Citation for published version


DOI

https://doi.org/10.1515/msr-2015-0014

Link to record in KAR

http://kar.kent.ac.uk/48966/

Document Version

Publisher pdf

Copyright & reuse
Content in the Kent Academic Repository is made available for research purposes. Unless otherwise stated all content is protected by copyright and in the absence of an open licence (eg Creative Commons), permissions for further reuse of content should be sought from the publisher, author or other copyright holder.

Versions of research
The version in the Kent Academic Repository may differ from the final published version. Users are advised to check http://kar.kent.ac.uk for the status of the paper. Users should always cite the published version of record.

Enquiries
For any further enquiries regarding the licence status of this document, please contact: researchsupport@kent.ac.uk

If you believe this document infringes copyright then please contact the KAR admin team with the take-down information provided at http://kar.kent.ac.uk/contact.html
An FPGA Architecture for Extracting Real-Time Zernike Coefficients from Measured Phase Gradients

Steven Moser¹, Peter Lee², and Adrian Podoleanu³

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

Zernike modes are commonly used in adaptive optic 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 $8 \times 8$. 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 $32 \times 32$.

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 $(r, \theta)$ and the $i$th Zernike mode is defined as

$$Z_{im}(r, \theta) = \begin{cases} \frac{R_{nm}(r)}{R_{nm}(r)} & m = 0 \\ \frac{R_{nm}(r) \cos(m\theta)}{R_{nm}(r)} & m > 0 \\ \frac{R_{nm}(r) \sin(m\theta)}{R_{nm}(r)} & m < 0 \end{cases}$$

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

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

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

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

where $Z_i(x, y)$ is the $i$th 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
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

\[
S_x = \frac{\partial W(x,y)}{\partial x} = \sum_{i=1}^{j} c_i \frac{\partial Z_i(x,y)}{\partial x},
\]

\[
S_y = \frac{\partial W(x,y)}{\partial y} = \sum_{i=1}^{j} c_i \frac{\partial Z_i(x,y)}{\partial y}.
\]

In matrix form (4) may be expressed as

\[
S_{xy} = A c
\]

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

\[
S_{xy} = \begin{bmatrix}
\frac{\partial W(x_1,y_1)}{\partial x_1} & \frac{\partial W(x_1,y_1)}{\partial y_1} & \cdots & \frac{\partial W(x_1,y_1)}{\partial y_n} \\
\frac{\partial W(x_n,y_1)}{\partial x_n} & \frac{\partial W(x_n,y_1)}{\partial y_1} & \cdots & \frac{\partial W(x_n,y_1)}{\partial y_n}
\end{bmatrix}^T.
\]

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

\[
A = \begin{bmatrix}
\frac{\partial Z_1(x_1,y_1)}{\partial x_1} & \frac{\partial Z_2(x_1,y_1)}{\partial x_1} & \cdots & \frac{\partial Z_{m}(x_1,y_1)}{\partial x_1} \\
\frac{\partial Z_1(x_1,y_1)}{\partial y_1} & \frac{\partial Z_2(x_1,y_1)}{\partial y_1} & \cdots & \frac{\partial Z_{m}(x_1,y_1)}{\partial y_1} \\
\frac{\partial Z_1(x_2,y_2)}{\partial x_2} & \frac{\partial Z_2(x_2,y_2)}{\partial x_2} & \cdots & \frac{\partial Z_{m}(x_2,y_2)}{\partial x_2} \\
\frac{\partial Z_1(x_2,y_2)}{\partial y_2} & \frac{\partial Z_2(x_2,y_2)}{\partial y_2} & \cdots & \frac{\partial Z_{m}(x_2,y_2)}{\partial y_2} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial Z_1(x_n,y_n)}{\partial x_n} & \frac{\partial Z_2(x_n,y_n)}{\partial x_n} & \cdots & \frac{\partial Z_{m}(x_n,y_n)}{\partial x_n} \\
\frac{\partial Z_1(x_n,y_n)}{\partial y_n} & \frac{\partial Z_2(x_n,y_n)}{\partial y_n} & \cdots & \frac{\partial Z_{m}(x_n,y_n)}{\partial y_n}
\end{bmatrix}
\]

and \( c \) is a column vector of Zernike coefficients

\[
c = \begin{bmatrix} c_1 & c_2 & c_3 & \cdots & c_j \end{bmatrix}^T.
\]

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, \( A = U_m S_m V_m^T \), where the columns of \( U_m \) are the eigenvectors of \( A A^T \), the columns of \( V_m \) are orthonormal eigenvectors of \( A^T A \), and \( S_m \) is a diagonal matrix, which contains the singular values of \( A \). \( A^\dagger \) denotes the pseudoinverse of \( A \) which is \( A^\dagger = U_m S_m^\dagger V_m^T \). Therefore, the Zernike coefficients may be obtained via

\[
c = A^\dagger S_{xy} = U_m S_m^\dagger V_m^T S_{xy}.
\]

