# A Dynamically Reconfigurable NoC for Double-Precision Floating-Point FFT on FPGAs

Thanh Thi Thanh Bui, Braden Phillips, and Michael Liebelt

School of Electrical and Electronic Engineering
The University of Adelaide, Adelaide, Australia
Email: {thanh.bui, braden.phillips, michael.liebelt}@adelaide.edu.au

***Abstract*—This paper presents a dynamically partially reconfigurable network on chip (NoC) on a field-programmable gate array (FPGA) for double-precision floating-point Fast Fourier Transforms (FFTs). This is one of the first published examples of a practical system using a dynamically reconfigurable NoC that has been implemented in existing FPGA technology. Up to 16 parallel double-precision floating-point processing elements (PEs) can be implemented on the FPGA. Using dynamic partial reconfiguration, a user can change the number of running PEs to choose an optimal power-performance operating point. The design provides much better performance than i7-3.4GHz CPUs running Matlab and competitive performance with static-only FFT systems and the Xilinx FFT IP core, but it has the advantage of saving power and releasing hardware resources when maximum FFT performance is not required. With all 16 PEs running, the design can process an FFT of up to 131072 points and achieves its maximum throughput of of 33.5 FLOPs/cycle on a Xilinx Virtex-7 XC7VX485T FPGA.**

***Keywords*—Network-on-chip, partial reconfiguration, floating point, FFT, parallel architecture, FPGA.**

## I. INTRODUCTION

Dynamically partially reconfigurable FPGAs allow hardware modules to be placed and removed at runtime while other parts of the system keep working [1]. This permits a radical departure from the way application-specific hardware is usually designed. In a static system, there must be a fixed set of processing resources sufficient to meet performance requirements under worst-case load conditions. If the workload changes, processing resources sit idle. A system that moves through modes with distinctly different processing needs, should provide different resources for each mode. Dynamic partial reconfiguration can: reduce power consumption by removing resources not currently required; achieve better utilization by changing the mix of processing resources as the requirements of the system change; and deliver better performance by using heterogeneous processing resources optimized for particular stages in an algorithm, rather than making do with static generic processing resources that must serve all stages.

To exploit this new capability, there is a need for efficient, dynamically adaptive communication infrastructure that automatically adapts as modules are added to and removed from the system. Many network-on-chip architectures have been proposed in the last decade to exploit dynamic reconfiguration on FPGA technology. Examples include DyNoC (Dynamic Network on Chip) [2], CuNoC (Communication Unit Network on Chip) [3], CoNoChi (Configurable Network on Chip) [4], DRNoC (Dynamic Reconfigurable Network on Chip) [5] and OCEAN (On-Chip Efficiently Adaptive Network) [6]. However, most of these have been described in theory only or evaluated using general traffic models. Few have been realized in a practical application. To the best of our knowledge, only DyNoC has been applied to a traffic light controller and CoNoChi has been demonstrated in a dynamically reconfigurable network coprocessor called DynaCORE. However, they have not been fully realized and validated.

While dynamic reconfiguration offers clear benefits in theory, more application experiments are required to understand the benefits and limitations of dynamically reconfigurable NoCs and to guide their further development. The aim of this paper is to begin to address this research gap by using an adaptive NoC to connect parallel Processing Elements (PEs) in a dynamically reconfigurable implementation of the Fast Fourier Transform (FFT) on an FPGA. We have chosen to begin with the FFT because it is widely used in a diverse variety of applications in engineering, science and mathematics [7]. It is also commonly implemented using FPGAs, which can exploit parallel hardware to achieve power-efficient, high-speed performance.

The main contributions of this paper are:

1) To the best of our knowledge, this is the first publication that fully realizes a dynamically partially reconfigurable NoC in a specific application, a double-precision floating-point FFT. The system is implemented on an FPGA platform and evaluated for latency, area and power consumption. We compare it with static-only systems, software implementation on CPUs and Xilinx FFT IP core.
2) We show that using dynamic partial reconfiguration of the FPGA allows the user to efficiently change between power-performance operating points while still maintaining competitive performance with the static-only systems. In fact, the real-time performance of the system is among the fastest FFTs published so far.

