Proceedings of the International Conference on Industrial Engineering and Operations Management

Publisher: IEOM Society International, USA DOI: 10.46254/GC03.20250146
Published: February 2, 2026

DSP Optimization vs FPGA Performance: A Study of FIIR Filter Implementations

Farid Benaida
Computer Engineering Dept.
College of Engineering and Computer Sciences,
Mustaqbal University, Buraydah, Qassim, 51411, Saudi Arabia
fabenaida@uom.edu.sa
PHD Student, Computer Science Department
Badji Mokhtar University, Annaba, Algeria
benaidafarid@yahoo.fr

Habiba Beleili
Computer Science Department
Badji Mokhtar University, Annaba, Algeria
h.belleili@gmail.com

Nacer Eddine Hammami
AIMD Laboratory, Computing College
Fahad Bin Sultan University, Tabuk City, KSA
nhammami@fbsu.edu.sa

Radia Touahri
Computer Engineering Dept.
College of Engineering and Computer Sciences
Mustaqbal University, Buraydah, Qassim, 51411, Saudi Arabia
ratouahri@uom.edu.sa

Abstract

When we compare the execution of a C program on a general-purpose CPU versus an FPGA, as the FPGA's superior performance, especially in parallel processing, FPGA will win the contest. Instead, this article aims to demonstrate how an FPGA implementation can outperform a specialized digital signal processor (DSP) specifically designed for signal processing tasks. This article revolves around DSPs, their definition and their optimization tactics, and we have taken as an example the DSP C6000 from the giant Texas Instrument. We chose an FIR filtering algorithm to implement it under DSP. We have proposed an optimization strategy on this DSP to make the most of the reliability of the C6000 DSPs. Then the same algorithm will be carried out and materialized under FPGA in sequential and then in parallel. The results and discussions will come at the end of this article.

Keywords
Digital filter, fiir, fpga, dsp, Parallel

1. Introduction

The most famous processing applied to a signal is filtering. And more the signals that we are going to process are of an analog nature we need to make assemblies using analog components (resistor, capacitor, coil, etc.) (Frénéa M. 2013).

Analog components are always susceptible and sensitive to temperature, mechanical vibrations, variation in supply voltage and the tolerance of the values of each component.

Each time we are going to have a change in the characteristics of the filter to modify or adjust the transfer function of an analog filter, we are obliged to change the electronic card or some elements of this card because its components are less configurable and more than; not reconfigurable. To overcome the limitation of DSP we have to digitize the analog signal then apply digital filters.

2. Literature Review

The focus of research in this area is primarily focused on the benchmarking, optimization, and efficient hardware implementation of Finite Impulse Response (FIR) filters. This theme focuses on increased performance, reduced resource utilization, and lower power consumption for real-time digital signal processing (DSP) applications. Several studies directly compare the efficiency of implementing FIR filters on DSPs and FPGAs (Al-Sarayreh and Amleh 2015). While DSPs specialize in signal processing with optimized instruction sets and architectures (superscalar architectures), FPGAs are increasingly recognized for their superior parallelism capabilities. This study shows that highly parallelized FPGA implementations can often outperform highly optimized DSP implementations in terms of speed and energy efficiency, especially for computationally intensive tasks. A significant portion of research is dedicated to optimizing the implementation of FIR filters specifically on FPGAs. Raju and Ramana (2015) Techniques are being explored to achieve high sampling rates and throughput by exploiting the inherent parallelism of FPGAs. Various parallel processing methodologies are being studied to minimize hardware cost and maximize throughput. This often involves designing custom architectures that fully utilize FPGA resources.

Focusing on optimization techniques for DSP implementations, optimization strategies are crucial, including software pipelining, loop unrolling, and leveraging the Very Long Instruction Word (VLIW) architecture of some DSPs to maximize instruction-level parallelism. Kumar and Gupta (2014) On the other hand, for FPGAs, optimization revolves around hardware architecture design, parallel processing, and efficient resource allocation to achieve the desired performance metrics (http://www.ti.com/sc/docs/dsps/support.htm. 2021).

