Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2009, Article ID 678045, 12 pages
doi:10.1155/2009/678045

Research Article

Non-Power-of-Two FFTs: Exploring the Flexibility of the Montium TP

Marcel D. van de Burgwal, Pascal T. Wolkotte, and Gerard J. M. Smit

Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Drienerlolaan 5, 7522 NB Enschede, The Netherlands

Correspondence should be addressed to Marcel D. van de Burgwal, m.d.vandeburgwal@utwente.nl

Received 4 December 2008; Revised 2 April 2009; Accepted 9 July 2009

Recommended by Scott Hauck

Coarse-grain reconfigurable architectures, like the Montium TP, have proven to be a very successful approach for low-power and high-performance computation of regular digital signal processing algorithms. This paper presents the implementation of a class of non-power-of-two FFTs to discover the limitations and Flexibility of the Montium TP for less regular algorithms. A non-power-of-two FFT is less regular compared to a traditional power-of-two FFT. The results of the implementation show the processing time, accuracy, energy consumption and Flexibility of the implementation.

1. Introduction

More and more functionality is integrated in state-of-the-art mobile systems. One of the applications is digital broadcasting of audio and video. As the reception of digital broadcasts requires a considerable amount of digital processing, efficient architectures are required that can provide the necessary processing at a low energy budget.

Until recently, the processing resources of mobile systems were mainly provided by means of Application Specific ICs (ASICs). Such architectures have the advantage of low energy consumption, but do not provide flexibility. Therefore, for a multistandard system a multichip solution can be much more expensive than a highly integrated reconfigurable single-chip architecture. Therefore, heterogeneous reconfigurable multicore architectures are getting more and more attractive for multistandard appliances. Such an architecture consists of multiple processor types (heterogeneity) that can be used to execute applications within a bounded application domain. In order to support multiple emerging applications from this application domain, the processors can be reconfigured.

An example broadcast application is Digital Radio Mondiale (DRM) [1]. The DRM standard specifies the digitization of radio broadcasting in frequency bands below 30 MHz. DRM is the upcoming successor of AM radio and it is based on Orthogonal Frequency Division Multiplexing (OFDM) and MPEG-4 audio source coding. In the baseband processing of a DRM receiver several demodulation schemes have to be supported, each with their own characteristics and processing requirements [2].

The most challenging algorithms in the DRM application are the Fast Fourier Transform (FFT) and the Inverse Fast Fourier Transform (iFFT) required in the baseband processing of the OFDM receiver, as presented in [2]. Several of these FFTs/iFFTs have a length that is not a power-of-two and show a less regular implementation than the commonly used power-of-two FFTs. This paper presents the mapping of a class of non-power-of-two FFTs, in which the FFTs required for DRM can be included, at the Montium TP. The mapping is analyzed by means of performance, accuracy and energy consumption to demonstrate the flexibility of the Montium TP.

1.1. Overview. The paper is organized as follows. Section 2 introduces the non-power-of-two FFTs and their decomposition, such that they can be mapped on the Montium TP architecture. This architecture is presented in Section 3. More details on the decomposition, the mapping and

its accompanying and challenging difficulties are given in Section 4. The results, in particular processing time, accuracy and power consumption for each of the phases of the FFT, are presented in Section 5. Finally, in Section 6 some conclusions are drawn.

2. Non-Power of Two FFTs for DRM

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 input samples $x[n]$ and N output samples $X[k]$ (all complex numbers):

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

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, and so a direct computation of this formula requires $O(N^2)$ operations.

The FFT efficiently implements the DFT, by exploiting symmetry in its twiddle factors. A well-known FFT algorithm is the "divide and conquer" approach reintroduced by Cooley and Tukey [3]. This FFT algorithm recursively reexpresses a DFTs of length $N = N_1 \cdot N_2$ into smaller DFT of size N_1 and N_2. For an FFT with a length that is a power of x (called radix-x), the recursion can be done in $\log_x(N)$ stages using an x-inputs butterfly. Equation (2) shows how (1) can be rewritten into a radix-2 FFT:

$$\begin{aligned} X[k] &= \sum_{m=0}^{(N/2)-1} x[2m] W_N^{(2m)k} \\ &\quad + \sum_{m=0}^{(N/2)-1} x[2m+1] W_N^{(2m+1)k} \\ &= \sum_{m=0}^{(N/2)-1} x[2m] W_{N/2}^{mk} \\ &\quad + W_N^k \sum_{m=0}^{(N/2)-1} x[2m+1] W_{N/2}^{mk}, \end{aligned} \tag{2}$$

where $k = 0, 1, \ldots, N-1$.

The restriction of the radix FFT is that it can only handle FFTs that have a length that is a power of the radix value (e.g., two for radix-2). If other lengths are required a mixed-radix algorithm [4] can be used. For example an FFT-288 can be reexpressed with a radix-2 and radix-3 FFT (FFT-32 × FFT-9).

2.1. Good's Mapping. Another more efficient approach was introduced by Good [5], to eliminate the intermediate multiplications W_N^k required in the Cooley-Tukey approach. The algorithm is also known as the Prime Factor Algorithm (PFA). It makes use of Good's mapping to convert the 1-dimensional $N = N_1 \cdot N_2 \cdot \cdots \cdot N_L$ point DFT into a L-dimensional DFT equation. The lengths of the small DFTs have to be coprime and can be implemented with an arbitrary algorithm.

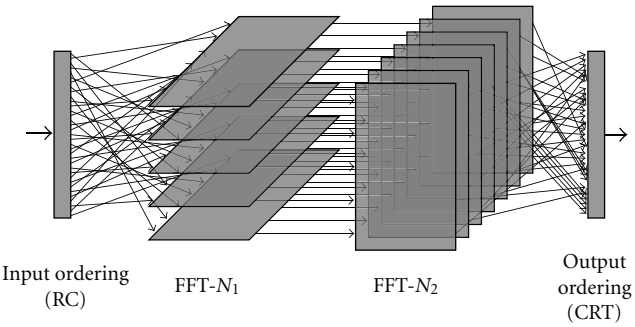


Figure 1: Steps in a PFA decomposed FFT.

Good's mapping optimizes the PFA for the number of calculations to be done, but assumes that input data is ordered in Ruritanian Correspondence (RC) order and output data in Chinese Remainder Theorem (CRT) order or vice versa, as presented by [6]. Assume that RC ordering is used for the input mapping in a two-dimensional array with the dimensions $N_1 \cdot N_2$. Then, the input vector $x[n]$ can be mapped on the array $x_{\mathrm{RC}}[n_1, n_2]$ as follows:

$$x_{\mathrm{RC}}[n_1, n_2] = x[n], \quad n = \langle N_1 \cdot n_2 + N_2 \cdot n_1 \rangle_N, \tag{3}$$

where $\langle a \rangle_N$ denotes a modulo N.

