Non-Power-of-Two FFTs: Exploring the Flexibility of the MONTIUM

Pascal T. Wolkotte, Marcel D. van de Burgwal and Gerard J.M. Smit
University of Twente, Department of EEMCS
P.O. Box 217, 7500 AE Enschede, The Netherlands
E-mail:{P.T.Wolkotte, M.D.vandeBurgwal, G.J.M.Smit}@utwente.nl

***Abstract*— Coarse-grained reconfigurable architectures, like the MONTIUM, have proven to be a successful approach for low-power and high-performance computation of regular DSP algorithms. The main research question posed in this paper is: Can such architectures also take over less regular algorithms from general purpose processors? This paper presents the implementation of non-power-of-two Fast Fourier Transforms (FFT) to discover the limitations and flexibility of the MONTIUM. The results of the implementation show a order of magnitude reduction of the processing time and energy consumption compared to an ARM processor. Furthermore, we show the accuracy and flexibility of the implementation.**

I. INTRODUCTION

Implementations of FFTs are mainly focussed on the power-of-two FFTs that use the radix-2 FFT approach. Those are widely used to compare implementations and for benchmarking processor architectures. The algorithms for non-power-of-two FFTs are mainly described at the algorithm level to reduce the number of multiplications and additions as for example discussed by Good [1] and Winograd [2]. More recently a special class of non-power-of-two FFTs are even further optimized to reduce the number of operations [3] [4].

Despite the reduction of operations, the non-power-of-two algorithms are not efficiently executed on processors. A frequently used method to overcome this problem is padding, which adds zeros to the input vector and increases its length to a power-of-two FFT. However, this changes the filter response of the FFT and it will lose its orthogonal characteristic. This orthogonality is required in Orthogonal Frequency Division Multiplexing (OFDM) systems and, therefore, efficient non-power-of-two FFT implementations are required. Rivaton et al. [5] present a comparison of non-power-of-two FFTs on two coarse-grained reconfigurable architectures and a general purpose processor. A high-speed FFT-1872 implementation on an FPGA is presented in [6].

In the Smart chipS for Smart Surroundings (4S) project [7] we propose a heterogeneous tiled architecture with run-time software and tools. This architecture contains small processing tiles interconnected by a Network-on-Chip. The architecture is an energy-efficient solution for flexible and computational intensive applications. The general purpose processing tiles, like an ARM, can offload their computational intensive tasks to coarse-grained reconfigurable processing tiles, like the MONTIUM. This reduces both processing time and energy consumption of the architecture.

The main driver application used to validate this architecture is Digital Radio Mondiale (DRM) [8]. The DRM standard proposes the digitization of radio broadcasting in frequency bands below 30 MHz. DRM will be the upcoming successor of AM radio and it is based on OFDM and MPEG-4 audio source coding. In the baseband processing of a DRM receiver several demodulation schemes have to be supported. All schemes heavily rely on several non-power-of-two FFTs.

This paper proceeds on the results of Rivaton et al. [5], in which the implementation of the FFT-1920[1] on the coarse-grained reconfigurable MONTIUM architecture was discussed (see [9] for more information about the MONTIUM). The algorithm implementation is extended such that it now supports a wide range of non-power-of-two FFTs and iFFTs on the same architecture, including all required in the DRM application. The extended implementation showed to be an ideal test-case to explore and validate the flexibility of the coarse-grained architecture. Furthermore, we validated new possibilities in the MONTIUM tooling to enable partial reconfiguration of the algorithm.

The paper is organized as follows. Section II introduces the non-power-of-two FFTs and their decomposition, such that they can be mapped on the MONTIUM architecture. More details on the decomposition and the mapping itself are given in section III. The results, by means of processing time, accuracy, and power consumption, are presented in section IV. Consequently, in section V we discuss why executing this less regular type of algorithms can be done on the MONTIUM very well. Section VI concludes the paper.

II. NON-POWER-OF-TWO FFTS

The Discrete Fourier Transform (DFT) transforms a digital signal from the time domain to the frequency domain. It is defined by the following relation between N complex input samples x and N complex output samples X:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{nk} \tag{1}$$

where $W_N^{nk} = e^{-j2\pi nk/N}$ are primitive roots of the unit circle also called *twiddle factors*. Each of the N outputs is the sum of N terms, so a straight-forward evaluation of this formula requires $O\left(N^2\right)$ operations.

