VLSI Signal Processing Final Project

Fast convolution Implementation using FNT

R89921145
林秉勳

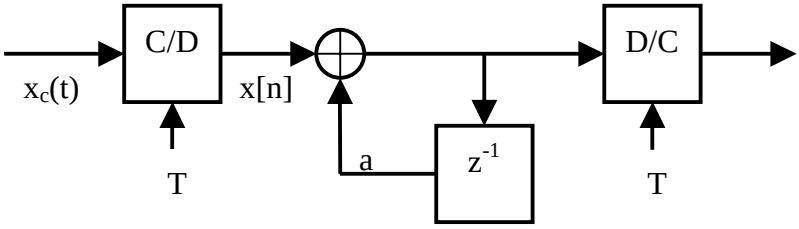
Contents

I. Abstract

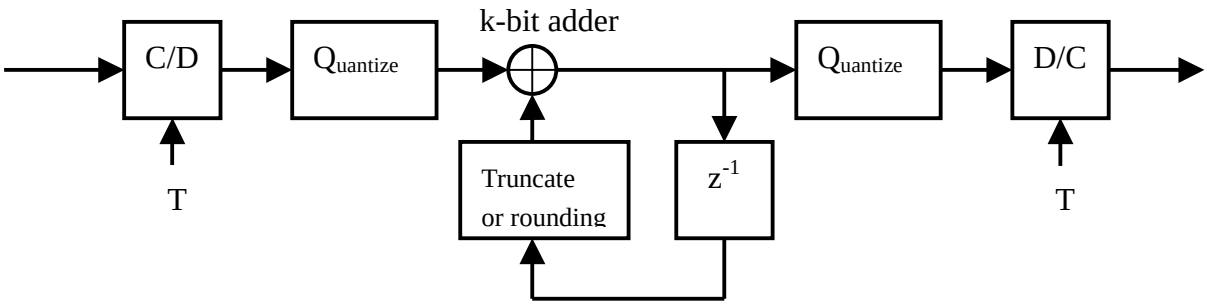
Convolution is a basic operation in digital signal processing. When finite word length is used in calculating the convolution, there exist rounding and truncation errors. These errors can be eliminated by utilizing the number theoretic transform(NTT). The NTT which is based on Fermat number is called Fermat number transform(FNT) FNT can retain both large transform length and multiplierless property. But it has the disadvantage that an inconvenient word length and arithmetic required for its calculation. In this term project I study the number theoretic transform(NTT) and its special case the Fermat number transform (FNT), especially focusing on the latter. The main advantages of the NTT are that there are no rounding or truncation errors and under some conditions the transform can be done without multiplications. So in section II I introduce the round –off and truncation issues which will happen in real system. The mentioned advantages can be achieved by using transformed domain fast convolution which will be better than the conventional method under some conditions. So I'll introduce some fast convolution method and discuss the transform domain convolution in section III. Because the NTT is build on the number theory, so in section IV I list the necessary definitions and theories about NTT. In section V I introduce the NTT. In section VI I introduce the FNT. In section VII I do the simulation of the transform domain convolution using FNT by Matlab. In the following two sections I design the architecture and gate level circuit to implement FNT, and I use Verilog to verify the design. The last section is the conclusion of this job.

II. The round-off and truncation issues

Assume we have a system like this:



But in practical, there doesn't exist infinite length of registers to save the data. So if we want to keep the *closure* (use the finite word length to present the original data) of data, we must truncate or round the data. As the following block diagrams shows:



Although the closure of data is kept, the *equality* (the two data are different) is lost after truncating and rounding procedure. In an IIR filter, such small error will accumulate and result in enormous error. When we use FFT to implement the transform domain fast

convolution, the twiddle factors saved in the rom(read only memory) are also truncated.

III. Fast convolution

There are several types of fast convolution, as the following shows:

- Cook-Toom algorithm: it's a linear convolution algorithm for polynomial multiplication which is based on *Lagrange interpolation theorem*.
- Winograd algorithm: it's based on the Chinese remainder theory over an integer ring.
- Multirate method: using multirate technique like noble identity which can lower the operating clock of the convolution system under the same throughput. Or if we keep the operating frequency the same, we can get higher throughput, which means faster.
- Transform based convolution: transform the data into another domain, like frequency domain using FFT, processing them, and transform back. We can gain the less use of the multipliers by this method.

