10.1515/msr-2015-0014

MEASUREMENT SCIENCE REVIEW, Volume 15, No. 2, 2015

An FPGA Architecture for Extracting Real-Time Zernike Coefficients from Measured Phase Gradients

Steven Moser[1], Peter Lee[2], and Adrian Podoleanu[3]

[1]Embedded Systems Group, University of Kent, Canterbury, United Kingdom, S.J.Moser@kent.ac.uk
[2]Embedded Systems Group, University of Kent, Canterbury, United Kingdom
[3]Applied Optics Group, University of Kent, Canterbury, United Kingdom

Zernike modes are commonly used in adaptive optics systems to represent optical wavefronts. However, real-time calculation of Zernike modes is time consuming due to two factors: the large factorial components in the radial polynomials used to define them and the large inverse matrix calculation needed for the linear fit. This paper presents an efficient parallel method for calculating Zernike coefficients from phase gradients produced by a Shack-Hartman sensor and its real-time implementation using an FPGA by pre-calculation and storage of subsections of the large inverse matrix. The architecture exploits symmetries within the Zernike modes to achieve a significant reduction in memory requirements and a speed-up of 2.9 when compared to published results utilising a 2D-FFT method for a grid size of 8×8. Analysis of processor element internal word length requirements show that 24-bit precision in precalculated values of the Zernike mode partial derivatives ensures less than 0.5% error per Zernike coefficient and an overall error of <1%. The design has been synthesized on a Xilinx Spartan-6 XC6SLX45 FPGA. The resource utilisation on this device is <3% of slice registers, <15% of slice LUTs, and approximately 48% of available DSP blocks independent of the Shack-Hartmann grid size. Block RAM usage is <16% for Shack-Hartmann grid sizes up to 32×32.

Keywords: Singular value decomposition, field programmable gate array.

1. Introduction

Real-time estimation of optical wavefronts is critical for using adaptive optics (AO) in diverse fields such as astronomy, microscopy, and biomedical imaging. In the biomedical field, modalities such as Optical Coherence Tomography (OCT) [1] employ AO in high-speed depth resolved imaging [2]. OCT, in particular, generates extremely large data sets which are well suited for parallel processing techniques, especially if real-time operation is desired. For this reason interest has grown in utilising Field Programmable Gate Arrays (FPGA) and Graphics Processing Units (GPU) for such modalities. Similar situations arise in the large adaptive optics systems in modern ground based astronomical observatories. Li, et al., have shown that the increasing demands of OCT type systems will be best serviced by FPGA processing [3]. With this being the case, the authors have chosen FPGAs as our processing platform, as our wavefront sensing technique will also be applied to OCT type modalities.

Modal wavefront estimation is well known in the fields of adaptive optics and wavefront sensing [4]. Southwell [5] shows that Zernike modes make excellent descriptors due to their orthogonality, definition over a circular pupil, and ability to capture common aberrations. Generally, Zernike modes are defined over the polar coordinates (r,θ) and the ith Zernike mode is defined as

$$Z_{mn}(r,\theta) = \begin{cases} R_{mn}(r) & m = 0 \\ R_{mn}(r)\,cos(m\theta) & m > 0 \\ R_{mn}(r)\,sin(m\theta) & m < 0 \end{cases} \quad (1)$$

where i is related to m and n from Noll [6] and $R_{mn}(r)$ is the radial polynomial

$$R_{mn}(r) = \sum_{k=0}^{(n-m)/2} \frac{(-1)^k (n-k)!}{k!(\frac{n+m}{2}-k)!(\frac{n-m}{2}-k)!}\, r^{n-2k} \quad (2)$$

Due to the factorial components in (2), direct calculation of the Zernike modes requires many operations making processing in real-time impractical. In this paper, a fast parallel method for calculating Zernike coefficients from measured phase gradients is presented. Section 2 reviews the general modal method for obtaining coefficients via the singular value decomposition (SVD) of the partial derivatives of the Zernike modes. Section 3 presents the parallel method as a restructuring of the general modal method by separating the large matrix multiplication of the singular value decomposition into inner products; two for each Zernike mode. In Section 4 hardware implementation is presented along with error analysis, timing and resource utilisation.

2. Coefficients via the Modal Method

In general, the modal method attempts to reconstruct the wavefront phase $W(x,y)$ at the sensor plane via a sum of orthogonal functions of the form

$$W(x,y) = \sum_{i=1}^{j} c_i Z_i(x,y) \quad (3)$$

where $Z_i(x,y)$ is the ith Zernike polynomial and c_i is the weighting coefficient. Note $i = 0$, the piston term (DC component), is omitted as it cannot be determined from phase derivative measurements [4] and is typically unimportant for these

