ISSN XXXX XXXX © 2017 IJESC

Research Article Volume 7 Issue No.7

FPGA Implementation of Variable-Length Split-Radix FFT Algorithm

Erkan İnceöz[1], Enver Çavuş[2]
Researcher[1], Assistant Professor[2]
Department of Space Technologies Research Institute[1], Department of Electrical and Electronics Engineering[2]
The Scientific and Technological Research Council of Turkey[1]
Ankara Yıldırım Beyazıt University, Turkey[2]

Abstract:
In this study, an FPGA implementation of a programmable length Fast Fourier Transform (FFT) is presented. The proposed implementation uses a pipelined Split-Radix FFT structure to support FFT lengths from 8 to 1024 with real and complex inputs as fractional numbers. After verifying functionality of floating and fixed point models, variable length Split-Radix FFT algorithm is implemented on Kintex-7 FPGA and implementation results are compared with previously reported designs. The proposed implementation achieves computation time reductions from 68% to 90% compared to earlier studies.

Keywords: Fast Fourier Transform (FFT), FPGA, Inverse FFT, Split-Radix FFT.

I. INTRODUCTION

Discrete Fourier Transform (DFT) is one of the most widely used operation in digital signal processing algorithms such as array antenna analysis, multichannel carrier modulation, and video/image compression. An efficient implementation of DFT is Fast Fourier Transform (FFT), also known as Cooley-Tukey algorithm, which is developed at mid-1960s, and reduced the complexity of the DFT from $O(N^2)$ to O(NlogN). The Cooley-Tukey algorithm became known as Radix-2 Algorithm and it was followed with Radix-3, Radix-4 and Mixed-Radix algorithms. The further researches provided Fast Hartley Transform (FHT), Split-Radix FFT (SRFFT), Quick Fourier Transform (QFT) and Decimation-in-Time-Frequency (DITF) algorithms to compute the Fast Fourier Transform. One of the efficient methods for implementing FFT algorithm, which achieves the lowest number of arithmetic operations to compute an FFT, is the Split-Radix structure[1]. Split-Radix FFT algorithm reduces the complexity of computation by combining Radix-2 and Radix-4 algorithms. In literature, there are only a few number of studies about FPGA implementation of Split-Radix FFT algorithm. An FPGA implementation of a variable-length Split Radix FFT implementation[2] is reported with latencies varying from 3.6 µs to 448.8 µs for transforms lengths from 16 to 1024. There are some other studies of FPGA implementation of Split-Radix FFT algorithm[3-4] but they are not constructed as parametric for transform length and other variables. In another study, a variable-length FFT implementation[5], which does not use Split-Radix algorithm, has 200 MHz clock frequency for 8-point FFT. In this study, variable-length split-radix FFT algorithm, which works for real and complex inputs, is implemented on FPGA. The performance of the design is improved using the pipelining strategy that increases the throughput of the design by making all parts work collaterally instead of working sequentially. The proposed design is implemented on Kintex-7 FPGA and implementation results are compared with previously reported designs. The proposed implementation achieves computation time reductions from 68% to 90% compared to earlier studies.
The rest of paper is organized as follows. In Section II, Split-Radix FFT algorithm, C++ implementation of algorithm in floating point and fixed-point formats are defined. Fixed-point Verilog HDL model is described in Section III. Section IV discusses results of Verilog HDL model including resource usage, power consumption and timing analysis. Finally, Section V concludes the paper.

II. SPLIT-RADIX FFT ALGORTIHM

SRFFT uses the principle of independent computations of odd-indexed and even-indexed transform data. Radix-2 structure is used to compute even-indexed transform data whereas Radix-4 structure is used for odd-indexed transform data. Figure 1 shows the four blocks of SRFFT algorithm[6]; L-Shaped Butterfly, 2-Length Butterfly, Bit-Digit Reversal and Inverse Transform blocks.

