Parallel Extensions to Single-Path Delay-Feedback FFT Architectures

Brett W. Dickson, and Albert A. Conti

***Abstract*— Pipelined Fast Fourier Transform (FFT) architectures, which are efficient for long instances (32k points and greater), are critical for modern digital communication and radar systems. For long instances, Single-Path Delay-Feedback (SDF) FFT architectures minimize required memory, which can dominate circuit area and power dissipation. This paper presents a parallel Radix-2^2 SDF architecture capable of significantly increased pipelined throughput at no cost to required memory or operating frequency. A corresponding parallel coefficient generator is also presented. Resource utilization results and analysis are presented targeted for a 45nm silicon-on-insulator (SOI) application-specific integrated circuit (ASIC) process.**

***Index Terms*— FFT, high throughput, low-power, parallel, Radix-2^2, Single-Path Delay-Feedback**

I. Introduction

THE Fast Fourier Transform (FFT) is an efficient algorithm for computing the Discrete Fourier Transform (DFT) [1]. The FFT is a common digital signal processing function used across a multitude of application domains. Modern communication systems such as Orthogonal Frequency Division Multiplexing (OFDM) rely on the high-speed computation of the FFT. Radar systems also employ the FFT for matched filtering and Doppler processing.

Pipelined FFT processors compose a sub-class of architectures that are computationally efficient in hardware. These processors are capable of processing an uninterrupted stream of input data samples while producing a stream of output data samples at a matching rate. A variety of architectures for pipelined FFT processing have been proposed [5-9]. The desire for more precision, longer FFTs and increased power efficiency has motivated architectural innovations aimed at hardware reuse and the overall reduction in the number of adders, multipliers and words of memory required to implement FFT algorithms.

As a function of the number of stages in pipelined FFT architectures, the lower bound for butterfly and twiddle modules grows linearly while the lower bound for the number of words of memory grows exponentially [6]. For this reason, long FFTs can be dominated by memory with respect to resource utilization and power dissipation. Power dissipation can be further compounded when implementing in advanced silicon technology nodes. Excessive power dissipation will occur if the memory has not been optimized for leakage current since the active silicon area for the memory is proportional to its size. For these reasons, optimizing long FFT instances usually involves focusing on the minimization of required memory.

Single-Path Delay-Feedback FFT architectures have the most efficient memory utilization for pipelined FFT processors [4]. Due to the exponential growth of the number of memory words required with respect to the number of FFT stages (or the logarithmic growth of butterfly and twiddle modules required with respect to the number of FFT points), there will always be a point at which memory dominates circuit area and power dissipation. For this reason SDF FFT architectures are always optimal for long FFT instances [10, 11].

In this paper, we propose parallel extensions to the SDF FFT architecture [9] to significantly improve throughput without incurring an increase in memory or operating frequency. While many of these techniques are equally applicable to other SDF FFT architectures such as Radix-2 SDF [6] and Radix-4 SDF [7], discussion is focused on the challenges and tradeoffs associated with the Radix-2^2 SDF processing and twiddle generation.

The parallel extensions presented in this paper increase the number of butterflies and twiddle modules proportional to the parallelization while maintaining memory size of the FFT. This results in highly efficient throughput rates with a minimal increase in area and power for large FFTs.

This paper will present results demonstrating the area and power savings achieved by parallel extensions using a 45nm SOI process. The benefits in terms of area and energy efficiency become more apparent as the number of points in the FFT grows.

Applying the parallel extensions outlined in this paper allow for lower clock frequencies (inversely proportional to parallelization) and in the case of ASIC implementations, more leakage-efficient memories can be leveraged while maintaining pipelined throughput performance. Results will show that lowering the clock rate and increasing parallelism by the same factor does not change the throughput of the FFT processor and has a negligible impact to area for large FFTs.

This work was supported in part by The MITRE Corporation MSR Project: Emerging Technologies for VLSI Application (51MSR666-AA). Approved for Public Release; Distribution Unlimited. 13-3317

B. W. Dickson is with The MITRE Corporation, 202 Burlington Road, Bedford, MA 01730-1420 (phone: 781-271-2821; fax: 781-271-8915; e-mail: bdickson@mitre.org)

A. A. Conti is with Cognitive Electronics Inc., 201 South Street, Boston, MA 02111 (phone: 617-607-7199; fax: 888-209; e-mail: al@cognitive-electronics.com)

The remainder of this paper is structured as follows: in Section II the Radix-2^2 SDF FFT architecture is reviewed; Section III describes the proposed parallel architectural extensions to the SDF FFT; in Section IV the proposed architectural extensions are compared to previous approaches; finally Section V presents performance and utilization results targeted for a 45nm SOI ASIC. Final conclusions are drawn in Section VI.

II. Radix-2^2 SDF FFT Architecture

SDF FFT architectures make use of delay-lines implemented using memory and shift registers to reorder data at each butterfly stage. Delay-lines of length 2^m are required for all m from 0 to $\log_2(N) - 1$ where N is the number of FFT points the SDF FFT processor is capable of computing. This requirement is due to the data shuffling intrinsic to the decimate-in-time (DIT) and decimate-in-frequency (DIF) algorithms.

The Radix-2^2 SDF architecture is a hybrid of Radix-2 SDF and Radix-4 SDF designs [9]. The simplicity of the Radix-2 two-point butterfly structure is maintained while only needing $\log_4(N) - 2$ twiddle multiplies as is the case in Radix-4 architectures. This flexibility is achieved by using a second type of butterfly structure that performs $\pm j$ multiplications through sign inversion and real-imaginary sample swapping. This simplification eliminates half of the complex multipliers required for Radix-2 SDF implementations.

Fig. 1 shows Radix-2^2 SDF pipelines for both DIF and DIT implementations. DIF SDF architectures require a natural-ordered input stream to generate a bit-reversed output stream. Contrarily, DIT implementations expect bit-reversed input samples and produce natural-ordered output samples. This symmetry is often exploited in systems that transform data, perform processing in the frequency domain, and then apply an inverse transform. For large block sizes, incorporating additional memory buffers for data reordering are costly in area and power.

