Acceleration of Computation Speed for Wavefront Phase Recovery Using Programmable Logic

Eduardo Magdaleno and Manuel Rodríguez
University of La Laguna
Spain

1. Introduction

Atmospheric turbulence introduces optical aberration into wavefronts arriving at ground-based telescopes. Current adaptive optics (AO) systems use vector-matrix-multiply (VMM) reconstructors to convert gradient measurements to wavefront phase estimates. Until recently, the problem of an efficient phase recoverer design has been implemented over PC or GPU platforms. 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. Furthermore, an increase in telescope size requires significant computational power. 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. For this reason other hardware technologies must be taken into account during the development of a specific processor.

In recent years, programmable logic devices called FPGA (Field Programmable Gate Array) are seriously taken into account like a technological alternative in those fields where fast computations are required. The FPGA technology makes the sensor applications small-sized (portable), flexible, customizable, reconfigurable and reprogrammable with the advantages of good customization, cost-effectiveness, integration, accessibility and expandability. Moreover, an FPGA can accelerate the sensor calculations due to the architecture of this device. In this way, FPGA technology offers extremely high-performance signal processing and conditioning capabilities through parallelism based on slices and arithmetic circuits and highly flexible interconnection possibilities (Meyer-Baese, 2001 and Craven & Athanas 2007). Furthermore, 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 (Deschamps 2006).

Thanks to computer-aided design tools, FPGA circuits can be implemented in a relatively short space of time. For these reasons, FPGA technology features are an important consideration in sensor applications nowadays.

In particular, the algorithm of the phase recoverer is based on the estimation of Fast Fourier Transforms (FFT) (Roddier & Roddier, 1991). For this reason, we have to do a careful choice of the architecture of the FFT. An efficient design of this block can result in substantial benefits in speed in comparison with the GPU, DSP or CPU solution.

This chapter presents the design of a fast enough wavefront phase reconstruction algorithm that is based on FPGA technology, paving the way to accomplish the extremely large telescope's (ELT) number of actuators computational requirements within a 6 ms limit, which is the atmospheric response time. The design was programmed using the VHDL hardware description language and XST was used to synthesize into Virtex-6 and Virtex-7 devices. The FPGA implementation results almost 30 times faster than using GPU technology for a 64x64 phase recoverer. The chapter presents a comparative analysis of used resources for several FPGAs and time analysis.

2. A descriptive approach to the wavefront phase recovery

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 (figure 1):

$$I^{ij}(x,y) = \left|FFT[\psi^{ij}_{telescope}(u,v)]\right|^2 \tag{1}$$

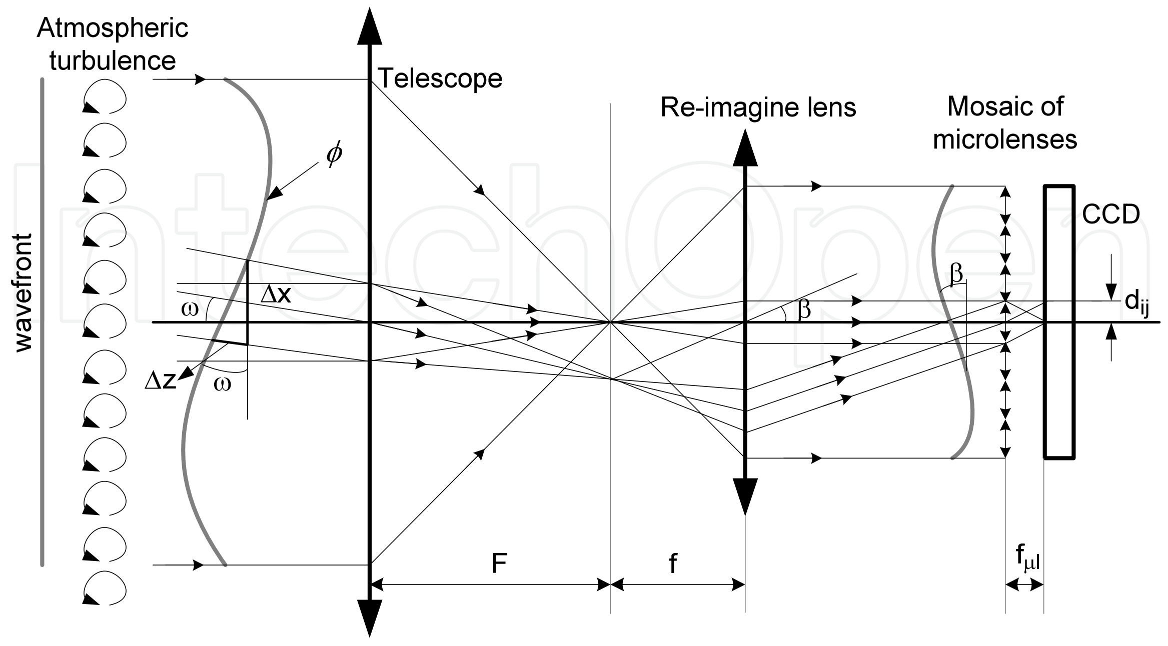


Fig. 1. Diagram of the light travel across a Shack-Hartmann sensor to measure the phase of the wavefront in the telescope pupil

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:

$$d_{ij} = K \cdot \frac{\partial \langle \varphi \rangle_{subp_{ij}}}{\partial \vec{r}} \tag{2}$$

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 Φ(u,v) can be recovered using an expansion over complex exponential polynomials (Poyneer et al., 2002):

$$\varphi(u,v) = \sum_{p,q=0}^{N-1} a_{pq} Z_{pq}(u,v) = \sum_{p,q=0}^{N-1} a_{pq} \frac{1}{N} e^{\frac{2\pi i}{N}(pu+qv)} = IFFT(a_{pq}) \tag{3}$$

The gradient is then written:

$$\vec{S}(u,v) = \vec{\nabla} \varphi(u,v) = \frac{\partial \varphi}{\partial u}\vec{i} + \frac{\partial \varphi}{\partial v}\vec{j} = \sum_{p,q} a_{pq} \vec{\nabla} Z_{pq} \tag{4}$$

Making a least squares fit over the F function:

$$F = \sum_{u,v=1}^{N} [\vec{S}(u,v) - \sum_{p,q} a_{pq} (\tfrac{\partial Z_{pq}}{\partial u}\vec{i} + \tfrac{\partial Z_{pq}}{\partial v}\vec{j})]^2 \tag{5}$$

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:

$$a_{pq} = \frac{ipFFT\left\{S^x(u,v)\right\} + iqFFT\left\{S^y(u,v)\right\}}{p^2 + q^2} \tag{6}$$

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

$$\varphi(u,v) = FFT^{-1}[a_{pq}] \tag{7}$$

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 FFT to be specifically adapted to the modal wavefront recovery pipeline and the FPGA architecture.

3. A descriptive approach to the FPGA architecture

A FPGA device is essentially a matrix of logic cells (called slices). These slices are connected among themselves and with input/output blocks (IOB) through routing channels. These channels are distributed in the FPGA in horizontal and vertical form and its connexions are fixed using a programmable switch matrix (figure 2).

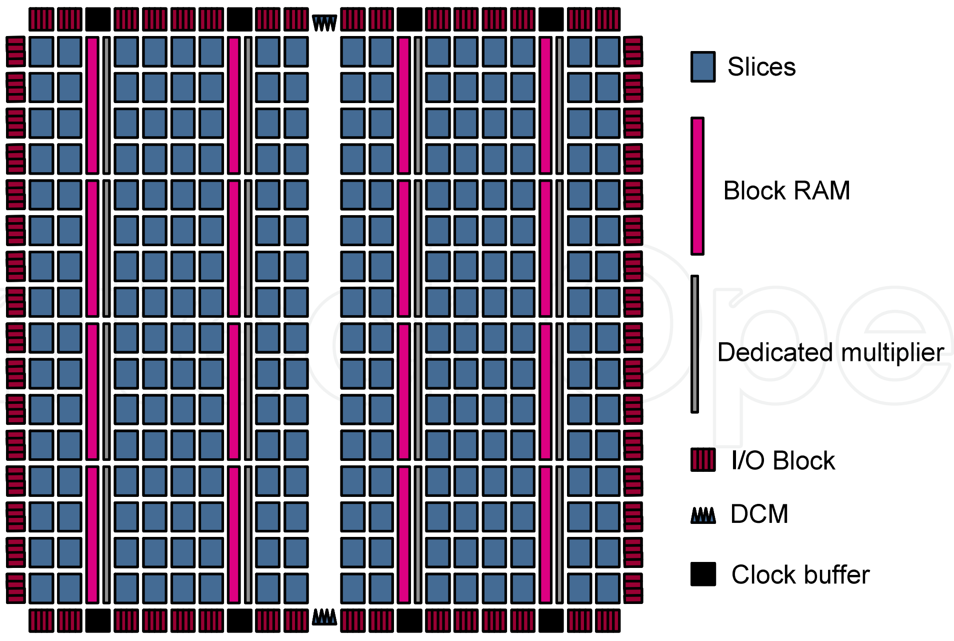


Fig. 2. FPGA generic architecture

Each slice is formed by look-up tables (LUT) and several flip-flops and programmable multiplexers (figure 3). The number and features of these elements depend on the FPGA family. Slices can implement memory (called distributed memory) or simple logic functions. Complex functions and parallel circuits are implemented when they are associated among themselves. Furthermore, FPGAs contain specific RAM blocks (BRAM) and arithmetic circuits.