The rest of the paper is organized as follows. In Section II, an overview of the FFT algorithm and its hardware implementation are presented. The proposed FFT architecture is detailed

 ISBN: 978-1-61208-496-1 

in Section III. Section IV presents the implementation results and design performance. Finally, the paper is concluded in Section V.

## II. FFT Algorithm and Hardware Implementation

In this paper, we use the radix-2 decimation in frequency variation of the FFT Cooley-Tukey algorithm [7]. This algorithm is composed of butterfly operations, as in Figure 1. The signals are represented by complex variables and for our implementation, they all use double-precision floating point representation. Radix-2 is chosen due to its simplicity and flexibility. Figure 2 shows an example of these butterfly operations arranged in a decimation in frequency architecture.

Many different hardware architectures for the FFT have been proposed. These include the parallel architecture [8], pipelined architecture [9]–[12], and combined parallel-pipelined architecture [13][14]. In this paper, the parallel architecture is chosen in preference to the more common pipelined architecture as it proves a good match with the dynamic NoC approach, achieving efficient hardware utilization without excessive memory bandwidth requirements.

## III. Dynamically Reconfigurable FFT Architecture

### A. Processing Elements

A PE performs a radix-2 butterfly operation. Each PE consists of: four floating-point multipliers and six floating-point adders as in Figure 3 for the full radix-2 butterfly operation; memory storage for input data, twiddle phase factors and intermediate results; a local controller; and a NoC interface. The local controller coordinates the read/write memory operation and defines which PE to communicate with at each stage. The NoC interface works as a bridge between the PE and the NoC. The floating-point multipliers and adders support IEEE 754 double-decision floating-point normal and abnormal numbers. They are deeply pipelined to improve speed.

The floating-point multiplier is based on the design in [15]. Its operation includes multiplying the mantissas, adding the exponents, calculating the result sign, normalizing and finally rounding the result according to the IEEE 754 double-decision floating-point standard. The multiplier consists of 17 pipeline stages.

The floating-point adder is also based on [15]. It consists of several steps as follows: ensure that the exponents of the two operands are equal by increasing the smaller one and shifting right its corresponding mantissa; add/subtract the mantissas if they have the same/opposite signs; normalize and finally round

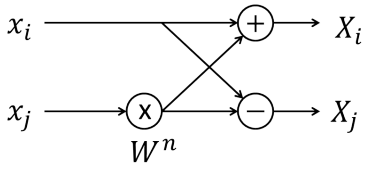


Figure. 1. Radix-2 butterfly datapath.

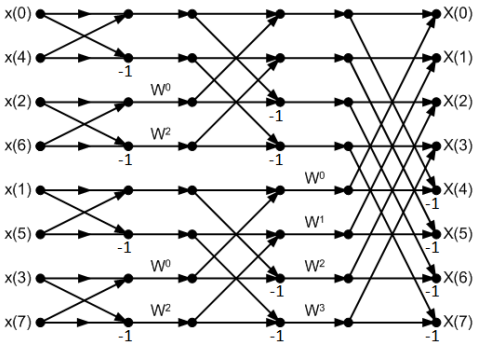


Figure. 2. Decimation in frequency diagram of 8-point radix-2 FFT.

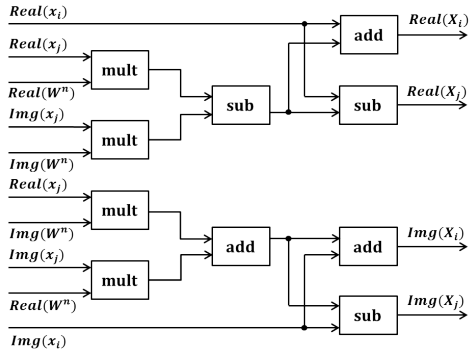


Figure. 3. Radix-2 butterfly implementation.

the result according to the IEEE 754 double-decision floating-point standard. The adder is implemented with 10 pipeline stages. The subtractor can be simply implemented using an adder by inverting the sign of the second operand.

