A Family of Scalable FFT Architectures and an Implementation of 1024-Point Radix-2 FFT for Real-Time Communications

Adnan Suleiman[1], Hani Saleh[2], Adel Hussein[3], and David Akopian[3]
[1]*Cirrus Logic*, [2]*Intel Corporation, and* [3] *University of Texas at San Antonio, Electrical and Computer Engineering Department*

***Abstract*— The paper presents a family of architectures for FFT implementation based on the decomposition of the perfect shuffle permutation, which can be designed with variable number of processing elements. This provides designers with a trade-off choice of speed vs. complexity (cost and area.). A detailed case study is provided on the implementation of 1024-point FFT with 2 processing elements using 45nm process technology, including area, timing, power and place-and-route results.**

I. INTRODUCTION

The Fast Fourier Transform (FFT) is a conventional method for an accelerated computation of the Discrete Fourier Transform (DFT) [1], which has been used in many applications such as spectrum estimation, fast convolution and correlation, signal modulation, etc. Even though FFT algorithmically improves computational efficiency, additional hardware-based accelerators are used to further accelerate the processing through parallel processing techniques. A variety of architectures have been proposed to increase the speed, reduce the power consumption, etc. [18].

A single memory architecture consists of a scalar processor connected to a single N-word memory via a bidirectional bus. While this architecture is simple, its performance suffers from inefficient memory bandwidth [18]. A cache memory architecture adds a cache memory between the processor and the memory to increase the effective memory bandwidth. Baas, in [2], presented a cache FFT algorithm which increases energy efficiency and effectively lowers the power consumption.

Dual memory architecture, implemented in [3], [22], [23], uses two memories connected to a digital array signal processor. The programmable array controller generates addresses to memories in a ping-pong fashion.

The processor array architecture [4], consists of independent processing elements, with local buffers, which are connected using an interconnect network.

Pipeline FFT architectures, introduced in [5], contain logrN blocks; each block consists of delay lines, arithmetic units that implement a radix-r FFT butterfly operation and ROMs for twiddle factors. A variety of pipeline FFTs have been implemented [6]-[9]. Most pipeline FFT realizations use delay lines for data reordering between the processing elements. Although this gives simple data flow architecture, it causes high power consumption.

Several techniques have been proposed for memory address generation. Cohen described an address generation scheme based on a counter, shifters and rotators [10]. It allows parallel organization of memory so that the data used at any instant reside in different memories. Pease proposed dividing the memory into sub-memories for overlapping the access [11]. A multi-bank memory address assignment for a radix-r FFT was developed in [12]. The memory assignment minimizes the memory size and allows conflict-free simultaneous memory access. Ma developed a fast address generation scheme [13] with hardware cost comparable to the address generation scheme in [10]. Ma and Wanhammar proposed an address generation scheme in [14] to reduce the hardware complexity and power consumption.

Many of the FFT algorithms relate to the "butterfly structure" presented first by Cooley and Tukey [1] where separate processing element (PE) is assigned for each node of the FFT flow. FFT algorithms have several stages of so-called butterfly computations, and a number of butterflies are calculated at each stage. In the pipeline FFT architecture all the butterflies of each stage are computed using a single PE and PE assigned to different stages form a line of processors. It is also possible to map the computation network into another line of processing where the stages of FFT are sequentially computed by parallel PE's connected by the perfect shuffling network. All the butterflies of a single stage are computed in parallel. It is called iterative architecture in [18].

This paper describes a scalable architecture which is a compromise between various implementations as it provides a systematic approach of FFT computing on the selected number of processing elements connected through the perfect shuffle interconnection network.

The method proposed in this paper is based on a decomposition of perfect shuffle which results in a constant geometry structures. Each stage of the FFT is computed in by the available number of processing elements. The PEs computes several butterfly operations in parallel and the process is repeated until all butterflies of the stage are computed. Then the PEs start the computation of the next stage and so on. The data flow between processors is automated guaranteeing correct uninterrupted data flow. This approach allows for choice of specific number of processing elements thus, the scalable architecture. This work is the continuation of the effort started in [19]-[21]. Other scalable approaches in [15]-[17] use complicated

978-1-4244-2658-4/08/$25.00 ©2008 IEEE

sequential perfect shuffling networks while our proposed approach presents a more systematic alternative method.

