Abstract—This paper shows the current stage of development of a FFT processor in VHDL. This processor will use fixed-point as numeric representation, taking advantage of the facilities provided by IEEE fixed point package. Its main advantages is that it is being developed as fully parameterizable processor, in a way that number of bits, fixed point position and number of points computed in the FFT can be easily changed and that it can be used in various applications such as classification algorithms and communications systems. A prototype core has been developed and it can perform a complete FFT transform using radix-2 with decimation in time. Results and details of this implementation will be shown in this work.

Index Terms—VHDL language, digital hardware design, FFT.

I. INTRODUCTION

The Fourier transform is a mathematical operation involving complex exponentials that is able to represent a by its frequency components. The resulting function is same signal represented in the frequency domain [1]. It is base for many science subareas, such as physics for studying wave motion and optics, in electrical engineering for telecommunications [2], and others.

A discrete version of the Fourier transform was created to work with discrete signals called Discrete Fourier Transform (DFT) [1], and from this, in turn, was created a “faster” version called Fast Fourier Transform (FFT) which made possible faster executions in computers and recently in dedicated hardware.

The algorithm that performs the basic operations of the FFT are called butterfly. The butterfly operation commonly uses 2 or 4 samples, that are named radix-2 [3] and radix-4 [4], respectively. In this work the radix-2 FFT algorithm was implemented due it is simpler and uses less resources.

Applications that use FFT and require real-time response are implemented using hardware description language aiming a dedicated hardware design, because of its parallelized architecture and throughput. The hardware description language is a type of language designed to bypass the difficulties found when using traditional programming languages for modeling the hardware present digital systems, due to the hardware high parallel nature. In this work has been chosen VHDL (VHSIC hardware description language) that is designed for treating natively the parallelism and delay inherent of logic circuits that composes the digital hardwares.

This work presents a VHDL description of a FFT algorithm to take advantage of the digital hardware high parallelization and performance to build a digital FFT processor. This FFT processor uses fixed point as numeric representation and is being developed in order to get a synthesizable and flexible processor in a way that number of bits, fixed point position and number of points computed in the FFT can be easily changed and that it can be used in various applications. This description uses the FFT radix-2 using decimation in time.

This paper is organized as follow: Section II gives details about how data is treated by the FFT processor, its internal blocks organization and input-output pins. Section III explains some design issues related to the main block composing the processor. Section IV shows the synthesis results of this processor. Section V explains some timing issues and Section VI show the conclusions and future works.

II. FFT PROCESSOR IMPLEMENTATION

The algorithm used in the VHDL implementation was FFT radix-2 with decimation in time [1]. The decimation in time means that before computing the transform the samples must be stored following the bit reversed order and read sequentially, otherwise the samples are read in the bit reversed order, this is defined in the DFT decomposition that will result in the FFT algorithm. Radix-2 means that the DFT will be decomposed in two parallel equations to make-up the FFT algorithm, using this decomposition the whole basic mathematical operations that will result in the transform are done using two samples at a time.

The FFT processor treats the data as fixed point [5]. The number of bits reserved to the integer and decimal parts are input parameters of the FFT processor. To use this data representation the FFT processor uses the IEEE fixed point VHDL package [6] which includes facilities for working with fixed point, such mathematical operations and fixed point conversion routines.

The number of points to the FFT can be either parametrized before compilation, but it must be a power of two less than 512. This limitation exists because it is necessary to store the complex exponentials values in the BUTTERFLY code. Currently are stored the complex exponentials values for 512 point and these numbers can be used to lower transforms. But this limitation is not a problem, a simple high level language (for instance, Java) application can be written to
write automatically a specific set of complex exponentials values for each transform length.

To describe the algorithm in VHDL has been adopted the internal organization of the FFT processor components shown in Fig. 1. The following subsections provides a brief description of the input-output signals and internal blocks of Fig 1.