The on-chip memory storage of each PE is determined according to the number of block RAMs available on the FPGA and the number of PEs implemented. A single PE synthesized for a Xilinx Virtex-7 XC7VX485T FPGA occupies 8739 register slices, 13022 LUT slices and 36 DSP48E1 slices. This accounts for 1%, 4% and 1% of the available registers, LUT and DSP slices respectively. The PE achieves a maximum operating frequency of 364 MHz. Up to 25 PEs could be implemented on the device but the number of PEs in a parallel architecture must be a power of two; hence only 16 PEs are implemented. There are a total of 1030 x 36Kb block RAMs; hence each PE can include up to 64 block RAMs. The total on-chip memory required for an $N$-point FFT is approximately $2N$ elements (including data elements and twiddle phase elements). With elements of 128 bits for double precision complex numbers (64 bits for the real part and 64 bits for the imaginary part), the maximum FFT size the design can process is $(1030 \times 36 \times 1024)/(2 \times 128) = 148320$ points. The actual maximum size is 131072 ($2^{17}$) points due to the use of a radix-2 design.

### B. Data Scheduling

Data scheduling is based on the decimation-in-frequency variation of the FFT Cooley-Tukey algorithm. The twiddle

phase factors are pre-calculated and stored in the external ROM. The input data and twiddle phase factors are loaded from the external memory to the on-chip memory of each PE by a global controller. To ensure the parallel operation of PEs, all input elements are transferred to the PEs before they start their operation. The number of elements transferred per cycle depends on the bandwidth of external memories. In this case, one element is transferred per cycle. If the system runs at 200 MHz, the transfer of one 128-bit element per cycle requires a memory bandwidth of 25.6 Gbps. This can be easily obtained by current memory technology.

With an $N$-point FFT and $P$ PEs, $\mathrm{PE}_i$ $(0 \leq i \leq P-1)$ is scheduled with the $\{x_j : Ni/P \leq j \leq N(i+1)/P-1\}$. With this scheduling strategy, there is no communication among the PEs in the first $(log_2(N/P)-1)$ stages; PEs feedback their own results to their inputs for the next stage. In the next $log_2 P$ stages, PEs transfer their results to each other as given in Table I. At each stage, the twiddle phase factors are read in a manner so that they can be reused. Communications among PEs and between PEs and external memory use the NoC.

### C. Network-on-chip Communication

The FFT implementation involves complex routing infrastructure for transferring input data and twiddle phase factors from external memory to the PEs, transferring intermediate results among PEs and outputting the final results from each PE to the external memory. These can all be efficiently implemented by a light-weight 2-D mesh NoC. The modularity and scalability of a NoC can reduce interconnect routing complexity and maintains sustainable performance. The NoC uses small routers, circuit switching and deterministic routing. Circuit switching is used for flow control because it does not require input buffers on the NoC routers. Deterministic routing is used to maximize the throughput of the NoC.

Each router has five input/output ports as in Figure 4. Four from the four cardinal directions (North, East, South and West) and one from the local PE. There are no input buffers at the input ports. Each output port consists of an arbiter and a crossbar switch. The arbiter determines which input port is selected to proceed in the next stage. Then, the crossbar switch matches the successful input port with the desired output port. Each router is connected to its neighbors and local PE through a bidirectional link with 130 bits for each direction. The synthesized single router occupies 758 register slices and 1442 LUT slices, which is relatively small compared with 8739 register slices and 13022 LUT slices for a PE. The router achieves a maximum operating frequency of 522 MHz, which is much higher than 364 MHz for the PE.

In circuit switching flow control, a link between source and destination PEs is set up before a transfer is performed and it is maintained throughout the transfer. Data transferred in the NoC is in 130-bit flit format. When a PE wants to communicate to another, it will assert a Send signal and send a set-up flit to its target PE through the NoC. If the target PE accepts the request, it will assert an Accept signal and a channel is constructed between the two PEs. Data is transferred and the channel is released when the last flit is received. There are 4 types of flits defined by the two most significant bits: set-up flits, data flits, tail flits and control flits.

