Hybrid Architecture for Data-dependent Superimposed Training in Digital Receivers

Fernando Martín del Campo, René Cumplido,
Roberto Perez-Andrade
National Institute of Astrophysics, Optics and Electronics
Computer Science Department
C.P. 72840, Puebla, Mexico
{fmartin,rcumplio, j_roberto_pa}@inaoep.mx

A. G. Orozco-Lugo
CINVESTAV-IPN
Section of Communications
C.P. 07360, Mexico City, Mexico
aorozco@cinvestav.mx

Abstract

Many digital communications algorithms present characteristics that make very difficult to implement them in either a software solution or as a fully custom hardware architecture. Their inherent complexity implies two challenges at the same time: to process the information as fast as possible to present the results when they are required, and to build a system that meets the power consumption and space constraints imposed by the application, while trying to maintain a low design intricacy. This work describes a hybrid hardware-software architecture designed to run a wireless communication algorithm named Data-dependent Superimposed Training. The resulting system can be used partially or in its totality to implement many other algorithms with similar needs, and in fact it is an interesting source of information for implementing solutions for some of the most common operations encountered in the DSP field.

Keywords

Communications Algorithms, Data-dependent Superimposed Training, Hybrid Architecture, Hardware Accelerators.

1 Introduction

As higher amounts of information are transmitted through the current communication systems, processing speed requirements have risen more and more. In addition, many of the operations required for the modulation, channel encoding and decoding, and encryption, among others, are very complex. Nevertheless, the cost and time required by the design, building and testing of ASICs solutions have motivated the emergence of new tools for the creation of proof of concept designs, like the FPGAs. These platforms have evolved to the point where it is possible to create full computational systems in one programmable chip, that can run a series of software applications and can communicate to other peripherals both in the FPGA board and outside of it. All these characteristics are, at a certain point, very useful to implement a solution for a great variety of DSP problems found in digital communications algorithms. This work concentrates in a wireless communications method called Data-dependent Superimposed Training (DDST), which is deeply discussed in [4], and whose preliminary implementation (which only covers the channel estimation stage without the data recovery and with no acceleration for the complex exponentials problem) can be found in [6].

1.1 Processes commonly found in Digital Communications Systems

There are many operations that DSP techniques, used in digital communications, have to solve in order to accomplish their purpose. As the number of embedded multipliers in integrated circuits is continuously increased, the efforts in this field are concentrated in solving more complex operations, like mathematical transforms (Fourier, cosine) or transcendental functions, as exponential, hyperbolic, logarithmic, power, and periodic functions.

Discrete Fourier transform and its inverse have an efficient and relatively high speed implementation in the form of the Fast Fourier Transform (FFT) algorithm. Several architectures have been proposed ([1], [2]), and the leading FPGA manufacturers offer them as modules in their IP libraries.

Transcendental functions have been solved through tech-

978-0-7695-3474-9/08 $25.00 © 2008 IEEE
DOI 10.1109/ReConFig.2008.52

niques and implementations that are rarely as efficient as the FFT. For example, square root (an operation that belongs to the power functions family) is usually implemented through restoring and non-restoring iterative algorithms, which increment the resolution of the calculation in only one bit per iteration [3]. On the other hand, solutions for periodic functions can be divided in two groups: iterative methods like the CORDIC and BKM generators, and methods based on mathematical series, like the Taylor and Chebyshev solutions. This last kind of implementations have been out of reach for the majority of the hardware architectures, due to its high complexity and to the great amount of resources that are required. Iterative solutions are commoner, but one of their characteristics is its usually low speed, unless a full pipelined approach is used, which increments the amount of necessary resources and the complexity of the design. Moreover, this technique is not suitable for applications where just a few functions have to be resolved each time.

An additional characteristic of several digital communications algorithms is the large quantity of information that has to be processed and the high amount of data dependencies among the different stages of the execution. The first of these qualities makes it difficult to fit the design into the available resources, complicating the management of all these data as the implementation of the algorithm runs. The second one difficults to use techniques as parallel processing and pipelining, because a stage of the process has to wait for the results of previous ones before performing its functions.