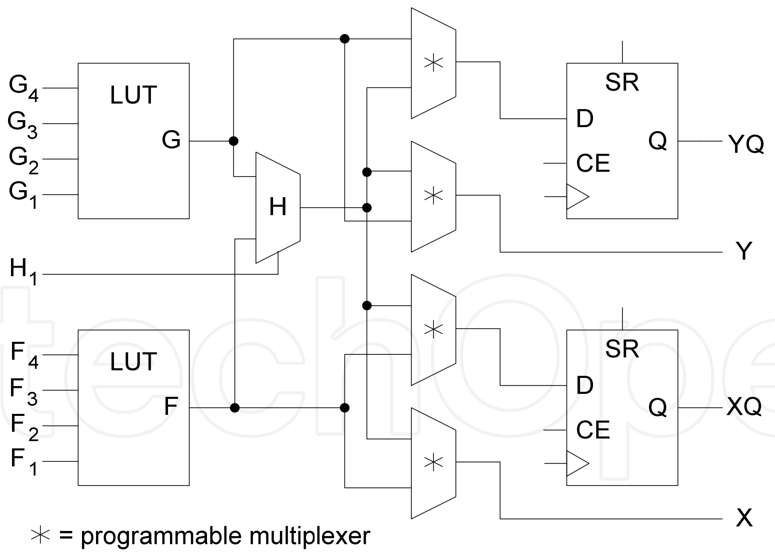


Fig. 3. Block diagram of a slice

In particular, FPGAs allow a wide variety of computer arithmetic implementations for the desired digital signal processing algorithms like FFT, because of the physical bit-level programming architecture. FFT algorithm can be parallelized and we can implement a full pipeline architecture using FPGA. This feature contrasts with DSPs and GPUs, with the fixed multiply accumulator core.

In addition, Xilinx Virtex FPGA devices incorporate DSP48 arithmetic modules. Each DSP48 slice has a two-input multiplier followed by multiplexers and a three-input adder/subtracter. The multiplier accepts two 18-bit, two's complement operands producing a 36-bit, two's complement result. The result is sign extended to 48 bits and can optionally be fed to the adder/subtracter. The adder/subtracter accepts three 48-bit, two's complement operands, and produces a 48-bit two's complement result. Two DSP48 slices, a shared 48-bit C bus, and dedicated interconnect form a DSP48 tile (figure 4).

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 complex arithmetic without the use of general FPGA fabric (Hawkes, 2005).

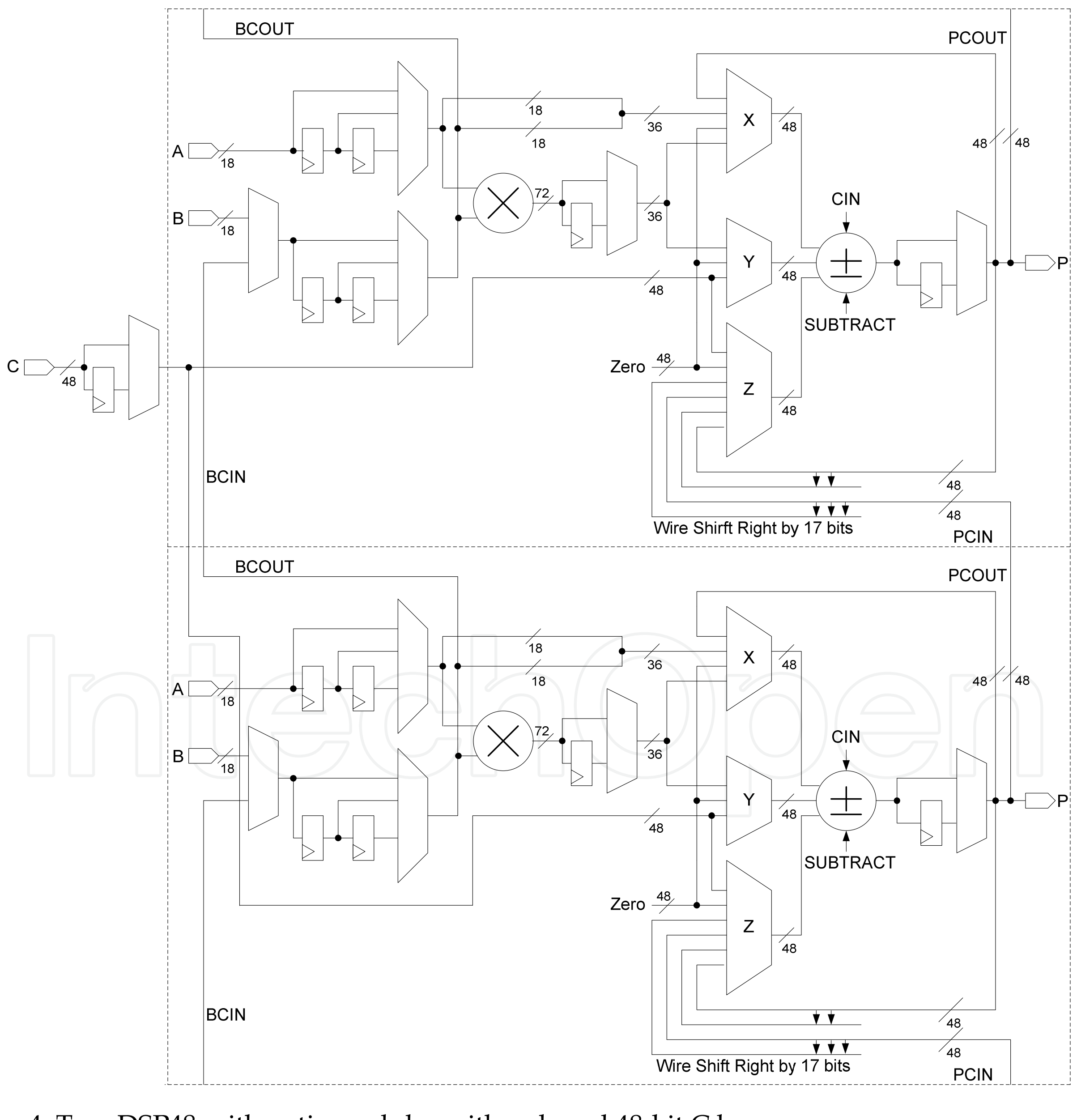


Fig. 4. Two DSP48 arithmetic modules with a shared 48-bit C bus

4. Design of an efficient one-dimensional FFT

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

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{kn},\ k = 0,1,\ldots N-1 \tag{8}$$

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 (Cooley & Tukey, 1965). 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,

$$\begin{aligned} X(k) &= \sum_{n\ even} x(n) \cdot W_N^{kn} + \sum_{n\ odd} x(n) \cdot W_N^{kn} = \\ &= \sum_{m=0}^{(N/2)-1} x(2m) \cdot W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1) \cdot W_N^{k(2m+1)} \end{aligned} \tag{9}$$

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 figure 5.

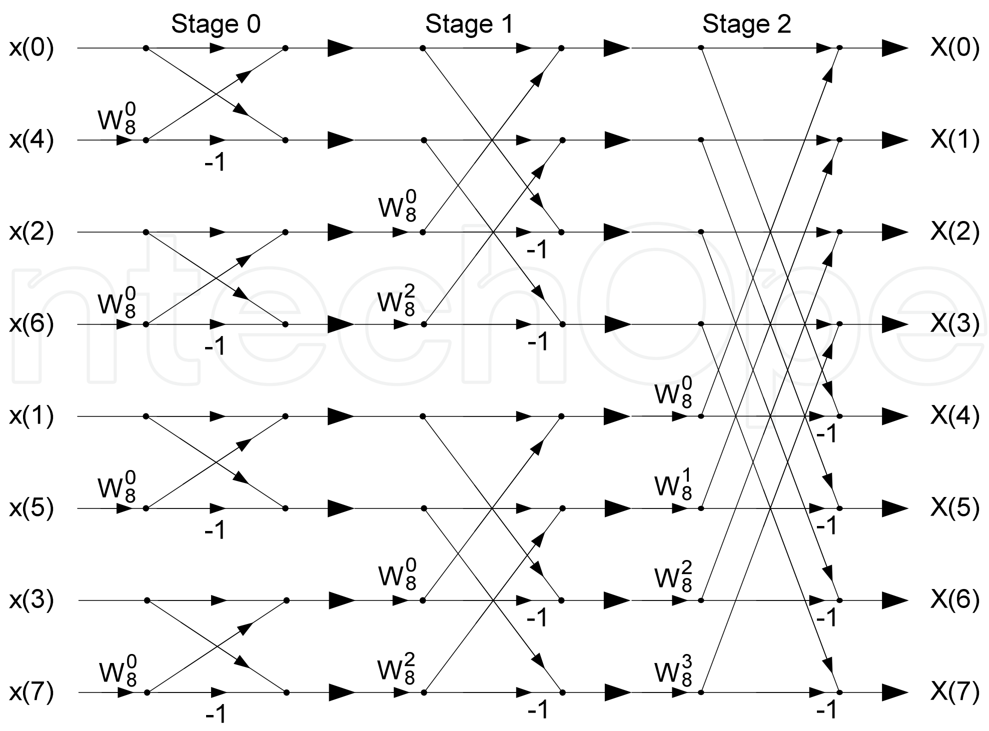


Fig. 5. Computation of FFT by decimation in frequency for N=8

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 (Zhou et al. 2007, Chang & Jen, 1998, Guo, 2000 and Chien et al., 2005). Instead, 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 FFT calculation in comparison with a sequential algorithm or adder trees solutions. A complex pipeline multiplier is implemented using only four DSP48 (figure 6) to calculate:

$$\begin{aligned} P_{real} &= A_{real} \cdot B_{real} - A_{imag} \cdot B_{imag} \\ P_{imag} &= A_{imag} \cdot B_{real} + A_{real} \cdot B_{imag} \end{aligned} \tag{10}$$

