A&QT-R 2002 (THETA 13)
2002 IEEE-TTTC International Conference on Automation, Quality and Testing, Robotics
May 23 25, 2002, Cluj-Napoca, Romania

# Co-FFT Design: FFT Implementation on CSoC

S. Hessabi
*hessabi@sharif.edu*

A. Ahmadinia
*ahmadinia@ce.sharif.edu*

G. Asadi
*asadi@ce.sharif.edu*

S. Bayat Sarmadi
*bayat@ce.sharif.edu*

M. Gudarzi
*gudarzi@mehr.sharif.edu*

*Department of Computer Engineering*
*Sharif University of Technology*
*Tehran, IRAN*

**Abstract**

The most considerable criteria in Fast Fourier Transform (FFT) implementation is performance. Digital Signal Processors are used for performing FFT. In this paper, FFT algorithms are implemented on a System-on-Chip using codesign technique. The results show speed up in some cases.

**Keywords**

System-on-Chip, Codesign, Fast Fourier Transform, Digital Signal Processor

## 1. Introduction

Fast Fourier Transform (FFT) is an efficient method of computing discrete fourier transform (DFT.) In FFT, the symmetry and periodicity properties of DFT are used. The trend of designers is to implement FFT as efficient as possible. Both digital signal processors and FPGAs have been used to implement FFT algorithms. In this paper, we have implemented FFT on a System-on-Chip, and have tried to make it as fast as possible.

High complexity of modern digital signal processing systems and the increasing demand for short time-to-market are the current challenges of VLSI designers. Improvements in silicon technology have enabled us to combine more functionality on a single chip. These functionalities are implemented using different architectures (e.g., hardwired ASIC blocks, Full-Custom Macros, off-the-shelf DSPs, etc.) within the intellectual property (IP blocks).

Systems-on-Chip (SoC) represents the following advantages in comparison to the well known "Systems-on-a-board":

- lower cost for consumer applications (high volume)
- higher inter-block communication bandwidth (shorter delays, no pin-limit)
- higher flexibility because of programmable components
- low-power consumption

Programmable SoC is ideally suited for embedded systems, where time-to-market is a primary concern. In particular, the time-to-market benefits and high integration level are proving to be a strong fit for communications, portable, embedded Internet, and instrumentation markets [2]. We have used E5 CSoC from Triscend for implementing FFT.

To evaluate the FFT performance implemented on E5 CSoC (Configurable System-on-Chip) [2], the results should have been compared with FFT implemented on a digital signal processor such as TMS320C25 [9]. Therefore, the FFT algorithm has been implemented on both and the result is shown using a proper algorithm (Radix-4) and hand-optimized assembly code. It can be seen that CSoC has a better performance, but there are some problems such as Special Function Registers (SFRs) [5] limitation and long communication time, which prevent increasing the number of FFT point.

Here, radix-4 and radix-2 cores are used. In addition, time consuming modules such as radix cores and multiplications are implemented in hardware while the other parts are implemented in software.

The rest of this paper is organized as follows. In Section 2, we describe the fast fourier transform algorithms. FFT implementation on TMS 320C25 is introduced in Section 3. In Section 4, we describe the FFT implementation on Triscend CSoC. Experimental results are introduced in Section 5. Finally, the conclusions is given in Section 6.

## 2. Fast Fourier Transform Algorithms

The fourier transform is a versatile and powerful analysis tool for a variety of fields of science and engineering. Common applications of the fourier transform include convolution, filtering, power spectral analysis, correlation and autocorrelation of data. FFT is an efficient algorithm used to compute a fourier transform in O (NlogN) operations (order of NlogN).

The FFT methods are [6]:

1- Decimation in Time 2- Decimation in Frequency

Both approaches have equivalent number of multiplications and additions. Common FFT algorithms are as follows [8]:

1.Radix-2 FFT 2.Radix-4 FFT 3.Split-Radix FFT

**Table 1. Number of Multiplications for FFT algorithms**

| | |
|---|---|
| Radix-2 | 1/2 N log2(N) |
| Radix-4 | 3/8 N $\log_2(N)$ |
| Split-Radix | 1/3 N $\log_2(N)$ |

The number of complex additions is $N\log_2 N$ for all above algorithms and the number of complex multiplications is shown in table 1 [8].