Because the RC mapping is used for the input vector, the CRT mapping has to be applied to retrieve the correct output vector. The output vector $X[k]$ can be mapped on the array $X_{\mathrm{CRT}}[k_1, k_2]$ where

$$X[k] = X_{\mathrm{CRT}}[\langle k \rangle_{N_1}, \langle k \rangle_{N_2}]. \tag{4}$$

A graphical description of the steps required for a PFA decomposed FFT using Good's mapping is given in Figure 1.

2.2. Performance versus Flexibility. The DRM receiver has to process the DRM audio samples at a certain rate, to avoid loss of synchronization, unwanted noise and clicks in the audio stream. Every 400 milliseconds, a DRM frame is transmitted. Depending on the transmission mode, such a frame consists of 15 up to 24 symbols. For each symbol, both the OFDM baseband processing and the audio source decoding have to be performed. Since the FFT is the most computational intensive task in the baseband processing, it should be executed efficiently and fast.

Depending on the channel quality, the transmission mode of the broadcast can change. If the receiver does not immediately adapt to the accompanying decoding scheme, data gets lost. Changing the coding scheme requires different lengths of FFTs and iFFTs. Therefore, the receiver should be very flexible.

During the last few decades, many efficient implementations of FFTs have been proposed. Often, the gain in computational performance obtained when optimizing the algorithm leads to irregularity in the algorithm. This requires the architecture to be flexible. Thus, performance and

flexibility are tightly linked, which seems to be in contrast with many architectures used nowadays.

2.3. Efficiency versus Accuracy. Generally, the amount of operations required to perform a DFT can be reduced by transforming it to smaller FFTs. However, for DFTs with a length that is a non-power-of-two, this transformation introduces some irregularity in the operations. This makes it hard to perform such an FFT efficiently.

A frequently used method to overcome this problem is "zero padding", which appends zeros to the input vector and increases its length to a power-of-two, such that a regular power-of-two FFT can be applied. However, this changes the filter response of the FFT and it will lose its orthogonal characteristics. To illustrate the effects of zero padding on OFDM systems, we simulated an OFDM system with QAM-16 modulation. Figure 2(a) shows the input bits after modulation by the transmitter. After transforming the input samples with an iFFT, it was sent via a channel which adds a small amount of Gaussian white noise. Two different receivers were used to transform the samples back to the frequency domain: one of them used an FFT with a length equal to the length of the sender's iFFT (see Figure 2(b)), while the other receiver used a larger FFT with zero padding (see Figure 2(c)).

The effect of the white noise added by the channel is clear: small errors occur in the received samples. However, for the zero padding based-receiver the input samples are not recognizable at all.

Usually, in most applications the error introduced by zero padding is acceptable as the gain in performance is more important. However, OFDM-based applications use the orthogonal characteristics of FFTs to improve the spectral efficiency and, therefore, the requirements for the FFT are more stringent. In order to obtain an acceptable performance, efficient non-power-of-two FFT implementations are required.

2.4. FFTs Required for DRM. For the DRM receiver, a large variety 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). The non-power-of-two FFTs mentioned before can be generalized to a group of 2-dimensional PFA-decomposable DFTs of the following form:

$$N = N_1 \cdot N_2 = (2p + 1) \cdot 2^q. \tag{5}$$

Table 1 gives an overview of the FFTs required for DRM (depicted underlined) as mentioned above and shows with which parameters p and q the PFA decomposition was done.

Strictly taken, a DFT-N_1 cannot be named FFT-N_1 as it is not as efficient as a true FFT. However, in this paper the term FFT is used to indicate that we use an optimized version of the DFT-N_1.

2.5. Related Work. 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 Digital Signal Processor (DSP) architectures. The algorithms for non-power-of-two FFTs are mainly focused on reducing the number of multiplications and additions, as, for example, discussed in [5, 7]. More recently a special class of non-power-of-two FFTs is further optimized to reduce the number of operations [8, 9]. Implementations of non-power-of-two FFTs are mainly described at the algorithm level, for example, [10].

Table 1: A selection of the FFTs that can be generated with the PFA mapping. FFTs used in DRM are underlined.

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

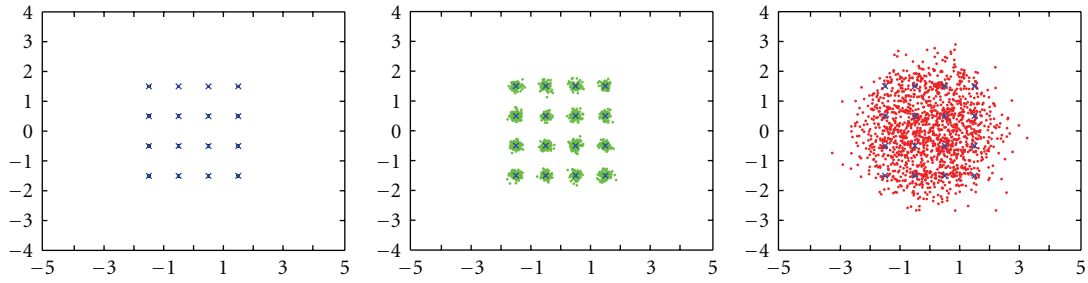
We have not found many articles that implement non-power-of-two FFT on a DSP. Several ASIC implementations were found, but due to outdated process technology used, the results are not very useful [11, 12]. A reconfigurable mixed-radix FFT processor is presented in a paper written by Jacobson et al. [13], who focus on the address generation and accuracy analysis. However, the design only supports FFTs of size 2^N using an efficient decomposition. In [14], a comparison of non-power-of-two FFTs on two coarse-grained reconfigurable architectures is presented. Those implementations are compared with an existing algorithm in software on a 32-bit ARM9 core.

A high-speed FFT-1872 implementation on an Field Programmable Gate Array (FPGA) is presented in [15]. This core is based on a mixed-radix DFT using the optimizations proposed by Winograd [7]. It supports the class of FFTs where $N = 2^p 3^q 13^r$, where all radix-2, radix-3 and radix-13 blocks are instantiated in parallel. Hence, the required area is very large and the design is relatively inflexible. Moreover, it does not support the FFT sizes required for DRM.

3. Montium TP Architecture

As described in Section 2.2, future architectures should provide both high performance and flexibility. With a continuously increasing number of transistors per chip, the processing capacity is also increasing. However, using all transistors efficiently gets more difficult. By designing multiprocessor architectures, the computational performance of such a chip can be increased considerably.

An example of multiprocessor architectures is a heterogeneous tiled System-on-Chip (SoC), that consists of several (possibly small) processors connected in a very regular Network-on-Chip (NoC) topology [16]. Figure 3 shows how such an architecture may be organized.



(a) Transmitted data (b) Received data (c) Received data (using zero padding)

Figure 2: QAM-16 bit errors occurring due to transmission and decoding.

