ANALYSIS AND VALIDATION OF 8 POINT RADIX-2 TIME DECIMATION FFT ALGORITHM

Rafael Mallmann, Fernanda Kastensmidt
rafael-mallmann@uergs.edu.br, fglima@inf.ufrgs.br

Engenharia em Sistemas Digitais – UERGS
Estrada Santa Maria, 2300 – 92500-000 – Bairro Ramada – Guaíba – RS – Brazil

ABSTRACT

Here we present a pipelined implementation of 8 point radix-2 time decimation FFT algorithm to solve the Discrete Fourier Transform (DFT). The main goals of this paper are to discuss this FFT algorithm and design a digital circuit that leads to its solving. The project documentation consists in a datapath, an ASM flowchart and a finite state machine. Thus, software simulation can provide the DFT calculus in 52 clock cycles and one butterfly in 4 clock cycles.

1. INTRODUCTION

The FFT (Fast Fourier Transform) is used in digital systems, allowing a fast DFT calculation. Many devices that use digital signals (such as digital cameras, ADSL modems, mp3 players) have specific modules to make this calculation. With the advance of these systems we need dedicated hardware circuits that can even more solve that transform.

There are some ways to calculate the DFT, such as solving simultaneous linear equations or the correlation method. The Fast Fourier Transform (FFT) is another method for calculating the DFT. While it produces the same result as the other approaches, it is incredibly more efficient, often reducing the computation time by hundreds [2].

Thus, FFT is an efficient algorithm to compute the DFT and IDFT (inverse). The FFT is of great importance to a wide variety of applications, from digital signal processing and solving partial differential equations to algorithms for quickly multiplying large integers [1].

On the other hand, the development of embebed system has resulted in a tremendous potential for designing dedicated hardware for FFT calculus. This paper shows an algorithm project and a hardware implementation. Our approach uses 8 point radix-2 time decimation FFT. The algorithm has been coded and simulated in Scilab® [3]. After that, the circuit has been coded in VHDL and simulated in ModelSim®.

2. OUR FFT APROACH

The FFT operates in three steps. First it decomposes an N point time domain signal into N time domain signals, each one composed of a single point. The second step is to calculate the N frequency spectra corresponding to these N time domain signals. Lastly, the N spectra are synthesized into a single frequency spectrum [2].

The first algorithm step can be greatly simplified. The decomposition is nothing more than a reordering of the samples in the signal. The Figures 1 and 2 shows this process called Bit Reversal. The second step is also very simple. The frequency spectrum of 1 point signal is equal to itself. This means that nothing is required to do this step [3].

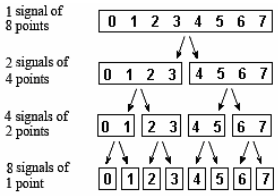

Figure 1 - 8-point time domain decomposing.

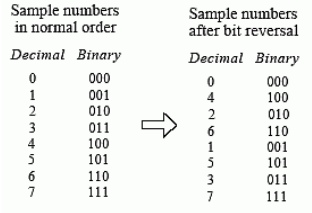

Figure 2 – Bit Reversal.

Now we describe the last computational step, the most difficult one. This step is to combine the N frequency spectra in the exact reverse order that the time domain decomposition took place. This is where the algorithm gets confused. Unfortunately, the bit reversal shortcut is not applicable, and we must go back one stage at a time. In the first stage, 8 frequency spectra (1 point each) are synthesized into 4 frequency spectra (2 points each). In the second stage, the 4 frequency spectra (2 points each) are synthesized into 2 frequency spectra (4 points each), and the last stage results are the output of the FFT, a 8 point frequency spectrum.

To synthesize the frequency spectra we introduce the simple flow diagram called butterfly (Figure 3). The butterfly is the basic computational element of the FFT which transforms two complex points into two other complex points. The equations below show the butterfly calculus:

$$A = x + jX$$

$$B = y + jY$$

$$Wn^k = Cos(\frac{2\Pi K}{N}) - jSin(\frac{2\Pi K}{N})$$

$$A' = A + BWn^k$$

$$B' = A - BWn^k$$

Substituting and separating in real part and imaginary part:

$$RA = RA + [RBCos(\frac{2\Pi K}{N})] + [IBSin(\frac{2\Pi K}{N})]$$

$$IA = IA - [IBCos(\frac{2\Pi K}{N})] - [RBSin(\frac{2\Pi K}{N})]$$

$$RB = RA + [RBCos(\frac{2\Pi K}{N})] - [IBSin(\frac{2\Pi K}{N})]$$

$$IB = IA - [RBCos(\frac{2\Pi K}{N})] + [IBSin(\frac{2\Pi K}{N})]$$

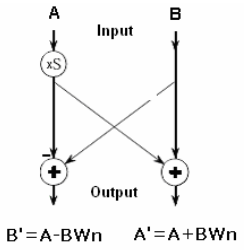

Figure 3 – Butterfly.

