

# A High Performance Scalable FFT

J. Greg Nash

Centar

Los Angeles, CA, USA

([jgregnash@centar.net](mailto:jgregnash@centar.net), [www.centar.net](http://www.centar.net))

**Abstract-** An FPGA implementation of a fast Fourier transform (FFT) circuit is described as an example of a new class of parallel FFT architectures that can provide better performance and functionality than traditional pipelined FFTs and is well suited to wireless applications. Any circuit implementation can perform any transform size that is a multiple of 256 as long as adequate memory is provided. Throughput can be adjusted by adding/subtracting identical blocks of hardware. Power is minimized by using only small memories, providing systolic data flow to avoid memory reads/writes, and localizing connections to reduce interconnect overhead. High dynamic range is obtained through a unique multiple block floating point (BFP) feature. The design is regular in structure and is built with only three simple cells. An FPGA based demonstration circuit with 89db signal-to-quantization-noise has been implemented that can perform a 256-point transform continuously at 0.66  $\mu$ sec/transform. Circuit comparisons are made with a modern commercial pipelined FFT.

**Keywords-** fast Fourier transform; discrete Fourier transform; orthogonal frequency division multiplexing; parallel; circuit

## I. INTRODUCTION

The discrete Fourier transform (DFT) appears prominently throughout a large number of communications signal processing applications [1]. In particular it has been proposed for use in a variety of wireless transmission standards, as for example orthogonal frequency division multiplexing (OFDM) [2], scaleable Orthogonal Frequency Division Multiple Access (OFDMA) [3] and use of digital beamforming to reduce co-channel interference [4].

Future wireless protocols will also be influenced as future FFT circuits offer higher performance, lower power, and more flexibility and functionality. Here a new type FFT architecture is described that offers a variety of features well suited to wireless communications applications. For example, with regard to OFDM-based protocols an important issue is flexibility in choosing the number of sub-carriers. Here, a traditional FFT suffers a power-of-two limitation that severely restricts the number of reachable points and causes their distribution to be highly non-uniform. Alternatively, more control in the choice of transform sizes can benefit overall system performance as in the recently announced Chinese Digital Multimedia Broadcasting Terrestrial/Handheld standard (DMB-T/H) which uses OFDM based on 3780 sub-carriers rather than the power-of-two value 4096 [5]. In this case a 3780-point FFT circuit was developed which provided

better overall system performance even though it was slower and used more memory than a 4096 split-radix pipelined FFT circuit with which it was compared. Non-power-of-two transform sizes also have been proposed in new wireless protocols that use OFDMA as in 802.22 (1024, 2048, 4096, 6144 points) [6] and 3GPP LTE (128, 256, 1024, 1536, 2048 points) [7].

In OFDMA based protocols considerable reduction in computational complexity and power consumption can be realized via the process of “pruning”, because the number of desired FFT outputs or inputs can be small compared to the transform length. For example, a base station transmitter may send the same OFDM symbol to many (inexpensive) receivers, each of which may only use a small fraction of the transmitted data [9]. Similarly, in an uplink scenario (inexpensive) transmitters may each simultaneously send a smaller number of separate signals. Pruning refers to the process of eliminating unnecessary links between stages in an FFT signal flow graph so that fewer computations are required.

The “base-4” architecture described here is designed specifically to address such future requirements for FFT computation. In this paper Section II discusses related work. Section III and IV provide a mathematical foundation and overall architecture. Section V describes a 256-point FFT circuit implementation on an FPGA and compares this result to a high performance commercial device. Section VI discusses scalability and partitioning, Section VII summarizes pruning options and Section VIII is the conclusion.

## II. RELATED WORK

Avoiding the transform “power-of-two” limitation has most commonly been addressed by using more flexible systolic array architectures [10]. However, the single biggest drawback to past use of systolic arrays has been the substantial arithmetic hardware that is required because they use a number of complex multipliers equal to the size of the transform. Thus, a 256-point DFT would require a prohibitive 256 complex multipliers, compared to only 4 complex multipliers needed in a base-4 implementation [10].

