A 2048 COMPLEX POINT FFT PROCESSOR USING A NOVEL DATA SCALING APPROACH

Thomas Lenart and Viktor Öwall

CCCD, Department of Electroscience, Lund University
Box 118, SE-221 00 Lund, Sweden
Phone: +46 (0)46 222 91 05
Email: {thomas.lenart, viktor.owall}@es.lth.se

ABSTRACT

In this paper, a novel data scaling method for pipelined FFT processors is proposed. By using data scaling, the FFT processor can operate on a wide range of input signals without performance loss. Compared to existing block scaling methods, like implementations of Convergent Block Floating Point (CBFP), the memory requirements can be reduced while preserving the SNR. The FFT processor has been synthesized and sent for fabrication in a 0.35μm standard CMOS technology. In netlist simulations, the FFT processor is capable of calculating a 2048 complex point FFT or IFFT in 27μs with a maximum clock frequency of 76MHz.

1. INTRODUCTION

The Fast Fourier Transform (FFT) has a wide range of applications in digital signal processing [1]. For instance, FFT's are used in communication systems like DAB, DVB and IEEE 802.11a. The FFT is also used for analysing sound, images and video when removing undesired or perceptual irrelevant information, in radar applications as well as in different instrumentations. The N-point Discrete Fourier Transform (DFT) is defined as

$$X(n) = \sum_{k=0}^{N-1} x(k) W_N^{nk} \qquad n = 0,1\ldots,N-1 \tag{1}$$

where $W_N = e^{-j(2\pi/N)}$. The direct implementation of the DFT have a complexity of $O(N^2)$. Using the FFT, the complexity can be reduced to $O(N \cdot \log_2(N))$. The FFT is also more suitable for hardware implementation due to the physical regularity of the algorithm, but requires memory buffers to store parts of the sequence during calculation. The memory requirements are a crucial parameter since memories are rather expensive in both terms of area and power consumption. Reducing the memory requirement will therefore significantly affect the total size of the design. In this paper, the implementation of a high performance pipelined FFT processor with low memory requirements using a novel scaling approach is presented.

2. FFT ARCHITECTURES

There are many different ways to implement an FFT processor. The computations can be done in a number of iterations by time multiplexing a single memory and arithmetic unit, Fig. 1.a, or by using a pipelined architecture, Fig. 1.b. A pipelined radix-2 architecture requires $\log_2(N)$ arithmetic units, one for each butterfly stage, and is therefore more area expensive than using one single radix-2 unit. In return, the calculations will be $\log_2(N)$ times faster when using pipelining.

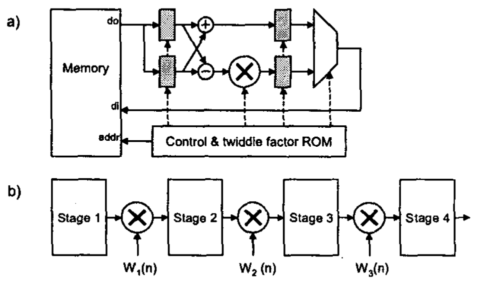

Fig. 1. a) Time multiplexed FFT processor using a single memory and butterfly unit. b) Pipelined FFT processor.

The problem with a fixed point FFT is to maintain the accuracy and preserve the dynamic range at the same time. One way to achieve a high signal-to-noise ratio is to increase the internal wordlength for every stage in the pipelined FFT (variable datapath), i.e. the wordlength will be wider at the output than at the input. Another way to improve the signal-to-noise ratio without increasing the internal wordlength is to use data scaling. One example of data scaling is *block floating point* that uses exponents, or scaling factors, for internal representation to improve the SNR. The exponents are usually shared between the real and imaginary part of a complex value, or even shared among a set of complex values [2-4], unlike normal floating point representation. Fig. 2 shows the almost constant internal wordlength when using data scaling and also the increasing internal wordlength when using a variable datapath. The width of the internal scale factor representation can be optimised for each stage. In this paper, focus will be on different scaling approaches.

