Modal Fourier Wavefront Reconstruction Using FPGA Technology

Eduardo Magdaleno*, Manuel Rodríguez, José Manuel Rodríguez-Ramos and Alejandro Ayala

Grupo de Comunicaciones y Teledetección, Departamento de Física Fundamental y Experimental, Electrónica y sistemas, La Laguna University, La Laguna, Spain

Abstract: Atmospheric turbulence introduces optical aberration into wavefronts arriving at ground-based telescopes. The wavefront reconstruction using fast Fourier transforms (FFT) and spatial filtering is computationally tractable and sufficiently accurate in large Shack-Hartmann-based adaptive optics systems (up to 10,000 actuators). A first design has been developed using a Graphical Processing Units (GPU) platform. However, an increase in telescope size requires significant computational power. For this reason other hardware technologies must be taken into account during the development of a specific processor.

In this paper, an improvement on the efficiency of the above methods is proposed based on Field Programmable Gate Array (FPGA) technology. To the best of our knowledge, there are not other wavefront reconstruction implementations based on FPGA technology. The basic advantages of this technology are its flexible architecture and extremely high-performance signal processing capability which are captured through parallelism. The implemented phase recoverer is, consequently, faster than the CPU or the GPU solution. Furthermore, the implemented design incorporating this technology meets current and future adaptive optics image processing frame rate requirements.

Keywords: Adaptive optics, FPGA, Phase recovery, Fast fourier transform, DSP.

1. INTRODUCTION

Current adaptive optics (AO) systems use vector-matrix-multiply (VMM) reconstructors to convert gradient measurements to wavefront phase estimates. As the number of actuators n increases, the time to compute the reconstruction by means of the VMM method scales as $O(n^2)$. The number of actuators involved in AO systems is expected to increase dramatically in the future. For instance, the increase in the field of astronomy is due to increasing telescope diameters and new higher-resolution applications on existing systems. The size increase ranges from hundreds up to tens of thousands of actuators and requires faster methods to complete the AO correction within the specified atmospheric characteristic time. The next generation of extremely large telescopes (with diameters measuring from 50 up to 100 meters) will demand important technological advances to maintain telescope segment alignment (phasing of segmented mirrors) and posterior atmospheric aberrations corrections. Adaptive optics includes several steps: detection, wavefront phase recovery, information transmission to the actuators and their mechanical movements. A quicker wavefront phase reconstruction appears to be an extremely relevant step in its improvement.

The modal estimation of the wavefront consists in using the slope measurements to fit the coefficients of an aperture function in a phase expansion of orthogonal functions. These functions are usually Zernike polynomials or complex exponentials, but there are other possibilities, depending on the pupil mask. Very fast algorithms can be implemented when using complex exponential polynomials because the FFT kernel is the same [1, 2].

Until recently, the problem of an efficient phase recoverer design has been implemented over a GPU platform with satisfactory execution time results [3]. GPUs provide great computational power in a low cost broadly used peripheral but the results are not optimized due to the rigid architecture of this device. An FPGA prototype can accelerate the calculation due to the architecture of this device. In this way, FPGA technology offers extremely high-performance signal processing capability through parallelism based on slices and arithmetic circuits and high flexible interconnection possibilities [4, 5]. Moreover, FPGA technology is an alternative to custom ICs (integrated circuits) for implementing logic. Custom integrated circuits (ASICS) are expensive to develop, while generating time-to-market delays because of the prohibitive design time. Thanks to computer-aided design tools, FPGA circuits can be implemented in a relatively short amount of time [6]. FPGA technology features are an important consideration in our proposed FPGA implementation of the phase recoverer.

In this work, our principal objective is to design and implement a good and fast enough wavefront phase reconstruction algorithm using a Virtex-4 FPGA platform, paving the way to accomplish the extremely large telescope's (ELT) number of actuators computational requirements within a 6 ms limit. We select a modal solution based on spatial filtering and FFTs, because a careful choice of the architecture of the FFT to implement can result in substantial benefits in speed in comparison with the GPU solution. Given that FFTs are the kernel of this solution, we studied several FFTs by modifying their precision and size. Finally we implemented a first functional phase recoverer based on a Shack-Hartmann wavefront sensor that supply 8x8 phase gradient values.

The rest of this work is organized as follows: Section 2 gives an overview on the modal Fourier wavefront phase reconstruction algorithm. A brief description of the FFT and the implementation of the 1D-FFT architecture, features and advantages are shown in section 3. Section 4 is concerned with the description of the proposed architecture for the implementation of a pipeline 2D-FFT using parametrizable VHSIC hardware description language (VHDL) code. In sections 5 and 6 we show how to implement a wavefront

*Address correspondence to this author at the Grupo de Comunicaciones y Teledetección, Departamento de Física Fundamental y Experimental, Electrónica y sistemas, La Laguna University, La Laguna, Spain; Tel: + 34 922 845035; E-mail: emagcas@ull.es

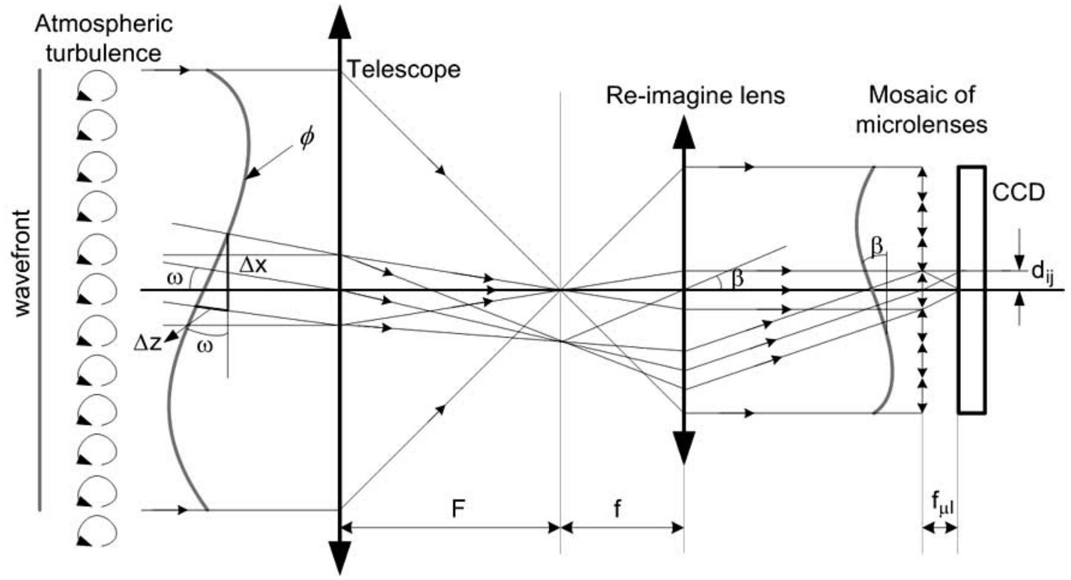

Fig. (1). Diagram of the light travel across a Shack-Hartmann sensor to measure the phase of the wavefront in the telescope pupil.

phase recoverer using the Virtex-4 FPGA and analyze the obtained efficiency and compare it to the GPU alternative implementation, respectively. Suggestions for future research are also presented.