Achieving variable length (“run time”) FFT performance can be done in straightforward way using specially designed lower performance single processor implementations [11] and reconfigurable versions of them [12]. For higher performance there are a few pipelined FFT implementations that permit multiple transform sizes (power-of-two) to be calculated

[13][14][15]. Typically this is done by picking off smaller transform results earlier in the pipeline. However, in each case the choices are limited to a maximum transform size and full scale hardware is used. In contrast the base-4 architecture can do any transform size on any implementation as long as the required amount of memory is available. This is possible because the architecture is highly structured and scalable which considerably simplifies the partitioning issues.

Pruning can be done in a straightforward way in software and approaches have been proposed for parallel hardware implementations [8], although irregularities are introduced and a dynamic (run-time) choice of pruning options are not possible.

### III. FACTORIZATION METHOD

The algorithm described here makes use of two levels of factorization. The first is the well-known row/column factorization,  $N=N_r N_c$ , where  $N$  is the desired transform length and  $N_c$  and  $N_r$  are the number of columns/rows. This approach requires calculation of two sets of DFTs,  $N_c$  transforms of length  $N_r$  (referred to as ‘‘column’’ transforms) and  $N_r$  transforms of length  $N_c$  (referred to as ‘‘row’’ transforms). In between column and row transforms it is necessary to multiply each of the  $N$  points by a corresponding twiddle factor,  $W_N^{n,k}$ ,  $n=0,1,\dots,N_c-1$ ,  $k=0,1,\dots,N_r-1$ . (Without the twiddle multiplication a 2-D DFT is performed.)

The second level of factorization is applied separately to each row or column DFT. The index remapping for each column or row DFT starts with the DFT defined as

$$Z(k) = \sum_{n=0}^{M-1} W_M^{nk} X(n), \quad (1)$$

where  $M$  is the transform length,  $X(n)$  are the time domain input values,  $Z(k)$  are the frequency domain outputs and  $W_M = e^{-j(2\pi/M)}$ . In matrix terms (1) may be represented as

$$Z = CX, \quad (2)$$

where  $C$  is a coefficient matrix containing elements  $W_M^{nk}$ . If  $M$  can be factored as  $M = N_1 N_2$ , then applying the reindexings  $n = n_1 + N_1 n_2$  and  $k = k_1 + N_1 k_2$  with  $n_1 = 0, 1, \dots, N_1 - 1$ ,  $k_1 = 0, 1, \dots, N_1 - 1$ ,  $n_2 = 0, 1, \dots, N_2 - 1$ ,  $k_2 = 0, 1, \dots, N_2 - 1$ , it can be shown [10] that if  $N_1 / N_2$  is an integer value (1) becomes

$$\begin{aligned} Y_b &= W_b \bullet C_{M1} X_b \\ Z_b &= C_{M2} Y_b' \end{aligned}, \quad (3)$$

where  $W_b$  is an  $N_1 \times N_1$  matrix with elements  $W_b[k_1, n_1] = W_M^{n_1 k_1}$ ,  $C_{M1}$  is an  $N_1 \times N_2$  coefficient matrix with elements

$C_{M1}[k_1, n_2] = W_{N_2}^{n_2 k_1}$ ,  $X_b$  is an  $N_2 \times N_1$  matrix with elements  $X_b[n_2, n_1] = X(n_1 + N_1 n_2)$ ,  $Y_b$  is a  $N_1 \times N_1$  matrix,  $C_{M2}$  is an  $N_2 \times N_1$  coefficient matrix with elements  $C_{M2}[k_2, n_1] = W_{N_2}^{n_1 k_2}$ ,  $Z_b$  is an  $N_2 \times N_1$  matrix containing the transform outputs  $Z_b[k_2, k_1] = Z(k_1 + k_2 N_1)$ , ‘‘ $\bullet$ ’’ indicates element-by-element multiplication and  $t$  denotes matrix transposition. In (3)  $C_{M1}$  and  $C_{M2}$  contain  $M / N_2^2$  sub-matrices  $C_B = [c_1 | c_2 | \dots | c_{N_2}]^t$  with the form  $C_{M1} = [C_B^t | C_B^t | \dots]^t$  and  $C_{M2} = [C_B | C_B | \dots]$  due to the periodicity of  $W_{N_2}$ , and  $c_i$  are constant vectors.