As a case study this paper also presents the implementation of a radix-2 1024-point FFT using two PEs. The architecture and algorithm can be easily extended to higher radix implementation; however, this is beyond the scope of this paper.

II. FFT Constant Geometry

In [1] Radix-2 FFT of N points –where N is integer power of 2- requires N log2N complex operations compared to N2 of direct DFT computation. Also, algorithms are known in two types DIT, decimation in time, where complex multiplication occurs after the two-point DFT; and DIF, decimation in frequency, where complex multiplication occurs before the two-point DFT. Further more, the notations "in-place" and "not-in-place" refer whether an input point ends up in the same register it came from or not. The FFT algorithm consists of log2N stages. Each stage consists of N/2 radix-2 butterfly operations. Fig. 1 shows a radix-2 16-point for N=16 and W=2 constant geometry algorithm with ordered input and bit-reversed output [18]. Since output of the butterflies did not go back from where it came, this refers to not-in-place algorithm.

Constant geometry algorithm is well suited for hardware implementation due to the symmetry rippling across stages. Eventually, only one stage is translated into hardware, the rest of stages utilize the same hardware with additional control. The architecture is designed to exploit operation-level parallelism in each stage.

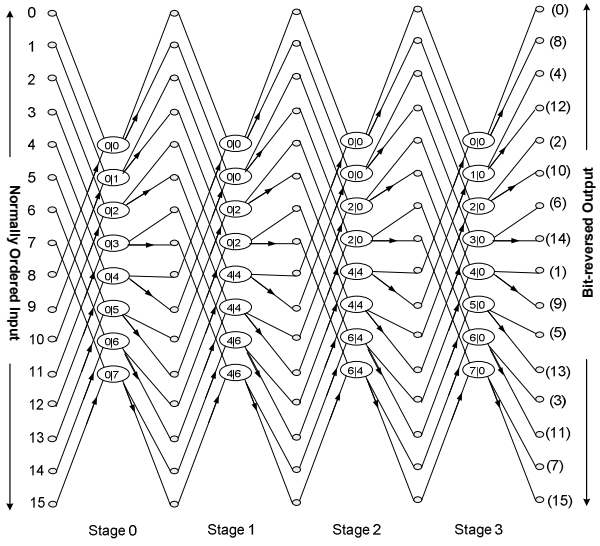

Fig. 1. Constant Geometry Not-in-place16-point FFT.

III. Decompositions of Perfect Shuffle (Unshuffle) Reordering

In [15] data permutations are considered as operators acting on the indices of data where code word for binary representation of an index is specified by the fields:

$$[x, y, z] = [x_u, ..., x_1\ y_v, ..., y_1\ z_w, ..., z_1] \quad (1)$$

Where: $[x_i, y_i, z_i] \in \{0,1\}$ and $x.2^{w+v} + y.2^v + z$ is index value.

The representation describes data flow in parallel structure and translated [15] as [cycle, PE, path] where cycle, references the cycle at which data is available for computation, PE is a processing element –otherwise butterfly- and path, the particular input channel of a processing element. The idea behind this is that each bracketed representation corresponds to an architecture where 2w butterflies of 2v data items are computed in parallel in one column of 2w processor element (PE) with 2v inputs each, and the x coordinate establishes the sequence process in each stage of the transform (2w+v). Calculation time per stage is 2u clock cycles where time period is limited by computational time at a butterfly of 2v data items.

If R is the reordering operator acting on an index then applying the operator R to this index yields:

$$R[a_{l-1}, ..., a_0] = [b_{l-1}, ..., b_0] \quad (2)$$

Applying several iteration of the data permutation sequentially is possible yielding the following:

$$R_2(R_1[a_{l-1}, ..., a_0]) = R_2([b_{l-1}, ..., b_0]) = [c_{l-1}, ..., c_0] \quad (3)$$