Thus, 8 point radix-2 time decimation needs 3 stages and 12 butterflies to give the output. Table 1 shows all the butterflies and the k (twiddle factor) value in each stage.

Stage 1	*Stage 2*	*Stage 3*
BTF(0,4) K=0	BTF(0,2) K=0	BTF(0,1) K=0
BTF(1,5) K=0	BTF(1,3) K=0	BTF(2,3) K=2
BTF(2,6) K=0	BTF(4,6) K=2	BTF(4,5) K=1
BTF(3,1) K=0	BTF(5,7) K=2	BTF(6,7) K=3

Table 1 – Butterflies in each stage.

3. EXPERIMENTAL RESULTS

The test and the simulation have been designed in Scilab®. We compared the 8 point test vectors of Scilab® FFT functions with our own FFT script coded in Scilab® language. Table 2 gives an overview of the results in each stage using our FFT script. We show in this table the address map using 16 words. The first eight are used to allocate the real numbers and the last ones to allocate the imaginary numbers.

4. HARDWARE IMPLEMENTATION

We designed the FFT hardware using datapath and control unit. Our circuit uses 8 registers, 9 counters, 1 comparator (4 bits), 12 multiplexers, 4 demultiplexers, 1 adder (8 bits), 2 adders (32 bits), 1 multiplicator (32 bits), 1 RAM memory (16 words x. 32 bits), 1 ROM memory (8 words x. 32 bits). Figure 4 describes the butterfly architectures design.

The RAM memory needs 16 words (8 to the real part and other 8 to the imaginary part). The same memory positions are used for the input and output. This efficient use of memory is important for designing fast hardware to calculate the FFT. The term in-place computation is used to describe this memory usage. The ROM memory stores the cossine and sine twiddle factor in IEEE 754 standard. The FSM module uses 14 cycles.

Table 3 shows the pipeline approach used. We can note that this table shows the simultaneous register transfer in the same clock cycle. By the way, the butterfly calculation needs 4 clock cycles and 57 cycles to complete the FFT calculus. The non pipeline approach needs 9 clock cicles to solve the butterfly and 108 cycles to give the FFT output.

5. CONCLUSION

The 8 points radix-2 time decimation FFT has been designed in VHDL and simulated in ModelSim®. The FFT algorithm has been validated in Scilab®. We can solve 8 complex points with a FFT in 57 clock cycles and one butterfly in 4 cycles. This approach is more than 50% efficient than the non pipeline approach.

The major FFT hardware implementation use 1024 point [3]. Finally, in future works we intend to design and prototyping of a 1024 point radix-2 time decimation FFT to compare with other FFT hardware implementations.

6. REFERENCES

[1] ÇETIN, Ediz; MORLING, Richard; KALE, Izzet. *An Integrated 256-point Complex FFT Processor for Real-time Spectrum Analyses and Measurement.* In: IEEE Proceedings of Instrumentation and Measurement Technology Conference, vol. 1, pp. 96-101, Ottawa, Canada, May 1997.

[2] SMITH, Steven W. The *Scientist and Engineer's Guide to Digital Signal Processing*. San Diego: California Techinical Publishing, 1999.

[3] Baas, Bevan M. *An Approach to Low-Power, High-Performance, Fast Fourier Transform Processor Design.* In: IEEE Journal of Solid-State Circuits, March 1999, pp. 380-387.

[4] WESTE, N.; BICKERSTAFF, M.; ARIVOLI, T.; RYAN, P.; DALTON, J.; SKELLERN, D.; PERCIVAL, T. *A 50 MHz 16-point FFT Processor for WLAN Applications*. In: IEEE Custom Integrated Circuits Conference, May 1997, pp. 457-460.