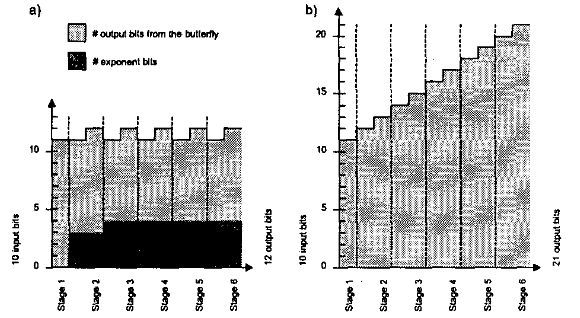

Fig. 2. Internal butterfly wordlength for a 10-bit 2048 complex point FFT processor using a) data scaling and b) variable datapath.

2.1 Block floating point

Time multiplexed FFT processors can use a data scaling method called Block Floating Point (BFP). After calculating all outputs from stage N, the largest output value can be detected and the intermediate result is scaled to improve the precision. When using BFP, all values share one single scale factor. BFP requires that the scale factor for stage N can be determined before starting the calculations of stage N+1. This approach cannot be applied to pipelined architectures due to the continuous dataflow.

2.2 Convergent block floating point

When a pipelined architecture is used, it is not efficient to wait until stage N has finished to determine the scaling factor. Instead a method called Convergent Block Floating Point (CBFP) has been proposed [2-4] as shown in Fig. 3. The basic idea is that the output from a radix-4 stage is a set of 4 independent groups that can use different scale factors. After the first stage there will be 4 groups, after the second stage 16 groups and so on. This will converge towards one exponent for each output sample from the FFT. The same scheme can be applied for a radix-2 stage, generating 2 independent groups at each stage. If the initial butterfly is of radix-2 type, most implementations omit the CBFP logic in the first stage due to the large memory overhead.

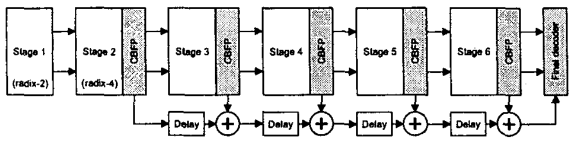

Fig. 3. A pipelined 2048-points FFT using CBFP.

The drawback with CBFP is that it requires a lot of memory. For the 2048 point pipelined FFT in Fig. 3, the input values are split into 2 groups of size 1024 in the initial radix-2 stage. The second stage produces 4 new groups containing 256 values each. These 256 values from the complex multiplier have to be stored in a buffer, as illustrated in Fig. 4, before the scaling factor for the group can be determined. Furthermore, it has to be saved in full precision because normalization cannot be done until the scaling factor is known. The length of the delay buffer after stage k is

$$l = 4^{\lceil \log_4(N) \rceil - k} \tag{2}$$

Another drawback is that the latency will increase, caused by the delay in the intermediate buffer.

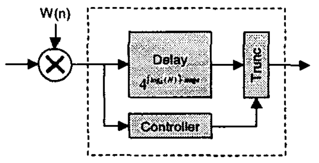

Fig. 4. CBFP logic between the consecutive stages.

2.3 Presented approach

Current CBFP techniques require a buffer to store N/4 outputs from a radix-4 stage with N inputs. The proposed method in this paper is to remove this buffer and rescale the data on the fly. After the complex multiplier, the result is normalized without delay and sent to the next butterfly stage. Therefore, each butterfly must be able to rescale one of the input values, if they are represented with different exponents. Starting from the second stage, the wordlength in the delay feedback has to be widened to hold both the complex value and the scale factor. This approach towards a hybrid floating-point processor still has much in common with the block scaling technique since only negative scale factors are allowed, leading to a reduced exponent representation and simple scaling logic. Furthermore, there is no input exponent, which reduces the memory overhead in the first (and largest) memory stage.

For further memory reduction, recall that each block in CBFP is represented with the same exponent. Accordingly, if the exponent in a block is only allowed to increase, the number of possible changes is limited. This will lead to minor performance degradation, but it will be shown that this limited number of changes can be stored in a special way to lower the memory requirements.