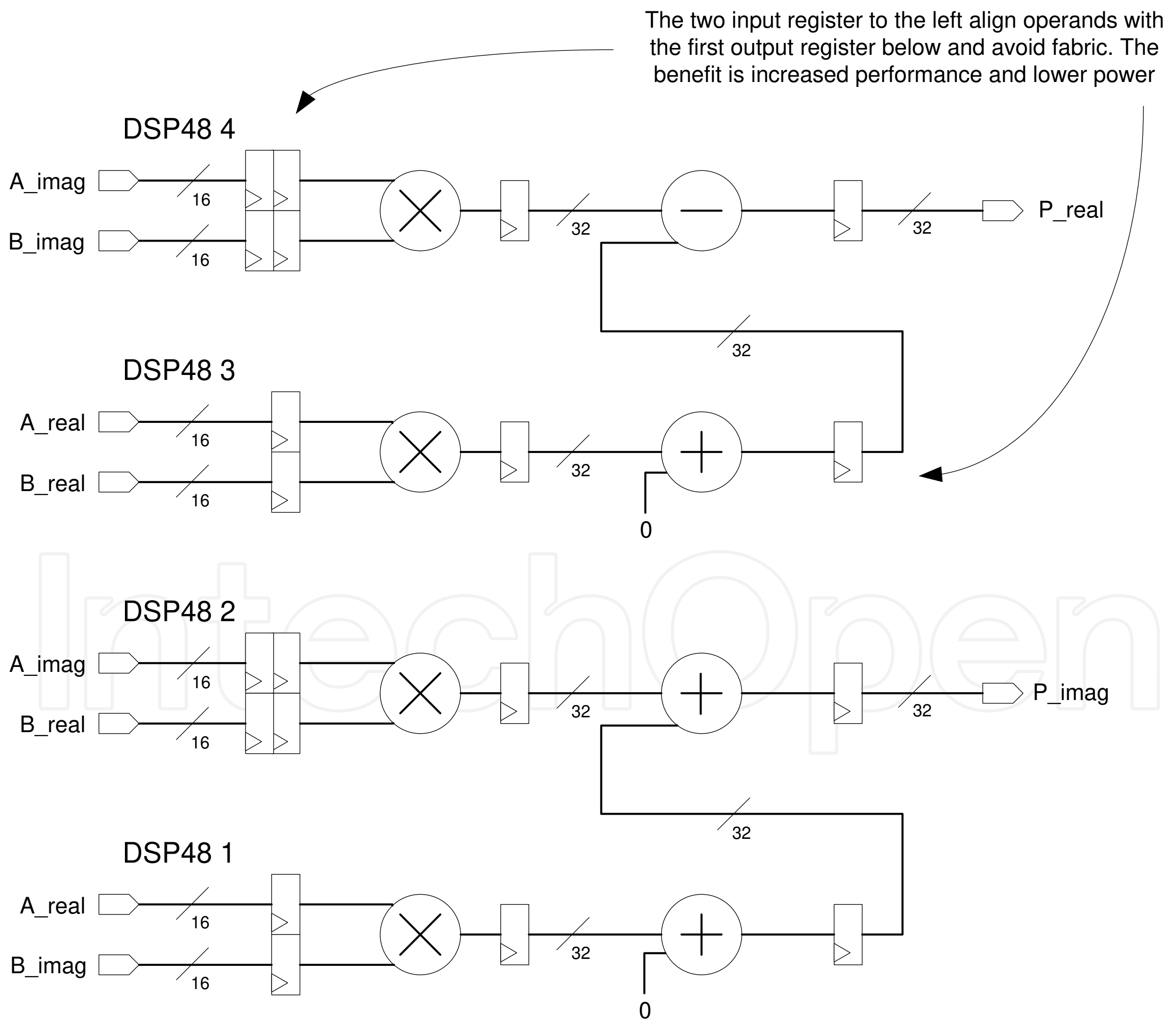

The two input register to the left align operands with the first output register below and avoid fabric. The benefit is increased performance and lower power
DSP48 4
A_imag
B_imag
16
16
32
32
P_real
32
DSP48 3
A_real
B_real
16
16
32
0
DSP48 2
A_imag
B_real
16
16
32
32
P_imag
32
DSP48 1
A_real
B_imag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

Fig. 6. Pipeline, complex multiplier with four parallel real multipliers

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 (Hawkes, 2005).

The complete pipeline radix-2 butterfly can be easily implemented with this specialized multiplier. It is necessary to use a FPGA Look-Up Table (LUT) (configured as SRL16 3-bits shift register) to preserve the synchronism. The butterfly implemented is depicted in Figure 7 and it needs only seven clock cycles to carry out the computation.

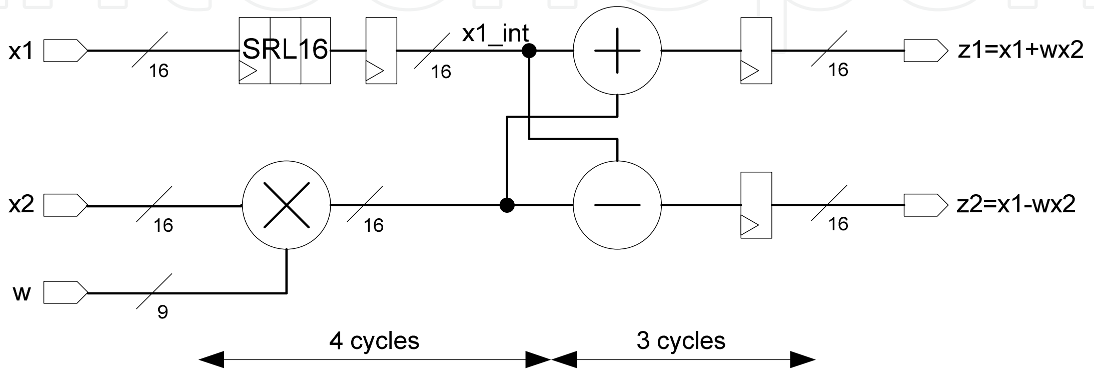


Fig. 7. Pipeline radix-2 butterfly in FPGA

A pipeline radix-2 FFT can be implemented using one butterfly in each stage. The twiddle coefficients used in each stage are stored in twiddle LUT ROMs in the FPGA. The logic resources and the clock cycles of the FFT module is reduced in our implementation using specific butterflies modules at the first and second stages. The first stage utilizes the feature of the twiddle factors related to the first stages of the pipeline.

$$W_N^{N/2} = 1 \tag{11}$$

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

$$W_N^{N/4} = j \tag{12}$$

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 equation 13.

$$(a+bj)j = aj - b = -b + aj \tag{13}$$

Taking into account these features, the 1D-FFT architecture implementation is depicted in figure 8. The swap-blocks arrange the data flow (according figure 5) 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 (figure 9).

