2D-FFT implementation on FPGA for wavefront phase recovery from the CAFADIS camera

J. M. Rodríguez-Ramos*, E. Magdaleno Castelló, C. Domínguez Conde, M. Rodríguez Valido, and J.G. Marichal-Hernández
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas;
University of La Laguna, Avda. Francisco Sánchez s/n. 38203, Spain

ABSTRACT

The CAFADIS camera is a new sensor patented by Universidad de La Laguna (Canary Islands, Spain): international patent PCT/ES2007/000046 (WIPO publication number WO/2007/082975). It can measure the wavefront phase and the distance to the light source at the same time in a real time process. It uses specialized hardware: Graphical Processing Units (GPUs) and Field Programmable Gates Arrays (FPGAs). These two kinds of electronic hardware present an architecture capable of handling the sensor output stream in a massively parallel approach. Of course, FPGAs are faster than GPUs, this is why it is worth it using FPGAs integer arithmetic instead of GPUs floating point arithmetic.

GPUs must not be forgotten, as we have shown in previous papers, they are efficient enough to resolve several problems for AO in Extremely Large Telescopes (ELTs) in terms of time processing requirements; in addition, the GPUs show a widening gap in computing speed relative to CPUs. They are much more powerful in order to implement AO simulation than common software packages running on top of CPUs.

Our paper shows an FPGA implementation of the wavefront phase recovery algorithm using the CAFADIS camera. This is done in two steps: the estimation of the telescope pupil gradients from the telescope focus image, and then the very novelty 2D-FFT over the FPGA. Time processing results are compared to our GPU implementation. In fact, what we are doing is a comparison between the two different arithmetic mentioned above, then we are helping to answer about the viability of the FPGAs for AO in the ELTs.

Keywords: Adaptive optics, wavefront phase recovery, FPGA, GPU real-time processing.

1. INTRODUCTION

Next generation of extremely large telescopes (50 up to 100 meter of diameter) will demand big technological advances to maintain aligned the segments of the telescopes (phasing of segmented mirrors) and then to correct from atmospheric aberrations. Adaptive optics includes several steps: detection, wavefront phase recovery, information transmission to the actuators and mechanical movements of them. In order to help on this, a quicker wavefront phase reconstruction seems to be really relevant, and new wavefront sensors designs must be explored. The CAFADIS camera presents a robust optical design that can accomplish the Adaptive Optics objectives even for extended objects as reference (elongated LGS and solar observations). The CAFADIS camera is an intermediate sensor between the Shack-Hartmann and the pyramid sensors. It samples an image plane using a microlens array. The pupil phase gradients can be obtained from there, and after that, the problem for phase recovery is the same as in the Shack-Hartman.

When using complex exponential polynomials, very fast modal algorithms can be implemented because the kernel of the Fast Fourier Transform (FFT) is the same[1,2]. The modal algorithms produce more precise results than the zonal solution and it is the estimation selected by Rodriguez-Ramos et al. [3] to accomplish the phase reconstruction.

Until recently, the problem of the design of an efficient phase recoverer has been implemented over a GPU platform with satisfactory execution time results[3]. GPUs provide a great computation power in a low cost broadly used peripheral but the results are not optimized due to the rigid architecture of these device. A FPGA prototype can accelerate the calculation due to the architecture of this device. In this way, FPGA technology offer extremely high-performance signal processing capability through parallelism based on slices and arithmetic circuits and high flexible interconnection possibilities[4]. Moreover, FPGA technology is an alternative to custom ICs (integrated circuits) for implementing logic.

*jmramos@ull.es; phone +34 922 318 249; fax +34 922318228.

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 short time[5]. With a clear idea of the FPGA technology features, we propose a 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 over a Virtex-4 FPGA platform, paving the road to accomplish the ELT's number of actuators computational requirements within a 6 ms limit. The wavefront phase recovery algorithm, using the CAFADIS camera, is done in two steps: the estimation of the telescope pupil gradients from the telescope focus image, and then the very novelty 2D-FFT over the FPGA.

The remainder of the paper is structured as follows. Section 2 outlines a descriptive approach of the mathematical algorithms implemented. A description of the implemented system is discussed in section 3. In Section 4 we analyse the results obtained for the wave-front phase recovery from our CAFADIS camera. Conclusions and future work are shown in Section 5.