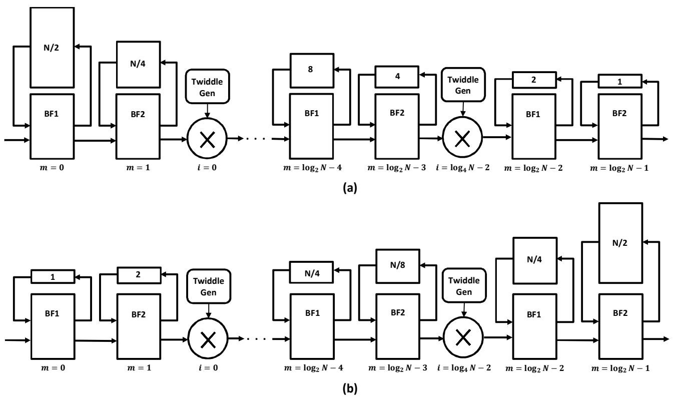

Fig. 1. (a) Radix-2^2 DIF SDF Pipeline, (b) Radix-2^2 DIT SDF Pipeline

A. Delay-Lines

From Fig. 1, where m corresponds to the butterfly index and i corresponds to the twiddle generator index, it can be seen the memory requirements at each butterfly stage differ between DIT and DIF implementations. The depth of a delay-line at a given butterfly stage is 2^m for DIT architectures and $2^{\log_2 N - 1 - m}$ for DIF architectures. The width of the memories is dependent on the bit width of the I and Q input samples and any internal bit growth maintained through the pipeline.

It should be noted that the total memory requirements may differ between the two algorithms even when computing the same number of FFT points with equivalent data widths. SDF FFT architectures may allow bit growth to occur at butterfly additions which requires growth in the widths of the delay-line memories through the pipeline. For DIF architectures, data widths increase linearly as delay-line memory depths decrease exponentially. This means that restraining bit growth in DIF FFT processors results in minimal savings as compared to the potential impacts of quantization. On the other hand, internal bit growth can have a significant effect for DIT FFT processors. In DIT implementations, delay-line memory bit widths will increase linearly while depths increase exponentially. If possible, samples should be quantized after butterfly additions to minimize memory in DIT pipelines.

Memories used to implement delay-lines for SDF FFT processors do not require random access. A straightforward sequential access scheme in which read and write pointers are simultaneously incremented for each pair of complex data samples requires a delay-line with a single dual-port static random-access memory (SRAM). For SRAMs with a single address port, two memories, each with one-half the number of required words, can be used with a similar scheme. Read and write address pointers will alternate between one memory instance and the other as they increment allowing memories to be written to and read from in ping-pong fashion. Some additional silicon overhead is involved when a single instance of memory is replaced by two of half the size, but this is minimal for large instances of memory. Fig. 2 depicts delay-lines implemented using both dual-port (a), and single-port (b) SRAMs.

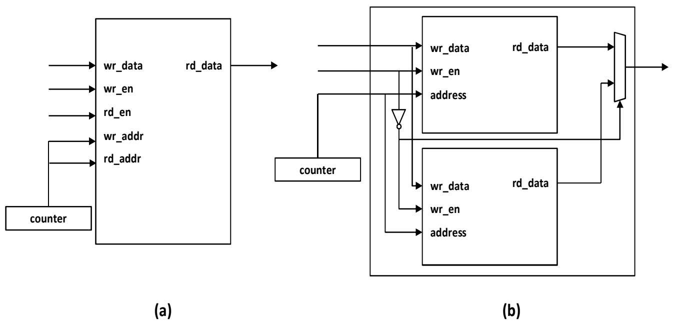

Fig. 2. (a) Delay-line implemented as dual-port RAM, (b) Delay-line implemented as two single-port ping-pong RAMs

B. Butterflies

Butterfly circuitry at each stage combines data which are $n/2$ samples apart where n is equal to 2^m for DIT and $2^{\log_2 N - 1 - m}$ for DIF architectures, and m is the incrementing butterfly stage number. What is important to note in the context of subsequent discussions is that contiguous data samples are not combined through processing until n equals 1. This remains true for butterfly circuitry with any radix. While butterflies with radices beyond two need to combine data samples from multiple delay-lines, this is restricted to non-contiguous data samples for all n greater than one.

In the Radix-2^2 SDF architecture, two unique types of butterfly structures are used (BF1 and BF2). The BF1 butterfly, which is identical to those used in Radix-2 SDF pipelines, computes a 2-point DFT. As previously stated, the depth of the delay-line (d) is a function of the number of points in the transform (N) and the stage number (m). Fig. 3 shows the BF1 structure from the proposed design where the EN_BF1_SUM level signal is negated every d cycles. During the first d samples when EN_BF1_SUM is low, multiplexors direct the input data to the feedback registers. On the next d cycles after EN_BF1_SUM is asserted high, the multiplexors are switched and the butterfly addition is performed between the input data and feedback output. This periodic process is continued until N samples have been processed.

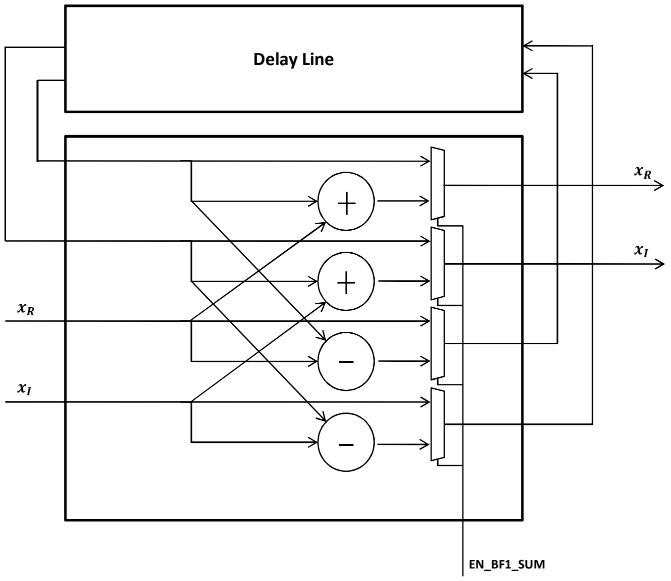

Fig. 3. Butterfly 1 (BF1) Architecture

The BF2 structure, shown in Fig. 4, has some added logic to perform a $\pm j$ multiplication without the need of a multiplier. Like the BF1 function, the BF2 directs the input to the feedback line for the first d cycles while EN_BF1_SUM and EN_BF2_SUM are both low. For the next $d/2$ samples, EN_BF1_SUM is active while EN_BF2_SUM remains zero. The result is the same as the summing state of the BF1 operation. For the final $d/2$ cycles, both EN_BF1_SUM and EN_BF2_SUM are high which causes in I and Q input samples to be swapped and the I sample to be negated (a multiply-by-j operation). Finally, EN_BF1_SUM and EN_BF2_SUM are both negated to return to the initial state. This routine is repeated until a full block of data (N samples) has been processed.