[1] In this paper, FFT-N means an FFT of length N

1-4244-0622-6/06/$20.00 ©2006 IEEE.

TABLE I

FFTS THAT CAN BE GENERATED WITH THE PFA MAPPING

		q			
		4	5	6	7
p	2	80	160	320	640
	3	112	224	448	896
	4	144	288	576	1152
	5	176	352	704	1408
	6	208	416	832	1664
	7	240	480	960	1920

The Fast Fourier Transform (FFT) is a set of algorithms that improves the efficiency of the DFT. A well known FFT algorithm is the *divide and conquer* approach reintroduced by Cooley and Tukey [10]. This radix-x FFT algorithm recursively re-expresses a DFT of length $N = N_1 \cdot N_2$ into N_2 DFTs of size N_1. For an FFT with a length that is a power of x the recursion can be done in ${}^{x}\log(N)$ stages using a x-inputs butterfly.

For lengths that are not a power of x other optimized algorithms exist. Several algorithms have been proposed to optimize these DFTs to mixed-radix FFTs. Another mapping introduced by Good [1], optimized for the number of multiplications and additions, is known as the Prime Factor Algorithm (PFA). This algorithm uses special ordering of input and output vectors to reduce the number of multiplications and additions. This ordering is not so regular as for example the bit-reversed order used in the the radix-2 FFT.

A. FFTs used in DRM

For the DRM receiver, a large number of FFTs is required. DRM can be used in several modes, each requiring a different set of FFTs. The OFDM processing requires a number of radix-2 FFTs (512, 256) and a set of non-power-of-two FFTs (1920, 576, 352, 288, 224, 176 and 112). These non-power-of-two FFTs can be generalized to a a group of 2-dimensional PFA decompositions that satisfy the following constraints:

$$N = N_1 \cdot N_2 = (2p+1) \cdot 2^q \tag{2}$$

where $2 \leq p \leq 7$, $q \geq 4$ and $N < 2048$.

We developed a generic source-code that covers all cases of the equation above. This makes it possible to quickly generate a large number of configurations. A selection of the cases is given in Table I. In this paper we use the FFT-1920 as an example. Section III gives more details on the implementation, where we use FFT-1920 as an example.

III. IMPLEMENTATION

In this section we take the FFT-1920 to describe the implementation of any non-power-of-two FFT. As explained in section II, any FFT can be decomposed in several smaller FFTs. For the FFT-1920, the parameters are $N_1 = 2{\cdot}7{+}1 = 15$ and $N_2 = 2^7 = 128$. According to the PFA approach, the FFT is decomposed into 128 times FFT-15 followed by 15 times FFT-128. The FFT-128 is implemented using a radix-2 approach [9]. The FFT-15 uses the symmetry in the twiddle factors that results in a reduction of the number of multiplications by a factor of 4. Using this symmetry results in a regular DFT-15, which consists of a chain of additions:

$$\begin{aligned}
T_R[k] &= x[0] + \sum_{n=1}^{\frac{N_1-1}{2}} (x[n] + x[N_1-n]) \cdot \Re\left(W_{N_1}^{nk}\right) \\
T_I[k] &= \sum_{n=1}^{\frac{N_1-1}{2}} (x[n] - x[N_1-n]) \cdot \Im\left(W_{N_1}^{nk}\right) \\
X[0] &= \sum_{n=0}^{N_1-1} x[n] \\
X[k] &= \begin{cases} T_R[k] + T_I[k], & 1 \leq k \leq \frac{N_1-1}{2} \\ T_R[N_1-k] - T_I[N_1-k], & \frac{N_1-1}{2} < k < N_1 \end{cases}
\end{aligned} \tag{3}$$

For each addition, the operands are multiplied with a twiddle factor. Two inputs are added before a multiplication and the butterfly structure of the MONTIUM is used to calculate $X[k]$ and $X[15-k]$ concurrently. The execution of this optimized implementation for the MONTIUM requires 56 clock cycles. For other odd-number FFTs the number of clock cycles equals $\left(\frac{N-1}{2}\right)^2 + \frac{N-1}{2}$. The FFT-15 could have been computed more efficiently by again applying the PFA with FFT-3 and FFT-5, but this requires reordering of the intermediate results which cannot be implemented efficiently on the MONTIUM architecture.