2. A DESCRIPTIVE APPROACH

A wave-front sensing scheme based on placing a subpupil array at the focal plane of the telescope with each subpupil reimaging the aperture is analyzed by Clare and Lane[6]. This wave-front sensing arrangement is the dual of the Shack–Hartmann sensor, with the wave front partitioned in the focal plane rather than in the aperture plane. This arrangement can be viewed as the generalization of the pyramid sensor. In their paper, they propose that the partial derivatives of the pupil wave-front aberration with respect to ξ and η coordinates for an MxN array of subpupil are given by two gradient formulas which are the generalization of those derived by Ragazzoni for the pyramid wave-front sensor[6]:

$$\frac{\partial\phi(\xi,\eta)}{\partial\eta}=\frac{\sum_{m=-M/2}^{M/2-1}\sum_{n=-N/2}^{N/2-1}(nd-\delta_v)I_{m,n}(\xi,\eta)}{\sum_{m=-M/2}^{M/2-1}\sum_{n=-N/2}^{N/2-1}I_{m,n}(\xi,\eta)}=S_x \quad , \quad \frac{\partial\phi(\xi,\eta)}{\partial\xi}=\frac{\sum_{m=-M/2}^{M/2-1}\sum_{n=-N/2}^{N/2-1}(md-\delta_u)I_{m,n}(\xi,\eta)}{\sum_{m=-M/2}^{M/2-1}\sum_{n=-N/2}^{N/2-1}I_{m,n}(\xi,\eta)}=S_y \qquad (1)$$

where d is the size of lenslets in the array, (δ_u,δ_v) is the displacement of the central subpupil from the origin in the u and v directions and $I_{m,n}(\xi,\eta)$ is the aperture image formed from the $(m,n)_{th}$ subpupil in the focal plane given by:

$$I_{m,n}(\xi,\eta)=\left|d^2\operatorname{sinc}(\xi d)\exp[j2\pi\xi(md-\delta_u)]\operatorname{sinc}(\eta d)\exp[j2\pi\eta(nd-\delta_v)]\otimes P(\xi,\eta)\exp[j\phi(\xi,\eta)]\right|^2 \qquad (2)$$

Therefore, the algorithm depends on subpupil position into image and every pixel into every subpupil. In order to implement the phase recovery from an aperture image on a FPGA, it is necessary to get a high level of parallelism as far as possible, using the FPGA features, and to speed up all calculus. Therefore, some operations can be made at the same time to estimate the pupil gradients, as they are independents among themselves. The calculus of the denominators is the same for both gradients, and it is independent of the numerators calculus, which are independents between themselves too.

From these gradients estimation, the wavefront phase $\phi(\xi,\eta)$ can be recovered using an expansion over complex exponential polynomials:

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

The gradient is then written:

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

Making a least squares fit over the F function:

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

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

$$a_{pq} = \frac{ipFFT\{S^x(\xi,\eta)\} + iqFFT\{S^y(\xi,\eta)\}}{p^2 + q^2} \quad (3)$$

And finally the phase can be recovered from the gradient data by transforming backward those coefficients:

$$\phi(\xi,\eta) = FFT^{-1}[a_{pq}] \quad (4)$$

So a filter composed by three Fourier transforms must be done to recover the phase. Because of the telescope annular mask, the Fourier transform introduces big spatial frequencies that produce wrong recovered phases. This can be avoided using a padding different from zero and coherent with the definition of gradient[1,2] although the latter implies much more time consuming operations. To accelerate the process, an exhaustive study of the FFT algorithm has been crucial and has allowed to specifically adapt the FFT to the modal wavefront recovery pipeline and the FPGA architecture.

3. SYSTEM ARCHITECTURE IMPLEMENTATION

The global system implemented is depicted in figure 1. The FPGA implementation has three sub-modules. At the front of the system the camera link module receives data from CAFADIS in a serial mode. The following stages perform the digital data processing using the FPGA resources. So, the implemented mathematical algorithm could be divided in two stages: the estimation of the phase gradients from the camera-link data and the calculation of the wavefront phase.

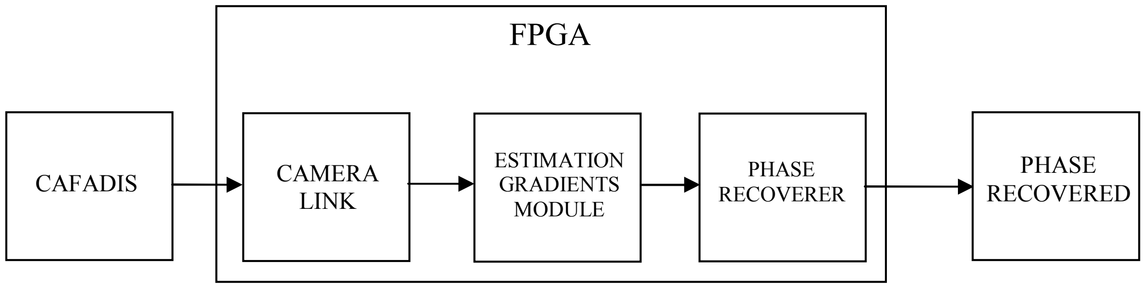


Fig. 1. Complete system.