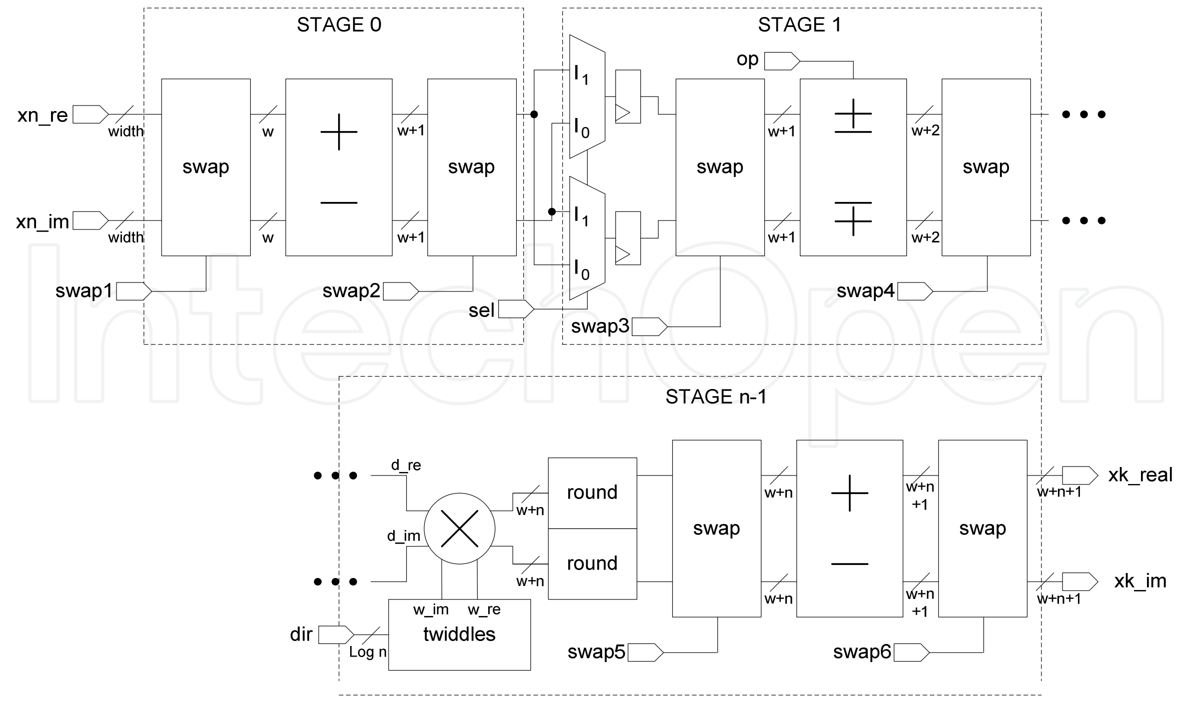


Fig. 8. Architectural block diagram of a radix-2 pipeline FFT

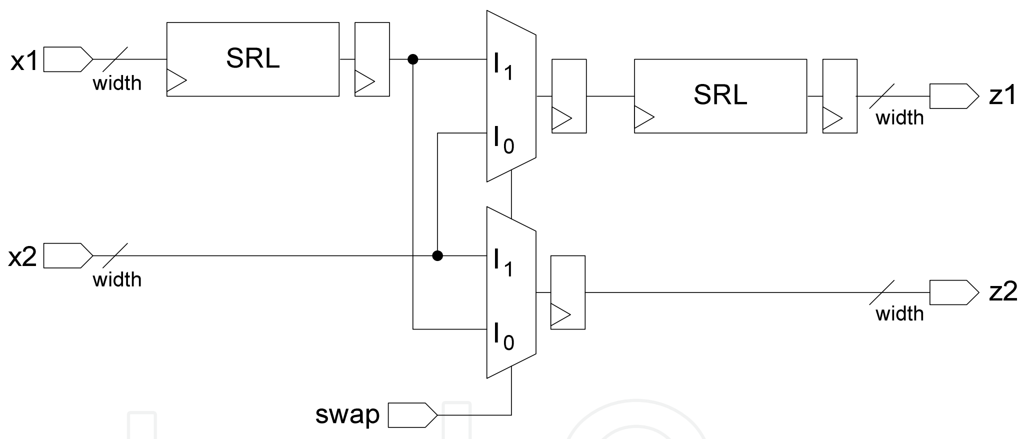


Fig. 9. Swap unit

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:

$$\textit{output width} = \textit{input width} + \log_2 \textit{points} + 1 \tag{14}$$

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 (figure 8). 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.

4.1 Temporal analysis for the radix-2 pipeline FFT module and superior radix

In Table 1 is depicted the latency for each stage of an 8-points FFT (figure 8).

Stage	Module	Cycles
0	Swap 1	5
	Adder 1	2
	Swap 2	5
1	Multiplexers	1
	Swap 3	3
	Add/Sub 2	2
	Swap 4	3
2	Multiplier	4
	Round	1
	Swap 5	2
	Adder 3	2
	Swap 6	2
	Total	**32**

Table 1. Latency of each module in a pipeline 8-points FFT

Taking into account the clock cycles of each block in table 1, the latency of the N-points FFT module ($\log_2 N$ stages) can be written as

$$\begin{aligned} latency &= 2\left(\frac{N}{2}+1\right)+2+1+2\left(\frac{N}{4}+1\right)+2+ \\ &+\sum_{n=2}^{(\log_2 N)-1}\left[7+2\left(\frac{N}{2^{n+1}}+1\right)\right], \quad n=2,3,\ldots(\log_2 N)-1 \end{aligned} \tag{15}$$

where the two first stages are considered separately, and N and n are the number of points of the transform and the number of stages of the module respectively. Operating,

$$\begin{aligned} latency &= 9 + \frac{3N}{2} + \sum_{n=2}^{(\log_2 N)-1} \left[9 + \frac{N}{2^n} \right] = \\ &= 9 + \frac{3N}{2} + 9(\log_2 N - 2) + \sum_{n=2}^{(\log_2 N)-1} \frac{N}{2^n} = \\ &= \frac{3N}{2} + 9\log_2 N - 9 + N \sum_{n=2}^{(\log_2 N)-1} \frac{1}{2^n} = \\ &= \frac{3N}{2} - \frac{N}{2} + 9\log_2 N - 9 + N \sum_{n=1}^{(\log_2 N)-1} \frac{1}{2^n} = \\ &= N + 9\log_2 N - 9 + N \sum_{n=1}^{(\log_2 N)-1} \frac{1}{2^n} \end{aligned} \tag{16}$$

The last summand of this equation is a geometric series with common ratio equal to ½. This is a convergent series and the partial sum to *r* of the series is

$$S_r = \sum_{n=1}^{r} \frac{1}{2^n} = \frac{2^r - 1}{2^r} \tag{17}$$

Where in this case *r=log₂N-1*, so

$$S_{(\log_2 N)-1} = \frac{2^{(\log_2 N)-1} - 1}{2^{(\log_2 N)-1}} = \frac{2^{\log_2 N} 2^{-1} - 1}{2^{\log_2 N} 2^{-1}} = \frac{\frac{N}{2} - 1}{\frac{N}{2}} = \frac{N-2}{N} \tag{18}$$

Adding the geometrical series in equation 17 and grouping, finally

$$latency = 2N + 9\log_2 N - 11\,, \quad N = 8, 16, 32, \ldots \tag{19}$$

When the number of points of the FFT is a power of 4 is computationally more efficient to use a radix 4 algorithm instead of used radix 2. The reasoning is the same than radix 2 but subdividing iteratively a sequence of N data into four subsequences, and so on. The radix-4 FFT algorithm consists of $\log_4 N$ stages, each one containing N/4 butterflies. As the first weight is $W_N^0 = 1$, each butterfly involves three complex multiplications and 12 complex sums. Performing the sum in two steps, according to Proakis & Manolakis (1996), it is possible to reduce the number of sums (12 to 8). Therefore, the number of complex sums to perform is the same (*$N\log_2 N$*) than the algorithm in base 2, but the multipliers are reduced by 25% (of *$(N/2)\log_2 N$* to *$(3N/8)\log_2 N$*). Consequently, the number of circuits for use DSP48 is reduced proportionately.

For number of point power of 4, the pipeline radix-4 FFT module has half of arithmetic stages, but the swap modules need twice clock cycles to arrange the data. Then, the latency is expressed as

$$\begin{aligned} latency(radix4) &= 2\left(\frac{3}{4}N+1\right)+2+ \\ &+\sum_{n=1}^{\log_4 N-1}\left[7+2\left(\frac{3}{4^{n+1}}N+1\right)\right]= \\ &=9\log_4 N-5+6N\sum_{n=1}^{\log_4 N-1}\frac{1}{4^n} \end{aligned} \tag{20}$$

Again, the series can be estimated for *log₄N-1* terms, and finally

$$\begin{aligned} latency(radix4) &= 2N+9\log_4 N-13, \\ N &= 16, 64, 256, 1024,\ldots \end{aligned} \tag{21}$$

This time estimation has been realized for other radix, as shown in the following equations:

$$\begin{aligned} latency(radix\ 8) &= 2N+9\log_8 N-21, \\ N &= 64, 512,\ldots \\ latency(radix\ 16) &= 2N+9\log_{16} N-37, \\ N &= 2^8, 2^{12},\ldots \end{aligned} \tag{22}$$

Generalizing

$$\begin{aligned} latency(radix\ i) &= 2N+9\log_i N-5-2i, \\ N &= i^{n+1}, n=1,2,3,\ldots \\ i &= 4,8,16,\ldots \end{aligned} \tag{23}$$

In Figure 10 is depicted the clock cycles of each algorithm and a proposed resolution with an orange line. All implementations are close to 2 when the number of points grows (figure 10a). The improvement in terms of computing speed of the algorithm using other radix is relevant when the number of samples is small. For example, the improvement factor for a 1024-point FFT is less than 7% using a radix-32 algorithm and less than 3% using a radix-4 (Figure 10b). However, in our astronomical case the proposed size is relativity small and the improvement using superior radix is relevant. Examining figure 10b, we can observe that the improvement factor is about 20% using radix-8 and 30% using radix-16. Thus, we are considering implementing these algorithms in the future.

4.2 Resources analysis

Several FFTs were implemented over a XC6VLX240T Virtex-6 device and numerical results were satisfactorily compared with Matlab simulations. Resources in this FPGA include 301440 flip-flops, 150720 6-LUTs, 416 BRAM and 768 DSP48s. The syntheses were achieved by changing the size of the FFT and the data precision.

Figure 11 shows the resource utilization to implement a pipeline 1024-FFT when the input data precision is between 8 and 24 bits. The resource utilization is greater when the FFT has more precision due to the increase of the intrinsic complexity found when the precision is increased. From 17 bits of precision, DSP48 circuits are increased and the maximum operating frequency is drastically smaller. It is due to high-precision complex multipliers need eight DSP48 circuits instead four in Figure 6. These high-precision multipliers are increased in each stage of the FFT from 17 bits data precision.