As 128 is a power of two, the FFT-128 can be performed using radix-2 algorithms that require $\left(\frac{N_2}{2} + 2\right) \cdot \log_2(N_2)$ clock cycles on the MONTIUM.

The total number of clock cycles to calculate a FFT of length $N = N_1 \cdot N_2$ equals: $N_1 \cdot \left(\left(\frac{N_2}{2} + 2\right) \cdot \log_2(N_2)\right) + N_2 \cdot \left(\left(\frac{N_1-1}{2}\right)^2 + \frac{N_1-1}{2}\right)$ plus a few clock cycles to initialize some of the registers. For the FFT-1920 the exact numbers are given in section IV-B.

A. Scaling

A fixed-point implementation of a DSP algorithm is liable to overflow after an addition. To prevent overflow the amplitude of the input signal can be limited or the intermediate values can be scaled. Scaling the fixed-point numbers results in a shift of the notation. Scaling a number in (1,15)-fixed-point notation by 64 results in a number in (7,9)-fixed-point notation. For an FFT the worst-case required scaling factor equals $\sqrt{2}N$. In normal operation, with a signal that contains several frequency components, the scaling factor can be smaller, which results in a more accurate output signal. Therefore, we implemented a flexible solution, that supports scaling the signal at predefined positions. Figure 1 depicts these positions in the algorithm where scaling can be applied. The scaling factor consists of the following factors:

$$S = S_0 \cdot \prod_{i=1}^{m} S_i \tag{4}$$

where S_0 is the input scaling factor, m is the total number of stages in the radix-2 FFT and S_i denotes the scaling factor

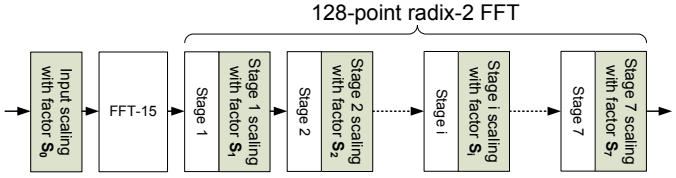

Fig. 1. Positions where scaling can be applied

during stage i of the radix-2 FFT ($S_0 \in \left[1, 2^{15}-1\right]$, $S_i \in \{1, 2\}$).

The scaling can be positioned in the beginning ($S_0 = S$), which results in a less accurate result and lower overflow risk. Moving scaling to the end of the algorithm improves the accuracy but increases the risk of overflow. Therefore, the programmer has to do the adjustment of the scaling factors depending on the expected input signal level. Section IV-D shows the effect of changing the scaling. The implementation of a dynamic scaling algorithm, based on the intermediate signal level, would cost too much resources in the MONTIUM.

B. Run-Time FFT/iFFT reconfiguration

The FFT and inverse FFT (iFFT) equations are quite similar, as can be seen in equations 1 and 5:

$$x(n) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X(k) \cdot W_N^{-nk} \tag{5}$$

The main differences between the FFT and the iFFT is the $\frac{1}{N}$ scaling factor that has to be applied to the results and the usage of complex conjugated twiddle factors. The twiddle factors can be updated easily, since they are located in the data memory map of the MONTIUM. For the $\frac{1}{N}$ scaling factor the designer has to compensate for the scaling applied in the FFT. If the designer has a scaling factor of S in the FFT then a scaling factor of $\frac{S}{N}$ has to be implemented in the iFFT. This scaling factor can be spread of over the 8 scaling positions $S_0 \ldots S_7$. The resulting output of the iFFT will then have the same fixed-point notation as the input of the FFT.

IV. DRM AS A CASE-STUDY FOR PERFORMANCE EVALUATION

This section describes the performance figures of the FFT-1920, which is required in DRM next to a variety of other FFTs. The smaller DRM FFTs have a shorter execution time and a lower energy consumption.