DDST is the perfect subject of study, due to the necessity for performing several multiplications, square roots, direct and inverse FFTs, trigonometric functions (sine and cosine calculations), and other complex operations on a high quantity of data. All the stages of the process depend on the results of the previous ones, so it is difficult to implement parallel processing in the architecture. The following section will shortly describe the main problems faced while designing the architecture, and the solutions implemented to overcome them.

2 The implementation

Before going into detail, it is important to mention that the resulting architecture combines both a hardware and a software approach, because this solution gives the system a high processing speed for large amounts of data, while maintaining a simple control interface. The final result is easily configurable and to modify and even to replace an important section of it by another one, as improvements in the DDST algorithm are made, is a simple process.

2.1 Vector reshaping and arithmetic mean obtaining

Obtaining the arithmetic mean, or average, from a set of data, is an operation required by a great amount of DSP techniques, as its statistical properties can be used to exploit a pattern in the information, or just because it allows the management of a smaller amount of data that still exhibit some characteristics of the original set. DDST requires the execution of this operation several times along its processing, but the average depends on the reshaping of a vector into a matrix, that is, starting from a vector V of $Nelements$, a matrix is obtained by the rearrangement of such elements as a matrix of dimensions $|M \times P|$, where P is equal to N/M. An example of this operation is a vector V of 12 elements, that is then reshaped as a matrix Mv of dimensions $|4 \times 3|$. Once the matrix has been obtained, the average of each row is obtained as shown in figure 1.

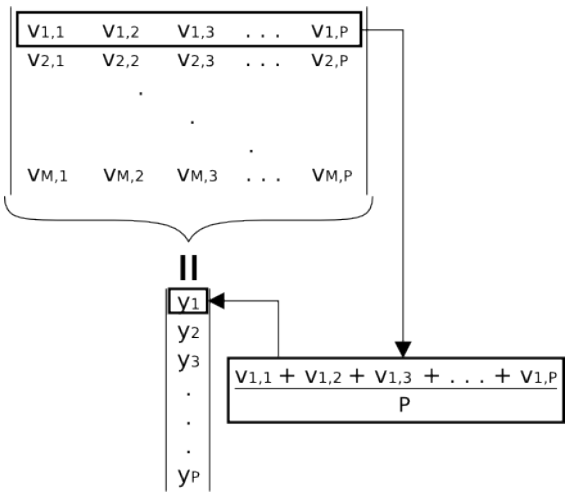

Figure 1. Arithmetic mean from the rows of a matrix.

As it can be appreciated, the result of applying all the averages is a new vector Y of M elements $(y_1, y_2, \cdots, y_P)$.

These two operations are accelerated by a hardware module that reads, directly from a dedicated on-chip memory (a storage structure that manage one or more of the memory blocks of the FPGA), **M** data, each one of 32 bits, accumulating their respective values to **M** 32 bits registers. At the end of the process, they are multiplied by 1/(P), so now they contain the arithmetic means of the rows from the reshaped matrix (the explained vector Y). Finally, the results are stored in another on-chip memory. Figure 2 shows the architecture of this module (control is not explicitly included). The shown multipliers (right side of the picture) work with 64 bits arithmetic, so it does not experiment

any resolution loss until the final result is truncated to 32 bits. This way, the same values obtained by a single precision floating point unit (or software implementation) are obtained by this module with more simple operations.

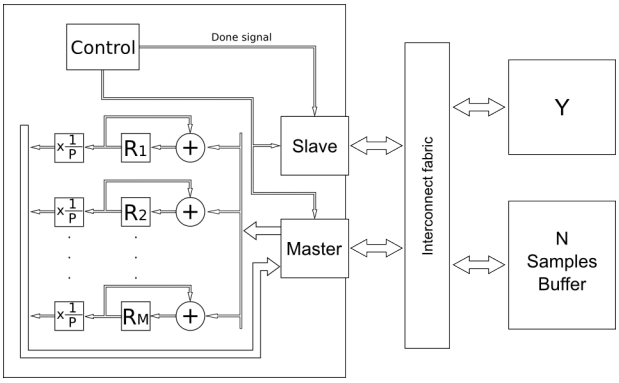

Figure 2. Architecture of the vector reshaping and average accelerator.