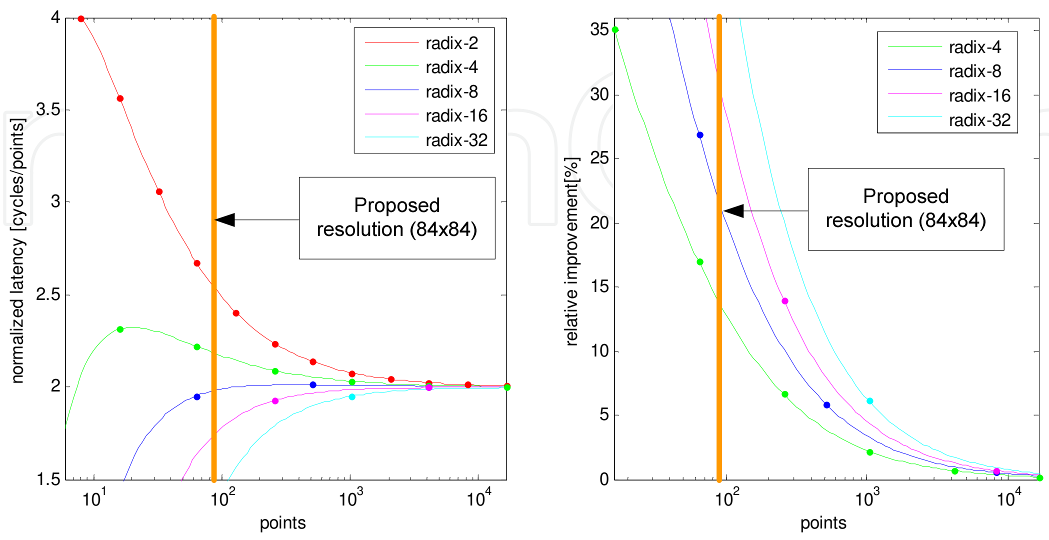


Fig. 10. (a) Normalized latency. (b) Relative improvement respect of radix-2 algorithm

In figure 12 is depicted the resource utilization to implement one-dimension FFT when de number of points of the transform is increased in a power of 2.

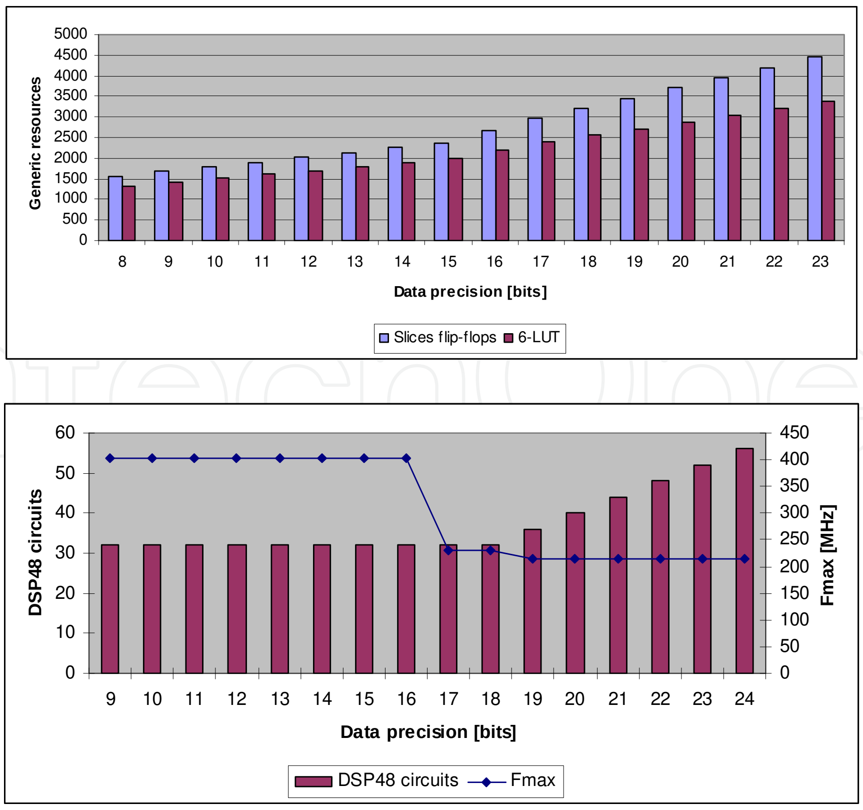
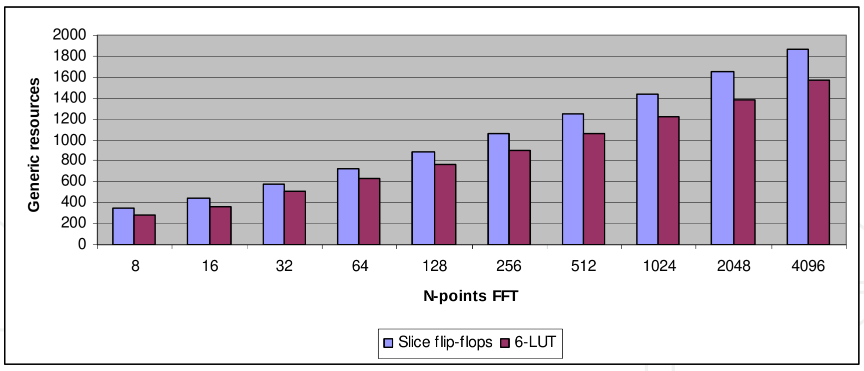


Fig. 11. FPGA Resources and maximum operating frequency for 1024-FFT



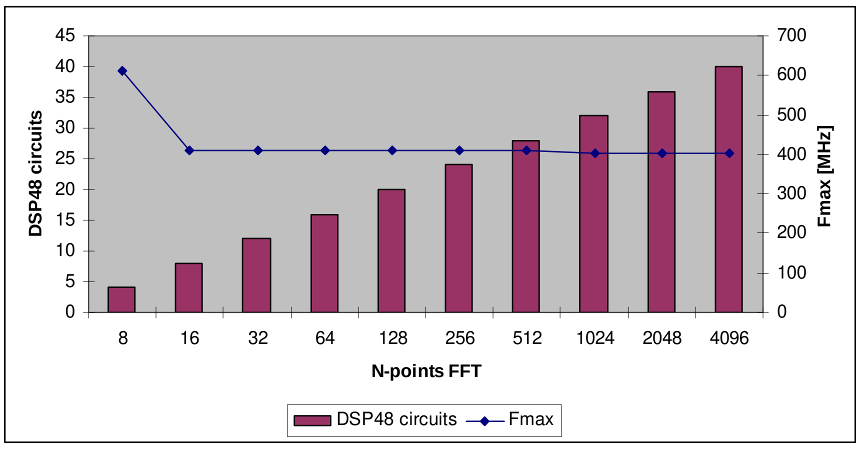


Fig. 12. FPGA Resources and maximum operating frequency for N-FFT with N from 8 to 4096

4.3 Comparison with others implementations

A comparison has been carried out between our design and others implementations. The combined use of the FPGA technology and the developed architecture achieves an improved performance compared to other alternatives. This is showed in figure 13 where our implementation executes an 1024-point FFT operation in 10.64 µs at 200 MHz.

5. Wavefront phase recovery FPGA-implementation

We will focus on the FPGA implementation from equations (6) and (7) 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.