In the deterministic routing algorithm, the route is defined by the source PE and stored in the set-up flit, beginning with the least significant bit. The routing information is a set of 2-bit directions, beginning with the source to the destination: 00 for North, 01 for South, 10 for East and 11 for West. When a router receives a set-up flit, it extracts the two least significant bits to find out the routing direction and then shifts the set-up flit two bits to the right before sending the flit to the next router. When the output direction is equal to the input direction, the flit is transferred to its attached PE. To ensure the highest throughput and no collision of the NoC, the position of each PE is given in Figure 5.

### D. The Dynamic Partial Reconfiguration System

To support dynamic partial reconfiguration, the system is divided into two areas: a static area with functionality unchanged during system operation; and a dynamic area. This arrangement is shown in Figure 5. The dynamic area is divided into partial reconfiguration regions which can be configured as a PE or a router. Each region must contain sufficient resources−such as slices, block RAMs, and DSP slices−to implement the modules assigned to it.

Based on the user's power-performance requirement, 1, 2, 4, 8 or 16 PEs and their attached routers can be implemented in the dynamic region. The maximum FFT size that each design variation can handle is given in Table II. With dynamic partial reconfiguration, time and power consumption for the reconfiguration process can be reduced significantly and the system can keep running during reconfiguration. For example, when the design is running with 4 PEs and the user wants to upgrade to 8 PEs, only 4 additional PEs are configured. This can save half of the time and power consumption required to

TABLE I
COMMUNICATIONS AMONG PES IN EACH STAGE

| Stage | Communications among PEs |
|---|---|
| $log_2(N/P)$ | $PE_i \rightleftharpoons PE_{i+1}$ $(i = 2j, 0 \leq j \leq 7)$ |
| $log_2(N/P)+1$ | $PE_i \rightleftharpoons PE_{i+2}$ $(i = 0, 1, 4, 5, 8, 9, 12, 13)$ |
| $log_2(N/P)+2$ | $PE_i \rightleftharpoons PE_{i+4}$ $(i = 0, 1, 2, 3, 8, 9, 10, 11)$ |
| $log_2(N/P)+3$ | $PE_i \rightleftharpoons PE_{i+8}$ $(0 \leq i \leq 7)$ |

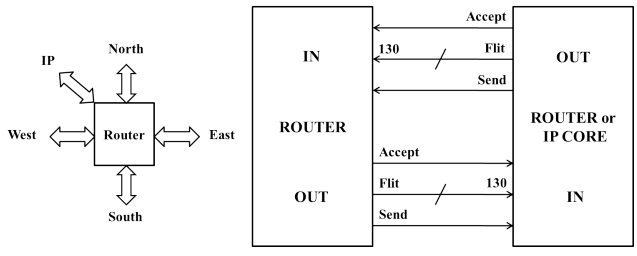


Figure. 4. Router datapath and interface.

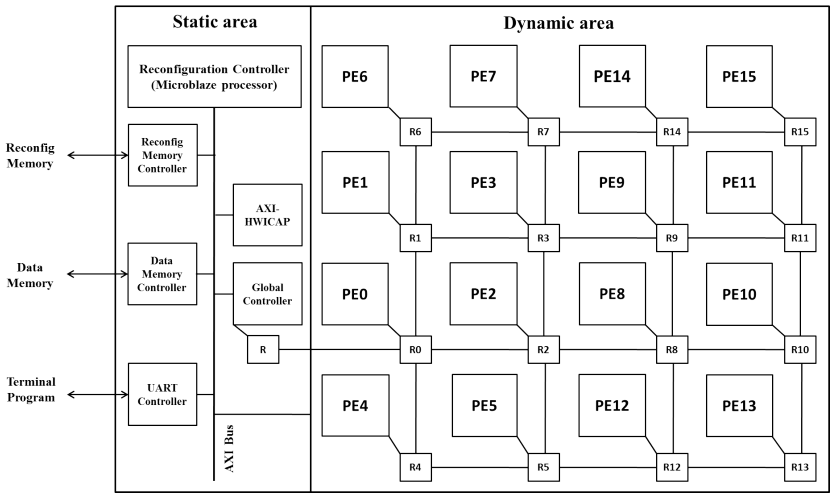


Figure. 5. FFT architecture.

fully configure a system with 8 PEs. In addition, the 4-PE design can keep running until the 8-PE design is completely configured. It is much simpler when the design is changed from a large number of PEs to a smaller one. In this case, the unused PEs are reconfigured with blanking bitstreams.