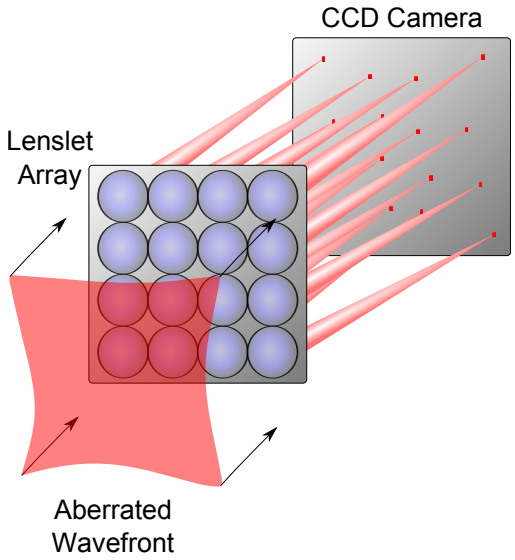

Fig. 1: Components of a Shack-Hartmann wavefront sensor. In this example the wavefront is sampled by a 4×4 grid of lenslets.

applications. The most common wavefront sensor, the Shack-Hartmann is shown in Figure 1. A Shack-Hartmann samples an incoming wavefront with a lenslet array arranged in a regularly spaced Cartesian grid. Zernike polynomials are typically defined in a polar coordinate system. However, they are represented here in the Cartesian system to better match the Shack-Hartmann layout. This method assumes that the measured phase gradients are available over a circular pupil enclosed within a square grid. All values outside of that circle are set to zero, as shown in Figure 2. For purposes of precalculation, points are generated over an evenly spaced Cartesian grid of size $n \times n$ and converted to polar coordinates for calculation of the Zernike modes.

Wavefront sensing methods, such as the Shack-Hartmann [7], or Laser Ray Tracing [8], provide measurements of local phase gradients at the sensor pupil, usually by division of the pupil into $n \times n$ sub-apertures. The gradient is related to (3) via

$$\begin{aligned} S_x &= \frac{\partial W(x,y)}{\partial x} = \sum_{i=1}^{j} c_i \frac{\partial Z_i(x,y)}{\partial x} \\ S_y &= \frac{\partial W(x,y)}{\partial y} = \sum_{i=1}^{j} c_i \frac{\partial Z_i(x,y)}{\partial y}. \end{aligned} \tag{4}$$

In matrix form (4) may be expressed as

$$S_{xy} = A\,c \tag{5}$$

where S_{xy} is a vector of the measured phase gradients with alternating x and y values

$$S_{xy} = \left[\frac{\partial W(x_1,y_1)}{\partial x_1} \frac{\partial W(x_1,y_1)}{\partial y_1} \cdots \frac{\partial W(x_n,y_n)}{\partial x_n} \frac{\partial W(x_n,y_n)}{\partial y_n} \right]^T \tag{6}$$

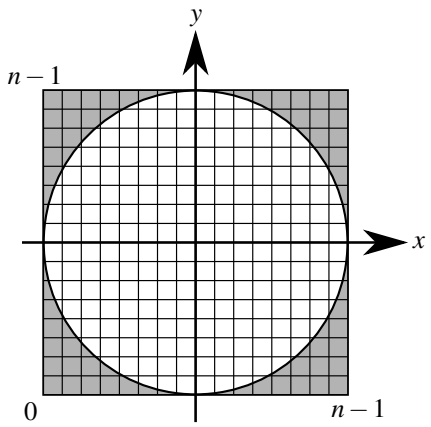

Fig. 2: Circular pupil mapped into a square grid. Grey cells outside the pupil are set to zero.

where T denotes the transposed matrix. A is a matrix of the partial derivatives of the Zernike modes

$$A = \begin{bmatrix} \frac{\partial Z_1(x_1,y_1)}{\partial x_1} & \frac{\partial Z_2(x_1,y_1)}{\partial x_1} & \cdots & \frac{\partial Z_j(x_1,y_1)}{\partial x_1} \\ \frac{\partial Z_1(x_1,y_1)}{\partial y_1} & \frac{\partial Z_2(x_1,y_1)}{\partial y_1} & \cdots & \frac{\partial Z_j(x_1,y_1)}{\partial y_1} \\ \frac{\partial Z_1(x_2,y_2)}{\partial x_2} & \frac{\partial Z_2(x_2,y_2)}{\partial x_2} & \cdots & \frac{\partial Z_j(x_2,y_2)}{\partial x_2} \\ \frac{\partial Z_1(x_2,y_2)}{\partial y_2} & \frac{\partial Z_2(x_2,y_2)}{\partial y_2} & \cdots & \frac{\partial Z_j(x_2,y_2)}{\partial y_2} \\ & & \vdots & \\ \frac{\partial Z_1(x_n,y_n)}{\partial x_n} & \frac{\partial Z_2(x_n,y_n)}{\partial x_n} & \cdots & \frac{\partial Z_j(x_n,y_n)}{\partial x_n} \\ \frac{\partial Z_1(x_n,y_n)}{\partial y_n} & \frac{\partial Z_2(x_n,y_n)}{\partial y_n} & \cdots & \frac{\partial Z_j(x_n,y_n)}{\partial y_n} \end{bmatrix} \tag{7}$$

