EFFICIENT DESIGN OF A FFT/IFFT-64 MODULE ON ASIC

A. Millán, M. J. Bellido, J. Juan, P. Ruiz-de-Clavijo, D. Guerrero, E. Ostúa, and J. Viejo *

Instituto de Microelectrónica de Sevilla - Centro Nacional de Microelectrónica
Av. Reina Mercedes, s/n (Edificio CICA) - 41012 Sevilla (Spain)
Tel.: +34 955056666 - Fax: +34 955056686
http://www.imse.cnm.es

Departamento de Tecnología Electrónica - Universidad de Sevilla
Av. Reina Mercedes, s/n (E. T. S. Ingeniería Informática) - 41012 Sevilla (Spain)
Tel.: +34 954556161 - Fax: +34 954552764
http://www.dte.us.es

{amillan,bellido,jjchico,paulino,guerre,ostua,julian}@dte.us.es

ABSTRACT

In this work we present the VHDL implementation of a FFT/IFFT-64 module. This implementation: (a) is relatively quick and (b) occupies a limited amount of area. The module operation is based on a radix-8 butterfly and it allows the calculation of a complex 64-element FFT/IFFT in 290 clock cycles providing a precision of 98.8% on the magnitude of the output samples. Area saving is achieved mainly by using a RAM macrocell in order to store intermediate calculations. The synthesis process has been carried out on ASIC using AMS 0.35 μm technology and reaching the place & routing level in the test process.

1. INTRODUCTION

The work that is presented in this paper is included inside the design and implementation of a modem. This modem bases its operation on the Orthogonal Frequency Division Multiplexing (OFDM) modulation/demodulation system [1]. OFDM transmits data by using a set of narrow bandwidth carriers. The frequency spacing of the carriers is chosen in such a way that the carriers are orthogonal in order to avoid interference among them. For both processes of modulation and demodulation, an OFDM modem needs to calculate the Discrete Fourier Transform of the data (in both direct and inverse ways). For this, a module that implements the FFT (Fast Fourier Transform) becomes an essential part of the system.

*This work has been partially supported by the MCYT META project TEC 2004-00840/MIC and the MECD/SEEU/DGU project PHB2002-0018-PC of the Spanish Government.

To be precise, the implemented system processes 64-sample complex symbols using 26-bit data (13 bits for the real part and 13 bits for the imaginary one). So, the dynamic range at the module input covers from -4096 up to 4095 (each part). Actually, this range represents values between -1 and 1. However, the FFT/IFFT module works with an internal 38-bit precision (19 bits for each part) in order to minimize the precision loss in both the internal operations and the final transform result. In any case, as the system works with 26-bit precision, the module output is truncated (being lost the six less significant bits of each part).

In a previous work [2], we have shown a first approach of the implementation of the system on FPGA. However, due to cost requirements, we have had to develop a new design. These requirements are:

- The system is going to be implemented on ASIC (instead of FPGA).
- The calculation time of the design keeps as a critical point due to timing restrictions in the whole modem.
- Now, the area occupied by the FFT/IFFT module becomes critical too, because the cost of the ASIC depends on the amount of used area. For this, we have been required to employ a RAM macrocell instead of a register set. As we will show, this reduces the area used significantly. However, it is a obstacle to the timing requirements because this type of RAM needs two cycles per access (a register set requires one cycle only).

The final publication is available in Proc. 11th Iberchip Workshop (IWS). pp. 305-306. Salvador de Bahia (Brazil). ISBN: 959-261-105-X

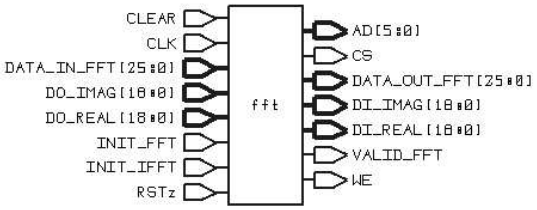

Fig. 1: Interface of the FFT/IFFT module.

So, due to these requirements we have to change the whole design in order to adapt it to the new specifications.

The organization of the paper is as follows: in Sect. 2 the interface and the internal structure of the module are presented; Sect. 3 shows the operation of the module; in Sect. 4, we present the design and synthesis details; Sect. 5 presents the tests we have carried out on the module; finally we will finish with the main conclusions of this work.