Regarding Analog signal digitization method which were invented to make flexible modifications or adjustments, (GARNIER 2016); these algorithms must be implemented on a processor dedicated to digital signal processing like DSPs or reconfiguration and programmable platforms such as CPLD and FPGA circuits (Figure 1)

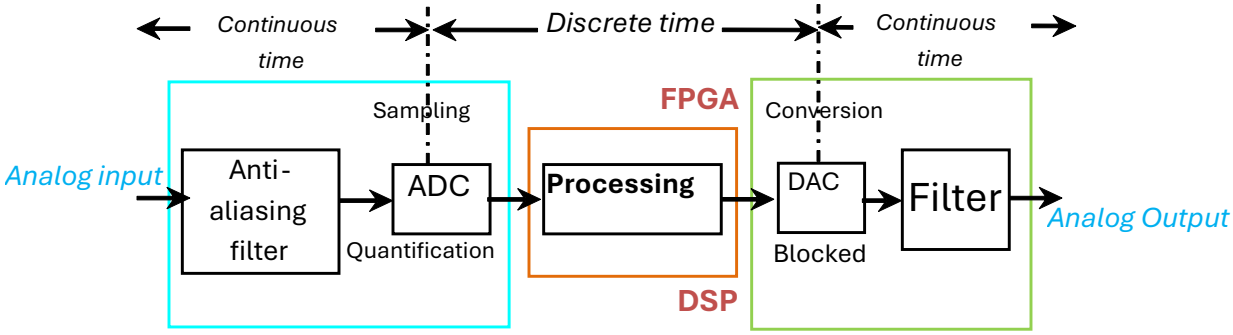


Figure 1. Digitization cycle and signal filtering

3. Methods

This article focuses on study of digital filters, their behavior, implementation in real time with DSP µP and FPGA to compare performances.

3.2 Position of the problem

We talk about signal processing when we talk about Speech processing, video processing or image processing. We need micro-processors that can run around 25 MOPS.

Table 1. processing capacity for one µP at medium power (25 MOPS)

	Frequency of sampling	Period between two samples	µP of 25 MOPS
speech	8 KHZ	125 µs	1250 operations
Audio	44 KHZ	22.7 µs	227 operations
Video	5 MHZ	200 ns	2 operations

(Table 1) gives the approximate number of operations achievable between two samples for a µP of 25 MOPS. As can be seen, the applications that can be implemented may be relatively more complex for signals in the speech band, while the processing of a video signal in real time seems out of reach for such µP (Ricardo 2016). This explains the major efforts made by manufacturers to market architectures capable of supporting multimedia applications.

3.3 Need for a specialized µP (DSP):

DSPs are able to work in real time, do not require an operating system to supervise, small in size, architecture adapted for digital signal processing, RISC coding and low power consumption (Texas Instrument 2017).
With the key to digital signal processing algorithms which is the sum of products (SOP) we can ask this questions:
Is a DSP equipped with a unit that calculates the sum of the products (MAC) more than enough to work comfortably in real time?
What are the modifications introduced on the architectural axis of µP to move towards real-time processing?

3.4 Filter operation

In the literature of digital filters Frénéa(2013). we find two classes: Classe of FIR filters (with finite impulse response) and Classe of IIR filters (infinite impulse response)
In what follows we will study only the class of FIR filters.

The FIR filter is a FEED FORWARD SYSTEM, its output is given by the following equation (digital convolution).

$$y(n) = \sum_{k=0}^{M=2} b(k)x(n-k)$$

$$y(0) = \sum_{k=0}^{2} b(k)x(0-k) = b(0)x(0) + b(1)x(-1) + b(2)x(-2)$$

Causal x(t)=0 if t<0

$$y(1) = \sum_{k=0}^{2} b(k)x(1-k) = b(0)x(1-0) + b(1)x(1-1) + b(2)x(1-2)$$

$$= b(0)x(0) = b(0)x(1) + b(1)x(0)$$

Either the filter FIR b = {-1, 0, 1}.

- Implementation (Figure 2)

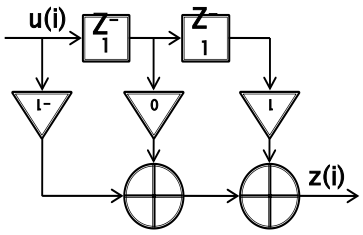


Figure 2. Implementation

We note that after each calculation of z(i) by the SOP formula, the memory elements in Z-1 form a buffer with a shift of the samples by x(i).
So the filter behavior is as follows:

- Loading of the 1st buffer element by x(i).
- Calculation of z(i) by the SOP.
- Buffer is shifting by one position.