2. A DESCRIPTIVE APPROACH

The Shack-Hartmann wavefront sensor samples the signal $\Psi_{telescope}(u,v)$ (complex amplitude of the electromagnetic field) to obtain the wavefront phase map: $\phi(u,v)$. This is only possible if the sampling is done by microlenses or subpupils with r_0 dimension (that is, inside the phase coherence domain). An array of microlenses is used to sample the wavefront. This array is a rigid piece that fixes the sampling rate. Each (i,j) microlense produces a spot (Fig. **1**):

$$I^{ij}(x,y) = \left|FFT[\psi^{ij}_{telescope}(u,v)]\right|^2$$

The displacements d_{ij} of the spots centroid with regard to the references centroid (associated with a plane wavefront phase), are a proportional estimation of the average subpupil phase gradient (Fig. **2**):

$$d_{ij} = K \cdot \frac{\partial \langle \phi \rangle_{subp_{ij}}}{\partial \vec{r}}$$

Where $\vec{r}$ is the position with components (u, v), and K is a constant. K depends on the wavelength and the focal distances of the telescope, reimaging lens and microlenses. From these gradients estimation, the wavefront phase $\phi(u,v)$ can be recovered using an expansion over complex exponential polynomials:

$$\phi(u,v) = \sum_{p,q=0}^{N-1} a_{pq} Z_{pq}(u,v) = \sum_{p,q=0}^{N-1} a_{pq} \frac{1}{N} e^{\frac{2\pi i}{N}(pu+qv)} = IFFT(a_{pq})$$

The gradient is then written:

$$\vec{S}(u,v) = \vec{\nabla}\phi(u,v) = \frac{\partial \phi}{\partial u}\vec{i} + \frac{\partial \phi}{\partial v}\vec{j} = \sum_{p,q} a_{pq} \vec{\nabla} Z_{pq}$$

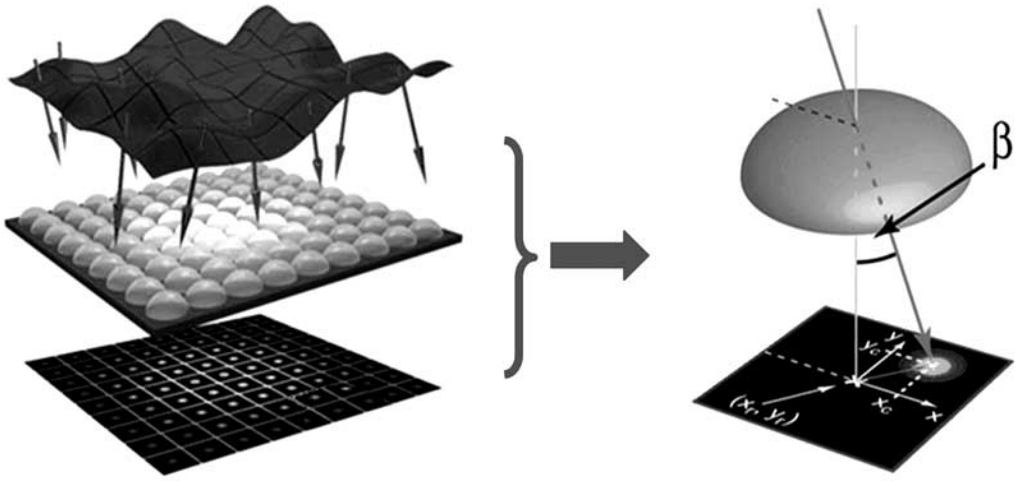

Fig. (2). Estimation of the phase gradient based on the displacements of the spots centroid in the array.

Making a least squares fit over the F function:

$$F = \sum_{u,v=1}^{N} [\vec{S}(u,v) - \sum_{p,q} a_{pq} (\tfrac{\partial Z_{pq}}{\partial u}\vec{i} + \tfrac{\partial Z_{pq}}{\partial v}\vec{j})]^2$$

where $\vec{S}$ are experimental data, the coefficients, a_{pq}, of the complex exponential expansion in a modal Fourier wavefront phase reconstructor (spatial filter), can be written as:

$$a_{pq} = \frac{ipFFT\{S^x(u,v)\} + iqFFT\{S^y(u,v)\}}{p^2+q^2} \tag{1}$$

The phase can then be recovered from the gradient data by transforming backward those coefficients:

$$\phi(u,v) = FFT^{-1}[a_{pq}] \tag{2}$$

A filter composed of three Fourier transforms therefore must be calculated to recover the phase. In order to accelerate the process, an exhaustive study of the crucial FFT algorithm was carried out which allowed the FTT to be specifically adapted to the modal wavefront recovery pipeline and the FPGA architecture.

3. 1D-FFT IMPLEMENTATION

The discrete Fourier transform (DFT) of an N-point discrete-time complex sequence x(n), indexed by n=0, 1, N-1, is defined by

$$X(k) = \sum_{n=0}^{N-1} x(n)\cdot W_N^{kn}, \; k = 0,1, ... \; N-1 \tag{3}$$

where $W_N = e^{-j2\pi/N}$ and is referred to as the twiddle factor. The number of complex multiply and add operations for the computation of an N-point DFT is of order N^2 but the problem is alleviated with the development of special fast algorithms, collectively known as fast Fourier transforms [7, 8]. These algorithms reduce the number of calculations to $N\log_2 N$.

In the decimation-in frequency (DIF), the FFT algorithm starts with splitting the input data set X(k) into odd- and even-numbered points,

$$X(k) = \sum_{n\ even} x(n)\cdot W_N^{kn} + \sum_{n\ odd} x(n)\cdot W_N^{kn} = \sum_{m=0}^{(N/2)-1} x(2m)\cdot W_N^{2mk}$$

$$+ \sum_{m=0}^{(N/2)-1} x(2m+1)\cdot W_N^{k(2m+1)}$$

So, the problem may be viewed as the DFT of N/2 point sequences, each of which may again be computed through two N/4 point DFTs and so on. This is illustrated in the form of a signal flow graph as an example for N=8 in Fig. (**3**).

Finally, for the radix-2 FFT algorithm, the smallest transform or butterfly (basic computational unit) used is the 2-point DFT.

Generally, each butterfly implies one complex multiplier and two complex adders. In particular, multipliers consume much silicon area of FPGA because they are implemented with adder trees. Various implementation proposals have been made to save area removing these multipliers [9-12].

However, in order to implement an efficient multiplier, the last Virtex-4 FPGA devices incorporate specific arithmetic modules, called DSP48. Each DSP48 slice has a two-input multiplier followed by multiplexers and a three-input adder/subtracter.