3.1 Module for estimation of the telescope pupil gradients

In previous section, mathematical resources were shown to obtain the estimation of the telescope pupil gradients. This section exposes the implementation on FPGA of the equation (1). First of all, it is necessary to detail some features of the full design. Pixels of CAFADIS image are received using the camera link standard, that is, the first module receives serial data with a frequency fixed by the camera. The natural order of pixels reception is by rows, that means, the system must be ready to use and store them in the correct way. A block diagram of this module is depicted in figure 2.

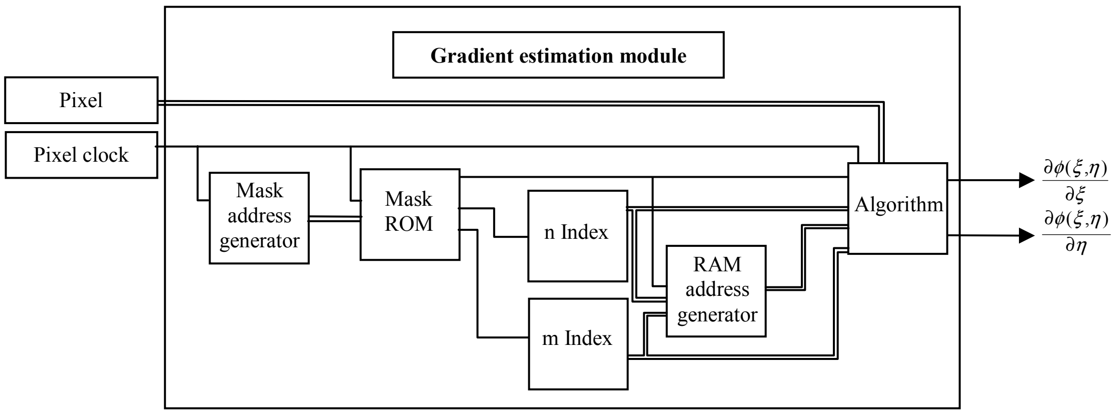

Gradient estimation module
Pixel
Pixel clock
Mask address generator
Mask ROM
n Index
m Index
RAM address generator
Algorithm
$\frac{\partial\phi(\xi,\eta)}{\partial\xi}$
$\frac{\partial\phi(\xi,\eta)}{\partial\eta}$

Fig. 2. Conceptual design of the calibration module.

In order to tidy received data, we use a mask. This mask has the structure shown in table 1. Bit number 0 reports that there is a data valid and enables the module. The function of bit number 1 is to indicate there is an N subpupil increment. Finally, bit number 2 allows an M subpupil change.

Bit 2	**Bit 1**	**Bit 0**
m increment	n increment	Data valid

Table 1: Mask structure.

The mask is stored in a ROM memory and the address to this memory is generated with a counter. The increment of counter takes place when a rising edge from camera clock happens. It is created like a VHDL module using MATLAB to generate the VHDL file.

The blocks named “n index” and “m index” (figure 2) generate the relative place of the subpupil into image. Those values are computed using a counter and there is a count increment when mask bit 1 (in case of n) or mask bit 2 (in case of m) is asserted. The counts of counters have as range $[-N/2+1, N/2]$ in case of index n and $[-M/2+1, M/2]$ in case of index m.

The algorithm block depicted in figure 2 is the core of this module. It carries out tasks necessary to work out all correct results. As shown figure 3, the architecture implemented to this block is a pipeline[7], so every rising edge of pixel clock there is a displacement of data from one stage to next stage and this happens for all stages at the same time. The stages of this architecture are interconnected with a register that holds data stable until next rising edge of clock. There are some signals which are propagated through all stages, the mask and divisor enable. Divisor enable is asserted only when a pixel belongs to last subpupil of the received image.

The main task of this block is to calculate three summatories, whose calculated values are stored in a memory cell. The addresses to store the partial results are obtained with a counter. The counter allows the correct subpupil change in order to obtain the correct partial summatory and store it in the right memory position to tidy results when necessary (see in figure 2 the RAM address generator).