![Fig. 1. The internal organization of the FFT processor.](image)

A. Input-output signals description

- **CLK:** it is a input pin to provide to the FFT processor the clock signal that serves as time base.
- **RST:** it is a input pin to restart the FFT processor. When this pin is set to high (logical level 1) the registers are loaded with zeros and the state machines that perform the control of operations go to its initial state.
- **START:** it is a input pin used to flag to the FFT processor begin its operation. When this pin is set to high the FFT processor starts its operations.
- **RE_DATA & IM_DATA:** these are a set of input pins that represents the real and imaginary parts of the input values. The data length (number of bits) are defined before compilation.

The FFT processor is not completely done yet, therefore after set to high the START signal the device that will provide the data must wait for five clock cycles before to start transferring data (this corresponds to the internal components initialization) and after these clock cycles a sample is transferred per clock cycle. Hereafter the data input will be smarter and signalization pins will be used (such as a protocol).
- **RE_OUT & IM_OUT:** these are a set of output pins where the resulting data can be transferred. Each data in the memory is transferred per clock cycle and a protocol will be used in the future, similarly to the data input pins.

B. Internal blocks description

The FFT processor is composed by five blocks that are called INPUT, OUTPUT, ARBITER, MEMORY and BUTTERFLY. These blocks were designed as state machines [7], because this technique provides a good predictability of the hardware design peculiarities. Each block has the following functionality:

- **INPUT:** this block is responsible to store the input samples in the memory. After the START pin is set to high the ARBITER flags to this block to start its operation. It receives the input samples through RE_DATA and IM_DATA and generates the correct address to store the sample. The addresses are generated following the bit reverse order, therefore the decimation in time is performed during the INPUT operation. After it has done the operation it flags to ARBITER indicating the end of operation.

In the future this block will have some handshake pins (will be compliant with a protocol), so the device that will provide the data will be able to work at different frequency.
- **BUTTERFLY:** when the samples have been stored in the memory the ARBITER flags to it starts its job. This block is the kernel of the FFT processor and it can perform transforms in any length, it is only limited by the values of the complex exponentials that must be previously computed and stored. More details concerning this block is given in Section III.
- **OUTPUT:** at the end of the BUTTERFLY operation this block puts the resulting data stored in memory in RE_OUT and IM_OUT (real and imaginary part of data, respectively), during it is working a memory positions is read and written to RE_OUT and IM_OUT at a clock cycle.

As the INPUT this block will have some handshake pins to be smarter when transfering the data.
- **MEMORY:** it is composed by two storage blocks, for the real and imaginary parts of data.

Its implementation was made in a way that the synthesis tool will map automatically the code to the internal FPGA memory. The tool used is the Altera Quartus II [8], it is able to map automatically the code to the FPGA memories, but this behavior have not been tested with other synthesis tool.
- **ARBITER:** only one block must be able access the memories at a time. Thus, this block is responsible for managing the other blocks, providing them with signal to start theirs operations in a order that the transform can be successful and selecting signals coming from them (like read or write flags and addresses) to the memories.

For example, when the input signal START is set to high the ARBITER flags to the INPUT block to start its operation and waits for its flag indicating the end of its operation. The same is done to the BUTTERFLY and OUTPUT block, in this order.

During the time the ARBITER is waiting for flags coming from a block, it selects only the signals coming from that block to the memory and the signals coming from memory goes to it. For instance, during the BUTTERFLY operation the ARBITER will wait for its flags and in this time signals like read/write commands and data are multiplexed from the BUTTERFLY to the MEMORY and data coming from MEMORY is multiplexed to BUTTERFLY.
III. MORE ABOUT THE INTERNAL OPERATIONS IN BUTTERFLY BLOCK

In order to implement the FFT using radix-2 and decimation in time the data is treated as shown in Fig. 2. Incoming arrows are sums and terms along the arrows are multiplications, where \( W^y_x \) are complex exponentials in the form \( e^{-j2\pi y/x} \), and it is called twiddle factor. The decimation in time is either shown, the butterflies are performed using the samples following the bit reversed order.

Fig. 2. An example of 8-point FFT using radix-2 decimation in time algorithm.

The algorithm in VHDL has been designed as a state machine [7], thereby it is possible to have the algorithm working with the particularities of digital hardware design. For example, when performing a operation and storing the result in a register, the new value will be available some time after the next rising (or falling) clock edge. This behavior can be modeled as states in a state machine, where in a state an operation is done and the result is prepared to be stored in a register, in the clock edge the value is effectively stored and the state is changed, so in the next state the value can be used.