The static area contains a configuration controller, a configuration port, interfaces with external memories, a serial interface with an external host, and the global controller. The reconfiguration controller is responsible for loading the partial configuration bitstreams stored in the configuration memory through the configuration port. We chose to implement it using a MicroBlaze soft-core processor on the FPGA. The configuration port transmits the configuration data to the assigned region. The internal configuration access port (ICAP) primitive is used in this case since it provides the access to the configuration logic of the FPGA from within the FPGA fabric. The terminal program with the external host allows users to select the number of running PEs. The global controller is responsible for data scheduling and informing PEs when the design is changed.

## IV. Implementation Results and Design Performance

### A. Implementation Results

The design has been implemented in Verilog and verified using Modelsim. Synthesis and power analysis were performed with the Xilinx ISE Design Suite, targeting a Xilinx Virtex-7 XC7VX485T FPGA. Table III shows the results. Synthesis and power analysis of static-only FFT systems are also performed for comparison. Static-only FFT systems here are five design variations (1, 2, 4, 8, 16 PEs) based on point-to-point connections only. Figure 6 compares the power consumption of five differently sized static configurations with that of the new dynamically reconfigurable design with different numbers of active PEs.

Without dynamic partial reconfiguration, the user can choose between two options. The first one is using a fixed design of 16 PEs to perform all FFT sizes. This option is simple and does not require hardware reconfiguration but is not power-efficient since the biggest design is used for all FFT sizes. It can be seen from Figure 6 that the power consumption of the static 16-PE design is only smaller than the dynamic 16-PE design and much higher than the rest all four dynamic design variations.

The second option is using a static design appropriate for each FFT size. The system is fully reconfigured when changing between design variations. With this option, each design variation is smaller and more power-efficient than the corresponding variation with dynamic partial reconfiguration. However, the time and power consumption for reconfigura-

TABLE II
Maximum FFT Size of Each Design Variation

| Design Variation | Maximum FFT size |
|---|---|
| 1 PE | 8192 ($2^{13}$) |
| 2 PEs | 16384 ($2^{14}$) |
| 4 PEs | 32768 ($2^{15}$) |
| 8 PEs | 65536 ($2^{16}$) |
| 16 PEs | 131072 ($2^{17}$) |

TABLE III
Synthesis and Power Analysis Results

| | A single PE | A single router |
|---|---|---|
| Number of Slice Registers | 8739 | 758 |
| Number of Slice LUTs | 13022 | 1442 |
| Number of DSP48E1s slices | 36 | 0 |
| Number of 36Kb Block RAMs | 62 | 0 |
| Maximum Frequency | 364.458MHz | 522.575MHz |
| Estimated Power at 200MHz | 1.418W | 0.443W |