The DSP48 slices support many independent functions, including multiplier, multiplier-accumulator (MAC), multiplier followed by an adder, three-input adder, barrel shifter, wide bus multiplexers, magnitude comparator, or wide counter. The architecture also supports connecting multiple DSP48 slices to form wide math functions, DSP filters, and

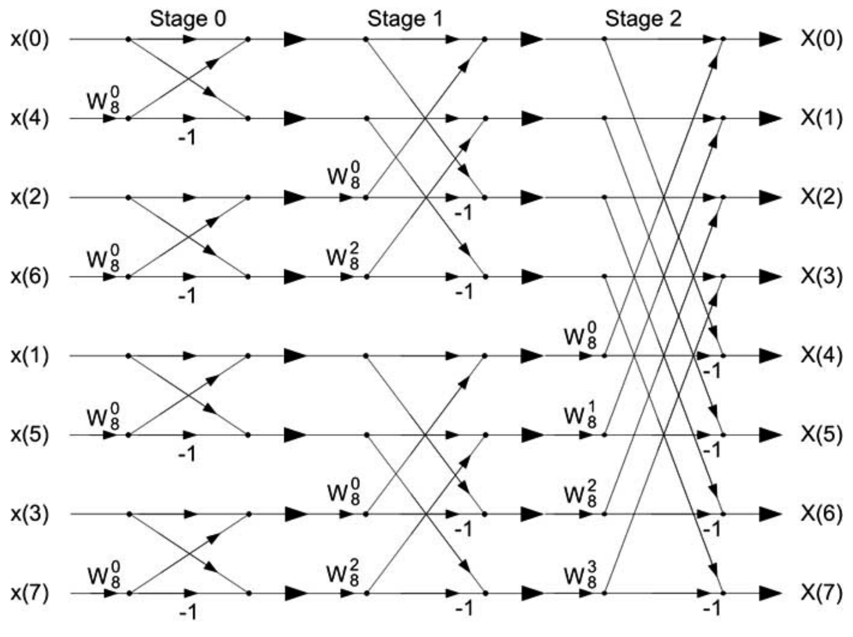

Fig. (3). Computation of FFT by decimation in frequency for N=8.

complex arithmetic without the use of general FPGA fabric [13].

DSP48 circuits allow some internal calculations of the Fourier transform algorithm or the filter of the phase reconstruction to be parallel, such as in complex multiplications. In this way, the use of these components accelerates the calculation in comparison with the above referenced designs. A complex pipeline multiplier is implemented using four DSP48 (Fig. **4**) to calculate:

$$A = A\_real + iA\_imag$$
$$B = B\_real + iB\_imag$$
$$P = A \cdot B = (A\_real + iA\_imag)\cdot(B\_real + iB\_imag) =$$
$$= (A\_real \cdot B\_real - A\_imag \cdot B\_imag)$$
$$+ i(A\_imag \cdot B\_real + A\_real \cdot B\_imag)$$

The real and imaginary results use the same DSP48 slice configuration with the exception of the adder/subtracter. The adder/subtracter performs subtraction for the real result and addition for the imaginary one. This implementation only needs four clock cycles to calculate the complex multiplication with up to 550 MHz in XC4VSX35 Virtex-4 [14].

The pipeline radix-2 architecture (Fig. **5**) uses one butterfly in each stage to calculate the 1D-FFT. The twiddle coefficients used in each stage are stored in twiddle look-up table (LUT) ROMs in the FPGA. Pipelined FFT architectures cannot be offered as a solution for processing large FFTs because they consume a large amount of hardware area which makes them unsuitable for implementation on a single FPGA chip (especially for 2D FFT implementations). In our particular application, a small 32x32 2D-FFT is sufficient in order to recover the wavefront and we select the faster architecture. The user can stream in input data and, after the latency time, can continuously unload the results because the number of clock cycles to calculate the transform is the number of points of the FFT. Other architectures have been implemented to save area, including a single radix-2 butterfly, a radix-4 butterfly, splix-radix and others [15, 16].

Usually, the butterfly implementation requires multiple pipeline stages and apart from adders and subtractors also requires multipliers, which correspond to the W^k coefficients. The butterfly in the first and second stages of the implementation can be reduced and they are computed faster than the other stages. The first stage (see Fig. **3**) utilizes the feature of the twiddle factors related to the first stages of the pipeline.

$$W_N^{N/2} = 1$$

So, the first stage can be implemented in a very simple way with an adder/subtracter. In the second stage, the next twiddle factors are

$$W_N^{N/4} = j$$

This twiddle suggests a similar splitting structure in the second pipeline stage as in the first one; however, the imaginary unit imposes a special consideration: two additional multiplexers change real and imaginary data, and the pipeline adder/subtracter works according the following equation.

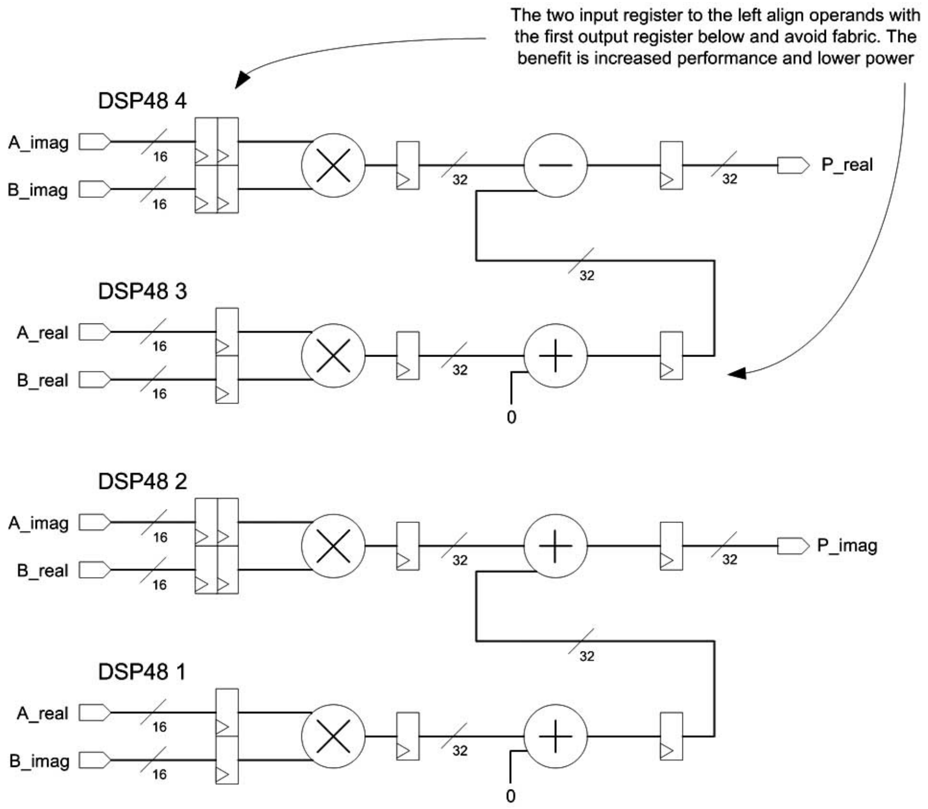
The two input register to the left align operands with the first output register below and avoid fabric. The benefit is increased performance and lower power
DSP48 4
A_imag
B_imag
16
16
32
32
P_real
32
DSP48 3
A_real
B_real
16
16
32
0
DSP48 2
A_imag
B_real
16
16
32
32
P_imag
32
DSP48 1
A_real
B_imag
16
16
32
0

Fig. (4). Pipeline, complex multiplier based on DSP48 circuits.