The ‘‘base’’  $b$  for the architecture corresponds to the value of  $N_2$  that is chosen in the reindexed formulation (3). Here, we have chosen  $N_2=4$  (‘‘base-4’’) because it represents a good tradeoff between circuit performance and circuit complexity. This selection results in

$$c_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, c_2 = \begin{bmatrix} 1 \\ -j \\ -1 \\ j \end{bmatrix}, c_3 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}, c_4 = \begin{bmatrix} 1 \\ j \\ -1 \\ -j \end{bmatrix} \quad \text{and}$$

$$C_B = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix},$$

where  $C_B$  above is the coefficient matrix for a 4-point DFT and also describes a radix-4 decimation in time butterfly. Consequently, in (3) the matrix multiplications by  $C_{M1}$  and  $C_{M2}$  represent repeated use of a radix-4 butterfly.

The computational advantages the matrix algorithm form (3) is evident when compared to the traditional direct form (2) in that the size of the coefficient matrix  $W_b$  in (3) is  $(M/4) \times (M/4)$  vs. the  $M \times M$  size of  $C$  in (2); consequently the number of overall direct multiplications is reduced by a factor of x16 compared to the direct form on which past systolic FFT implementations are based.

As noted above, the derivation leading to (3) restricted  $N_1/4$  to integer values. Therefore, in the column/row factorization it follows that both  $N_c$  and  $N_r$  must be multiples of 16 because  $M = N_c N_r = N_1 N_2 = 4(N_1) = 4(4n)$ , where  $n$  is an integer. Then, since  $N = N_r N_c$ , transform lengths are restricted to integer multiples of 256. This restriction is the result of choosing the base  $b$  to be four. However, if the base  $b$  is chosen to be two, then a similar analysis would show that a base-2 circuit design could perform any transform that is an integer multiple of 16. Finally, if the first level factorization  $N = N_r N_c$  is not used and  $b=2$ , the direct transform (3) itself can be used so that reachable transform values would be integer multiples of 4. The same architecture supports any of these implementation

approaches, so there are a number of options for matching desired and available transform lengths.

#### IV. ARCHITECTURE

The expression (3) consists of two matrix-matrix multiplies ( $C_{M1}X_b$  and  $C_{M2}Y_b^t$ ) with an element-wise multiplication in between. The matrix-multiplication for both can be done using systolic data flow on an  $N_1 \times 4$  array of elemental processing elements (PEs), each containing nominally two registers and an adder, because the matrices have compatible dimensions [16]. The multiplications by  $W_b$  and  $W_N$  can be done using a single  $N_1$  element linear array in between the two  $N_1 \times 4$  or  $(N_r/4) \times 4$  sized arrays.

The computational flow is illustrated with the architecture shown in Fig. 1 for  $N_r=4$ , corresponding to a column length  $M=16$ . The matrix multiply of  $C_{M1}X_b$  is performed on the left hand side array (LHS). It is obtained from the  $(4 \times N_r/4)$  input stream  $X_b$ , that is clocked into the PE array at time steps  $\tau=1..6$ , and multiplied by the values of  $C_{M1}$  that are stored internally in the LHS array. The multiplier PEs contain coefficients  $W_{ri}$  for row  $i$  of  $W_b$  and produce  $Y_b^t = W_b \bullet C_{M1}X_b$ . Finally the right hand side (RHS) array of PEs accumulates in place (one matrix element per PE) the result  $Z_b$ , from  $Y_b^t$  and the streaming input  $C_{M2}$ . Note that the matrix products  $C_{M1}X_b$  and  $C_{M2}Y_b^t$  involve only exchanges of real and imaginary parts plus additions because the elements of  $C_{M1}$  and  $C_{M2}$  contain only  $\pm 1$  or  $\pm j$ , whereas the product in (2) requires complex multiplications. Also, the distribution of the elements in  $C_{M2}$  does not impose significant bandwidth requirements because full complex numbers are not used.

In general the three basic steps in computing the DFT of  $N=N_r N_c$  are

1. Compute successively  $N_c$  column DFTs of length  $N_r$  on column inputs  $X_{ci}$ ,  $i=1..N_c$ , where the computational flow for a single column is as shown in Fig. 1. The column results  $Z_{ci}$ ,  $i=1..N_c$ , are stored  $N_c$  values per PE in small RHS PE memories.
2. Multiply the  $Z_{ci}$  by the twiddle factors  $W_N^{n,k}$  by moving the  $Z_{ci}$  values in systolic fashion from RHS array through the multipliers to the LHS array. These results, stored  $N_c$  values per PE in small LHS memories, become the row inputs  $X_{ri}$  for the last step.
3. Compute successively  $N_r$  row DFTs of length  $N_c$  on row inputs  $X_{ri}$ ,  $i=1..N_r$ . These are performed in logically the same way as in step 1 using the same  $(N_r/4) \times 4$  sized arrays.