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)

\[
\begin{align*}
    c_{ix} &= A_{ix}^T S_x, \\
    c_{iy} &= A_{iy}^T S_y
\end{align*}
\]

where

\[
S_x = \begin{bmatrix}
    \frac{\partial W(x_1,y_1)}{\partial x_1} & \frac{\partial W(x_2,y_2)}{\partial x_2} & \ldots & \frac{\partial W(x_n,y_n)}{\partial x_n}
\end{bmatrix}^T, \quad (11)
\]

\[
S_y = \begin{bmatrix}
    \frac{\partial W(x_1,y_1)}{\partial y_1} & \frac{\partial W(x_2,y_2)}{\partial y_2} & \ldots & \frac{\partial W(x_n,y_n)}{\partial y_n}
\end{bmatrix}^T, \quad (12)
\]

\[
A_x = \begin{bmatrix}
    \frac{\partial Z_1(x_1,y_1)}{\partial x_1} & \frac{\partial Z_2(x_2,y_2)}{\partial x_2} & \ldots & \frac{\partial Z_j(x_j,y_j)}{\partial x_j} \\
    \frac{\partial Z_1(x_2,y_2)}{\partial y_2} & \frac{\partial Z_2(x_2,y_2)}{\partial y_2} & \ldots & \frac{\partial Z_j(x_j,y_j)}{\partial y_j} \\
    \vdots & \vdots & \ddots & \vdots \\
    \frac{\partial Z_1(x_n,y_n)}{\partial x_n} & \frac{\partial Z_2(x_n,y_n)}{\partial x_n} & \ldots & \frac{\partial Z_j(x_j,y_j)}{\partial x_j}
\end{bmatrix}, \quad (13)
\]

and

\[
A_y = \begin{bmatrix}
    \frac{\partial Z_1(x_1,y_1)}{\partial y_1} & \frac{\partial Z_2(x_2,y_2)}{\partial y_2} & \ldots & \frac{\partial Z_j(x_j,y_j)}{\partial y_j} \\
    \frac{\partial Z_1(x_2,y_2)}{\partial y_2} & \frac{\partial Z_2(x_2,y_2)}{\partial y_2} & \ldots & \frac{\partial Z_j(x_j,y_j)}{\partial y_j} \\
    \vdots & \vdots & \ddots & \vdots \\
    \frac{\partial Z_1(x_n,y_n)}{\partial y_n} & \frac{\partial Z_2(x_n,y_n)}{\partial y_n} & \ldots & \frac{\partial Z_j(x_j,y_j)}{\partial y_j}
\end{bmatrix}, \quad (14)
\]

and \( A_{ix} \) and \( A_{iy} \) are row vectors taken column-wise from \( A_x \) and \( A_y \), 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

\[
c_i = c_{ix} + c_{iy}. \quad (15)
\]

Assuming the fixed grid size is known \emph{a priori} each \( A_{ix} \) and \( A_{iy} \) vector may be precalculated and stored in memory. Two memory elements are required for each Zernike mode, one for \( A_{ix} \) and another for \( A_{iy} \), allowing \( \partial x \) and \( \partial y \) 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 \( A_{ix} \) and \( A_{iy} \) 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.

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 \( A_{ix} \) and \( A_{iy} \) values for the given Zernike mode.

Figure 4 illustrates internal elements of the ZCP blocks. In each, the \texttt{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 \( A_{ix} \) and \( A_{iy} \) vectors, respectively (see subsection 4.1). For the odd ZCP blocks the \texttt{select} signal toggles the negated signal through the mux when counting down, as shown in Figure 4.

Each pair of gradients \( \partial x \) and \( \partial y \) 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 \( c_{ix} \)
and $c_y$ values are summed to produce the final $c_i$. This is only valid once the entire $n^2$ input pairs are processed.

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, $A_{ix}^\dagger$ and $A_{iy}^\dagger$ also exhibit even or odd symmetry as shown in Figures 5 and 6. This redundancy enables storage of only half the $A_{ix}^\dagger$ and $A_{iy}^\dagger$ 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 $A_{ix}^\dagger$ and $A_{iy}^\dagger$'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 $z_{im}$ 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 $A_{ix}^\dagger$ and $A_{iy}^\dagger$ 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 $A_{3x}^\dagger$ vector about the midpoint (index = 2048) is apparent. Zoomed in subimage shows the symmetry is odd.

Fig. 7: Simulated Input Wavefront

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

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

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 $32 \times 32$. Above that, RAMB16 usage increases dramatically. A grid size of $128 \times 128$ was attempted but was too large to fit on this device due to the system memory required for 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.