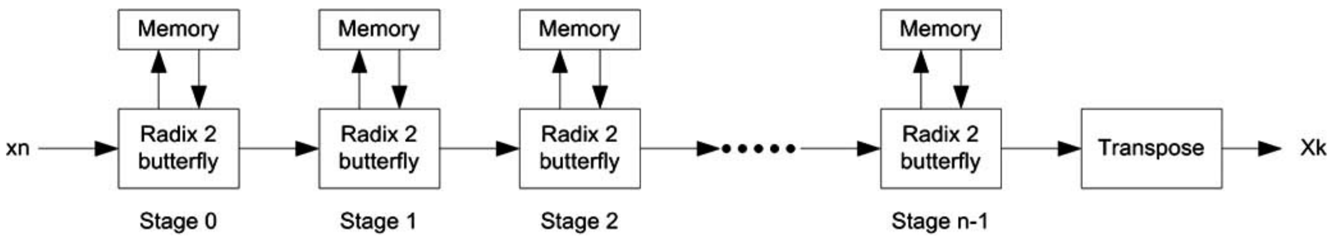

Fig. (5). Pipeline architecture of the 1D-FFT.

$$(a+bj)\,j = aj - b = -b + aj$$

Taking into account these features, the 1D-FFT architecture implementation is depicted in Fig. (**6**).

The swap-blocks arrange the data flow and preserve the pipeline feature. It consists of two multiplexers and two shift registers. These shift registers are implemented using look-up tables (LUT) in mode shift register (SRL16) for synchronization (Fig. **7**).

The system performs the calculation of the FFT with no scaling. The unscaled full-precision method was used to avoid error propagations. This option avoids overflow situations because output data have more bits than input data. Data precision at the output is:

$$output\ width = input\ width + \log_2\ points + 1 \qquad (4)$$

The number of bits on the output of the multipliers is much larger than the input and must be reduced to a manageable width. The output can be truncated by simply selecting the MSBs required from the filter. However, truncation introduces an undesirable DC data shift. Due to the nature of two's complement numbers, negative numbers become more negative and positive numbers also become more negative. The DC shift can be improved with the use of symmetric rounding stages (Fig. **6**). The periodic signals of the swap units, *op* signal, and the address generation for the twiddle memories are obtained through a counter module that acts like a control unit.

The pipeline 1D-FFT architecture design is completely parametrisable and portable. The VHDL module has three generics in order to obtain a standard module: *fwd_inv*, *data_width* and *lognpoints* (the logarithm in base 2 of the number of elements that fit with the number of stages in the 1-D FFT calculation). These generics then select direct or inverse transform, data value precision and the transform length.

A comparison has been carried out between our functional implementation and the Xilinx pipeline core [16].

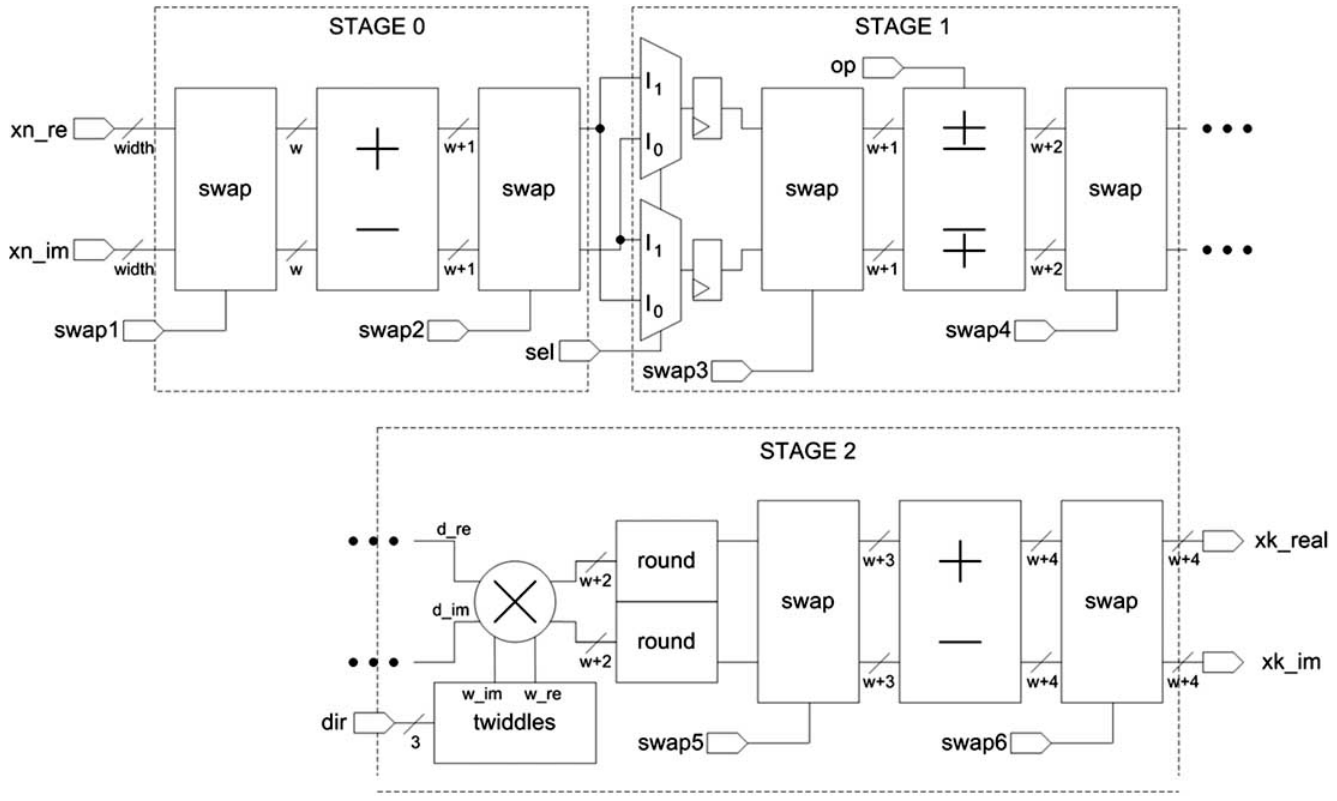

Fig. (6). Architectural block diagram of an 8-point radix-2 pipeline 1D-FFT.

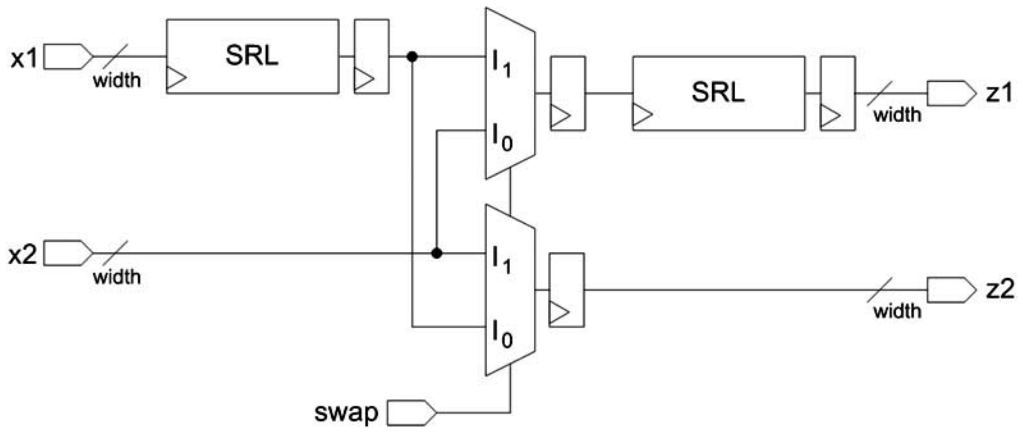

Fig. (7). Swap unit.

