Custom Instruction Set NIOS-based OFDM Processor for FPGAs

Uwe Meyer-Bäse[a], Divya Sunkara[a], Encarnacion Castillo[a,b], and Antonio Garcia[b]

[a]FAMU-FSU, ECE Dept., 2525 Pottsdamer Street, Tallahassee, FL USA-32310;
[b]Dep. of E&C Technology University of Granada, Spain

ABSTRACT

Orthogonal Frequency division multiplexing (OFDM) spread spectrum technique, sometimes also called multi-carrier or discrete multi-tone modulation, are used in bandwidth-efficient communication systems in the presence of channel distortion. The benefits of OFDM are high spectral efficiency, resiliency to RF interference, and lower multi-path distortion. OFDM is the basis for the European digital audio broadcasting (DAB) standard, the global asymmetric digital subscriber line (ADSL) standard, in the IEEE 802.11 5.8 GHz band standard, and ongoing development in wireless local area networks.

The modulator and demodulator in an OFDM system can be implemented by use of a parallel bank of filters based on the discrete Fourier transform (DFT), in case the number of subchannels is large (e.g. $K > 25$), the OFDM system are efficiently implemented by use of the fast Fourier transform (FFT) to compute the DFT. We have developed a custom FPGA-based Altera NIOS system to increase the performance, programmability, and low power in mobil wireless systems.

The overall gain observed for a 1024-point FFT ranges depending on the multiplier used by the NIOS processor between a factor of 3 and 16. A careful optimization described in the appendix yield a performance gain of up to 77% when compared with our preliminary results.[12, 13]

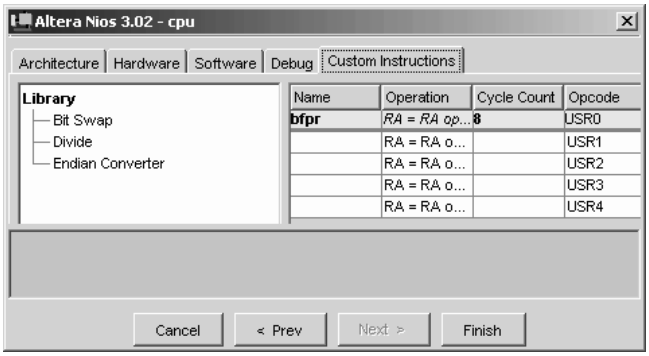
Keywords: OFDM, FFT, FPGA, NIOS

1. INTRODUCTION

Limitations in the performance obtainable with standard PDSPs has led to the development of specially designed chips, such as media processors.[1] However, these devices have often proved to be too inflexible in all but a narrow range of applications, and can suffer from performance bottlenecks. The limitations of a processor-based approach become especially apparent in high-resolution real-time communication processing systems. Fundamentally, a PDSP solution is restricted in how many cycles can be allocated to each tap of a filter, or each stage of a transform. Once the performance limits have been reached there is often no other way around the problem than to add extra PDSP parts. An FPGA, however, can be custom tailored to provide the maximum efficiency of utilization and performance. Its possible for a user to trade off area against speed, and invariably perform a given function at a much lower clock rate than a PDSP would require. There are a wide number of real-time multimedia processing functions that are well fitted for implementation in FPGA devices these include real-time functions such as FFTs used in OFDM systems. Many of these functions are both application specific and system-specific, and are based around core structures such as multiple MACs. Such functions can be implemented quickly using HDL language design or by exploiting the DSP building blocks found in high-level core design tools such as the Altera SOPC Builder software. Its also possible to reduce both design and simulation time by employing a system-level design approach, using products such as MathWorks MATLAB^{TM} and Simulink^{TM} software.

Altera embedded processor solutions include Excalibur devices that offer integrated processor subsystems and the soft-core (configurable) NIOS embedded processor for Altera Field Programmable Gate Arrays (FPGAs).

Further author information: (Send correspondence to U.M.-B.)
U.M.-B.: E-mail: umb@eng.fsu.edu, Telephone: (850) 410-6220
D.S.: E-mail: Divya.Sunkara@analog.com, Telephone: (850) 410-6438
E.C.: E-mail: encas@ditec.ugr.es, Telephone: (850) 410-6438
A.G.: E-mail: agarcia@ditec.ugr.es, Telephone: +34-958240482



(a)

Unit	LE	EAB
16 bit data path NIOS	950	2
32 bit datapath NIOS	1250	3
UART, fixed baud rate	170	
Timer	244	
serial peripheral interface(SPI):		
8-bit master, 1 slave	103	
SPI: 8-bit master, 2 slaves	108	
General-purpose I/O: 32-bit, tri-state	138	
SDRAM controller	380	
External memory/peripheral: 32-bit	110	
External memory/peripheral: 16-bit	85	

(b)

Figure 1. (a) Custom instructions feature of the NIOS processor. (b) NIOS core and peripherals sizes logic elements (LE) count and embedded array blocks (EAB).