There are two parallel tasks at first stage, one memory access to recover the appropriate partial summatory from RAM memory and the computation of $W_n = \left((nd-\delta_v)I_{m,n}(\xi,\eta)\right)$ and $W_m = \left((md-\delta_u)I_{m,n}(,\eta,)\right)$. Each RAM memory is implemented with dual port memory core from Xilinx[8]. It allows to read and write the memory simultaneously. The reading is done in the first stage and the writing is done at third stage. The data at first stage which pass to next stage are: partial sums read from memories, W_n, W_m and $I_{m,n}(\xi,\eta)$.

At the second pipeline stage, the system makes three parallel sums, between the partial summatories read from memories and new pixel data, in case of denominator, or the calculated weights in other case. Third pipeline stage consists in three multiplexers to initialize the adequate memory cell and to perform division in case divisor enable is asserted. Otherwise, results of previous stage are written in memory.

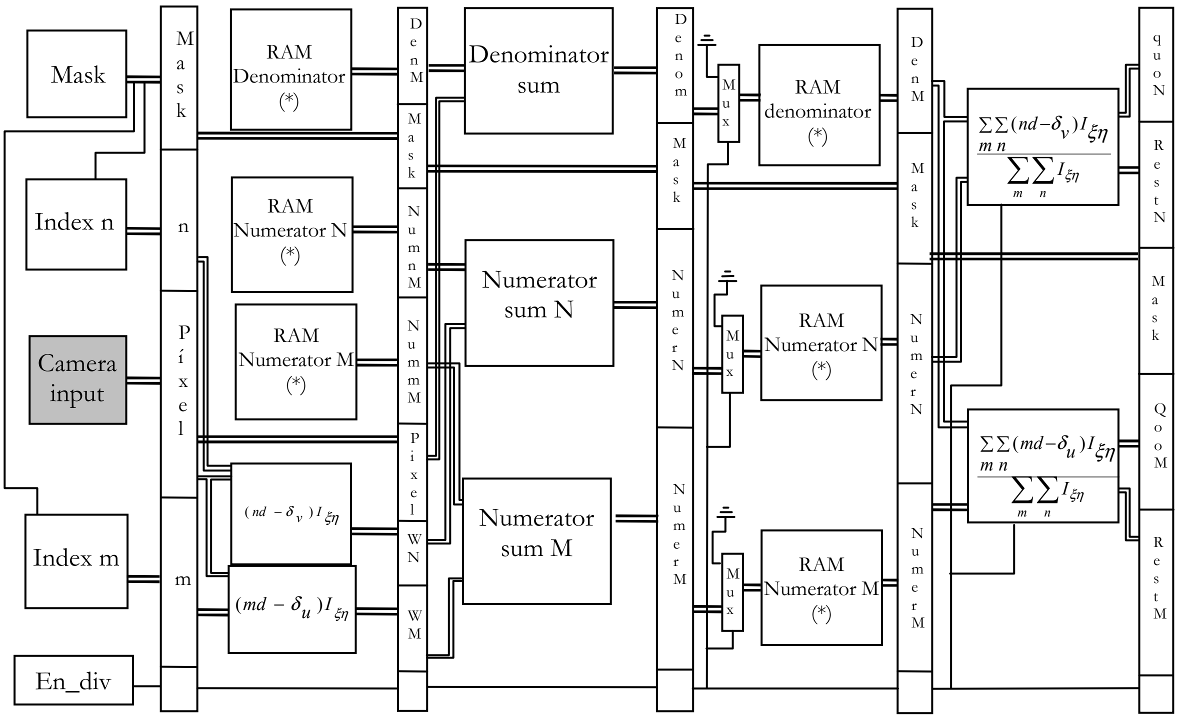
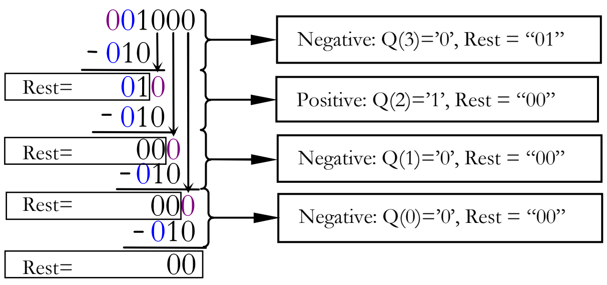


Fig. 3. Block diagram of Pipeline. (*) Memories in first stage are the same than memories in third stage.