So, really in C/C++ we will program the behavior of the filter

Code in C/C++

```
// load the 1st buffer element with x(i)
buffer = x[i];
// calculation of the output of the filter by the SOP
z(i) = 0;
for(k=0;k<3;k++) {
       z(i) += buffer[k]*b[k];
}
// shift the buffer by one position
for(k=2;k>0;k--) {
       buffer[k] = buffer[k-1];
}
```

3.5 Implementation of the algorithm under DSP C6x (C6000 CPU):

The DSP C6x (C6000 CPU) for effectively implement the behavior of the FIR filter, it is necessary to work with a specialized processor such as DSPs. In this article, we will use the DSPs of Texas Instrument (TI) class C6x. This class has the following general view in Figure 3:

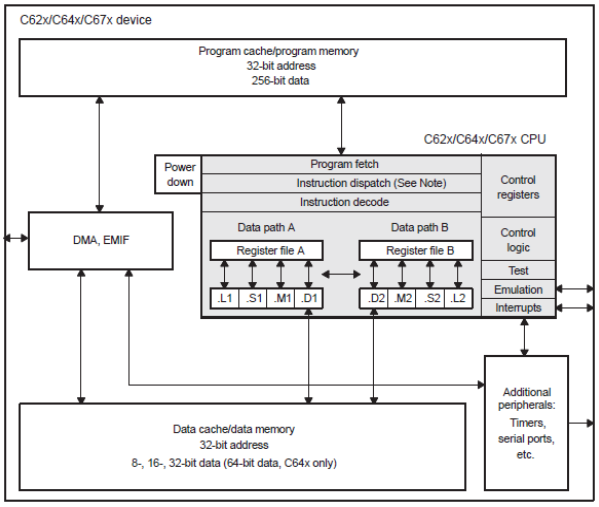


Figure 3. Architecture of DSPs RI C6X

Texas Instrument (2017)C6x DSP (C6000 CPU) has :

- A CPU of two REG FILE A & B banks, 8 units (4 per bank).
- 2 cache memories (DATA CACHE & CODE CACHE).
- A DMA (Direct Memory Access).
- An EMIF (External Memory Interface).
- A set of peripherals (Timers, Serial ports, etc.(Figure 4)

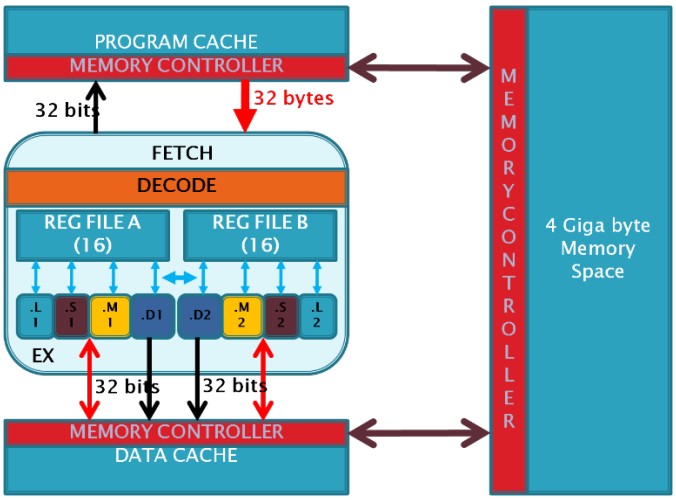


Figure 4. programmable unities of DSP C6X

The CPU (DSP) in Figure 4 and Figure 5. is a 32-bit extended to 40-bit processor based on : LOAD/STORE technology, (http://www.ti.com/sc/docs/dsps/support.htm. 2021) (RISC type instruction set), a addressable memory space of 4 Giga byte, and it uses a pipeline of 11 stages corresponding to the three phases FETCH (4 stages), DECODE (2 stages) and EXECUTE (5 stages). (Texas Instrument 2017)
The goal is not the study of DSP, but we will exploit it to implement our FIR filter in order to compare with a hardware solution on FPGA circuits.
In this section, we will present how to optimally implement the FIR filter on the Texas Instrument C6x DSP.

