POZNAN UNIVERSITY OF TECHNOLOGY ACADEMIC JOURNALS
No 104 Electrical Engineering 2020

DOI 10.21008/j.1897-0737.2020.104.0002

Robert SMYK*, Maciej CZYŻAK**

ON IMPLEMENTATION OF FFT PROCESSOR IN XILINX FPGA USING HIGH-LEVEL SYNTHESIS

The paper presents results of the high level synthesis of an 1024-point radix-2 FFT processors in Xilinx Vivado FPGA environment. The use of various directives controlling the synthesis process is examined. The results indicate that using the proper set of directives the latency of the processor can be reduced by 95% from about 35k for the default parameters to 1.5k cycles after optimizations.

KEYWORDS: Fast Fourier Transform Processor, FPGA, High-level synthesis.

1. INTRODUCTION

Discrete Fourier Transform (DFT) [1-5] is one of the most important algorithms in digital signal processing (DSP). The DFT transforms an input sequence of sampled data and produces its frequency content. The inverse transform IDFT perfoms the reverse operation. The DFT is needed in various areas of science and technology. Doppler processing and matched filtering in radars [6] belong to the most demanding algorithms with respect to the short calculation time and accuracy. Other applications can be found in the physical layer of orthogonal frequency-division multiplexing systems(OFDM) [7], where the frequency content has to be converted to the time domain. Other exemplary applications encompass spectral analysis[3, 5], signal synthesis and radiotelescopes, where the spectrum of radio frequency signals from celestial objects is analyzed [8], and tomography image reconstruction [9]. The DFT calculation using its definition is generally not possible in the real-time because of the computational complexity of the DFT equal to $O(N^2)$, where N is the transform length. The direct calculation of the DFT calls for N^2 complex multiplications and N^2 complex additions and can be computed in real-time only for small N, eg. for $N \leq 32$. The breakthrough with respect to computational complexity came in 1965 due to the invention of the Fast Fourier Transform (FFT) by Cooley and Tukey [10], where the computational complexity was reduced to $O(N \log N)$ for $N = 2^n$. This means that for $N = 1024$ instead of 1048576 complex multiplications for the DFT, the FFT requires only 5120. Nevertheless the only way to cope with the real-time calculation requirement is the

* Państwowa Wyższa Szkoła Zawodowa w Elblągu

use of specialized FFT processors. Their parameters have been limited by the available IC technology. The FFT can be calculated using various FFT processors in dependence upon the requirements with respect to speed and accuracy. Probably the highest requirements for both factors exist in radar systems. The urgent need for the fast FFT calculation in radars led already in the second half of sixties of the XXth century to the implementation of specialized hardware FFT processors. The hardware multiplier was the crucial component and the main stumbling block in these processors. Such multiplier was relatively costly because, for example, the parallel matrix 16x16-bit multiplier can be built using the equivalent of 290 full-adders (FA), that corresponds typically to 9280 transistors, hence the complex multiplier would need 37120 transistors and four 16-bit adders, that for ripple-carry adders would add 64 FA(1984 transistors) which gives in total 39104 transistors. These figures made the implementation of FFT processors difficult and expensive until the mid 80's of XXth century. Principally an FFT processor may use input data in the form of segments or there is a constant inflow of data in real-time. The starting point of an FFT processor design is its specification. The main properties are N, which is typically 1024 points but the transform size may extend from 64 through $64K$ to 0.75 10^6 points [11], the dynamic range of the input signal from 12 to 32 bits [12], latency, L even under 1 μs, accuracy and power consumption. The required parameters determine the choice of fixed-point two's complement, block-floating, floating-point or residue arithmetic. The latency determines the maximum frequency of the input signal that can be processed. Nowadays four main approaches to the implementation are possible: Application Specific Circuits(ASICs), field programmable gate arrays(FPGAs). standalone digital signal processors(DSP) with/or without hardware accelerator and general-purpose Graphic Processing Units(GPU). ASICs allow to attain the assumed short FFT execution time and fulfil the design goals. The ASIC design is expensive and may be time-consumming, moreover, ASICs can not be modified when needed. FPGAs in their early stage were about ten times slower than ASICs so they could serve only as the prototyping platform. The advantages of FPGAs include the shorter design time and lower costs compared with ASICs. The progress in FPGAs is due not only to the increase in the number LUTs and the clocking frequences exceeding several hundred MHz[13, 14], but the main innovation was the introduction of DSP units that contain 18x18 bit or 18x24 bit multipliers with their number up to 2048 [13]. In FPGAs an FFT processor can be implemented using one of the three approaches: the description of the structure in the VHDL or Verilog, using FFT cores supplied by FPGA manufacturers or the high-level synthesis(HLS). The HLS allows to describe the FFT algorithm in the specially adapted C language version. The several variants of the C FFT program can be examined with respect to the use of the FPGA resources, latency and pipelining rate. The digital signal processors have been used extensively for the FFT compu-