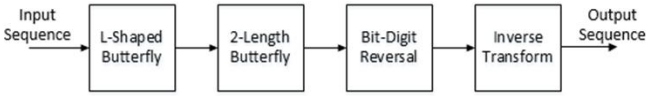


Figure.1. Split-Radix FFT (SRFFT) Algorithm

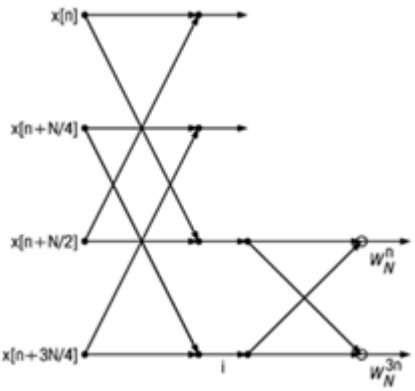
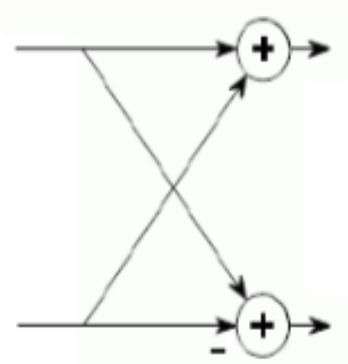


Figure.2. L-Shaped Butterfly Structure

Figure 2 describes the operations performed in L-Shaped Butterfly block. L-Shaped Butterfly takes 4 inputs at a time and computes 4 outputs for each 4 inputs by making some computations of additions, subtractions and complex multiplications. Pseudo code for the L-Shaped Butterfly Structure is shown in the below algorithm.

```
N2 := 2*N
for k from 1 to log2N do
  N2 := N2/2
  N4 := N2/4
  for t from 0 to N4-1 do
    ix := t
    id := 2*N2
    while ix < N-1
      i0 := ix
      while i0 < N do
        i1 := i0 + N4
        i2 := i1 + N4
        i3 := i2 + N4
        {r1,s1} := {x[i0]–x[i2],y[i0]-y[i2]}
        {r2,s2} := {x[i1]–x[i3],y[i1]–y[i3]}
        {x[i0],y[i0]} := {x[i0]+x[i2],y[i0]+y[i2]}
        {x[i1],y[i1]} := {x[i1]+x[i3],y[i1]+y[i3]}
        {r1,s3} := {r1+s2,r1-s2}
        {r2,s2} := {r2+s1,r2-s1}
        x[i2] := r1*cos(t*2*Pi/N2)-s2*sin(t*2*Pi/N2)
        y[i2] := -s2*cos(t*2*Pi/N2)–r1*sin(t*2*Pi/N2)
        x[i2] := s3*cos(3*t*2*Pi/N2)+r2*sin(3*t*2*Pi/N2)
        y[i2] := r2*cos(3*t*2*Pi/N2)–s3*sin(3*t*2*Pi/N2)
        i0 := i0 + id
      end while
      ix := 2*id – n2 + t
      id := 4*id
    end while
  end for
end for
```

Following structure in the Split-Radix FFT algorithm is 2-Length Butterfly Structure. This structure designed to compute an addition and subtraction of 2 inputs and produce 2 outputs as



in Figure 3.

Figure3. 2-Length Butterfly Structure

Pseudo code for the 2-Length Butterfly Structure is shown in the below algorithm.

```
ix := 1
id := 4
while ix < N do
  for i0 from ix-1 to N-id step id do
    i1 := i0 + 1
    {x[i0],x[i1]} := {x[i0]+x[i1],x[i0]-x[i1]}
    {y[i0],y[i1]} := {y[i0]+y[i1],y[i0]-y[i1]}
  end for
  ix := 2*id – 1
  id := 4*id
end while
```

The next structure in the Split-Radix FFT algorithm is the Bit-Digit Reversal structure. This structure is constructed with different algorithms[7-9]. The Bit-Digit Reversal algorithm is chosen after benchmarking of the algorithms. The Bit-Digit Reversal structure is illustrated in Figure 4 for an example of 4-bit.

Sample numbers in normal order		Sample numbers after bit reversal	
Decimal	*Binary*	*Decimal*	*Binary*
0	0000	0	0000
1	0001	8	1000
2	0010	4	0100
3	0011	12	1100
4	0100	2	0010
5	0101	10	1010
6	0110	6	0110
7	0111	14	1110
8	1000	1	0001
9	1001	9	1001
10	1010	5	0101
11	1011	13	1101
12	1100	3	0011
13	1101	11	1011
14	1110	7	0111
15	1111	15	1111

Figure.4. Bit-Digit Reversal Example of 4-Bit

Pseudo code for the Bit-Digit Reversal Structure is shown in the below algorithm.

```
N2 := N/2
N4 := N/4
N21 := N2 + 1
k := 0
for i from 0 to N2-2 step 2 do
  if (i < k) do
    swap{(x[i],y[i]),(x[k],y[k])}
    swap{(x[i+N21],y[i+N21]),(x[k+N21],y[k+N21])}
  end if
  swap { (x[i+1], y[i+1]), (x[k+N2], y[k+N2]) }
  t := N4
  while (t <= k) do
    k := k - t
    t := t/2
  end while
  k := k + t
end for
```

The last structure in the Split-Radix FFT algorithm is Inverse Transform structure. Inverse transform results are achieved from the forward transform results by making some combinations in the indices of output data and some division operations on output data when inverse transform is required.