3.6 Implementation on the C6x DSP:
In this section, we will present how to optimally implement the FIR filter on the Texas Instrument C6x DSP

- First: make the C/C++ of the algorithm to implement.

```
N = 20; // nombre de coefs dans le filtre
S = 0;
for (i=0;i<N;i++) {
      S = S + TAB1[i]*TAB2[i];
}
```

The SOP formula is a simple Dot.Prod of two arrays as coded in C/C++

- Second: make a detailed organization diagram.

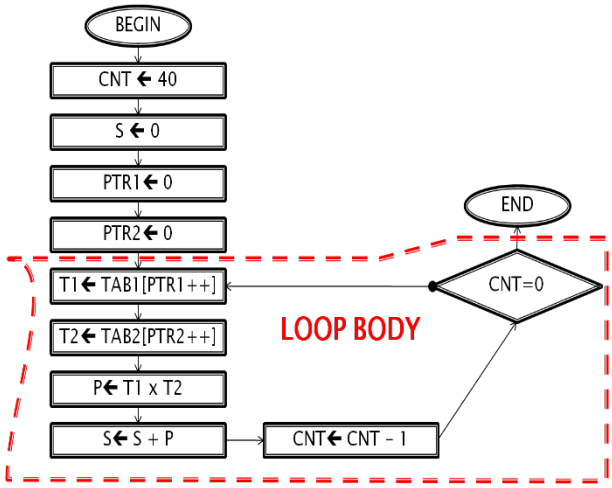


Figure 5. Diagram of FIR program

- Third step: translate the flowchart into linear assembler (LOOP BODY ONLY)

```
LOOP :      LDH         *PTR1++, T1
            LDH    *PTR2++, T2
            MPY         T1, T2, P
            ADD         P, S, S
            SUB         CNT, 1, CNT
[CNT] B           LOOP
```

- Fourth step: make the dependency graph as shown in Figure 6 (DEPENDENCY GRAPH,)

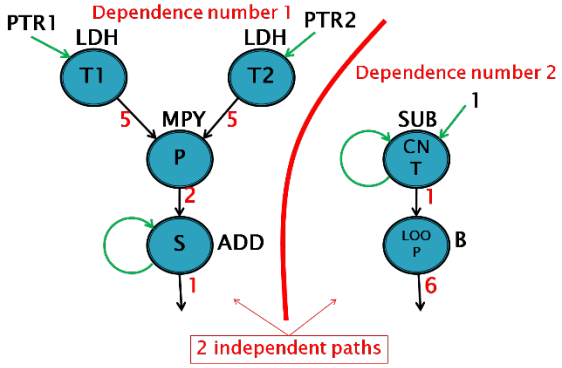


Figure 6. dependency graph of the instructions

- Fifth step: calculate the number of cycles per branch:

1st branch = 5 + 2 +1 = 8 cycles

2nd branch = 1 + 6 = 7 cycles

- Sixth step: select the longest branch and align the independent paths to end at the same cycle by the instruction, Figure 7 (SCHEDULING TABLE).

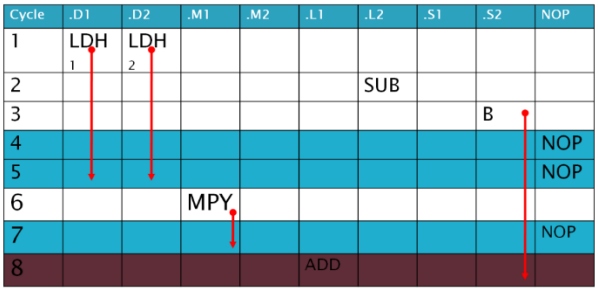


Figure 7. SCHEDULING TABLE

- Seventh step: allocation of registers on the light of the scheduling table.

```
PTR1, T1  → A4, A6 (.D1)
PTR2, T2 → B4, B6 (.D2)
P → A5 (.M1x)
S → A0 (.L1)
CNT → B0 (.L2)
LOOP → (.S2)
```

- Eighth step: translates the scheduling Table with register allocation to the real C6000 assembler (LOOP BODY ONLY)
-

```
LOOP:       LDH.D1      *A4++, A6
            ||    LDH.D2      *B4++, B6 (1)
                  SUB.L2      B0, 1, B0 (1)
[B0]        B.S2        LOOP      (1)
                  NOP         2           (2)
                  MPY.M1X     B6, A6, A5(1)
                  NOP         1           (1)
                  ADD.L1      A5, A0, A0(1)
```

- Ninth step: calculate the number of cycles to execute the code.