3. IMPLEMENTATION

The presented FFT processor is based on the radix-2^2 decimation-in-frequency algorithm [5]. The radix-2^2 algorithm is well suited for the presented approach because of the simple single-path delay feedback structure. Fig. 5 shows the modified FFT radix-2^2 processor, based on three different kinds of building blocks. The IBF unit is a normal butterfly that is only used in the first stage. The MUL unit contains a complex multiplier and a normalizing unit with proper rounding. The output value, $z(x)$, from a MUL block is represented by a complex value, $a(x)+jb(x)$, and a positive scale factor $s(x)$ as

$$z(x) = 2^{-s(x)}\left(a(x) + jb(x)\right) \tag{3}$$

The MBF unit is a radix-2^2 butterfly with a wider delay feedback to hold scale factors. To avoid problems with data alignment, equalizing units are used in conjunction with the butterfly units. Unlike CBFP, the large intermediate buffers between the FFT stages are not needed and replaced with the extra logic required to implement the equalizing units.

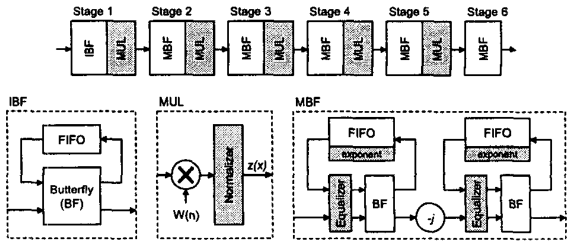

Fig. 5. The 2048 complex point FFT mainly consists of three different kinds of building blocks.

Compared to a floating-point implementation, no changes to the butterfly and complex multiplier units are required. The equalizer aligns the information to the butterfly, while the input to the complex multiplier is aligned by default.

3.1 Butterfly and complex multiplier

There are two butterfly stages in each radix-2^2 stage, calculating the sum and the difference between the input values and the output from the single-path delay feedback. When scale factors are used, it must be possible to align the inputs if they do not share the same exponent. An equalizer unit, only activated when the butterfly is not filling or draining the delay feedback, performs the alignment of input values to the butterfly stage. The equalizer compares the exponents of the two inputs to detect if the values are aligned or not. If there is a difference, the smallest input value is right shifted with the same number of bits as the difference between the two exponents. The aligned values are propagated to the butterfly unit. The output from the complex multipliers is normalized and sent to the next FFT stage. The normalizing unit is based on a number of compare and shift units connected in series. At the same time as the value is shifted by the normalizing unit, the exponent is incremented accordingly.

3.2 Delay feedback

The reordering method in a radix-2^2 FFT is the single-path delay feedback [5]. For shorter delays, several flip-flops can be connected in series. However, when the length of the FIFO increases, this approach is no longer area efficient. The large flip-flop cells can then be replaced with a single or dual port memory together with logic for control and address generation. Three different approaches to memory-based delays has been synthesized to a 0.35μm cell library for the Alcatel Microelectronics CMOS process and compared. The most straightforward approach is to use a dual port memory connected to an address generator. This allows simultaneous reads and writes to any memory location. One drawback with dual port memories is the required area, which is considerably larger than for single port memories. It is also possible that dual port memories are not always available, which makes the implementation process dependent.

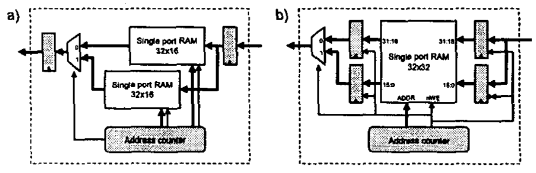

Fig. 6. a) Two single port memories. b) Single port memory with double wordlength.

One solution is to use two single port memories, alternating between reading and writing every clock cycle, Fig. 6.a. The drawback is the duplication of the address logic when using two memories instead of one. A third approach uses only one single port memory but with double wordlength, Fig. 6.b. This is possible, due to the consecutive addressing scheme used in the delay feedback. In addition to removing the duplicated address logic, the total number of memories for placement will be reduced. An area comparison of four different ways of building a delay feedback, or FIFO, is presented in Fig. 7.