The following discussion focuses on transform based convolution.

Finite digital convolution is a numerical procedure defined by:

$$y[n] = \sum_{m=0}^{N-1} h[n-m]x[m],\ n = 0,1,2,... \quad \textit{denoted by } y[n] = h[n] * x[n]$$

Using the property: $x[n] \otimes h[n] \xleftrightarrow{DFT} X_k \cdot Y_k$

We can get the normal convolution by padding the original sequence with zeros, doing FFT with them, then multiply them in frequency domain, and do IFFT with them, as mentioned in lecture. This convolution procedure needs $2N\log_2 2N$ multiplies to do the FFT and IFFT, and 2N multiplies to do the frequency domain multiply which is a considerable saving compared with the regular method, N^2. But the disadvantages of this approach are significant amounts of round-off errors of the twiddle factors mentioned as above and the additional memory used for saving them.

IV. The structure of transforms having the convolution property

Let x_n and h_n be two sequences which are to be circularly convolved as the following:

$$y_n = x_n \otimes y_n = \sum_{k=0}^{N-1} x_{<<k>>_N} h_{<<n-k>>_N} \quad n = 0 \sim N-1$$

Here the class of transforms considered is the set of length N to another sequence of length

N. The linear invertible transforms can be represented by an N×N nonsingular matrix T whose elements are t_{km} where k, m=0~N-1. The capital letters denote the transformed sequence: X=Tx, H=Th, Y=Ty, where x, h, and y are vectors constructed by x_n, h_n, and y_n.

Then $y_k = \sum_{n=0}^{N-1} t_{kn} y_n = \sum_{n=0}^{N-1} t_{kn} \sum_{m=0}^{N-1} x_{<<m>>_N} h_{<<n-m>>_N} = \sum_{m=0}^{N-1} \sum_{l=0}^{N-1} x_m h_l t_{km+l}$ (*paper is wrong here*)

Since $X_k = \sum_{m=0}^{N-1} t_{km} x_m \quad H_k = \sum_{l=0}^{N-1} t_{kl} h_l \quad k = 0 \sim N-1$ and $Y_k=X_kH_k$

$$\Rightarrow \sum_{m=0}^{N-1} \sum_{l=0}^{N-1} x_m h_l t_{km+l} = \sum_{m=0}^{N-1} \sum_{l=0}^{N-1} x_m h_l t_{km} t_{kl}$$

$$\Rightarrow t_{km+l} = t_{km} t_{kl} \quad k,l,m = 0 \sim N-1$$

Iteration by the above equation we can get:

$$\Rightarrow t_{km} = t_{k1}{}^{m} \quad k,m = 0 \sim N-1$$

Because the convolution is circular, the indices are added modulo N:

$$\Rightarrow t_{km} = t_{km+N} = t_{km} (t_{k1})^N$$

$$\Rightarrow (t_{k1})^N = 1 \quad \text{k=0~N-1}$$

For T to be nonsingular, all the t_{k1}s must be distinct. *Because if t_{m1} equals to t_{n1}, the m_{th} row equals the n_{th} row. Then the T matrix will be singular*. And since there are only N distinct Nth roots of unity, the t_{k1}s must be these N distinct roots. Without loss of generality, t_{11} can be seen as the root of order N, and other t_{k1}s can be written in the form of the power of t_{11} as:

$$t_{k1} = t_{11}{}^{k}$$

Thus t_{km} can be written as: $t_{km} = t_{11}{}^{km}$.

Replace t_{11} by α we can get: $t_{km} = \alpha^{km} \quad k,m = 0 \sim N-1$

The above reveals that ***the existence of an N×N transforms having the cyclic convolution property depends only on the existence of an α that is a unity of order N and the existence of N^{-1}***. The structure of T is the only structure support the circular convolution property. This structure is called the DFT structure. The most important of the structure is that any transforms having the cyclic convolution property also have a fast computational algorithm similar to the FFT if N is highly composite. The DFT with $\alpha = e^{-j\frac{2\pi}{N}}$ is the only transform in the complex number field (I'm not clear at this point). But in a finite field or, more generally, a ring there might also have transforms with the cyclic convolution property if there exists a root of unity of order N and if N^{-1} exists(these properties will be introduced in the next section).