Radix-2 is not as efficient as Radix-4. On the other hand, Split-Radix algorithm is really

complex [1][3]. Also, in Radix-n algorithms (for n = 8, 16, and so on) performance decreases. So, the Radix-4 algorithm is the best choice.

## 3. FFT Implementation on TMS320C25

The TI TMS320C25 digital signal processor offers many advantages for the implementation of FFT algorithms. Its 80-ns cycle time and special features, such as the single cycle multiplication, allow high execution speed [4][7]. The 544 16-bit words of on-chip memory permit the implementation of a 256-point complex FFT without access to external memory, thus further reducing execution time. Furthermore, special instructions, such as RPTK and BLKD, allow the quick transfer of data from external to internal memory, so the portions of large FFTs can be implemented with the on-chip RAM. Due to the flexibility of the TMS320C25, the designer can make a trade-off between program memory versus execution speed. The execution time of different number of FFT points for TMS320C25 is given in table 2.

**Table 2. FFT Performance for a TMS320C25 Implementation**

| FFT EXECUTION | | | |
|---|---|---|---|
| SIZE | CYCLES | CLOCK | TIME |
| 16-Pt | 2780 | 12.5 MHz | 222.4 us |
| 32-Pt | 5478 | 12.5 MHz | 438.24 us |
| 64-Pt | 10945 | 12.5 MHz | 875.6 us |
| 128-Pt | 21879 | 12.5 MHz | 1750.32 us |

In general, bit reversal or data scrambling must be performed either at the input stage on the time samples or at the output stage on the frequency samples. Bit reversal can be performed in-place. Such a process generally requires the use of one temporary data memory location.
Because of its double-precision accumulator and its versatile instructions set, the TMS320C25 processor can perform in-place bit reversal or data scrambling without the use of a temporary data memory location.

## 4. FFT Implementation on Triscend E5 CSoC

The Triscend E5 configurable system-on-chip integrates, on a single device, a performance-enhanced 8032 "Turbo" embedded micro controller , a block of SRAM, a high-speed dedicated system bus, and configurable logic intimately connected to the processor and system bus. Its SRAM includes internal RAM (256 Byte Special Function Registers) and external RAM (XData) where the internal one is faster than the external. The E5 family is a highly integrated, fully static single-chip system optimized for embedded systems applications [2]. Therefore, we used E5 CSoC for FFT implementation.

For implementing FFT, floating-point format or fixed-point format can be chosen. Floating-point is much more accurate than fixed-point, but fixed-point, which is used in TMS320C25, is faster and simpler. Therefore, for comparing the results of SoC implementation with TMS320C25 implementation, it would be better to use fixed-point format. Also, by using floating-point format, the hardware becomes more complicated. On the other hand, there are some optimized built-in modules in Triscend software tool (Fastchip) for implementing fixed-point format. As a result, fixed-point is the choice.

At first, the FFT algorithm was simulated by software programming (by C++ language under VC6). After verifying the functionality of the algorithm, the execution time of each program

part was analyzed by C++ Profiler. Some of the time consuming parts were implemented in hardware and the best case was chosen. Implementation details are as follows.

**4.1. Format Determination**

Initially, floating-point IEEE standard format was chosen. There are two ways for implementing multiplier and adder: 1- using Hardware Description Languages (HDLs), 2- using Fastchip built-in modules.

The multiplier, which was implemented by HDL, used 52% of available Configurable System Logics (CSLs). Since, four multipliers were needed, the HDL module could not be used. Even if the number of bits was reduced, only 20% of CSLs were released, which could not solve the problem. Also, the multiplier, which was implemented by Fastchip built-in modules, used 18% of CSLs. But the adder implemented by these modules, are very complicated. As a result, floating-point IEEE standard format is not suitable for this platform.

The next choice is fixed-point. As the Fastchip built-in modules are optimum, the multiplier and adder were implemented by them. The amount of used CSLs for an adder and a 16-bit multiplier are at most 2% and 12%, respectively. Since a 32-bit multiplier needs four 16-bit multipliers and three 32-bit adders, the amount of used CSLs for it is more than 50%. As a result, using 16-bit fixed-point format is the best case.

**4.2. Simulation Step**

FFT algorithms are inherently recursive, but this recursive approach is not appropriate for our design, because there are a large number of calls which result in a huge stack memory. In addition, the program performance is lower. Thus the algorithm has been converted into an iterative one.