The Alteras NIOS development system has been very successful in recent years: 10,000 systems[2] have been sold to more than 3000 different customers since the year 2000. These devices include robust software support that provides easy-to-use design environment. One reason for the success is the fact that the NIOS development system provides VHDL and Verilog code to synthesize a full functional processor with all required peripherals at no additional charge (as long as you use Altera FPGAs). The second even more important reason is the fact, according to Altera,[2] that a full set of development tools like C compiler, assembler, linker, and profiler is provided. Most of the free public domain processors (see `www.opencores.org` or `www.fpgacpu.org`), have an assembler, but most do not have an instruction set simulator, or support for a C compiler, and are therefore of limited value.

2. FPGA NIOS SYSTEM DESIGN FLOW

To better understand all necessary steps in the current state-of-the-art design flow of FPGA microprocessor systems, we will discuss next the necessary steps regarding the design of an NIOS processor augmented by a FFT co-processor butterfly. It is in general agreed that programmable system design might be the next development challenge we face. With the advent of the Xilinx and Altera platform FPGA and the NIOS and MicroBlaze soft processor, we now have access to unrivalled levels of product flexibility and performance. This technology for the first time enables us to partition and re-partition systems between hardware and software at any time during the development cycle. We can even repartition the hardware and software mix after the product has gone to market. This complete re-programmability means the system can be optimized over and over again. Altera embedded processor solutions include Excalibur devices that offer integrated processor subsystems and the soft-core (configurable) NIOS embedded processor for Altera FPGAs. Xilinx offers a state-of-the-art microprocessor PowerPC (embedded as hard-core in Virtex 2 PRO devices), and Xilinx MicroBlaze softcores. These devices include robust GNU GCC software support that provides easy-to-use design environment.

The Altera NIOS differs from other soft-core processor solutions in the market by including custom instruction feature, see Fig. 1(a). Custom instruction design is a process of implementing a complex sequence of standard instructions in hardware in order to reduce them to a single instruction macro that could be accessed by software. The custom instructions can be used to implement complex processing tasks in single-cycle (combinatorial) and multi-cycle (sequential) operations. In addition, these user-added custom instructions can access memory as well as logic outside the NIOS system. As a example design, we choose a radix-2 FFT for custom implementation due to its wide range of possible transform lengths (all power of two transform lengths) DFT calculations, decreased memory requirement, and easy hardware implementation of the small butterfly processor. The butterfly processor is implemented as custom logic block and its software macro generated is then used in the software code for the radix-2 FFT. The performance of the software code with custom instruction for the butterfly processor with different multiplier optimizations available with the NIOS processor is then compared with software-only code.

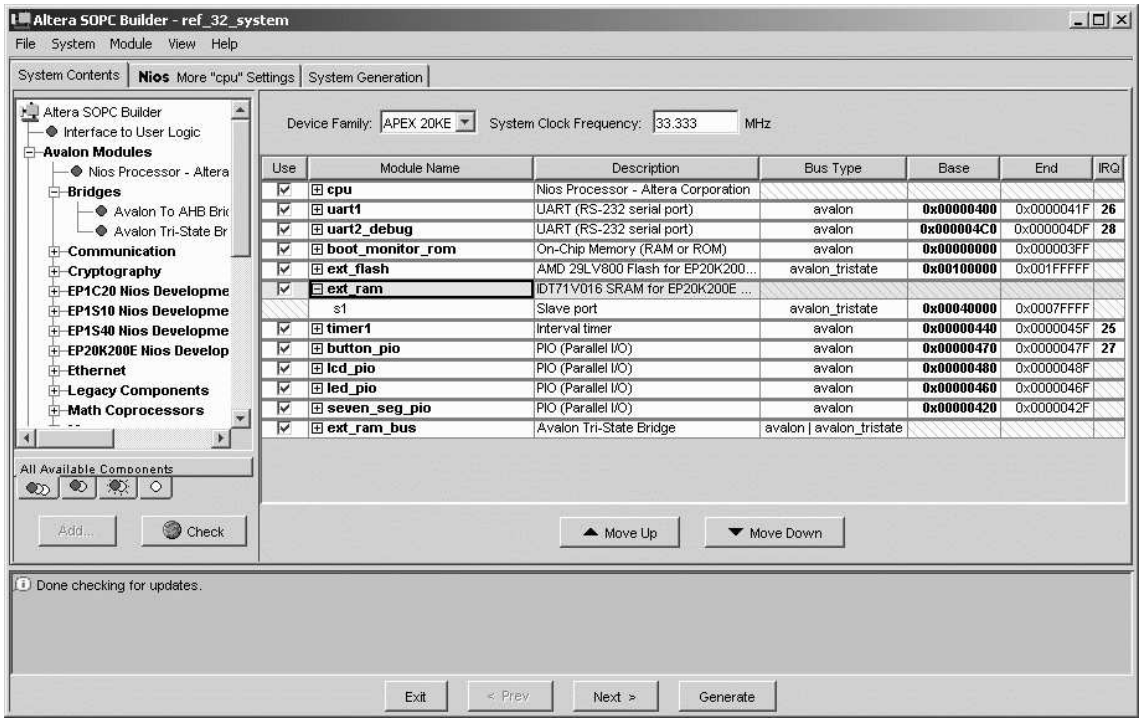


Figure 2. SOPC NIOS 32 bit standard processor template.

