IET Journals
The Institution of Engineering and Technology

Compact and High-throughput Parameterizable Architectures for Memory-based FFT Algorithms

ISSN 1751-8644
doi: 0000000000
www.ietdl.org

Daniel Valencia[1*], *Amirhossein Alimohammad*[1]

[1] *Department of Electrical and Computer Engineering, San Diego State University, 5500 Campanile Drive, San Diego, U.S.A*
* *E-mail: dlvalencia@sdsu.edu*

Abstract: Designers have to carefully choose the best-suited fast Fourier transform (FFT) algorithm among various available techniques for the custom implementation that meets their design requirements, such as throughput, latency, and area. This article, to the best of our knowledge, is the first to present a compact and yet high-throughput parameterizable hardware architecture for implementing different discrete FFT algorithms, including radix-2, radix-4, radix-8, mixed-radix, and split-radix algorithms. These algorithms are especially suitable candidates for compact hardware realizations due to their regular structures and symmetries properties in their signal flow graphs. The designed architectures are fully-parameterizable to support a variety of transform lengths (powers of 2, 4, 8) and variable word-lengths. The FFT algorithms have been modeled and simulated in double-precision floating-point and fixed-point representations using our custom-developed library of numerical operations. The designed synthesizable FFT architectures are modeled in Verilog hardware description language and their cycle-accurate and bit-true simulation results are verified against their fixed-point simulation models. The characteristics and implementation results of various FFT architectures on a Xilinx Virtex-7 FPGA. Compared to recently published works, our memory-based FFT architectures utilize less reconfigurable resources while maintaining comparable or higher operating frequencies. The ASIC implementation results in a standard 45-nm CMOS technology are also presented for the designed memory-based FFT architectures. The execution times of FFTs on a workstation and a graphics processing unit are compared against our FPGA implementations.

1 Introduction

Designers have to carefully choose the best-suited fast Fourier transform (FFT) algorithms among numerous available techniques for custom implementations that meet their design requirements, such as throughput, latency, and area. The FFT algorithm, presented by James Cooley and John Tukey [1], is a simple yet very efficient alternative to computing the discrete Fourier transform (DFT) directly. The Cooley-Tukey FFT was effectively able to reduce the computational complexity of the DFT from N^2 to only $N\log(N)$, where N denotes the transform length. The Cooley-Tukey algorithm is quite versatile and can be combined in many different ways to compute the DFT for any transform length. These radix-based FFTs are used in a wide variety of digital signal processing applications, in particular the radix-2 and radix-4 algorithms. Cooley-Tukey generalized the FFT, with support for any radix-r. In this way, one could mix radices, as is done in the mixed-radix algorithm, and use the Cooley-Tukey FFT on transforms whose lengths are not powers of r. There have been other variants on the Cooley-Tukey algorithm, such as the split-radix FFT by Duhamel and Hollmann [2], which combines the benefits of both radix-2 and radix-4 FFTs. The split-radix FFT can also be extended to support different combinations of radices.

Various hardware architectures have been proposed for implementing FFT algorithms [3] [4] [5] [6] [7] [8] [9] [10] [11]. However, many of the previously published designs are either large and high-throughput or relatively small, but rather low-throughput. In applications where silicon area is an important concern, such as neural signal processing, a low-throughput FFT may not meet the target throughput. Additionally, while some recent designs have focused on FFT optimizations, such as twiddle factor generation [12] or memory address generation [13], they fail to produce small, yet high-throughput designs. To the best of our knowledge, this is the first work presenting a compact and yet high-throughput parameterizable hardware architecture for implementing different discrete FFT algorithms, including radix-2, radix-4, radix-8, mixed-radix, and split-radix algorithms. The rest of this article is organized as follows. The reconfigurable architecture of radix-2, radix-4, radix-8, mixed-radix, and split-radix FFT algorithms are presented in Sections 2 to 6, respectively. The characteristics and implementation results of each architecture on a Xilinx Virtex-7 field-programmable gate array (FPGA) are presented. The bit-true and cycle-accurate simulation results of hardware implementation of FFT algorithms are compared against their fixed-point simulation results. The ASIC implementation results of radix-2, radix-4, radix-8, mixed-radix, and split-radix FFTs in a standard 45-nm CMOS technology are presented. Comparison remarks are made in Section 7. Finally, Section 8 makes some concluding remarks.

2 Radix-2 FFT

In general, a radix-m algorithm performs computations that involve m inputs and produces m outputs at a time. This is the source of potential performance increase when several elements are calculated simultaneously, compared to one at a time with the discrete Fourier transform algorithm. With m inputs, m equations for each of the outputs are required. These equations can be derived by solving for $z_1, z_2, \ldots, z_m$ as follows:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} W_m^0 & W_m^0 & W_m^0 & \cdots & W_m^0 \\ W_m^0 & W_m^1 & W_m^2 & \cdots & W_m^{m-1} \\ W_m^0 & W_m^2 & W_m^4 & \cdots & W_m^{2(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W_m^0 & W_m^{m-1} & W_m^{2(m-1)} & \vdots & W_m^{(m-1)^2} \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ \vdots \\ x \end{bmatrix}, \quad (1)$$

Table 1 Cooley-Tukey radix-r characteristic parameters for transforms of length N.

Parameter	Description	Equation
STGS	Number of stages	$\log_r N$
BS	Number of butterfly sets in current stage	r^{s-1}
k	Increment value of k in Eq. (2)	r^{s-1}
OFF	Spacing between inputs to the butterflies	N/r^s
BPS	Butterflies per set	N/r^s
IND	Starting index of each set	$(SetCounter-1)(\frac{N}{r^{s-1}})$

where W_m^k, known as the twiddle factor and is defined as:

$$W_m^k = e^{\frac{-2\pi ik}{m}}, \tag{2}$$

where k is a value that begins at zero and is incremented by a step defined by the current stage of the algorithm, and a, b, c, ..., are the m inputs to the butterfly. The butterfly structure is the fundamental component of the Cooley-Tukey algorithms. The Cooley-Tukey signal flow graphs (SFGs) of an N-point radix-r FFT have regular and symmetric structures and hence, they can be described by the characteristic parameters listed in Table 1. The characteristic equations in Table 1 can be verified by examining the SFG shown in Fig. 1.

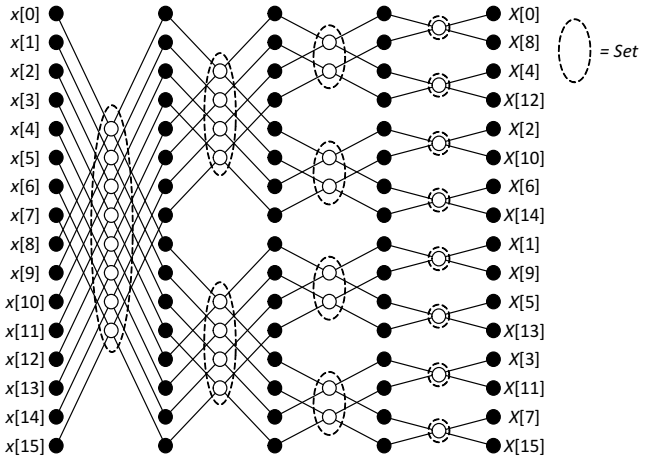


Fig. 1: 16-point radix-2 signal flow graph.