tation since 90's of XXth. The representative example can be TMS320C6678 processor from the Texas Instruments C6000 family [15]. GPUs can be especially useful for multidensional FFTs, an example of the GPU is described in [16]. First designs of FFT processors appeared shortly after the FFT algorithm was published in 1965. Bergland [17] in 1969 presented the features of over twenty existing at that time hardware FFT processors. The processor architecture, type of arithmetic used and performance system and cost were shown. Most of these processors were produced by research laboratories, such as Bell Laboratories, Stanford Research Institute, MIT Lincoln Laboratory for specific applications, especially radars. As a standard measure to compare individual processors was the execution time for 1024-point complex FFT. The speed range was from 1 ms for the MM DSP of Emerson Electric (only designed) to 600 ms for the CSS-3 of Computer Signal Processors, Inc. Typically, the execution time for a 1024-point complex FFT was in the tens of milliseconds (22-56 ms). The technology used was Medium Scale of Integration(MSI) or ECL (Emmiter Coupled Logic). Over a decade of 70's several FFT processors architectures were presented [18-22]. Many special purpose high-speed FFT processors have been built by various research laboratories and in the industry. However, neither their technical characteristics were available nor processors which could be applied to real-time radar signal were commercially accesible. Commercially available FFT processors usually took the form of an array processor. Array processors have high flexibility and tend to be software orientated. Examples of these processors include the Floating Point System, Inc. (FPS) family of array processors. FPS AP-120B processor, available in 1976, was a 38-bit pipeline-oriented array processor. It was designed to be attached to a host computer such as a DEC PDP-11 as an hardware accelerator. Data transfer was accomplished using the direct memory access. The execution time for a 1024-point complex FFT was 5 msec. Another example could be Star ST-50 array processor developed by Star Technologies Inc. in 1986 with 50 MFLOPs and the TASP array processor developed by ESE**.** The throughput rates of these processors ranged from 25 MFLOPs for an AP-120B to over 100 MFLOPs for the TASP which could calculate the 1024-point complex FFT in 800 μs**.** Families of IC chips specially designed for FFT processing were available from a device manufacturers such as AM29500 family by Advanced Micro Devices and the Weitek family signal processing ICs. These devices could be used to form the building blocks of a high-speed FFT processor for real-time radar applications. However, the cost of these devices was high. Also in 1980 first NEC μ7720 digital signal processor became available. A good example of the progress in FFT processors is given in [23] where the Modular Transform Processor that computed the 4096-point FFT and IFFT was built by TRW, Inc. in 1982-1984. It operated in a pipeline fashion at a clock rate of 10 MHz. Six large circuits boards were used that dissipated about 1200W. Each board contained 200 integrated circuits. Two decades after an FFT processor for the same transform size was built by Mayo Foundation in 2002–

2003. The Mayo single-chip 4096-point ASIC FFT processor was manufactured using 0.25μm CMOS, could be clocked at 100 MHz and consummed 2.6 W. Cuurently the most effective approach to the FFT processor design is the use of pipelined architectures. In principle there exist four basic types of pipelines architectures: for radix-2 Multi-path Delay Commutator(R2MDC) [3], Single-path Delay Feedback(R2SDF) [18], and for radix-4 R4SDF [19], R4MDC [3, 20], R4SDC [21, 22]. In [22] the largest single-chip FFT processor ever built was presented. But the most effective with respect to the number of complex multiplications is the radix-2^2 Single-path Delay Feedback(R2^2SDF) architecture presented in [24–25]. Next, radix-2^3 and radix-2^4 architectures which enable certain complex multipliers to be simplified, were also elaborated. A description and explanation of radix-2^k SDF architectures is given in [26]. Recent works in FFT processors concentrated mainly on their applications to OFDM and UWB [27–36]. Other works encompass various problems of the FFT processor implementation [37–47]. In the current decade the tendency can be observed of implementing FFT processors in FPGA structures. Altera's Stratix 5SGSD8A [48] 28 nm chip allowed to calculate 4096-point FFT in fixed-point in 12.04 μs, whereas for 1024-point FFT [49] using Stratix V 5SGS D5 3.31μs was attained. Centar LLC [50] reports in 2018 that using Intel's Arria 10 and the proprietary FFT core it was possible to compute the fixed-point 1024-point FFT in 1.92 μs and for IEEE 754 in 1.79μs. However, the design of an FPGA FFT processor is still a complex task. It involves not only timing requirements but the choice of the binary representation length of intermediate results and W^{nk} and also the interstage scaling procedures. When using an FFT core the suitable choice is usually possible. The design description in the VHDL or Verilog is complex and poses a serious obstacle. This problem can be partially relieved by using tools termed High Level Synthesis (HLS), such as Xilinx HLS [51] or Altera's(Intel's) OpenCL [52]. The aim of this work is to show exemplary results of the HLS of an FFT processor and examine the possible ways to improve the properties of the design. In Section 2 the FFT algorithm is reviewed, and in Section I3 strategies are considered to improve the properties of the FFT processor design, and in Section 4 the exemplary implementations of the FFT processor using the HLS synthesis are shown.