III. IMPLEMENTATION OF SRFFT ALGORITHM

After floating point and fixed-point C++ models of SRFFT algorithm are finalized and verified, Verilog model is designed. The most important structural property of the SRFFT Verilog model is the parametric structure. The parametric structure provides to use same design for different size of transform data and different transform length with small parameter changes. The transform length and the input data width are parameterized in terms of integer bit size and fraction bit size because of the fact that data is in the fixed-point Qn.m format. All variables, signals and memory units are also parameterized properly to achieve a parametric structure. The sine and cosine values required to be used in the SRFFT algorithm are provided by look-up tables. This issue limits the parametric structure of SRFFT algorithm due to the fact that the creation of the look-up

tables must be in the design stage. The boundary of parametric design of the SRFFT algorithm is defined as 1024 for the transform length. All parameters, variables and memory units designed to support up to transform length of 1024. Sequential operations in the SRFFT algorithm are realized by a Finite State Machine (FSM) in the Verilog HDL, where each sequential operation is handled as a different state. A pipelining structure is utilized in the design, where each block in the SRFFT algorithm is implemented in a separate FSM and connected with pipeline registers. Pipelining provides that each FSM can operate at the same time instead of waiting for previous FSM to start to operate. Figure 5 illustrates the pipeline registers and FSM connections of the implemented structure. The first FSM takes the input frame serially and captures the input data in the memory. The next 3 FSMs implement L-Shaped Butterfly, 2-Length Butterfly, and Bit-Digit Reversal respectively. The last FSM is implemented to give output of the SRFFT algorithm by controlling forward or inverse transform to be applied. All FSMs are connected to each other with pipeline registers that allows them to work independently.

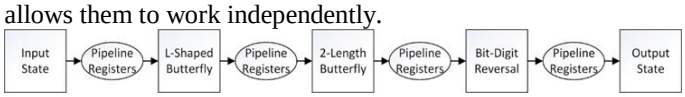


Figure.5.Pipelined Structure of Split-Radix FFT Algorithm

After SRFFT algorithm is designed in the FSM structure, the look-up tables for sine and cosine values is constructed. Firstly, required sine and cosine values for 1024-point FFT are described. For 1024-point FFT sine and cosine periods need to be sampled with 1024 sample. The algorithm needs 3/4 samples of sine period and 3/4 samples of cosine period. Instead of having two look-up tables for sine and cosine, the sine values written in the look-up table and cosine values are achieved by applying a shift in indexes which regards to the phase shift. The sine values are produced in format of Q2.15 in MATLAB and the look-up table is constructed with these values.

IV. IMPLEMENTATION RESULTS

After modelling Variable Length Split-Radix FFT algorithm in Verilog, the Verilog model is verified by matching the results of Verilog simulations with the fixed-point C++ model. The error in transformation of floating point to fixed-point model between Verilog results and the C++ floating point model results are also compared. The maximum absolute error between floating and fixed point results is 0.6%, whereas the mean of the error is obtained as 0.4%. The simulation of the Variable Length Split-Radix FFT algorithm for 16-point FFT is shown in the Figure 6.

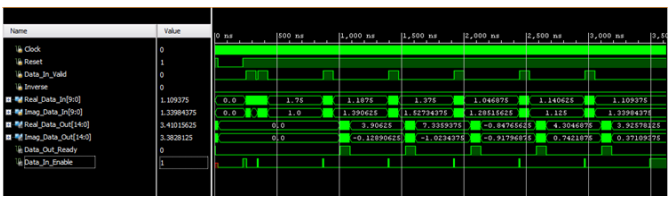


Figure.6. Simulation of Variable-Length Split-Radix FFT Algorithm

Simulations for the other transform lengths from 8 to 1024 are also done and the results of the simulations are verified. The latency of the algorithm is calculated from the simulations for each transform length and it is shown in the Table I. After the simulations, the Variable Length Split-Radix FFT algorithm is implemented on Xilinx Kintex-7 XC7K160T FPGA. After the implementation of the algorithm, the timing analysis is performed and the maximum clock frequency of the algorithm as the result of timing analysis for each transform length is obtained. Table I shows the latency (given in clock cycles), maximum clock frequency and the computation time for each transform length of the proposed implementation.