In step 3 above the inputs  $X_{ri}$  in the matrix multiply  $C_{M1}X_{ri}$  are supplied internally from within the LHS PE memories. Also, because in general  $N_c \neq N_r$ , the matrix multiplications here would require arrays of size  $(N_c/4) \times 4$ . Therefore, matrix multiplications in step 3 use a different computational flow. Specifically, this to map both the  $C_{M1}X_{ri}$  and  $C_{M2}Y_{ri}^t$  matrix multiplies to a single physical PE row in the architecture used

to do step 1 (Fig. 2). This can be done by projecting the 2-D matrix multiplies onto a 1-D linear systolic array associated with a physical PE row [16]. The number of row DFTs to be performed is  $N_r$ , so with  $N_r/4$  physical PE rows available, each PE row will perform 4 DFTs. Specifically, physical PE row  $i$  (numbered from bottom to top in Fig. 1) calculates row DFTs  $i+jN_r/4$ ,  $j=0,1,2,3$ .

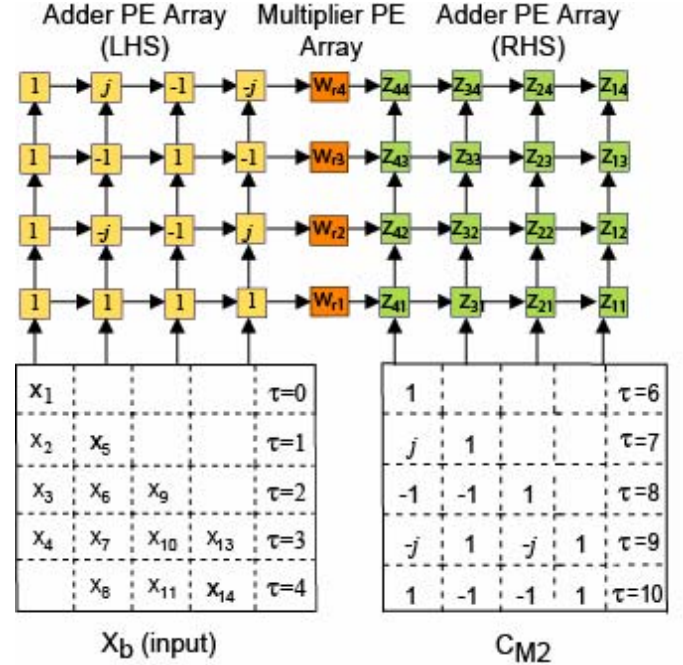


Fig. 1 Basic base-4 architecture showing calculation of a single 16-point column DFT (step 1). Inputs  $X$  (LHS array) and  $C_{M2}$  (RHS array) are shown at the bottom. (“b” subscripts for matrix elements  $z$  are not shown.)

Using the basic “row/column” technique requires a data transpose operation in between column and row DFTs. However, special hardware support for these operations is avoided because it is handled by appropriate shifts in  $C_{M1}$ ,  $C_{M2}$  and  $W_b$  which occur during processing [10].

All the important processing listed in steps 1-3 occurs along the rows of the DFT. The only variables that move vertically are  $X_b$ ,  $C_{M1}$  and  $C_{M2}$  and these propagate unchanged. Therefore, it is easy to provide a small amount of local circuitry in each row to enhance the dynamic range. This is easy to do because both the intermediate result  $Z_{ci/ri}$  of each column/row DFT are accumulated in place on RHS PEs by successive additions. Then, before each PE stores this result to its local memory, the final sum can be normalized and stored with a small exponent value. During twiddle multiplication (step 2), inputs associated with the same row DFT are normalized to the same exponent. During the row DFT computations each computed value of  $Z_{ri}$  keeps its exponent and this exponent is combined with the step 2 exponents to produce a single exponent associated with each FFT output value. Thus, a BFP operation is performed as part of steps 1 and 2, and a floating point operation is performed in computing each row DFT (step 3).