2. FFT ALGORITHM DESCRIPTION

The Discrete Fourier Transform has the following form

$$X(k) = \Sigma_{n=0}^{N-1} x(n) \cdot W_N^{kn} \text{ for } k = 0,1,...,N-1. \quad (1)$$

The computation of the DFT requires $O(N^2)$ complex multiplications and additions. In order to make the DFT calculations more efficient, the following properties of coefficients $W_N^{kn} = e^{-j(2\pi/N)kn}$ can be used: symmetry

$W_N^{k+N/2} = -W_N^k = e^{j\pi} W_N^k$, periodicity $W_N^{k+N} = W_N^k$ and recursion $W_N^2 = W_{N/2}$. By taking advantage of these properties we can decompose sequence $x(n)$ into the shorter sequences. Such approach is called the decimation-in-time (DIT) algorithm. The DIT principle can be presented by considering a special case when N is the power of 2. Since N is an even number, we can determine $X(k)$ by dividing the sequence $x(n)$ into two ($N/2$) - point sequences containing even and odd elements of $x(n)$. In case of $X(k)$ given by (1) we have two sequences

$$X(k) = \sum_{n\,even} x(n) \cdot W_N^{nk} + \sum_{n\,odd} x(n) \cdot W_N^{nk}. \tag{2}$$

Making the substitution of $n = 2r$ for n even and of $n = 2r+1$ for n odd we obtain

$$\begin{aligned} X(k) &= \sum_{r=0}^{N/2-1} x(2r) \cdot W_N^{2rk} + \sum_{r=0}^{N/2-1} x(2r+1) \cdot W_N^{(2r+1)k} \\ &= \sum_{r=0}^{N/2-1} x(2r) \cdot (W_N^2)^{rk} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1) \cdot (W_N^2)^{rk}, \end{aligned} \tag{3}$$

where

$$W_N^2 = e^{-2j(2\pi/N)} = e^{-j(2\pi/(N/2))} = W_{N/2}. \tag{4}$$

Finally, the equation (3) can be written in the following form

$$X(k) = \sum_{r=0}^{N/2-1} x(2r) \cdot W_{N/2}^{rk} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1) \cdot W_{N/2}^{rk}. \tag{5}$$

The basic DFT operation is called the butterfly (Fig. 1) and has the followig form

$$\begin{aligned} X_{m+1}(p) &= X_m(p) + W_N^r X_m(q), \\ X_{m+1}(q) &= X_m(p) + W_N^{r+N/2} X_m(q). \end{aligned} \tag{6}$$

If we substitute $W_N^{N/2} = e^{-2j(2\pi/N)N/2} = e^{-j\pi} = -1$ into (6) we get

$$\begin{aligned} X_{m+1}(p) &= X_m(p) + W_N^r X_m(q), \\ X_{m+1}(q) &= X_m(p) - W_N^r X_m(q). \end{aligned} \tag{7}$$

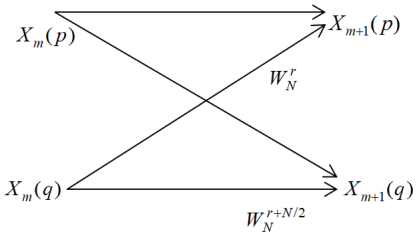

Fig. 1. Basic flow graph of butterfly calculation

The radix-2 FFT equation (5) decomposes the DFT into two smaller DFTs. Each of the smaller DFTs is then further decomposed into smaller ones. The full structure consists of $log_2 N$ stages and each stage consists of $N/2$ butterflies (Fig. 2).

Each butterfly (7) performs two additions for the input data and one multiplication by the twiddle factor.

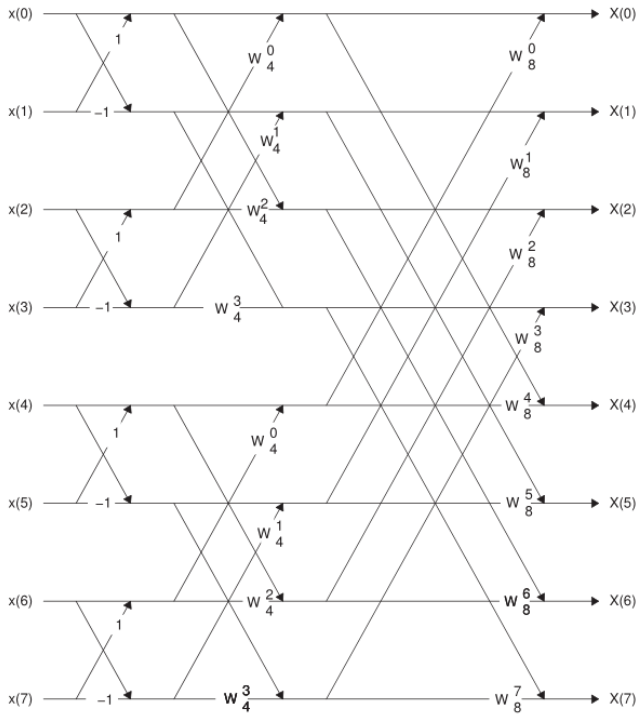

Fig. 2. 8-Point DIT-Radix2-FFT

The other popular algorithm is the radix-4 FFT, which is more efficient than the radix-2 FFT. The radix-4 FFT equation (8) is listed below

$$X(k) = \sum_{r=0}^{N/4-1} x(r)W_N^{rk} + \sum_{r=N/4}^{N/2-1} x(r)W_N^{rk} + \sum_{r=N/2}^{3N/4-1} x(r)W_N^{rk} + \sum_{r=3N/4}^{N-1} x(r)W_N^{rk}$$
$$= \sum_{r=0}^{N/4-1} x(r)W_N^{k/4} + W_N^{kN/4} \sum_{r=0}^{N/4-1} x(r+N/4)W_N^{rk} + \tag{8}$$
$$W_N^{kN/2} \sum_{r=0}^{N/4-1} x(r+N/2)W_N^{rk} + W_N^{k3N/4} \sum_{r=0}^{N/4-1} x(r+3N/4)W_N^{rk}.$$