The last stage implements the division by restoration algorithm (figure 4). The system needs to work with highest possible system clock frequency. In order to get it, at last stage the divisor groups in eight stages and computes the results of division. Each stage provides to the next stage the rest and two bits of every quotient (Q).



001000
- 010
Rest= 010
- 010
Rest= 000
- 010
Rest= 000
- 010
Rest= 00
Negative: Q(3)='0', Rest = "01"
Positive: Q(2)='1', Rest = "00"
Negative: Q(1)='0', Rest = "00"
Negative: Q(0)='0', Rest = "00"

Fig. 4. Division algorithm.

3.2 The phase recoverer

From the gradient estimations we have to obtain the phase data (figure 1) and equations 3 and 4. In order to implement the phase recover block, we could choose a sequential architecture with a unique 2D-FFT module where data values use three times this module in order to calculate the phase. This architecture represents an example of an implementation using minimal resources of the FPGA. However, we search a fast implementation of the equations in order to not exceed the 6 ms limit of the atmosphere turbulence, so we have chosen a parallel architecture to implement the algorithm. Although the resources of the device increase considerably, we insure to keep the temporal restriction using extremely high-performance signal processing capability through parallelism. With this consideration in mind, we synthesize three 2D-FFT instead only one.

The block diagram of the implemented architecture is depicted in figure 5. For this implementation, we select a 16 bits input data precision for Sx and Sy values. Sx and Sy are 8x8 elements arrays. These pixels represent the image displacement into each subpupil.

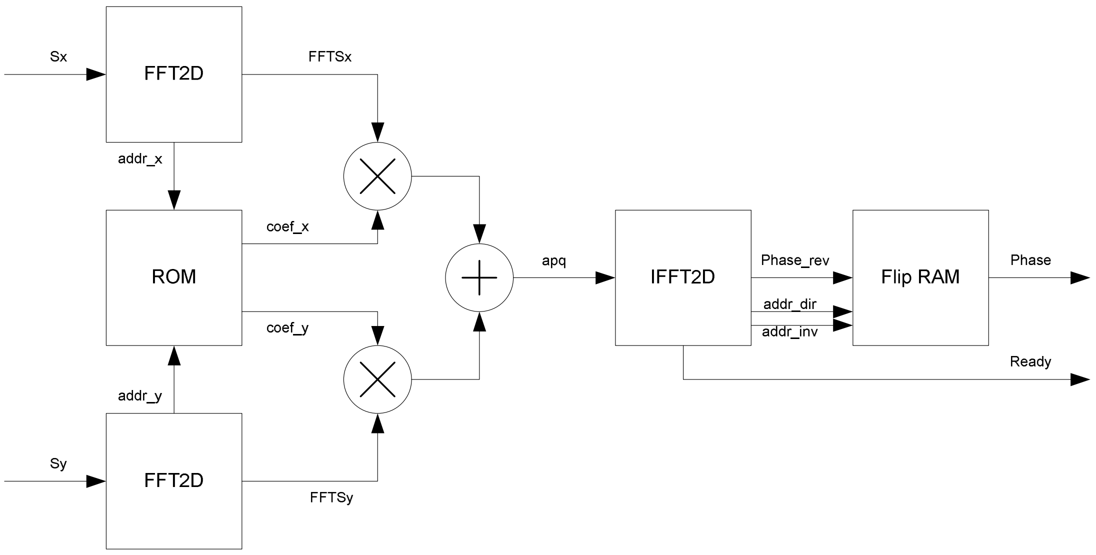


Fig. 5. Block diagram of the synthesized phase recoverer. The module was implemented on a XC4VSX35 Virtex-4 device. Clock, reset and chip enable (ce) signals have been omitted in order to simplify the diagram.

Taking into account the analysis of the equations and a parallel architecture for its implementation, we have broken down the design in the following steps or stages:

1. Two real forward 2D FFT that compute FFT(Sx) and FFT(Sy).
2. The spatial filtering to compute the coefficients:

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

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

The first step above is a direct use of the 2D FFT proposed. To accelerate the process, the two real transforms are executed simultaneously through a parallel implementation. As we can notice, the two 2D-FFT of the Sx and Sy phase gradients are multiplied by some constant factors according to the phase recovery formula. An adder is necessary in the following stage to calculate the frequency coefficients and to achieve an inversed 2D-FFT. At last, the phase values are transmuted, so we need an intermediate memory to show the output data properly.

We would like to emphasize that the S_x and S_y transforms are parallel executed. This allows to receive both inputs simultaneously and to obtain the calculation of both data just like in the previous case at the output of the 2-D FFT units.