V. Basic number theory

The following are some definitions and theories(with proof reference to *"ELEMENTARY NUMBER THEORY AND ITS APPLICATIONS" KENNETH H. ROSEN*) that will need to Derived some important property of the NTT.

Definition 1. *Euler phi − function* $\phi(n)$: Let n be a positive integer. The function is defined to be the number of positive integers not exceeding n that are relatively prime to n.

Definition 2. *Order of a modulo m:* Let a and m are relatively prime positive integers. Then the least positive integer x such that $a^x \equiv 1 (\mathrm{mod}\, m)$ is called the order of a modulo m which is denoted by $\mathrm{ord}_m a$.

Definition 3. A *reduced residue system modulo n* : A set of integers such that each element of the set is relatively prime to n, and no two different elements of the set are congruent modulo n.

Definition 4. *Ring*: A ring is a nonempty set, denoted R, together with two operaqtions + and · satisfying the following properties for each a,b,c∈ R:

1. (a+b)+c=a+(b+c) addition associative property

2. There exists an element 0 in R such that a+0=0+a=a additive identity

3. There exists an element –a in R such that a+(-a)=0=(-a)+a additive inverse

4. a+b=b+a addition commutative property

5. (a · b) · c=a · (b · c) multiplication associative property

6. (a+b) · c=a · c+b · c distributive property

Theorem 1. If a,b,c,d and m are integers such that m>0, $a \equiv b (\mathrm{mod}\, m)$ and $c \equiv d (\mathrm{mod}\, m)$ then $ac \equiv bd (\mathrm{mod}\, m)$

proof: ac-bd=ac+bc-bc-bd=c(a-b)+b(c-d)=ckm+blm=m(ck+bl) where k and l are integers.

$\Rightarrow$ m|ac-bd

$\Rightarrow$ $ac \equiv bd (\mathrm{mod}\, m)$

Theorem 2. $r_1, r_2, \ldots, r_{\phi(n)}$ is a reduced residue system(rrs) modulo n, and if a is a positiveinteger with (a,n)=1, then the set $ar_1, ar_2, \ldots, ar_{\phi(n)}$ is also a reduced residue system modulo n.

proof: Assume (ar_j, n)>1, then there exists a divisor p such that either:
1.{p|$r_j \cap$ p|n} or 2.{p|a$\cap$ p|n}.
1. cannot happen ∵ r_j is in a rrs modulo n.
2. cannot happen ∵ (a,n)=1
$\Rightarrow$ $ar_1, ar_2, \ldots, ar_{\phi(n)}$ is also a rrs modulo n.#

Theorem 3. *Euler's theorm*: If m is a positive integer and a is an integer with (a,m)=1, then $a^{\phi(m)} \equiv 1(\mathrm{mod}\, m)$.

proof: Let $r_1, r_2, \ldots, r_{\phi(m)}$ denote a rrs modulo m. $\because$ (a,m)=1 $\cap$ theorem2
$\Rightarrow ar_1, ar_2, \ldots, ar_{\phi(m)}$ is also a rrs modulo m. The least positive residues of
$ar_1, ar_2, \ldots, ar_{\phi(m)}$ must be $r_1, r_2, \ldots, r_{\phi(m)}$ in some orders.
Multiply all terms in both rrs and use theorem 1:
$\Rightarrow ar_1\, ar_2 \ldots ar_{\phi(m)} \equiv r_1\, r_2 \ldots r_{\phi(m)} (\mathrm{mod}\, m)$
$\Rightarrow a^{\phi(m)} r_1 r_2 \ldots r_{\phi(m)} \equiv r_1\, r_2 \ldots r_{\phi(m)} (\mathrm{mod}\, m)$
$\because (r_1, r_2, \ldots, r_{\phi(m)}, \mathrm{m})=1$
$\therefore a^{\phi(m)} \equiv 1(\mathrm{mod}\, m)$ #