2.2 Magnitude of a vector of complex elements

There are several algorithms in which it is necessary to work with only the magnitude of the complex elements of a vector. The high complexity of these operations comes not from the two multiplications required to obtain the squares of the real and imaginary parts of a complex number, but from the necessity to perform a square root.

Figure 3 presents a block diagram of the implemented magnitude accelerator that, as its name says, calculates each one of the norms from the complex samples in a vector V of n elements, as shown in (1).

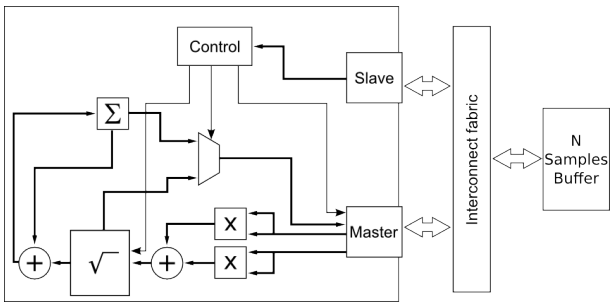

Figure 3. Architecture of the magnitude hardware accelerator.

$$\sqrt{{v_r}^2(k) + {v_i}^2(k)} \tag{1}$$

with $V = \{v_r(k) + v_i(k) * i \in \mathbb{C} \mid \forall k \text{ from 0 to n-1}\}$

The module has several advantages over the software-only version:

1. It fetches both the real and the imaginary parts of the complex numbers each time it performs a read operation.

2. It works with 64 bits arithmetic, so there is no change in the accuracy of the result.

3. As the square root is calculated by the means of a hardware submodule (explained bellow), it runs faster than a software-only version running in the same system, as it does not need to change from fixed to floating point arithmetic.

4. It accumulates all the magnitudes as it works, so at the start of the process it can be decided if it will return either all the norms of the vector, or only their summation, depending on the requirements of the stage in which the module is used.

5. It is possible to assign an *"offset"* so, for example, the module only obtains the norm of the complex samples in positions 0, 4, 8, ..., etcetera, and not from every sample in the input vector.

6. It has little latency, as it obtains and stores data from and to dedicated on-chip memories.

2.2.1 Square root

The square root is solved by a submodule in which the 8 more significant bits of the root are obtained from a look-up table. This could be considered an approximated root, that then can be *fine tunned*. Each iteration calculates, in parallel, 4 bits of the root, and not only one. This system is depicted in figure 4.

The approximated root is obtained from the look-up table using as index the most significant bits of the radicand. Then this value is appended to a set of possible roots that are squared and compared to the original radicand. A comparator tree evaluates all the results and decides which of the possible roots gave the smallest error. This value is then updated, as the new approximated root and the next four bits are calculated. With each iteration, the approximated root of the possible set of roots grows four bits, until it reaches the least significant bit. In addition, the submodule stops as soon as it finds an exact root, so not all entries take the same amount of cycles to be calculated. The bit length of the look-up table memory is important because of the trade-off between the the number of iterations necessary to have the best square root calculation and the amount of storage capacity necessary to accommodate the approximated roots.