This convention permits the identification of intermediate location during sequential data permutation as can be illustrated in fig. 2. Furthermore, let us define the operator acting on indices of data locations and performing a perfect shuffle, $P^T_{N,2}$ – un-shuffle, $P_{N,2}$ - permutation of order k as:

$$\sigma_{(k)}[x, y, z] = [[z_k ... z_1\ x_u ... z_{k+1}], [x_k ... x_1\ y_v ... y_{k+1}], [y_k ... y_1\ z_w ... z_{k+1}]] \quad (4)$$

And

$$\sum_{(k)}[x, y, z] = [[x_{u-k} ... x_1\ y_v ... y_{v-k+1}], [y_{v-k} ... y_1\ z_w ... z_{w-k+1}], [z_{w-k} ... z_1\ x_u ... x_{u-k+1}]] \quad (5)$$

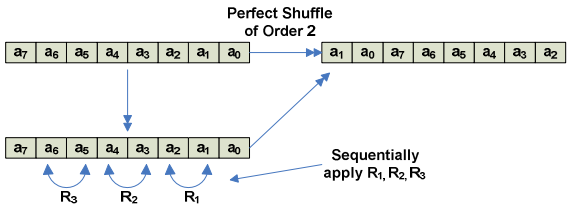

Fig. 2. Decomposition of the Perfect Shuffle.

Additionally, these operators could be defined acting on two indices as:

$$\sigma_{(k)}^{x,y} = [[y_k \dots y_1\ x_u \dots x_{k+1}],$$
$$[x_k \dots x_1\ y_v \dots y_{k+1}], \quad (6)$$
$$[z_w \dots z_1]].$$

Then according to [17] the following fact holds:

$$\sigma_{(w)}[x,y,z] = \sigma_{(w)}^{x,z}\ \sigma_{(w)}^{y,z}[x,y,z].$$
$$\sum_{(w)}[x,y,z] = \sum_{(w)}^{y,z} \sum_{(w)}^{x,z}[x,y,z]. \quad (7)$$

Local ordering $\sum_{(w)}^{x,z}$ is implemented by way of FIFO associated with each processing element (PE) for how many are there exist in design. While $\sum_{(w)}^{y,z}$ can be implemented as the interconnect network between PEs represented in external hard connection. In general, the implementation of the external interconnection between PEs and the sequential reordering network are done by a network of hard connections. Additionally, as mentioned earlier, the internal reordering is implemented using special memory organization comprising the sequential perfect shuffle and or un-shuffle networks (SPSN). The next two sections discuss the architecture and hardware implementation of the SPSN in detail. Now, propositions of a family of SPSN obtained form the following two statements for an arbitrary W [19]:

A. *Proposition 1*

(The internal) perfect un-shuffle or and shuffle reordering of order w could be decomposed into set of reordering R¬i as follow:

$$[\sigma_{(w)}][x^p \dots x^0] = R^p \cdot R^{p-1} \cdot \dots R^1 \cdot [x^p \dots x^0],$$
$$[\sum_{(w)}][x^p \dots x^0] = R^1 \cdot \dots \cdot R^{p-1} \cdot R^p \cdot [x^p \dots x^0]. \quad (8)$$

Where:

$$x_i = [x_w^i \dots x_1^i],\ x_i^j \in \{0,1\} \quad and$$
$$R^i[x^p \dots x^i\ x^{i-1} \dots x^0] = [x^p \dots x^{i-1}\ x^i \dots x^0].$$

B. *Proposition 2*

$$R^i = R_1^i \cdot \dots \cdot R_{w-1}^i \cdot R_w^i \quad (9)$$
$$R_t^i[x^p \dots x^i\ x^{i-1} \dots x^0] =$$
$$R_t^i[* \dots * x_w^i \dots x_t^i \dots x_1^i x_w^{i-1} \dots x_t^{i-1} \dots x_1^{i-1} * \dots *] = \quad (10)$$
$$[* \dots * x_w^i \dots x_t^{i-1} \dots x_w^{i-1} x_1^i \dots x_t^i \dots x_1^{i-1} * \dots *]$$

IV. Architecture

This section explains the system using top-down analysis. Fig. 3 shows block diagram of the overall system with a perfect shuffle interconnect network. Similar to any FFT algorithm, however, this is more generalized statement, the scalable FFT comprises of number of computational elements –including complex adders and multipliers, reordering network, storage elements– for both twiddle factors storage and inter-stages data exchange.