The FFT is composed by various stages and in each one of these stages the butterflies are performed using two data. The Fig. 2 shows these stages and the butterflies operations to a FFT length of 8 points. The number of stages is the logarithm base 2 of the number of points in the FFT and in each stage the butterfly operations are done in all data present in memory. Thereby, it is possible to infer the amount of operations performed and its consumption.

The BUTTERFLY block is independent of the others and it can be attached in other system and work as a internal block. For instance, in a system that perform sampling, the sampled data can be stored in a shared memory and the address of the first samples is passed to the BUTTERFLY, so it performs the transform and flags when it has done, thus the resulting data can be used. This represents a great clock cycle saving, hence time saving and efficiency.

IV. SYNTHESIS RESULTS

The FFT processor was synthesized using the Altera Quartus II software for 512 and 256 point transform, using data length of 16 bits (8 bits reserved for both integer and decimal parts) and the Cyclone II EP2C70F896I8 device.

The resources consumption remains the same for both excepting the memory bits. This happens because to change FFT length the only block significantly changed is the MEMORY (sized to store 256 or 512 data).

In the BUTTERFLY is used the same set of twiddle factors for both 256 and 512 FFT length, but using an specific set of twiddle factors for smaller transforms is necessary to avoid resource wasting. But this behavior shows that for a FFT with larger number of points the consumption will remain the same except for the memories and the pre-computed set of twiddle factors. The Table I shows some numbers regarding the synthesis using the same set of twiddle factors for both.

<table>
<thead>
<tr>
<th>Resource</th>
<th>512 points</th>
<th>256 points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combinational Functions</td>
<td>2698 (4%)</td>
<td>2673 (4%)</td>
</tr>
<tr>
<td>Logic Registers</td>
<td>463 (&lt;1%)</td>
<td>531 (&lt;1%)</td>
</tr>
<tr>
<td>Pins</td>
<td>68</td>
<td>68 (4%)</td>
</tr>
<tr>
<td>Memory Bits</td>
<td>32768 (3%)</td>
<td>16384 (1%)</td>
</tr>
<tr>
<td>Multipliers (9 bits)</td>
<td>8</td>
<td>8 (3%)</td>
</tr>
</tbody>
</table>

V. TIMING RESULTS

The report given by Altera Quartus II after the synthesis provides some knowledge to infer the maximum clock frequency that this FFT processor can work for the Cyclone II EP2C70F896I8 FPGA. With data provided by the report, was inferred a maximum clock frequency of 43.072 MHz.

The simulation for this clock frequency gives that the time required to compute a FFT using the the complete FFT processor is 0.7 ms, whereas using the BUTTERFLY block alone the time is 0.6 ms. The Table II summarizes the data presented.

<table>
<thead>
<tr>
<th>Number of points</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transform time:</td>
<td>- Complete processor 0.7 ms</td>
</tr>
<tr>
<td></td>
<td>- Only BUTTERFLY block 0.6 ms</td>
</tr>
<tr>
<td>Data width:</td>
<td>- Imaginary 16 bits</td>
</tr>
<tr>
<td></td>
<td>- Twiddle Factor</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>fixed-point</td>
</tr>
<tr>
<td>Maximum clock frequency</td>
<td>43.072 MHz</td>
</tr>
</tbody>
</table>

For some application that need lower delay it could not work, but several optimizations have been identified in the code and must be implemented and tested to assess
improvements, for instance the parallelization level can be increased to decrease the amount of time needed and memory access can be different in a way that the memory access is faster.

VI. CONCLUSION

This work has presented implementation, synthesis results, details regarding the internal operation and some issues for improvements of the implementation of a FFT algorithm in VHDL, aiming a synthesizable and flexible code.

This FFT processor is being designed to compose a classification algorithm, but its implementation has been done as a generic and fully parameterizable hardware description and it so can be used in several applications in areas like telecommunications, signal processing and several others, due of its large usage. As future works improvements should be done, such as implementation of radix-4 algorithm that enables to perform the same transform faster as shown in [3], to describe a faster memory and to increase its parallelization level.

REFERENCES