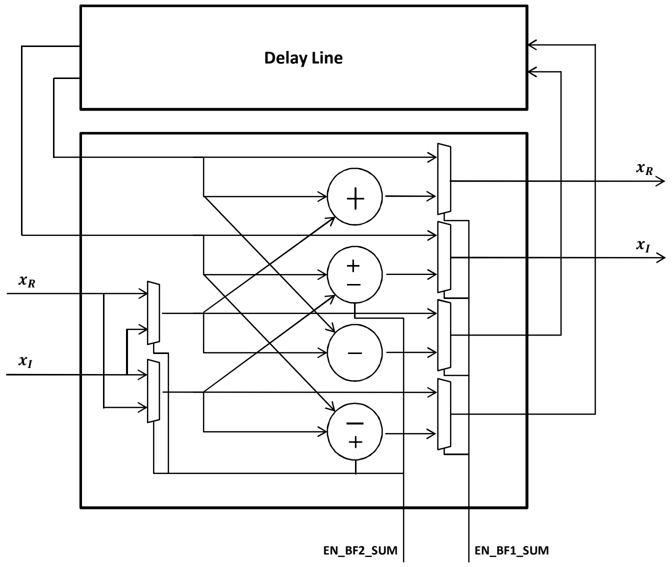

Fig. 4. Butterfly 2 (BF2) Architecture

C. Twiddle Generation

In the Radix-2^2 SDF architecture, a twiddle multiplication stage is implemented after every two butterfly stages. At every twiddle stage, a complex hardware multiplier is used to multiply each data sample by a corresponding complex twiddle coefficient of unit magnitude. The product is then truncated down to the bit width of the data stream before entering the subsequent butterfly stage. The algorithm used to generate the twiddle coefficients is as follows [12]. The twiddle factors at stage i where $0 \leq i \leq \log_4 N - 2$ is given by the set $W_i = \{W_N^k\}$ where:

$$W_N^k = e^{\frac{-j2\pi k}{N}}; \quad x = 0, 1, \dots, \frac{N}{2^{2i}} - 1; \quad c = 0, 1, 2, 3$$

$$k = \begin{cases} 0, & 0 \leq x < a \\ s_1 * (x - a), & a \leq x < 2a \\ s_2 * (x - 2a), & 2a \leq x < 3a \\ s_3 * (x - 3a), & 3a \leq x < 4a \end{cases} \tag{1}$$

$$s_c = \begin{cases} 0 \\ 2 * 4^i \\ 1 * 4^i \\ 3 * 4^i \end{cases} \tag{2}$$

and

$$a = \frac{N}{2^{2+2i}} \tag{3}$$

The equation shows that for any twiddle stage (i), there are four different states (n) which use a unique step size (s_n) to rotate the unit circle. The step size for each state is constant resulting in a linear progression around the unit circle. IFFT implementations use the same step sizes as forward transforms, however the coefficients traverse the unit circle in the opposite direction which requires a negation of the step size or phase increment.

The combination of simple butterfly processing and sequential access memories provides opportunities to exploit additional parallel processing per stage to increase overall throughput performance. The following sections outline methods for increasing performance through parallel hardware as well as the challenges and trade-offs associated with doing so.

III. Parallel Radix-2^2 SDF

This section will present novel modifications to the Radix-2^2 SDF pipeline (outlined in Section II) which allow for complex samples to be processed in parallel to increase the overall throughput of the processor. The extensions discussed for the proposed design presume that the parallelization factor (P), which corresponds to the number of data samples to be processed concurrently, will always be a power of two. Assuming a constant clock rate, the throughput of the FFT pipeline is directly proportional the number of parallel samples processed per second. Likewise, the latency required to process a full FFT block is inversely proportional to the parallelism.

Parallelization does not affect the critical path of the circuit; hence increasing P does not impact the maximum achievable clock frequency. Additionally, the memory requirements of the delay-lines in a SDF architecture are independent of parallelization. Because the control logic for each parallel butterfly is identical, the feedback outputs can be concatenated and written to one SRAM in a single transaction. As P grows, the depth of the delay-lines is reduced while the width is increased by the same factor. While the shape of the memories will change, changing P does not alter the aggregate number of bits that must be stored per delay-line.

One of the advantages of a parallel SDF FFT architecture is the ability to trade additional arithmetic hardware for lower operating frequencies or higher throughput. It provides system architects a larger design space and the power to tailor an FFT processor to best fit underlying implementation technology. This does not come without cost. There are many tradeoffs to consider. For example, all non-delay-line logic including butterflies, twiddle generators and complex multipliers must be duplicated for each additional sample to be processed in parallel. The remainder of this section will discuss the considerations that need to be taken into account when increasing the parallelization factor of the Radix-2^2 SDF pipeline.

A. No-Feedback Butterfly

To account for parallelization, the depth of each delay-line is decreased by a factor of P. For any P greater than one, there comes a point in the pipeline where the depth of the delay-line is less than one which indicates a traditional Radix-2 butterfly is no longer necessary. In this case, a third type of butterfly architecture is required. In this no-feedback butterfly (BF_NF) shown in Fig. 5, the delay-line of the conventional butterfly is abandoned. Instead of using a delay-line to align the operands of the adders, the BF_NF accepts two time delayed samples on the same clock period and generates two output samples. For inclusion in the Radix-2^2 SDF pipeline, the BF_NF must be able to mimic the operation of both the BF_1 and BF_2. Because there is no feedback state, the BF1 operation is performed when EN_BF2_SUM is low and the BF2 operation is executed once it is asserted.