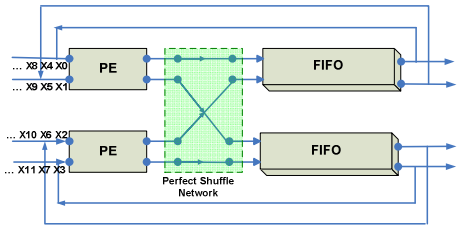

Fig. 3 2PE W=1 FFT Architecture

Fig. 3 shows a Radix-2 scalable FFT model for W = 2 with two PEs, associated FIFOs and perfect shuffle interconnect network. For the purpose of simplicity of our system description, the loop back connections are shown in here. The system is as simple as it can be. Here, two PEs are serving the entire system for how many number of 2n point the system is designed for. The PE in this case also serves as data feeder for on-the-fly or the inter-stages data inputs besides hosting the complex computational butterfly unit. Note that PE location can be on either sides of the diagram without impacting or changing the functionality of the system. Furthermore, regardless of PE location, the data feeder part is kept on the input side of the module. In fact our simulated model conforms to this configuration.

FIFO's comprised of SPSN network, which its shift-exchange units (SEU) are proportional to the size of FFT computational points N in this case. For the scalable architecture, first, the number of PEs is decided then FIFO size becomes totally dependant on the FFT size N.

With very large N, the number of SEUs in a FIFO increases dramatically. For example, for an N = 1024 and PE =2 radix-2 FFT, the number of SEUs reaches 8 with last SEU holding 128 delay storage.

A third component of major importance is the perfect shuffle (unShuffle) network. As mentioned earlier, this part of the architecture is implemented with direct connections either prior to FIFO inputs forming (perfect shuffle) or after FIFO outputs forming (perfect undhuffle). Both configurations facilitate external data reordering in coordination with FIFO's SEUs forming an uninterruptible data exchange and reordering system.

The processing element is considered by far the most critical components of the architecture because it contains the FFT computational unit, otherwise butterflies. Each PE is comprised of one feeder and one butterfly. A bi-directional input register – controlled by a select signal- and one direction output form the unit feeder. Depending on the FFT radix r, feeder's inputs and outputs are varied via parametrical instantiation of the PE module. When select signal is enabled, the feeder presents a set of new input data to the butterfly for computation as shown in fig. 4. In another configuration, the output of the feeder is sent to the interconnect network or directly into FIFO – this is the case used in our simulation. The last stage output forms the second feeder's input.

Butterfly is integral part of the PE unit. It could be located

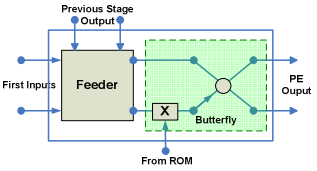

Fig. 5. PE Structure.

before or after the feeder depending on desired system configuration –there are no variations in overall system's attributes for various configurations. Butterflies perform complex addition and multiplication operations; thus, for radix-2 FFT, there are four real adders and 6 real multipliers. Our RTL model is based on fixed-point operations in which, real and fractional parts of a data point are split in half. Depending on configuration, the output of a PE is sent to either reordering interconnect network or FIFOs. Note that Butterflies and interconnect network are configured similar to that of fig. 3 and used throughout the rest of this paper.

Fig. 5 shows top-down decomposition structure of a FIFO and its components. A FIFO is formed by way of SPSN which, essentially composed of number of smaller units called SEUs. The number of SEUs in an SPSN is calculated as

$$\#SEU = \log_2(\frac{N}{PE}) - 1 \quad (11)$$

Equation (11) says that when number of PEs increases, the number of delay elements is reduced dramatically. This in tern decreases overall system delay. Section 5 presents examples illustrating data iteration using same FFT size N with 2 and 4 PEs. Also, Table 1 presents SEUs calculation for various N samples.

Generally, SEUs have one common structure; two equally sized arrays of delay elements separated by criss-cross switch – also known as commutator. However, they differ by the size of delay arrays, which in their part dependant on their sequence order (i) in the SPSN. Therefore, SEUs are sized in the order of 20, 21, 22, 23… 2i.