The modules of the synthesized prototype are described below. 2-D FFT blocks were implemented using structural VHDL and the others modules were implemented using functional VHDL[4]. XST was used to synthesize these modules[5].

- **2D-FFT**

In figure 6, the diagram of the implemented 2D transform module is depicted. The operation of the developed system is as follow: 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 are introduced in a second 1D-FFT in column format. Finally, at this block output the 2D-FFT is obtained.

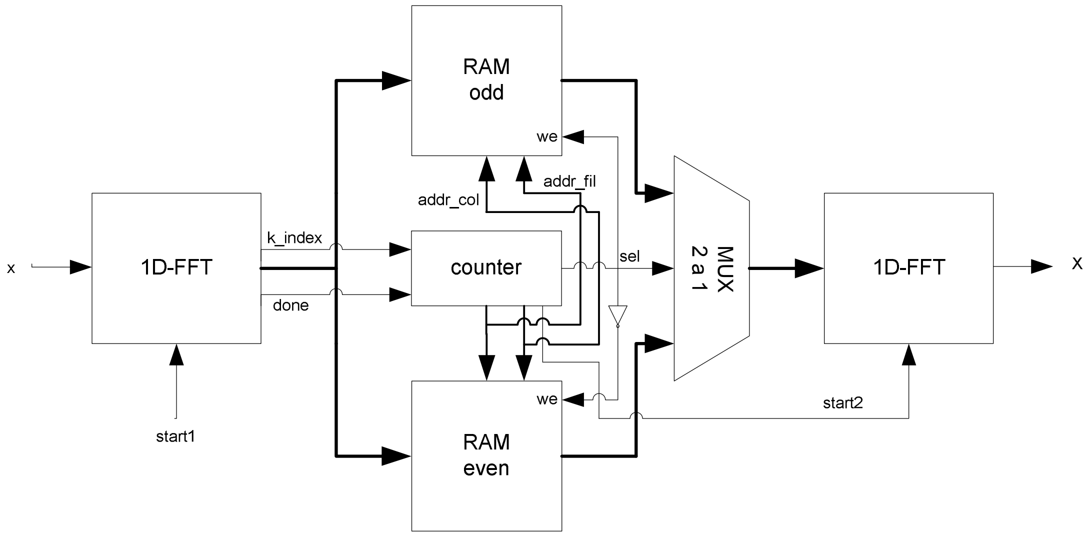


Fig. 6. Diagram of the implemented 2D-FFT

The FFT blocks were designed using a core that supplies Xilinx to implement one-dimension transforms[9]. Three architecture options are available with this core: radix-2 (minimum resources), radix-4 (that have a load/unload phase and a processing phase) and the pipeline architecture. All the options calculate the 1D-FFT using the Cooley and Tukey[10] algorithm. We have chosen the pipeline architecture because it is the only option that allows continuous data processing even though it uses more logic resources. The user can stream in input data and, after the calculation latency, can continuously unload the results. In this way, for the pipeline transform, the number of clock cycles to calculate the transform is the number of points of the FFT.

The module supports scaling at every stage of the transform using a fixed-scaling schedule because of the values can experience a certain intrinsic growth due to the butterfly calculations[9]. However, in this implementation we have preferred to perform the calculations with no scaling and the unscaled full-precision method was used to avoid error propagations. Using this option the overflow situations are avoid because the output data have more bits than the input data. Taking this into account, the row FFT block and the column FFT block are not identical due to the data precision.

The values calculated by the row FFT block are stored continuously in a RAM memory (odd RAM, for example). In fact, this block contains two double-port RAMs, the first one for the real part of the data and the other one for the imaginary part. A counter is used to address the memory in write mode. The Xilinx FFT core supplies an index for data output, named *k_index*. This index is extended adding 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. To obtain these addresses, we commute the most significant bits by the less significant ones.

It is not possible a continuous processing of the data using a single dual-port memory (real and imaginary). In that case, the new transform data must wait until the old data stored in the memory were introduced in the second FFT block. In other case, the data are overwritten. So, the pipeline property of the FFT architecture can not be used. This problem is solved using two memories instead only one. The 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 to come from the first FFT. So, data flow is continuous during all the calculations of the bidimensional transform. The modes of the memories are continuously alternating and the function is selected by the counter. The same signal is used to commute the multiplexer that select the income data into the column transform unit.