and c is a column vector of Zernike coefficients

$$c = \begin{bmatrix} c_1 & c_2 & c_3 & \ldots & c_j \end{bmatrix}^T. \tag{8}$$

This may be solved for c, in the least-squares sense, by utilising the pseudoinverse $A^\dagger$ where $A^\dagger$ is obtained via the SVD. From the SVD, $A = U_{nn} S_{nn} V_{nn}^T$ where the columns of U_{nn} are the eigenvectors of AA^T, the columns of V_{nn} are orthonormal eigenvectors of $A^T A$, and S_{nn} is a diagonal matrix, which contains the singular values of A. $A^\dagger$ denotes the pseudoinverse of A which is $A^\dagger = U_{nn} S_{nn}^\dagger V_{nn}^T$. Therefore, the Zernike coefficients may be obtained via

$$c = A^\dagger S_{xy} = U_{nn} S_{nn}^\dagger V_{nn}^T S_{xy}. \tag{9}$$

A parallel method for solving for the elements of c in real-time is presented in the following section.

3. Parallel Modal Method

The proposed parallel method involves separating the x and y components of (6) and (7). For each element c_i in (8) two

coefficients, c_{ix} and c_{iy}, may now be calculated solely from the x and y components, respectively, as shown in (10)

$$\begin{aligned} c_{ix} &= A_{ix}^{\dagger} S_x, \\ c_{iy} &= A_{iy}^{\dagger} S_y \end{aligned} \tag{10}$$

where

$$S_x = \left[\begin{array}{cccc} \frac{\partial W(x_1,y_1)}{\partial x_1} & \frac{\partial W(x_2,y_2)}{\partial x_2} & \cdots & \frac{\partial W(x_n,y_n)}{\partial x_n} \end{array} \right]^T, \tag{11}$$

$$S_y = \left[\begin{array}{cccc} \frac{\partial W(x_1,y_1)}{\partial y_1} & \frac{\partial W(x_2,y_2)}{\partial y_2} & \cdots & \frac{\partial W(x_n,y_n)}{\partial y_n} \end{array} \right]^T, \tag{12}$$

$$A_x = \left[\begin{array}{ccccc} \frac{\partial Z_1(x_1,y_1)}{\partial x_1} & \frac{\partial Z_2(x_1,y_1)}{\partial x_1} & \cdots & \frac{\partial Z_j(x_1,y_1)}{\partial x_1} \\ \frac{\partial Z_1(x_2,y_2)}{\partial x_2} & \frac{\partial Z_2(x_2,y_2)}{\partial x_2} & \cdots & \frac{\partial Z_j(x_2,y_2)}{\partial x_2} \\ & & \vdots & \\ \frac{\partial Z_1(x_n,y_n)}{\partial x_n} & \frac{\partial Z_2(x_n,y_n)}{\partial x_n} & \cdots & \frac{\partial Z_j(x_n,y_n)}{\partial x_n} \end{array} \right] \tag{13}$$

and

$$A_y = \left[\begin{array}{ccccc} \frac{\partial Z_1(x_1,y_1)}{\partial y_1} & \frac{\partial Z_2(x_1,y_1)}{\partial y_1} & \cdots & \frac{\partial Z_j(x_1,y_1)}{\partial y_1} \\ \frac{\partial Z_1(x_2,y_2)}{\partial y_2} & \frac{\partial Z_2(x_2,y_2)}{\partial y_2} & \cdots & \frac{\partial Z_j(x_2,y_2)}{\partial y_2} \\ & & \vdots & \\ \frac{\partial Z_1(x_n,y_n)}{\partial y_n} & \frac{\partial Z_2(x_n,y_n)}{\partial y_n} & \cdots & \frac{\partial Z_j(x_n,y_n)}{\partial y_n} \end{array} \right] \tag{14}$$

and $A_{ix}^{\dagger}$ and $A_{iy}^{\dagger}$ are row vectors taken column-wise from $A_x^{\dagger}$ and $A_y^{\dagger}$ respectively. In this manner, the large matrix multiplication in (9) is decomposed into simple inner products, two for each desired mode. Finally, c_i can be obtained by

$$c_i = c_{ix} + c_{iy}. \tag{15}$$

Assuming the fixed grid size is known *a priori* each $A_{ix}^{\dagger}$ and $A_{iy}^{\dagger}$ vector may be precalculated and stored in memory. Two memory elements are required for each Zernike mode, one for $A_{ix}^{\dagger}$ and another for $A_{iy}^{\dagger}$, allowing ∂x and ∂y to be processed independently, in parallel. The size of each memory element will vary directly with the number of sample points in S_{xy} and the word length of each precalculated $A_{ix}^{\dagger}$ and $A_{iy}^{\dagger}$ values. This method trades off long calculation times inherent in the direct method with larger memory requirements due to the necessity to store the precalculated vectors.