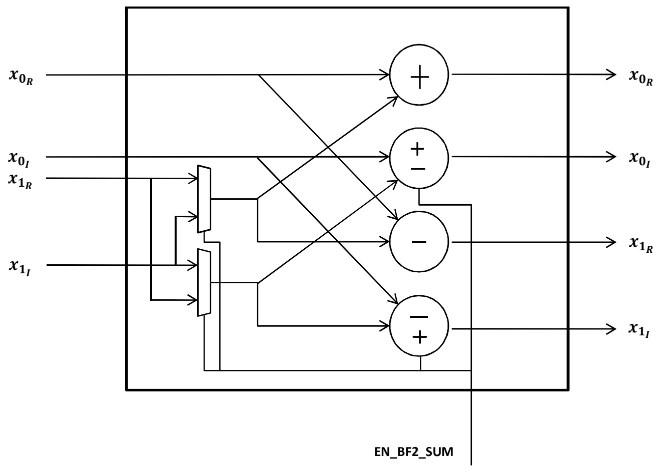

Fig. 5. No-Feedback Butterfly (BF_NF) Architecture

This new butterfly architecture is required when samples that are delayed in time are processed on the same clock edge. For a given parallelism, the number of non-feedback stages (butterfly stages that implement BF_NF) required in the pipeline is equal to $\log_2 P$. For DIF implementations, the non-feedback stages appear at the end of the pipeline whereas they show up at the beginning stages of DIT designs. Because each BF_NF processes two samples per clock, the number of BF_NF required per non-feedback stage is $P/2$.

Fig. 6 depicts a parallel-by-2 Radix-2^2 SDF DIF pipeline with the BF_NF butterfly at the final stage. For parallel implementations, the input data stream is broken up into P parallel streams each of which is defined by a unique identifier P_{idx} between 0 and $P-1$. Each parallel data stream ($x_{P_{idx}}$) is indexed using this identifier. In the case of Fig. 6, the upper half of the pipeline which processes the x_0 stream has a P_{idx} of 0, whereas the lower half a P_{idx} of 1. This paper will follow the convention that a lower P_{idx} value corresponds to an earlier sample in time.

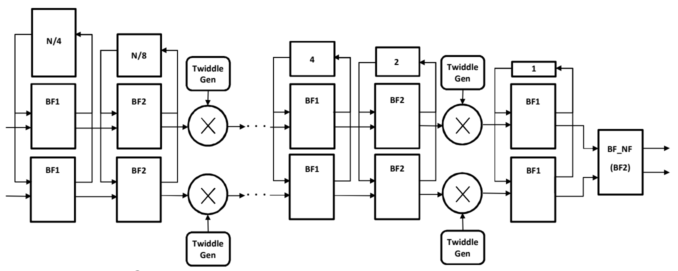

Fig. 6. Radix-2^2 SDF DIF Parallel-by-2 Architecture

B. Data Reordering

In parallel FFT architectures, it is necessary to reorder the data streams in the non-feedback butterfly stages. Since there are only $P/2$ BF_NF instances per non-feedback stage, a second identifier (nf_{idx}) is used to distinguish the index of a given non-feedback butterfly (BF_NF$_{nf_{idx}}$). Because multiple time-delayed samples are processed in the same pipeline stage, the processor must supply each BF_NF instance with the appropriate data streams to match the delay offset for that stage. The proposed design achieves this by indexing the data streams to the inputs of each BF_NF using the following formulas. For a given butterfly stage m where m is incremented from 0 to $\log_2 N-1$, the indices of the two input data streams (P_{idx_0} and P_{idx_1}) for a given nf_{idx} is as follows:

$$n=\begin{cases}\log_2 N-m & \text{for a DIF pipeline}\\ m & \text{for a DIT pipeline}\end{cases} \tag{4}$$

$$d \;=\; 2^n \tag{5}$$

$$P_{idx_0}=\begin{cases}nf_{idx} & \text{if } nf_{idx}<d\\ (2*nf_{idx})-mod(nf_{idx},d) & \text{if } nf_{idx}\geq d\end{cases} \tag{6}$$

$$P_{idx_1}=\begin{cases}nf_{idx}+d & \text{if } nf_{idx}<d\\ (2*nf_{idx})-mod(nf_{idx},d)+d & \text{if } nf_{idx}\geq d\end{cases} \tag{7}$$

Though other methods may exist for indexing the data streams, the important feature of (4)-(7) is that the inputs to each BF_NF are always offset d samples in time. Fig. 7 shows the last four stages of a parallel-by-8 Radix-2^2 DIF pipeline where the last three stages are implemented as non-feedback stages. Conversely, a DIT implementation would require the BF_NF instances at the beginning of the pipeline. The twiddle generators for the parallel pipeline, which are discussed in the following section, are not shown in Fig. 7.

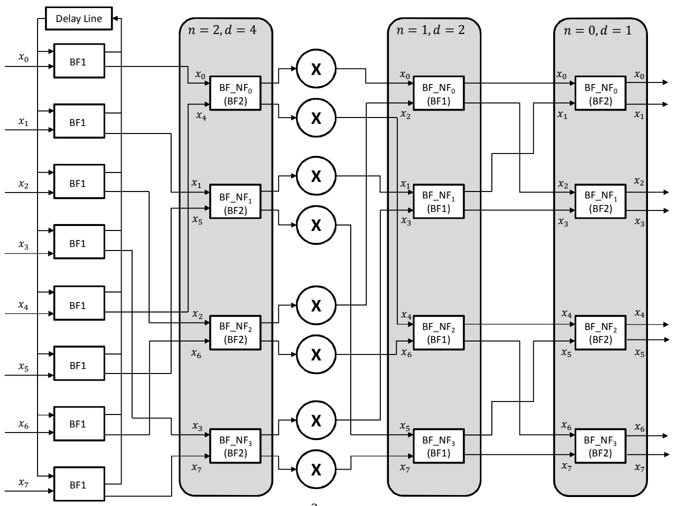

Fig. 7. Data Reordering for Radix-2^2 SDF DIF Parallel-by-8 configuration

As P increases, more and more of the Radix-2^2 stages will consist of BF_NF butterflies. It is interesting to note that in addition to duplicating hardware at each stage to support the processing of multiple samples per cycle, the architecture proposed also relies on a fusion of SDF and the more traditional signal flow graph (SFG) style FFT processing. The inclusion of the SFG processing stages is represented in Fig. 7.