A. Configuration

Suppose the MONTIUM needs to be reconfigured to run an FFT-1920. This generic configuration requires three phases. First, we configure the configuration registers (2982 bytes), then we initialize some of the register files (44 bytes) and the last step is to write the twiddle factors into the local memory (904 bytes). These three steps have to be performed once before the algorithm can be started. The configuration size depends on the actual algorithm settings. However, to support partial reconfiguration, a generic configuration is used that is about 10% larger than a specific configurations.

TABLE II

NUMBER OF BYTES REQUIRED FOR PARTIAL RECONFIGURATION OF THE FFT-1920

Partial reconfiguration	Size
Enabling / disabling input scaling	2
Adjustment of the input scaling factor (S_0)	8
Adjustment of the FFT-128 scaling factors ($S_{1\ldots7}$)	14
Change from FFT to iFFT (or visa-versa)	8

The generic configuration can be tuned, for example, to change from FFT to iFFT, or to change the scaling factor. Table II gives an estimation of the number of bytes needed for tuning the configuration.

B. Performance

The execution of an FFT-1920 can be separated in several phases. Table III gives an overview of the steps that have to be taken in each phase (initialization, pre-processing, processing and post-processing). The loading of the input and input scaling is handled concurrently. The same holds for the output ordering and retrieving of the output.

Using the figures of Table III, we can calculate the execution time of an FFT-1920 on the MONTIUM. When its runs at 50 MHz, the calculation of one single block of data requires 398 μs. The optimized implementation on a 32-bit ARM9 RISC processor requires 4958 μs with a clock frequency of 96 MHz [5], which is a factor 12.5 slower compared to the MONTIUM. Both implementations perform exact the same operation.

C. Energy consumption

From the number of clock cycles, we can derive the energy consumption of the FFT-1920. For the MONTIUM the worst-case energy consumption is estimated at 0.577 mW/MHz for an synthesized design with a frequency of 100 MHz [9]. Based on the number of clock cycles the energy consumption for a single FFT-1920 equals 11.5 μJ where 3.4 μJ is consumed by the input and output ordering. This shows a energy saving of a factor 10 compared to the ARM9 implementation [5], which consumed 119 μJ excluding external RAM.

D. Accuracy

To test the accuracy of the algorithm we created a number of FFT-1920 cases. In these cases (see Table IV) we changed

TABLE III

OVERVIEW OF REQUIRED CLOCK CYCLES OF THE PHASES

Phase	Operation	Clock cycles
Initialization	Configuration	2113
Pre-processing	Load input	1922
	Input scaling	
	Input ordering	2114
Processing	FFT execution	14098
Post-processing	Output ordering	1927
	Retrieve output	
Total processing		19911

TABLE IV

11 CASES TO TEST THE ACCURACY OF THE FFT-1920

Case	Scaling factors							
	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
1	1	2	2	2	2	2	2	2
2	2	1	2	2	2	2	2	2
3	2	2	2	2	2	2	2	1
4	4	1	1	2	2	2	2	2
5	4	2	2	1	2	1	2	2
6	4	2	2	2	2	2	1	1
7	8	1	1	1	2	2	2	2
8	8	2	1	2	1	2	1	2
9	8	2	2	2	2	1	1	1
10	16	1	1	1	1	2	2	2
11	16	2	2	2	1	1	1	1

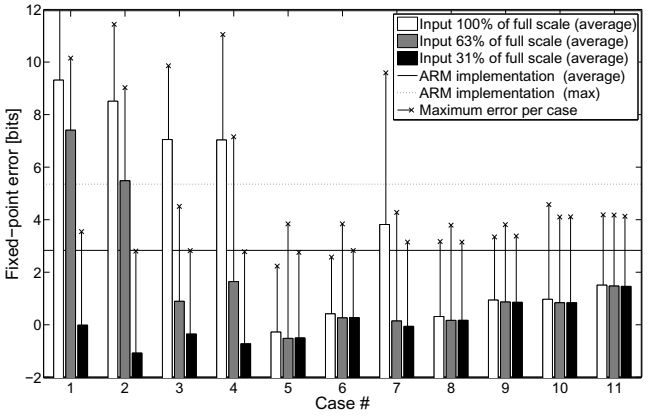

Fig. 2. Rounding errors for various scaling combinations

the scaling factors $S_0 \ldots S_7$. In all cases, the overall scaling factor S is 128.

