Hardware Decompression for Compressive Sensing of Images

Frank Zovko
Keith Dronson
Samuel Subbarao
Federico Garcia

A. Introduction:

Most conventional approaches to sampling a signal are based on Shannon's sampling theorem: the sampling rate should be twice the maximum frequency in the signal (a.k.a. Nyquist rate). When it comes to pictures, which are not bandlimited, the sampling rate is determined by the desired resolution of the picture. Compressive sensing (CS) provides a way to recover an image from far fewer samples than would normally be necessary. CS relies on two basic principles: sparsity and incoherence. Sparsity is the idea that the bandwidth of a signal may be larger the actual number of "information" samples. This leads to the fact that if these samples were represented in the right basis ψ, they would be less sparse (more compressed). Incoherence extends the duality between time and frequency: something that is compressed in ψ will be spread out in the domain that it was acquired in.

The typical approach to sensing is the following:

$y_k = \langle f, \varphi_k \rangle$

where f is the image to be sampled, φ_k is the sensing waveform, and y_k is the sampled data. If the φ_k's are indicator functions of pixels, then the y_k's are the typical image data collected from a camera. The complexity arises from the number of dimensions of y, which we'll call n. One could try to take n measurements (more pixels in a CCD) or one could be clever and find a solution that allows him to undersample, collecting m samples instead of n ($m << n$). In that case, one could create an m x n sensing matrix, A, composed of m rows of the φ_k's: φ_1^*, φ_2^*, ... , φ_m^* (where a* denotes the complex transpose of a). Since f is n-dimensional, but y is of dimension m and y = Af, there are an infinite number of possibilities for f. However, in some cases, there is a way out of this.

SPARSITY:
If f was an element of R^n and sampled in an n dimensional basis φ_1, φ_2, ... , φ_n, then we have the following relationship:

$$f = \sum_{i=1}^{n} x_i \varphi_i$$

However if some of those x_i's are small there may be a subset of the φ_i's that almost add up to f. In that case:

$$f = \sum_{i=1}^{s} x_i \varphi_i$$

or
$f = \Phi * x_s$ where Φ is an n x n matrix of φ_1 - φ_n as columns, and x_s are the s largest coefficients of the x_i's. The figure below shows how this works, and can be quite good at reconstructing the image.

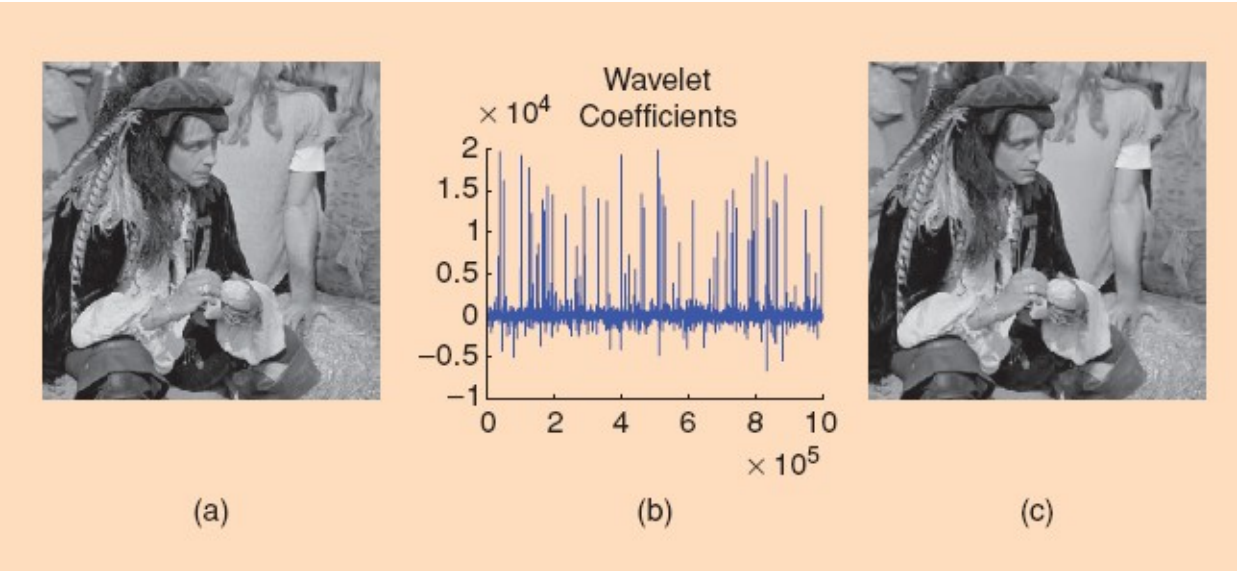

Fig. 1

Part a of Fig. 1 [1] shows the initial image. Part b is the image in the φ basis. Note that there are only a few discrete φ_i's that have x_i's with large coefficients. Part c is the reconstruction of the image using the φ_i's linked to the largest 25,000 coefficients. This means that 97.5% of the sampled data was thrown away and the picture still looks quite good.