C. Parallel Twiddle Generation

The formulas presented in (1)-(3) produce the sequential twiddle coefficients at a given twiddle stage. For a parallel implementation, each twiddle stage demands multiple coefficients per clock cycle. In the proposed design, P coefficient generators are required per twiddle stage, each of which produce a subset of the necessary twiddle factors at that stage. The following set of equations define the twiddle factors required at a given P_{idx}. The twiddle factors for the P_{idx} data stream at stage i where $0\leq i\leq \log_4 N-2$ is given by the set $W_i\;=\;\{W_N^k\}$ where:

$$W_N^k=e^{\frac{-j2\pi k}{N}};\quad x=0,1,\ldots,\frac{N}{P*2^{2i}}-1;\quad c=0,1,2,3$$

$$k=\begin{cases}0, & 0\leq x<a\\ s_1*(x-a)+\;(P_{idx}*s_1), & a\leq x<2a\\ s_2*(x-2a)+\;(P_{idx}*s_2), & 2a\leq x<3a\\ s_3*(x-3a)+\;(P_{idx}*s_{3)}, & 3a\leq x<4a\end{cases} \tag{8}$$

$$s_c=\begin{cases}0\\ 2*P*4^i\\ 1*P*4^i\\ 3*P*4^i\end{cases} \tag{9}$$

and

$$a=\frac{N}{P*2^{2+2i}} \tag{10}$$

For parallel twiddle generation, the number of twiddle factors produced per generator (x) is decremented by a factor of P while the step size (s_n) grows by the same factor. Additionally, an offset based on P_{idx} must be applied at each twiddle generator which corresponds to the $(P_{idx}*s_j)$ term when calculating k.

For pipelined FFT architectures, there are a variety of methods that can be used to generate the twiddle factors including ROM-based lookup tables, CORDIC functions and recursive multiplication. Similar to the memory requirements for the butterfly delay-lines, the number of twiddle factors required per stage grows exponentially as the number FFT points is increased. In efforts to reduce the memory requirements of long FFTs, the recursive multiplication approach was applied since the size of the circuit is independent of the number of twiddle factors that need to be generated. The recursive multiplier architecture implemented in the proposed design is shown in Fig. 8.

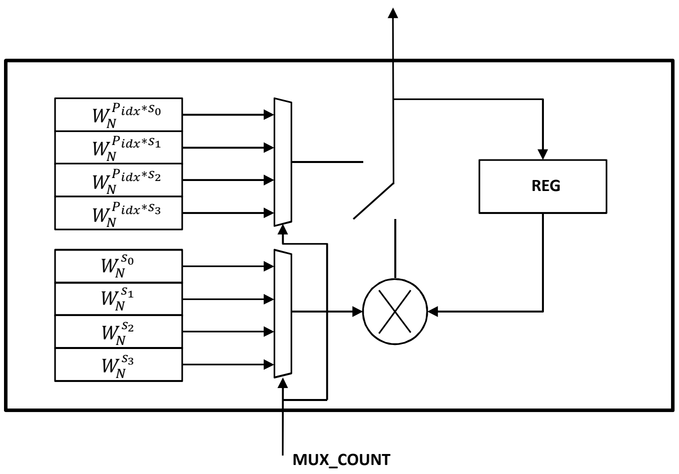

Fig. 8. Recursive twiddle generation architecture for parallel implementations

To implement recursive twiddle generation for a Radix-2^2 SDF architecture, a minimum of eight complex values must be calculated and stored. Each twiddle generator requires the offset ($W_N^{P_{idx}*S_c}$) and step size ($W_N^{S_c}$) for all four of the step size states, which correspond to c from (8)-(10). When the generator is started, the switch is connected to the output of the offset mux, and the offset is read out directly. The initial offset is also fed as an operand to a complex multiplier which multiplies the offset by the step size. After the first cycle, the switch is connected to the output of the complex multiplier which outputs the second twiddle coefficient. This result is then fed back to the multiplier to generate the next twiddle value. This process continues until a, from (10), coefficients are generated, after which MUX_COUNT is incremented and the process starts over for the next step size state.

In efforts to limit quantization error, the pre-computed offset and step sizes contain additional error bits which are carried throughout the computation and subsequently truncated at the output of the twiddle generator. The number of additional bits is a function of the maximum number of consecutive multiplies at a given twiddle stage. The rounding module used to truncate the output is not shown in Fig. 8. Alternative designs which account for pipelining of the multiplier are feasible; however additional offsets and step sizes must be pre-computed and stored.

IV. Prior Work

Parallel processing is inherent to pipelined FFT implementations. The ideas presented here focus on data-parallel processing on a per stage basis with the benefit of increasing throughput performance while sustaining optimal memory requirements.

Li and Meijs proposed a data-parallel SDF FFT architecture [3] which restructures the signal flow graph into even and odd sections. By separating data and then recombining in the final stage, the processing clock frequency can by reduced by a factor of two while maintaining throughput performance. This method increases control complexity and is not scalable beyond a factor of two without additional re-order buffers which would increase the required memory beyond the optimal level for pipelined implementations.

Ayinala, Brown, and Parhi proposed a data-parallel SDF architecture [2] which restructures the signal flow graph to reuse hardware based on the assumption that the input signal contains only real data. The architecture proposed is capable of processing two real data samples per clock cycle thus doubling the throughput performance but not the data rate or processing rate as compared to standard SDF FFT architectures.

The main distinction between the proposed design and prior work is the proposed design is scalable to any level of parallelism assuming sufficient resource availability. Each of the reviewed designs does not extend parallelism beyond a factor of two. The proposed design also offers a great deal of configuration flexibility. For example, the FFT length, transform type (FFT vs. IFFT), algorithm (DIF vs. DIT), data type (real vs. complex) and scheduling of internal bit growth are all programmable parameters that can be used to tailor the design to a desired application space and hardware platform.

V. Results

Numerous parallel configurations of the Radix-2^2 SDF DIT pipeline were synthesized to observe the effects of parallelism on the throughput, area and power dissipation of the circuit. The design was targeted for IBM’s 45nm silicon-on-insulator (SOI) ASIC process using a standard voltage threshold (SVT) cell library from ARM. The synthesis runs were completed using Design Compiler version E-2010.12-SP1 from Synopsys.

The various delay-line memories were provided by IBM as hard IP. In the proposed design, the delay-lines were architected as two single-port memories accessed in a ping-pong fashion as discussed in Section II.A. The IBM part numbers used in the proposed implementation were RF1CSN and SRAM1DCSN.