Table 1. Resources, Latency and Computation Time Using a Virtex-4 XC4VSX35. Data Sample Precision is 8-Bit

8 Points 1D-FFT	Xilinx FFT Core	Functional Implementation	Improvement
Slices	355	199	44%
Slices Flip-flops	632	371	41%
4-LUT	557	328	41%
DSP48s	10	4	60%
Latency (clock cycles)	50	32	36%
Fmax [MHz]	313.852	396.652	26%

Resources and computation time is showed in Table **1**. It can be clearly seen that our pipeline architecture outperforms the Xilinx core implementation. The reasons for the better system performance are the special consideration for the first and second stages and the unnecessary features that the Xilinx core implements.

4. 2-D FAST FOURIER TRANSFORM ON THE VIRTEX-4 FPGA

The fundamental operation in order to calculate the 2D-FFT is equivalent to doing a 1D-FFT on the rows of the matrix and then doing a 1D-FFT on the columns of the result. Traditionally, the parallel and pipeline algorithm is then implemented in the following four steps.

1. Compute the 1D-FFT for each row
2. Transpose the matrix
3. Compute the 1D-FFT for each row
4. Transpose the matrix

Fig. (**8**) depicts the diagram of the implemented transform module. The operation of the developed system takes place when image data is received in serial form by rows. These data are introduced in a block that carries out a one dimensional FFT. As this module obtains the transformed data, the information is stored in two double-port memories (real and imaginary data). To complete the bidimensional FFT, the stored data is introduced in a second 1D-FFT in column format. The 2D-FFT is then obtained from the output of this block.

It is worth mentioning that the transposition step defined in the above (step 2) is implemented simultaneously with the transfer of column data vector to the memory with no delay penalty. In this way, the counter acts as an address generation unit. The last transposition step (step 4) is not implemented in order to save resources and obtain a fast global system. So, the last transposition step is taken into account only at last of the algorithm described in Eq. (1) and (2).

Row 1D-FFT block and column 1D-FFT block are not identical due to the unscaled data precision. So, the desired 8x8 2D-FFT for the phase recoverer must meet certain requirements. If the precision of data input is 16 bits, the output data of 1D-FFT of the rows has to be 20 bits according Eq. (4). 1D-FFT of the columns accepts a 20 bits data format and 24 bits at the output.

The values that are calculated with the row 1D-FFT are stored continuously in a RAM memory (odd RAM, for example). This block actually contains two double-port RAMs, the first one for the real part of the data and the other for the imaginary part. A counter is used to address the memory in write mode. The 1D-FFT module supplies an index of data output, *k_index*, that is obtained using the counter/control unit. This index is extended by introducing additional bits in order to address all the elements of the image matrix. This counter is enabled when the *done* signal of the row FFT is activated. This signal indicates when the transformed data are available. The counter supplies the address for column data of the memory. These directions are obtained by commuting the most significant bits by the less significant ones.

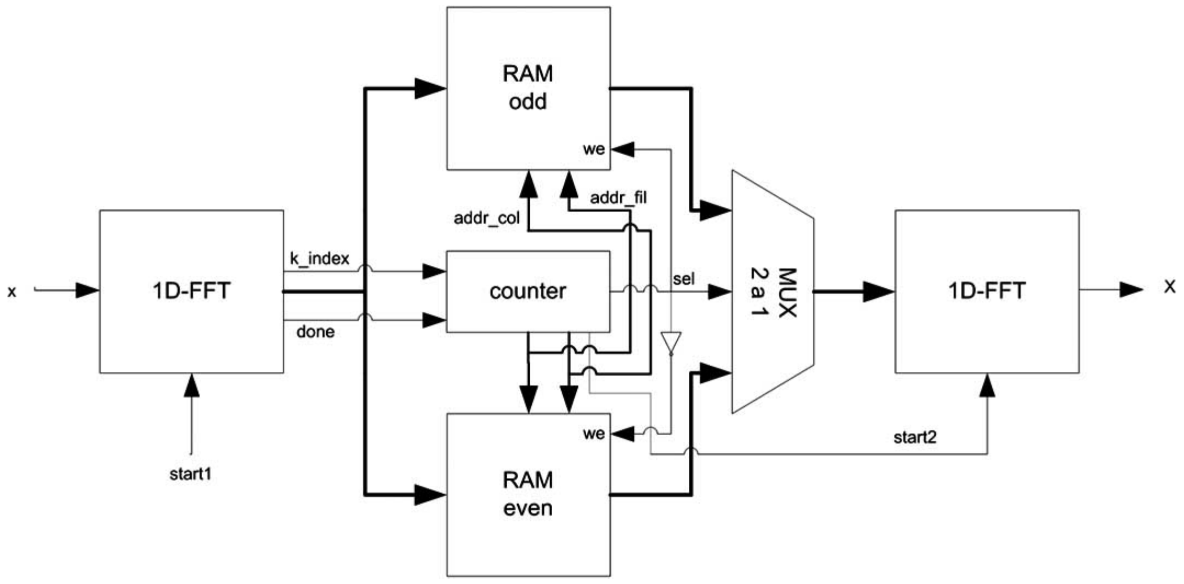

Fig. (8). Structural diagram of. the.implemented 2D-FFT.

Continuous data processing using a single dual-port memory (real and imaginary) is not possible. In that case, the new transformed data must wait for the old data to be introduced in the second FFT block. Otherwise data are overwritten. As a result, the pipeline property of the FFT architecture can not be used. This problem can be averted by using two memories instead of one, where memories are continuously commuting between write and read modes. When the odd memory is reading and introducing data values in the second FFT module, the even memory is writing data which arrives from the first FFT. So, data flow is continuous during all of the calculations in the bidimensional transform. The memory modes are always alternating and the function is selected by the counter. The same signal is used to commute the multiplexer that selects the data that enter the column transform unit.

The designed 2-D FFT module needs 162 clock cycles to calculate the 8x8 transform. This time is considered as latency time. Given that the system is fully pipeline, we obtain a new transform for each of the 64 clock cycles. The use of a 100 MHz clock produces 1620 ns and 640 ns, respectively.

Again, the module implemented is parametrizable using three generics: *fwd_inv*, *data_width* and *lognpoints*. Several FFTs were implemented over a XC4VSX35 Virtex-4 device and numerical results were satisfactorily compared with Matlab simulations. Resources in this FPGA include 15360 slices, 30720 flip-flops, 192 BRAM and 192 DSP48s. The syntheses were achieved by changing the image size and the data precision.

Table **2** shows the resource utilization to implement a pipeline 2D-FFT when the input data precision is 8 bits. The largest transform that we can synthesize is 128x128 points. Notice that the maximum frequency of operation decreases when the size of FFT increases.

The resource utilization is greater and the maximum frequency is smaller when the FFT has 16 bits precision due to the increase of the intrinsic complexity found at this level of precision (Table **3**).

As we show in Eqs. (1) and (2), three 2D-FFTs are needed to implement a pipeline wavefront phase reconstructor. It can only recover the phase with a sensor up to 64x64 subpupils using 8 bits precision and up to 32x32 with 16 bits precision. This size is sufficient at these moments for the prototype, however, if we want to implement a greater recoverer, we should select an FPGA with more resources or we should have to change the design architecture or use several FPGAs.