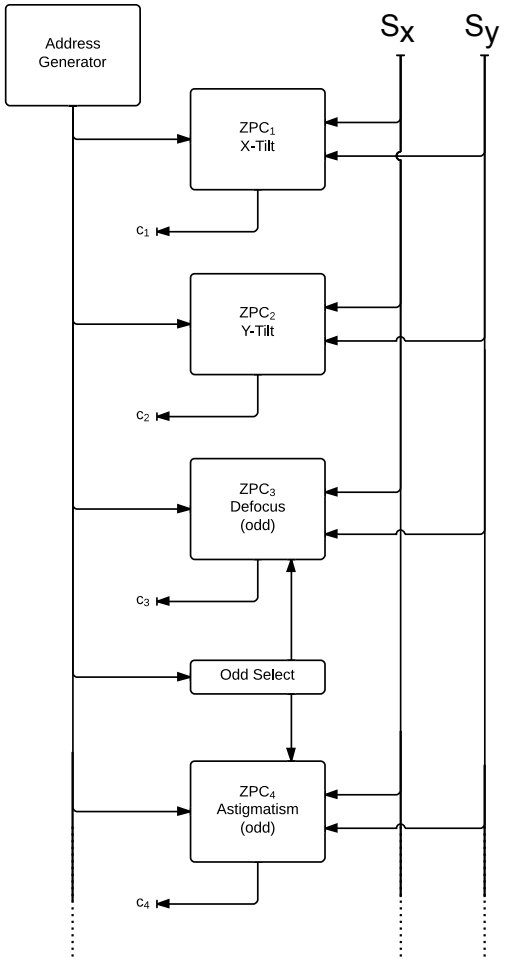

Fig. 3: Block diagram for top level view of Zernike processing blocks.

4. Hardware Implementation

A hardware implementation was undertaken using a Xilinx Spartan-6 FPGA device utilising the Simulink System Generator suite of functions in MATLAB® [9]. The top level layout is shown in Figure 3 illustrating the Zernike Coefficient Processing (ZCP) blocks (labeled X-Tilt, Y-Tilt, Defocus, etc), Address Generator, S_x and S_y inputs and the c_i outputs. Each block accepts the address input from the Address Generator block. In each ZCP block 2 ROMs store the precalculated $A_{ix}^{\dagger}$ and $A_{iy}^{\dagger}$ values for the given Zernike mode.

Figure 4 illustrates internal elements of the ZCP blocks. In each, the **addr** signal cycles from 0 to $(n^2/2)-1$ and then from $(n^2/2)-1$ back to 0 as each ROM holds half of the symmetric $A_{ix}^{\dagger}$ and $A_{iy}^{\dagger}$ vectors, respectively (see subsection 4.1). For the odd ZCP blocks the **select** signal toggles the negated signal through the mux when counting down, as shown in Figure 4.

Each pair of gradients ∂x and ∂y is sent, in parallel, to each ZCP block. The inner product is formed via a multiply and accumulate operation, which is implemented using DSP slices on the FPGA. After the multiply and accumulate stage the c_{ix}

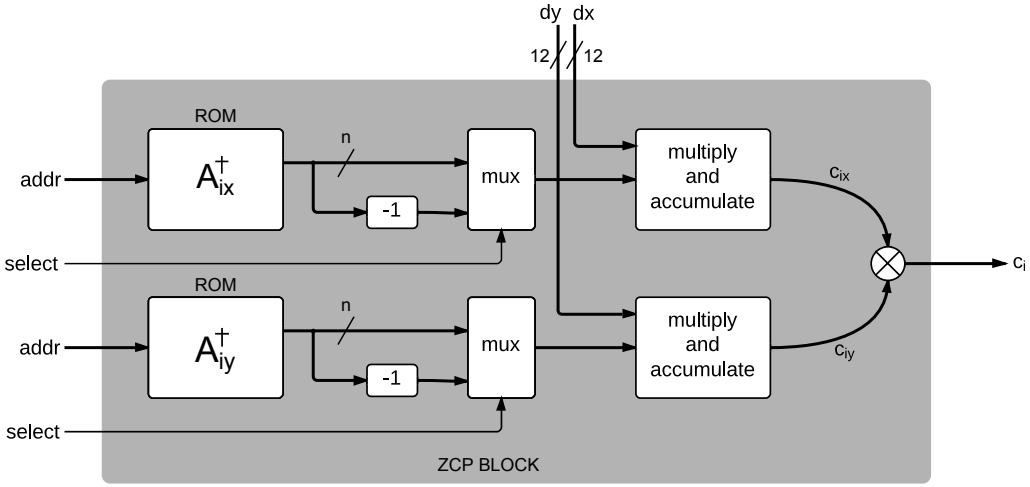

Fig. 4: Block diagram for a Zernike Coefficient Processor.

and c_{iy} values are summed to produce the final c_i. This is only valid once the entire n^2 input pairs are processed.