Theorem 4. if a and n are relatively prime integers with m>0, then the positive integer x is a solution of the congruence $a^x \equiv 1(\mathrm{mod}\, m)$ iff $\mathrm{ord_n}\, a|x$.
proof: If $\mathrm{ord_m} a|x \Rightarrow$ x=k $\mathrm{ord_m} a$ where k is a positive integer. Hence,

$$a^x = a^{k ord_m a} = (a^{ord_m a})^k \equiv 1(\mathrm{mod}\, m)$$

conversely, if $a^x \equiv 1(\mathrm{mod}\, m) \Rightarrow x = q \cdot ord_m a + r,\ 0 \le r < ord_m a$
$\Rightarrow a^x = a^{q \cdot ord_m a + r} = (a^{ord_m a})^q a^r \equiv a^r (\mathrm{mod}\, m)$
$\because a^x \equiv 1(\mathrm{mod}\, m)$
$\therefore a^r \equiv 1(\mathrm{mod}\, m) \cap\ 0 \le r < ord_m a$
$\Rightarrow$ r=0
$\mathrm{ord_m}\, a|x$ #

from Euler's theorem and theorem 4 we can deduce:
Theorem 5. If a and m are relatively prime integers with m>0, then $\mathrm{ord_m}\, a|\phi(m)$.

Theorem 6. If gcd(m,n)=1 then $a \equiv b(\mathrm{mod}\, m \cdot n)$ *iff* $a \equiv b(\mathrm{mod}\, m)$ *and* $a \equiv b(\mathrm{mod}\, n)$
proof: (only prove single direction)
m|a-b, n|a-b $\Rightarrow$ lcm(m,n)|a-b
if gcd(m,n)=1 $\Rightarrow$ lcm(m,n)=m*n
$\Rightarrow$ m*n|a-b $\Rightarrow a \equiv b(\mathrm{mod}\, m \cdot n)$

Theorem 7. Let p be a prime and a a positive integer. Then $\phi(p^a) = p^{a-1}(p-1)$.
Proof: The positive integers less than p^a that are not relatively prime to p are:
$p \le kp < p^a \Rightarrow 1 \le k < p^{a-1}$ there are p^{a-1} such numbers.
$\Rightarrow \phi(p^a) = p^a - p^{a-1} = p^{a-1}(p-1)$ #

Assume α is a root of order N modulo M and the modulus M= $p_1^{r1} p_2^{r2} \cdots p_i^{ri}$ where p_k^{rk} are distinct primes. Let gcd(M,N)=1. If $\alpha^N \equiv 1(\mathrm{mod}\, M)$ and use theorem 6
$\Rightarrow \alpha^N \equiv 1(\mathrm{mod}\ p_j^{rj}) \quad j = 1,2,\cdots,i$ and use theorem 5
$\Rightarrow$ N| $\phi(p_j^{rj})$ j=1,2,…,i
using theorem 7 we can get N| $p_j^{rj-1}(p_j - 1) \quad j = 1,2,...,i$

since N is relatively prime to M
$\Rightarrow N|(p_j-1) \quad j=1,2,\ldots,i$
$\Rightarrow$ **$N|gcd(p_1-1,p_2-1,\ldots,p_i-1)$**

VI. Number theoretic transform (NTT)

Since the digital device can represent data in finite precision, if we could find an operation that can be restricted in the finite set of number, we may keep both the closure and equality. The number theoretic transforms are defined over a finite ring (as the definition mentioned) of integers and are operated in modulo arithmetic. In this ring, using transforms similar to the DFT, the cyclic convolution can be performed very efficiently and without any round-off errors. In modulo arithmetic, all basic operations such as addition, subtraction, and multiplication are carried out modulo an integer M.
The NTT can be represented as the following equations:

$$X(t) = \sum_{n=0}^{N-1} x(n)\alpha^{nk} \bmod M \quad k = 0,\ldots,N-1$$

$$x(n) = N^{-1}\sum_{k=0}^{N-1} X(k)\alpha^{-nk} \bmod M \quad n = 0,\ldots,N-1$$

From the last result derived from "Basic number theory" and "The structure of transforms having the convolution property", we know the transform exists iff
$N|gcd(p_1-1,p_2-1,\ldots,p_i-1)$ $M= p_1^{r1} p_2^{r2} \cdots p_i^{ri}$ and $\alpha^N = 1(\bmod M)$ and the existence of N^{-1}.