Number of cycles=(1+1+1+2+1+1+1)xN cycles
If N = 40 then 8x40 = 320 cycles. (N is the number of samples to process.)

There are other optimization methods on the DSP C6000 that give a fairly significant gain on the execution speed axis, such as the software pipelining method. Also, Loop Unrolling, long instruction Word(VLIW),

3.7 Implementation

To test our program written in C, we used the DSP Starter Kit (DSK) development board connected to a PC via JTAG and the Code Composer Studio (CCS) IDE. Linear assembly is the programming language best suited to the DSP.

3.8 Materialization of the program

Under FPGA (low density) → sequential approach
Under high density FPGA → Parallel approach

a) Sequential approach:

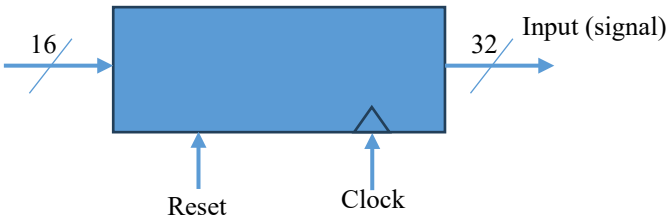


Figure 8. general view of FIR filter circuit

- Input x[i]
- Pointer (k)
- B constants → ROM
- Accumulator S

Input (signal)

The proposed circuit, Figure 8, to materialize the FIR filter in its global view is composed of:
- Input x[i] of size 16 bits containing the digitized value of the signal to be processed
- Clock: to synchronize circuit operations
- Reset to return to the initial circuit state
- The 32-bit size signal values after processing by the circuit.

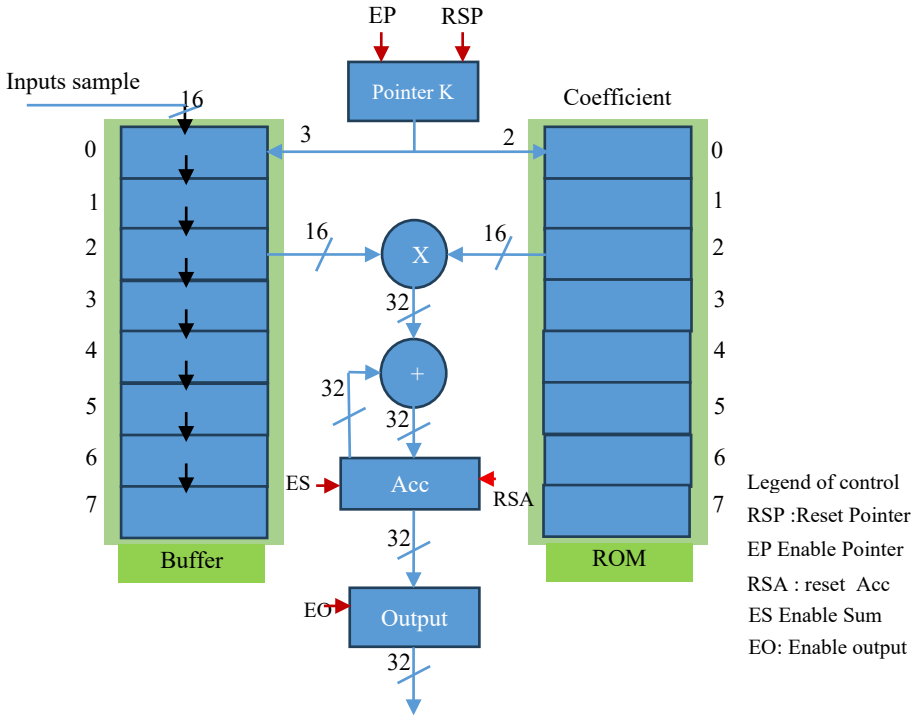
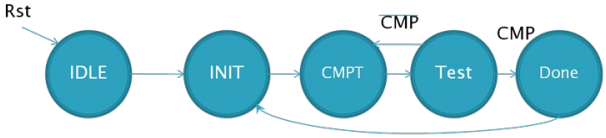