It can be seen that for the 128x128 2-D FFT, the implemented algorithm executes each operation in 170.84 µs, which implies a 5853 frame rate per second. This rate meets astronomical image processing frame rate requirements.

Table 2. XC4VSX35 Virtex-4 Resource Utilization Using an Input Data Precision of 8 Bits

2D-FFT 8 Bits	Slices	Slices Flip-Flops	RAMB16	DSP48s	Fmax	Clock Cycles	Duration [@ 100 MHz]
8x8	1141 (7%)	1958 (6%)	4 (2%)	8 (3%)	305.599	162	1620 ns
16x16	1667 (10%)	2839 (9%)	4 (2%)	8 (3%)	304.404	412	4120 ns
32x32	2855 (18%)	4517 (14%)	4 (2%)	15 (7%)	292.757	1304	13.04 µs
64x64	4613 (30%)	6318 (20%)	16 (8%)	15 (7%)	189.519	4516	45.16 µs
128x128	8135 (52%)	9755 (31%)	64 (33%)	25 (13%)	161.300	17084	170.84 µs

Tabla 3. XC4VSX35 Virtex-4 Resource Utilization Using an Input Data Precision of 16 Bits

2D-FFT 16 Bits	Slices	Slices Flip-Flops	RAMB16	DSP48s	Fmax	Clock Cycles	Duration [@ 100 MHz]
8x8	1650 (10%)	2809 (9%)	4 (2%)	10 (5%)	285.193	162	1620 ns
16x16	2302 (15%)	4076 (13%)	4 (2%)	10 (5%)	278.384	412	4120 ns
32x32	3992 (25%)	6315 (20%)	8 (4%)	22 (11%)	195.223	1304	13.04 μs
64x64	6472 (42%)	8870 (28%)	24 (12%)	22 (11%)	165.272	4516	45.16 μs
128x128	11326 (73%)	13522 (44%)	24 (12%)	36 (18%)	161.300	17084	170.84 μs

Table 4. 2D-FFT Performance Comparison with other Designs

2D-FFT	CPU Rodríguez-Ramos *et al.* [3]	GPU Rodríguez-Ramos *et al.* [3]	FPGA Uzun *et al.* [15]	FPGA Proposed
64x64	8733	633	-	22143
128x128	1233	592	420	5853

Table **4** shows a performance comparison of existing 2D-FFTs implementations using FPGA and others technologies in terms of frame rate per second (fps) for matrix sizes 64x64 and 128x128. Rodríguez-Ramos *et al.* [3] implemented 2D-FFT on a CPU AMD XP 3500+, 2211 GHz, with 512 KB L2 cache and on a GPU nVidia GeForce 7800 GTX graphical engine with 256 MB RAM. Uzun *et al.* [15] implemented several algorithms on a Virtex-2000E FPGA chip where the fastest design is depicted in the table. It can be seen that our design shows improvements when compared to [3] and [15] in terms of frame rate performance.

5. APPLICATION CASE STUDY: PHASE RECOVERER DESIGN

We will focus on the FPGA implementation from Eqs. (1) and (2) to improve processing time. These equations can be implemented using different architectures. We could choose a sequential architecture with a unique 2D-FFT module where data values use this module three times in order to calculate the phase. This architecture represents an example of an implementation using minimal resources of the FPGA. However, we are looking for a fast implementation of the equations in order to stay within the 6 ms limit of the atmospheric turbulence. Given these considerations we chose a parallel and completely pipeline architecture to implement the algorithm. Although the resources of the device increase considerably, we can maintain time restrictions by using extremely high-performance signal processing capability through parallelism. We therefore synthesize three 2D-FFTs instead of one 2D-FFT.

The block diagram of the designed recoverer is depicted in Fig. (**9**). For this implementation, we select a 16 bits input data precision for Sx and Sy values. Sx and Sy have 8x8

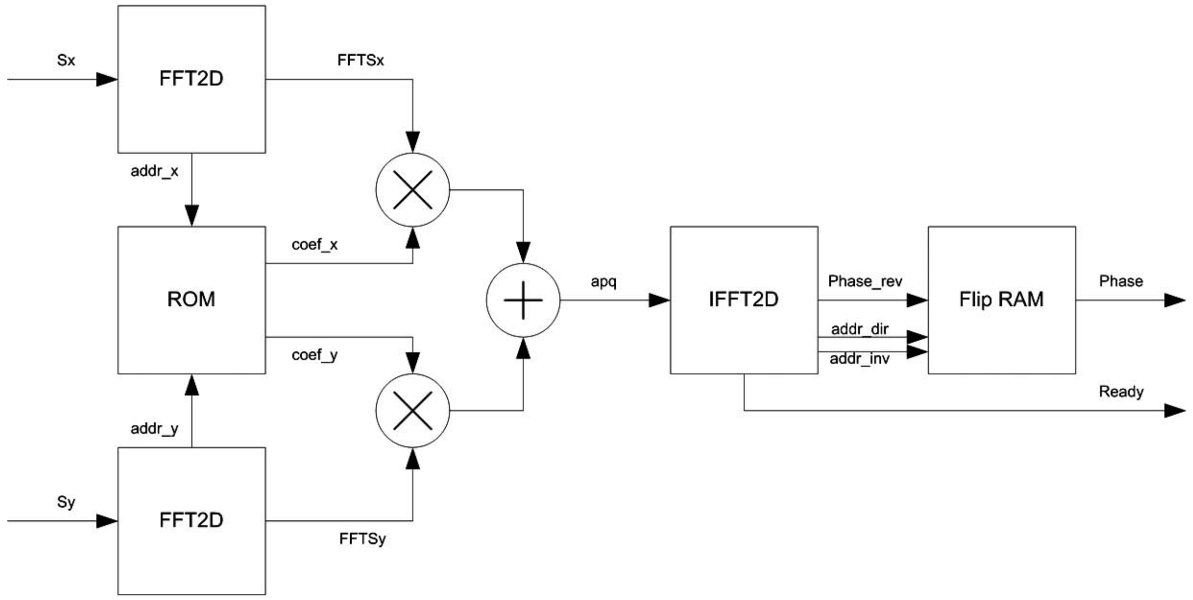

Fig. (9). Block diagram of the synthesized phase recoverer. The module was implemented on a XC4VSX35 Virtex-4 device. *Clock*, *reset* and chip enable (*ce*) signals have been omitted in order to simplify the diagram.

elements or pixels for each one. These pixels represent the image displacement into each subpupil.

An analysis of the equations and a parallel architecture of its implementation are taken into account. We then break down the design into the following steps or stages:

1. Compute two real forward 2D FFT that compute FFT(Sx) and FFT(Sy).
2. Compute the complex coefficients:

$$a_{pq} = \frac{ipFFT\left\{S^x(u,v)\right\} + iqFFT\left\{S^y(u,v)\right\}}{p^2+q^2}$$

3. Carry out a complex inverse 2D FFT on a_{pq}.
4. Flip data results.

The first step is a direct use of the proposed 2D FFT. To accelerate the process, the two real transforms are executed simultaneously through a parallel implementation. Observe how the two 2D-FFT of phase gradients Sx and Sy are multiplied by some constant factors according to the formula of the phase recoverer. An adder is necessary in the following stage to calculate the frequency coefficients and achieve an inversed 2D-FFT. Phase values are then transposed, which require an intermediate memory to properly show output data (flip-ram module). The direct 2D-FFTs are real, so the imaginary components are zero. The inversed transform allows complex input data.