In many cases, the word length required by the delay-line exceeded the maximum allowable width of the IBM memories which was 288 bits. In such instances, multiple memories of equal depth were instantiated allowing the data word to span several memories at equivalent addresses.

As mentioned in Section III, as parallelism increases, the shapes of the delay-lines change but the number of bits do not. If one assumes the area per bit and power per bit to be constant, it would be expected that the power and area consumed by the delay-lines should remain constant for a given FFT configuration across multiple parallel implementations. However, this is not the case in an actual ASIC implementation. Arbitrarily sized memories are not always an option. Often, memories conforming to a subset of viable dimensions must be chosen from an IP vendor. In cases where the word width exceeds this threshold, multiple memory instances are required. In addition to the data array, each memory instance also contains control and decode logic that is replicated per instance. This can lead to higher power and area utilization for the same number of bits. This is especially apparent for larger values of P, where the memories are growing width-wise and shrinking depth-wise. Such oddly shaped memories require the concatenation of multiple conforming SRAM macros.

The remainder of this section will discuss how area and power are affected by changing the parallelism (and thus the throughput) of the pipeline. The experiments focused on larger FFTs where the delay-line memories dominate resource utilization.

A. Area

Even when accounting for different memory instance requirements for different parallel implementations, it is clear that the circuit area penalty for increasing parallelism is dominated by the FFT logic and not the delay-line memories. Fig. 9 shows a 64k-point DIT FFT synthesized at 250 MHz for five different values of P. The FFT maintains precision by growing a single bit at each butterfly. The chosen input data width is 18 bits resulting in a 34 bit output word. The "FFT Logic Area" refers to all circuit components that are not delay-lines including butterflies, complex multipliers, twiddle generators, counters and other control. "FFT Memory Area" includes all delay-line memory instances and their associated control.

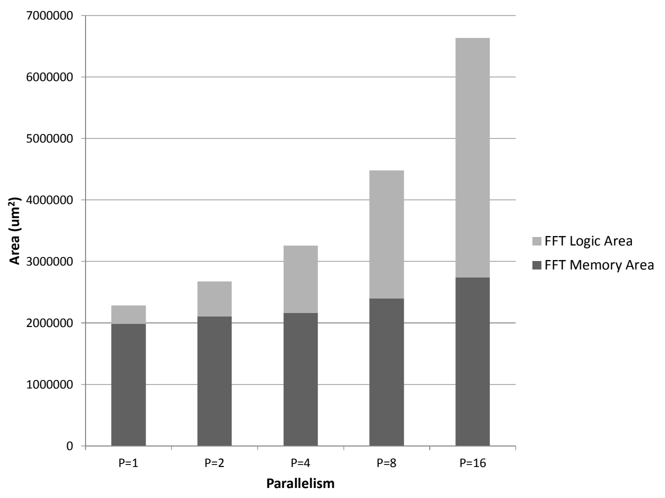

Fig. 9. Area (um^2) vs. Parallelism for 64k-point DIT FFT, 18 bit input, 34 bit output

What is obvious from Fig. 9 is that as P increases, the growth in FFT logic area is exponential. On the other hand, the increase in memory area as a result of concatenating memory instances is much less severe since the total number of memory bits has not changed as $f(P)$. For the $P = 1$ case, the throughput of the FFT processor is 250 Mega-Samples per second (MSps) while for $P = 16$, the throughput is 4 GSps. Alternatively, the area of the $P = 16$ circuit is only 2.9X (times) greater than that of $P = 1$. These results show that for the proposed parallelization techniques, a 16X increase in throughput only requires a 2.9X increase in area for this given FFT configuration.

To observe how parallelism affects different FFT sizes, synthesis experiments were conducted sweeping P across different FFT lengths. The circuit area was then divided by the throughput of the FFT to indicate a measure of area efficiency. The results can be seen in Fig. 10.

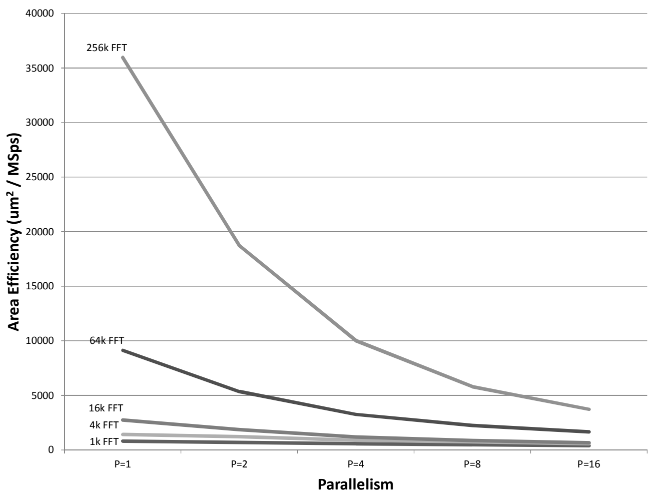

Fig. 10. Area Efficiency expressed in um^2/MSps as a function of Parallelism for various FFT lengths

While all of the FFTs improve area efficiency through parallelization, what is obvious from the curves is how larger FFTs improve their area efficiency at a much greater rate than smaller FFTs. This is due to the fact that for smaller FFTs, logic dominates area while memories dominate area in larger FFTs. For example, at $P = 1$, the 1k FFT requires 808 um^2 of silicon to process 1 MSps. However, when P is increased to 16, the area requirement to process 1 MSps drops to 392 um^2 resulting in a 2X improvement in area efficiency. The efficiency gain through parallelism is even greater for the 256k FFT. The $P = 1$ configuration requires roughly 36,000 um^2 of silicon to process 1 MSps while that number drops to just 3,700 um^2 at $P = 16$. In this case, the improvement in area efficiency is close to 10X.

The results from Fig. 10 show that in general, the area efficiency of the Radix-2^2 SDF pipeline increases as parallelism increases. It is evident that this characteristic is more pronounced for longer FFTs where delay-line memory requirements dominate area utilization.