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

2D-FFT 8 bits	Slices	Slices Flip-Flops	RAMB16	DSP48s	Fmax	Clock cycles	Duration [@ 100 MHz]
8x8	1141 (7%)	1958 (6%)	4 (2%)	6 (3%)	305.599	162	1620 ns
16x16	1667 (10%)	2839 (9%)	4 (2%)	6 (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

Table 2. XC4VSX35 Virtex-4 Resource utilization using an input data precision of 8 bits.

The resource utilization is greater and the maximum frequency is smaller when the FFT has a 16 bits precision due to the increase of the intrinsic complexity with this precision (table 3).

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 3. XC4VSX35 Virtex-4 Resource utilization using an input data precision of 16 bits.

As we show in (1) and (2), it is necessary three 2D-FFTs to implement a pipeline wavefront phase reconstructor. So, it only can 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 enough at these moments for the prototype, but if we want to implement a greater recoverer, we should select a FPGA with more resources or we should have to change the architecture of the design.

It can be seen that for the 128x128 2-D FFT, the implemented algorithm executes each operation in 170.84 μs, that implies 5853 frame rate per second, which meets astronomical image processing frame rate requirements. This contrasts with the analysis carried out by Uzun et al. for parallel 2-D FFT for different architectures using a Virtex-2000E[11]. The faster 128x128 implementation has a performance of 420 fps. So, our architecture results an improvement in terms of speed.

- **The other components**

The design has three 8x8 points 2-D FFTs using a precision of 16 bits for phase gradients. The three blocks in the figure 6 are identical and they have a generic value (*fwd_inv*) that determines if the module is an inversed or directed transform. Direct transform are real, so the imaginary components are zero. The inversed transform allows complex input data.

The bidimensional transforms of Sx and Sy have to multiply by $\frac{ip}{p^2+q^2}$ and $\frac{iq}{p^2+q^2}$ respectively according eq. (1). These two 8x8 points matrix are identical if we change rows by columns. So, we can store a unique ROM (really two 64x16 bits ROM, a ROM for the real part of the factor and the other for the imaginary part).

The outputs at the two complex multipliers are added using an adder implemented using slices exclusively. This adder sums real an imaginary parts.

The FLIP-RAM module synthesizes a double dual-port memory like the memories that were described in the bidimensional transform section. This module is necessary because of the disordered phase data provided by the inverse 2D-FFT.

4. RESULTS

We have implemented a system to recover the wave-front phase from the CAFADIS camera. This system is organized in two blocks. The first one estimates the phase gradients from pixels of the incoming camera image. The second achieves the objective, the phase recovery. These blocks have been evaluated and synthesized in a XILINX XC4VSX35 Virtex-4. This FPGA is provided in a ML402 XTREME DSP evaluation platform. The prototype has been designed using ISE Foundation 8.2 and MODELSIM simulator.

The design was programmed using the VHDL hardware description language (except Xilinx 1-D FFT core). All system was successfully tested in circuit using ChipScope Pro software. It inserts logic analyzer and bus analyzer directly into the design, allowing to view any internal signal. 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. Also, the numeric results were successfully compared with the same obtained in Matlab.

Table 4 shows Virtex-4 resource utilization of the complete system. Most of the FPGA resources are consumed by the phase recoverer unit because it has three 2D-FFT modules. The rest of the mathematical components of the design (memories, adders, divisors) are simpler and they need less hardware logic.

Resource	Used/Available	Percentage
Slices	6218/15360	40%
Flip-flops	9681/30720	32%
Block RAM	21/192	11%
DSP48	38/192	21%

Table 4. Virtex-4 XC4VSX35 resource utilization to implement the complete system, taking in account an image size is 128x128 pixels and subpupil size is 8x8 in this implementation.

Module	Cycles	Duration (@ 100 MHz)
Gradient estimation	15480	154.80 μs
Phase recoverer	396	3.96 μs
Total	**15876**	**158.76 μs**

Table 5. Execution time for the first data valid

Table 5 shows the spent time in starting the results transmission. The number of clock cycles for gradient estimation coincides to the cycles needed to generate the first output which is input to phase recoverer unit. The system can process data in very lower times than the atmosphere response. This allows to introduce more phases in the adaptive optics process. The Following subsections will explain certain special features of each module.

4.1 Analysis of the phase recoverer

Table 6 shows the total time broken down into the stages of the phase recoverer unit. 396 clock cycles are necessary to recuperate the phase, from the receive data beginning to the activated ready signal. This value is the latency time for the phase recoverer. At 100 MHz frequency clock, the system needs less than 4 μs to recover the phase.

Table 6 Execution time (latency) for the different stages of the phase recoverer

Module	Cycles	Duration (@ 100 MHz)
2D-FFT (S_x and S_y)	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**

The implemented architecture is pipeline. That implies we can obtain phase data every 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. So, we obtain data values with a throughput of 1.56M frame per second.