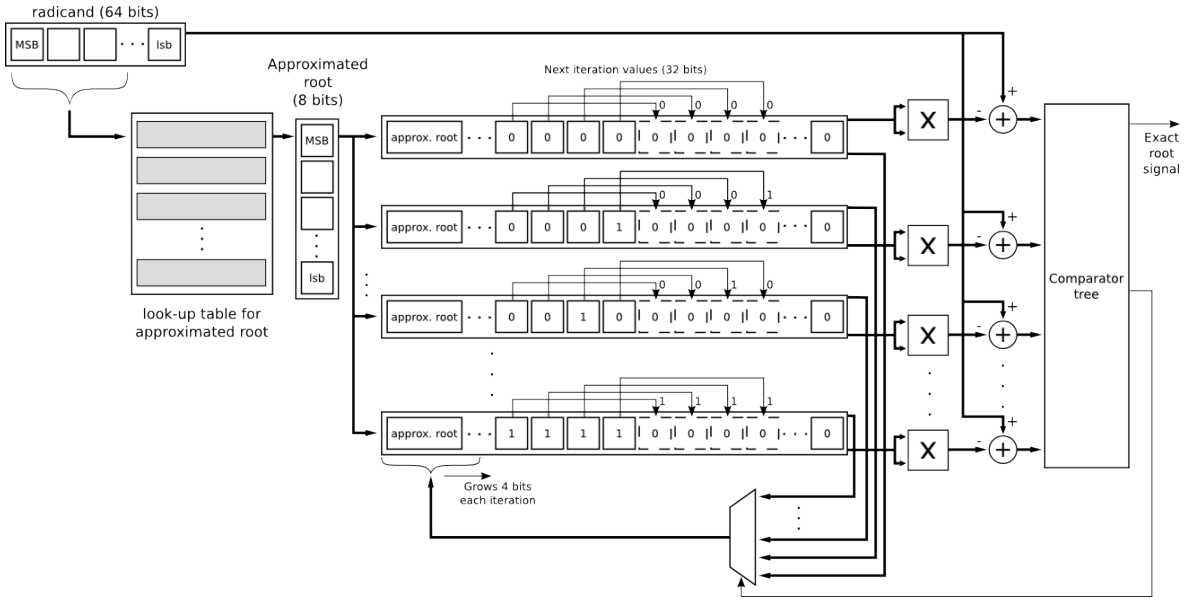

Figure 4. Architecture of the root hardware submodule.

2.3 FFT

As it was mentioned in the introduction, there are many FFT implementations, so it was decided that this problem was tackled from the perspective of the following fact: DDST,as many other algorithms, is a technique that is currently under research, so it is possible that several parts of its associated algorithm change when a better option is discovered. An easy to modify and flexible architecture is a very desirable quality in this situation. In addition, it would be better if the changes to the architecture could be done by a person with little knowledge of the system. Because of this, the objective of this section of the architecture was to found a solution that would satisfy as much as possible these characteristics.

The implementation uses a modified version of an Altera FFT example that uses the manufacturer's tool named C-to-Hardware Acceleration Compiler, or C2H compiler for short, that can convert C language subroutines into hardware accelerators. Even though the designer does not have full control over the process, C2H can improve some of the architecture's design and implementation time. Obviously, this technology is hardly dependent on the Altera technology, but the combination of other options like Impulse C with Xilinx ISE and XPS, or the use of Celoxica's Handel C along with the tools of Xilinx or Altera, can be used to obtain very similar characteristics. Under these schemes, modifications on the original algorithm can be implemented by modifying a section of the software part of the system, which is then compiled into a hardware accelerator and not only into an executable code.

2.4 Complex Exponential (as trigonometric functions)

The majority of the implementations that deal with complex exponentials exploit the Euler formula ($e^{ix} = cosx + isinx$) to solve them as a pair of trigonometric functions (sine and cosine).

Advantages and disadvantages of the different approaches were already discussed, but it was also mentioned that the implementations based on mathematical series have been out of reach for the majority of the hardware architectures, until recently. While Taylor series are still very complex and slow (because of their large convergence time), there is another kind of series that are more suitable for hardware solutions: the Chebyshev polynomials, that converge faster and need fewer calculations to obtain the same resolution [5].