The appendix A reports on the detail improvements implemented, both on software as well on the HDL hardware side.

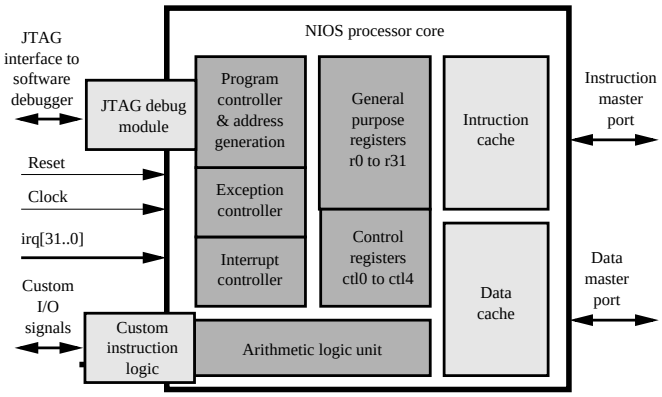
3. OVERVIEW OF NIOS PROCESSOR

The NIOS embedded processor is a configurable 16 or 32-bit RISC processor. NIOS embedded systems can be created with any number of peripherals. Figure 2 shows the SOPC builder 32 bit standard configuration of the NIOS processor.

Fig. 1(b) shows the base core sizes for the NIOS embedded processor and some of the IP core peripherals that integrate with the standard NIOS embedded processor to form complete micro processing units. Most peripherals can be parameterized to fit the specific application and can be instantiated multiple times within a single μP. In addition, customer-designed logic and peripherals can be integrated with the NIOS processor to deliver a unique μP. The creation of these custom μPs can be done in minutes using the Altera SOPC Builder tool, and synthesized to run on any Altera FPGA. In addition to the IP cores listed in Figure 2, SOPC Builder features additional IP cores available from Altera and Alteras Megafunction Partners Program (AMPP) partners.

With the wide range of densities available in FPGA devices and the small sizes of NIOS embedded systems, system designers can divide complex problems into smaller tasks and use multiple NIOS embedded processors. These NIOS processors can be customized with a wide-selection of peripherals, defining very simple to very complex μP systems. By targeting low cost devices, powerful, customized embedded systems can be realized at the best cost in the industry.

The NIOS processor shown in the Fig. 3(a) has a pipelined general-purpose RISC architecture.[3–8] The 32-bit processor has separate 16-bit instruction bus and 32-bit data bus. The register file is configurable to have 128, 256 or 512 registers but at a time only 32 of these registers are accessible as general purpose registers through software using a sliding window. These large numbers of internal registers are used to accelerate subroutine calls and local variable access. The CPU template is in general configurable with instruction and data cache memory



(a)

Multiplication option	Clock cycles 32 bit product	Hardware effort
Software	80	0
MSTEP	18	14-24 LCs
MUL	3	427-462 LCs

(b)

Figure 3. (a) NIOS processor core. (b) Multiplier optimizations.

that increases its performance, but the APEX CPLD device with the limited EAB memory block architecture, used by Altera in the NIOS development board, does not support this feature. The NIOS instruction set can be configured to increase the software performance and can be modified either by adding custom instructions or by using predefined instruction set extensions provided with the processor template. The predefined 3 multiplier optimizations for the NIOS processor are listed in Fig. 3(b) giving the number of clock cycles and size required for each of the multiplier options:

- MUL instruction includes a hardware 16x16 bit integer multiplier.
- MSTEP instruction provides hardware to execute one step of a 16x16 bit multiply in one clock cycle.
- Software multiplication uses the C-runtime libraries to implement integer multiplication with sequences of shift and add instructions.

Any of the above three multiplier options could be used to implement 16x16 bit multiplication in software. Depending on the overall processor architecture the additional hardware effort may vary.