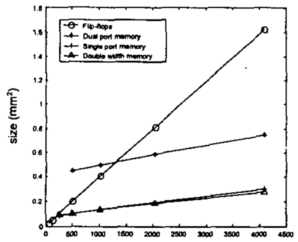

Fig. 7. Area requirements for different FIFO implementations.

Considerations have been taken to compare the actual size on the chip by adding an additional 50μm space for the power ring around the memories. When less than approximately 250 bits are required, flip-flops is the preferred method. In the presented design, single port memories with double wordlength have been used for the largest delay feedbacks. Flip-flops have only been used for the four shortest delays.

In order to remove the intermediate buffers between the stages, the exponents have to be saved in the delay feedback. When using floating point, two separate exponents are required for the real and imaginary parts. However, both CBFP and the presented approach represent a complex value with one shared exponent. Allocating space for only one exponent for each complex value reduces the memory requirements for the delay feedbacks.

3.3 Further memory reductions

The memory requirements can be reduced even further, by analysing the characteristics of block scaling. In the CBFP approach, the output from a radix-4 stage is a set of 4 independent groups, where each group share a single scale factor. The same principle can be applied to the current approach by introducing a Restricted Delay Line (RDL).

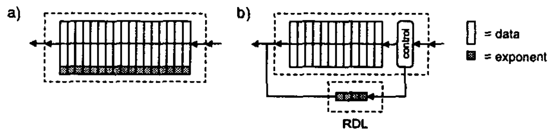

Fig. 8. a) FIFO with shared exponent. b) FIFO using RDL.

The basic idea is to only update the RDL with important scale factors, with a small performance penalty, instead of storing all scale factors in the delay feedback. A scale factor can be considered important if the current value is larger than the previous value and thus has to be stored, otherwise data is shifted corresponding to the scale factor most recently saved in the RDL. The maximum number of changes for a delay feedback of size N using a scale factor representation of K bits is 2^K, which is the minimal length of the RDL. The RDL is restarted periodically, initialised by storing the current scale factor. The stored value appears on the output after N cycles and is used for consecutive output values until a new scale factor is present. In addition to the performance penalty, the drawback with using a RDL is the additional logic required. Therefore it is only useful when building large sized FFT's. The RDL will however affect the size in a greater extent if the FFT processor is implemented so that it supports a scale factor input. In this case, the largest delay feedback can take advantage of the RDL, instead of storing scale factors the traditional way. For the current implementation, the largest delay feedback could be reduced by 15%, assuming a 4-bit input scale factor.

4. COMPLEXITY ANALYSIS

In this section, the presented approach will be compared with other implementations in terms of memory requirements, chip area and accuracy. An early version of the design has been presented at NORCHIP [6], and sent for fabrication in a standard 0.35μm CMOS process. The presented FFT processor is capable of performing a 2048 complex point FFT or IFFT in approximately 27μs, running at a maximum clock frequency of 76MHz. The FFT sent for fabrication is limited to 50MHz.

4.1 Memory and area requirements

For a 2048 complex point FFT processor, the memory occupies approximately 55% of the chip area, as can be seen in Diagram 1. Reducing the memory requirement will therefore significantly affect the total size of the design. A comparison between an FFT using variable data path, CBFP and the presented approach, all with 10 bit inputs has been made. Fig. 9 shows the number of memory elements required and it can be seen that the CBFP implementation requires substantially more memory than the other two, despite that CBFP logic has not been used in the initial stage. Fig. 10 shows the size of the total design, which follows the same trend. According to Diagram 1, the logic overhead in the presented approach, i.e. the equalizing units, does not have a large impact on the total chip area. Consequently, the area expensive intermediate buffers that are used in CBFP can be replaced with logic that requires less space.

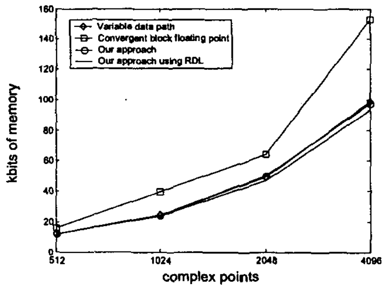

Fig. 9. Total number of memory elements required in the delay feedbacks for the different implementations.