Fig. 2(a) shows the hardware architecture of the radix-2 butterfly. It accepts six inputs, i.e., the real and imaginary parts of a, b, and W_N^k. The dotted lines show the cutset lines where the pipeline registers will be added at the crossing points with the datapath signals. The pipelining technique is used for minimizing the critical path delay of FFT SFGs by inserting registers between combinational data paths while maintaining the input/output characteristics of the design [14]. A cutset intersects a set of edges of an SFG such that if these edges are removed from the graph, the graph becomes disjoint. In cutset pipelining, a register is added at the intersection of a cutset line with an edge in the SFG.

While one could instantiate the butterfly module to directly implement a radix-2 DIF SFG, this approach becomes infeasible for a relatively large transform length. For a compact implementation, butterfly units are time-multiplexed in the radix-2 FFT datapath, as shown in Fig. 2(b). While only two of these butterfly units are instantiated, however, one could use any 2^n number of butterfly units, up to half the number of data points. The datapath in Fig. 2(b) consists of a main memory unit, two processing elements (butterfly units), and registers and multiplexers. The memory unit of the datapath is parameterizable for variable depths and widths (i.e., various transform lengths and wordlengths). All other components in the datapath are also parameterizable to adjust the wordlength to the desired precision. The select lines of the multiplexers, enable inputs of the data memory and registers, and the read/write ports of the data memory

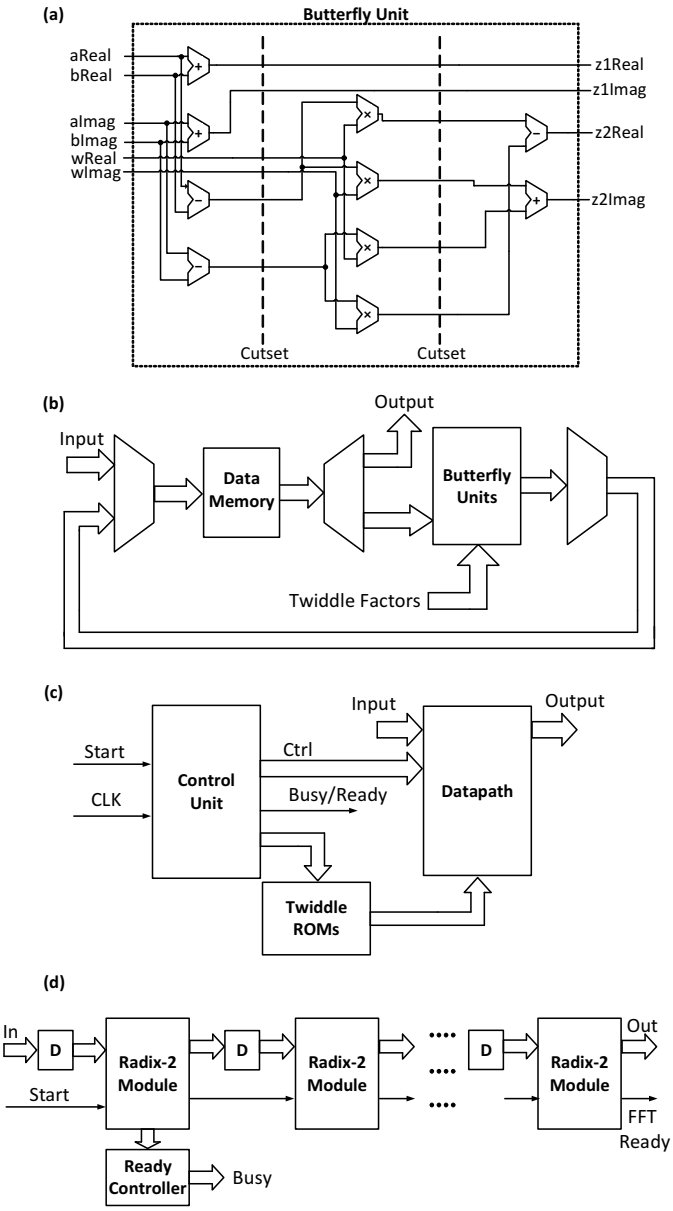


Fig. 2: (a) The datapath of the pipelined radix-2 butterfly, (b) the FFT architecture using time-multiplexed butterfly units, (c) the high-level block diagram of the radix-2 FFT module, and (d) cascaded radix-2 modules for implementing an N-point FFT.

are all driven by the control unit. This enables utilizing the same datapath and allows the control unit to change the behavior based on the current stage. Additionally, this general structure can easily be modified when using a different radix FFT.

The high-level block diagram of the radix-2 FFT module is shown in Fig. 2(c). To avoid using more processing elements, the twiddle factors are precomputed and stored in a read-only memory (ROM). The twiddle factors are represented using 16 bits, with 14 bits

reserved for the fractional part to provide a maximum absolute error of about 8.5×10^{-5}. Thus, for a 1024-point radix-2 FFT, the ROM size is 8.192 Kb. The control unit finite state machine (FSM) which drives the hardware has the following parameters: *NUM_POINTS* indicates the number of data points; *NUM_STAGE* indicates the number of the current stage of the FFT; *ADDRWIDTH* indicates the minimum number of bits required for the address of the memory unit; and *ROMADDRWIDTH* indicates the minimum number of bits required for the read address of the twiddle factor ROM. The parameter *NUM_STAGE* allows a local parameter to compute the *BS* of the current stage, as well as computing the *OFF* and *BPS*.