The operation of SEU is simple; an element entering an

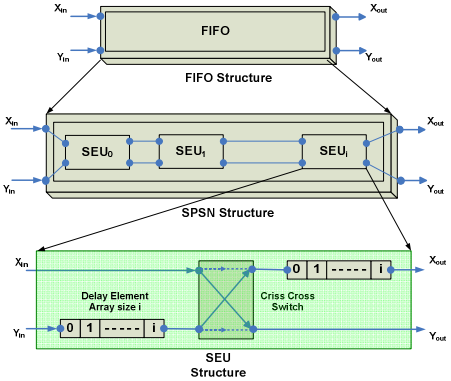

Fig. 4. FIFO Structure.

Table 1. Calculation of Number of SEUs in a Design.

N	2PE	4PE	Num Stages
16	2	1	4
32	3	2	5
64	4	3	6
1024	8	7	10

SPSN is delayed by the length of the delay array of current SEU from its coupled input at the second SPSN. Data shuffle and reordering occurrence is based on the status of current switch –commutator- and current location of data datum in the delay arrays. Reordering occurs during SEU's switch on position at which, an element moves from the first SPSN to the other and vise versa. The output of an SEU is sequentially fed into subsequent SEUs with longer delay and slower switching rate according to a ratio of order 20, 21, 22, 23… 2i. The total delay in FIFO is the sum of delay in all SEUs and the total delay in the entire structure is the sum of delays in all FIFOs.

One final thought about presented system is that FFT radix intentionally was not specified. This is to continue with the notation that the scalable architecture is also radix scaled and could easily be extended to higher radix

V. Algorithm Illustration

Next, we present two examples illustrating the functionality of data reordering and shuffling for a 16-point and 64-point FFTs, DIT with out of place output using 2 and 4 PEs respectively. In the first example of fig. 6, a 16-point FFT with 2 PE is shown. Data at the Feeder is represented in column one, columns with colors represent data in SUEs – yellow for SEU0 with one delay element and dark green is for SEU2 wit two delays,- and data in last column represents butterflies computations. Interconnect network is resembled by thin arrows.

NOTE: Numbers in the boxes represent data indices, color boxes show reordered indices, and thick blue arrows represent iteration.

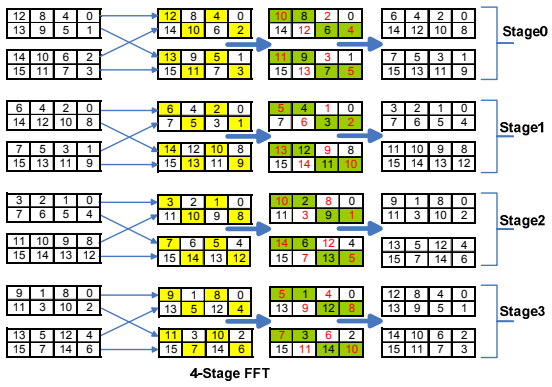

Fig. 6. PE=2, W=1, N=16, All Stages FFT .

Given N = 16 and PE = 2, four data sets presented by the feeder through interconnect network. It is clear from the figure above that first data sets (0,8) reaches butterfly after three clock cycles. It also shows that it takes four cycles to complete one full stage. This means a total of 8 cycles are required for first set of computation and total of 19 clock cycles to have all 16 results at the output. With 4 PE and same number of stages, only one SEU is required. In this case, the first set (0,8) arrives at the butterfly after 1 cycle delay and each stage requires 2 clock cycles to compute, a total of 4 x 2 +1 = 9 clock cycles. That is a saving over 2PE-design of 10 clock cycles; that is over 50% faster.

NOTE: it seems at first glance that number of delay elements in 2PE-design is much larger than that of 4PE-design, thus, larger hardware but it is not the case. A 4PE-design requires twice the number of complex adders and multipliers, which in tern increases area versus 2PE-design.

Fig. 7 shows only the first stage of 64-point FFT transfer of 2PE-design. Making the same observations as last example, it takes 16 clock cycles for first set (0,32) to appear at butterflies with 2PE-design and 8 cycles with 4PE-design; also, there are 4 SEUs in the 2PE vs. 3 SEUs in the 4PE. In this example we also see a speed improvement of 50% going from 2PE to 4PE design. Total cycles to calculate one stage of the 64-point for 2PE and 4PE is 16 and 8 cycles respectively and overall calculation time is 6 x 16 + 15 = 111 cycles for 2PE and 6 x 8 + 7 = 55 cycles for 4PE designs.