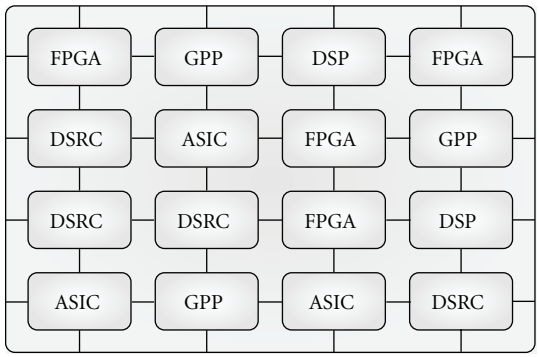


Figure 3: Heterogeneous tiled SoC.

In such a heterogeneous system, several types of processors can be combined: General Purpose Processor (GPP), DSPs, FPGAs, ASIC and Domain Specific Reconfigurable Core (DSRCs).

A typical GPP found in mobile devices is the Advanced RISC Machine (ARM) family (http://www.arm.com/). They can be employed for any arbitrary task, while their energy consumption is low. The general purpose design is useful for control intensive applications.

DSPs are designed for high performance and flexibility. Compared to a GPP, a DSPs performs much better within a bounded application domain, while its energy consumption is relatively low compared to the GPP.

FPGAs are bit-level reprogrammable. This allows a dedicated configuration for a specific application. Although operations on word level can be performed well on an FPGA, the infrastructure is more suited to bit-level operations. This can be seen when the performance and energy consumption are compared to the other architectures' figures: FPGAs provide a huge processing capacity, but at the cost of a relatively high energy consumption.

The ultimate processor for a certain task is the ASIC: once it is produced, it can only execute the application it was designed for. Therefore, the ASIC processor has both a high performance and low energy consumption, but is not flexible at all.

DSRCs are used to fill the gap between GPPs, DSPs, FPGAs and ASICs. Similarly to DSPs, their algorithm domain is bounded, but the computational performance is close to that of an ASIC.

An example of such a reconfigurable processor is the Montium tile processor [17, 18], which is developed by Recore Systems (http://www.recoresystems.com/). Flexibility and energy efficiency were considered to be the most important optimizations, while maximizing the performance for streaming algorithms in the digital signal processing domain.

Figure 4 shows a processing tile that employs a Montium TP. Such a tile consists of a Montium TP for processing and a Communication and Configuration Unit (CCU) that implements the Network Interface (NI) and controls the Montium TP. Computation of algorithms is done in the Processing Part Array (PPA), as shown in the top part of figure. This array consists of five identical processing parts, each connected to its neighbors for fast communication. Such a processing part consists of an Arithmetic Logic Unit (ALU) and two memory units to exploit locality of reference. The ALUs can perform operations that require up to four 16-bit inputs; each ALU has one multiply/accumulate and four function units that can perform additions, subtractions, shifting and all bitwise logic functions.

The ALU's input operands are fetched from a register file, which can store up to four values per input. A large crossbar connects all ALUs, memories and register files, providing a very high connectivity that is required to utilize the ALU's processing blocks as much as possible. Since there are 10 memories available which can be accessed simultaneously, the crossbar is based on 10 bidirectional 16-bit buses that can be read and written by each of the ALUs.

Each memory unit consists of an 1024×16-bit SRAM memory and an Address Generation Unit (AGU) that generates address patterns for this SRAM. Typical address patterns are increments and decrements of the address, masking and bit-reversing. When less regular address patterns are required, the AGUs can be fed with addresses generated by the ALUs or with addresses read from other memories.

The data path is controlled by a centralized sequencer, that contains an SRAM memory in which the instructions are stored. The program memory is not accessible via the datapath, hence it cannot be modified during execution. Within the sequencer, a program counter exists that addresses the instruction memory. Since there is a direct connection

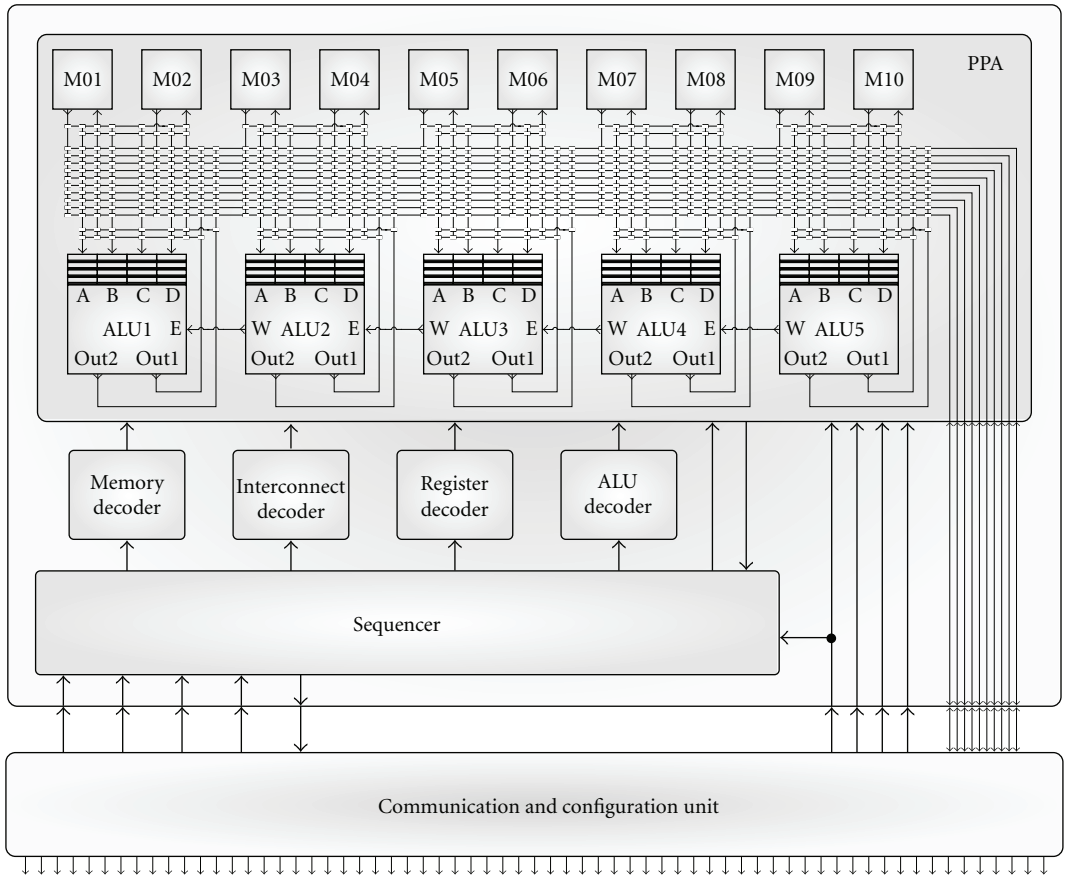


Figure 4: Montium processing tile, consisting of a Montium TP and a CCU.