The control unit for the radix-2 FFT architecture is shown in Fig. 3. The FSM which drives the control unit begins and remains in state *Read Inputs/Idle* until an input signal *Start* is asserted high. Once the signal *Start* has been asserted, the memory unit is enabled for writing, and the write address of the memory is set to an internal counter, that counts from 0 to $N-1$. After this internal counter has counted N inputs, the module can begin performing the current stage of the FFT. Also, to be able to store the butterfly data back into the memory, the select line of the multiplexer *Source Select* is set such that the data will feed back into the memory and will no longer be coming from the input port. The FSM keeps track of how many sets have been performed in the state *Check Set Counter*, and keeps track of how many butterfly computations have been performed in the state *Check Butterfly Counter*. If an internal variable *setCounter* has counted to the number of sets defined earlier, then the current stage of the FFT has been completed. If an internal *Butterfly counter* has counted to the number of butterfly computations within the current set, then the *setCounter* is incremented and checked. To perform a butterfly computation, two inputs are passed to the butterfly in the state *Read Mem*. To read and write data properly, the FSM sets the select lines *Register Destination* and *ROM Destination* accordingly. The state *Enable Regs* enables registers to store the values being read from the memories, as they will later propagate through the butterfly units. Once all of the values are in their registers, the state advances to *Collect Butterfly Output*, where registers at the output of the butterfly units are enabled to store the result. These values are then written one at a time (i.e., real and imaginary) back into the memory in the state *Write Mem*, where the multiplexer select line *Out Destination* is set to read the correct pair of real and imaginary data. After the state *Write Mem*, the *Butterfly Counter* is incremented and the FSM returns to the state *Check Butterfly Counter*. Once all computations have been performed, the memory is enabled for reading and the memory contents are passed to the output in the state *Send Output*. In the state *Send Output*, an output signal *FFT Ready* goes high to indicate to the next stage or module that data will begin to stream from the output ports. Additionally, when the FFT is being processed, there is an output signal *Busy* to indicate that the module is currently processing data. When the output signal *Busy* is low in the state *Read Inputs/Idle*, this informs a wrapper that the next set of data inputs can be received.

There is a special exception made for the final stage of the radix-2 FFT, as there are no longer twiddle factor multiplications. Thus, the ROMs, the demultiplexers, and the output registers are removed from the high-level module and from the basic datapath. The butterfly unit itself is also now smaller, as the multipliers are no longer required. Omitting complex multipliers simplifies the butterfly unit to just 4 adders/subtractors. The FSM is also adjusted to remove the unnecessary output signals to enable the ROM and to set the read address. Also, to receive the outputs in the correct order, the read address is bit-reversed. To do so, a module is placed in between the control unit's output *Memory Read Address* and the datapath's input *Memory Read Address*. Once the signal *FFT Ready* is high, the module will reverse the bits and send the memory contents to the output ports in the correct order.

After connecting the control unit to the datapath, these modules are instantiated and cascaded as shown in Fig. 2(d), with each of the modules computing a single stage. However, the control unit takes one clock cycle to accept the first input as it awaits the start signal to be asserted high. To account for this one clock delay, registers are placed between the outputs and inputs of each stage. Also, some rest

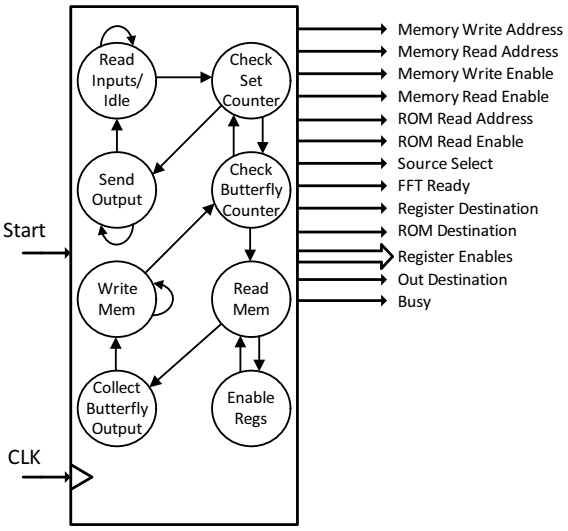


Fig. 3: Control unit of the radix-2 FFT.

time is alloted for the second stage to read its values before the first stage receives its next set of inputs. Thus, the output signal *Busy* is connected to an FSM called the *Ready Controller*, which simply waits for the negative edge of the signal *Busy* and asserts its own *Busy* output *NUM_POINTS* number of clock cycles later. When the *Ready Controller's* output *Busy* goes low, the first stage module is once-again ready to receive a new FFT. The benefit of cascading multiple radix-r modules is that there is no need for a conflict-free memory access scheme and its associated address generators [15], as the memory elements are local to their particular stage module. This also allows processing up to $log_r N$ FFTs at a time with pipelining. Additionally, compared to the traditional single-delay commutator (SDC) or single-delay feedback (SDF) schemes, this hybrid memory-based approach is readily scalable when modifying the transform length, especially with regards to the delay registers used in the SDC and SDF growing by powers of 2 in every stage. However, is a more complex control unit for routing of data.

To choose an acceptable representation and sufficient number of bits for signals, we measure the mean square error (MSE) of the fixed-point output values against the floating-point results. Figure 4 shows the MSE of the output values for the designed FFT algorithms for wordlengths ranging from 10 bits to 22 bits.

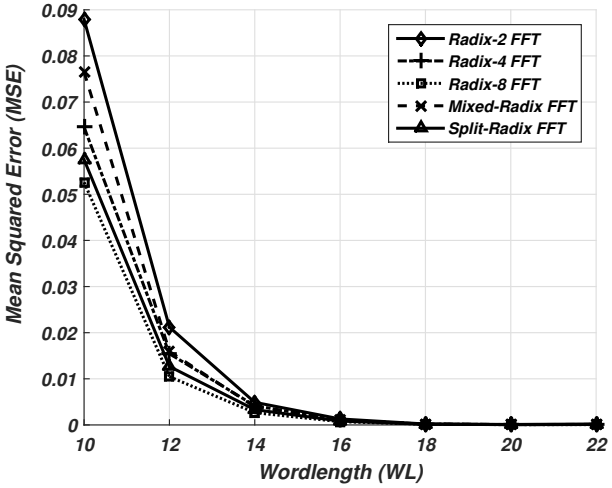


Fig. 4: Mean-squared error of FFT algorithms for various wordlengths.

Table 2 gives the characteristic and implementation results of the 1024-point radix-2 FFT on a Xilinx Virtex-7 xc7vx690tffg1157-2 FPGA for a variety of wordlengths, where (*WL*, *WF*) denotes the wordlength *WL* and the number of bits reserved for the fractional part *WF*. The latency of the 1024-point radix-2 FFT is 56870 clock cycles for the first set of FFT outputs. Due to serial processing of data, 20480 clock cycles are used for input/output (I/O) buffering, and 36390 clock cycles are used for processing. If there are multiple data sets being processed due to the pipelined design, it will take 6663 clock cycles for each subsequent set of FFT outputs.

Table 2 Characteristics and implementation results of 1024-point radix-2 FFT for different wordlengths.

(WL,WF)	Regs. (%)	LUTs. (%)	DSP48s. (%)	Freq. (MHz)
(16, 11)	5786 (0.67)	5800 (1.34)	72 (2.0)	317
(20, 10)	6794 (0.78)	6548 (1.51)	72 (2.0)	302

3 Radix-4 FFT

Based on the radix-2 algorithm, the radix-4 algorithm performs the computation of two radix-2 butterflies, although a bit differently. The radix-4 butterfly accepts four inputs and produces four outputs. An important difference of the radix-4 algorithm is that the number of points N must be an integer power of 4. However, similar rules apply when defining generalized parameters that govern the SFG behavior as given in Table 1 for $r = 4$.