From the definition of the twiddle factors we obtain $W_N^{k3N/4} = j^k$, $W_N^{kN/2} = (-1)^k$, $W_N^{kN/4} = (-j)^k$. After substituting the values into (8) we have the final form

$$X(k) = \sum_{r=0}^{N/4-1} \left[\begin{array}{c} x(r) + (-j)^k x(r+N/4) + \\ +(-1)^k x(r+N/2) + (j)^k x(r+3N/4) \end{array} \right] \cdot W_N^{rk}. \tag{9}$$

The radix-4 FFT in (8) in the simplified form combines two stages of a radix-2 FFT into one. As the result the number of required stages is halved. Since the radix-4 FFT requires fewer stages and butterflies than the radix 2 FFT, the computations of FFT can be further improved. For example, to calculate the 16-point FFT, the radix-2 requires $log_2 16 = 4$ stages but the radix-4 only $log_4 16 = 2$ stages.

3. STRATEGIES TO IMPROVE PERFORMNCE USING HLS

FPGAs are the very attractive platform for high-speed digital signal processing, especially for radar technology [48, 49] but the implementation of the demanding signal processing algorithms as the FFT in the real time is complex because firstly the circuit structure must be designed and next implemented at the register-transfer level (RTL) using a hardware description language (HDL) such as VHDL and then synthesized and tested in the FPGA chip. However, do exists FFT IP cores, for example Xilinx [53] or Intel [54], but their configuration is difficult because of the required in-depth knowledge of the FFT core processor structure and arithmetic properties. This may make difficult the choice between the FPGA and digital signal processor such as TMS320C66x [15]. In order to simplify the design and implementation process the algorithmic approach was introduced by FPGA manufacturers. This approach is becoming more and more popular nowadays due to the accelerated design time and time-to-market (TTM). Large hardware projects pose major challenges in the design and verification of hardware at the HDL level. An increasing trend is observed as moving towards hardware acceleration to enhance performance of CPU-intensive tasks. It can be offloaded to hardware accelerator in FPGA. The HDL synthesis can be performed using behavioral or structural descriptions with Verilog or VHDL. In general, the design of complex digital systems using these languages is laborious and difficult. In order to make the design process faster and more effective the design methods are known that make use of C-based languages. The digital system description in this case has the form of a program in one of such languages. However, the program must have the form that could be translated into the register transfer level (RTL)form. With the HLS tool, as Xilinx Vivado HLS, an algorithmic description written in C-based language that maps the design flow, can be converted into the RTL form. During this conversion the control and dataflow structures from the source code are extracted. The key attributes of C-based code are functions, arguments of the top-level function (the interface), data types and loops. The way they are handled can have the significant impact on the area and performance. Also the translation of arrays and operators has considerable influence on the form of RTL description. Various implementations are possible when using the same source as a starting point. For example, it is possible to obtain as a result smaller or faster designs or nearly optimal designs. In the Xilinx HLS this can be done during the

design exploration using dedicated directives applied to critical fragments of the source code. However, the HLS synthesizer has the certain initial configuration options to be used to match the C code to the dedicated FPGA structure, afterwards the use of special preprocessor directives in critical places to improve the efficiency of implementation may be needed in the testing and startup phase of the design in HLS.

In the following we shall discuss the basic principles of the HLS optimization methodology. The most important steps can be summarized as: the determination of the interfaces, the application of the pipelining to the design, the addressing the issues the which could prevent the optimal pipelining by optimizing data structures and then consider any latency and area problems.

The form of C-function signature determines I/O of the circuit. The type of the I/O is a key factor to determine what kind of hardware implementation can be achieved by synthesis. It is recommended in the Vivado HLS to use the INTERFACE directive to specify how RTL ports are created from the function signature. This step should be considered yet before going over to the optimization of the design.

One of the main performance factors are latency and throughput. The latency of the design is the number of cycles it takes to output the result. The throughput of the design is the number of cycles between accepting two succeeding inputs. If there is no concurrency in the design the throughput and the latency are the same. Let us consider the example as in Fig. 3. Without the optimization the top task `f_top()` has a latency of 8, the tasks `f_A()` / `f_C()` of 3 and the task `f_B()` of 2 clock cycles (Fig 4a). If the data flow optimization is applied we can reduce the throughput to 3 cycles (Fig. 4b). This example uses functions as a tasks, but this kind of optimization can be performed in the same way between functions, between functions and loops, and between loops. In the C-based language the function call denotes the jump to the function code, its execution and possible return to the caller. However, in the HLS such function calls translate to the separate circuit structure that performs the respective tasks.

```
void f_top(a,b,c,d)
{
    f_A(a,b,x1);
    f_B(c,x1,x2);   } 8C
    f_C(x2,d);
return d;
}
```