Table 2: Characteristics of the Montium TP.

Word size	16 bits
Area	1.8 mm^2
Memory size	10×2 kB
Clock frequency	100 MHz
CMOS technology	0.13 μm TSMC
Voltage	1.2 V
Power	577 μW/MHz

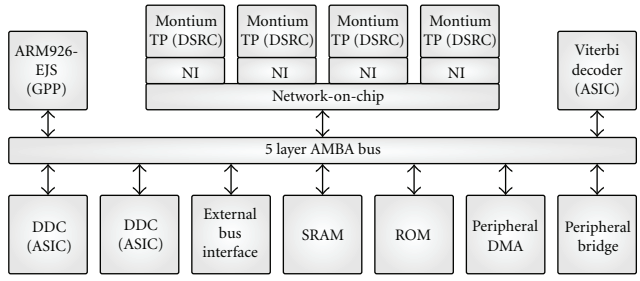


Figure 5: Annabelle SoC block diagram.

to the instruction memory, the fetch of an instruction only requires a single clock cycle. The selected instruction is decoded in two stages by decoders and configuration registers. Simultaneously, the program counter selects the next instruction in the instruction memory or, in case of a jump instruction, jumps to the address specified in the program. Since all instructions are single-cycle executions, the program behaves deterministic. Therefore, the instruction fetch can be considered to be transparent.

For each ALU, AGU, register file or interconnect component a small number of configurations (varying from 4 to 16 configurations) is stored in a local configuration register. The decoders contain combinations of these configurations (varying from 16 to 64 combinations) that are addressed by the sequencer. All configuration registers and decoders are implemented as asynchronous memories, which have to be filled prior to execution of a program. Therefore, the ALUs can be considered to be pipline-less, such that typical problems like pipeline stalls will never occur.

Table 2 summarizes the characteristics of the Montium TP.

An example of a SoC in which the Montium TP is used, is the Annabelle SoC [19], shown in Figure 5. The Annabelle SoC was developed within the 4S project and was built in ATMEL 130 nm process technology. It consists of four Montium TP s connected to a NoC via a NI (implemented

by the CCU), on-chip SRAM and ROM memories, hardware accelerators (Viterbi and DDC) and other peripherals. All devices are controlled by an ARM926 processor.

3.1. Program Control. The streaming nature of the Montium TP's algorithm domain is based on data-driven operations. Typically, operations are performed on chunks of data for several tens to thousands of iterations before the operation has to be changed.

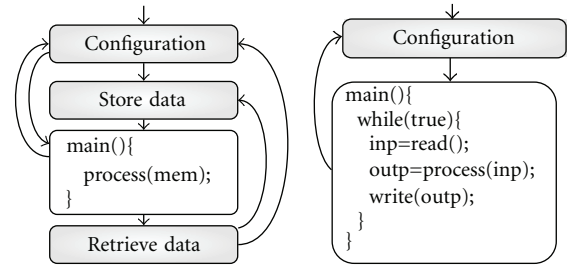
The Montium TP's instructions are programmed by filling the instruction memory, configuration memory and decoders. These memories can be modified by an external controller that takes care of the configuration and program control: the CCU [20].

Centralized in the CCU is a state register, which determines the Montium TP's current state. Possible states are the following: *configure* the configuration memories and instruction memories, *store* input data in the SRAMs, *retrieve* the results from these memories, *start*/*stop*/*pause* the configured computation, and *reset* the current state and store the current state for *debug*ging. For making a transition from one state to another, the CCU supports a message protocol that is used by any external control processor to write the new state into the state register.

3.2. Communication. The Montium TP can communicate via the CCU in two modes: *block mode* and *streaming mode*. In the block mode, the input samples are stored in the memories by means of a Direct Memory Access (DMA) transfer by the CCU. The execution of the configured algorithm is enabled by using a *start* command. When the execution has finished, the CCU retrieves the results from the memories with another DMA transfer, as can be seen in Figure 6(a). The streaming mode operation requires less explicit control overhead by the CCU. In this mode, the Montium TP sends communication requests to the CCU, which is depicted in Figure 6(b) by `read()` and `write()` functions. The CCU then executes these requests. Simultaneously, the Montium TP can continue its computation. This enables the overlap of communication and computation, avoiding costly wait cycles during the computation. Due to the data driven behavior of streaming mode applications, the communication requests can be embedded within the program itself such that communication and computation are automatically synchronized.

4. Implementation

Using the PFA decomposition (see (5)), any FFT required for DRM mentioned in Table 1 can be implemented. The non-power-of-two FFTs with $2 \leq p \leq 7$ and $N < 2048$ can be mapped on the Montium TP architecture in a similar way. This makes it possible to generate a large number of configurations based on the same decomposition and mapping structure. As an example of these FFTs, we use the FFT-1920 to describe the implementation of any non-power-of-two FFT.



(a) Block mode (b) Streaming mode

Figure 6: State transition diagrams for the two communication modes.

The FFT-1920 is partitioned using the parameters $N_1 = 2 \cdot 7 + 1 = 15$ 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 order of decomposition can be chosen arbitrarily, but the proposed decomposition leads to a better fit on the Montium TP architecture.

Figure 1 shows the steps that need to be taken in order to compute the FFT-1920. Firstly, the input data is distributed over the input memories (see Section 4.3). Secondly, the FFT-15 is then performed on 128 blocks of 15 input samples (see Section 4.1) and its results are distributed over the memory such that the FFT-128 can be operated. Thirdly, each of the 15 FFT-128s then processes a block of data (see Section 4.2) and writes its results in the memory. Finally, the output data from the FFT-128 is reordered (see Section 4.3).

4.1. FFT-15. Generally, the $N_1 = 2p + 1$ FFTs can be simplified by exploiting the symmetry in the twiddle factors. Because N_1 is odd, (1) can be rewritten to (6):

$$X[0] = \sum_{n=0}^{N_1-1} x[n],$$

$$X[k] = \begin{cases} T_{\mathrm{R}}[k] + T_{\mathrm{I}}[k], & 1 \leq k \leq \dfrac{N_1 - 1}{2}, \\ T_{\mathrm{R}}[N_1 - k], & \\ -T_{\mathrm{I}}[N_1 - k], & \dfrac{N_1 + 1}{2} \leq k < N_1, \end{cases} \tag{6}$$

where

$$T_{\mathrm{R}}[k] = x[0] + \sum_{n=1}^{(N_1-1)/2} (x[n] + x[N_1 - n]) \cdot \Re\left(W_{N_1}^{nk}\right),$$

$$T_{\mathrm{I}}[k] = \sum_{n=1}^{(N_1-1)/2} (x[n] - x[N_1 - n]) \cdot \Im\left(W_{N_1}^{nk}\right), \tag{7}$$

For the summations in T_{R} and T_{I}, the operands are multiplied with a twiddle factor. Two inputs are added before a multiplication and the butterfly structure of the Montium TP is used to calculate $X[k] = T_{\mathrm{R}}[k] + T_{\mathrm{I}}[k]$ and $X[N_1 - k] = T_{\mathrm{R}}[k] - T_{\mathrm{I}}[k]$ concurrently. Four ALUs are occupied to compute both outputs in parallel, while the fifth ALU is used to compute the $X[0]$ component simultaneously with the

computation of $X[1]$ and $X[14]$. For the calculation of $X[k]$, the values of $x[n]$ and $x[N_1 - n]$ are required simultaneously. Therefore, $x[0 \cdots (N_1 - 1)/2]$ and $x[(N_1 + 1)/2 \cdots N_1 - 1]$ are stored in different memories such that these values can be accessed simultaneously. These simultaneous calculations result in a reduction of the number of multiplications by a factor of 4.