Similarly, in this example the number of SEUs is reduced by 1 when using 2-PE vs. 4-PEs yet number of delay elements is the same for both designs. As an observation from this example, the complexity of design increases in two areas: first the area of processing elements; and second, doubling the size of perfect shuffle interconnect network – mainly datapath sizing. Additionally, number of clock cycles for 64-point FFT computation is reduced by more than 50% contributing to increase in throughput of about the same factor as well.

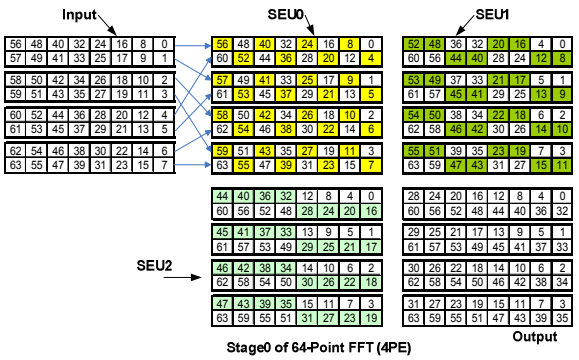

Fig. 7. PE=4, W=1, N=64, Stage0 FFT.

Other performance parameters are in one way or the other impacted by the total number of PEs available in a design, the picture becomes clearer when analyzing hardware implementation. Additionally, as a comparative edge, table 2 shows a summary of important architecture characteristics comparison between conventional pipeline architecture –see

Table 2. Conventional Pipeline and Scalable Architectures Characteristics Comparison.

Factor	Pipeline Architecture	Scalable Architecture
Coefficient Storage	N - 2	N / PE
Delay Elements	N – 2	N / PE
Multipliers (real)	4 $\log_2 N$	4 PE
Adders (real)	6 $\log_2 N$	6 PE
Total time to process first N point	N/2 cycles	N/2PE – 1
Time for subsequent N points	N/2 cycles	N/2PE

[6], [18] – and the scalable architecture.

First, number of coefficients storages is divided by the available number of PEs, then, in general, N/PE. As was shown in examples, the overall delay element is equal to N but the total observed delay is actually proportional to PE and equals to N/PE. Unlike pipeline FFT, The number of complex multipliers is directly related to number of processing elements in the scalable architecture. Each butterfly requires 4 multipliers and 6 adders for complex operation, thus, 4*PE real multipliers and 6*PE real adders are required in a design. Time to process the first set of data points is a function of FFT size N and number of processing elements PE; generally is N/2PE -1 while the next N points will require N/2PE cycles. Number of FIFOs in a design is equal to number of PEs. Number of SEUs in a FIFO is calculated by equation presented in section 4. Number of switches in a design is equal to PE * SEUs in one FIFO.

VI. Implementation of a 1024-Point FFT

A. Placement and Route

The FFT core was designed using Verilog-HDL and implemented using an automatic synthesizes place and route approach. The FIFOs were implemented using normal D-Flops. A very high performance, 45nm, process was used for the implementation with standard cell library carefully designed for high speed applications. The routing was limited to metal layer-7. Tables 3 summarize the implementation results. Fig. 8 shows the floorplan of the core, the design elements are colored differently to show their relative size to each others. Fig. 9 shows the routed FFT core. The FFT core occupied an area of 569.5µm by 570.0µm, of which the memory elements occupied 43.7% while the combinational logic occupied 56.3% with a total utilization of ~69.3%.

B. Timing

The placed, routed and tapeout ready FFT core meets timing for setup and hold at 653.6 MHZ (~1530ps period)

Table 3. Design Specifications.