Fig. 3. An example of the HLS code

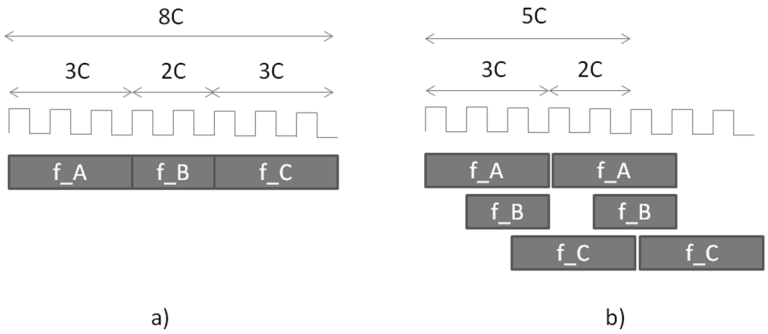

Fig. 4. Optimization scheme of data flow in the HLS design

The latency minimization in the Vivado HLS is done by default and throughput is prioritized above latency. Additionally the HLS offers four directives to achieve the required pipeline performance: PIPELINE, DATAFLOW, RESOURCE and configuration of config profile. The PIPELINE directive reduces the initiation interval by allowing the concurrent execution of operations within a loop or function. The DATAFLOW can be used to minimize interval and enables task level pipelining by allowing the concurrency of internal functions and loops operations. The RESOURCE directive specifies the choice of the specific resource that will be used to implement an array, arithmetic operation, or function argument in the RTL. There are several rules that govern the efficiency of pipelining strategies. For example, if the design uses sub-functions and the sub-functions should be pipelined, it should be explicitly imposed using the appropriate pipelining directive. Otherwise the non-pipelined functions could be the bottleneck. Another example are some functions and loops that contain internal loops. The PIPELINE directive automatically unrolls all loops in the nested hierarchy, which in the case of the multilevel nesting can lead to a complicated structure. But it may be more reasonable to use this directive to pipeline the loops that need pipelining. Loops that contain certain bounds imposed on variables cannot be unrolled. Moreover, any loops that contain such loops can be unrolled. In order to achieve better performance the DATAFLOW directive can be used or alternatively we have to modify the loop to remove variable bound.

The default option in the HLS is to enforce parallelism. Therefore, the tasks (functions) are scheduled to start as soon as they can and they are synthesized to execute in parallel. It looks slightly different for the loop which are by default executed sequentially. Note that Vivado HLS automatically performs latency optimizations during synthesis, but there are certain additional directives to control it. The LATENCY directive allows to set up latency boundaries in advance, the LOOP_FLATTEN and LOOP_MERGE directives allow for loop merging and flattening, and UNROLL directive provides the loop unrolling. Summarizing thus far, we would like to show a short benchmark. In Fig. 5 the source code of the

exemplary top-level function `mtrx_el_mul()` created for the performance optimization analysis is shown.

```
void mtrx_el_mul(uint16 w1[H][W], uint16 w2[H][W], uint16 o1[H][W])
{
    int t=0;
    l1: for(int i=0;i<H;i++)
    {
        l2: for(int j=0;j<W;j++)
        {
            o1[i][j] = w1[i][j] * w2[i][j];//element by element matrix mul
            t=0;
            l3: for(int k=0;k<16;k++)
            {
                if((o1[i][j]<<k)==1)//counting bits==1
                    t++;
            }
            o1[i][j]=o1[i][j]^t;//bit mask
        }
    }
}
```

Fig. 5. Source code of mtrx_el_mul() for performance optimization analysis

After the directive LOOP_FLATTEN was used the performance is the same as before optimization (Tab. 1, Tab. 2), since the Vivado synthesizer flattens loops by default. The application of both the PIPELINE and UNROLL directives to `l2` and `l3` in our case did not increase the throughput. The synthesizer removed the PIPELINE because the loops are completely unrolled. It is worth to mention, that after applying more optimization directives the HLS the compilation time increased several times.

Table 1. Performance analysis of mtrx_el_mul() with optimization.

Optimization	Latency [clk]	Interval [clk]
Default	14953	14954
L3 unrolled	2409	2410
L1-L3 unrolled	393	394
Pipelined	393	394

Table 2. Hardware analysis of mtrx_el_mul() with optimization.

Optimization	BRAM_18K	DSP48E	FF	LUT
Default	0	1	385	266
L3 unrolled	0	1	176	131
L1-L3 unrolled	0	784	12938	37708
Pipelined	0	784	12938	37708

Concluding it should be remarked that in the HLS only a subset of type and grammar constructs of C-code is allowed. As the data types the primitive types like unsigned char, unsigned short int, unsigned int long with their signed counterparts and real types float and double can be applied. In the HLS the arbitrary precision integer and floating-point types are preferred. These data types allow for the implementation of bit vectors with selected lengths.

4. IMPLEMENTATIONS OF FFT PROCESSOR USING HIGH LEVEL SYNTHESIS

As the part of this study a 1024-point radix-2 FFT algorithm has been implemented. The C source code has been tested and synthesized in the Xilinx Vivado HLS 2018.1 environment. We used the Virtex-7 xc7vx485tffg1761 FPGA. The structure of the implemented program includes a top function (Fig.6) representing the data input and output interface by the six parameters list, consisting of two arrays for real and imaginary parts of coefficients, two arrays for real and imaginary parts of input samples and the same for output samples. In the case when specific I/O interface has not been specified the HLS synthesizer treats arrays as buffers. It is worth to note that AXI is the standard interface for data entry in Xilinx FPGAs to realize interconnections between building blocks inside the system. Here, the FFT processor was synthesized in two versions, using the AXI interface and without it. The top function implements the full mechanism for performing butterfly operations. The single butterfly operation requires the implementation of the algorithm given in Fig. 6. The full FFT structure was implemented in the form of ten blocks with each of them representing the consecutive layer of the FFT algorithm. In every layer two nested loops are used to generate the connections between 512 butterflies (Fig. 7).