In general, the number of clock cycles required to compute an odd-size FFT-N_1 on the Montium TP, using the partitioning presented above, equals $((N_1 - 1)/2)^2 + (N_1 - 1)/2 = 1/4(N_1^2 - 1)$. So, this approach is only viable for small N_1 as for larger odd values the complexity grows exponentially with the size of N_1. The execution of an FFT-15 based on this optimized implementation for the Montium TP requires 56 clock cycles. Theoretically, the FFT-15 could have been computed more efficiently by again applying the PFA with FFT-3 and FFT-5. Using the PFA, the FFT-3 has to be performed 5 times ($5 \times 2 = 10$ clock cycles), followed by 3 times the FFT-5 ($3 \times 6 = 18$ clock cycles), resulting in 28 clock cycles. However, this requires reordering of the intermediate results which cannot be implemented efficiently on the Montium TP architecture. The reordering costs for an FFT-15 would at least require another 15 cycles (for both input and output reordering), hence adding 30 cycles to the costs. Therefore, the presented mapping is the best alternative.

The optimization proposed in [7] also leads to an irregular algorithm. Although those optimizations would reduce the number of operations required to perform the FFT-15, its irregularity makes it hard to map the algorithm to the Montium TP. Moreover, the kernel operations done in the Winograd-optimized FFTs are different for each FFT, such that it is very hard to find a generic solution for generating all required non-power-of-two FFTs.

4.2. FFT-128. The FFT-128 is implemented using a standard radix-2 approach. Radix-2 algorithms can be calculated efficiently on the Montium TP, since one FFT-2 can be executed in a single clock cycle. A detailed explanation of the mapping is presented in [18, 21]. The computation of such an FFT-N_2 requires $(N_2/2 + 2) \cdot \log_2(N_2)$ clock cycles. Hence, the execution of an FFT-128 requires 462 clock cycles.

4.3. Input and Output Ordering. In traditional radix-2 FFT implementations the most difficult part is the bit-reversed addressing scheme of either the input or output values. In most DSP architectures, and Montium TP as well, special hardware in the AGUs overcomes this problem. However, in the PFA both input and output have to be ordered according the RC or CRT mapping. The input reordering in the Montium TP gives the user of the algorithm the possibility to stream in the data into the Montium TP in-order.

The address patterns for RC ordering cannot be generated efficiently with the AGUs. Moreover, since the input values for the FFT-15 are stored in two memories, address patterns become even less regular. A straight-forward solution for the ordering would be to use an indirection table of 1920 entries. However, there is not enough free memory space in the Montium TP for a table with this size. For smaller FFTs this is the preferred approach.

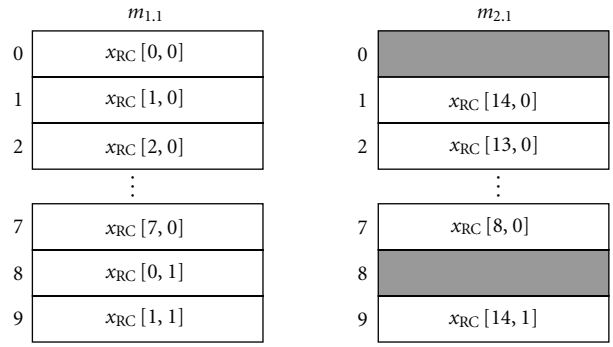


Figure 7: Memory organization before FFT-15 (RC order).

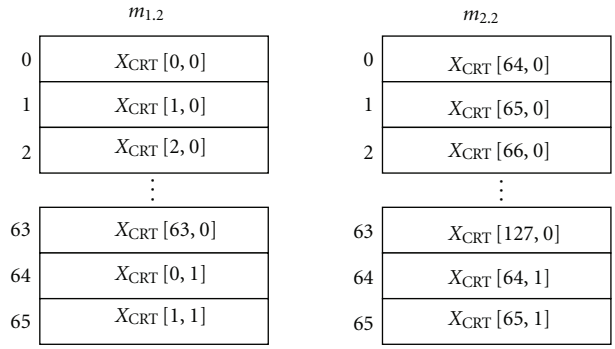


Figure 8: Memory organization after FFT-128 (CRT order).

For the input reordering for the FFT-1920 we use the following steps.

(1) The complex input vector is written in-order into 2 local memories $m_{1.1}$ and $m_{2.1}$.

(2) An indirection read address is calculated using (3).

(3) Using the indirection address, an input value is selected from the local memories $m_{1.1}$ and $m_{2.1}$.

(4) The value is stored in the other local memories $m_{1.2}$ and $m_{2.2}$ using the current write address.

(5) The write address in memories $m_{1.2}$ and $m_{2.2}$ are incremented by one.

Steps 2 to 5 are repeated 1920 times, until all values x_{RC} have been reordered. Figure 7 shows the FFT-15 input data after input reordering. Note that only the real part of x_{RC} is shown the imaginary part is stored in the memories $m_{3.1}$ and $m_{4.1}$ in an identical order and the 5 steps are taken simultaneously for the imaginary part.

Figure 8 shows how the results of the FFT-128 are stored in the memory. For the output ordering we use the same principle, but the selected complex values are streamed to the NoC via the CCU.

The most complex step in the ordering process of the outputs is the calculation of the indirection address. This address has to be calculated using modulo operations. In the

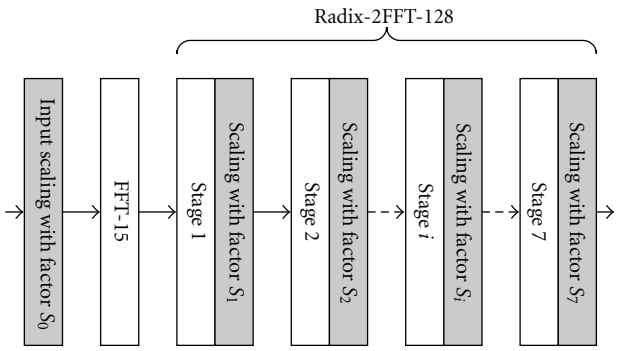


Figure 9: Positions in the FFT-1920 algorithm where scaling can be applied.