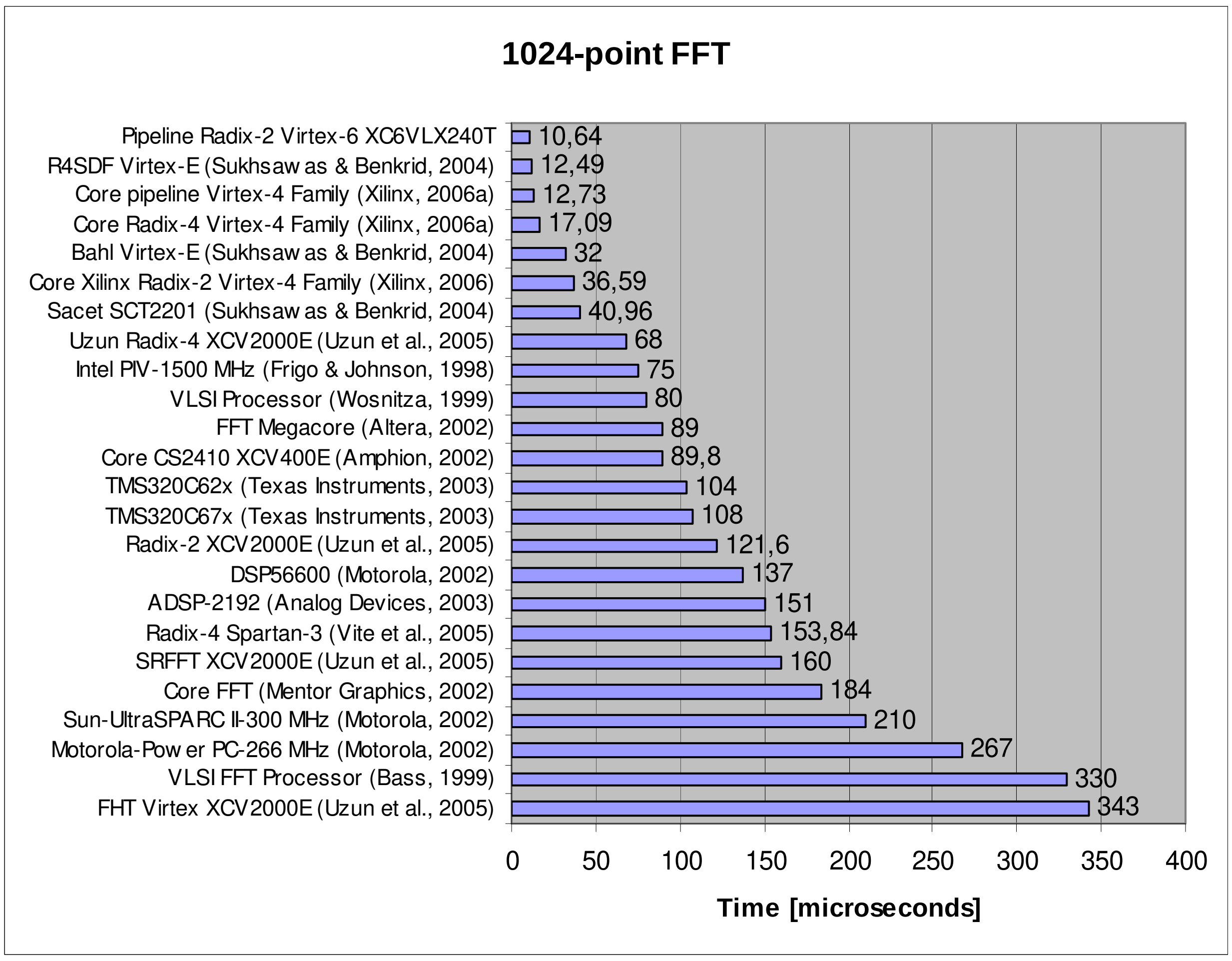


Fig. 13. FPGA Resources and maximum operating frequency for N-FFT with N from 8 to 4096

The block diagram of the designed recoverer is depicted in figure 14 where S_x and S_y represent the image displacement into each subpupil. The bidimensional transforms of S_x and S_y have to be multiplied by ip/p^2+q^2 and iq/p^2+q^2 respectively according to equation 6. These two matrices are identical if we change rows by columns. We can therefore store a unique ROM. The results of the adders (a_{pq} 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.

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(S_x) and FFT(S_y).
2. Compute the complex coefficients
3. Carry out a complex inverse 2D FFT on a_{pq}.
4. Flip data results.

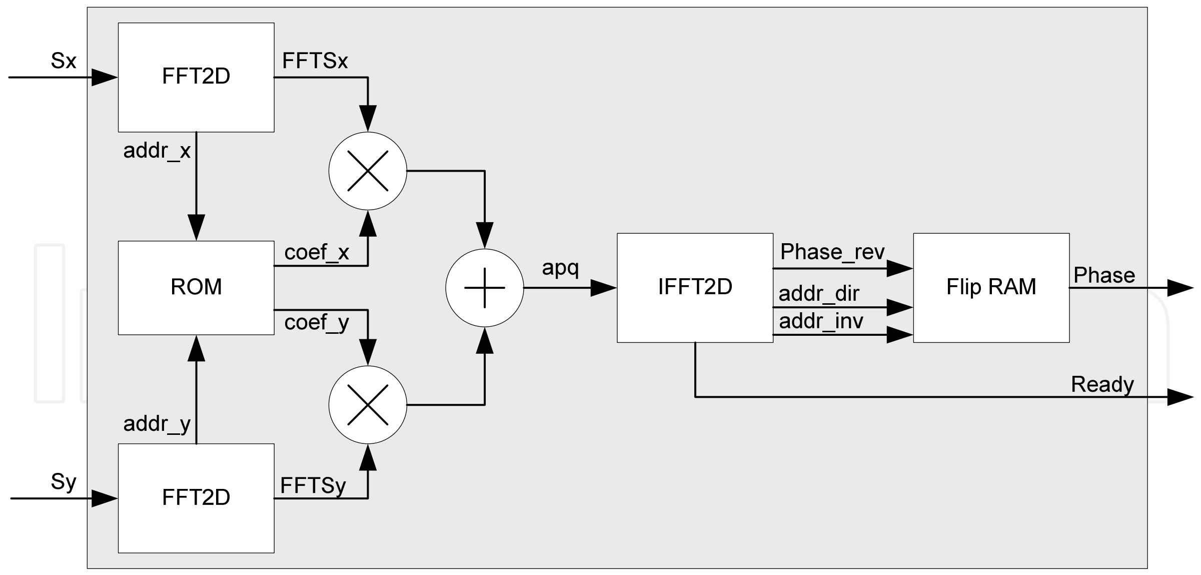


Fig. 14. Architecture of the synthesized phase recovery

5.1 2D-FFT on the Virtex-6 FPGA

The fundamental operation in order to calculate the 2DFFT 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

Figure 15 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.

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.

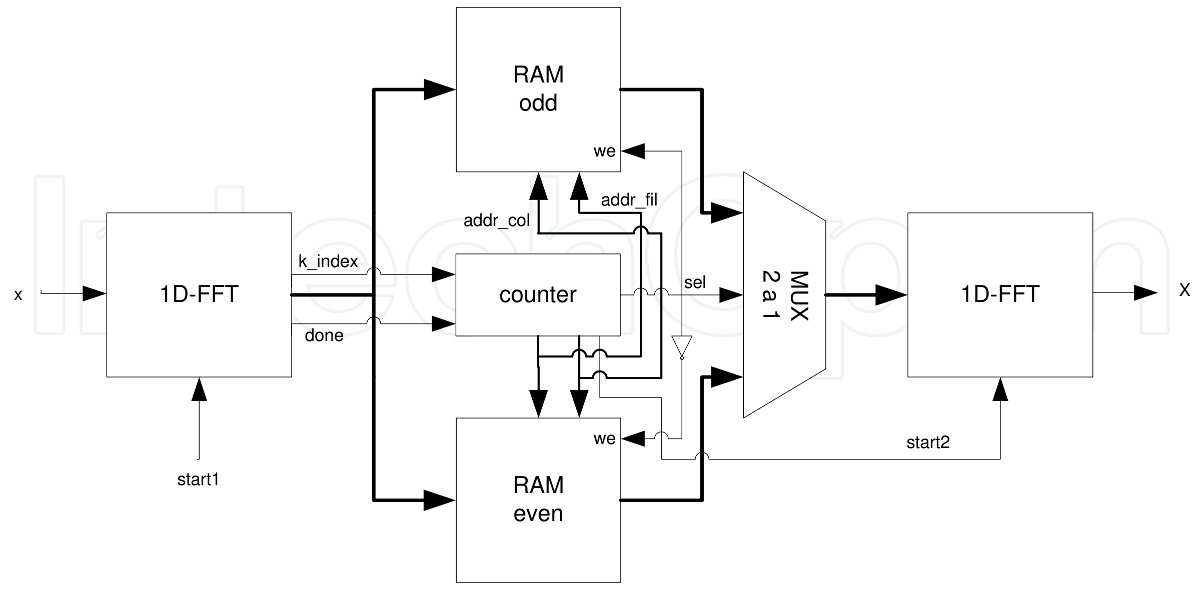


Fig. 15. Block diagram of the implemented 2D-FFT

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. (4)-(5) and showed in figure 13 (Flip-RAM module).

Row 1D-FFT block and column 1D-FFT block are not identical due to the unscaled data precision. For example, a 64x64 2D-FFT for the phase recoverer must meet certain requirements. If the precision of data input is 8 bits, the output data of 1D-FFT of the rows has to be 15 bits according equation 13. 1D-FFT of the columns accepts a 15 bits data format and 22 bits at the output.

Taking into account the latency of the FFT (equation 18) and the pipeline operation of the memory modules, the latency of the 2D-FFT module can be written as

$$latency = N^2 + 4N + 18\log_2 N - 22\,, \quad N = 8, 16, 32, \ldots \qquad (24)$$

In Figure 16 is depicted the latency for different sizes of 2D-FFT according to equation 18.

Table 2 shows a performance comparison of existing 2D-FFTs implementations using FPGA and other technologies for matrix sizes 64x64 and 128x128. Rodríguez-Ramos et al. 2007 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. 2007 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. Magdaleno et al. 2010 implemented 2D-FFT in a XC4VSX35 Virtex-4 operating at 100 MHz. Virtex-6 family can operate twice faster and the time operation is decreased in this way.