The hardware implementation of the butterfly unit is trivial, as the multiplication by an imaginary simply causes the real and imaginary parts to swap and switch signs. While only a single complex multiplication is required in the radix-2 butterfly datapaths, three complex multiplications are required in the radix-4 butterfly unit. Thus, a pipelined complex multiplier was implemented, as shown in Fig. 5 (a). An alternative datapath for the complex multiplication using 3 multipliers is shown in Fig. 5 (b). However, this complex multiplier is not utilized as it would add extra latency to our design if the critical path is maintained to that of a single multiplier. Fig. 6 shows the architecture of the pipelined radix-4 butterfly using cut-set registers. Note the three complex multipliers are shown by the X^* symbol. The rightmost cutset shown in Fig. 6 is added to account for the internal one clock latency of the pipelined complex multiplier. Once all inputs are ready, the butterfly has a latency of four clock cycles.

The datapath for the radix-4 algorithm uses the same general structure of the radix-2 FFT shown in Fig. 2(b); the main changes consist of radix-4 butterfly units and more registers at the input and output of the butterfly units. Also larger multiplexers are required to direct the data into the corresponding registers. There is an exception made for the final stage as the multiplications by twiddle factors are no longer required, and hence the ROM and additional logic elements required to support it are removed from the final stage. The datapath is driven by the control unit, which also has a similar diagram as the one shown in Fig. 3. The main difference, however, is that the control signals adhere to the radix-4 SFG equations described at the beginning of this section. With the control unit connected to the datapath, stage modules are cascaded, essentially the same as was done for the radix-2 FFT. However, in the final stage, a *Digit Reversal* module is inserted between the control unit's read address output and the datapath's read address input to produce the outputs in the correct order.

Table 3 gives the characteristic and implementation results of a 1024-point radix-4 FFT on the same Xilinx Virtex-7 FPGA. The key advantage of the radix-4 FFT is for N points it takes significantly fewer clock cycles compared to the radix-2 FFT. For $N = 1024$, our radix-4 implementation has a latency of 29118 clock cycles. Due to serial processing of data, 10240 clock cycles are used for I/O buffering, and 18278 clock cycles are used for processing. If

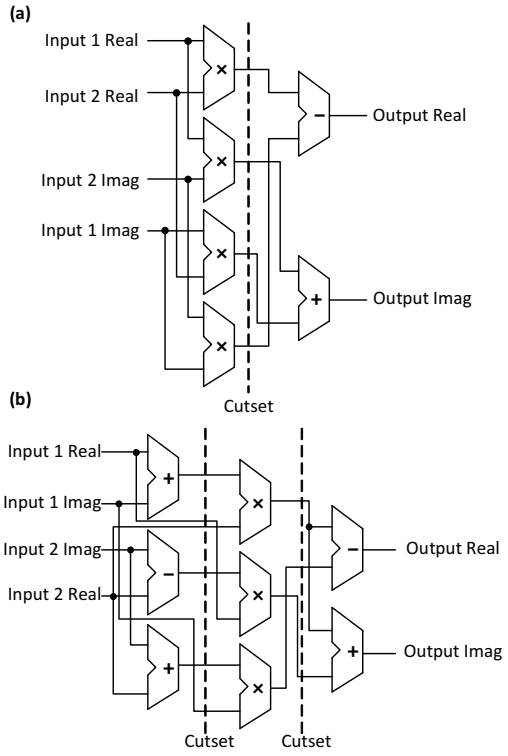


Fig. 5: (a) The datapath of the pipelined complex multiplier using 4 multipliers and 2 adders and (b) the datapath of the pipelined complex multiplier using 3 multipliers and 5 adders.

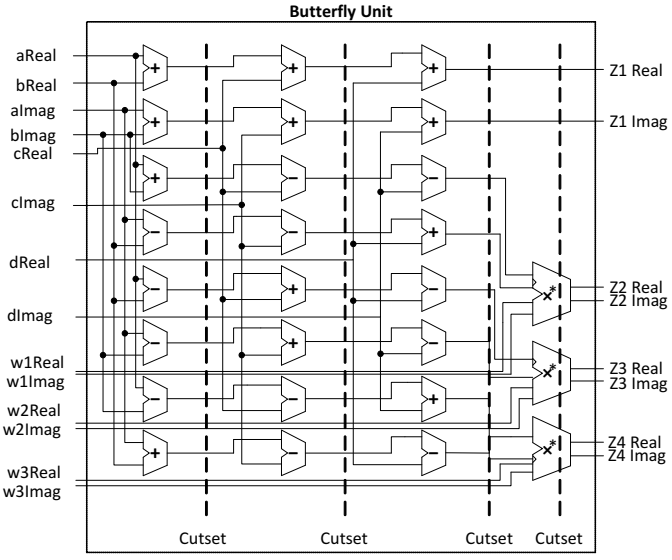


Fig. 6: Datapath of the pipelined radix-4 butterfly.

there are multiple data sets to process, each additional FFT will take 6918 clock cycles to complete. Even though there are fewer stages, $\left(\log_4 N \text{ vs. } \log_2 N\right)$, each butterfly computation has to perform three non-trivial multiplications, which results in a slightly larger design and also the error may be slightly larger than the error using radix-2 FFT.

4 Radix-8 FFT

Unlike the equations of radices 2 and 4, radix-8 uses nontrivial multiplications, in addition to the twiddle factor multiplications.

Table 3 Characteristics and implementation results of 1024-point radix-4 FFT for various wordlengths.

(WL,WF)	Regs. (%)	LUTs. (%)	DSP48s. (%)	Freq. (MHz)
(16, 11)	9517 (1.10)	7247 (1.67)	96 (2.6)	317
(20, 10)	11205 (1.29)	8640 (1.99)	96 (2.6)	302

There are different ways to approach this challenge from the hardware perspective. The simplest solution would be to include these extra multipliers within the butterfly unit itself. However, this would require 16 complex multipliers, in addition to the 7 already needed for multiplications with W^k through W^{7k}. We propose to create a coefficient-multiplier unit that consists of four pipelined complex multipliers, which would be time-multiplexed. The datapath and control units also account for the added latency. Because the coefficient-multiplier unit has already computed the products, they are simply treated as inputs to the butterfly. The coefficients themselves, which do not change regardless of the transform length, are obtained by solving Eq. (1) for $m = 8$ and are stored in internal registers.