When we want to use the NTT to implement circular convolution efficiently, there are three constraints:

1. N should be highly composite so that the NTT may have a fast algorithm, and it should be large enough for application to long sequence length.

2. Multiplication by powers of α should be a simple operation. If α and its power have a simple binary representation, then this multiplication reduces to bit shift.

3. In order to facilitate arithmetic modulo M, M should also have a very few bit binary representation.

In the ring of integers modulo M, conventional integers can be unambiguously represented if their absolute value is less than half of M. If the input integer sequences x[n] and h[n] are so scaled that |y[n]| never exceed M/2, we would get the same results by implementing convolution in the ring of integers modulo M as that obtained with normal convolution.

VII. Fermat number transform(FNT)

When we replace the above M by Fermat Number F_t (Fermat conjectured that numbers in the form $2^{2^t}+1$ are primes, but the conjecture was wrong. Only when t=0~4 this conjecture is valid) .i.e., $2^{2^t}+1$, t is a positive integer, the transform is called Fermat number transform. Recall that we have derived that the transform length must divides the greatest common divisor of the factorized prime number minus one. If we choose M to be even ($M=2^k*P$), for N | gcd(2-1=1, P-1) to be valid, N must be 1, which means the

transform length is only one. It's not possible for applications. But if M is prime, N can be as large as M-1. Since Fermat number up to t=4 are prime, we can have an FNT for any length $N=2^m$, $m \leq t$.

In the structure of FNT, it doesn't need multipliers when doing transform, because multiply by 2^k can be accomplished by only right shift. Thus the FNT seems to provide a more efficient transform domain convolution than utilizing FFT. In fact, after each butterfly computation, the intermediate data must be done modulus arithmetic. This is a very complicated procedure if we use the conventional 2's complement representation. Leibowitz proposed a method called "diminished-1 representation" which makes the modulo procedure a easy thing. The diminished-1 representation is easy to think. In a digital system, it is easiest to take the modulo of a number in the form of power of two. For example if we want to take the modulo of 2^k, we just take the least significant k bits of the number. Because the Fermat number is power of two plus one, if we minus the Fermat number with one, and map the modulus to another set of numbers, we can do modulo without additional operation. First, let's look at the following table:

Normal value	Binary representation	Diminished-1 value
0	00000	1
1	00001	2
2	00010	3
3	00011	4
4	00100	5
5	00101	6
6	00110	7
7	00111	8
8	01000	9(-8)
9(-8)	01001	10(-7)
10(-7)	01010	11(-6)
11(-6)	01011	12(-5)
12(-5)	01100	13(-4)
13(-4)	01101	14(-3)
14(-3)	01110	15(-2)
15(-2)	01111	16(-1)
16(-1)	10000	0

From the above table we can find the diminished-1 value is a circular shift (right shift one) version of the normal value. Under this shift, he change the basic arithmetic as follows:

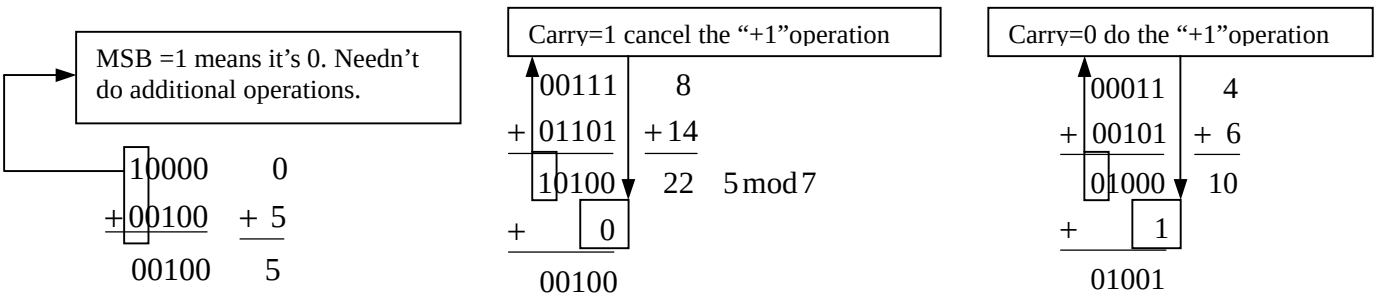
Negation: $\overline{A-1} = -A-1$

Addition: (A+B)-1=[(A-1)+(B-1)]+1

When a carry is generated from the b-bit sum, a residue reduction modulo F_t requires the subtraction of a 1 since 2^b= -1 mod F_t and no corrective addition is necessary. Thus to add two numbers in diminished-1 representation:

1. If the MSB of either addend is 1, inhibit the addition and the remaining addend is the sum.
2. If the MSB of both addends are 0s, ignoring the MSB add the b lsb's, complement the carry and add it to the b lsb's of the sum.

There are some examples:



Multiplication: $A \cdot B - 1 = (A-1)(B-1) + (A+B-1) - 1$

The main disadvantage of FNT is the restriction of the transform length. We can't select any transform length to do the FNT, because the constraints between N, M and α. Since the word length is proportional to the transform length, if we want higher transform length, the word length may be ridiculously large. In normal convolution method, i.e., direct form 1, the word length of the adders and registers can be increased when they are near the output. Unlike the transform domain convolution utilizing FNT, all adders and registers must use very long word length.

VIII. Matlab Simulation

The followings are simulations to verify the reversibility of FNT convolution done by FNT, and compare with the :

Case 1. Reversibility :N=8,F_t=17, $\alpha = 2$,input signal x=[1 2 3 4 5 6 7 8]:

Using the following Matlab code:

```
clear;
num=8;
modu=17;
alpha=2;
n=1:8;

for i=1:num;
for j=1:num;
t1(i,j)=mod(alpha^(mod((i-1)*(j-1),num)),modu);
end
end

y=t1*n';

for i=1:num;
for j=1:num;
t2(i,j)=mod(32*mod(alpha^(mod(-(i-1)*(j-1),num)),modu),modu);
end
end

out=mod(t2*y,modu)';
```

The output is:

out =

1 2 3 4 5 6 7 8

which is the same as the input. Thus verifies the exact reversibility.

Case 2. Convolution: N=16, F_t=257, $\alpha = 2$, input signal x=[1 1 1 1 1 1 1 1], y=[1 1 1 1 1 1 1 1]

Using the following Matlab code:

```
clear;
num=16;
modu=257;
alpha=2;
n=[ones(1,num/2) zeros(1,num/2)];

for i=1:num;
for j=1:num;
t1(i,j)=mod(alpha^(mod((i-1)*(j-1),num)),modu);
end
end

y=t1*n';
y1=y.*y;

for i=1:num;
for j=1:num;
t2(i,j)=mod(241*mod(alpha^(mod(-(i-1)*(j-1),num)),modu),modu);
end
end
out=mod(t2*y1,modu)';
```

In the above code we first padding the original data with zeros. Then do the multiplication in the transform domain. Finally transform the sequence back to the time domain. All the computations are in integer field. In fact we can replace the variable "alpha" with only shift operation, if $\alpha = 2$ **.** The output is:

```
out =

   Columns 1 through 14

     1     2     3     4     5     6     7     8     7     6     5     4     3     2

   Columns 15 through 16

     1     0
```

which is the same as the normal convolution

case 3. verify the precision of reversibility of DFT/IDFT with the following word lengths: 8-bit, 10-bit, 12-bit, 14-bit, 16-bit :

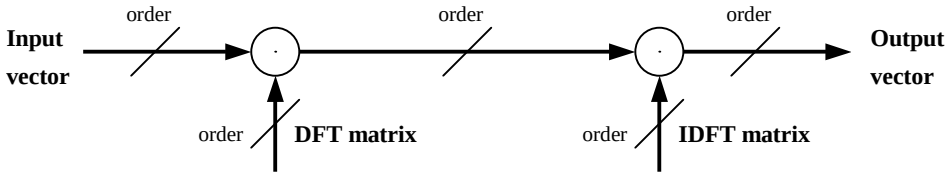
using the following Matlab code:

```
clear;
num=16;
order=8;
n=1:16;
for p=1:num;
for k=1:num;
t1(p,k)=floor(2^order*exp(-j*2*pi*p*k/num));
end
end
y0=t1*n';
normal=max(max(real(y0)),max(imag(y0)));
y0=floor(y0/normal*2^order);
for p=1:num;
for k=1:num;
t2(p,k)=floor(2^order*exp(2*j*pi*p*k/num))/16;
end
end
```