Table 3: Implementation costs of FFTs used in DRM.

FFT	N_1	N_2	clock cycles
112	7	16	472
176	11	16	960
224	7	32	1014
288	9	32	1450
352	11	32	1950
576	9	64	3116
1920	15	128	14098

appendix we explain the output ordering for streaming out the complex sample $X[k]$ in linear order.

4.4. Computational Complexity. The total number of clock cycles required to calculate a non-power-of-two FFT of length $N = N_1 \cdot N_2$ equals $N_1 \cdot (N_2/2 + 2) \cdot \log_2(N_2) + N_2 \cdot 1/4(N_1^2 - 1)$ plus a few clock cycles to initialize some of the registers. In Table 3 the complexity of the FFTs used in DRM is listed. For the FFT-1920 the exact numbers are given in Table 7.

4.5. Scaling. A fixed-point implementation of a digital signal processing 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 down. Scaling the intermediate fixed-point numbers results in a shift of the decimal point. Scaling a number in $(1, 15)$-fixed-point notation by 64 results in a number in $(7, 9)$-fixed-point notation.

In the FFTs considered in this paper, the signal is always scaled down with an integer factor S:

$$\text{Scaling} = \frac{1}{S}. \tag{8}$$

To improve readability, the scaling factors mentioned from this point indicate the denominator S as indicated above.

For an FFT the worst-case required scaling factor equals $\sqrt{2}N$, but in normal operation with a signal that contains several frequency components the scaling factor can be smaller. This results in a more accurate output signal. Therefore, we implemented a flexible solution, that supports scaling the signal at predefined positions. Figure 9 depicts these positions in the algorithm where scaling can be applied. Note that scaling does not necessarily have to be applied at once. Hence, multiple scaling positions can be used to obtain the total scaling by a factor S. The total scaling factor can be created using the following factors:

$$S = S_0 \cdot \prod_{i=1}^{m} S_i, \tag{9}$$

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 during stage i of the radix-2 FFT ($S_0 \in \{1, 2, \ldots, 2^{15} - 1\}$, $S_i \in \{1, 2\}$). It is up to the user of the algorithm to choose these values. Suppose we have a required total scaling factor of 128. The scaling can be positioned in the beginning ($S_0 = 128$), which results in a less accurate result and low risk of overflow. Moving scaling to the end of the algorithm will improve the accuracy but increases the risk of overflow. In any of the combinations the designer has to adjust the correct fixed-point notation of the output result.

4.6. Communication Mode. For both communication modes (block mode and streaming mode), we created a Montium TP implementation for our FFT. In the block mode, the input samples need to be ordered (in RC order as explained before) before they can be transferred to the memories of the Montium TP. When the FFT has finished, the results can be transferred from the memories and then need to be reordered. Both ordering steps have to be done outside the Montium TP.

The streaming mode version requires no external processing. Simultaneously with the input ordering, the input can be scaled. Input scaling is applied by multiplying the input stream with a factor $1/S_0$, as defined in (9). When the FFT computation is finished, the results are read in CRT order from the memories (as explained in Section 4.3) and written to the network via the CCU.

4.7. Run-Time FFT/iFFT Reconfiguration. The computation of the iFFT is almost similar to the computation of the FFT, as can be seen when comparing the FFT (1) with the iFFT (10):

$$x[n] = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X[k] W_N^{-nk}. \tag{10}$$

The main difference between the FFT and the iFFT is the $1/N$ scaling factor that has to be applied to the results and the usage of complex conjugated twiddle factors. The twiddle factors can be updated very easily, since they are located at a reserved position in the memory map of the Montium TP.

Additionally, for the $1/N$ scaling factor the designer has to compensate for the scaling applied in the FFT. If a scaling factor of S was used in the FFT then a scaling factor of S/N has to be implemented in the iFFT. Again, this scaling factor can be spread of over the 8 scaling positions $S_0 \cdots S_7$. The resulting output of the iFFT will then have the same fixed-point notation as the input of the FFT.

Table 4: ARM9 reference architecture.

Word size	32 bits
Area	4.7 mm^2
Clock frequency	96 MHz
CMOS Technology	0.13 μm TSMC
Voltage	1.2 V
Power	250 μW/MHz

Table 5: Number of bytes required for configuration of the FFT-1920.

	Streaming	Block
Configuration memory	2970	2438
Register files	44	8
Coefficient memory	904	452
Total	3918	2898

5. Performance Evaluation

As described in Section 2.4, DRM requires a large set of FFTs. This section describes the performance figures of the FFT-1920, which is the largest FFT that is required in DRM. The smaller FFTs will have smaller configuration, shorter execution time and lower energy consumption. Since the number of arithmetic operations in the smaller FFTs is less, the risk for overflow is smaller and therefore, scaling is less important. Therefore, in this section we will focus on the FFT-1920 to analyze its performance, accuracy and energy consumption.

As an indication of the performance and accuracy, we compared the Montium TP implementation with a 32-bit reference implementation for an ARM9 platform (see Table 4), which was used in [14].

5.1. Configuration. Suppose the Montium TP needs to be reconfigured to run an FFT-1920. This configuration requires three phases. First, we configure the configuration registers, then we initialize some of the register files and the last step is to write coefficients into the local memory. These three steps have to be performed once before the algorithm can be started. The configuration size depends on the actual algorithm settings and whether block mode or streaming mode is used. For streaming we use the most generic configuration, which enables very fast partial reconfiguration. This results in a configuration size as depicted in Table 5. If only one specific instantiation of the algorithm is required the configuration size can be reduced by 10% or more as shown by the block mode configuration size.

The main differences between the generic streaming mode configuration and the specialistic block mode configuration are (1) the extra input and output reordering of the samples and (2) the enabling of partial reconfiguration.

After the main configuration it is possible to adjust the generic streaming configuration with small modifications.

Table 6: Number of bytes required for partial reconfiguration of the FFT-1920.

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

Table 7: Comparison of required clock cycles between streaming mode and block mode.

Phase	Operation	Streaming	Block
Initialization	Configuration	2113	1871
Preprocessing	Load input	1922	1924
	Input scaling		964
	Input ordering	2114	N/A
Processing	FFT execution	14098	14098
Postprocessing	Output ordering	1927	N/A
	Retrieve output		1924
Total (w/o initialization)		20061	18910

For example, to change from FFT to iFFT or to change the scaling factor. These modifications can be done via partial reconfiguration and require a limited amount of bytes as given in Table 6.

