AN OPTIMIZED PIPELINE FFT PROCESSOR BASED ON FPGA

1ST NGO THANH NHAN
Integrated Circuit Design and Education Center (ICDREC) – Vietnam National University - Ho Chi Minh (VNU-HCM)
nhan.ngothanh@icdrec.edu.vn

2ND TRAN CONG DANH
Integrated Circuit Design and Education Center (ICDREC) – Vietnam National University - Ho Chi Minh (VNU-HCM)
danh.trancong@icdrec.edu.vn

3RD VU DUC LUNG
University of Information Technology, Vietnam National University Ho Chi Minh City
lungvd@uit.edu.vn

4TH NGUYEN THE DAI DUONG
Integrated Circuit Design and Education Center (ICDREC) – Vietnam National University - Ho Chi Minh (VNU-HCM)
ntdduong@icdrec.edu.vn

ABSTRACT
How to design and optimize a pipeline FFT Processor based on FPGA is presented. The architecture of FFT core is based on a new approach [1] using pipeline technique to increase the performance of the design. Following that, Radix-2^2 algorithm is chosen to implement the FFT processor. Some necessary techniques are mentioned. This is just the straightforward adjustment; however it improves the frequency, performance and saves resources considerably for the design.
KEY WORDS
FFT, FPGA, Radix-2^2 SDF, Pipelining architecture, optimization

1. INTRODUCTION

Fast Fourier Transform (FFT) has been used in many applications, especially in telecommunication. There was the number of approaches with different forms of algorithm also pipeline architectures for FFT design. Radix-2^2 is still the advantageous algorithm which has the liable multiplicative complexity as Radix-4 algorithm; moreover it keeps the butterfly structure of radix-2 algorithm. Due to the characteristics of FFT, performance always has high priority in design. Consequently, pipeline architecture is essential technique applied popularly in most of FFT design. In this paper, the supposed pipeline architecture [1] is considered and analyzed for more details about the advantages of this architecture in improving performance. In fact, the suitable form of algorithm and pipeline architectures are absolutely beneficial in making the design reach to the desirable targets with higher performance and lower resources. Even the most innovative techniques among, some optimized techniques could create significant results in the design. When design is implemented on FPGA, it is necessary to coding in the convenient way for the particular synthesis tool. It is definitely effective for the design. Besides, using necessary amount of registers in the correlative place helps to save considerable amount of resources in the design.

3. PIPELINE ARCHITECTURE RECOMMENDATION

Pipeline architecture is the enhance technique using to increase the performance of IP core. Besides, it helps the design operate more effective in real-time. In this section, the worthwhile pipeline architecture is introduced and analyzed dramatically.

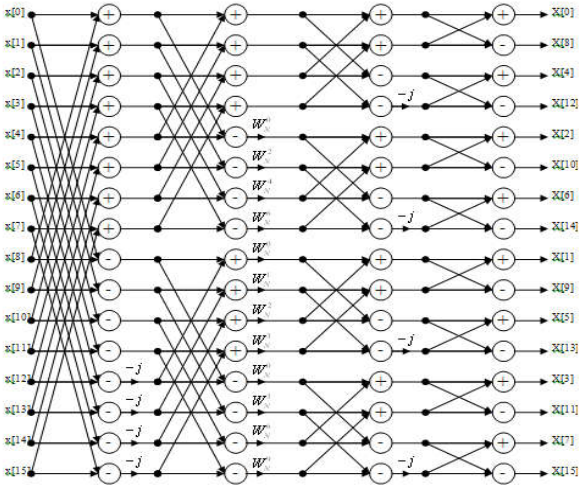


Fig.1: Dataflow of 16 points Radix-2^2 SDF

For instance, Fig.2 outlines the block diagram of 16 point FFT implemented following the Radix-2^2SDF algorithm, as shown in Fig.1. The implementation consists three main blocks. Two first blocks are used to compute two different kinds of butterflies called butterfly 1 and butterfly 2. The other block is multiplier used as trivial twiddle factor multiplication [3]. Between each block, the addition of pipeline registers breaks the critical paths. As a result, it improves the operating frequency for system.