4. FFT BACKGROUND

For the design, the DIF radix-2 FFT[9, 10] is implemented using the custom instruction for the butterfly processor implementation. The discrete Fourier transform for an N-point input signal $x[n]$ is given by

$$X[k] = \sum_{n=0}^{N-1} x[n] \mathrm{e}^{-\mathrm{j}2\pi kn/N} = \sum_{n=0}^{N-1} x[n] W_N^{kn}. \quad (1)$$

For the DIF radix-2 FFT algorithm, decimation is used to decompose the N-point DFT into successively smaller DFT. The decimation process for an N-point input sequence is carried out $\log_2(N)$ times. Each decimation step rearranges the input sequence into even and odd indexed sequences. The total number of complex multiplications is therefore reduced to $(N/2)\log_2(N)$. Figure 4(a) illustrates the signal flow graph of eight-point decimation-in-frequency radix-2 FFT algorithm showing the different stages, groups, and butterflies. The basic computation performed at each stage is called a butterfly shown in Figure 4(b). In this algorithm, the input is in normal order while the DFT output is in bit reverse order.

For evaluating the custom instruction feature of the NIOS processor, the DIF radix-2 FFT algorithm described above is implemented in two steps.

Table 1. Characteristics of DIF radix-2 algorithm

Stage Number k	Stage 0	Stage 1	Stage 2	...	Stage $\log_2(N)-1$
Number of groups per stage ($p=2^k$)	1	2	4	...	$N/2$
Butterflies per group ($t=N/2p$)	$N/2$	$N/4$	$N/8$	...	1
Increment exponent Twiddle-factors	1	2	4	...	$N/2$

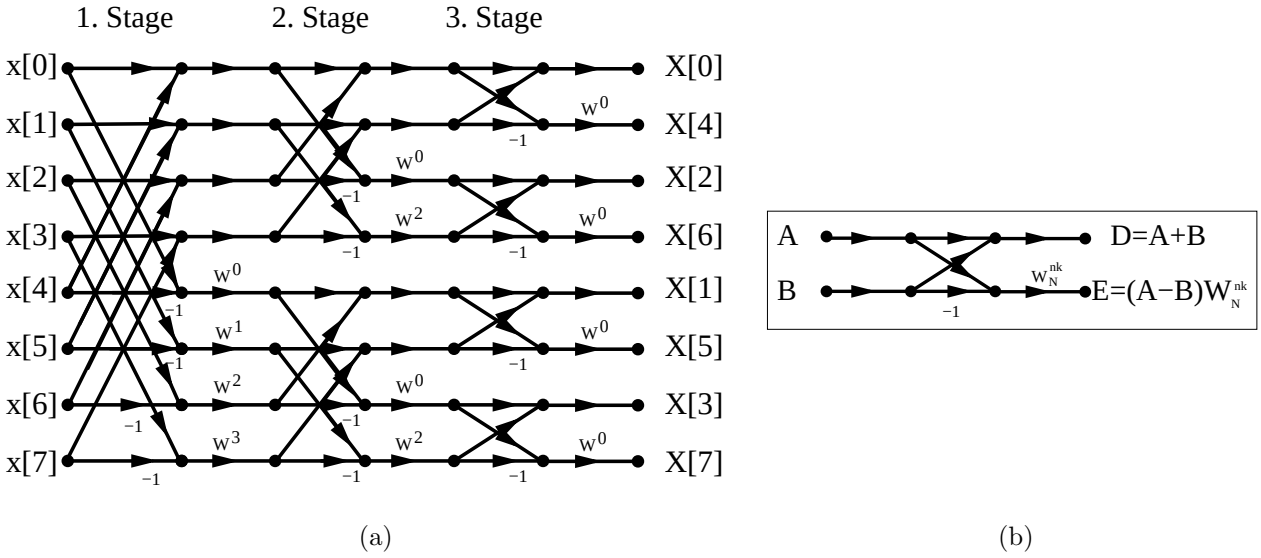


Figure 4. DIF FFT. (a) Signal flow graph length-8. (b) Butterfly computation.

- Software implementation
- Software implementation with custom instruction for butterfly processor

4.1. Software Implementation

Software implementation uses the characteristics of the algorithm illustrated in Table 1. As the Gnupro compiler provided with the NIOS development board, APEX edition, supports C programs, the software code for the algorithm is written in C language.[5–7] The first row gives the stage number for a length-N FFT and is designed as the outer-most loop. The second row gives the number of groups based on the stage number and is considered the second loop in the algorithm implementation. The third row gives the number of butterflies per group based on the group number and forms the inner most loop. The final row shows how the twiddle factors increment based on a particular stage. The hardware effort for the NIOS processor depending on the multiplier implemented without a custom logic block ranges from 2701 to 3163 logic cells as shown in the second column of Table 2.