The parameters for describing the behavior of the radix-8 FFT are essentially equal to those for radix-2 and radix-4 in Sections 2 and 3, respectively, but for $r = 8$ for Table 1. More multipliers are required in radix-8 FFT and for a compact implementation, a single butterfly unit is time-multiplexed. Using a single butterfly also means that only a single coefficient multiplier is required. Not only does this reduce the number of adders and multipliers utilized, the total number of registers is also decreased, as well as the size of the multiplexers, from 16-to-1 to 8-to-1. The radix-8 FFT architecture is similar to the ones for the radix-2 and radix-4 FFTs, shown in Fig. 2(c). A major change, aside from the singular butterfly unit, is the additional de-multiplexers and registers at the output of the coefficient multiplier. This is why only a single butterfly unit is utilized, as the design would be substantially larger otherwise with only marginal decreases in processing time. The final stage also includes the digit reversal module. These modules are cascaded in a similar fashion as was done for the radix-2 FFT shown in Fig. 2(d).

Table 4 gives the characteristic and implementation results for a 4096-point radix-8 FFT (1024 is not an integer power of 8) on the same Xilinx Virtex-7 FPGA. The latency is 131234 clock cycles for the first FFT output. Due to serial processing of data, 32768 clock cycles are used for input/output I/O buffering, and 98466 clock cycles are used for processing. If there are multiple data sets being processed, the latency is 36870 clock cycles for each subsequent FFT output. This latency is for a transform consisting of 4 times as many points as those in Tables 2 and 3 and is a result of a trade-off between having a more compact design and a larger and higher-throughput design. One could add a second butterfly unit for reducing the latency, at the cost of a larger design.

Table 4 Characteristics and implementation results of 4096-point radix-8 FFT for various wordlengths.

(WL,WF)	Regs. (%)	LUTs. (%)	DSP48s. (%)	Freq. (MHz)
(16, 11)	16033 (1.85)	14100 (3.25)	100 (2.7)	317
(20, 10)	19816 (2.29)	17609 (4.06)	100 (2.7)	302

5 Mixed-Radix FFT

One restriction of the radix-4 and radix-8 algorithms is that they cannot be used for all powers of two, as they require lengths that are integer powers of four and eight, respectively. One solution is to mix the radices; that is, using different radix algorithms in different stages of the FFT. One decides which radix to use in a particular stage by factorizing N. One can choose to factor N into prime numbers, but our approach is to reuse our verified radix-2, 4, and 8 modules. For example, one may use only a radix-2 FFT for a 32 point transform or factor 32 as 4×8 and perform a 32-point FFT in only two stages: the first stage is processed using a radix-4 FFT and the second stage is processed using a radix-8 FFT. Note that the previously discussed FFTs were all implemented using a single radix, such as radix-2, radix-4, and radix-8, which allows us to use the generalized equations given in Table 1. However, because mixed-radix FFTs utilize stages of different radices, two important variables, ***BS***, which denotes the number of butterfly sets per stage, and ***BPS***, which denotes the number of butterflies per butterfly set, must use updated equations. Wang et. al. [16] presented the following formulas to compute the values for ***BS*** and ***BPS***:

$$BS = \frac{\prod_{m=0}^{i} R_m}{R_i} \qquad (0 \leq i \leq s-1),$$

$$BPS = \frac{\prod_{m=i}^{s-1} R_m}{R_i} \qquad (0 \leq i \leq s-1),$$

where R is an array containing the factors, R_m and R_i are the m-th and i-th elements of that array, respectively, and s is the the number of elements in R. The number of elements in R is also equivalent to the number of stages. In the 32-point example, R consists of $[4, 8]$ and listed are the following ***Sets*** and ***BPS*** values per stage:

Stage	Sets	BPS
1 (Radix-4)	1	8
2 (Radix-8)	4	1

The previously defined ***OFF*** remains equal to ***BPS***, while the increment of the twiddle factors is defined as:

$$\frac{N}{r \times BPS},$$

where r is the number of the current radix.

To implement the mixed-radix FFT architecture, the radix-2, radix-4, and radix-8 FFT architectures are utilized and cascaded, as shown in Fig. 2(d) but each stage of the mixed-radix FFT is performed using a different radix-r stage module. The order of radices in the mixed-radix algorithms is not chosen arbitrarily. A 1024-point FFT can be implemented with a number of different radix combinations. For example, one can choose $\big(8 \times 8 \times 4 \times 2 \times 2\big)$, $\big(2 \times 8 \times 4 \times 2 \times 8\big)$, or $\big(4 \times 4 \times 8 \times 8\big)$. For our 1024-point FFT, we chose the order of $\big(2 \times 4 \times 4 \times 4 \times 8\big)$. Firstly, because the radix-2 and radix-4 FFT modules are time-multiplexing two butterfly units each, they do not have to wait for the higher latency radix-8 FFT to finish. Additionally, we use the radix-8 FFT in the final stage to avoid the extra multipliers, as the final stage of the mixed-radix algorithm does not involve multiplications with twiddle factors. The lack of multipliers also decreases the clock latency of the radix-8 butterfly unit, as the clock delay of the complex multiplications is no longer present.

Table 5 gives the characteristic and implementation results of a 1024-point mixed-radix FFT on the same Xilinx Virtex-7 FPGA. Our implementation has a latency of 31594 clock cycles for the first FFT output. Due to serial processing of data, 10240 clock cycles are used for I/O buffering, and 21354 clock cycles are used for processing. If there are multiple data sets being processed, there is a latency of 8200 clock cycles for each subsequent set of FFT outputs.

Table 5 Characteristics and implementation results of 1024-point mixed-radix FFT for various wordlengths.

(WL,WF)	Regs. (%)	LUTs. (%)	DSP48s. (%)	Freq. (MHz)
(16, 11)	10126 (1.17)	8050 (1.86)	84 (2.3)	317
(20, 10)	11262 (1.30)	8986 (2.07)	84 (2.3)	302

6 Split-Radix FFT

The split-radix FFT algorithm presents an efficient way to combine the benefits of the radix-2 and radix-4 FFT algorithms. The butterfly for the split-radix algorithm, as shown in Fig. 7, is similar to the radix-4 butterfly, but it is different in how its equations are defined:

$$z_1 = a + c,$$

$$z_2 = b + d,$$
$$z_3 = \big((a-c) - j(b-d)\big)W_N^k,$$
$$z_4 = \big((a-c) + j(b-d)\big)W_N^{3k}.$$

Note that the bottom half of the butterfly processes one more "step"

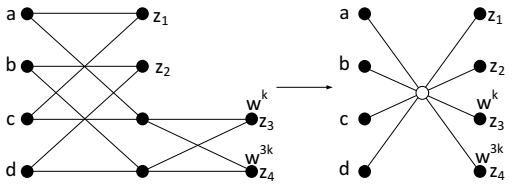
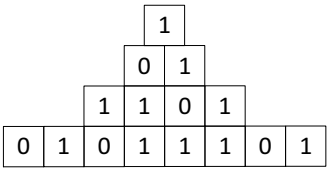


Fig. 7: Original and simplified split-radix butterfly.