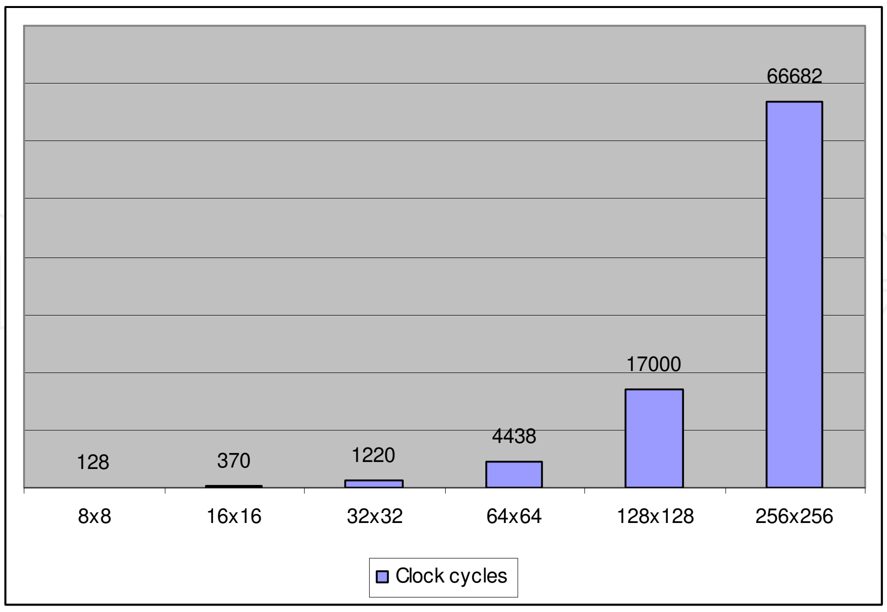


Fig. 16. Estimation of latency for several 2D-FFT

2D-FFT	CPU Rodríguez-Ramos et al. [2007]	GPU Rodríguez-Ramos et al. [2007]	FPGA Uzun et al. [2007]	FPGA Magdaleno et al. [2010]	FPGA Proposed
64x64	114.5 µs	1.58 ms	-	44.4 µs	22.2 µs
128x128	811.0 µs	1.68 ms	2.38 ms	170.8 µs	85.0 µs
256x256	-	-	-	-	333.4 µs

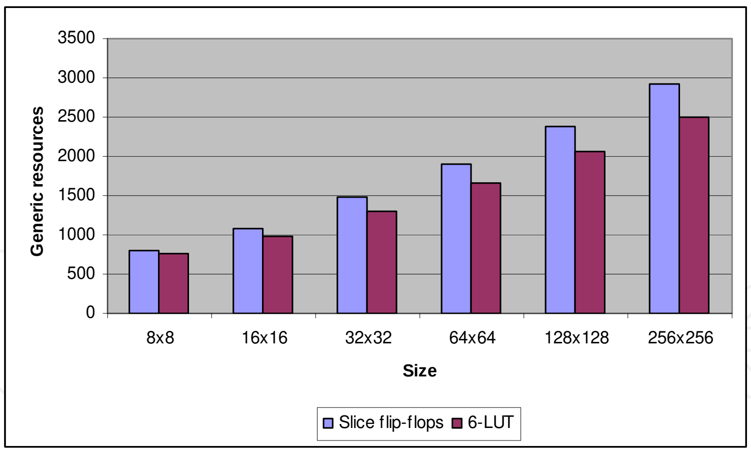
Table 2. 2D-FFT performance comparison with other designs

In figure 17 is depicted the resource utilization to implement 2D-FFT when de number of points of the transform is increased in a power of 2. The relevant resource is the Block-RAM. The 256x256 FFT occupies the 34% of BRAM for a XC6VLX240T Virtex-6 and the maximum operating frequency is only 297 MHz.

5.2 An 64x64 phase recoverer implementation

For a first prototype of the phase recoverer, we have selected a plenoptic sensor with 64x64 pixels sampling each microlens.

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 S_x and S_y are multiplied by some constant factors according to the formula of the phase recoverer (Figure 14). 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.



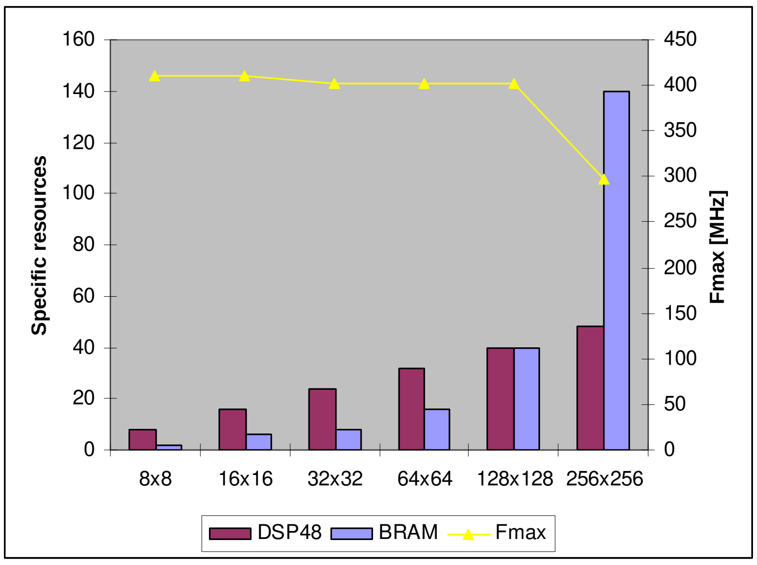


Fig. 17. Estimation of latency for several 2D-FFT

The importance of the parallel execution of the S_x and S_y 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 S_x and S_y have to be multiplied by ip/p^2+q^2 and iq/p^2+q^2 respectively according to Equation 6. These two 64x64 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 8 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 figure 18.

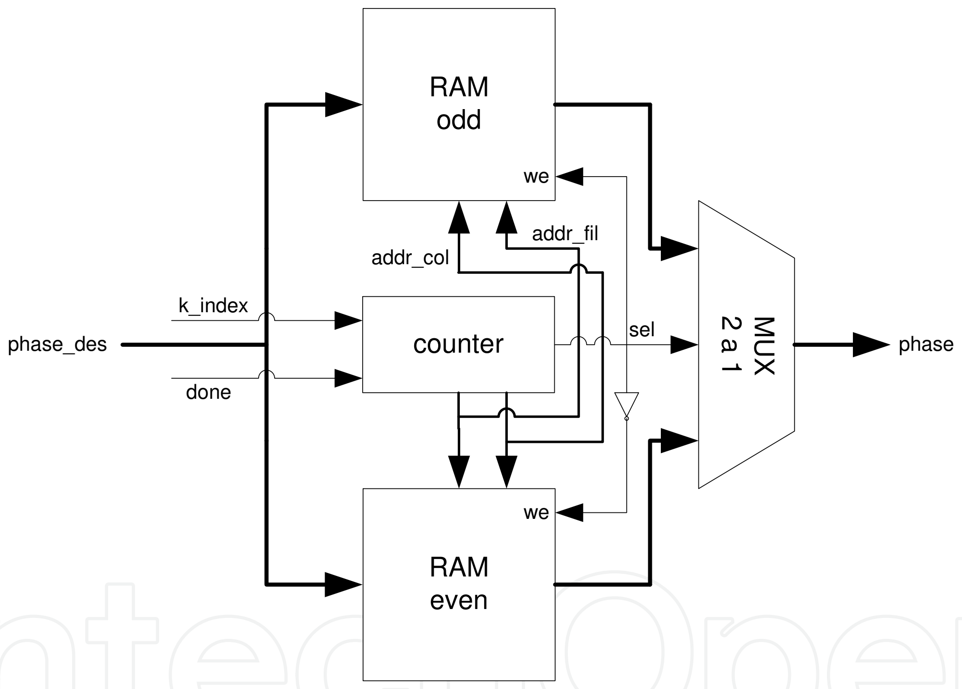


Fig. 18. Block diagram of the output stage of the wavefront recoverer (Flip-RAM)

The design of a 64x64 phase recoverer was programmed using the VHDL hardware description language (IEEE, 2000) and XST (Xilinx, 2006b) was used to synthesize these modules into a XC6VLX240T Virtex-6 FPGA.

The complete system was successfully tested in circuit using ChipScope Pro software (using phase gradients obtained in simulations) 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. Figure 19 shows an example of a wavefront reconstruction using a 64 × 64 subpupil recoverer. The two first images show the phase gradients (S_x, S_y) given to the module. The last picture is the recovered phase using the implemented module.

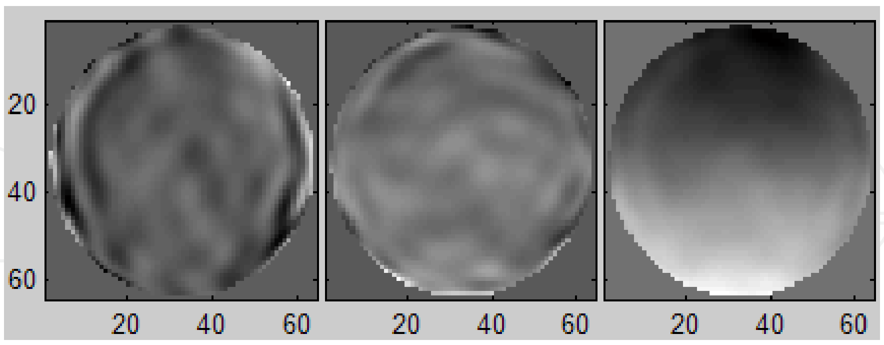


Fig. 19. Phase gradients (S_x and S_y) and the recovered phase for a Shack-Hartmann sensor with 64x64 subpupils

Table 3 shows the total time broken down into the stages of the total system (depicted in figure 13). 12980 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 200 MHz frequency clock, the system needs less than 65 µs to recover the phase.

