the maxima value leaves the buffer. Since it is extremely unlikely that another maxima will occur between the maxima and minima, this detector will resolve the center of the line with high probability. Second, this algorithm has the potential for higher resolution because it detects the minima and maxima and returns the sum of the two rows. Dividing the sum gives the center of the line with half-pixel resolution.

4. RESULTS

Our architecture was designed and implemented using Xilinx's Vivado IDE, version 2012.4, with all synthesis and place & route settings set to maximum effort. The utilization results are shown in Table 1 while, timing, and throughput results from this design are contained in Table 2. These results are reported from the auto-generated, post-place and route results of the Vivado suite.

Table 1. Resource Utilization

Resource	Quantity Used	Percent Utilization
Slice	46836	61.70 %
LUT	110048	36.24 %
Register	154971	51.04 %
BRAM	272	26.40 %
DSP	2048	73.14 %

Table 2. Timing & Throughput Results

Clock Period	5.75 ns
Clock Frequency	174 MHz
Profile Rate	14.15K Profiles per second

Our current design reconstructs an image of dimensions 768 rows $\times$ 1024 columns and extracts the features of the laser line to produce a 1024 measurement array, called a profile. The parameter M for our architecture is equal to 576, and $m = \frac{M}{9} = 64$. Since all 9 pixels in a 3×3 neighborhood are reconstructed simultaneously, the output interval of the *Reconstruction Stage* is equal to the bounds of the summation of i in Eq. 9, or m cycles. On every output 4 rows in $\hat{Z}$ are produced. Together this corresponds to an average output interval of $\frac{m}{4}$ cycles, or 16 cycles per output row. For this set of parameters, the *Reconstruction Stage* completes a single $\hat{Z}$ frame reconstruction every $768 \times \frac{m}{4}$ cycles, or 12288 cycles per frame.

Finally, the input interval of the *Line Detector* stage is equal to W, which has been set to 16. This matches the output interval of the *Reconstruction Stage*, so the output rate of the entire core is equal to one profile every 12288 cycles. The *Profile Rate* reported in Table 2 reflects this result.

5. CONCLUSION

We have presented a design enabling a single-chip FPGA device to operate in cooperation with a special purpose compressive sensor to provide a system for extracting image features from a measurement stream at a rate of 14K images per second. The system is intended to address the digital-image capture-rate limitation associated with a common class of triangulation-based active 3D scanners. More generally, this research demonstrates that, in certain application domains, image feature extraction can be performed on the measurements stream of a compressive sensor to realize a practical throughput advantage, relative to systems employing conventional methods of digital-image capture.

6. REFERENCES

[1] A. Legarda, A. Izaguirre, N. Arana, and A. Iturrospe, “A new method for scheimpflug camera calibration,” in *ECMS 2011*, 2011, pp. 1–5.

[2] M. El-Desouki, M. Jamal Deen, Q. Fang, L. Liu, F. Tse, and D. Armstrong, “Cmos image sensors for high speed applications,” *Sensors*, vol. 9, no. 1, pp. 430–444, 2009.

[3] D. L. Donoho, “Compressed sensing,” *IEEE Trans. Inform. Theory*, vol. 52, pp. 1289–1306, 2006.

[4] M. Lustig, D. Donoho, and J. M. Pauly, “Sparse mri: The application of compressed sensing for rapid mr imaging,” *Magnetic Resonance in Medicine*, vol. 58, no. 6, pp. 1182–1195, 2007.

[5] J. Chen, J. Cong, M. Yan, and Y. Zou, “Fpga-accelerated 3d reconstruction using compressive sensing,” in *ACM/SIGDA FPGA*, ser. FPGA ’12. New York, NY, USA: ACM, 2012, pp. 163–166.

[6] D. Wu, W.-P. Zhu, and M. Swamy, “A compressive sensing method for noise reduction of speech and audio signals,” in *MWSCAS 2011*, Aug., pp. 1–4.

[7] L. Jacques, J. N. Laska, P. Boufounos, and R. G. Baraniuk, “Robust 1-bit compressive sensing via binary stable embeddings of sparse vectors,” *CoRR*, vol. abs/1104.3160, 2011.

[8] P. Boufounos and R. Baraniuk, “1-bit compressive sensing,” in *Information Sciences and Systems, 2008. CISS 2008. 42nd Annual Conference on*, march 2008, pp. 16 –21.

[9] E. J. Candes, “The restricted isometry property and its implications for compressed sensing,” *Comptes Rendus Mathematique*, vol. 346, no. 910, pp. 589 – 592, 2008.

[10] R. Baraniuk, M. Davenport, R. Devore, and M. Wakin, “A simple proof of the restricted isometry property for random matrices,” *Constr. Approx*, vol. 2008, 2007.

[11] J. Stanislaus and T. Mohsenin, “High performance compressive sensing reconstruction hardware with qrd process,” in *ISCAS 2012*, may 2012, pp. 29 –32.

[12] V. Cevher, , M. F. Duarte, C. Hegde, and R. G. Baraniuk, “Sparse signal recovery using markov random fields,” in *NIPS*, Vancouver, B.C., Canada, 8–11 December 2008.

[13] P. Maechler, “Vlsi design of approximate message passing for signal restoration and compressive sensing,” *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 2, no. 3, pp. 579 –590, sept. 2012.

[14] M. Jacobsen, Y. Freund, and R. Kastner, “Riffa: A reusable integration framework for fpga accelerators,” in *21st IEEE FCCM*, ser. FCCM ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 216–219.