B. Power

The synthesis experiments also provided insight into how power dissipation is distributed within the Radix-2^2 SDF pipeline and how that distribution is affected by parallelism. Fig. 11 shows the power dissipation of a 64k-point DIT FFT with full bit growth (18 bit input, 34 bit output). To attain dynamic power numbers, a global toggle rate of 53.2% was applied during synthesis. The toggle rate was determined by generating a Switching Activity Interchange Format (SAIF) file from actual simulation results which used random input data as stimulus to simulate a worst-case scenario for dynamic power. In all cases, a clock rate of 250 MHz was used.

The results show that the main driver of power dissipation in a Radix-2^2 SDF FFT is dynamic power from the FFT logic portion of the pipeline. For $P = 1$, the dynamic power dissipation from the FFT logic accounts for about half of the total power. However, this percentage grows as P increases. This is because the power dissipated by the FFT logic roughly doubles each time P doubles while the memory power increases at a slower rate since the number of memory bits remains constant.

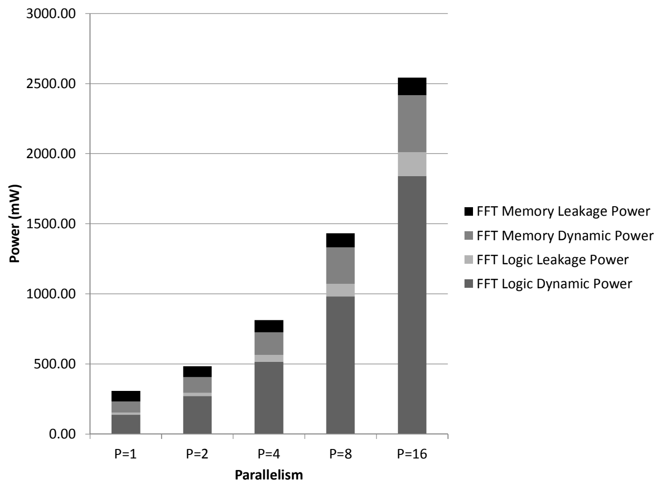

Fig. 11. Power (mW) vs. Parallelism for 64k-point DIT FFT, 18 bit input, 34 bit output at 250 MHz

Similar to the area analysis, energy efficiency was anlazyed across various FFT lengths to observe how power dissipation is affected by parallelism. To do this, different FFT sizes were synthesized at 250 MHz and power dissipation was recorded. Though increasing parallelism will increase the total power dissipated, higher parallelizations process samples at a higher rate. For example, at $P = 1$, the throughput will be 250 MSps while at $P = 16$ it will be 4 GSps. To calculate energy efficiency (J per sample), the power disspation (Watts) was divided by the FFT throughput (MSps) to determine how much energy is required to process each sample.

The energy efficiency curves are shown in Fig. 12. Naturally, larger FFTs will dissipate more power than smaller FFTs on a per sample basis since the hardware structure is much larger. On the other hand, energy efficiency can be improved through parallelism, and this behavior is more apparent in larger FFTs. For example, in the 1k FFT case, the $P = 1$ structure requires .35 J/MSample while the $P = 16$ structure requires only .2 J/MSample. This suggests that the engergy used to process each sample can be reduced by about 43% when going from $P = 1$ to $P = 16$. For the 256k FFT, increasing parallelism from $P = 1$ to $P = 16$ results in close to a 65% reduction in the amount energy used per sample. The improvement in energy efficiency can be attributed to the fact that larger FFTs are dominated by memory, and the power dissipated in the memories grows at a slower rate than does throughput as parallelism increases.

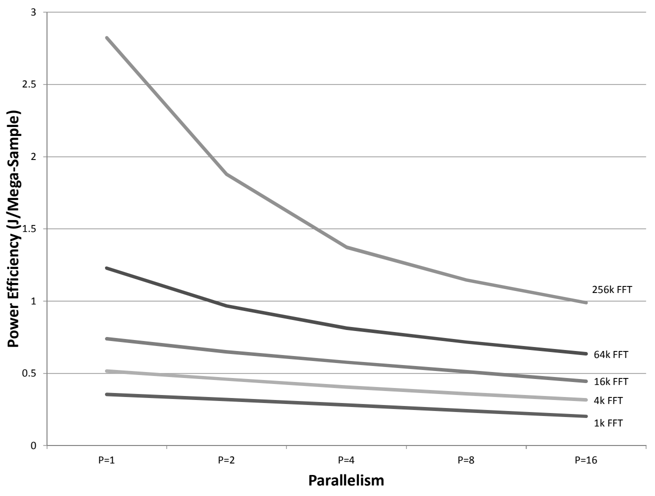

Fig. 12. Energy Efficiency (Joules per Mega-Sample) as a function of Parallelism for various FFT lengths at 250 MHz

C. *Core Clock Frequency*

An alternative application of parallelism is clock reduction to decrease dynamic power dissipation. If P is increased, the core clock frequency can be reduced by the same factor while maintaining the overall throughput of the system. However, there are area vs. power tradeoffs to consider when increasing the parallelization factor and reducing the clock rate.

For example, a 64k-point DIT FFT with 18 bit input and 34 bit output was synthesized at 400 MHz with $P = 1$ and also at 200 MHz with $P = 2$. Both configurations have equivalent throughputs of 400 MSps. In the $P = 1$ case, the resulting circuit required 2.30 mm^2 of area and dissipated 448 mW of power. On the other hand, the $P = 2$ configuration was 2.66 mm^2 and consumed 401 mW. In this case, doubling P and halving the clock frequency leads to a 15.5% area increase, however the total power dissipation decreased by 10.5%. The comparison of the two FFT instances used in this experiment can be seen in Table I.

If targeting a low-power design, it may make sense to incur the area penalty to save power by increasing parallelism. The power savings that can be achieved by increasing P and reducing the clock rate are more apparent at higher frequencies where the dynamic power dominates total power dissipation.

TABLE I: AREA AND POWER COMPARISON OF 400MSPS FFT CONFIGURATIONS

Parallelism	1	2
Clock Rate (MHz)	400	200
Throughput (MSps)	400	400
Area (mm^2)	2.3	2.66
Logic Area	0.31	0.55
Memory Area	1.99	2.11
Total Power (mW)	448	401
Logic Dynamic	230	211
Logic Leakage	13	23
Memory Dynamic	129	89
Memory Leakage	76	78

D. *ASIC Implementation*

Two versions of the proposed design were recently implemented as part of a pulse-compression radar application.