than the top half, thus giving the split-radix SFG it's signature 'L' shape. Due to it's asymmetry, it is challenging to develop generalized equations to describe important parameters, such as the number of *Sets* and the starting index of each particular set. The other equations still apply, such as the number of stages defined as $\log_2 N$; *OFF* and *BPS* are given by $N/2^{s+1}$, where s denotes the number of the current stage, which ranges from 0 to $\log_2 N$; and the twiddle factor increment which is 2^{s-1}. Note that the number of sets in each stage is defined by the Jacobsthal integer sequence, with the presumption that the first stage always consists of a single set. This forms the beginning of a binary tree as 1, [0,1], [1,1,0,1], [0,1,0,1,1,1,0,1], etc. Each 1 represents a set of butterflies; additionally, each 1 will generate another 1 but also a 0, which indicates how half of the sets in the next stage are skipped due to the extra processing of the bottom half of the butterfy. Each 0, however, will generate two 1s. Thus, the gen-



erating formula of the Jacobsthal integer sequence given by Sloane [17] is used to compute the number of sets:

$$NumSets = \frac{2^s - (-1)^s}{3},$$

where s is the number of the current stage, from 1 to $\log_2 N$.

With the number of sets computed, the starting indeces for each set of butterflies are precomputed and stored. First assume that we already have the binary Jacobsthal sequence of any given stage, in this case stage 3, of an N point split-radix FFT as $M = [1011]$. We then iterate through the M array. If the current element is a 1, this means that there is a set and it will have an index given by:

$$index = (n-1) \times (\frac{N}{2^{s-1}}),$$

where n is the indexing variable. The calculated *index* is added to an array that will contain the indeces for that particular stage. As it is not a trivial method to compute the indeces, they are simply precomputed and stored in a block memory.

The control unit and datapath are similar to that of the radix-8, with the main difference being that the indeces are now stored in a separate memory. Additionally, there is no need for a coefficient multiplier, as the multiplications by j are trivial. With the control unit and datapath connected, we simply cascade as many modules as necessary to compute the desired N-point FFT. As opposed to mixed-radix FFT, the outputs of the split-radix FFT will be in bit-reversed order and hence, a bit-reversal module is inserted between the control unit and datapath in the final stage to produce in-order outputs.

Table 6 gives the characteristic and implementation results of our 1024-point split-radix FFT on the same Xilinx Virtex-7 FPGA. Our design has a latency of 54662 clock cycles for the first output. Due to serial processing of data, 20480 clock cycles are used for I/O buffering, and 34182 clock cycles are used for processing. If there are multiple data sets being processed, each subsequent FFT output will have a latency of 8198 clock cycles. This latency could be further lowered by multiplexing more butterfly units. This would come at the cost of a more complex control unit, larger multiplexers, and more registers. The only possible overhead is that the number of sets does not follow a regular order, and one or more of the extra butterfly units might remain unused in certain stages.

Table 6 Characteristics and implementation results of 1024-point split-radix FFT for various wordlengths.

(WL,WF)	Regs. (%)	LUTs. (%)	DSP48s. (%)	Freq. (MHz)
(16, 11)	6184 (0.71)	6392 (1.48)	64 (1.7)	317
(20, 10)	7192 (0.83)	7397 (1.71)	64 (1.7)	302

7 Comparisons and suggestions

Table 7 gives the characteristics and implementation results of our FFT architectures and several relevant published work. The radix-2 FFT developed in [3] has directly implemented the SFG for 16 and 32 points. As a result, a large number of butterfly units is required and hence, a relatively larger number of on-chip DSP48s is used. Our radix-2 realizations use fewer DSP48s, due to time-multiplexed butterfly units rather than using multiple instances. However, due to pipelining of the FFT stages, slightly more registers are required. The authors of [4] present a 64-point Radix-2^2 FFT processor. For a fair comparison, we have implemented the same transform length FFT on the same Virtex-7 FPGA. Unfortunately, the authors do not mention their input and output wordlengths, nor their output throughput. However, their emphasis is for high-speed applications. We can see that our memory-based approach utilizes fewer reconfigurable resources while maintaining a higher operating frequency. The design in [5] presents a 64-point radix-4 FFT processor in which a butterfly unit is time-multiplexed. We also compare our designs with Xilinx's commercial FFT IP core [6]. As Xilinx offers several implementations, we have chosen the FFT IP core in Radix-2 Burst I/O mode, which also uses a memory-based, iterative approach. While the design is smaller than our cascaded architecture, the computational latency is longer, with 7311 clock cycles. The mixed-radix FFT presented in [7] uses radix-25, radix-16, and radix-9 FFTs implemented using Winograd Fourier transform (WFT) algorithms for 2-, 3-, 4-, and 5-point DFTs. Unfortunately, they do not directly report the number of utilized registers, DSP48s, and signals' wordlengths in their manuscript. Uzun et al [8] have also presented memory-based designs for radix-2, radix-4, and split-radix FFTs. Their designs are built upon a common framework to ours, such that a butterfly unit is time-multiplexed and the FFT is computed in an iterative fashion. They have listed their device utilization for radix-2 and radix-4 in terms of the number of slice's utilized. Thus, we have estimated their register and look-up table (LUT) utilization

Table 7 Characteristics and the implementation results of various FFT realizations on different FPGAs.

Work	Algorithm	Device	N	WL	Regs.	LUTs.	DSP48s.	Freq. (MHz)	Throughput (MSamp/Sec)
[3]	Radix-2	Virtex-6	32	*input*: 16 *output*: 32	-	(*Slices*) 5620 (1%)	320 (37%)	19	-
Ours	Radix-2	Virtex-6	32	16	2657 (0.28%)	3552 (0.75%)	32 (3%)	285	285
Ours	Compact Radix-2	Virtex-6	32	16	579 (0.06%)	785 (0.16%)	8 (0.92%)	285	285
[4]	Radix-2^2	Virtex-7	64	-	538 (0.13%)	1345 (0.66%)	-	158	-
Ours	Compact Radix-2	Virtex-7	64	16	626 (0.15%)	848 (0.42%)	8 (0.71%)	335	335
[6]	Radix-2	Virtex-6	1024	16	4699 (0.49%)	6298 (1.32%)	16 (1.85%)	366	366
[8]	Radix-2	Virtex-II	1024	-	4240* (0.49%)	16960* (1.32%)	-	84	-
[9]	Radix-2^2	Cyclone-IV	1024	16	2453 (-%)	3295 (-%)	56 (-%)	231	231
Ours	Radix-2	Virtex-6	1024	16	5487 (0.58%)	6926 (1.46%)	72 (8%)	285	285
Ours	Compact Radix-2	Virtex-6	1024	16	626 (0.06%)	916 (0.19%)	9 (1.04%)	285	285
[5]	Radix-4	Virtex-II	64	16	-	(*Slices*) 3304 (9%)	12 (8%)	27	27
Ours	Radix-4	Virtex-6	64	16	5121 (5%)	5292 (11%)	48 (16%)	285	285
Ours	Compact Radix-4	Virtex-6	64	16	1992 (2%)	2138 (4%)	24 (8%)	260	260
[8]	Radix-4	Virtex-II	1024	-	8940* (0.49%)	35760* (1.32%)	-	80	-
Ours	Radix-4	Virtex-7	1024	16	9517 (1%)	7274 (2%)	96 (3%)	317	317
Ours	Compact Radix-4	Virtex-7	1024	16	2065 (1%)	1671 (1%)	25 (3%)	317	317
[10]	Radix-4	Virtex-5	4096	24	1163 (2%)	1407 (2.4%)	98 (15.3 %)	200	400
Ours	Compact Radix-4	Virtex-5	4096	24	2614 (4%)	3921 (6 %)	25 (3%)	167	167
Ours	Radix-8	Virtex-7	4096	16	16033 (1.85%)	14100 (3.25%)	100 (2.78%)	317	317
[7]	Mixed-Radix 25/16/9	Virtex-5	1296	-	-	7791 + 16016	-	122	122
Ours	Mixed-Radix 2/4/8	Virtex-6	1024	16	9814 (1%)	10297 (2%)	96 (11%)	285	285
[11]	Hybrid-Radix	Virtex-7	4-2048	16	8723 (2%)	37896 (18%)	56 (5%)	61.83	-
Ours	Compact Radix-2	Virtex-7	2048	16	668 (0.16%)	921 (0.45%)	9 (0.8%)	229	229