4.2. Creation of Custom Logic Block

For the custom implementation of a butterfly processor, the custom instruction features of NIOS processor must be considered.[8] The custom logic block in the NIOS processor connects directly to the ALU of the

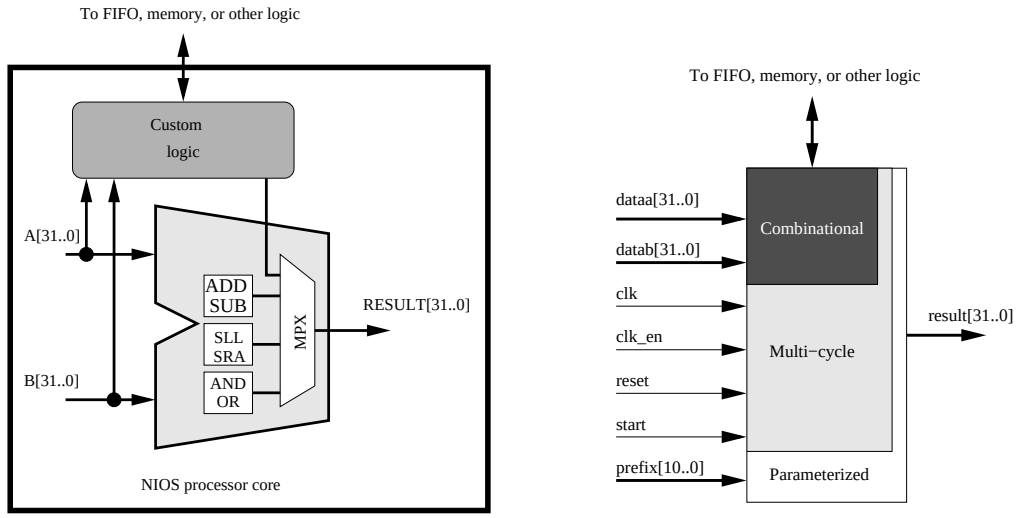
Table 2. Hardware effort of NIOS processor with different multiplier options and butterfly custom logic.

Multiplier option	Standard processor	Custom processor
Software	2701 LCs	4414 LCs
MSTEP	2716 LCs	4433 LCs
MUL	3163 LCs	4841 LCs

NIOS processor as a reconfigurable functional unit and therefore provides an interface with predefined ports and names present in the processor. The predefined physical ports and names that could be used for custom logic design are shown in Figure 5(a). The number of custom logic blocks that could be implemented in the NIOS embedded processor system is restricted to five blocks but with the presence of an 11-bit prefix port up to 2048 functions for each block can be performed. In the butterfly processor custom instruction design, the custom logic block for the butterfly processor is written in VHDL.[11] Meyer-Baese[12] has an efficient butterfly processor code (bfproc.vhd) for 8-bit data values. In this code, the processor is implemented with one adder, one subtraction and a component instantiation for twiddle-factor multiplier. The twiddle factor multiplication is efficiently computed using component instantiations of three `lpm_mult` and three `lpm_add_sub` modules. The output of the twiddle factor multiplier is scaled such that it has the same data format as the input. The algorithm used for the twiddle-factor multiplier uses three coefficients of the twiddle factor, C, C+S, and C-S where C and S are the real and imaginary coefficients of the twiddle factor. The complex twiddle factor multiplication R+j*I = (X+j*Y)(C+j*S) is efficiently performed when the above coefficients are used. The real and imaginary parts of complex twiddle factor multiplication using the above three coefficients can be computed as $R = (C - S) * Y + C * (X - Y)$ and $I = (C + S) * X - C * (X - Y)$ respectively. To ensure short latency for in-place FFT computation, the complex multiplier is implemented without pipeline stages. The butterfly processor is designed to compute scaled outputs where the output produced by the design is equal to half the actual output value. The butterfly processor design uses flip-flops for the input, coefficient and the output data to have single input/output registered design. For custom implementation, this design is modified to use the predefined physical ports for multi-cycle logic shown in Figure 5(b). The parameterized prefix port is used in the design to define various read and write functions required for reading and writing the complex input and output data present in butterfly computation. A total of eight prefix-defined functions are required for proper implementation of the design. The design is compiled with EPF20k200EFC484-2X (APEX device) present in NIOS development board using Quartus II software. It requires ca. 1700 logic cells of the APEX device. The performance (fmax) of the design is found to be 30.82 MHz. The simulation of the design shows that it requires two clock cycles for valid implementation of the design.

4.3. Instantiation of Custom Logic Block

The custom logic block is instantiated with the aid of the NIOS CPU configuration Wizard, which is a part of SOPC Builder (system integration tool) present in the Quartus II software available with the NIOS development kit. The only input for this instantiation process is the number of clock cycles required for valid implementation of the custom logic block created. For the butterfly processor custom logic block, the number of clock cycles required is eight, based on the simulation result. The instantiation process includes the custom logic block with NIOS ALU and generates a software macro for this block known as custom instruction. The new NIOS processor with the custom logic block is then recompiled and downloaded to the APEX FPLD device present in the NIOS development board. The entire design requires depending on the multiplier used between 4414 and 4841 logic cells of the APEX device, see third column in Table 2. The downloaded design acts as a platform for software implementation using custom instruction. The butterfly computation in software could then be performed using the software macro functions where the function of the software macro is defined by the prefix port values. Finally the software implementation of the DIF radix-2 FFT is modified by using custom instructions for the butterfly computation.