Item	Details
FFT Algorithm	Radix-2, Decimation-in-Frequency
N	1024 points
Format	Fixed-point (int.frac): 16.16
Number of PEs	4
Total Number of Cells	89,045
Combinational area %	56.3%
Non Combinational area %	43.7%
Height	569.52 uM
Width	570.08uM
Utilization	69.3%
Total Wire Length	1,937,640.81 uM
Frequency	653.6 MHz
Technology	45 nm Bulk CMOS
Supply Voltage	0.9 V
Dynamic Power	168.6mW
Leakage Power	14.7mW

using industry standard STA tools, an extracted and back-annotated netlist was analyzed. At this cycle speed, a 1024-point FFT will complete in ((1024/2*4) - 1 for first stage + (1024/2*4) * 9 for the rest of stags) = 1279 cycles. At a 1530ps cycle time, this translates to 1279* 1.53ns = 1.957µs. Fig. 10 shows the critical timing path for the design which was from one of the SPSN registers to an output.

VII. Conclusion

The proposed systematic scalable pipeline architecture presents a new efficient method for decomposition of perfect shuffle permutation and data reordering for FFT algorithm. Examples for both 2 and 4 PE radix 2 FFT discussed in detail. Placement and timing for 1024 points radix 2 presented as prove of concept. Also, proposed architecture can be easily proved extendable to higher FFT radix.

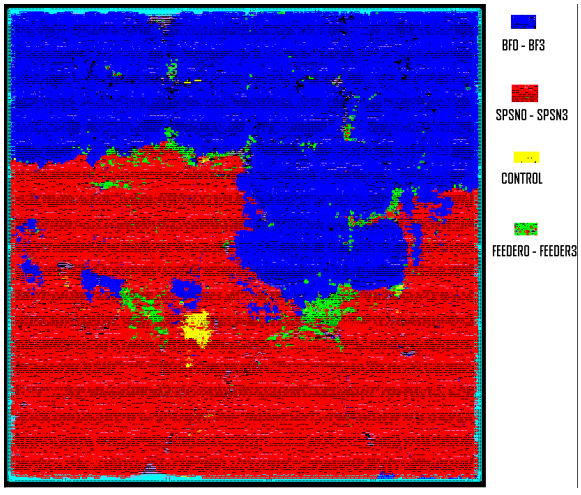

Fig. 10. FFT Core Floorplan.

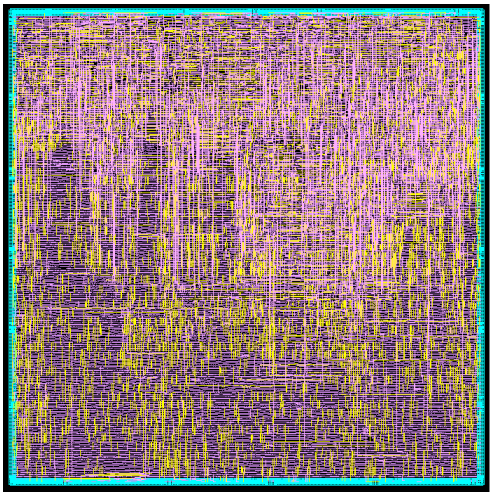

Fig. 8. FFT Core Routing.

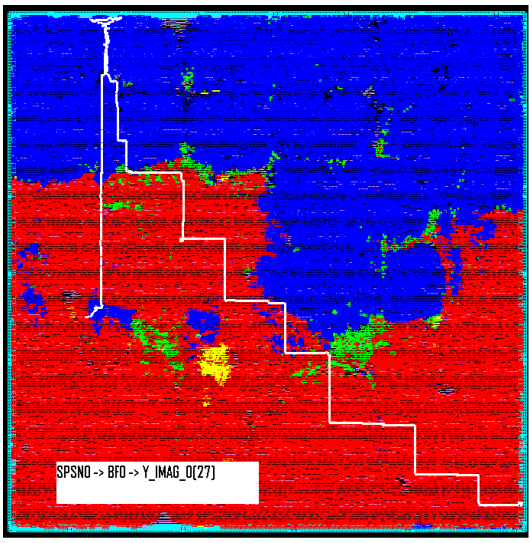

Fig. 9. FFT Core Critical Path.

References

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput., vol. 19, pp. 297-301, 1965.

[2] B. M. Baas, "A low-power, high-performance, 1024-point FFT processor," IEEE Journal of Solid-State Circuits, vol.34, no. 3, pp. 380–387, March 1999.