TABLE I. LATENCY, MAXIMUM CLOCK FREQUENCY AND COMPUTATION TIME OF PROPOSED DESIGN FOR DIFFERENT TRANSFORM LENGTHS

Transform Length	Latency (Clock Cycle)	Maximum Clock Frequency (MHZ)	Computation Time (μs)
8	43	300	0.143
16	105	277	0.378
32	241	243	0.988
64	555	217	2.553
128	1251	196	6.380
256	2797	178	15.663
512	6181	163	37.700
1024	13551	151	89.443

After the implementation of the algorithm, the resource usage and the power consumption of the algorithm on the Kintex-7 FPGA is obtained. The resource usage and the total power consumption for some transform lengths is shown in Table II.

TABLE II. RESOURCE USAGE AND POWER CONSUMPTION OF PROPOSED DESIGN

Transform Length	FF	LUT	BRAM Usage	DSP48 Usage	Total Power Consumption (mW)
8	1227	2747	0	8	230
16	2222	5685	0	8	318
32	4261	12150	0	8	510
64	8471	24467	0	8	881

A study of variable-length FFT implementation[5] has 200 MHz clock frequency for 8-point FFT whereas proposed design can support clock frequency up to 300 MHz for the 8-point FFT which means proposed design achieves 50% higher clock frequency for variable-length FFT. Table III compares the latency of proposed design in different FFT sizes with earlier reported design[2]. The proposed implementation achieves 90%, 68%, 83% and 80% latency reductions for 16, 32, 256 and 1024-point FFT sizes, respectively, as compared to the variable-length Split-Radix FFT implementation[2].

TABLE III. LATENCY COMPARISON (μS) OF PROPOSED DESIGN AND THE EARLIER DESIGN2 FOR DIFFERENT FFT SIZES

	16-Point FFT	32-Point FFT	256-Point FFT	1024-Point FFT
Earlier Design[2]	3.605	7.995	92.885	448.805
Proposed Design	0.378	2.553	15.663	89.443
Reduction in Latency	90%	68%	83%	80%

V. CONCLUSION

In this study, a Variable-Length Split-Radix FFT algorithm was implemented on FPGA. Firstly, the algorithm was constructed in C++ in floating point and the model was verified by MATLAB. Then the fixed-point model was

constructed in C++ and the results were compared and matched. As a last step, the fixed-point Split-Radix FFT algorithm was constructed in Verilog HDL. The results were exactly matched with the fixed-point C++ model for different transform length with different input data. The Verilog HDL model is implemented in Kintex-7 FPGA. The timing and resource analysis and optimizations of the model were done. The implemented Variable-Length Split-Radix FFT algorithm can be used with frequencies and latencies given in the Table I and resource usage and power consumption given in the Table II.

VI. REFERENCES

[1]. P. Duhamel and H. Hollmann, "'Split Radix' FFT Algorithm," Electron. Lett., vol. 20, no. 1, p. 14, 1984.

[2]. C. Watanabe, C. Silva, and J. Munoz, "Implementation of Split-Radix Fast Fourier Transform on FPGA," in 2010 VI Southern Programmable Logic Conference (SPL), 2010, pp. 167–170.

[3]. P. Xu and J. S. Chen, "FPGA Implementation of High Speed FFT Algorithm Based on Split-Radix," in 2008 International Symposium on Intelligent Information Technology Application Workshops, 2008, pp. 781–784.

[4]. M. R. Mohammadnia and L. Shannon, "Minimizing the Error: A study of the Implementation of an Integer Split-Radix FFT on an FPGA for Medical Imaging," in 2012 International Conference on Field-Programmable Technology, 2012, pp. 360–367.

[5]. V. Gautam, K. C. Ray, and P. Haddow, "Hardware Efficient Design of Variable Length FFT Processor," in 14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems, 2011, pp. 309–312.

[6]. J. Arndt, Matters Computational : Ideas, Algorithms, Source Code. (Springer, 2011).

[7]. J. J. Rodriguez, "An Improved FFT Digit-Reversal Algorithm," IEEE Trans. Acoust., vol. 37, no. 8, pp. 1298–1300, 1989.

[8]. A. A. Yong, "A Better FFT Bit-Reversal Algorithm without Tables," IEEE Trans. Signal Process., vol. 39, no. 10, pp. 2365–2367, 1991.

[9]. K. Drouiche, "A New Efficient Computational Algorithm for Bit Reversal Mapping," IEEE Trans. Signal Process., vol. 49, no. 1, pp. 251–254, 2001.