2. INTERFACE AND INTERNAL STRUCTURE OF THE FFT/IFFT MODULE

The module interface designed is shown in Fig. 1. Signals CLK, CLEAR, and RSTz are used to provide the module with clock, synchronous clear and asynchronous reset respectively. Signals INIT_FFT and INIT_IFFT indicate to the module in what way the transform must be calculated (direct or inverse way). Data buses DATA_IN_FFT and DATA_OUT_FFT (26-bit width) are the module data input and output (the 13 more significant bits store the real part and the 13 less significant bits the imaginary one). Signals AD, CS, WE, DI_REAL, DI_IMAG, DO_REAL, and DO_IMAG conform the RAM interface. Finally, signal VALID_FFT is generated by the module indicating that it has finished the transform calculation (Fig. 2).

Due to timing restrictions, it was required the transform to be calculated in less than 300 cycles. So, instead of choosing the classical structure based on a 2-point butterfly [3] (what would have required 384 cycles for the whole transform; since its order is $n \log_2 n$), it was necessary to choose a solution that maximizes the parallelism. The implemented solution is the one proposed in [4] and it is based on using a radix-8 butterfly [5, 6]. At a first approach, this butterfly allowed the calculation of the transform in 16 cycles only. However, this implementation enlarged too much the connection cost of the design (since it is necessary to access 16 data from RAM in parallel, leading to a data bus for the RAM of 304-bit width).

So, due to the requirements explained at the previous section, we have to develop a complex timing structure in order

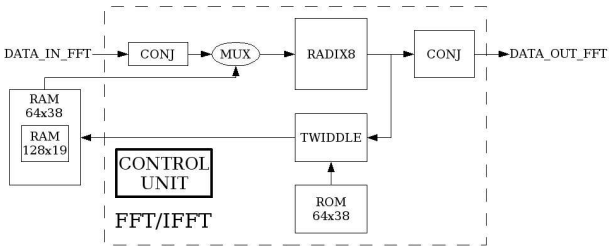

Fig. 3: Internal structure of the FFT/IFFT module. Conjugators operate in the IFFT case only.

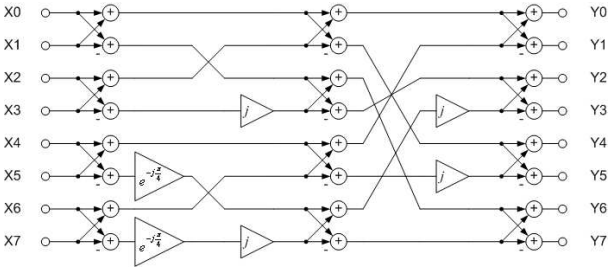

Fig. 4: Internal structure of the radix-8 butterfly.

to minimize the area while maintaining the maximum time restriction for the calculation.

The internal structure of the FFT/IFFT module is represented in Fig. 3. The most important component of the module is the radix-8 butterfly that performs an 8-element DFT in parallel. For it, the radix-8 is completed with two 8-register sets at its input and output data buses. Its implementation follows the structure proposed in [5] and it is shown in Fig. 4.

Around this main element, there are several components that allow to complete the operation of the system. The objective of each component is the next:

- CONJ: Performs the conjugate operation of the complex input data.
- RAM: Stores the intermediate calculations. Actually, this element is external to the module due to its macrocell nature.
- ROM: This component supplies the precalculated twiddle values (see next section).
- TWIDDLE: Multiplies the radix-8 complex data output and the corresponding twiddle value.
- CONTROL: It is the control unit of the module and coordinates the operation of all the components.

3. MODULE OPERATION

The module operation follows the algorithm proposed in [4]:

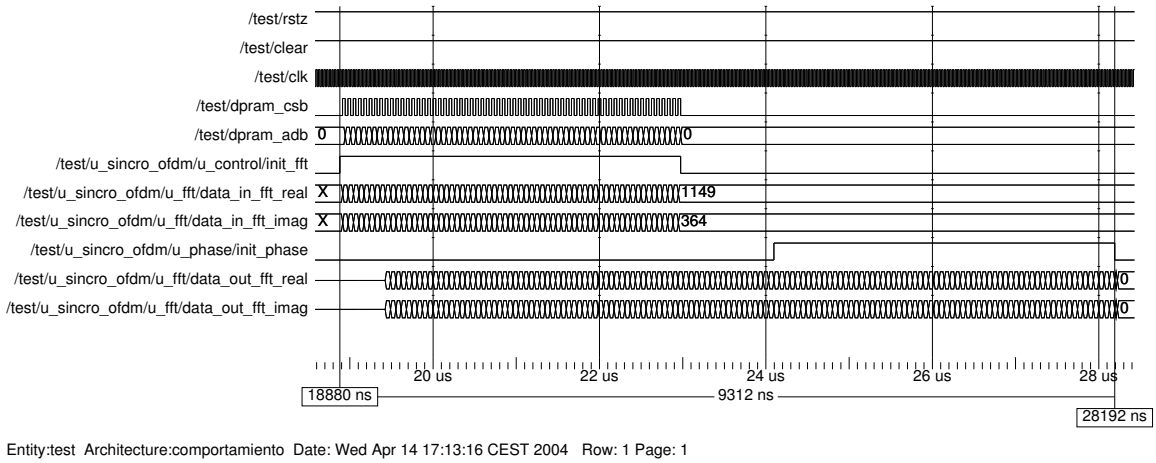

Fig. 2: Functional simulation. The calculation process starts when signal INIT_FFT becomes active. When the FFT results are ready, the module activates signal VALID_FFT (called INIT_PHASE by the next module in the modem). The whole process takes 9312 ns (ca. 290 cycles).

1. Consider that input data are stored in an 8×8 matrix A as shown in (1); the first eight data are stored in the first column (a_{i1} elements).

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{18} \\ a_{21} & a_{22} & \cdots & a_{28} \\ \cdots & \cdots & \cdots & \cdots \\ a_{81} & a_{82} & \cdots & a_{88} \end{bmatrix} \quad (1)$$

2. Perform the DFT of each row of A and store the result on the row itself (this is performed by the radix-8 butterfly).

3. Multiply each element of A by the corresponding twiddle (w). According to (2), twiddle values are into the range [-1, 1] (each part). So, the system works with their normalized equivalents in the range [-4096, 4095]. Actually, this step is performed at the same time that the previous one by multiplying each element before storing it on RAM.

4. Perform the DFT of each column of A and store the result on the column itself (performed by the butterfly).

$$w_{ij} = \exp\left(-j\frac{2\pi}{N}(i-1)(j-1)\right) \quad (2)$$

5. The final result is on matrix A considering the first eight output data to be at the first row (a_{1i} elements).

So, the operation can be summarized in four logical stages: (1) data input, step 1; (2a) first butterfly pass, steps 2 and 3; (2b) second butterfly pass, step 4; and (3) data output, step 5.

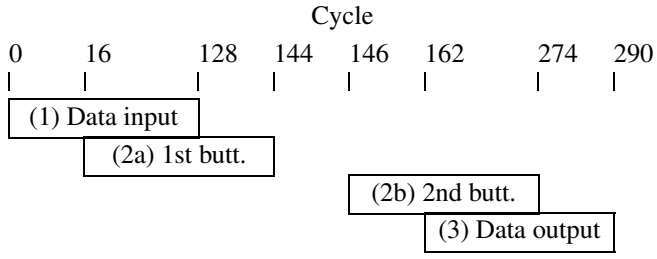

Fig. 5: FFT/IFFT module operation stages. Stages 2a and 2b can not be overlapped because the RAM data bus does not allow simultaneous read and write operations. Also, it is necessary to stablish a little wait stage of two cycles between them for synchronization purposes.

However, due to the hard timing requirements and the 2-cycle access RAM we have, these four logical stages have to be overlapped in order to minimize the total calculation time. So, the system operation is the one shown in Fig. 5. On the one hand, stages 1 and 2a are performed almost simultaneously. Now, input data are required to arrive in an interlaced way in order to dispose of the x_{1i} elements first. On the other hand, stages 2b and 3 are overlapped too. Now, output data is given in an interlaced way but this fact does not affect the operation of the next module in the modem.