Figure 9. detailed FIR circuit

The proposed circuit is detailed in Figure 9 as follows:
Input x[i]: contains the elements resulting from the digitization of the signal with a size of 16 bits
• Buff: Buffer (BRAM) of size (16*8) to store the signal elements
• A ROM of size (16*8) containing the filter coefficients in this case they are constant
• A Pointer (k) which points both to the buffer and to the ROM
• An accumulator S to store the result of the production of each element by the coefficient and which automatically calculates (accumulates) the sum of the products when it is full (8 multiplications) (Es=1)
• The result is routed to the circuit output and the control signal RSA (Reset Accumulator) is reset to 0 to process other signal elements
Finite state machine (FSM):



B) Parallel approach:

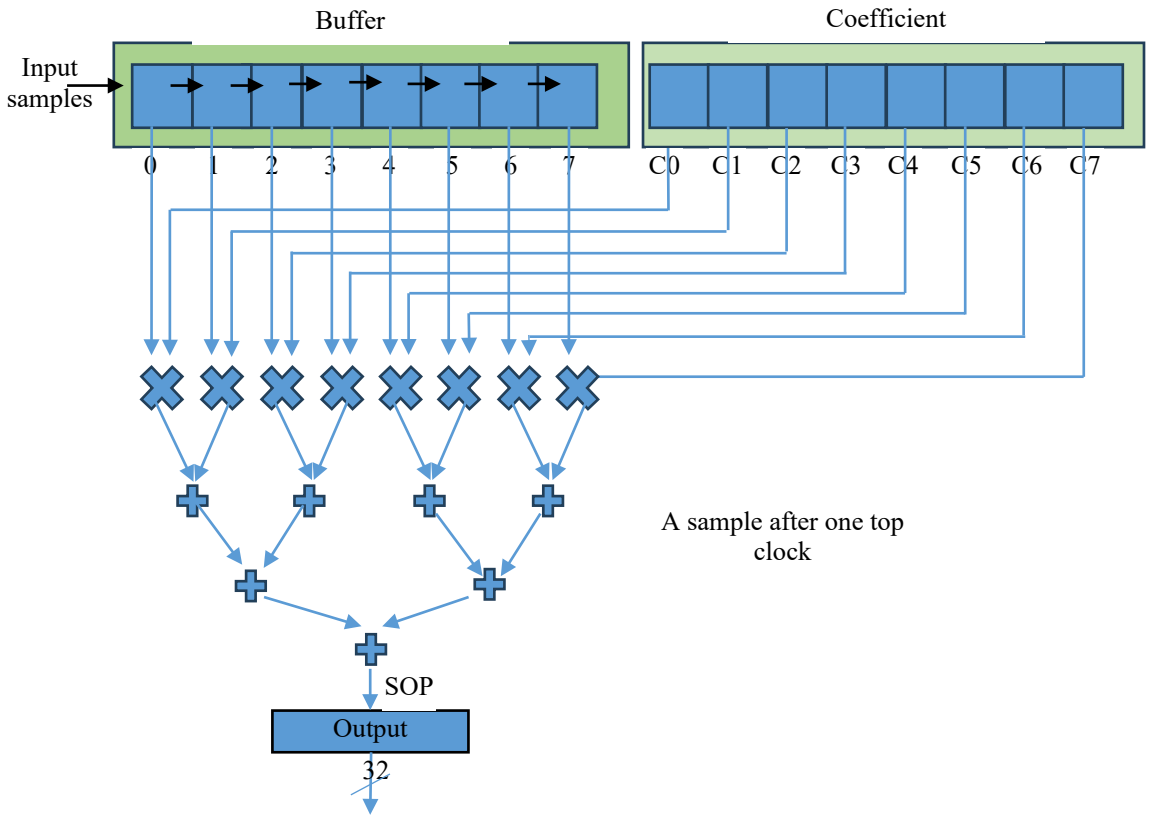


Figure 10. FIR circuit detailed in Parallel

According to Figure 10, the implementation of the FIR filter in parallel (in a case of 8 elements)
Shows a shift buffer where each element is pointed by a coefficient to do the multiplication the result and added "two by two" by an adder at the first level for the four multiplication results then the second level of two elements, the final result is routed to circuit output.

Noted, that at each clock top, an element is loaded into the buffer and when the buffer is full (full buffer=1) the calculation of the result is done by a single clock top.