(a) (b)

Figure 5. (a) Adding custom logic to the NIOS ALU. (b) Physical ports for custom logic block.

Table 3. Speed increase for single Butterfly.

Multiplier option	Clock cycles with software	Clock cycles with custom instruction	Improvement factor
Software	1227	119	10.3
MSTEP	698	119	5.8
MUL	295	119	2.47

5. RESULTS

By measuring the number of clock cycles required for custom implementation and software-only implementation of the butterfly processor, the speed relation between the two is measured with different multiplier optimizations as shown in Table 3. The clock cycle measurements are taken based on Gnupro C/C++ compiler available with the NIOS development board. The increase in speed due to custom implementation for a single butterfly computation is given by

$$\text{Speed increase} = \frac{\text{Clock cycles } \textit{software} \text{ only design}}{\text{Clock cycles for custom design}} \tag{2}$$

Furthermore, the performance of different length FFTs using custom implementation and software-only implementation of the butterfly processor are compared. Figure 6 and 7 shows this comparison along with reference DFT data using the direct computation as in equation (1).

It is observed that the overall performance decreases with an increase in the length of FFT for each implementation while the performance of custom implementation for each of the FFTs computed is higher than software-only implementation. Of all the multiplier optimizations the MUL optimization gives better overall performance for both implementations. Apart from the performance calculations, it is observed that custom implementation results in small quantization error in the output values that increases with an increase in the length of FFT.

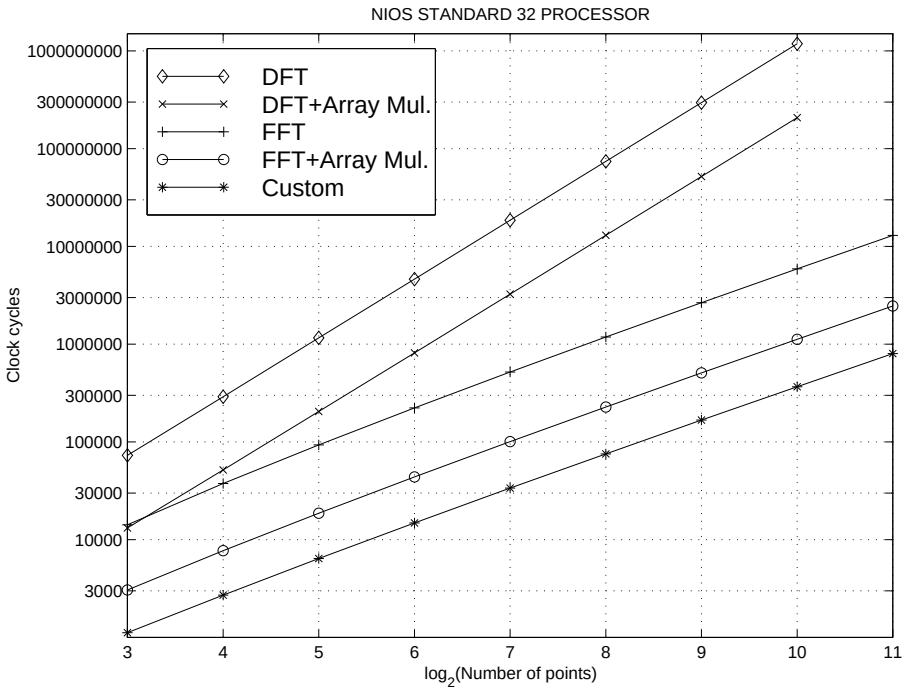


Figure 6. Number of clock cycles with and without array multiplier.

6. SUMMARY

With its configurable hardware and software features, the NIOS processor provides a powerful, reliable platform for developing and implementing complex algorithms. In addition, features such as low cost, smaller FPGA footprint, higher performance and robust software development tools make NIOS an attractive choice for software development. By using the custom instruction feature of NIOS processor for custom implementation of the butterfly processor in DIF radix-2 FFT, the performance of the algorithm is increased depending on the multiplier used by a factor between 3 and 16 for a 1024-point FFT. Additionally, the software code required for implementing the butterfly processor as custom instruction was comparatively smaller than that required for software-only implementation.

Still a careful analysis of the C-code is required. For instance the automatic conversion to floating point arithmetic for a short expression (e.g., $S = \log(N)/\log(2)$ to compute the number of FFT stages) requires depending on the available multiplier option between 25K-150K clock cycles that may exceed the time required to compute the FFT. A short table that matches FFT length and number of stage will save here many clock cycles.

The C and VHDL code of this paper[13] is available online via FSU library at `www.lib.fsu.edu`.