The importance of the parallel execution of the Sx and Sy transforms in the algorithm needs mentioning. This feature allows both inputs to be simultaneously received and to obtain the calculation of both data just as in the previous case at the output of the 2-D FFT units.

The bidimensional transforms of Sx and Sy have to be multiplied by $\frac{ip}{p^2+q^2}$ and $\frac{iq}{p^2+q^2}$ respectively according to Eq. **1**. These two 8x8 points matrix are identical if we change rows by columns. We can therefore store a unique ROM (two 64x16 bits ROM, one ROM for the real component of the factor and the other for the imaginary part). The addresses of these ROMs are supplied by the 2D-FFT module through *cnt_fil* and *cnt_col* signals. These signals are obtained from a built-in counter in this module. The ROM module has two generics (*data_width* and *addr_width*) in order to synthesize a standard single-port memory of any size. However, every time we change these generics, we have to use mathematical software (Matlab in this case) in order to calculate the elements and initiate ROM memory.

There are two complex multipliers. Each one performs the complex multiplication of the constant stored in the ROM by the 2-D FFT result (previously rounding to 18 bits). The complex multiplication needs four DSP48 circuits. Inside of this circuit, the complex multiplication uses multipliers, internal registers and adders/subtractors. These modules are completely standard using the *a_width* and *b_width* generics that select the precisions of the signals to multiply.

The sum of the outputs of the complex multipliers is implemented using slices exclusively. We implemented two adders, one for the real components and other for the imaginary ones. This module is also configured with a generic parameter that supplies data precision. The results of the adders (*apq* coefficients) are rounded appropriately to obtain 16 bits data precision according with the data input width of the inversed bidimensional transform that is executed at the next stage.

The FLIP-RAM module synthesizes a double dual-port memory similar to the memories that were described in the 2D-FFT section. While a memory is in read mode, the other one is in write mode. With this consideration, the total implemented system continues being pipeline. The addressing of the memories is obtained in a similar form, through a counter that is included in the inversed 2D-FFT. In this case, the memories only store real data (data values of the recovered phase). This module is necessary because the phase data that the inversed 2D-FFT provides are disorderly. The implemented module is depicted in Fig. (**10**).

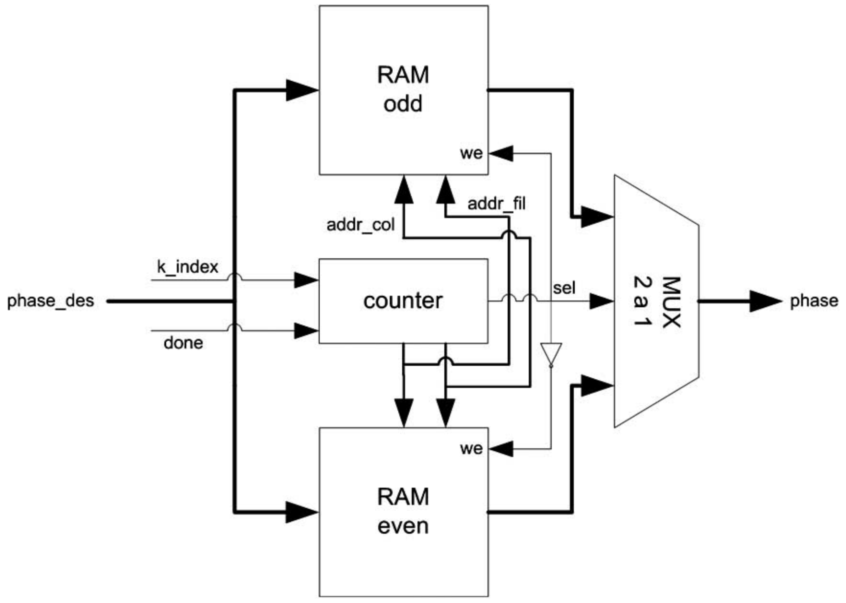

Fig. (10). Block diagram of the output stage of the wavefront recoverer (Flip-RAM).

6. RESULTS AND ANALYSIS

The wavefront recoverer prototype was synthesized with a Xilinx XC4VSX35 Virtex-4. This FPGA is provided in a ML402 Xtreme DSP evaluation platform. This development board has a 100 MHz clock. Our prototype was designed using ISE Foundation 8.2 and ModelSim simulator. Table **5** shows Virtex-4 resource utilization:

Table 5. Virtex-4 XC4VSX35 Resource Utilization to Implement the Phase Recoverer Prototype

Resource	Used/Available	Percentage
Slices	5128/15360	33%
Flip-flops	8716/30720	28%
Block RAM	18/192	9%
DSP48	38/192	19%

The design was programmed using the VHDL hardware description language [17] and XST was used to synthesize these modules [18]. The complete system was successfully tested in circuit using ChipScope Pro software that directly inserts a logic analyzer and bus analyzer into the design, allowing any internal signal to be viewed. Signals are captured at operating system speed and brought out through the programming interface. Captured signals can then be analyzed with the PC that acts as a logic analyzer. The numeric results were also successfully compared with those obtained in Matlab.

The system can process data in much lower times than the atmosphere response. This feature allows more phases to be introduced in the adaptive optic process. Table **6** shows the total time broken down into the stages of the total system. 396 clock cycles are necessary for phase recovery, starting with data reception to the activation of the *ready* signal. This value is the latency time for the phase recoverer. At a 100 MHz frequency clock, the system needs less than 4 μs to recover the phase.

The implemented architecture is pipeline. This architecture allows phase data to be obtained for each 64 clock cycles (this number coincides with the number of point of the transforms, that is, the number of subpupils, 8x8). Using the 100 MHz clock, the prototype provides new phase data each 640 ns. Data values are obtained with a throughput of 1.56 M frame per second.

Table 6. Execution Time (Latency) for the Different Stages of the Phase Recoverer

Module	Cycles	Duration (@ 100 MHz)
2D-FFT (Sx and Sy)	162	1620 ns
Multipliers	4	40 ns
Adder	1	10 ns
IFFT2D	162	1620 ns
Flip-RAM	64	640 ns
Rounding (3)	3	30 ns
Total	396	3960 ns

We carried out time estimates in order to consider the implementation of a phase recoverer for extremely large telescopes where the Shack-Hartmann sensor will have a greater number of subpupils. Only the 2D-FFT and flip-RAM modules are affected by the size of the sensor in Table **6**. For the 2D-FFT modules, the durations are given in Tables **2** and **3**. Clock cycles for the RAM coincide with the number of subpupils. Table **7** shows the temporal estimations of the different phase recoverers using FPGA and GPU calculated by Rodríguez-Ramos *et al.* [3]. We observed that for 256x256 subpupils latency is still below the 6ms limit of the atmosphere turbulence. Notice how the implementation based on FPGA is faster than the GPU solution. FPGA operation speed could be even faster if we use a greater frequency clock. This alternative is possible in Virtex-4 and Virtex-5 families. FPGA evolution is evidently an important aspect when considering improved recoverer design.