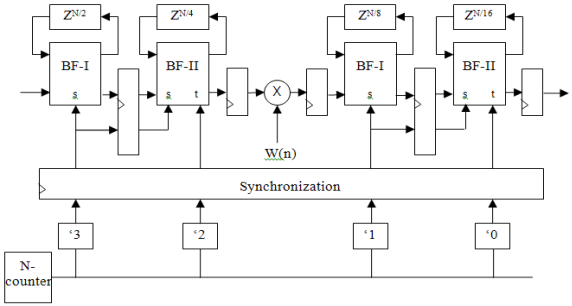
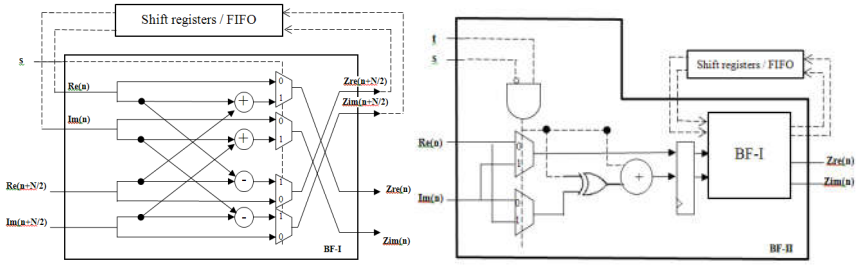


Fig.2: FFT pipeline architecture for N=16

Fig.3 show clearly the inside structure of two butterflies. It is easy to see that butterfly 1 and butterfly 2 have the same structure. Butterfly 2 is the duplication of butterfly 1 with the addition of the swapping combination circuit for some input signals. In this structure of butterfly 2, the swapping block separates with BF-I block by pipeline register. This fixes the critical path and improves operating frequency for system. The data supplying for FFT processor enters sequentially at input of butterfly 1. The first half of the data frame is

stored in shift registers. When the other half comes, the data stored in shift registers is loaded for implementing arithmetical calculations as addition and subtraction. For example, 8 first samples of 16 point FFT are stored in shift registers. As soon as the ninth sample comes, first sample in shift registers is loaded then the arithmetical computation between two samples is implemented. Butterfly 2 receives data directly from butterfly 1. Following the Radix-2^2 algorithm, as shown in Fig.1, butterfly 2 does the same work as butterfly 1 but with N/2 points. Due to the spatial regularity of the algorithm, the output from butterfly 2 after multiplying with suitable twiddle coefficients continues next stage with another butterfly 1, butterfly 2 and trivial twiddle multiplier until releasing the final result. After each stage, the number of point reduces to a quarter compared with the previous stage.



(a)BF-1 (b) BF-2

Fig.3: Structure of Butterfly 1 and Butterfly 2

To evaluate performance of the pipeline architecture, the waveform shown in Fig.4 clears this problem. The global controller includes 2 simple counters which cover all the FFT calculation process. The first counter considered the "discrete counter" is activated by the valid signal. After the first counter finishes its cycle, the second counter starts working and the flag is asserted to indicate that the core have received the entire package. Under high level of the flag signal, the second counter increases its value at every active clock edges. Because of the second counter's behavior, this counter is also considered as "continuous counter". The flag is de-asserted after the second counter finishes its cycle.

The number bits of first counter depend on the number points of FFT. For instance, as shown in Fig.2, the counter has 4 bits used to control the operation of 4 butterflies inside the 16 point FFT. In the waveform, these control signals derived from 4 bit counter are named *stg0_bf1_s, stg0_bf2_s, stg1_bf1_s, stg1_bf2_s*. Following that, upon high level of validity, data is stored in shift registers under low level of the switch signal and arithmetic calculation is come to fruition in the other level.

Following the wave form, the *valid_out* is ready one cycle before releasing output data. By the time *valid_out* is active in waveform, this means that as soon as all input data is sent to the FFT processor, output data is available. In reality, it needs to add the delay time caused by pipe registers used between butterflies and multiplication blocks. Therefore, the processing time could be evaluated to (number of points + number of pipeline registers)*cycle period.