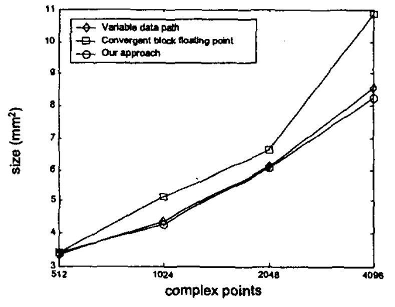

Fig. 10. Total chip size for the different implementations.

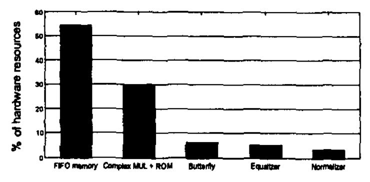

Diagram 1. Allocation of hardware resources.

4.2 Precision

When data scaling is used, the FFT processor can operate on a wide range of input signals. Even when the input signal has low amplitude, the signal will be scaled to full amplitude in the first stage, preserving the accuracy. The architectures described in section 4.1 have been simulated with various input signals including random noise, sine waves, OFDM and step response, all resulting in a higher SNR than for the CBFP. Fig. 11 shows the SNR for input signals starting with full dynamic range and then with down scaled input signal. The FFT with variable datapath produces a higher SNR when utilizing the full dynamic range. However, for arbitrary input signals, scaling is preferred.

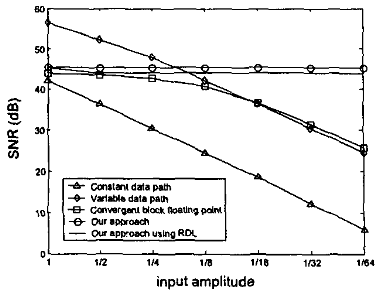

Fig. 11. Signal to noise ratio versus the amplitude of the input signal for different FFT implementations, all using 10-bit input wordlength.

As expected, the presented approach can maintain a high SNR even for down scaled input values. One of the reasons that CBFP is not capable of keeping a constant SNR is that there is usually no CBFP logic in the first radix-2 stage as shown in Fig. 3. The cost of adding CBFP logic to the first radix-2 stage is very high due to the large amount of intermediate buffer memory required. In the presented approach, the scaling part takes place in the subsequent pipeline stage. Hence, scaling is applied in all stages. If CBFP logic were added to the first stage, the corresponding SNR curve in Fig. 11 would remain constant as in our approach, but the memory requirements will be even higher than what is shown in Fig. 9.

5. CONCLUSION

An FFT processor using a novel data scaling approach that can operate on a wide range of input signals, keeping the SNR at a constant level, has been presented. Compared to block scaling approaches, such as CBFP, the proposed design requires significantly smaller chip area due to the reduced memory requirements. At the same time it is capable of producing a higher SNR since scaling is applied in all pipeline stages. The FFT processor has been sent for fabrication in a 0.35μm CMOS process.

6. REFERENCES

[1] E. Oran Brigham, *The fast Fourier transform and its applications*, Prentice-Hall, 1988.

[2] Se Ho Park *et.al.* "A 2048 complex point FFT architecture for digital audio broadcasting system", In *Proc. ISCAS* 2000.

[3] Se Ho Park *et.al.* "Sequential design of a 8192 complex point FFT in OFDM receiver", In *Proc. AP-ASIC*, 1999.

[4] E. Bidet, D. Castelain, C. Joanblanq and P. Senn, "A Fast Single-Chip Implementation of 8192 Complex Point FFT", *IEEE J. of Solid-State Circuits*, Vol. 30, NO. 3, 1995.

[5] Shousheng He, *Concurrent VLSI Architectures for DFT Computing and Algorithms for Multi-output Logic Decomposition*, PhD Thesis, Lund University, 1995.

[6] Thomas Lenart and Viktor Öwall, "A Pipelined FFT Processor using Data Scaling with Reduced Memory Requirements", In *Proc. NORCHIP*, 2002.