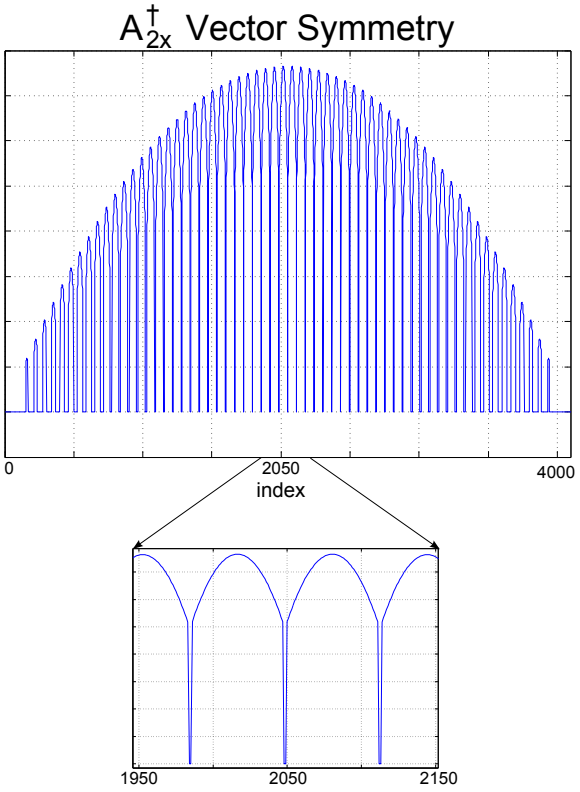

Fig. 5: Symmetry in the $A^{\dagger}_{2x}$ vector about the midpoint (index = 2048) is apparent. Zoomed in subimage shows the symmetry is even.

4.1. *Memory reduction*

Precomputation of the SVD matrices trades off computation time for larger memory usage; therefore, it is imperative that the inherent symmetry of the Zernike modes be exploited to keep the stored data to a minimum. Due to the symmetry of the Zernike modes, $A^{\dagger}_{ix}$ and $A^{\dagger}_{iy}$ also exhibit even or odd symmetry as shown in Figures 5 and 6. This redundancy enables storage of only half the $A^{\dagger}_{ix}$ and $A^{\dagger}_{iy}$ vectors in memory, significantly reducing the precomputation trade-off. To accomplish this in hardware, one counter is used for the forward addressing of the inner product multiplications and a second counter is used for the reverse addressing. A master counter is used as a controller, enabling each counter as needed. For the $A^{\dagger}_{ix}$ and $A^{\dagger}_{iy}$'s that exhibit odd symmetry a negation of each element is performed when the addressing is reversed. This is similar to the techniques used in direct digital synthesis of symmetric waveforms [10] and those used by Li, et al., [11] for wavefront reconstruction on a GPU.

5. Results

To test the design several wavefronts, one of which is shown in Figure 7, were generated in MATLAB. One arbitrary set of nine Zernike coefficients z_{in} is shown in Table 1. Note that the piston term is omitted as discussed in Section 2.

The gradient of the wavefront was calculated, serialized, and sent to Simulink to simulate outputs from the wavefront sensor. Inputs are converted from double precision format to fixed point 12 bit signed values with the binary point at 10 bits. The value of 12 bits was chosen to represent the typical precision of Shack-Hartmann CCD data, but the model can easily be scaled to accommodate other values. It is assumed that the inputs have been normalized to a value between 1 and -1 as the Zernike modes are also normalized between these values. Other input word lengths were tested and it was found that anything above 10 bits had a constant error contribution of less than 0.1%.

Zernike coefficients were calculated from the simulated measurements using word lengths of 18, 20, 22, 24, 26, 28, 30 and 32-bits for the precalculated $A^{\dagger}_{ix}$ and $A^{\dagger}_{iy}$ values respectively. These were tested to find the smallest word length that still gave acceptable accuracy (<1% error). For each word

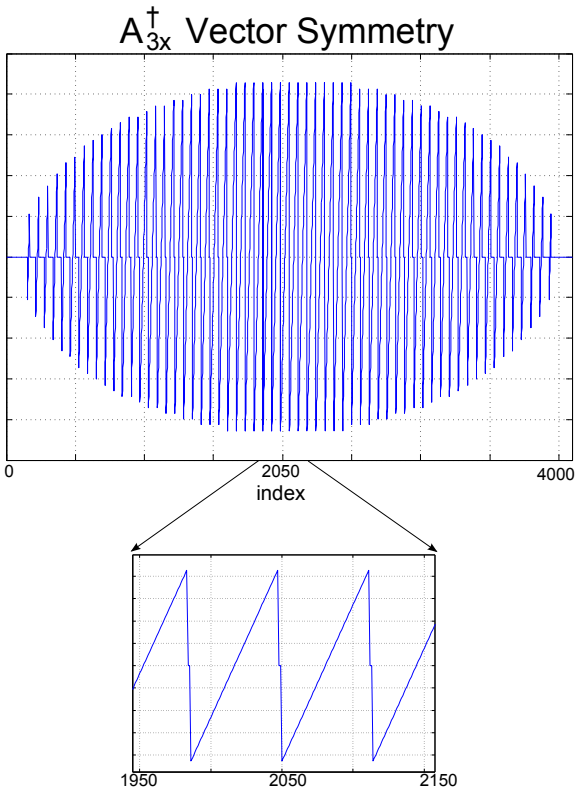

Fig. 6: Symmetry in the $A^{\dagger}_{3x}$ vector about the midpoint (index = 2048) is apparent. Zoomed in subimage shows the symmetry is odd.

length the percent error was calculated via:

$$\%\,\text{error} = \frac{|cin_i - c_i|}{cin_i} \times 100 \tag{16}$$

and results are shown in Figure 8. It is clear from the figure that word lengths above 22-bits ensure error of less than 1% for each of the 9 modes calculated. The difference in error between a 24-bit word length and 32-bit word length is small but not negligible and specific values are shown in Table 1. The total residual error between the reconstructed and input wavefronts is on the order of 10^{-7}.