From Fig. 1 it is clear that this architecture is very simple in that it avoids the stage-to-stage irregularities and the complex permutation networks, commutators and butterflies of conventional pipelined FFT implementations. Each LHS and RHS PE in Fig. 1 contains only a (complex) adder, a couple of registers and a small memory, and the center PEs contain just a (complex) multiplier plus registers and small memories as well. Thus, the entire implementation for any transform size is easy to design, test and maintain. Examples of estimated throughput performance for different transform sizes are shown in Table I.

TABLE I.  
PARAMETERS FOR DIFFERENT FFT SIZES

Points	$N_r$	$N_c$	Throughput (cycles/DFT)
256	16	16	220
512	32	16	284
768	32	24	460
1024	32	32	668
1536	48	32	796
2048	64	32	924

## V. IMPLEMENTATION EXAMPLE

As a demonstration (nonoptimized) example the base-4 architecture for a 256-point FFT has been implemented. This size was chosen in part because it's specified in the 802.16 (2004) standard and can therefore be compared to other in-place implementations. It accepts a continuous input stream  $X(n)$ , while generating a continuous output stream  $Z(n)$  at the same rate ("streaming"). The design has circuit pins for real and imaginary inputs/outputs,  $Z/X$ , a single global reset, and two clocks. (Because the base-4 design computes FFTs using a number of clock cycles that is less than the transform size, a separate higher speed clock is used for data I/O.)

An evaluation of the base-4 design is best done by comparison with another modern pipelined FFT design built to the same specifications, using exactly the same underlying hardware. For this purpose a streaming BFP 256-point radix-4 FFT design from Altera was chosen (FFT v2.2.0), since their pipelined BFP FFT circuits are the fastest of which we are aware. Both designs were targeted to an Altera Stratix II EP2S15F484C3 FPGA. Altera Quartus II tools (v5.1) were used to design and evaluate the two FFT circuits. The base-4 circuit operation was verified by comparing the Quartus simulator result with a Centar bit-accurate simulation model. The Quartus II timing analyzer finds the critical path that determines the maximum clock frequencies.

The base-4 and Altera results are summarized in Table II. Because the base-4 design provides higher dynamic range for a given bit-length, a 20-bit Altera FFT was used in the comparison because it provides approximately the same maximum signal-to-quantization-noise figure. The power dissipation numbers were obtained by running the same random (real and complex) numbers through both circuits. The results for the Altera circuit were obtained from a bit-accurate Matlab model that's created by the Altera FFT generator. For the Altera design the clock speed listed in

Table II is also the complex sample rate. For the base-4 design the complex sample rate is larger than the clock speed in the table by a ratio of 256/240 or 385 MHz. In the table the number of "adaptive logic modules" (ALMs) is included because this value is roughly comparable to a Xilinx "slice" for comparison purposes.

TABLE II.  
COMPARISON OF ALTERA AND BASE-4 FFT CIRCUITS

Category	Altera (20 bit)	Base-4 (16-bit)
Throughput (cycles/DFT)	256	240
Clock speed (MHz)	302	361
Throughput ( $\mu$ sec)	0.85	0.66
Signal/Quantization Noise (db)	86.9	89.0
ALMs	4192	4496
Memory Bits	48640	48880
18-bit multipliers	12	16
Dynamic Power (mW)	1752	2441
Energy per FFT (nJ)	1489	1611
Area-Throughput-Memory	0.17	0.15

In Table II the higher base-4 throughput derives from two factors: (1) the higher clock speed associated with the simple PEs and localized interconnections and (2) the reduced number of clock cycles per FFT. Typically, pipelined FFTs provide a throughput (clock cycles/DFT) equal to the number of transform points  $N$ , which is usually much more than that for the base-4 architecture as seen from Table I. This improved performance is achieved with roughly comparable resource usage in that the base-4 "Area-Throughput-Memory" figure of merit is slightly less, where "Area" is the number of ALMs (memory is in Kbits and clock speed is in hertz).

Keeping the circuit power low and performance high is achieved architecturally in three ways:

1. Use of many small memories (one per PE), so that they are both low power and fast. (Only 14% of the total circuit power dissipation comes from the memories.)
2. Reuse of data flowing through registers (systolic processing) so that unnecessary memory reads and writes are avoided.
3. Localized interconnects to minimize wiring overhead. (Total interconnect power is only 46% of the total dynamic power.)