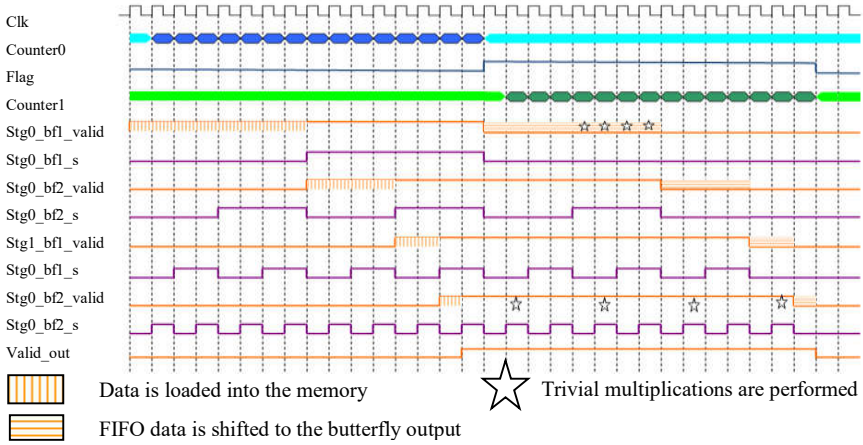


Fig.4: Dataflow of 16 points FFT

4. USING AVAILABLE MULTIPLIERS

Multiplier usually is the module that uses a lot of resources. The experience in writing RTL code is so important to make reduction when synthesizing the design. For the design based on FPGA, it is great to writing code using the available multipliers supported by FPGA vendors [4]. The reason is the way that synthetic tool does with a design depends so much on how designer codes their RTL.

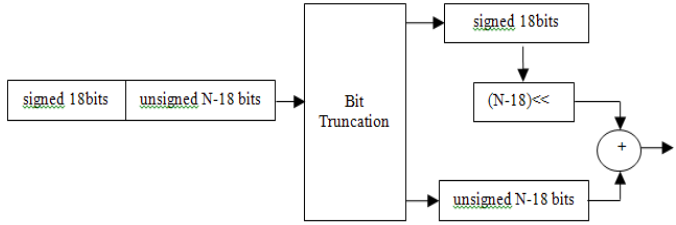


Fig.5: Bit truncation for greater than 18 bit number

For example, in order to using the 18x18 available multiplier, the operators of multiplier which bit-width are greater than the multiples of 18 must be split into parts which are less than 18. The method for bit truncation is shown in Fig.5. This solution is useful for the FPGA synthesis which the multipliers are completely implemented by the DSP block constructed by several 18 bit multipliers.

5. BIT GROWTH

For design FFT core, the presentation for data could be floating point or fixed point. With floating point, the width of data does not change after passing any butterflies. Besides, using floating point increases the accuracy of termination. However, calculation for floating point number needs particular processing block. This probably makes more serious critical path for design. Compared with floating point, using fixed point presentation makes designation

become easier. Therefore, this choice improves frequency of design. Nevertheless, the result gets a little bit lower in accuracy than one getting by using floating point presentation. On the other hand, with fixed point presentation, the data width will grow through the calculation. Data getting out of either butterfly or multiplier grows 1 bit, shown as Fig.6. Using function to calculate exactly the number of registers dedicated in each stage optimizes the design so much in resource.

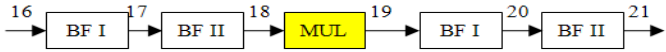


Fig.6: Bit grow in FFT stages.

6. EXPERIMENTAL RESULT

These proposed techniques have been applied in real design for 1024 point FFT with 16 bit data width, 16 bit twiddle width and using fixed point presentation. Below table shows the comparison result about the resources and frequency of our design compared with those from Altera FFT. The design is synthesized in Quartus with Altera FPGA CycloneIII.

Tab.1 Synthesis result compared between Altera FFT and proposed FFT

	LE (unit)	9-bit Mul (unit)	M9K (unit)	F-max (Hz)	Transform-time (us)
Altera FFT	5243	56	21	193	5.29
Proposed	4634	44	28	194	5.28

Synthesis result with EP3C10F256C6 indicates the frequency and transforming time between our FFT core and Altera FFT core is similar. However, our design has the advantage of optimizing resources.

7. CONCLUSION

In this paper, the effective pipeline FFT architecture is dramatically mentioned. Through that, it helps reader undergo not only inside structure but also operating principle of the architecture. Some other techniques are already discussed to make the design be more economical. There are such several methods to compute the twiddle value as using large ROM, using small ROM, multiplier-based or CORDIC. However, depending on the cost of different kinds of resources in an FPGA chip and goals of the design, designer decides to use the effective method. It is worth to work further in enhance techniques to make the design become more optimization.

REFERENCES

[1] Shousheng He and Mats Torkelson, "*A New Approach to Pipeline FFT Processor*", The 10th International, Honolulu (1996), ISBN 0-8186-7255-2.

[2] Jonas Claeson, "*Design and implementation of an asynchronous pipelined FFT processor*", Linköping (2003), LiTH-ISY-EX-3356-2003

[3] In-Cheol Park, WonHee Son, and Ji-Hoon Kim, "*Twiddle Factor Transformation for Pipelined FFT Processing*", ICCD 2007, ISBN 978-1-4244-1257-0

[4] "*Implementing Multipliers in FPGA Devices*", Altera Corp, July 2004, version 3.0.