Address (Re ; Im)	*Input (Re ; Im)*	*Output Stage 1 (Re ; Im)*	*Output Stage 2 (Re ; Im)*	*Output Stage 3 (Re ; Im)*	*Bit Reversal (Re ; Im)*
0000 ; 1000	-1 ; 0	-3 ; 0	-2.5 ; 0	0 ; 0	0 ; 0
0001 ; 1001	1 ; 0	-0.5 ; 0	2.5 ; 0	-5 ; 0	2.06 ; 4.97
0010 ; 1010	1.5 ; 0	0.5 ; 0	-3.5 ; 0	-3.5 ; 3.5	-3.5 ; 3.5
0011 ; 1011	2 ; 0	3 ; 0	-3.5 ; 0	-3.5 ; -3.5	-0,60 ; 0.02
0100 ; 1100	-2 ; 0	1 ; 0	1 ; 2.5	2.06 ; -4.97	-5 ; 0
0101 ; 1101	-1.5 ; 0	2.5 ; 0	2,5 ; -1	-0.06 ; -0.02	-0,06 ; -0.02
0110 ; 1110	-1 ; 0	2.5 ; 0	1 ; 2.5	-0.06 ; 0.02	-3.5 ; -3.5
0111 ; 1111	1 ; 0	1 ; 0	2.5 ; 1	2.06 ; 4.97	2,06 ; 4.97

Table 2 – FFT result in each stage using a test vector.

Cycle	*RAM Read*	*ROM Read*	*MUL*	*ADD1*	*ADD2*	*RAM Write*
0	RB	cos(w)	M4<=RB*sin(w) (pre.)	S1 <= M1–M2 (pre.)	RAO<= RA + S0 (pre.)	IBO
1	IB	sin(w)	M1<=RB*cos(w)	S2 <= M3 - M4 (pre.)	RBO<= RA - S1 (pre.)	RAO (pre.)
2	RA	cos(w)	M2<=IB*sin(w)	S3 <= M3 + M4 (pre.)	IAO <= IA + S2 (pre.)	RBO (pre.)
3	IA	sin(w)	M3<=IB*cos(w)	S0 <= M1 + M2	IBO <=IA -S3 (pre.)	IAO (pre.)
0	RB (next)	cos(w)	M4<=RB*sin(w)	S1 <= M1 - M2	RAO <=RA+S0	IBO (pre.)
1	IB (next)	sin(w)	M1<=RB*cos(w) (next)	S2 <= M3 – M4	RBO <=RA – S1	RAO
2	RA (next)	cos(w)	M2<=IB*sin(w) (next)	S3 <= M3 + M4	IAO <=IA + S2	RBO
3	IA (next)	sin(w)	M3<=IB*cos(w) (next)	S0 <= M1 + M2 (next)	IBO <=IA –S3	IAO
0	RB (next)	cos(w)	MUL4<=RB*sin(w) (next)	S1 <= M1 - M2 (next)	RAO <=RA + S0 (next)	IBO

Table 3 – Register transfer using pipeline.

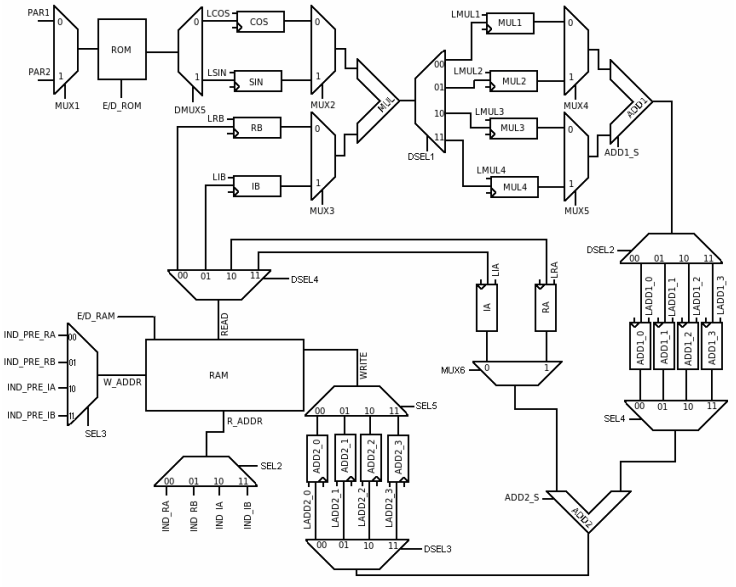

Figure 4 – Butterfly datapath.