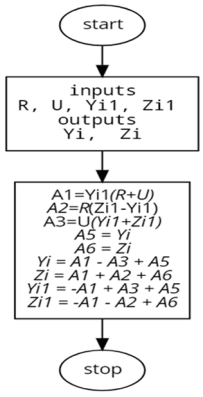

Fig. 6. Block diagram of FFT radix-2 butterfly procedure

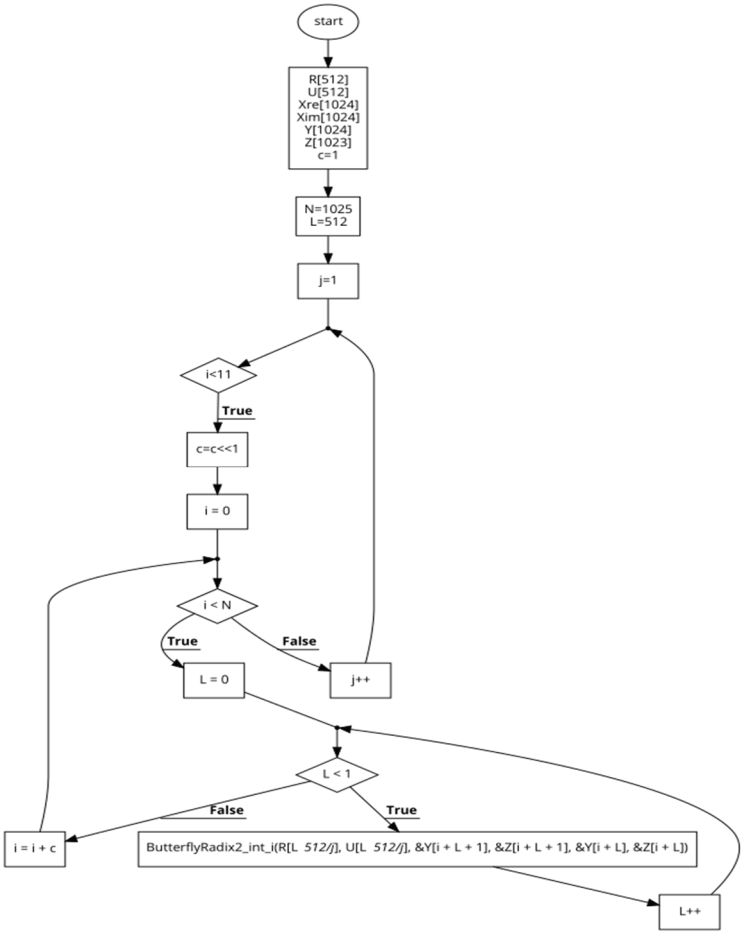

Fig. 7. Block diagram of FFT Radix-2 procedure

The synthesis results show that for implementation with default HLS optimizations (Fig. 9) the serial hardware structure utilizes four DSP48 multipliers and the proper number of adders. The structure has the highest latency of about 3500 clock cycles (Tab.3) but calls for the lowest hardware amount. After defining the AXI interface in the FFT processor structure and application of the pipelining but without loop unrolling we reduced the latency to about 15000 clock cycles (Tab. 3). It should be noted that AXI interface implementation consumes about 4000 cycles in our case and introduces the need for memory blocks (BRAM_18K) with total increase of required hardware amount (Fig. 9). Finally, after applying of pipelining and loop unrolling (Fig. 10) the hardware amount requirements strongly increased with utilization of 65% LUTs and 774 DSP48 blocks of all resources of the given FPGA chip.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	6161	3495
FIFO	-	-	-	-
Instance	-	4	188	155
Memory	-	-	-	-
Multiplexer	-	-	-	11565
Register	-	-	5325	-
Total	0	4	11674	15215
Available	2060	2800	607200	303600
Utilization (%)	0	~0	1	5

Fig. 8. Synthesis utilization estimates for FFT Radix-2 with default optimization (Virtex-7 xc7vx485tffg1761)

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	10497
FIFO	-	-	-	-
Instance	8	3	2212	2864
Memory	4	-	0	0
Multiplexer	-	-	-	21477
Register	-	-	38507	-
Total	12	3	40719	34838
Available	2060	2800	607200	303600
Utilization (%)	~0	~0	6	11

Fig. 9. Synthesis utilization estimates for FFT Radix-2 with without loop unrolling, with pipelining and AXI interface (Virtex-7 xc7vx485tffg1761)

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	8
FIFO	-	-	-	-
Instance	8	774	18660	42956
Memory	4	-	0	0
Multiplexer	-	-	-	155771
Register	-	-	102240	-
Total	12	774	120900	198735
Available	2060	2800	607200	303600
Utilization (%)	~0	27	19	65

Fig. 10. Synthesis utilization estimates for FFT radix-2 with loop unrolling, pipelining and AXI interface (Virtex-7 xc7vx485tffg1761)

Table 3. Performance analysis of 1024 FFT radix-2 with optimization.