5.2. Performance. The execution of a FFT-1920 can be separated in several phases. Table 7 gives an overview of the steps that have to be taken in each phase (initialization, preprocessing, processing and postprocessing) for both implementations (block mode and streaming mode FFT). For the streaming mode FFT the loading of the input and input scaling is handled concurrently. The same holds for the output ordering and retrieving of the output.

When adding up the processing time (initialization not considered), the block mode operation requires slightly less cycles than the streaming mode version. This is because the input and output ordering is not done in the block mode version; another processor or Montium TP has to take over this job. Obviously, it will be very hard for a GPP to efficiently reorder the data, due to the irregular address patterns (see also Section 4.3) and the GPP will require (much) more clock cycles to complete this processing.

Using the figures of Table 7 we can calculate the execution time of a streaming mode FFT-1920 on the Montium TP. When the clock frequency is 100 MHz, one FFT can be executed in 201 μs. The ARM9 implementation takes 4957 μs, which is about 25 times more.

5.3. Accuracy. To demonstrate the accuracy of the algorithm we executed the FFT-1920 with several combinations of the scaling factors (see Table 8, where the first factor, S_0, is the input scaling and next factors are intermediate scaling

Table 8: 11 cases to demonstrate 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

factors). The overall scaling factor S is 128 in all cases. The difference in the cases is the amount of scaling during the algorithm. For the lower numbers the scaling is put towards the end of the algorithm, which gives a higher accuracy. For the higher case numbers the risk of overflow is lower.

The input used for the test cases was a typical complex DRM sample stream consisting of 9600 samples. The sample stream was cut in 5 segments of 1920 samples and on each segment an FFT-1920 was applied. The amplitude of the stream was scaled to three levels (31%, 63% and 100% of the fixed-point scale) to analyze the effects of the input scaling and intermediate scaling. The results of the FFT computed by the Montium TP are compared with a floating-point FFT calculated by Matlab. For both, a total scaling factor of 128 was used.

Figure 10 depicts the maximum and average errors that occur in each of the cases and, per case, for 3 scaling levels of the input signal. The errors are calculated based on the error of all 1920 frequency bins, averaged over the 5 segments. By using a representation in terms of bits, the error compared to 16-bit accuracy is shown. Since parts of the internal datapath in the ALUs of the Montium TP are 18-bit wide, in an ideal situation the arithmetic operations can be performed more accurately than 16-bit. As a result, the error (in bits) can be negative, as shown in several of the cases in Figure 10. In the figure, the horizontal lines indicate the error of the ARM9 implementation. Note that the ARM9 implementation is 32-bit, while the Montium TP only operates in 16-bit mode.

From this figure it is clear that, for an input signal with 31% of the range, the low numbered cases have a higher accuracy. However, applying such input scaling decreases the dynamic range of the algorithm considerably, resulting in a less accurate Fourier transform. Therefore, input scaling should be avoided as much as possible. From Figure 10 we can conclude that cases 5, 6 and 8 perform best, independent of the input scaling. On the other hand, if input scaling is not applied and 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.

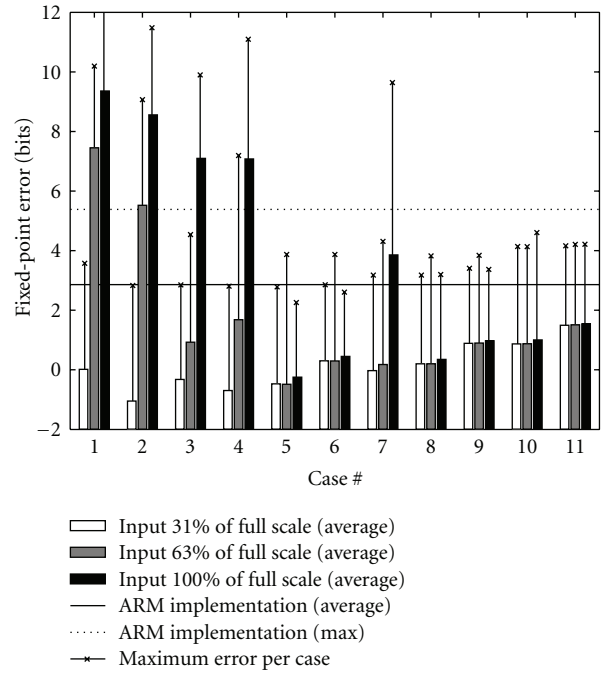


Figure 10: Rounding errors for various scaling combinations.

These results show the benefit for partial reconfiguration, where the system can quickly adjust the scaling factors depending on the input signal level. It can make a trade-off between accuracy the risk of an overflow.

5.4. Energy Consumption. From the number of clock cycles we can derive the power consumption of the FFT-1920. For the Montium TP the worst-case power consumption is estimated at 0.577 mW/MHz [18]. Based on the number of clock-cycles the power consumption for a single FFT-1920 equals 11.5 μJ where 3.4 μJ is consumed by the input and output ordering. The ARM implementation consumes 119 μJ, which is about 10 times as much as the Montium TP.

6. Conclusion

In this paper, the implementation of a wide range of non-power-of-two FFTs and iFFTs on the coarse-grain reconfigurable Montium TP architecture is discussed in detail. This range of FFTs showed to be an ideal test-case to explore and validate the flexibility of the coarse-grained architecture.

The Montium TP is very well suited for executing algorithms with a regular kernel operation. Due to the parallelism in the data path, it can perform up to 5 operations in parallel, while each operation can use up to 4 inputs. The memory bandwidth that is delivered by the 10 local memories is tremendous. For 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 TP, while the address patterns that are used for accessing the memories are generated quite easily.

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 a non-power-of-two FFT on the Montium TP. Using the Prime Factor decomposition, the class of non-power-of-two FFTs could be partitioned such that a radix-2 component was recognized (which can be mapped and executed very efficiently on the Montium TP) together with a small odd DFT that was manually mapped. The 10 memories available in the Montium TP enable parallel addressing of multiple inputs for a DFT, such that the DFT can be operated slightly more efficiently. Hence, the DFT's complexity was reduced from N^2 to $1/4(N^2-1)$. We showed that a further decomposition of the DFT is not desirable as the regularity of the decomposed algorithm decreases and performance will not increase. Hence, for the current Montium TP architecture, this is considered to be the best FFT mapping possible.

The possibility to use the data path for the generation of addresses makes it possible to map almost any algorithm with less regular addressing patterns to the Montium TP. Although this type of address pattern calculation is difficult, there is still enough regularity left to map the address calculation efficiently. Generic modulo-operations are difficult to implement in hardware; however, the (pseudo-) modulo operations required for address calculations can be implemented efficiently using the Compare/Select unit available in each ALUs in the Montium TP.

After adding the input scaling, input ordering and output ordering, the Montium TP's configuration space was almost 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).