4. DESIGN AND SYNTHESIS

The design of the module has been carried out using VHDL language and following a top-down methodology. For the whole RTL design we have used ModelSim SE PLUS v5.6d. In the first stage, we have designed the architecture of the module (Fig. 3). Next, we have developed the VHDL code of the module at the RT level. In this sense, it is important to denote how we have adapted the inter-

face needed to a high-level developing scheme but keeping it synthesizable at the logic level. So, the interface needed for the FFT/IFFT module was the next:

```
entity fft is
  port(
    DATA_IN_FFT  : in  std_logic_vector
      (2*(WIDTH-6)-1 downto 0);
    DATA_OUT_FFT : out std_logic_vector
      (2*(WIDTH-6)-1 downto 0);
    INIT_FFT     : in  std_logic;
    INIT_IFFT    : in  std_logic;
    VALID_FFT    : out std_logic;
    CLK          : in  std_logic;
    CLEAR        : in  std_logic;
    RSTz         : in  std_logic;
    CS           : out std_logic;
    WE           : out std_logic;
    AD           : out std_logic_VECTOR
      (5 downto 0);
    DI_REAL      : out std_logic_VECTOR
      (WIDTH-1 downto 0);
    DI_IMAG      : out std_logic_VECTOR
      (WIDTH-1 downto 0);
    DO_REAL      : in  std_logic_VECTOR
      (WIDTH-1 downto 0);
    DO_IMAG      : in  std_logic_VECTOR
      (WIDTH-1 downto 0)
  );
end fft;
```

However, a very important objective was to obtain a VHDL code that was defined at a high-level programming style. This fact has two main advantages because it makes easier: (a) the debug process (b) the future adaptation of the system to a new specification (e.g. different data width). In order to achieve this, the module actually has two main entities: (a) FFT, that implements the interface to the external modules and (b) FFTCORE, that implements the module operation itself. So, it allows the FFTCORE entity to have a high-level interface:

```
entity fftcore is
  port(
    clk     : in  std_logic;
    rst     : in  boolean;
    clr     : in  boolean;
    init    : in  boolean;
    ifft    : in  boolean;
    idb     : in  complex;
    done    : out boolean;
    odb     : out complex;
    ram_ab  : out address_bus;
    ram_we  : out boolean;
    ram_cs  : out boolean;
    ram_idb : out complex;
    ram_odb : in  complex
  );
end fftcore;
```

Types COMPLEX and ADDRESS_BUS are defined as:

```
type complex is record r, i:
  signed(WIDTH-1 downto 0); end record;
type address_bus is record i, j:
  integer range 0 to 7; end record;
```

All modules are combinational mainly (except for the case of the CONTROL UNIT). So, the programming style has been very modular to minimize the cost of the logic synthesis proces . Due to the complex timing structure of the FFT/IFFT module. Development of the CONTROL UNIT has been complex too. So, in order to guarantee the correct operation of the module and to make easier the debug process, we have implemented the corresponding Finite State Machine (FSM) according to the canonic structure described in [7]. This structure is based on using two functions: (a) the first one determines the next state of the FSM depending on the current state and the input data and (b) the second one describes the outputs or commands of the FSM in a similar way.

Table 1: Area occupied for each design section

Area	FFT core (μm^2)	Register set (μm^2)	RAM (μm^2)
Combinational	1.342	0.350	-
Noncombinat.	0.357	1.508	-
Net interconnect	0.334	0.320	-
Total area	2.033	2.178	0.614

In the second stage, we have performed the logic synthesis of the module. For this task, we have used Design Analyzer v2001.08 on AMS 0.35 μm technology (CSD process). Also, we have verified the operation for the required work frequency of 42 MHz. This process has become very difficult because the area restrictions. The main point in saving area is the internal calculations store (Table 1). In a first approach, we employed a 64-register set; however this design enlarge too much the interconnection area (covering a total area of 4.211 μm^2). In a second approach, we decided to use a RAM macrocell. However, the minimum RAM macrocell that AMS provides is 128-word length. So we had to waste the half of the capacity. Altough this solution was better than the previous one, we thought it was mandatory to make good use of the whole area. So, we designed a 64$\times$38 single-port RAM from the 128$\times$19 double-port RAM that AMS provides (covering a total area of 2.647 μm^2 and saving about 37%).

Finally, in the third stage, we have performed the layout synthesis (Fig. 6) by using Silicon Ensemble v5.4. The layout final total area is 3.028 μm^2. The underestimation produced is due to two main reasons: (a) the previous estimation at the logic synthesis level is usually lower than the final one and (b) the previous estimation did not include the interconnection cost between the FFT module and the RAM macrocell.