Optimization	Latency [clk]
Default	35846
1st stage loop unrolling	33804
Pipelining without unroll (AXI)	14614
Pipelining without unroll (without AXI)	11285
Pipelining, unrolling (AXI)	1561

5. CONCLUSIONS

The paper presents the results of the use of high level synthesis for implementation of radix-2 FFT processors using Xilinx Vivado HLS tool. The various options of the design flow were examined in order to identify the proper set of directives that would lead to possibly fast implementation of the FFT processor. It was stated that the optimal choice is the use of pipelining and along with the proper form of loop unrolling. Such approach substantially reduces the latency of the processor from 35k cycles when default options are used to about 1.5 k cycles. It means that 1024-point FFT transform calculation for 500MHz clock without the bit reversal would require about 3μs.

REFERENCES

[1] Brigham E.O., The Fast Fourier Transform and its Applications, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1988.
[2] Brandwood D., Fourier Transforms in Radar and Signal Processing, Artech House, 2003.
[3] Rabiner L.R., Gold B., Theory and Application of Digital Signal Processing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.
[4] Oppenheim A.V., Schafer R.W., Digital Signal Processing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.
[5] Oppenheim A.V., Schafer R.W., Discrete-Time Signal Processing. Third Edition, Prentice-Hall, 2009.
[6] McClellan J. H., Purdy R. J., Applications of Digital Signal Processing. Prentice-Hall, 1978, (ch. 5, Applications of Digital Signal Processing to Radar).
[7] Hwang T.,Yang C., Wu G., Li S.G., Li Y., OFDM and Its Wireless Applications: A Survey, IEEE Transactions on Vehicular Technology, Volume. 58, Number 4, May 2009, pp. 1673-1694.
[8] Chikada Y. *et. al*., A very fast spectrum analyzer for radio astronomy, International Conference on Acoustics, Speech and Signal Processing, Tokyo, 1986, pp. 2907-2910.
[9] Brezinski M.E, Optical Coherence Tomography: Principles and Applications, Elsevier, 2006.

[10] Cooley J.W., Tukey J.W., An algorithm for the machine calculation of complex Fourier series, Mathematics of Computation., Volume. 19, pp. 297–301, 1965.
[11] Abari O., Hamed E., Hassanieh H., Agarwal A.,Katabi D., Chandrakasan A.P., Stojanovic V., A 0.75-millionpoint fourier-tranform chip for frequency-sparse signals, 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), pp. 458–460, Feb. 2014.
[12] Hopkinson T.M. Butler M, A pipelined high-precision FFT architecture, Mitre Corp.,1992.
[13] Virtex-7, Xilinx, Inc., 2020.
[14] Stratix IV, Altera Inc., 2014.
[15] Texas Instruments, 6000^{TM} high-performance and power efficient DSP, 2020.
[16] Fialka O., Cadik M., FFT and convolution performance in image filtering on GPU, 10th Conference on Information Visualisation, 5-7 July, London, UK,
[17] Bergland G.D., Fast Fourier Transform Hardware Implementations -A Survey, IEEE Transactions on Audio and Electroacoustics, Volume AU-17, Number 2, June 1969, pp. 109-119.
[18] Wold E. H., Despain A. M., Pipeline and parallel-pipeline FFT processors for VLSI implementations, IEEE Transactions on Computers, Volume C-31, Number 5, pp. 414–426, May 1984.
[19] Despain A. M., Fourier transform computers using CORDIC iterations. IEEE Transactions on Computers., Volume C-23, pp. 993-1001, Oct. 1974.
[20] Swartzlander E. E., Young W. K. W, Joseph S. J., A radix-4 delay commutator for fast Fourier transform processor implementation, IEEE Journal. Solid-State Circuits, Volume 19, Number 5, pp. 702–709, Oct. 1984.
[21] Bi G., Jones E.V., A pipelined FFT processor for word-sequential data. IEEE Transactions on Acoustics, Speech and Signal Processing, Volume 37, Number 12, pp. 1982-1985, Dec. 1989.
[22] Bidet E., Castelain D., Jaoanblanq C., Stenn P., A fast single chip implementation of 8192 complex point FFT, IEEE Journal of Solid-State Circuits., Volume 30, Number 3, pp. 300-315, March 1995.
[23] Swartzlander Jr E.E, Systolic FFT processors: A Personal Perspective, Journal of Signal Processing Systems, Number 53, pp. 3-14, 2008.
[24] He S., Torkelson M., A new approach to pipeline FFT processor, in; Proceedings of the 10th Parallel Processing Symposium, IPPS'96 ,pp.766-770, April 1996.
[25] He S., Torkelson M., Design and implementation of a 1024-point pipeline FFT processor, in: Proceedings of the IEEE Custom Integrated Circuits Conference, May 1998, pp. 131–134.
[26] Cores A., Velez I., Sevillano, J.F.,r^{k} Radix FFTs matrical representation and SDC/SDF implementation, IEEE on Signal Processing, Volune 57, Number 7, pp. 2824-2839, July 2009.
[27] Liu H., Lee H., A high performance four-parallel 128/64-point radix-24 FFT/IFFT processor for MIMO-OFDM systems, in: Proceedings of IEEE Asia Pacific Conference on Circuits and Systems, 2008, pp. 834–837.
[28] Cho S.-I., Kang K.-M., Choi S.-S., Implementation of 128-point fast Fourier transform processor for UWB systems, in: Proceedings of International Wireless Communication&Mobile Computing Conference, 2008, pp. 210–213.