 ISBN: 978-1-61208-496-1 

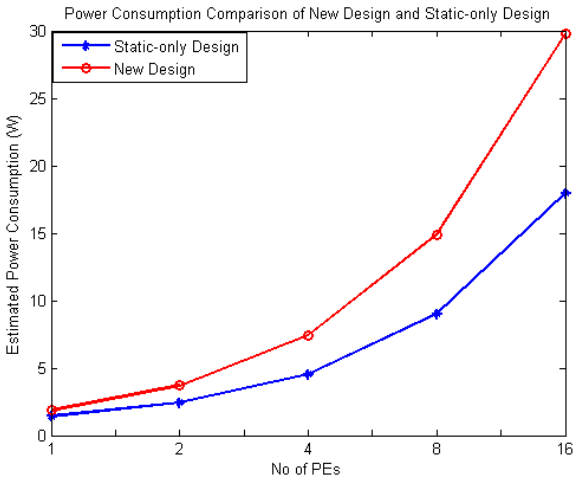


Figure. 6. Power comparison of new design and static-only design

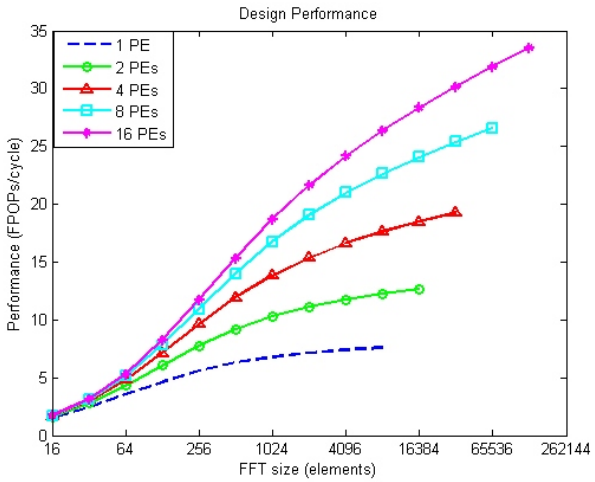


Figure. 7. Performance comparison of different design variations.

tion process is higher, which is a disadvantage for real-time applications.

Partial reconfiguration is performed using Xilinx PlanAhead. The partial bitstream of a single PE is 1212000 Bytes and that of a single router is 132512 Bytes. If the ICAP interface is 32-bit wide and clocked at 100 MHz, the configuration time of a single PE and a single router is 3.03 ms and 332 $\mu$s respectively. The partial reconfiguration time of $n$ PEs is 3.362$n$ ms.

### B. Design Performance

The total number calculation cycles of the dynamic system is slightly higher than the static system due to the NoC latency. The NoC latency (L) can be calculated as:

$$L = S + I + T \quad (1)$$

Here, $S$ is the link set-up time. It depends on the hop number ($h$) of the link. For each hop, it takes two cycles to process a Send signal and one cycle to process an Accept signal, hence $S = 3h$. The second term, $I$, is the initial latency, which is equivalent to the hop number. The last term, $T$, is the transfer time, which is the largest contribution to the NoC latency. The transfer time is equivalent to the number of flits in each packet. However, it is overlapped with the calculation time. Therefore, the NoC latency in this case is only $4h$, which is insignificant compared to the total number of calculation cycles.

The performance of the design in terms of floating point operations per cycle (FPOPs/cycle) with different numbers of PEs is as shown in Figure 7. Computing an $N$-point FFT takes $5Nlog_2N$ floating-point operations. With the number of calculation cycles ($C$) obtained from simulation, the performance of the design is $5Nlog_2N/C$ (FPOPs/cycle). It can be seen that the performance increases with an increase in the number of running PEs and the FFT size. The highest performance of 33.5 FPOPs/cycle is achieved by the 16-PE design with the 131072-point FFT. However, there is an insignificant difference in the performance of different designs when the FFT size is smaller than 256 points. This difference increases with larger FFT size. In other words, for small FFT size, a small number of PEs should be used. This will ensure power efficiency while obtaining satisfactory performance. For larger FFTs, users can select a design between different power-performance operating points.

A performance comparison in terms of calculation time between two design variations (1 PE and 16 PEs) running at 200 MHz and an Intel i7-3.4GHz CPU running Matlab R2014 is as shown in Figure 8. The calculation time of the Matlab program is determined using the 'tic' and 'toc' functions of Matlab. It can be seen that the calculation time of both design variations is much smaller that of the i7 CPU. The dynamically reconfigurable 16-PE design finishes the 131072-point FFT in 1.66 ms while the i7 CPU takes more than 6 ms. This proves that FPGAs can outperform software implementation on a general purpose processors for the FFT.