Power dissipation is higher than in the Altera design; however, the energy per FFT (dynamic power x transform time) is roughly comparable. No use of clock enable circuitry was used for any of the registers or memory input/output ports in the demonstration circuit. By adding this control it is estimated that there would be a reduction of 10-15% in power used, so that energy per FFT would be less than for the Altera circuit.

## VI. SCALING

The architecture described in Section IV is regular because it reflects the structure of the underlying DFT matrix equation (3). The array is two dimensional, but scales with

transform size in only one dimension (vertically in Fig. 1) since the LHS and RHS arrays are always  $(N_r/4) \times 4$  in size. Essentially, larger arrays are created by replicating identical sets of four PE rows. There is no corresponding growth in interconnect overhead because all the communication stays local.

The scalability of the architecture is also reflected in the ease with which operations can be partitioned in such a way as to reduce overall hardware. This possibility can be seen more easily if the column DFT expression in (3) is rewritten as

$$Y_b = W_b \bullet \begin{bmatrix} C_B \\ C_B \\ \vdots \\ C_B \end{bmatrix} X_b. \quad (4)$$

In (4) it can be seen that  $C_B$  is applied multiple times to  $X$ , computing the same result each time. This is reflected in the base-4 array architecture in which there are four PE rows for each element  $C_B$  in (4) (in Fig. 1 there are only four rows since  $C_{M1} = C_B$ ). Therefore, it is possible for a single set of four rows to compute all necessary elements of  $Y_b$  and then  $Z_b$ .

As an example, consider a 1024-point transform ( $N_r = N_c = 32$ ). This would nominally use LHR and RHS  $8 \times 4$  PE arrays. However, if only four PE rows are used, as shown in Fig. 1, then the computation could be partitioned, first by calculating

$$\begin{bmatrix} y_{b11} & \cdots & y_{b18} \\ \vdots & \vdots & \vdots \\ y_{b41} & \cdots & y_{b48} \\ \hline & & 0 \end{bmatrix} = W_b \bullet \begin{bmatrix} C_B \\ C_B \\ \vdots \\ C_B \\ 0 \end{bmatrix} X_b \quad (5)$$

$$\begin{bmatrix} z_{b11} & \cdots & z_{b14} & | & 0 \\ \vdots & \vdots & \vdots & | & \vdots \\ z_{b41} & \cdots & z_{b44} & | & 0 \end{bmatrix} = \begin{bmatrix} C_B & C_B \end{bmatrix} \begin{bmatrix} y_{b11} & \cdots & y_{b41} & | & 0 \\ \vdots & \vdots & \vdots & | & \vdots \\ y_{b18} & \cdots & y_{b48} & | & 0 \end{bmatrix}$$

which provides half the answer, and then by the same calculation with  $C_{M1} = \begin{bmatrix} 0 & C_B^t \end{bmatrix}$ . Operationally, this could be performed by doing step 1 from Section IV twice, i.e., stream the input  $X_b$  through the four rows two separate times, using different sets of coefficients  $W_{ri}$  each time. Then step 2 would take twice as long because there half as many multiplier PEs compared to the nominal  $8 \times 4$  array size. Finally, step 3 can be done as before since all the row DFT processing is confined to a single PE row rather than spread out across multiple PE rows as in the column DFT processing (step 1). The main difference is that now each physical PE row would do eight rather than four row DFTs. In this way a simple partitioning scheme is possible, whereby the entire computation can be processed by any set or sets of four PE rows.

If the processing flow is carried out as just described, it would be necessary increase the memory in proportion to the reduction in the number of physical PE rows used. However,

it would also be possible to sequentially perform steps 1-3 from Section IV for each partial execution like that shown in (5). In this way the intermediate storage requirements would be reduced significantly, although some efficiency would be lost due to additional control overhead of starting/stopping operations associated with separate steps 1-3 and additional memory access control overhead. Other “finer grain” partitioning schemes are possible as well.

Partitioning as described above permits two important scaling options. First, it allows a small array to compute any size transform as long as there is enough local PE memory to hold intermediate results. Second, it allows an option to add hardware, e.g., sets of four PE rows, to increase overall throughput (as long as the total number of rows is not more than the nominal array length of  $N_r/4 \times 4$ ). Any rows that are added extend the array vertically from the minimal array shown in Fig.1. This latter feature is particularly important when it is necessary to match required system throughput to hardware throughput.