In addition to verifying the proposed method with a simulated wavefront, experimental data was also utilised. Wavefront data was processed via the direct computation of Zernike coefficients in MATLAB and also via the proposed method. Results are shown in Table 2. For all values the proposed method agrees very well with the direct method giving percent differences of less than 1%.

5.1. Resource utilization and performance

Resource utilisation was determined for implementation on a Xilinx Spartan-6 XC6SLX45 FPGA (system clock = 100MHz) via synthesis of the HDL. This platform was chosen for its relatively low cost and to demonstrate that this design can be implemented on a general purpose FPGA, though the design also scales well to larger devices. Table 3 shows the

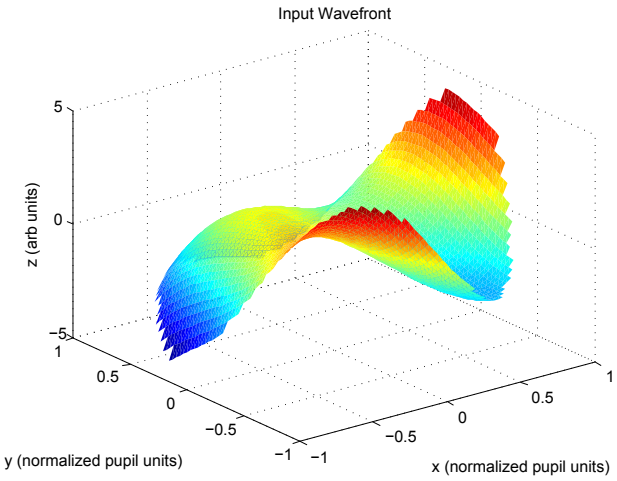

Fig. 7: Simulated Input Wavefront

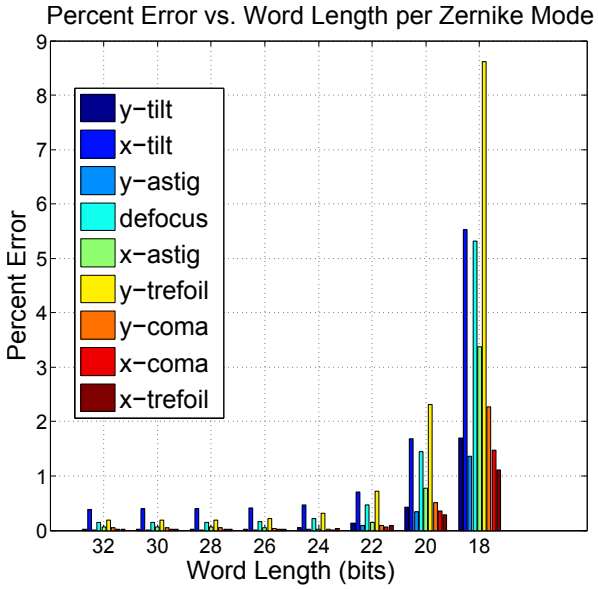

Fig. 8: Percent Error in Calculated Zernike Coefficients for given word lengths.

resource utilisation for four grid sizes and 24-bit ROM values, while Table 4 shows the 32-bit version. FPGA memory is available in pre-allocated sections of dedicated block RAM with sizes of 8kbits and 16kbits (RAMB8 and RAMB16 respectively). The tables show that RAMB8 and RAMB16 usage is unaffected by word length for grid sizes below 32×32. Above that, RAMB16 usage increases dramatically. A grid size of 128×128 was attempted but was too large to fit on this device due to the system memory required for synthesis. DSP48 (specialized processing blocks with embedded multipliers) usage is independent of the grid size chosen, but will become a limiting factor as more Zernike modes are included in the calculation. For a 32×32 grid it was found that a maximum of 28 Zernike modes could be included in the synthesis.

Tables 3 and 4 also show that, other than a slight increase LUT usage, there is no penalty for utilising 32-bit ROM val-

Table 1: Numerical error for calculated Zernike coefficients using 24-bit and 32-bit word lengths.

z_{in}	c_i (24-bit)	% error	c_i (32-bit)	% error
-0.6000	-0.600558519	-0.09	-0.600044608	-0.01
0.2000	0.199260473	0.37	0.199770331	0.11
0.7618	0.761454582	0.05	0.761984587	-0.02
-0.2000	-0.200686216	-0.34	-0.200161457	-0.08
0.3000	0.299332142	0.22	0.299838722	0.05
0.1200	0.119457960	0.45	0.119983137	0.01
-0.4500	-0.450639725	-0.14	-0.450115860	-0.03
0.6780	0.677210808	0.12	0.677734435	0.04
0.9230	0.922606468	0.04	0.923126817	-0.01

Table 2: Comparison of measured wavefront reconstruction via the Direct and Proposed method.