The performance of the design is also compared with the Xilinx FFT IP core [16] in terms of calculation cycles. The Xilinx FFT IP core supports the transformed size of $N = 2^m$ $(3 \leq m \leq 16)$, data precision of 8 to 34 bits, and fixed-point and block floating-point data format. The core provides four architectures: Radix-2, Radix-2 Lite, Radix-4 and Pipelined Radix-2. The Radix-2 Lite and Radix-4 architectures are not considered here because they have different butterfly structures, either lighter or heavier than the Radix-2.

The new 1-PE design has similar performance to that of the Xilinx Radix-2 FFT core. For example, the 1-PE design processes a 1024-point FFT in 7538 cycles; and the Xilinx Radix-2 FFT core needs 7367 cycles. The performance comparison between the dynamically reconfigurable 16-PE design and the Xilinx Pipelined Radix-2 FFT core is shown in Figure 9. However in making this comparison it should be noted that the number of PEs in the Xilinx Pipelined Radix-2 architecture in this case is smaller than 16. For example, only 13 PEs are used in the Pipelined Radix-2 architecture to perform the 8192-

 ISBN: 978-1-61208-496-1 

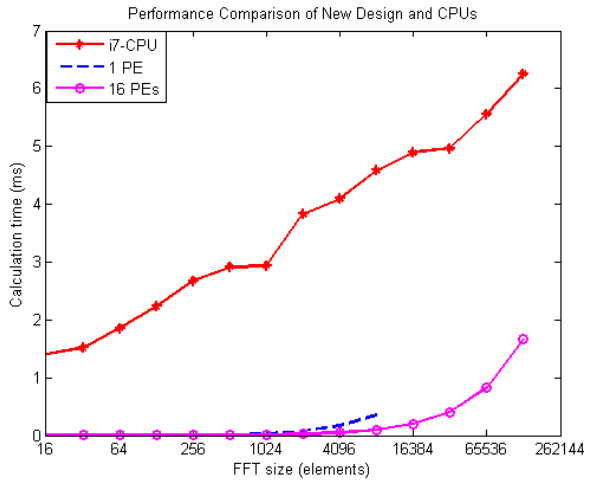


Figure. 8. Performance comparison of new design and CPUs.

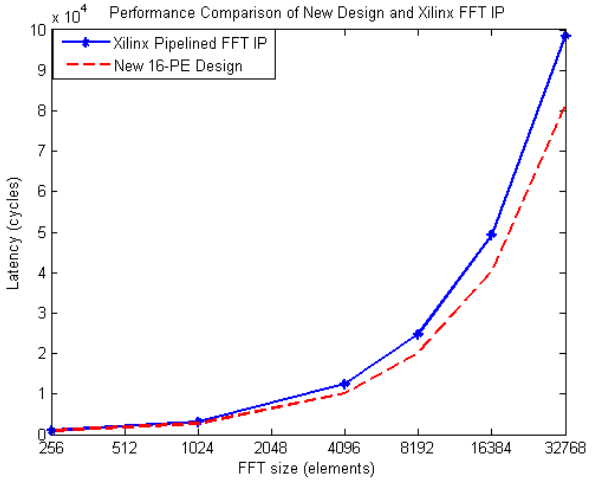


Figure. 9. Performance comparison of new design and Xilinx FFT IP.

point FFT. This comparison aims to illustrate the advantage of the parallel architecture over the pipelined architecture, in which more than $n$ PEs can be used to perform a $2^n$-point FFT to achieve smaller latency.

## V. Conclusion

In this paper, we have presented a practical system using a dynamically reconfigurable network on chip. Our implementation demonstrates that dynamically reconfigurable NoCs are feasible and can deliver power and performance benefits. In this case power is saved by only implementing sufficient hardware resources to meet current processing requirements. We have also shown how a dynamically reconfigurable NoC can be realized within the constraints of existing FPGA technology.