Module	**Cycles**	**Duration (@ 200 MHz)**
2D-FFT (S_x and S_y)	4438	22.19 µs
Multipliers	4	0.02 µs
Adder	1	0.005 µs
IFFT2D	4438	22.19 µs
Flip-RAM	4096	20.48 µs
Rounding (3)	3	0.015 µs
Total	**12980**	**64.90 µs**

Table 3. Execution time (latency) for the different stages of the 64x64 phase recoverer

Generalizing, the latency for a generic NxN phase recoverer is determined as follows:

$$\begin{aligned} latency &= 2\left(N^2 + 4N + 18\log_2 N - 22\right) + 4 + 1 + 3 + N^2 = \\ &= 3N^2 + 8N + 36\log_2 N - 36\,, \quad N = 8, 16, 32, \ldots \end{aligned} \tag{25}$$

Table 4 shows XC6VLX240T Virtex-6 resource utilization and the maximum operating frequency (pre-synthesis).

Slices FFs	**6-LUT**	**DSP48**	**BRAM**	**Fmax [MHz]**
5819 (1%)	5000 (3%)	104 (13%)	58 (13%)	402.188

Table 4. XC6VLX240T Virtex-6 resource utilization to implement the phase recoverer prototype.

The implemented architecture is pipeline. This architecture allows phase data to be obtained for each 4,096 clock cycles (this number coincides with the number of points of the transforms, that is, the number of subpupils, 64 × 64, of the Shack-Hartmann sensor). Using the 200 MHz clock, the prototype provides new phase data each 20.48 µs.

These results can be compared with other works. Rodriguez-Ramos *et al.* (2007) implemented a 64 × 64 phase recoverer using GPU. In this technology, the wavefront reconstruction needs 3.531 ms. The FPGA implementation results almost 30 times faster. Baik *et al.* 2007 implemented a wave reconstruction with a 14 × 14 Shack-Hartmann array, an IBM PC and a Matrox Meteor-2 MC image processing board. The wavefront correction speed of the total system was 0.2 s. Although the system includes the gradient estimation, it can be seen that the execution times are slower than in the proposed implementation. Seifer *et al.* 2005 used a sensor with 16 × 12 subpupils and a Pentium M, 1.5 GHz. The wavefront reconstruction in this case was 50 ms using Zernike polynomials to adjust to the coefficients of the aperture function. Again, our implementation using FPGA technology is comparatively faster.

6. Conclusion

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.

A 64 × 64 wavefront recoverer prototype was synthesized with a Xilinx XC6VLX240T Virtex-6 as sole computational resource. This FPGA is provided in a ML605 evaluation platform. Our prototype was designed using ISE Foundation 13.1. The system has been successfully validated in the FPGA chip using simulated data.

A two-dimensional FFT is implemented as nuclei algorithm of the recoverer: processing times are really short. The system can process data in much lower times than the atmospheric response. This feature allows more phases to be introduced in the adaptive optical process. Then, the viability of the FPGAs for AO in the ELTs is assured.

Future work is expected to be focused on the optimization of the 2D-FFT using others algorithms (radix-8, radix-16). Finally, next-generation Virtex-7 devices provide enough DSP48 resources in order to implement all the butterflies in the 64-FFT algorithm. Using these devices, phases in the adaptive optic process could be estimated in much lower times.

7. References

Analog Devices (2003). *Analog Devices DSP Selection Guide 2003 Edition*, Analog Devices, 2003, Available from www.analog.com/processors.

Amphion (2002). CS248 FFT/IFFT Core Datasheet, Amphion Ltd., 2002. Available from www. datasheetarchive.com/500—Amphion-datasheet.html

Altera (2002). *FFT Megacore Function User Guide*, 2002, Available from www.altera.com

Bass B. (1999). A low-power, high-performance 1024-point FFT processor, *IEEE J. Solid-State Circuits*, 34, 380-387, ISSN 0018-9200.

Baik, S. H.; Park, S. K.; Kim, C. J. & Cha, B. (2007). A center detection algorithm for Shack–Hartmann wavefront sensor, *Optics & Laser Technology*, 39, 262-267, 2007, ISSN 0030-3992.

Chang T.S. & Jen C.W. (1998). Hardware efficient transform designs with cyclic formulation and subexpression sharing, *Proceedings of th 1998 IEEE ISCAS 1998*, 2, 398-401, ISBN 0-7803-4455-3, Monterey, California, USA, May 31-Jun 3, 1998.

Chien C. D; Lin C. C.; Yang C. H. & Guo J. I. (2005). Design and realization of a new hardware efficient IP core for the 1-D discrete Fourier transform *IEE Proc. Circuits, Devices and Systems*, 152, 247-258, ISSN 1350-2409.

Cooley, J.W. & Tukey, J.W. (1965). An algorithm for the machine calculation of complex Fourier series, *Mathematics of Computation*, 1965, 19, 297-301, ISSN 0025-5718.

Craven, S. & Athanas (2007), P. Examinig the Viability of FPGA Supercomputing, *EURASIP Journal on Embedded Systems*, 2007, 8, ISSN 1687-3963.

Deschamps, J.; Bioul, G. & Sutter, G. (2006). *Synthesis of Arithmetic Circuits. FPGA, ASIC and Embedded Systems*, Wiley-Interscience, 2006, ISBN 0-471-68783-9.

Frigo M. & Johnson S. (1998). FFTW: An adaptive software of the FFT, *Proceedings of IEEE International Conference On Acoustics, Speech, and Signal Processing 1998*, 1381-1384, ISBN 0-7803-4428-6, Seattle, Washington, USA, May 12-15.

Guo, J. I. (2000). An efficient parallel adder based design for one dimensional discrete Fourier transform, *Proc. Natl. Sci. Counc. ROC*, 2000, 24, 195-204.

Hawkes G. C. (2005). *DSP: Designing for Optimal Results. High-Performance DSP Using Virtex-4 FPGAs*, Xilinx, Available from www.xilinx.com.

IEEE (2000). *Standard VHDL Language Reference Manual, IEEE-1076- 2000*, IEEE, 2000, ISBN 0-7381-3326-4.

Magdaleno, E., Rodríguez, M., Rodríguez-Ramos, J.M. (2010). An efficient pipeline wavefront phase recovery for the CAFADIS camera for extremely large telescopes, *Sensors*, ISSN 1424-8220.

Mentor Graphics (2002). *FFT/WinFFT/Convolver Transforms Core Datasheet*, 2002, Available from www.mentor.com.

Meyer-Baese, U. (2001). *Digital Signal Processing with Field Programmable Gate Arrays*, Springer-Berlag, 2001, ISBN 3-540-72612-8.

Motorola (2002). *Motorota DSP 56600 16-bit DSP Family Datasheet*, Motorola, 2002, Available from www.digchip.com.

Rodríguez-Ramos, J. M.; Marichal-Hernández, J. G. & Rosa F. (2007). Modal Fourier wavefront reconstruction using graphics processing units, *Journal of Electronic Imaging*, 16, 123-134, ISSN 1017-9909.

Poyneer, L. A.; Gave, D. T. & Brase, J. M. (2002). Fast wave-front reconstruction in large adaptive optics systems with use of the Fourier transforms, *Journal of the Optical Society of America A*, 2002, 19, 2100-2111, ISSN 1084-7529.

Roddier, F. & Roddier, C (1991). Wavefront reconstruction using iterative Fourier transforms, *Applied Optics*, 30, 1325-1327, ISSN 2155-3165.

Proakis, J.G. & Manolakis, D.K. (1996). *Digital Signal Proccesing. Principles, Algorithms and Applications*, Prentice Hall, 1996, ISBN 0-13-187374-1.

Seifert, L.; Tiziani, H. J. & Osten W. (2005). Wavefront reconstruction with the adaptive Shack–Hartmann sensor, *Optics Communications*, 245, 255-269, 2005, North Holland, ISSN 0030-4018.

Sukhsawas S. & Benkrid K. (2004). A high-level implementation of a high performance pipeline FFT on Virtex-E FPGAs, *Proceedings of the IEEE Computer Annual Symposium on VLSI Emerging Trends in VLSI Systems Design 2004*, ISBN 0-7695-2097-9, Lafayette, Louisiana, USA, February 19-20.

Texas Instruments (2003). Texas Instruments C62x and C67x DSP Benchmarks, Texas Instruments, 2003, Available from www.ti.com.

Uzun I. S.; Amira, A & Bouridane, A. (2005). FPGA implementations of fast Fourier transforms for real-time signal and image processing, *IEE Proceedings - Vision Image and Signal Processing*, 152, 283-296, ISSN 1350-245X.

Vite J. A.; Romero R. & Ordaz A. (2005). VHDL Core for 1024-Point Radix-4 FFT Computation, *Proceedings of the 2005 International Conference on Reconfigurable Computing and FPGAs*, Puebla, Mexico, September 28-30, ISBN 0-7695-2456-7.

Wosnitza, M. (1999). *High precision 1024-point FFT processor for 2-D object detection*, PhD thesis, Harung-Gorre Verlag, Germany, 1999.

Xilinx (2006a). Fast Fourier Transform v3.2, 2006, Available from www.xilinx.com.

Xilinx (2006b). XST User Guide, Xilinx, 2006, pp. 118-217, Available from www.xilinx.com.

Zhou Y.; Noras, J.M. & Shepherd S. J. (2007). Novel design of multiplier-less FFT processors, *Signal Processing*, 87, 1402-1407, ISSN 0165-0684.