All FFT sizes could be implemented on a single array of fixed size, as for example the four row “slice” shown in Fig. 1. In this case the throughputs associated with larger FFT sizes would be reduced proportionally. An estimate of the approximate number of clock cycles per transform can be obtained from multiplying that in Table I by the by the reduction in hardware used. For example, the 802.16 (fixed) standard is  $92.4\mu\text{s}$  for a 2048-point FFT. From Table I the base-4 2048-point transform in nominal array form ( $16 \times 4$  LHR and RHS arrays) takes 924 clock cycles. Using only a single four row slice, the processing time would be  $\sim 4$  times slower, or  $4 \times 924 = 3696$  cycles. Even at a conservative clock rate of 300 MHz, this leads to a computation time of only  $\sim 12\mu\text{sec}$ .

## VII. PRUNING

Pruning would be difficult to provide in conventional pipelined FFT circuitry because each pruning option could lead to different and less regular implementations, an unsuitable approach for providing for dynamic or run-time pruning selection. Here, the pruning problem can be handled in the context of partitioning inputs and outputs appropriately. The major difference in this effort vis-à-vis run-time FFT computations above is that (1) only specific sub-sets of inputs and outputs are needed and (2) these aren’t necessarily adjacent.

To achieve multiple-access data subcarriers are divided into groups called subchannels. The actual mapping of subcarriers to subchannels is complex because it is a function of data rates, signal characteristics and mode (uplink/downlink). Also, mappings are not yet defined in future protocols (3GPP LTE). However, the two basic mappings are distributed and adjacent. Therefore, it is necessary to support very fine-grained pruning, where very few of input or output coefficients are computed compared to  $N$ .

Output pruning can be done by selectively calculating the outputs in step 1 of Section IV. Hardware resources and

computation time is then saved by avoiding most of the twiddle multiplications (step 2) and associated row DFTs (step 3). (Each PE row  $k$  calculates elements  $z_{k-1+jNr/4}$ ,  $z_{k-1+N/4+jNr/4}$ ,  $z_{k-1+N/2+jNr/4}$ ,  $z_{k-1+3N/4+jNr/4}$  of the output vector  $Z$ , where  $j=0,1,..,N_c-1$  also specifies a memory location.) For example, if the only subcarriers used corresponded to the first four transform outputs,  $z_0, z_1, z_2, z_3$ , when  $N=1024$ , then only the first four PE rows of the nominal  $N_r/4=8$  rows would be needed ( $k=1,2,3,4$ ) and only the first memory location  $j=0$  would be used, as shown in Fig. 2. (In Fig. 2 the superscripts for  $X_{ri}$  refer to the matrix elements, each LHS PE storing  $N_c/4$  values,  $n=1,2,3,4$ , per row DFT) In Fig. 2 the four desired outputs are computed and stored in the first RHS PE column. The other RHS PEs are also computing results, but these are not stored. In this way the computation uses less than half the resources and can be done in approximately half the time of a fully computed FFT.

Pruning associated with inputs is performed in a similar way in that calculations associated with zero values of  $X_b$  are not performed.

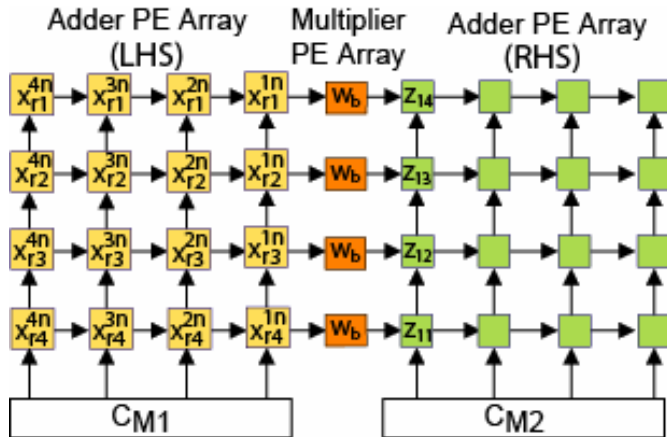


Fig. 2. Row DFT computations, corresponding to step 3 of Section IV, during pruning operation. (“b” subscripts for matrix elements  $x$  and  $z$  are not shown.)