INCOHERENCE:
Since f is an element of R^n, we can find two basis sets Φ and Ψ for the space. Φ will represent f as shown before, and Ψ will be used as the sensing basis. The coherence measures the largest correlation between any two elements of Φ and Ψ. Compressive sensing looks for low coherence pairs (maximum incoherence). Since Φ will be some fixed basis, it has been shown that the best basis for Ψ is a random basis (white Gaussian noise).

As mentioned before, y can be sensed in the Ψ basis: $y = \Psi f$ or $y_k = \langle f, \psi_k \rangle$ (dot product of f with each basis vector in Ψ). In order to recover the image we look at the following:

$y_k = \langle \varphi_k, \Psi^* f \rangle$, where f is the signal to be recovered

Solving this equation for f is impossible. However, we believe (or know) that f is sparse. In that case we can look for the following signal that will solve the minimization problem:

$\min \{||x||_0 , \Psi^* x = y\}$

Essentially here we are looking for an x with the least number of non-zero coefficients that will satisfy $\Psi * x = y$. This too is intractable. So we will use:

$\min \{||x||_1 , \Psi * x = y\}$

This can be done in a reasonable amount of time. Finally $f_{rec} = \Phi * x$.

L1 minimization algorithms are not the only way to recover compressively sensed data. There are greedy algorithms available that allow one to do is as well.

B. Overall Architecture:

The goal of this project is to implement the decompression side of a CS system on the Altera Cyclone II FPGA board. The CPU on the board will have a C program allowing it to get a y-dimensional compressed image from the computer. This image will then be decompressed and displayed on the VGA display. For our application we are going to use a 200x200 pixel resulting image (n = (3 colors per pixel) * (200 pixels wide) * (200 pixels high) * (8 bits per color)= 120,000 bytes). We are expecting the compressed image to be around 3000 bytes (40:1 compression). Both the compressed and decompressed images will be stored in the SRAM (which has a total of 512 kB available).