As the grade of the polynomial grows, the precision of the function approximation also grows. Equation (2) shows the iterative rule that allows the calculation of any Chebyshev polynomial:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \quad \forall k \geq 2 \qquad (2)$$

Figure 5 shows the block diagram of the accelerator based on the Chebyshev approximation. Cs_K and Cc_K indicate constants that are equal to the Chebyshev coefficients that multiply each element of the polynomial. The first cycle they are multiplied by the input value that is stored in β. The next multiplications depend on the degree of the polynomial element. For example, the third element (Rs_1) will be updated by the results of three multiplications, to obtain

the value $Cs_3 \cdot x^3$. Once all the elements from both sine and cosine polynomials have been added, the estimated values are multiplied by a complex number from the vector to operate. This value is read from the SDRAM and the complex product is written to the same memory. Once this process is finished, the value of β is updated according to k (from 0 to N), and the same explained operations are performed again. In total, the whole process is performed N times.

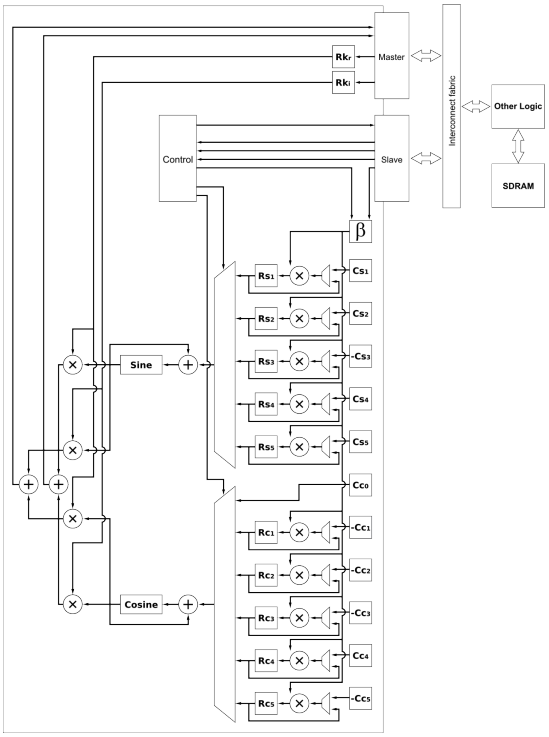

Figure 5. Architecture of the complex exponential accelerator.

3 Full system

A pure HDL implementation of the DDST architecture requires a very complex control, and also in a logic that cannot fit on the majority of FPGAs without sacrificing speed for resources usage. The alternative proposed in this work is a *system on a chip* that runs a series of C programs, but leaves the most *computer intensive* or *memory demanding* operations to the special hardware accelerators described in the past section. A very general architecture of the system can be seen on figure 6.

There are two kind of ports that can access or be accessed through the interconnect fabric: the slave and the master. Slaves are used to receive signals from other components of the system so they can be controlled. Meanwhile, masters can manage other components and perform actions like doing a memory read or write. In *SOPC builder* (the Altera tool for the building of embedded systems), a master can read or write up to 1024 bits on each memory access and not only can they communicate with on-chip memories, but also with any other memory device in the system. All that is needed is the base address of such memory and the existence of a controller for this last one. Those controllers are usually provided by Altera, like in the case of the SDRAM.

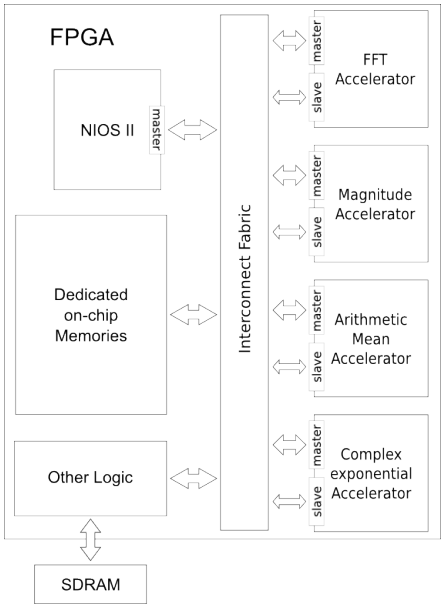

Figure 6. Architecture of the full system.

4 Results

It is difficult to compare the performance of a hybrid architecture with other systems, overall because the algorithm is expected to run in mobile devices. It was decided to compare the performance and physical characteristics of a software-only implementation also based on the NIOS II processor running in the same FPGA against the hardware accelerated system.