We have carried out a time estimation in order to consider the implementation of a phase recoverer for extremely large telescopes where the Shack-Hartmann sensor will have more number of subpupils. Only 2D-FFT and flip-RAM modules are affected by the size of the sensor in table 4. For 2D-FFT modules, the durations are given in tables 1 and 2. Clock cycles for the RAM coincide with the number of subpupils. Table 7 shows a temporal estimation comparison between several phase recoverers using FPGA and GPU (Graphical Processing Unit) calculated by Rodríguez-Ramos *et al*[3]. We can observe that for a 256x256 subpupils, the latency is yet below of the 6ms atmospheric characteristic time. Notice that the implementation based on FPGA results faster than the GPU solution. FPGA operation speed could be even faster if we use a greater frequency clock. This way is possible in Virtex-4 and Virtex-5 families. So, FPGA evolution is an important aspect to consider into an improved recoverer.

Size (gradient data)	**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

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].

5. CONCLUSIONS AND FUTURE WORK

An FPGA phase recoverer implementation is presented. This is a solution for our CAFADIS camera, but it is also solution for the Shack-Hartmann wavefront sensor: the pupil wavefront phase is obtained from there pupil gradients. The designed phase recoverer carries out the calculations inside the atmospheric characteristic time using really high sampling. Results for several input data precision are shown. A bidimensional FFT is implemented over an FPGA architecture as nuclei algorithm of the recoverer: processing times are really short. We conclude that the FPGAs integer arithmetic is not a drawback. Then, the viability of the FPGAs for AO in the ELTs is assured.

Even, the quantity of used resources devices could be reduced taking into account mathematical properties. In this way, two direct 2D-FFT are executed in parallel form and they could share some hardware resources (index, counters…). So, we could implement two ad-hoc FFT 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 $FFT[S_x]$ and $FFT[S_y]$ like:

$$FFT[S_x] = \frac{FFT[S_{xy}]+FFT[S^*_{xy}]}{2} \qquad FFT[S_y] = \frac{FFT[S_{xy}]-FFT[S^*_{xy}]}{2i}.$$

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[13]. This Virtex-5 can be operating up to 550 MHz and it has a lot of DSP48 circuits in comparison with the Virtex-4 family. So, we can minimize slices utilization using more of these components. The prototypes could be four times faster than with Virtex-4 FPGA device.

We will also implement an FPGA phase recoverer for the CAFADIS camera using very extended objects as reference.

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).

REFERENCES

[1] L. A. Poyneer, D. T. Gavel, and J. M. Brase, “Fast wave-front reconstruction in large adaptive optics systems with use of the Fourier transforms”, J. Opt. Soc. Am. A 19, pp. 2100-2111, 2002.

[2] F. Roddier and C. Roddier, “Wavefront reconstruction using iterative Fourier transforms”, Applied Optics 30, pp. 1325-1327, 1991.

[3] J. M. Rodríguez-Ramos, J. G. Marichal-Hernández, F. Rosa, “Modal Fourier wavefront reconstruction using graphics processing units”, Journal of Electronic Imaging, vol. 16, issue 2, June 2007.

[4] IEEE Standard VHDL Language Reference Manual, IEEE-1076-2000, 2000. 11.

[5] Xilinx, “XST User Guide”, pp. 118-217, 2006.

[6] R. M. Clare and R. G. Lane, “Wave-front sensing from subdivision of the focal plane with a lenslet array”, J. Opt.Soc. Am. A/Vol. 22, No. 1, pp. 117-125, 2005.

[7] D. A. Patterson and J. L. Hennesey, “Estructura y diseño de computadores”, Ed. Reverté, cap.6, pp.426-513, 2000.

[8] Xilinx “Dual-Port Block Memory core v6.3”, 2005.

[9] Xilinx, “Fast Fourier Transform v3.2”, 2006.

[10] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series", Mathematics of Computation 19, pp. 297-301, April 1965.

[11] I. S. Uzun, A. Amira and A. Bouridane, “FPGA implementations of fast Fourier transforms for real-time signal and image processing”, IEE Proc.- Vis. Image Signal Processing, vol. 152, no. 3, pp. 283-296, 2005.

[12] Xilinx, “XtremeDSP for Virtex-4 FPGAs user guide”, 2006.

[13] Xilinx, “Virtex-5 Family Overview. LX, LXT and SXT Platforms”, pp. 2-6, December 2007.