## VIII. CONCLUSION

An FFT circuit has been described that provides the high performance (throughput), dynamic range, low power, functionality, and flexibility that can benefit future needs of wireless communications systems. In particular it provides for run-time FFT transform length selection, pruning to support OFDMA protocols, and non-power-of-two FFT sizes. This is

achieved with a simple, localized, regular circuit which minimizes overall system support costs associated with design, test and maintenance. A demonstration 256-point test circuit was built using 90nm FPGA target hardware that can perform a transform at a rate of 0.66μsec/DFT with 89 db S/QN, which is better than the fastest traditional pipelined device of which we are aware.

## REFERENCES

- [1] Ronald N. Bracewell, *The Fourier Transform and Its Applications*, 3rd Edition, McGraw-Hill, 1999.
- [2] I. Koffman, V. Roman, “Broadband wireless access solutions based on OFDM access in IEEE802.16,” *IEEE Communications Magazine*, April 2002, pp. 96-103.
- [3] [www.wimax.org](http://www.wimax.org)
- [4] Ye Li and N. R. Sollenberger, “Adaptive antenna arrays for OFDM system with cochannel interference,” *IEEE Trans. Commun.*, vol. 47, pp. 217–229, Feb. 1999.
- [5] Zhi-Xing Yang, Yu-Peng Hu, Chang-Yong Pan, and Lin Yang, “Design of a 3780-point IFFT processor for TDS-OFDM,” *IEEE Trans. Broadcasting*, Vol. 48, pp.57-61, Mar. 2002.
- [6] A PHY/MAC Proposal for IEEE 802.22 WRAN Systems ([http://www.ieee802.org/22/Meeting\\_documents/2006\\_Mar/22-06-0005-05-0000\\_ETRI-FT-I2R-Motorola-Philips-Samsung-Thomson\\_Proposal.ppt](http://www.ieee802.org/22/Meeting_documents/2006_Mar/22-06-0005-05-0000_ETRI-FT-I2R-Motorola-Philips-Samsung-Thomson_Proposal.ppt)).
- [7] Physical layer aspects for evolved Universal Terrestrial Radio Access (UTRA), 3GPP TR 25.814 V7.1.0, Oct. 2, 2006. ([http://www.3gpp.org/ftp/Specs/archive/25\\_series/25.814/](http://www.3gpp.org/ftp/Specs/archive/25_series/25.814/)).
- [8] Zhong Hu and Honghui Wan, “A novel generic fast Fourier transform pruning technique and complexity analysis,” *IEEE Trans Signal Proc.*, Vol. 53, NO. 1, Jan. 2005, pp. 274-282.
- [9] Charles D. Murphy, “Low-complexity FFT structures for OFDM transceivers,” *IEEE Trans. Comm.*, Vol. 50, No. 12, Dec. 2002, pp. 1878-1881.
- [10] J. G. Nash, “Computationally efficient systolic architecture for computing the discrete Fourier transform,” *IEEE Transactions on Signal Processing*, Volume 53, Issue 12, Dec. 2005, pp. 4640 – 4651.
- [11] Jen-Chih Kuo, et. al, “VLSI Design of a variable-length FFT/IFFT Processor for OFDM-Based Communication systems”, *EURASIP Journal on Applied Signal Processing* 2003:13, pp. 1306–1316.
- [12] Yutian Zhao, Erdogan, A.T., Arslan, T., “A novel low-power reconfigurable FFT processor,” *Int. Symp. Circuits and Systems, ISCAS 2005*, pp. 41- 44.
- [13] S. M. Currie, et. al, “Implementation of a single chip, pipelined, complex, one-dimensional fast Fourier transform in 0.25μm bulk CMOS”, *Proc. Application-Specific Systems, Architectures and Processors*, 2002, pp. 335 – 343.
- [14] Shousheng He and M. Torkelson “Designing pipeline FFT processor for OFDM (de)modulation”, *Proc. Int. Symp. Signals, Systems, and Electronics*, 1998, pp. 257 - 262.
- [15] A. Cortes, et.al., “AFORE: an IFFT/FFT core generation tool different wireless communication standards,” *Proc. Int. Conf. Consumer Electronics*, 2006, pp. 193-194.
- [16] S. Y. Kung, *VLSI Array Processors*, Prentice Hall, 1988, pp. 138-139.