These results can be contrasted with other works. Baik *et al*. [19] implemented a wave reconstruction with a 14x14 Shack-Hartmann array, an IBM PC and a Matrox Meteor-2 MC image processing board. The wavefront correction speed of the total system was 0.2s. Although the system includes the gradient estimation, it can be seen that the execution times is slower than the proposed 16x16 implementation. Seifer *et al.* [20] used a sensor with 16x12 subpupils and a Pentium M, 1.5GHz. The wavefront reconstruction in this case was 50ms using Zernike polynomials to fit the coefficients of the aperture function. Again, our implementation using FPGA technology results faster than this one. To the best of our knowledge, there are not other wavefront reconstruction implementations based on FPGA technology.

Table 7. Execution Times of the Phase Recoverer for Different Sizes of Subpupils into the Virtex-4 XC4VSX35 and into the GPU nVidia GeForce 7800 GTX with 256 MB RAM [3]

Size (number subpupils)	Cycles	Time in FPGA [@ 100 MHz]	Time in GPU [3]
8x8	396	3.96 μs	
16x16	1088	10.88 μs	
32x32	3640	36.40 μs	
64x64	13136	131.4 μs	3.531 ms
128x128	50560	505.6 μs	3.875 ms
256x256	198664	1.987 ms	5.172 ms

7. CONCLUSIONS AND FUTURE WORK

A wavefront phase can be recovered from a Shack-Hartmann sensor using FPGA as exclusive computational resource. Wavefront phase recovery in an FPGA is an even more satisfying computational technique because recovery times result faster than GPU or CPU implementations.

The amount of used resources devices is the main disadvantage of the implementation into the Virtex-4 on the ML402 development board. We can only recover the phase up to 32x32 subpupils. Two direct 2D-FFT's are executed in parallel form and they could share some hardware resources (index, counters…). Two ad-hoc FFT's could be implemented for this stage of the algorithm. Furthermore, as they are real transforms, we could design a unique 2D-FFT to compute FFT[S_{xy}]=FFT[S_x+iS_y] and then obtain FFT[Sx] like $\frac{FFT[S_{xy}]+FFT[S_{xy}^*]}{2}$ and FFT[Sy] such as $\frac{FFT[S_{xy}]-FFT[S_{xy}^*]}{2i}$. Thus, we could save about 30% of the resources device.

We will soon implement a larger recoverer into a XC5VSX50T Virtex-5 that is faster and has more resources than the XC4VSX35 Virtex-4. The main features of this device are 52224 slices, 32640 slices flip-flops, 288 DSP48s and 264 BRAM blocks [21]. This Virtex-5 can be operating up to 550 MHz and it has significantly more DSP48 circuits in comparison with the Virtex-4 family. This way we can minimize slices utilization using more of these components. The prototypes could be four times faster than those developed with the Virtex-4 FPGA device.

Finally, the Virtex-5 device provides enough resources in order to implement all the phases in the adaptive optic process in the same circuit. This design includes the data acquisition of the camera through a camera-link module, the phase gradients estimations using a barycentre algorithm or with correlations, and a VGA interface module to monitor the recovered phase.

ACKNOWLEDGMENTS

This work has been partially supported by "Programa Nacional de Diseño y Producción Industrial" (Project DPI 2006-07906) of the "Ministerio de Ciencia y Tecnología" of the Spanish government, and by "European Regional Development Fund" (ERDF).

Received: November 18, 2008 Revised: December 20, 2008 Accepted: December 21, 2008

REFERENCES

[1] Poyneer, L.A.; Gavel, D.T.; Brase, J.M. Fast wave-front reconstruction in large adaptive optics systems with use of the Fourier transforms. *J. Opt. Soc. Am.,* **2002**, *A19,* 2100-2111.

[2] Roddier, F.; Roddier, C. Wavefront reconstruction using iterative Fourier transforms. *Appl. Opt*., **1991**, *30*, 1325-1327.

[3] Rodríguez-Ramos, J.M.; Marichal-Hernández, J.G.; Rosa, F. Modal Fourier wavefront reconstruction using graphics processing units. *J. Elect. Imag*., **2007**, *16*, 2.

[4] Meyer-Baese, U. *Digital Signal Processing with Field Programmable Gate Arrays*, Springer-Berlag, **2001**.

[5] Craven, S.; Athanas, P. Examinig the Viability of FPGA Supercomputing. *EURASIP J. Embed. Syst.*, **2007**, *2007*, 8.

[6] Deschamps, J.; Bioul, G.; Sutter, G. *Synthesis of Arithmetic Circuits. FPGA, ASIC and Embedded Systems,* Wiley-Interscience, **2006**.

[7] Proakis, J.G.; Manolakis, D.K. *Digital Signal Proccesing. Principles, Algorithms and Applications*, Prentice Hall, **1996**.

[8] Cooley, J.W.; Tukey, J.W. An algorithm for the machine calculation of complex Fourier series. *Math. Comput*., **1965**, *19*, 297-301.

[9] Zhou, Y.; Noras, J.M.; Shepherd, S.J. Novel design of multiplierless FFT processors. *Signal Proc*., **2007**, *87*, 1402-1407.

[10] Chang T.S.; Jen C.W. Hardware efficient transform designs with cyclic formulation and subexpression sharing. *Proc. 1998 IEEE ISCAS*, **1998**, *2*, 398-401.

[11] Guo, J.I. An efficient parallel adder based design for one dimensional discrete Fourier transfor. *Proc. Natl. Sci. Counc. ROC*, **2000**, *24*, 195-204.

[12] Chien, C.D.; Lin, C.C.; Yang, C.H.; Guo, J.I. Design and realization of a new hardware efficient IP core for the 1-D discrete Fourier transform. *IEE Proc. Circuits Dev. Syst.*, **2005**, *152*, 247-258.

[13] Hawkes, G.C. *DSP: Designing for Optimal Results. High-Performance DSP Using Virtex-4 FPGAs*, Xilinx, **2005**.

[14] Xilinx, *XtremeDSP for Virtex-4 FPGAs user guide*, Xilinx, **2006**.

[15] Uzun, I.S.; Amira, A.; Bouridane, A. FPGA implementations of fast Fourier transforms for real-time signal and image processing. *IEE Proc.- Vis. Image Signal Proc.*, **2005**, *152*, 283-296.

[16] Xilinx, *Fast Fourier Transform v3.2*, Xilinx, **2006**.

[17] IEEE, *Standard VHDL Language Reference Manual, IEEE-1076-2000*, IEEE, **2000**.

[18] Xilinx, *XST User Guide*, Xilinx, **2006**, pp. 118-217.

[19] Baik, S.H.; Park, S.K.; Kim, C.J.; Cha B.A center detection algorithm for Shack–Hartmann wavefront sensor. *Opt. Laser Technol*., **2007**, *39*, 262-267.

[20] Seifert, L.; Tiziani, H.J.; Osten, W. Wavefront reconstruction with the adaptive Shack–Hartmann sensor, *Opt. Commun*., **2005**, *245*, 255-269.

[21] Xilinx, *Virtex-5 Family Overview. LX, LXT and SXT Platforms,* Xilinx, **2007**, pp. 2-6.