In this program, 16 bits fixed-point format is used, where the first byte is integral and the second is fractional. After multiplying the two numbers, eight bit shift is needed to move the point to the correct position. But this way, the high byte will be missed. Therefore, before multiplication, each number is shifted four bits. The relevant code is given in Figure 1.a.

For increasing the program performance, the sinus and cosine functions have been implemented in a table. This can be done, because the maximum number of FFT points is known. Figure 1.b shows the main part of the program.The two time consuming parts of the algorithm are Radix-2_calculation_routine and twiddle_calculation_multiplication_routine. The twiddle function is shown in Figure 1.c.

**4.3. Mixed (Software/Hardware) Implementation Step**

According to the program, in time consuming section, there are three basic parts, as follows: 1. Adder 2. Multiplier 3. Sinus table

The ideal case is moving all of them to hardware. At first, because of using 32-bit format, hardware implementation of all three parts was impossible. But, by using 16-bit fixed-point format, all of them can be implemented in hardware.

In the final design the radix-2 block and twiddle block (Sin&Cos&Addition& Multiplication) were implemented in hardware, while other blocks were implemented in software.

## 5. Experimental results

At first, the software implementation part was written in C and translated into assembly using Keil Software. But in this case, the execution time was very high compared to TMS320C25

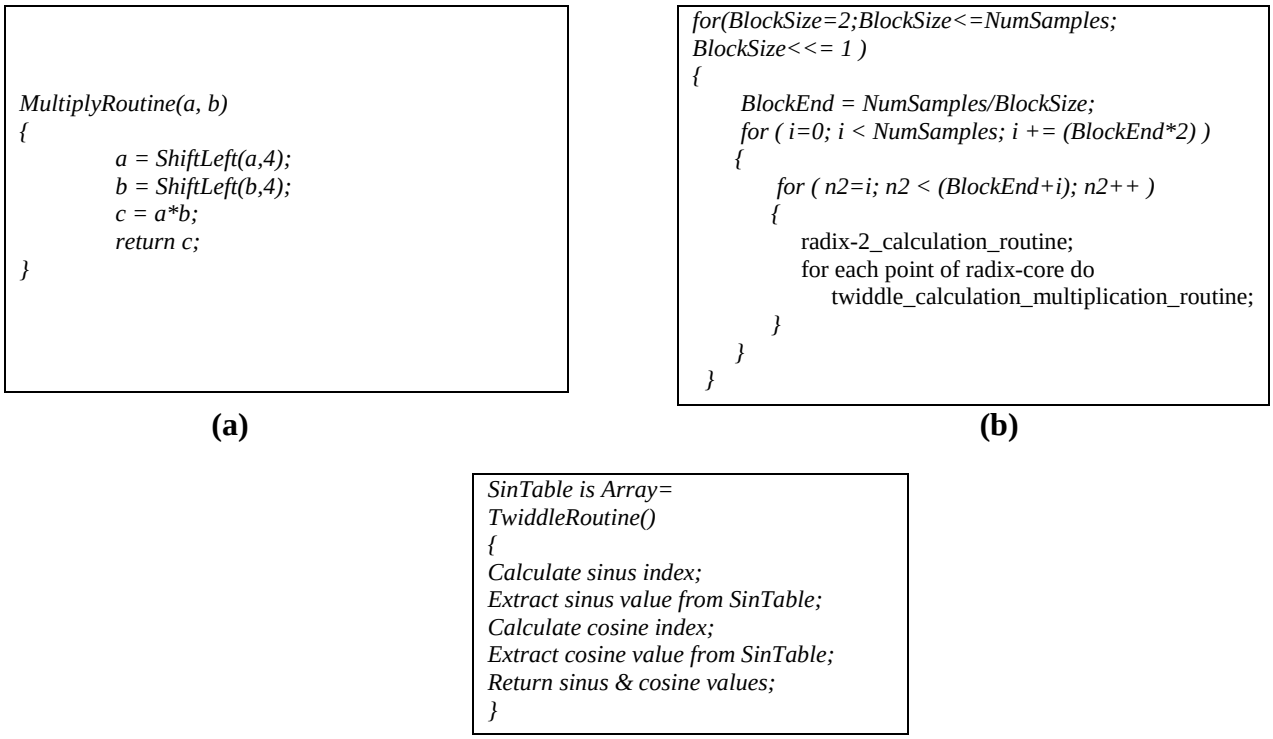


**Figure 1. (a) Multiplication routine (b) Overall algorithm (c) Twiddle routine**

execution time. In order to achieve a better result, the code was directly written in assembly and therefore, the consuming time of executed code was reduced.