5. TESTS

The debug process we have followed consists of providing the designed module with a set of signals (actually used on the system) covering SNRs from 5 dB up to 30 dB.

This test bench has been applied at three levels. At the first test stage we have verified the system at a functional

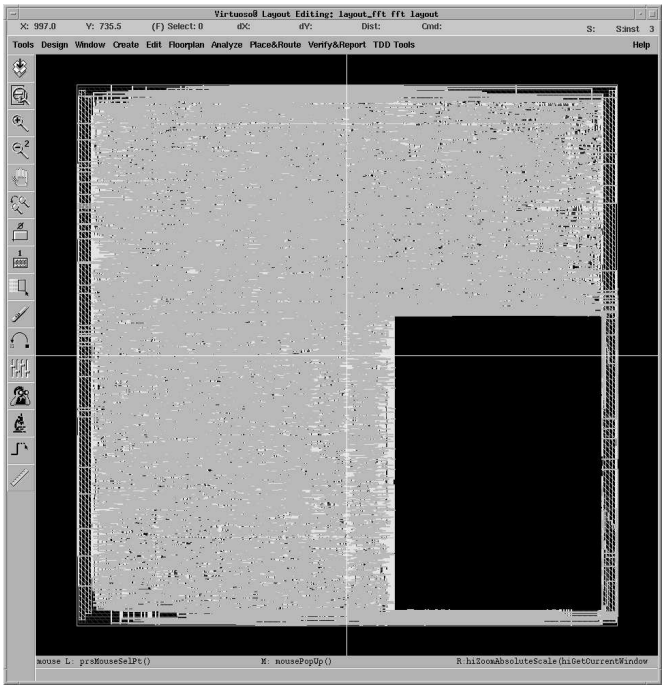

Fig. 6: FFT/IFFT module layout generated with Silicon Ensemble v5.4 on AMS 0.35 μm technology (CSD process). We can observe the RAM macrocell at the bottom right.

level. Also in this stage we have compared the results with the ones provided by the MATLAB system model in order to estimate the precision of the module. At the second test stage, we have synthesized the module and tested its operation considering the gate delay by post-synthesis simulation. Finally, at the third test stage, we have obtained the layout of the whole design and tested its global operation.

About the result error, it is lower than one bit of the output data. Taking into account that output data is truncated (loosing the six less significant bits), the maximum quantization error is of 2^6 levels of a total of 2^{19}. It produces a maximum relative error of 1.2%. So, the precision of the system is 98.8%.

6. CONCLUSIONS

In previous sections, we have presented the VHDL implementation of a FFT/IFFT-64 module on ASIC. This implementation is relatively quick by performing a complex 64-element FFT in 290 clock cycles (at a clock frequency of 42 MHz). Also, the design occupies a limited amount of area: it needs 2.647 μm^2 using AMS 0.35 μm technology. Area saving is achieved mainly by using a RAM macrocell in order to store intermediate calculations. Finally, the test process has reached the place & routing level and the precision of the FFT result is of 98.8%.

REFERENCES

[1] Weinstein, S.B., Ebert, P.M.: Data transmission by frequency-division multiplexing using the discrete Fourier transform. IEEE Transactions on Communications **19(5)** (1971) 628–634

[2] Millán, A., Bellido, M.J., Juan, J., Ruiz-de-Clavijo, P., Guerrero, D., Ostúa, E.: Diseño eficiente de un módulo FFT/IFFT sobre FPGA. Proc. III Jornadas sobre Computación Reconfigurable y Aplicaciones (JCRA) (September 2003) 107–114

[3] Smith, S.W.: The Scientist and Engineer's Guide to Digital Signal Processing, 2nd ed. California Technical Publishing, San Diego, California (1999)

[4] Bailey, D.H.: FFTs in External or Hierarchical Memory. NAS Technical Report RNR-89-004 (1989)

[5] Widhe, T., Melander, J., Wanhammar, L.: Design of efficient radix-8 butterfly PEs for VLSI. IEEE International Symposium on Circuits and Systems (ISCAS97), Vol. 3. Hong Kong (1997) 9–12

[6] Li, W., Wanhammar, L.: Efficient radix-4 and radix-8 butterfly elements. NorChip99. Oslo (1999)

[7] Bergé, J.M., Fonkoua, A., Maginot, S., Rouillard, J. VHDL Designer's Reference. Kluwer Academic Publishers (1992)