Both a 64k FFT and 64k IFFT, each with $P = 2$, were integrated as part of a signal processing chip targeted for IBM's 45nm SOI ASIC process. The FFT was computed using the DIF algorithm and maintained precision by allowing full bit growth. The IFFT used the DIT algorithm and contained some internal scaling logic in efforts to limit bit growth. The IC was sent for fabrication in August of 2011 and completed testing in 2012 at MITRE's VLSI Laboratory and IC test facility. Implementation details of the two FFT instances can be seen in Table II.

TABLE II: CONFIGURATION DETAILS OF FFT AND IFFT FOR 45 NM ASIC IMPLEMENTATION

	64k FFT	**64k IFFT**
Number of FFT points	65536	65536
Algorithm	DIF	DIT
Parallelism	2	2
Input Word Width (bits)	10	18
Output Word Width (bits)	26	30
Operating Frequency (MHz)	200	200
Maximum Throughput (MSps)	400	400
Power (mW)	192	365
Area (mm^2)	1.3	2.4

VI. CONCLUSION

This paper has proposed a set of extensions that can be used to apply parallelism to the Radix-2^2 SDF FFT pipeline. The proposed methods are flexible and allow for N-point FFT and IFFT computation such that N is a power of two. Additionally, both the DIF and the DIT algorithms are supported. Although the stated extensions apply specifically to the Radix-2^2 SDF algorithm, similar techniques could be used for all pipelined SDF FFT implementations. The proposed extensions impose no restrictions on the overall throughput of the FFT circuit given adequate resource availability.

Synthesis experiments were conducted to analyze how parallelization of the pipeline affects the size, throughput and power of the circuit. It was determined that there are significant benefits in terms of both area efficiency and energy efficiency when increasing the parallelism of the FFT. These benefits can be attributed to the fact that the memory requirements of the delay-lines remain approximately constant regardless of the parallelization factor.

REFERENCES

[1] J.W. Cooley and J. Tukey, “An algorithm for machine calculation of complex fourier series,” *Math. Comput.*, vol. 19, pp. 297-301, Apr. 1965.

[2] M. Ayinala, Michael Brown, and K.K. Parhi, "Parallel - pipelined radix-2^2 FFT architecture for real valued signals," *Signals, Systems and Computers* (ASILOMAR), 2010 Conference Record of the Forty Fourth. Asilomar Conference on , vol., no., pp.1274-1278, 7-10 Nov. 2010.

[3] Nuo Li and N.P. van der Meijs, "A radix 2^2 based parallel pipeline FFT processor for MB-OFDM UWB system," SOC Conference, 2009. SOCC 2009. IEEE International , vol., no., pp.383-386, 9-11 Sept. 2009.

[4] He Shousheng and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," *Signals, Systems, and Electronics*, 1998. ISSSE 98. 1998 URSI International Symposium on , vol., no., pp.257-262, 29 Sep-2 Oct 1998.

[5] L.R. Rabiner, B. Gold, and C.K. Yuen, "Theory and application of digital signal processing," *IEEE Trans. on Systems, Man and Cybernetics*, vol.8, no.2, pp.146, Feb. 1978.

[6] E.H. Wold and A.M. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementations," *IEEE Trans. on Computers*, vol.C-33, no.5, pp.414-426, May 1984.

[7] A.M. Despain,"Fourier transform computers using CORDIC iterations," *IEEE Trans. on Computers*, vol.C-23, no.10, pp. 993- 1001, Oct. 1974.

[8] G. Bi and E.V. Jones, "A pipelined FFT processor for word-sequential data," *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol.37, no.12, pp.1982-1985, Dec 1989.

[9] He Shousheng and M. Torkelson, "A new approach to pipeline FFT processor," *Parallel Processing Symposium*, 1996., Proceedings of IPPS '96, The 10th International , vol., no., pp.766-770, 15-19 Apr 1996.

[10] E.E. Swartzlander, W.K.W Young, and S.J. Joseph, "A radix 4 delay commutator for fast Fourier transform processor implementation," *IEEE Journal of Solid-State Circuits*, vol.19, no.5, pp. 702- 709, Oct 1984.

[11] E. Bidet, D. Castelain, C. Joanblanq, and Senn, P, "A fast single-chip implementation of 8192 complex point FFT," *IEEE Journal of Solid-State Circuits*, vol.30, no.3, pp.300-305, Mar 1995.

[12] A. Saeed, M. Elbably, G. Abdelfadeel and M. Eladawy. “FPGA implementation of Radix-22 Pipelined FFT Processor,” Proceedings of the 3rd international symposium on Wavelets theory and applications in applied mathematics, signal processing & modern science, Istanbul, Turkey, 2009.

[13] E. Bidet, D. Castelain, C. Joanblanq, and Senn, P, "A fast single-chip implementation of 8192 complex point FFT," *IEEE Journal of Solid-State Circuits*, vol.30, no.3, pp.300-305, Mar 1995.

Brett W. Dickson received the B.S. and M.S. degrees in Electrical and Computer Engineering from Worcester Polytechnic Institute (WPI), Worcester, MA, in 2006 and 2008 respectively.

In 2006, he joined the Ultrasound Research Laboratory at WPI where he worked on the system level integration of a voice-controlled mobile ultrasound system. In 2008, he was hired as a Verification Engineer at SiCortex, Maynard, MA, where he helped verify the memory management unit of a multi-core MIPS64 processor. In 2009, he joined the MITRE Corporation, Bedford, MA, as a Senior Integrated Electronics Engineer. He is currently involved in the design and verification of digital signal processing systems targeted for both FPGA and ASIC platforms.

Albert A. Conti received the B.S. degree in Computer Systems Engineering from Boston University, Boston, MA, in 2004 and the M.S. degree in Electrical Engineering from Northeastern University, Boston, MA, in 2007.

In 2003, he joined the CAAD Lab at Boston University where his research focus was hardware acceleration of computational biology and bioinformatics applications. In 2004, he joined the Reconfigurable Computing Lab at Northeastern University where his research was aimed at digital signal processing acceleration with massively parallel systems. In 2007, he joined the MITRE Corporation, Bedford, MA, where he later became the principle investigator for the Emerging Technologies for VLSI Applications research program. In 2012, Al joined Cognitive Electronics Inc., Boston, MA, where he is currently developing an in-memory, massively parallel processor for Big Data applications.