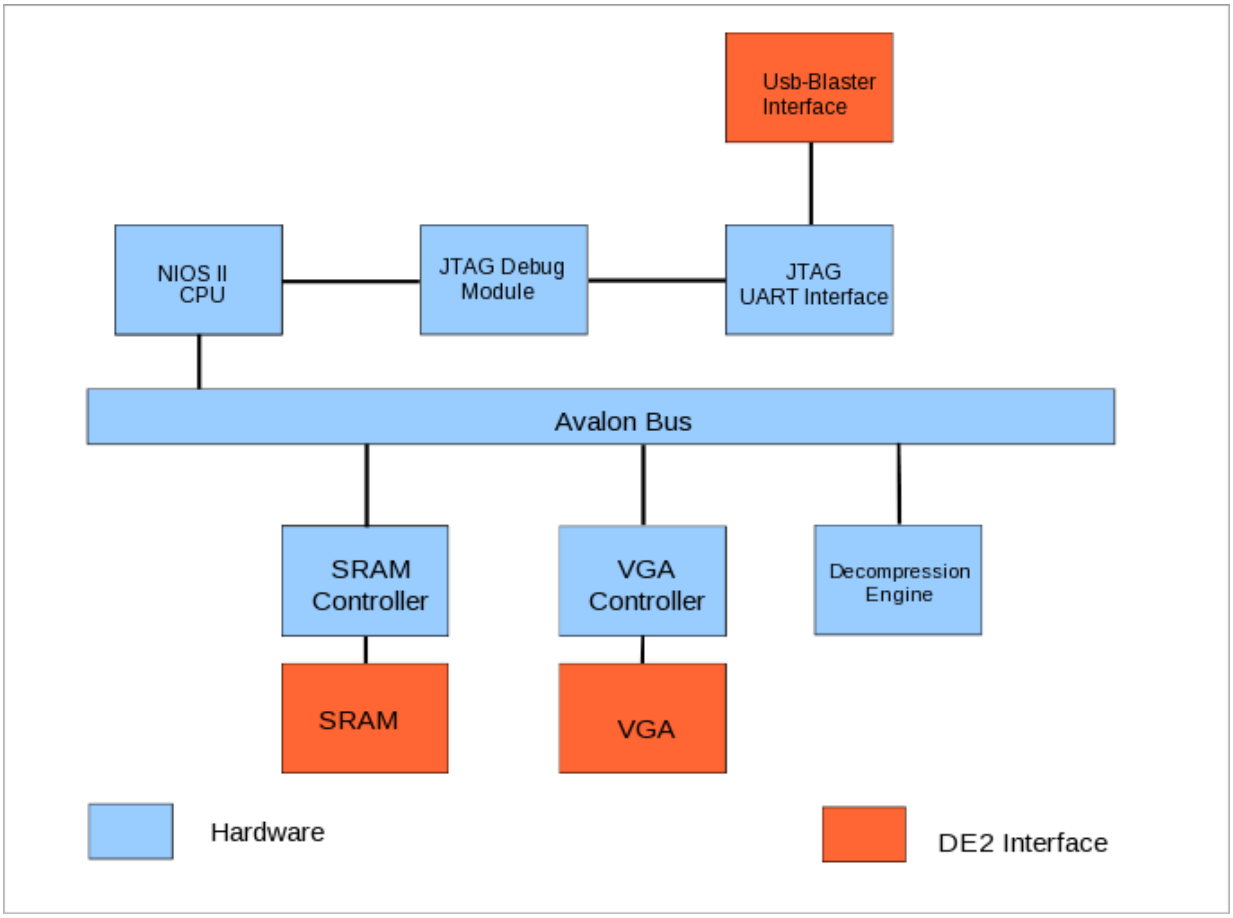

Fig. 2: Overall Hardware Design

C. Hardware Design:

1. VGA Controller

The VGA Controller will communicate with the VGA DAC on the board. The VGA controller will use data stored in the last 120,000 bytes of the SRAM as source data for the VGA (to display the image). This set of data will be updated by the ALU as the decompression progresses. If the algorithm is slow one should be able to see the image slowly appear on the screen as the 120,000 bytes go from 0 to the value of the original image.

2. SRAM Controller

The SRAM Controller will store both the compressed image and decompressed image. Initially the addresses associated with the decompressed image will store zeros. Thus the image on the VGA will be a black square. As the decompression algorithm is carried out, the image should start to form on the VGA.

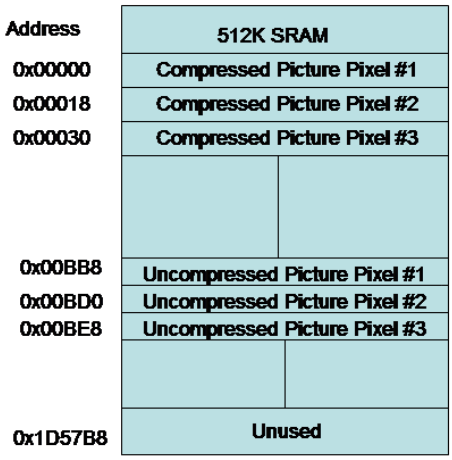

Fig. 3: Organization of 512 kB SRAM

3. Decompression Engine
This is the most significant part of the project. The goal here is to replicate what is normally done in software (Matlab, etc) in hardware to boost speed, both because calculations in hardware are faster and because we can compute much of the algorithm in parallel.
We will be implementing an algorithm called Regularized Orthogonal Matching Pursuit, or ROMP [8], which we expect to provide an appropriate balance between accuracy and ease of implementation. Using Matlab code provided by the developers of the algorithm [9], we intend to implement as much of the algorithm in hardware as is feasible, with the rest being coded on the NIOS II. While we hope to exploit all available opportunities for parallelism in the algorithm, the three color channels represent an embarrassingly parallel problem, in that each color can be decompressed independently of the other two.

D. Software Architecture:

1. VGA (Driver Layer)
This driver will simply display the decompressed image and is interfaced with the hardware VGA.
2. Image Compression in Software (Application)
a. Start with a .tif image. Use Matlab to generate three n x 1 matrices for R, G and B. Multiply each of these by a m x n matrix of Gaussian white noise (between 0 and 1, the result is rounded to the nearest whole number). This gives us compressed data. We can then use sparsity to generate a y-dimensional array (however this is not necessarily necessary… but preferred). This process is similar to how a JPEG image is compressed. Essentially, the y largest coefficients of the m-dimensional array are selected.
b. The compressed image will be transmitted to the FPGA CPU and stored in the first y bytes of the SRAM.

E. Milestones:

1. Milestone 1: Compression of image in software and detailed plan for hardware implementation of the decompression algorithm.
2. Milestone 2: Ability to load an uncompressed image into the SRAM and have the VGA display it.
3. Milestone 3: Load compressed image into SRAM and have the Decompression Engine working (no parallel processing).
4. Final: Parallel processing and optimization of the Decompression Engine

F. References:

1. http://people.ee.duke.edu/~willett/SSP//Tutorials/ssp07-cs-tutorial.pdf
2. http://www.lx.it.pt/~mtf/GPSR/
3. http://www.dsp.ece.rice.edu/cs
4. http://www.dsp.ece.rice.edu/files/cs/CSintro.pdf
5. http://cnx.org/content/m13133/latest
6. http://www.ece.rice.edu/~richb/talks/cs-tutorial-ITA-feb08-complete.pdf
7. http://nuit-blanche.blogspot.com

8. http://www.math.ucdavis.edu/~dneedell/papers/ROMPsampta.pdf
9. http://www.math.ucdavis.edu/~dneedell/romp.m