<table>
<thead>
<tr>
<th>$z_i$</th>
<th>$c_i$ (24-bit)</th>
<th>% error</th>
<th>$c_i$ (32-bit)</th>
<th>% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.6000</td>
<td>-0.600558519</td>
<td>-0.09</td>
<td>-0.600044608</td>
<td>-0.01</td>
</tr>
<tr>
<td>0.2000</td>
<td>0.199260473</td>
<td>0.37</td>
<td>0.199770331</td>
<td>0.11</td>
</tr>
<tr>
<td>0.7618</td>
<td>0.761454582</td>
<td>0.05</td>
<td>0.761984587</td>
<td>-0.02</td>
</tr>
<tr>
<td>-0.2000</td>
<td>-0.200686216</td>
<td>-0.34</td>
<td>-0.200161457</td>
<td>-0.08</td>
</tr>
<tr>
<td>0.3000</td>
<td>0.299332142</td>
<td>0.22</td>
<td>0.299838722</td>
<td>0.05</td>
</tr>
<tr>
<td>0.1200</td>
<td>0.119457960</td>
<td>0.45</td>
<td>0.119983137</td>
<td>0.01</td>
</tr>
<tr>
<td>-0.4500</td>
<td>-0.450639725</td>
<td>-0.14</td>
<td>-0.450115860</td>
<td>-0.03</td>
</tr>
<tr>
<td>0.6780</td>
<td>0.677210808</td>
<td>0.12</td>
<td>0.677734435</td>
<td>0.04</td>
</tr>
<tr>
<td>0.9230</td>
<td>0.922606468</td>
<td>0.04</td>
<td>0.923126817</td>
<td>-0.01</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Grid size</th>
<th>Slice Registers</th>
<th>Slice LUTs</th>
<th>RAMB8</th>
<th>RAMB16</th>
<th>DSP48s</th>
<th>Clock Cycles</th>
<th>Duration (µs @ 100MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8×8</td>
<td>1384</td>
<td>2049</td>
<td>18</td>
<td>0</td>
<td>28</td>
<td>57</td>
<td>0.57</td>
</tr>
<tr>
<td>16×16</td>
<td>1386</td>
<td>2060</td>
<td>18</td>
<td>0</td>
<td>28</td>
<td>239</td>
<td>2.39</td>
</tr>
<tr>
<td>32×32</td>
<td>1388</td>
<td>2068</td>
<td>0</td>
<td>18</td>
<td>28</td>
<td>984</td>
<td>9.84</td>
</tr>
<tr>
<td>64×64</td>
<td>1390</td>
<td>2083</td>
<td>0</td>
<td>54</td>
<td>28</td>
<td>4011</td>
<td>40.11</td>
</tr>
</tbody>
</table>
Table 4: Resource utilisation for 32 bit ROM values compiled for a Xilinx Spartan-6 XC6SLX45.

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

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

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

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.

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

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.

<table>
<thead>
<tr>
<th>Grid size</th>
<th>Slice Registers</th>
<th>Slice LUTs</th>
<th>RAMB8</th>
<th>RAMB16</th>
<th>DSP48s</th>
<th>Clock Cycles</th>
<th>Duration (µs @ 100MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8×8</td>
<td>1650</td>
<td>XX</td>
<td>4</td>
<td>10</td>
<td>162</td>
<td>1.62</td>
<td></td>
</tr>
<tr>
<td>16×16</td>
<td>2302</td>
<td>XX</td>
<td>4</td>
<td>10</td>
<td>412</td>
<td>4.12</td>
<td></td>
</tr>
<tr>
<td>32×32</td>
<td>3992</td>
<td>XX</td>
<td>8</td>
<td>22</td>
<td>1304</td>
<td>13.04</td>
<td></td>
</tr>
<tr>
<td>64×64</td>
<td>6472</td>
<td>XX</td>
<td>24</td>
<td>22</td>
<td>4516</td>
<td>45.16</td>
<td></td>
</tr>
<tr>
<td>128×128</td>
<td>11326</td>
<td>XX</td>
<td>24</td>
<td>36</td>
<td>17084</td>
<td>170.84</td>
<td></td>
</tr>
</tbody>
</table>
ues for grid sizes lower than $32 \times 32$ 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 $8 \times 8$ to $128 \times 128$ [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 receiver is considered, which is approximately $1.67\mu s$ compared to the proposed method’s $0.57\mu s$ (both running with a system clock = $100\,\text{MHz}$).

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 $8 \times 8$ and $16 \times 16$ 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 $8 \times 8$ grid of Shack-Hartmann spots, running at $80\,\text{MHz}$. The proposed method achieves these results in $0.57\mu s$. As a direct comparison, if the proposed method is run at $80\,\text{MHz}$ (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 $32 \times 32$; 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


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