[29] Liu L., Ren J., Wang X., Ye F., Design of low-power, 1GS/s throughput FFT processor for MIMO-OFDM UWB communication system, in: Proceedings of the IEEE International Symposium on Circuits and Systems, pp. 2594–2597, May 2007.
[30] Sanchez M. A., Garrido M., Lopez M. L., Grajal J., Implementing FFT-based digital channelized receivers on FPGA platforms, IEEE Transactions on Aerospace and Electronic Systems, Volume. 44, Number 4, pp. 1567–1585, Oct. 2008.
[31] Lee,J., Lee H, Cho S., Choi S.-S., A high-speed, low-complexity radix-2 4 FFT processor for MB-OFDM UWB systems, in: Proceedings of the IEEE International Symposium on Circuits and Systems, 2006, pp. 210–213.
[32] Lin Y.-W., Lee C.-Y., Design of an FFT/IFFT processor for MIMO OFDM systems, IEEE Transactions on Circuits and Systems I, Volume 54, Number. 4.
[33] Xudong W., Yu L., Special-purpose computer for 64-point FFT based on FPGA, in: Proceedings of the International Conference on Wireless Communication and Signal Processing, 2009, pp. 1–6.
[34] Liu S., Liu D., A High-Flexible Low-Latency Memory-Based FFT Processor for 4G, WLAN and Future 5G, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume 27, Number 3, pp. 511-523, March 2019.
[35] Tang S.-N., Tsai J.-W., Chang T.-Y., A 2.4-GS/s FFT processor for OFDM-based WPAN applications, IEEE Transactions on Circuits and Systems I, Volume 57, Number 6, pp. 451–455, June 2010.
[36] Li S., Xu H., Fan W., Chen Y., Zeng X., A 128/256-point pipeline FFT/IFFT processor for MIMO OFDM system IEEE 802.16e, in :Proceedings of the IEEE International Symposium on Circuits and Systems, June 2010, pp. 1488–1491.
[37] Yang L., Zhang K., Liu H., Huang J., Huang S., An efficient locally pipelined FFT processor, IEEE Transactions on Circuits and Systems II, Volume 53, Number 7, pp. 585–589, July 2006.
[38] Cheng C., Parhi K. K., High-throughput VLSI architecture for FFT computation, IEEE Transactions on Circuits and Systems II, Volume 54, Number 10. 10, pp. 863–867, Oct. 2007.
[39] Chang Y.-N., An Efficient VLSI Architecture for Normal I/O Order Pipeline FFT Design, IEEE Transactions on Circuits and Systems II, Volume 55, Number 12, pp. 1234–1238, Dec. 2008.
[40] Garrido M., Parhi K. K., Grajal J., A pipelined FFT architecture for real-valued signals, IEEE Transactions on Circuits and Systems I, Volume 56, Number 12, pp. 2634–2643, Dec. 2009.
[41] Garrido M., Grajal J., Gustafsson O., Optimum circuits for bitreversal, IEEE Transactions on Circuits and Systems II, Volume 58, Number 10, pp. 657–661, Oct. 2011.
[42] Garrido M., Gustafsson O., Grajal J., Accurate rotations based on coefficient scaling, IEEE Transactions on Circuits and Systems II, Volume 58, Number 10, pp. 662–666, Oct. 2011.
[43] Duan B., Wang W., Li X. , Zhang C., Zhang P., Sun N., Floating-point mixed-radix FFT core generation for FPGA and comparison with GPU and CPU, 2011 International Conference on Field-Programmable Technology (FPT), pp. 1–6, Dec.12-14, New Delhi, India, 2011.
[44] Huang S.-J., Chen S.-G., A high-throughput radix-16 FFT processor with parallel and normal input/output ordering for IEEE 802.15.3c systems, IEEE Transactions on Circuits and Systems I, Volume 59, Number 8, pp. 1752–1765, Aug. 2012.

[45]Wang Z., Liu X., He B., Yu F., A combined SDC-SDF architecture for normal I/O pipelined radix-2 FFT, IEEE Transactions on Very Large Scale Integration (VLSI) Systems,Volume 23, Number 5, pp. 973–977, May 2015.
[46] Wang M., Wang F., Wei S., Li Z., A pipelined area-efficient and high-speed reconfigurable processor for floating-point FFT/IFFT and DCT/IDCT computations, Microelectronics Journal, Volume 47, pp. 19–30, 2016.
[47] Garrido M., Huang S.-J., Chen S.-G., Gustafsson O., The serial commutator (SC) FFT, IEEE Transactions on Circuits and Systems II, Volume 63, Number 10, pp. 974–978, Oct. 2016.
[48] Altera, Radar Processing: FPGA or GPU, Altera Corp. 2013.
[49] Parker M. Radar Basics-Part 3, Beamforming and radar digital processing, Altera Corp., 2011
[50] Centar LLC, FFT Circuitry for 4G Age, 2018, www.centar.net.
[51] Vivado Design, Xilinx Inc., 2020.
[52] Intel® FPGA SDK for OpenCL, Intel. Corp., 2020.
[53] Xilinx, Fast Fourier Transform v9.1 LogiCORE IP Product Guide, May 22, 2019.
[54] Intel FFT IP MegaCore 17.1, Nov.2017.

(Received: 10.02.2020, revised: 13.03.2020)