Zernike mode	Direct Method	Proposed Method	% Difference
y-tilt	-0.0002	-0.0002	0.0
x-tilt	-0.0101	-0.0100	0.8
y-astig.	0.0181	0.0180	0.1
defocus	-0.0021	-0.0021	-0.7
x-astig.	-0.0254	-0.0255	-0.3
y-trefoil	0.0205	0.0206	-0.4
y-coma	-0.0025	-0.0026	-0.2
x-coma	0.0017	0.0017	0.5
x-trefoil	0.0080	0.0079	0.2

Table 3: Resource utilisation for 24 bit ROM values compiled for a Xilinx Spartan-6 XC6SLX45.

Grid size	Slice Registers	Slice LUTs	RAMB8	RAMB16	DSP48s	Clock Cycles	Duration (μs @ 100MHz)
8×8	1384	2049	18	0	28	57	0.57
16×16	1386	2060	18	0	28	239	2.39
32×32	1388	2068	0	18	28	984	9.84
64×64	1390	2083	0	54	28	4011	40.11

Table 4: Resource utilisation for 32 bit ROM values compiled for a Xilinx Spartan-6 XC6SLX45.

Grid size	Slice Registers	Slice LUTs	RAMB8	RAMB16	DSP48s	Clock Cycles	Duration (μs @ 100MHz)
8×8	1832	4138	18	0	28	57	0.57
16×16	1834	4144	18	0	28	239	2.39
32×32	1836	4160	0	18	28	984	9.84
64×64	1838	4164	0	72	28	4011	40.11

Table 5: Resource utilisation for 32 bit ROM values compiled for a Xilinx Virtex-6 XC6VLX240T-1FFG1156.

Grid size	Slice Registers	Slice LUTs	RAMB18	RAMB36	DSP48s	Clock Cycles	Duration (μs @ 200MHz)
8×8	1656 (1%)	2557 (9%)	18 (2%)	0	28 (4%)	57	0.285
16×16	1660 (1%)	2561 (9%)	18 (2%)	0	28 (4%)	239	1.195
32×32	1664 (1%)	2566 (9%)	18 (2%)	0	28 (4%)	984	4.92
64×64	1668 (1%)	2572 (9%)	0	36 (9%)	28 (4%)	4011	20.06

Table 6: Resource utilisation for 2D-FFT phase recovery circuit, 8 bit inputs, compiled for a Xilinx Virtex-4 XC4VSX35. Taken from [12]. "XX" indicates that slice LUTs and RAMB8 utilisation was not reported.

Grid size	Slice Registers	Slice LUTs	RAMB8	RAMB16	DSP48s	Clock Cycles	Duration (μs @ 100MHz)
8×8	1141	XX	XX	4	6	162	1.62
16×16	1667	XX	XX	4	6	412	4.12
32×32	2855	XX	XX	4	15	1304	13.04
64×64	4613	XX	XX	16	15	4516	45.16
128×128	8135	XX	XX	64	25	17084	170.84

Table 7: Resource utilisation for 2D-FFT phase recovery circuit, 16 bit inputs, compiled for a Xilinx Virtex-4 XC4VSX35. Taken from [12]. "XX" indicates that slice LUTs and RAMB8 utilisation was not reported.

Grid size	Slice Registers	Slice LUTs	RAMB8	RAMB16	DSP48s	Clock Cycles	Duration (μs @ 100MHz)
8×8	1650	XX	XX	4	10	162	1.62
16×16	2302	XX	XX	4	10	412	4.12
32×32	3992	XX	XX	8	22	1304	13.04
64×64	6472	XX	XX	24	22	4516	45.16
128×128	11326	XX	XX	24	36	17084	170.84

ues for grid sizes lower than 32×32 as neither overruns the pre-allocated RAMB8/16 dimensions. This design also scales very well to the larger FPGA devices. Table 5 shows the device utilization for a Virtex-6 XC6VLX240T-1FFG1156 FPGA and demonstrates that the resource utilisation is very low, consisting of 1% of slice registers, 9% of slice LUTs, 4% of DSP48s, and <10% of block RAM for the largest grid size.

6. Conclusion

The proposed method in this paper compares favorably with other methods of wavefront coefficient generation previously published. Rodríguez-Ramos, et. al, utilized a forward and inverse 2D-FFT method for grid sizes from 8×8 to 128×128 [12], though in this case performing a full phase recovery. In order to make a fair comparison only the time taken to calculate the Fourier coefficients from the forward 2D-FFT portion of the recoverer is considered, which is approximately 1.67μs compared to the proposed method's 0.57μs (both running with a system clock = 100 MHz).