ACKNOWLEDGMENTS

The authors would like to thank Altera for their support under the University programs. U. Meyer-Baese acknowledge the support of the Humboldt Foundation. A. Garca and E. Castillo were supported by the Ministerio de Ciencia y Tecnologa (MCyT, Spain) under project TIC2002-02227.

APPENDIX A. CODE OPTIMIZATION

A first direct approach to code the DIF FFT is to use a standard program like the FORTRAN code by Burrus,[9] convert the code the language C and introduce the custom instructions. Such a code[13] will look like follows:

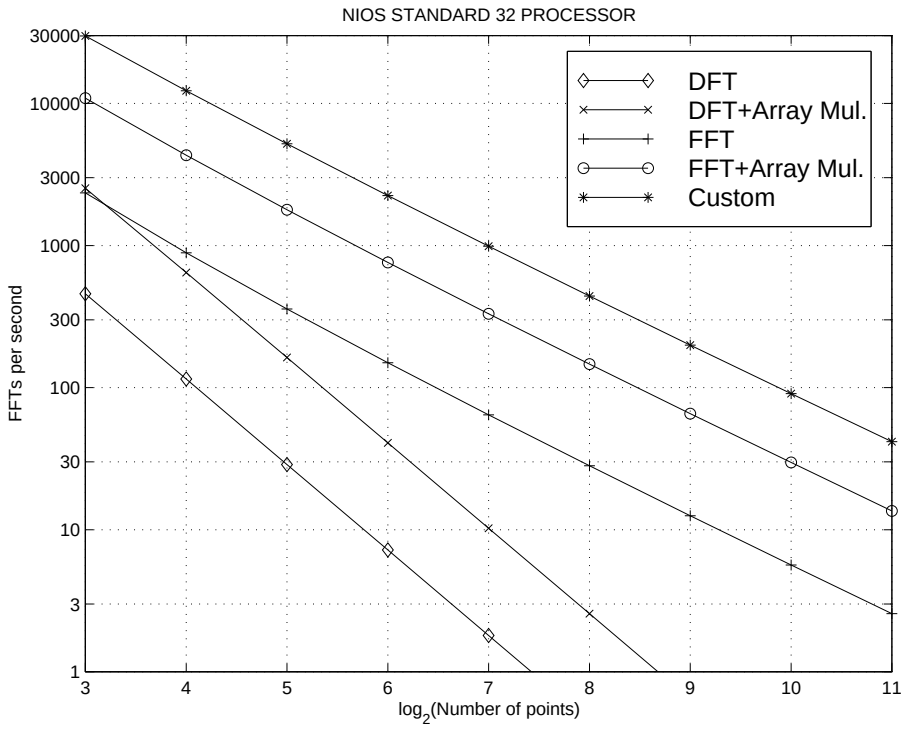


Figure 7. Number of DFT/FFT per second with and without array multiplier.

```
1: dwStartTick=GetTickCount();                          // Record Start Time
2: S = log10(N)/log10(2);                               // Number of stages
3: Stages for (k=0; k<S; k++)
4: Loop
5:  { p = 1<<k;                                         // Number of groups
6:    t = N/(p << 1);                                   // Number of butterflies
7:    f = t << 1;                                       // in each Group
8:    for (j=0; j<p; j++)                               // Group Loop
9:    {  I = f * j;                                     // Jump to each group in a stage
10:      for (i=0; i<t; i++)                            // Butterfly Loop
11:      {   a = I + t;                                 // Butterfly calculation
12:          Are = (*(xr+I) << 1); Aim = (*(xi+I) << 1);
13:          /***Custom Instructions**/
14:          nm_bfpr_pfx(1,Are,Aim);                     // Read Are and Aim
15:          Bre = (*(xr+a) << 1); Bim = (*(xi+a) << 1);
16:          nm_bfpr_pfx(2,Bre,Bim);                     // Read Bre and Bim
17:          Wr = *(wnr+(p*i)); Wi = *(wni+(p*i));
18:          nm_bfpr_pfx(3,Wr,0);                        // Read C
19:          W1 = Wr + Wi; W2 = Wr - Wi;
20:          nm_bfpr_pfx(4,W1,W2);                       // Read C+S and C-S
21:          *(xr+I) = nm_bfpr_pfx(5,0,0);               // Write the real value
22:          *(xi+I) = nm_bfpr_pfx(6,0,0);               // Write the imaginary value
23:          *(xr+a) = nm_bfpr_pfx(7,0,0);               // Write the real value
24:          *(xi+a) = nm_bfpr_pfx(8,0,0);               // Write the imaginary value
```

Table 4. Speed increase for 256-point FFT with software only improvements for different multiplier options.