using the datasheet of their implementation platform, the Virtex-II FPGA. We have estimated that their designs use considerably more LUTs than our radix-2 and radix-4 realizations. This may be due to the fact that the Virtex-II FPGAs do not have on-chip DSP48s. However, we note that our radix-2 and radix-4 designs multiplex two butterfly units compared to their single butterfly unit. Thus, a more fair comparison would be our compact radix-2 and compact radix-4 designs with those of [8]. They have also reported their latency for radix-2 and radix-4 as 10214 clock cycles and 5440 clock cycles, respectively. Thus, our radix-2 design completes the FFT 3551 clock cycles earlier and our radix-4 design completes the FFT 1478 clock cycles later. Moreover, our designs can run at considerably higher operating frequencies. Unfortunately, they have not reported their device utilization for the split-radix FFT. The work in [9] presents the FPGA implementation of a Radix-2^2 FFT on a Cyclone-IV device. The Radix-2^2 algorithm has the multiplicative complexity of the radix-4 FFT yet retains the regular structure of the Radix-2 butterfly unit. They have also aimed to improve the performance from a frequency standpoint. It is shown that while their design utilizes fewer computational resources compared to our normal, ten-stage Radix-2 implementation, our design can operate at a higher operating frequency. Our compact Radix-2 implementation is significantly smaller than their design while operating at a higher maximum clock speed. Granted, the compact architecture has a higher latency than our normal, ten-stage design. Garrido et al [10] have also presented a memory-based radix-4 FFT for a 4096-point transform length. Their design consists of four memory banks and a single butterfly unit and twiddle factor multipliers. Four values are read simultaneously and a set of multiplexers are used to permute the inputs into the appropriate butterfly unit input port. An address memory generation unit is used for the select lines of the permutation multiplexers. As their design utilizes only one butterfly unit, we compare it to our compact radix-4 implementation, which we have synthesized on the same Virtex-5 device. While the design in [10] utilizes fewer registers, LUTs, and runs at a higher operating frequency, they also utilize more block RAM units (31 vs 6) and more DSP48E units (98 vs 25). The design in [11] presents a hybrid-radix approach with a design capable of implementing 36 different transform lengths with a two dimensional FIFO storage scheme. For a fair comparison, we have compared our Radix-2 design at a transform length of 2048 points, as that is their largest supported transform length. While their design has run-time flexibility, it is clear that our memory-based FFT architecture utilized significantly fewer reconfigurable resources on the same Virtex-7 FPGA, while achieving a higher clock frequency. The reduction in operating frequency is likely due to the amount of flexibility offered by their design.

While the designs for the computational unit and memory feedback structure are rather compact, the memory overhead that arises when pipelining the FFT stages remains an issue that can be further addressed. One approach is to exploit the symmetric property of the FFT's SFGs, such that only the first half of the inputs going into a butterfly *set* need to be stored, while the second half can be used as they appear on the input ports. The split-radix and radix-8 implementations have relatively long latencies, which is due to the time-multiplexing of a single butterfly unit. It would be preferable to have a parameterizable folding factor, such that the end user can make the choice between a lower latency, but relatively larger implementation or a more compact implementation but with a relatively larger latency. Nevertheless, our designed FFT modules have a relatively high-throughput as they produce latency, producing more than 300 million samples per second. At the cost of increased latency, we have also developed more compact architectures for the radix-2 and radix-4 FFTs by utilizing only a single "Radix-2 Module" or "Radix-4 Module", respectively. The characteristics and implementation results of these two versions can be seen on Table 7, listed as *Compact Radix-2* and *Compact Radix-4*. Some optimizations can also be made with regards to the twiddle factors, as not all stages will read the entire twiddle ROM values. Finally, the wordlength, in particular the integer wordlength of the signal representation, has a significant impact on the maximum frequency and the area of the FFT designs. As there is no data-scaling between stages, one can reduce the absolute error by increasing the wordlength when using the presented designs for large transform lengths.

Fig. 8(a) shows the execution times of the MATLAB simulation, Graphics processing unit (GPU) simulation of FFT algorithm developed in CUDA, and the FPGA simulation. The MATLAB simulation of FFT was executed on a 3.3 GHz Intel Core i5-3550 processor, with 8 GB of DDR3 RAM. The GPU host computer is a Nvidia Tesla M2090 featuring 512 CUDA cores, 1331 Gigaflops peak single precision floating point performance, 6GB global RAM. Hardware simulation is performed on a Xilinx Virtex-7 xc7vx690tffg1157-2 FPGA. The simulation results show that the memory-based FFT computations on a Virtex-7 FPGA are faster than on an Nvidia Tesla M2090 GPU for transform lengths smaller than 2^{20}.

Table 8 gives the power consumption and area utilization of a 32-point radix-2 FFT, and 64-point radix-4, radix-8, mixed-radix, and split-radix FFT architectures with a 1.1-V core supply voltage synthesized in a standard 45-nm CMOS technology. Synthesis was performed using Synopsys Design Compiler while we use Cadence SOC Encounter for place and route. All designs run at 317 MHz. Fig. 8(b) shows the chip layout of the 64-point radix-8 FFT architecture.