Resource utilisation in the proposed method compares favourably with that in [12]. The resources consumed by *each* 2D-FFT block in [12] are shown in Tables 6 and 7, both taken directly from [12]. Table 6 shows utilisation for 8-bit inputs while Table 7 is for 16-bit inputs. To generate a complete set of Fourier coefficients, two such blocks must be used, doubling the resources reported. In each case the resource utilisation is less in the proposed method, except in the DSP usage for the 8×8 and 16×16 grid sizes, where the proposed method uses 28 DSP48s and [12] uses 12 DSP (8-bit) or 20 DSP (16-bit).

Saunter, et. al, [13] implemented a wavefront reconstruction scheme, producing Zernike coefficients, using an FPGA in approximately $1.19\,\mu$s for an 8×8 grid of Shack-Hartmann spots, running at 80 MHz. The proposed method achieves these results in 0.57μs. As a direct comparison, if the proposed method is run at 80 MHz (the same clock speed as the method in [13]) the results are achieved in $0.7125\,\mu$s, an improvement of 40%. Specifics of the implementation are not given by Saunter so a comprehensive comparison is not possible but, as above, illustrates the competitive nature of this design.

Utilising the high-level Simulink design environment greatly reduced the development and testing time required to implement this design. Precomputation of the ROM values may be done directly in MATLAB and imported into the hardware design with minimal effort. After the initial design was validated, addition of more Zernike modes, variations in grid size, or changes in binary word length are trivial. Compilation for simple devices such as the Spartan-6 may be done directly in MATLAB whereas more complex devices such as the Virtex-6 require a simple VHDL top-level wrapper to enclose the design and provide proper clocking. The trade-off in this approach is somewhat less efficient VHDL (as automatically produced by MATLAB) but the authors feel that the gain in development time is justified, especially considering the long development times typically associated with FPGA design.

In summary, this paper has shown a parallel method of calculating Zernike coefficients from measured phase gradients and its hardware implementation. Coefficients were fitted to the phase gradients, in the least-squares sense, via a precomputed singular value decomposition. Symmetries in the precomputed vectors are exploited to halve the required memory. Resource utilisation shows heavy usage of DSP slices with low usage of registers and LUTs. Memory usage is independent of word length below grid sizes of 32×32; above which it increases dramatically. Future work will entail performing a full phase recovery via another inner product between the Zernike coefficients and precomputed Zernike modes stored in ROM. This will allow for visualization of the sensed wavefront in real-time.

References

[1] Huang, D., Swanson, E. A., Lin, C. P., Schuman, J. S., Stinson, W. G., Chang, W., Hee, M. R., Flotte, T., Gregory, K., Puliafito, C. A., Fujimoto, J. G. (1991). Optical coherence tomography. *Science*, 254, 1178–1181.

[2] Podoleanu, A. (2005). Optical coherence tomography. *British journal of radiology*, 78 (935), 976–988.

[3] Li, J., Sarunic, M., Shannon, L. (2011). Scalable, high performance Fourier domain optical coherence tomography: Why FPGAs and not GPGPUs. In *IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 1-3 May 2011. IEEE, 49–56.

[4] Cubalchini, R. (1979). Modal wave-front estimation from phase derivative measurements. *Journal of the Optical Society of America*, 69 (7), 972–977.

[5] Southwell, W. H. (1980). Wave-front estimation from wave-front slope measurements. *Journal of the Optical Society of America*, 70 (8), 998–1006.

[6] Noll, R. J. (1976). Zernike polynomials and atmospheric turbulence. *Journal of the Optical Society of America*, 66 (3), 207–211.

[7] Shack, R. V., Platt, B. (1971). Production and use of a lenticular Hartmann screen. *Journal of the Optical Society of America*, 61 (5), 656.

[8] Navarro, R., Moreno-Barriuso, E. (1999). Laser ray-tracing method for optical testing. *Optics letters*, 24 (14), 951–953.

[9] The MathWorks Inc. (2012). *MATLAB 7.10.0 (R2012a)*.

[10] Analog Devices Inc. (1999). *A technical tutorial on digital signal synthesis*.

[11] Li, D., Hu, L., Mu, Q., Cao, Z., Peng, Z., Liu, Y., Yao, L., Yang, C., Lu, X., Xuan, L. (2014). Wavefront processor for liquid crystal adaptive optics system based on graphics processing unit. *Optics Communications*, 316, 211–216.

[12] Rodríguez-Ramos, J., Castelló, E. M., Conde, C. D., Valido, M. R., Marichal-Hernández, J. (2008). 2D-FFT implementation on FPGA for wavefront phase recovery from the CAFADIS camera. In *Adaptive Optics Systems, Proc. SPIE 7015*.

[13] Saunter, C., Love, G., Johns, M., Holmes, J. (2005). FPGA technology for high-speed low-cost adaptive optics. In: *5th International Workshop on Adaptive Optics for Industry and Medicine*. International Society for Optics and Photonics, 60181G.

Received October 15, 2014.
Accepted May 15, 2015.