3.9. Hardware implementation:

Hardware implementation: The plan was to implement the sequential solution using a simple, low-density FPGA board and the parallel solution with a higher-performance board. However, the unavailability of this board forced us to use the same board for both approaches.
The FPGA board used is the Xilinx Spartan 3AN Starter Kit, with IDE ISE version 14.5.

5. Results and discussion

We note from what has been presented in Table 2 to Table 4 that the execution of the FIR algorithm under DSP 6000 gives a satisfactory result, we note also, that execution on a DSP gives a satisfactory result, considering that DSPs are dedicated to this type of processing. After applying our optimization techniques, the execution time is reduced by half, the same gain obtained by sequential implementation under FPGA. The software pipelining method reduces this time to 47 cycles, otherwise, 13.6 times faster, but with a parallel implementation FPGA gives 16 cycles with resource consumption less than the consumption in sequential solution.

Table 2. Results

Implementation	Number of cycles	Gain (Real optimization)
Real code (sequential)	640	640/640 = 1
Optimized	320	640/320 = 2
Software pipelining	47	640/47 ≈ 13.6
FPGA sequential	320	640/320 = 2
FPGA parallel	40	640/40 = 16

Table 3. **D**evice Utilization Summary (estimated values) for sequential implementation FPGA

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	209	54576	0%
Number of Slice LUTs	103	27288	0%
Number of fully used LUT-FF pairs	39	273	14%
Number of bonded IOBs	50	218	22%
Number of BUFG/BUFGCTRLs	1	16	6%

Table 4. **D**evice Utilization Summary (estimated values) for parallel implementation FPGA

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	66	54576	0%
Number of Slice LUTs	35	27288	0%
Number of fully used LUT-FF pairs	18	83	21%
Number of bonded IOBs	50	218	22%
Number of BUFG/BUFGCTRLs	1	16	6%

6. Conclusion

We can conclude from this article that using DSPs with C compilers does not allow us to take full advantage of the superscalar architecture of these microprocessors; we need to descend to a lower level and program them in their own assembly languages to benefit from their specialized instruction set (like MPY). The application of software pipelining shows that DSPs remain unbeatable in their field of signal processing, but The parallelism offered by FPGAs ends the duality between DSPs and FPGAs with a gain of 16 times faster (40 cycles), noting that both platforms use the same clock frequency (20 MHz).

Under FPGA, although there is a great improvement in parallel design, we note that the consumption of the components in sequential seems high contrary to what was planned (sequential consumes less) this influences the choice of the FPGA type (low or high density).

7. Horizons and recommendations

We intend to improve this study by relying the most on optimization on the DSP side (other optimization techniques such as: Unrolling loop, Word wide, Software pipeline) and improve the design and implementation on FPGA solutions in parallel order to reduce the number of electronic components and promote its use on low density FPGAs, All of this is to minimize the execution time of filtering algorithms in general, and all other signal processing algorithms in particular, to allow for smooth real-time execution.

References

Instrument, T., SPRS762E –August 2011–*Revised January 2017. 2017.*

Frénéa M., Notes de cours (version2) Antenne de Bretagne, E.C.A.d.
Bretagne, Editor 2013.

Al-Sarayreh, M., & Al-Amleh, R. (2024, January).
A Comparative Study between FPGA and DSP for the Efficient Implementation of FIR Filters, 2024. *ResearchGate. https://www.researchgate.net/publication/387584166_*

Raju, M., Ramana, D. V., & Sarma, T. G. V. R. (2015, January). High Speed FPGA Implementation of FIR Filter for DSP Applications. *ResearchGate, 2015*.

Kumar, N., & Gupta, P. (2014, July). Comparative Analysis of Design Methodologies for Parallel FIR Filter. *International Journal of Computer Applications, 98(2, 2014).*

Liden, M. (2018, October). FIR Filter Features on FPGA. *DiVA portal.*

Padmavathi, V., & Bhavani, V. (2020, February). FPGA Implementation of Efficient FIR Filter. *International Journal of Engineering and Advanced Technology, 9(3, 2018).*

GARNIER, H., *Conception de filtres numériques*, 2016.

Ricardo, P., et al., Scalable Sensor Data Processor: Testing and Validation. 2016.

Seminar Report On The C6000 Family: Architecture, Pipelining and General Trends 2006 – 2007.