Implementation	Software	Hardware
Max. Frequency	223.67MHz	223.67MHz
Space (Total)	29%	87%
ALUTs	10%	80%
DSP Blocks	10%	100%

Table 1. Synthesis Summary

Operation	Implementation				Performance increment
	Software only		Hardware accelerated		
	cycles	time [ms]	cycles	time [ms]	
Vector reshape and average (1024 elements)	160645	1.61	2223	0.02	80.5x
1024 points FFT	1541710	15.42	56701	0.50	30.84x
Norm of 1024 complex data	3131035	31.33	116597	1.17	26.78x
18432 cosine and sine calculations	354246321	3542.19	3710402	36.93	95.91x
Total time of DDST execution	438653043	4386.53	14018812	45	31.29x

Table 2. Comparisons between software-only and hardware optimized operations

Data relative to the physical characteristics of the systems are shown in table 1, while time comparisons for different operations performed in both implementations can be found in table 2.

The number of performed operations in table 2 are typical of DDST simulations, so they give a pretty good idea of the time that a real implementation would require before obtaining the expected results.

5 Conclusions

An alternative solution that uses both a hardware and a software approach was developed to allow the implementation of a digital communications receiver based on Data-dependent Superimposed Training. A hardware only approach allows the building of fast processing systems, but problems like the one of the DDST receiver show that sometimes the necessary logic for the control of these systems presents a very high complexity. Moreover, when the amount of data to process is very high, and several data dependencies are present, techniques like parallel processing or pipelining are difficult to exploit, usually leading to architectures so large they cannot fit into mid-range FPGAs. On the other hand, when the solution is based only on a software approach, processing speed is very low for the requirements of many digital communications devices (like in the case of the mobile systems).

The hybrid software-hardware approach demonstrated to be very versatile and flexible, allowing fast implementation of several kinds of algorithms and their fast modification, from a small change in the input parameters values to the alteration of a full stage of the process. In fact, the built prototype fits perfectly into the study of the DDST algorithm, as this algorithm is still under study and constant modifications and improvements have been made over it. If future changes in the algorithm require a significant modification of any of the accelerators, it will be very easy to adapt the whole system.

Additionally, some efficient solutions for typical DSP problems were designed and tested, like the described *look-up tables / parallel / iterative* implementation for the square root calculation, and the parallel Chebyshev approximation architecture for the solution of trigonometric functions (and consequently of complex exponentials).

References

[1] S. Saponara, L. Fanucci, L. Serafini: Low-Power FFT/IFFT VLSI Macro Cell for Scalable Broadband VDSL Modem, iwsoc, p. 161, The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications (IWSOC'03), 2003.

[2] Jose Alberto Vite-Frias, Rene de Jesus Romero-Troncoso, Alejandro Ordaz-Moreno: VHDL Core for 1024-Point Radix-4 FFT Computation, reconfig, p. 24, 2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig'05), 2005.

[3] Piromsopa, K, Aportewan, C. and Chongstitvatana, P.: An FPGA implementation of a fixed-point square root operation, Inter. Symposium on Communications and Information Technology, Vol. 14-16, Thailand, 2001, pp. 587-589.

[4] Enrique Alameda-Hernandez, Desmond C. McLernon, Aldo G. Orozco-Lugo, Mauricio Lara and Mounir Ghogho: Synchronization and DC-Offset Estimation for Channel Estimation Using Data-Dependent Superimposed Training, EUSIPCO 2005, Turkey, September 2005.

[5] J. C. Mason, David, Christopher Handscomb: Chebyshev Polynomials, CRC Press, 2001, ISBN: ISBN:0849303559, pags. 1-3, 105-112.

[6] Fernando Martín del Campo, René Cumplido, Roberto Perez-Andrade, A. G. Orozco-Lugo: A SOPC Architecture for Data-dependent Superimposed Training Channel Estimation, Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC) 2008, Barcelona, July 2008.