[3] Magar, S., S. Shen, G. Luikuo, M. Fleming, and R. Aguilar, "An Application Specific DSP Chip Set for 100 MHz Data Rates," International Conference on Acoustics, Speech, and Signal Processing, vol. 4, pp. 1989–1992, April 1988.

[4] O'Brien, J., J. Mather, and B. Holland, "A 200 MIPS Single-Chip 1K FFT Processor," IEEE International Solid-State Circuits Conference, pp. 166–167, 327, 1989.

[5] H. L. Groginsky and G. A. Works, "A pipelined fast Fourier transform," IEEE Transactions on Computers, vol. C-19. pp. 1015-1019, 1970.

[6] E.H. Wold and A.M. Despain, "Pipeline and parallel-pipeline FFI7 processors for VLSI implementation," IEEE Transactions on Computers, vol. C-33, pp. 414-426, May 1984.

[7] G. Bi and E. V. Jones, "A pipelined FFT processor for word sequential data," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, pp. 1982-1985, December 1989.

[8] E. E. Swartzlander, V. K. Jain, and H. Hikawa, “A radix 8 wafer scale FFT processor,” Journal of VLSI Signal Processing, vol. 4, pp. 165-176, May 1992.

[9] He, S. and M. Torkelson. “Design and Implementation of a 1024-point Pipeline FFT Processor,” IEEE Custom Integrated Circuits Conference, pp. 131–134, May 1998.

[10] D. Cohen, “Simplified control of FFT hardware,” IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-24, pp. 577-579, 1976.

[11] M. C. Pease, “Organization of large scale Fourier processors,” Journal of the ACM, vol. 16, pp. 474-482, 1969.

[12] L. G. Johnson, “Conflict free memory addressing for dedicated FFT hardware,” IEEE Transactions on Circuits and Systems, II, vol. 39, pp. 312-316, 1992.

[13] Y. Ma, “An effective memory addressing scheme for FFT processors,” IEEE Transactions on Signal Processing, vol. 47, pp. 907-911, 1999.

[14] Y. Ma and L. Wanhammar, “A hardware efficient control of memory addressing for high-performance FFT processors,” IEEE Transactions on Signal Processing, vol. 48, pp. 917-921, 2000.

[15] E. L. Zapata and F. Arguello, ”Application-specific architecture for fast transforms based on successive doubling method,” IEEE Trans. Sig. Processing, vol.41, pp. 1476-1481, 1993.

[16] E. L. Zapata and F. Arguello, ”A VLSI constant geometry architecture for the fast Hartley and Fourier transforms,” IEEE Trans. Parallel Distrib. Syst.,vol.3, pp. 58-70, 1992.

[17] E. L. Zapata and F. Arguello, “A VLSI constant geometry architecture for the fast Hartley and Fourier transforms,” IEEE Trans. Parallel Distrib. Syst.,vol.3, pp. 58-70, 1992.

[18] Rabiner, L. R. and Gold, B.”Theory and Application of Digital Signal Processing,” Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[19] D. Akopian, J. Takala, J. Astola and J. Saarinen, “Multistage interconnection network for k/n rate parallel Viterbi decoders,” IEEE Trans. on Communications, Sep. 2003.

[20] *D. Akopian, Systematic Approaches to Parallel Architectures for DSP Algorithms, 1997, Acta Politechnica Scandinavica, EI 89, Espoo, Finland*.

[21] *D. Akopian and J. Astola, "Fast architecture oriented algorithms for trigonometric transforms and their mapping to scalable structures," Proceedings of First International Workshop on Transforms and Filterbanks, TICSP Series-1, June 1998, pp. 433-471*.

[22] Hani Saleh and Earl Swartzlander Jr., "A Contention-Free Radix-2 8k-points Fast Fourier Transform Engine Using Single Port SRAMs", IEEE SoutheastCon 2008.

[23] Hani Saleh, Bassam Mohd, Adnan Aziz and Earl Swartzlander Jr., "Contention-Free Switch-Based Implementation of 1024-point Fourier Transform Engine", ICCD-2007 (XXV IEEE International Conference on Computer Design), Lake Tahoe, CA, 2007.