Multiplier option	Clock cycles software mul.	Clock cycles MSTEP mul.	Clock cycles array mul.
Original 0	1345627	771295	331836
Improved 1	1236345	695673	281765
Improved 2	1228978	688306	268772
Improved 3	518953	643423	227663
Gain	61%	17%	31%

```
25:           I = I + 1;                                  // Jump to each butterfly
26:   }    }   }                                         // in a Group
27: lTicksUsed=GetTickCount();                           // Record end time
```

Several different DIF Radix-2 FFT version were tested and the following list shows the most successful changes:

0) Initial version using direct C-conversion from FORTRAN,[13] but with $\log_{10}()$ computation for number of stage outside the FFT routine, i.e. LUT. Now the FFTs with custom instruction should not depend on the NIOS multiplier type.

1) Improved software code: no multiplication, or divides in the index computations and by moving code to avoid multiply or divides. (see for instance listing line 5)

2) COS/SIN load outside the butterfly, i.e., computes all groups with same twiddle factor, rather then running through a whole group.

3) Uses a complex data type, i.e. real and imaginary part are stored in memory next to each other. This improves the memory access, which was verified analyzing the generated assembler code.

This different methods can improve the number of clock cycles depending on the multiplier type between 17% and 61% for 256-point FFTs, as shown in Table 4.

On the VHDL hardware side additional improvements were implemented:

0) Original Version from the Springer book DSP with FPGAs[12] adjusted in MS project[13]:
   - change I/O ports (dataa, datab result, etc.) as required by NIOS
   - change to 32 bit width I/O and intern 17x17 bit multiplier

1) Remove of 2. pipeline from CCMUL
   - CMUL included in the bfp2.vhd code
   - No lpm only `STD_LOGIC` functions for add and multiply
   - Move COS+/-SIN to FPGA side from software (remove one custom instruction)
   - No scaling by 2 in the hardware → remove `(<<1)` in software

2)
   - simplify result sign extension
   - Reset is asynchron
   - Start is used as enable for flip-flops

Table 5. Speed increase for 256-point FFT with software and hardware improvements.

	Clock cycles array mul.	Gain
Original Software	331836	
Custom Instructions	181747	45%
Software only	135615	59%
Hardware only	113054	65%
Software+Hardware	75516	77%

Overall gain using the custom instruction and further optimization is in the range of 45% to 65%. Together with a careful software development a gain of 77% for a 256-point FFT is observed, see Table 5. The following code shows the optimized C code. Both hardware as well as the software modification are present in the code.

```
1: dwStartTick=GetTickCount();
2: k2 = N; dw = 1;
3: for (l = 1; l <= S; l++)
4: {k1 = k2; k2 >>= 1; w = 0;
6:  for (k = 0; k < k2; k++) {
7:   Wr = coef[w].r;
8:   Wi = coef[w].i;
9:   nm_bfp3_pfx(3,Wr,Wi);                // Read COS+SIN
10:  w += dw;
11:  for (i1 = k; i1 < N; i1 += k1) {
12:   i2 = i1 + k2;
13:   /***Custom Instructions**/
14:   tr = x[i1].r; ti = x[i1].i;
15:   nm_bfp3_pfx(1,tr,ti);          // Read Are and Aim
16:   tr = x[i2].r; ti = x[i2].i;
17:   nm_bfp3_pfx(2,tr,ti);          // Read Bre and Bim
18:   x[i1].r = nm_bfp3_pfx(4,0,0);  // Write the real value
19:   x[i1].i = nm_bfp3_pfx(5,0,0);  // Write the imaginary value
20:   x[i2].r = nm_bfp3_pfx(6,0,0);  // Write the real value
21:   x[i2].i = nm_bfp3_pfx(7,0,0);  // Write the imaginary value
22:   }
23: }
24: dw <<= 1;
25: }
```

REFERENCES

1. F. Barat and R. Lauwereins, "Reconfigurable instruction set processors from a hardware/software perspective," *IEEE Transactions on Software Engineering* , pp. 847–862, 9 2002.
2. A. Corporation, "Netseminar nios processor." http://www.altera.com, 2004.
3. Altera Corporation, "Nios development kit, apex edition." Getting Started User Guide, 2004.
4. Altera Corporation, "Nios development board document." 2004.
5. Altera Corporation, "Nios software development tutorial.." 2004.
6. Altera Corporation, "Nios software development reference manual." 2004.
7. Altera Corporation, "Nios-32 bit programmer's reference manual." 2004.
8. Altera Corporation, "Custom instruction tutorial." 2004.

9. C. Burrus and T. Parks, *DFT/FFT and Convolution Algorithms*, John Wiley & Sons, New York, 1985.
10. S. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, California Technical Publishing, CA, available online.
11. J. Bhasker, *A VHDL Synthesis Primer*, Star Galaxy Publishing, Allentown, PA, 1998.
12. U. Meyer-Bäse, *Digital Signal Processing with Field-Programmable Gate Arrays*, Springer Press, Heidelberg, New York, 2004. 527 pages.
13. D. Sunkara, "Design of custom instruction set for fft using fpga-based nios processors," Master's thesis, FSU, 2004.