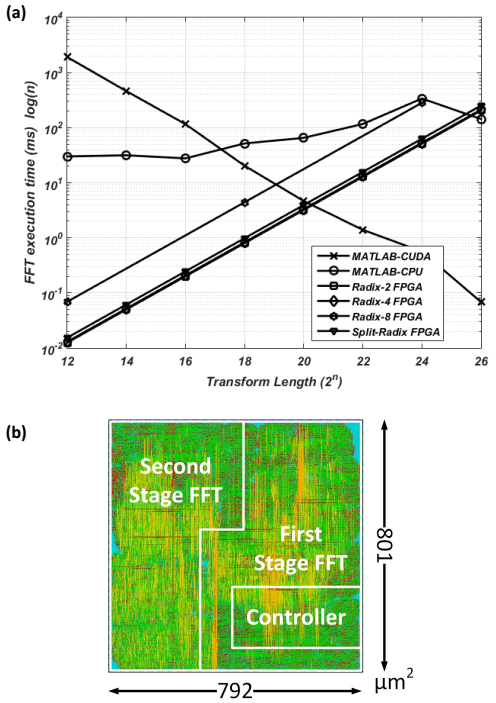


Fig. 8: (a) Execution times of the workstation, GPU, and FPGA implementations of FFT for different transform lengths and(b) the chip layout of the 64-point radix-8 FFT.

8 Conclusion

This article presented compact, yet high-throughput parameterizable hardware architectures for memory-based radix-2, radix-4, radix-8, mixed-radix, and split-radix fast Fourier transform (FFT) algorithms. The FFT architectures are parameterizable and can be synthesized for various transform lengths. The radix-2 architecture is the smallest design with regards to the number of lookup tables and registers. If the FFT modules will be used in a larger design where the number of DSP48 should remain relatively small, one can consider using the split-radix FFT or the compact architectures for radix-2 or radix-4 FFT. When latency is of the highest concern, one can use the radix-4 FFT at the cost of a design that is slightly larger than that of the radix-2 FFT, but completes an N-point FFT in almost half the number of clock cycles compared to radix-2. The limiting factor of the radix-8 FFT, aside from the computational complexity of its butterfly, is that the transform length must be an integer power of 8. One could avoid this restriction so long as N is a multiple of 8, however, the digit-reversal module would no longer be functional and sorting output values would not be trivial. The mixed-radix implementation could be used to a greater effect if some other prime radices are developed, such as radices 3, 5, and 7, allowing for a very wide range of transform lengths. The proposed compact and high-throughput reconfigurable FFT architectures were implemented on a field-programmable gate array (FPGA) and their characteristics and implementation results were presented and compared with the previously-published work. ASIC implementation results of FFT architectures in a standard 45-nm CMOS technology were also presented. Our simulation results showed that FFT computations on a Virtex-7 FPGA is faster than on an Nvidia Tesla M2090 GPU for transform lengths smaller than 2^{20}.

Table 8 Power consumption and area utilization of different FFT algorithms.

Design	N	Silicon Area (μm^2)	Total Power (mW)
Radix-2	32	590×590	10
Radix-4	64	715×708	18
Radix-8	64	801×792	23
Mixed-Radix	64	742×741	18
Split-Radix	64	731×731	16

9 Acknowledgment

This work was supported (in part) by the Center for Sensorimotor Neural Engineering (CSNE), a National Science Foundation Engineering Research Center (EEC-1028725).

10 References

1 Cooley, J.W., Tukey, J.W.: 'An algorithm for the machine calculation of complex Fourier series', *Mathematics of Computation*, 1965, **19**, (1), pp. 297–301
2 Duhamel, P., Hollmann, H.: 'Split radix FFT algorithm', *Electronics Letters*, 1984, **20**, (4), pp. 14–16
3 Saenz, J., Raygoza, J., Becerra, E., Cisneros, S., Dominguez, J.: 'FPGA design and implementation of radix-2 fast Fourier transform algorithm with 16 and 32 points.'. IEEE International Autumn Meeting on Power, Electronics and Computing, Ixtapa, Mexico, 2015. pp. 1–6
4 Bansal, M., Nakhate, S.: 'High speed pipelined 64-point FFT processor based on radix-2^2 for wireless LAN". International Conference on Signal Processing and Integrated Networks, Noida, India, 2017, pp. 607-612
5 Suleiman, I.: 'FPGA implementation of low power 64-point radix-4 FFT processor for OFDM system'. International Conference on Computers, Communications, & Signal Processing, Kuala Lumpur, Malaysia, 2005. pp. 278–281
6 'Xilinx Incorporation', http://www.xilinx.com/, accessed August 2018
7 Chen, J., Hu, J., Lee, S., Sobelman, G.: 'Hardware efficient mixed radix-25/16/9 FFT for LTE systems', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2015, **23**, (2), pp. 221 – 228
8 Uzun, I.S., Amira, A., Bouridane, A.: 'FPGA implementations of fast fourier transforms for real-time signal and image processing', *IEE Proceedings - Vision, Image and Signal Processing*, 2005, **152**, (3), pp. 283–296
9 Siu, T., Sham, C., Lau, F.: 'Operating frequency improvement on FPGA implementation of a pipeline large-FFT processor". International Conference on Advanced Communication Technology, Bongpyeong, South Korea, 2017, pp. 5–9
10 Garrido, M., Sanchez, M., Lopez Vallejo, M.L., Grajal, J.: 'A 4096-point radix-4 memory-based FFT using DSP slices', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017, **25**, (1), pp. 375–379
11 Shih, X., Chou, H.: 'Reconfigurable VLSI design of a changeable hybrid-radix FFT hardware architecture with 2D-FIFO storing structure for 3GPP LTE systems", *ICT Express*, 2018, **4**, (3), pp. 144-148
12 Yu, C., Lee, K., Kuo, C.: 'Low-complexity twiddle factor generator for FFT processors". International Conference on Consumer Electronics, Las Vegas, USA, 2017, pp. 350-351
13 Long, S., Hong, M., Shiue, M.: 'A low-complexity generalized memory addressing scheme for continous-flow fast Fourier transform". International Conference on Computer and Communication Systems, Nagoya, Japan, 2018, pp. 492-496
14 Parhi, K., Messerschmitt, D.G.: 'Pipeline interleaving and parallelism in recursive digital filters - Part I: Pipelining using scattered look-ahead and decomposition', *IEEE Transactions on Acoustics, Speech and Signal Processing*, 1989, **37**, (7), pp. 1099–1117
15 Ma, C., Chen, H., Yu, J., Long, T.: 'A novel conflict-free parallel memory access scheme for FFT constant geometry architectures', *Science China Information Sciences*, 2013, **56**, (4), pp. 1–9.
16 Wang, L., Zhou, X., Sobelman, G., Liu, R.: 'Generic mixed-radix FFT pruning', *IEEE Signal Processing Letters*, 2012, **19**, (3), pp. 167–170
17 'On-line Encyclopedia of Integer Sequences', https://oeis.org/A001045, accessed December 2018