As an input we used a typical DRM sample stream with its maximum amplitude related to the maximum value of full-scale. The MONTIUM output values are compared with a floating-point FFT calculated in Matlab, scaled with the same scaling factor of 128.

Figure 2 depicts the maximum and average absolute error for all 11 cases and for 3 levels of the input signal. As a reference, the error in the ARM implementation is included. This figure shows that the accuracy is higher when scaling is applied at the end of the FFT. However, if the input signal is too strong, the risk of overflow is higher. Overflow is noticed if the maximum error is above the 4.5 bits. In this figure, case 5 is the optimal trade-off between accuracy and risk of overflow.

V. DISCUSSION

The MONTIUM is very well suitable to execute algorithms with a regular kernel operation. Looking at algorithms like the FFT, it is clear that the kernel operation (a butterfly) is done repeatedly. The memory bandwidth required for executing several butterfly operations in parallel can be provided by the MONTIUM.

The non-power-of-two FFT is a less regular algorithm. By optimizing the algorithm for regularity and not for the number of multiplications we managed to map an irregular algorithm on the MONTIUM. The possibility to use the data path while generating addresses makes it possible to map almost any algorithm with less regular operations and addressing patterns to the MONTIUM.

VI. CONCLUSION

With the tooling currently available, it is relatively easy to reshape an algorithm to its parameterizable counterpart, as we showed with the non-power-of-two FFTs. Adding additional functionality to an existing algorithm might not be trivial; with only limited configuration resources available it can be hard to fit in new functionality, while leaving the existing functionality untouched. Again, tooling can be a good helper to find optimal solutions.

After adding the input scaling, input ordering and output ordering the MONTIUM is fully utilized. This implies both the physical usage (e.g. the bandwidth provided by the memories, interconnections and the ALUs) and the logical usage (e.g. the amount of instructions stored in the configuration space).

When looking at the performance by means of accuracy and energy consumption, the MONTIUM outperforms the reference ARM9 implementation by a factor of 10. This shows that computational intensive and less regular algorithms can be offloaded from the general purpose processing tiles to coarse-grained reconfigurable processing tiles.

By choosing smart scaling factors $S_0 \ldots S_7$ the intermediate results are stored as accurate as possible, while the computations are still done with complex 16-bit floating point samples. With partial reconfiguration the programmer can make the trade-off between accuracy and the risk of overflow.

ACKNOWLEDGEMENT

This research is conducted within the Smart Chips for Smart Surroundings project (IST-001908) supported by the Sixth Framework Programme of the European Community.

REFERENCES

[1] I. J. Good, "The interaction algorithm and practical fourier series," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 20, no. 2, pp. 361–372, 1958.

[2] S. Winograd, "On computing the discrete fourier transform," *Mathematics of Computations*, vol. 32, pp. 175–199, January 1978.

[3] G. Bi and Y. Q. Chen, "Fast dft algorithms for length n = q * 2^m," *IEEE Transactions on Circuits and Systems II*, vol. 45, no. 6, pp. 685–690, June 1998.

[4] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, "A new radix-2/8 fft algorithm for length-q x 2^m dfts," *IEEE Transactions on Circuits and Systems I*, vol. 51, no. 9, pp. 1723–1732, September 2004.

[5] A. Rivaton, J. Quevremont, Q. Zhang, P. T. Wolkotte, and G. J. M. Smit, "Implementing non power-of-two ffts on coarse-grain reconfigurable architectures," in *Proceedings of the International Symposium on System-on-Chip (SoC 2005)*. IEEE, November 2005, pp. 74–77.

[6] *Mixed-Radix 'Dual Speed' FFT Product Specification*, RF Engines Ltd, April 2004.

[7] "http://www.smart-chips.net."

[8] *Digital Radio Mondiale (DRM); System Specification*, ETSI ES 201 980 ed., European Telecommunication Standard Institute (ETSI), Sophia Antipolis, France, April 2003.

[9] P. M. Heysters, "Coarse-grained reconfigurable processors (flexibility meets efficiency)," Ph.D. dissertation, University of Twente, Enschede, The Netherlands, September 2004.

[10] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computations*, vol. 19, no. 90, pp. 297–301, April 1965.