Our system, a double-precision floating-point FFT unit using network-on-chip communication and dynamic partial reconfiguration has been implemented on a Xilinx Virtex-7 XC7VX485T FPGA. The design provides much better performance than i7-3.4GHz CPUs running Matlab and competitive performance with static-only FFT systems and a Xilinx FFT IP core but with double-precision floating-point data format and the ability to adapt power-performance to suit the current workload. It can be concluded that the use of a dynamically partially reconfigurable NoC is a feasible and potentially beneficial solution for systems with all homogeneous PEs like the FFT system. This allows the system to scale performance at an acceptable cost of additional on-chip hardware. However, effectively exploiting dynamically reconfigurable systems requires a change in design practice. Designers conventionally craft algorithms to make good utilization of a fixed set of processing resources. Future work to demonstrate the benefit of dynamic reconfiguration will need to begin with algorithms redesigned for the new approach, which can take advantages of a changing set of heterogeneous processing elements.

## References

[1] Xilinx, "Partial Reconfiguration User Guide, UG702, V14.5," 2013.

[2] C. Bobda *et al.*, "DyNoC: A Dynamic Infrastructure for Communication in Dynamically Reconfugurable Devices," in International Conference on Field Programmable Logic and Applications 2005, pp. 153–158.

[3] S. Jovanovic, C. Tanougast, S. Weber, and C. Bobda, "CuNoC: A Scalable Dynamic NoC for Dynamically Reconfigurable FPGAs," in International Conference on Field Programmable Logic and Applications 2007, pp. 753–756.

[4] T. Pionteck, C. Albrecht, R. Koch, and E. Maehle, "Adaptive Communication Architecture for Runtime Reconfigurablle System-on-Chips," Parallel Processing Letters, vol. 18, no. 02, 2008, pp. 275–289.

[5] Y. E. Krasteva, E. de la Torre, and T. Riesgo, "Reconfigurable Networks on Chip: DRNoC Architecture," Journal of Systems Architecture, vol. 56, no. 7, 2010, pp. 293 – 302. Special Issue on HW/SW Co-Design: Systems and Networks on Chip.

[6] L. Devaux and S. Pillement, "OCEAN, A Flexible Adaptive Network-on-Chip for Dynamic Applications," Microprocessors and Microsystems, vol. 38, no. 4, 2014, pp. 337 – 357. Selected Papers from Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2012).

[7] E. O. Brigham, The Fast Fourier Transform and Its Applications. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[8] J. Palmer and B. Nelson, "A Parallel FFT Architecture for FPGAs," in Field Programmable Logic and Application (J. Becker, M. Platzner, and S. Vernalde, eds.), vol. 3203 of Lecture Notes in Computer Science, 2004, pp. 948–953, Springer Berlin Heidelberg.

[9] S. He and M. Torkelson, "Design and Implementation of a 1024-Point Pipeline FFT Processor," in Proceedings of the IEEE Custom Integrated Circuits Conference 1998, pp. 131–134.

[10] Y. Jung, H. Yoon, and J. Kim, "New Efficient FFT Algorithm and Pipeline Implementation Results for OFDM/DMT Applications," IEEE Transactions on Consumer Electronics, vol. 49, no. 1, Feb 2003, pp. 14–20.

[11] M. Garrido, J. Grajal, M. Sanchez, and O. Gustafsson, "Pipelined Radix-$2^k$ Feedforward FFT Architectures," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 1, Jan 2013, pp. 23–32.

[12] Y. N. Chang and K. Parhi, "An Efficient Pipelined FFT Architecture," IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 50, no. 6, June 2003, pp. 322–325.

[13] M. Ayinala and K. Parhi, "Parallel Pipelined FFT Architectures with Reduced Number of Delays," in Proceedings of the Great Lakes Symposium on VLSI 2012, GLSVLSI '12, pp. 63–66, ACM.

[14] J. You and S. Wong, "Serial-Parallel FFT Array Processor," IEEE Transactions on Signal Processing, vol. 41, no. 3, Mar 1993, pp. 1472–1476.

[15] Z. Jovanovic and V. Milutinovic, "FPGA Accelerator for Floating-Point Matrix Multiplication," Computers Digital Techniques, IET, vol. 6, no. 4, July 2012, pp. 249–256.

[16] Xilinx, "LogiCORE IP Fast Fourier Transform v8.0," 2012.

 ISBN: 978-1-61208-496-1 