Considering the performance regarding accuracy and energy consumption, the Montium TP outperforms the reference 32-bit ARM9 implementation by a factor of 10. By choosing smart scaling factors $S_0 \cdots S_7$ the intermediate results are stored as accurate as possible, while the computations are still done with complex 16-bit fixed point numbers. With partial reconfiguration the system can make the trade-off between accuracy and risk of overflow.

Appendix

Address Calculation

This example explains the output ordering for streaming out the complex sample $X[k]$ in linear order. The CRT mapping on the Montium TP's memories can be described as follows (only the real part of X_{CRT} is shown):

$$X_{\mathrm{CRT}}[k_1,k_2] = \begin{cases} m_{1.2}\left[\langle k_1\rangle_{N_2/2} + k_2\dfrac{N_2}{2}\right], & k_1 < \dfrac{N_2}{2}, \\ m_{2.2}\left[\langle k_1\rangle_{N_2/2} + k_2\dfrac{N_2}{2}\right], & k_1 \geq \dfrac{N_2}{2}, \end{cases} \tag{A.1}$$

When (4) and (A.1) are combined, the mapping of $X[k]$ can be described as follows (again only showing the real part):

$$X[k] = \begin{cases} m_{1.2}\left[\langle p[k]\rangle_{N/2}\right], & p[k] < \dfrac{N}{2}, \\ m_{2.2}\left[\langle p[k]\rangle_{N/2}\right], & p[k] \geq \dfrac{N}{2}, \end{cases} \tag{A.2}$$

where $p[k]$ indicates the indirection address that is calculated as follows:

$$p[k] = \left\langle 65\cdot k + 896\cdot\left\lfloor\frac{k}{64}\right\rfloor + (1024+(16-N_1)\cdot 64)\cdot\left\lfloor\frac{k}{15}\right\rfloor\right\rangle_N. \tag{A.3}$$

For example, to obtain the location of $X[3]$ from (A.2), one has to calculate $p[3]$ using (A.3). As $p[3] = 65\cdot 3 = 195 < 1920/2$, it is stored in $m_{1.2}$ at address position 195.

Equation (A.3) can also be written in a differential form:

$$c_1[k] = \begin{cases} 896, & \langle k\rangle_{64} = 0, \\ 0, & \langle k\rangle_{64} > 0, \end{cases}$$

$$c_2[k] = \begin{cases} 1024 + (16-N_1)\cdot 64, & \langle k\rangle_{15} = 0, \\ 0, & \langle k\rangle_{15} > 0, \end{cases} \tag{A.4}$$

$$\Delta[k] = p[k-1] + c_1[k] + c_2[k] + 65,$$

$$p[k] = \begin{cases} \Delta[k], & \Delta[k] < 1920, \\ \Delta[k] - 1920, & \Delta[k] \geq 1920, \end{cases}$$

One ALUs is used for determining the values of $c_1[k]$ and $c_2[k]$, which depend on the current value of k. The calculation of the new $p[k]$ requires a pseudo-modulo operation. As mentioned before, an ALUs has four inputs which can be used by four function units. Two function unit are used for the calculation of the two cases of $p[k]$. A third function unit performs the test $\Delta[k] < 1920$. Then, a so-called Compare/Select unit is used to select the result of one of the first two function units depending on the test performed by the third. All these operations can be executed on one ALUs in one clock cycle. By subtracting $\Delta[k] - 1920$, the sign is used to select the correct value for the output $p[k]$. This equals the operation $p[k] = \langle p[k-1] + c_1[k] + c_2[k] + 65\rangle_N$ because $p[k-1]$ and $c_1[k] + c_2[k]$ both are larger than 0 and smaller than 1920.

Acknowledgment

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

References

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

[2] P. T. Wolkotte, G. J. M. Smit, and L. T. Smit, "Partitioning of a DRM receiver," in *Proceedings of the 9th International OFDM-Workshop*, pp. 299–304, Dresden, Germany, September 2004.

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

[4] R. C. Singleton, "An algorithm for computing the mixed radix fast Fourier transform," *IEEE Transactions on Audio and Electroacoustics*, vol. 17, no. 2, pp. 93–103, 1969.

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

[6] C. Temperton, "Implementation of a self-sorting in-place prime factor FFT algorithm," *Journal of Computational Physics*, vol. 58, no. 3, pp. 283–299, 1985.

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

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

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

[10] T. Dettbarn and F. Mayer, "Using divide-andconquer on custom lengthened Fourier transforms," in *Proceedings of the International Conference on Consumer Electronics (ICCE '06)*, pp. 333–334, January 2006.

[11] R. W. Linderman, C. G. Shephard, K. Taylor, et al., "A 70-MHz 1.2-μm CMOS 16-point DFT processor," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 2, pp. 343–350, 1988.

[12] P. Lavoie, "A high-speed CMOS implementation of the winograd fourier transform algorithm," *IEEE Transactions on Signal Processing*, vol. 44, no. 8, pp. 2121–2126, 1996.

[13] A. T. Jacobson, D. N. Truong, and B. M. Baas, "The design of a reconfigurable continuous-flow mixed-radix FFT processor," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '09)*, May 2009.

[14] 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 '05)*, J. Nurmi, J. Takala, and T. D. Hamalainen, Eds., pp. 74–77, Piscataway, NJ, USA, November 2005.

[15] RF Engines Ltd., "Mixed-radix 'dual speed' FFT product specification," April 2004, http://www.rfel.com/download/D04022_Matrix_FFT_MR-DS_Product_Spec.pdf.

[16] G. J. M. Smit, A. B. J. Kokkeler, P. T. Wolkotte, P. K. F. Hölzenspies, M. D. van de Burgwal, and P. M. Heysters, "The chameleon architecture for streaming DSP applications," *EURASIP Journal on Embedded Systems*, vol. 2007, Article ID 78082, 10 pages, 2007.

[17] P. M. Heysters, G. J. M. Smit, and E. Molenkamp, "A flexible and energy-efficient coarse-grained reconfigurable architecture for mobile systems," *The Journal of Supercomputing*, vol. 26, no. 3, pp. 283–308, 2003.

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

[19] G. J. M. Smit, A. B. J. Kokkeler, P. T. Wolkotte, and M. D. van de Burgwal, "Multi-core architectures and streaming applications," in *Proceedings of the 10th International Workshop on System-Level Interconnect Prediction (SLIP '08)*, I. I. Mandoiu and A. A. Kennings, Eds., pp. 35–42, ACM, April 2008.

[20] M. D. van de Burgwal, G. J. M. Smit, G. K. Rauwerda, and P. M. Heysters, "Hydra: an energy-efficient and reconfigurable network interface," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems & Algorithms*, T. Plaks, Ed., pp. 171–177, CSREA Press, Las Vegas, Nev, USA, June 2006.

[21] P. M. Heysters and G. J. M. Smit, "Mapping of DSP algorithms on the Montium architecture," in *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, p. 180b, IEEE Computer Society, Washington, DC, USA, April 2003.