The maximum path delay in hardware implementation for each algorithm with different number of points is shown in Table 3.

The execution time of FFT algorithms on CSoC and TMS320C25 for different number of points is obtained (Table 4 shows the results.) CSoC_Time1 represents total elapsed time with assumption of sufficient internal RAM and CSoC_Time2 represents spent time with 256 byte internal RAM.

For obtaining, time values of CSoC_Time1, we have calculated software time and hardware time seperately. Software time by counting number of cycle times of codes, and worst case of hardware time, maximum path delay of hardware ,as shown in Table 3, have been obtained.

**Table 3. Maximum FFT Delays in Hardware**

| FFT | TIME | |
|---|---|---|
| SIZE | Radix2 | Radix4 |
| 16-Pt | 8.9 us | 3.4 us |
| 32-Pt | 22.3 us | - |
| 64-Pt | 51.8 us | 20.2 us |
| 128-Pt | 125.0 us | - |
| 256-Pt | 285.7 us | 107.5 us |

As shown in Table 4, the performance of radix-2 FFT on CSoC is better than that of TMS, except for 128-point. In case of having sufficient internal RAM, 128-point FFT on CSoC becomes better, too.

**Table 4. FFT Performance for (a) Radix-2 (b) Radix-4 on E5 CSoC**

| Points | CSoC_Time1 | CSoC_Time2 | TMS_Time |
|---|---|---|---|
| 16 | 111.3 | 111.3 | 218.8 |
| 32 | 278.3 | 278.3 | 437.6 |
| 64 | 666.2 | 819.8 | 875.2 |
| 128 | 1558.6 | 2096.2 | 1750.3 |

**(a)**

| Points | CSoC_Time1 | CSoC_Time2 | TMS_Time |
|---|---|---|---|
| 16 | 54.6 | 54.6 | 77.2 |
| 64 | 327.4 | 404.2 | 424.6 |
| 256 | 1745.9 | 2462.7 | 2335.4 |

**(b)**

Similar to Radix-2 implementation, except for the 256-point FFT, CSoC is faster than TMS and with sufficient internal RAM, 256-point FFT on CSoC becomes better as well.

Software and communication times are a large percentage of total elapsed time. Therefore, they dominate the very short time of hardware execution. In fact, when the number of points increases, communication time increases and takes a larger percentage of total time.

## 6. Conclusions

FFT algorithms can be implemented efficiently on TMS320C25, which is a digital signal processor. In this paper, two FFT algorithms are implemented on a CSoC using co-design techniques. In general, three major factors which increase the FFT performance are:

- Choosing a proper algorithm (e.g., Radix-4 rather than slow Radix-2 and complex Split-Radix.)
- Assembly code optimization.
- Available resources on CSoC, specially the availability of the microprocessor.

Our experiments show that CSoC FFT, in small number of points, is faster than TMS FFT. This is due to the communication time, which is the bottleneck

## 7. References

[1] Balducci, M., Choudary, A.,Ganapathiraju, A., Hamaker, J., Picone, J., Skjellum, A. [1997], "Benchmarking of FFT Algorithms," *Proceedings of IEEE Southeastcon*, Blacksburg, VA.

[2] Triscend[2000], Triscend E5 CSoC Data Sheet, Triscend Corporation.

[3] Duhamel, P.[1986], "Implementation of Split-Radix FFT Algorithms for Complex, Real, and Real-Symmetric Data," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, pp. 285-95.

[4.] Lin, K. S.[1987], "Digital Signal Processing Applications with the TMS320 Family," Prentice Hall, Vol. 1.

[5] Scott.MacKenzie[1994], I. "The 8051 Microcontroller," Prentice Hall.

[6] Oppenheim, A. V., and, Schafer, R. W.[1999], "Discrete-Time Signal Processing," second edition, Prentice Hall, Englewood Cliffs, New Jersey.

[7] Papamichalis, P., So, J.[1989], "Implementation of Fast Fourier Transform Algorithms with the TMS32020," Application report: SPRA122, Texas Instruments.

[8] Proakis, J., and, Manolakis, D.[1996], "Digital Signal Processing: Principles, Algorithms

[9] TMS320C2x User's Guide[1993], Texas Instruments.