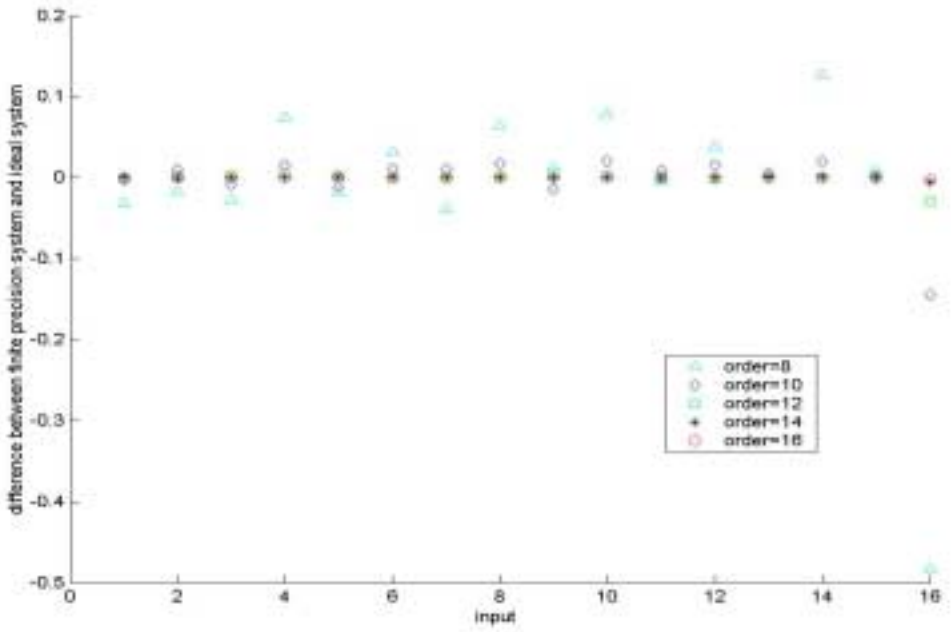
```
out0=(t2*y0)/2^(3*order)*normal;
%stem(n,abs(out1)-n','c^');
hold on

num=16;
order=10;
n=1:16;
for p=1:num;
for k=1:num;
t1(p,k)=floor(2^order*exp(-j*2*pi*p*k/num));
end
end
y1=t1*n';
normal=max(max(real(y1)),max(imag(y1)));
y1=floor(y1/normal*2^order);
for p=1:num;
for k=1:num;
t2(p,k)=floor(2^order*exp(2*j*pi*p*k/num))/16;
end
end
out1=(t2*y1)/2^(3*order)*normal;
%stem(n,abs(out1)-n','d');
hold on
num=16;
order=12;
n=1:16;
for p=1:num;
for k=1:num;
t1(p,k)=floor(2^order*exp(-j*2*pi*p*k/num));
end
end
y2=t1*n';
normal=max(max(real(y2)),max(imag(y2)));
y2=floor(y2/normal*2^order);
for p=1:num;
for k=1:num;
t2(p,k)=floor(2^order*exp(2*j*pi*p*k/num))/16;
end
end
out2=(t2*y2)/2^(3*order)*normal;
%stem(n,abs(out2)-n','gs');
hold on

num=16;
order=14;
n=1:16;
for p=1:num;
for k=1:num;
t3(p,k)=floor(2^order*exp(-j*2*pi*p*k/num));
end
end
y3=t3*n';
normal=max(max(real(y3)),max(imag(y3)));
y3=floor(y3/normal*2^order);
for p=1:num;
for k=1:num;
t2(p,k)=floor(2^order*exp(2*j*pi*p*k/num))/16;
end
end
out3=(t2*y3)/2^(3*order)*normal;
```

```
%stem(n,abs(out3)-n','k*');
hold on

num=16;
order=16;
n=1:16;
for p=1:num;
for k=1:num;
t1(p,k)=floor(2^order*exp(-j*2*pi*p*k/num));
end
end
y4=t1*n';
normal=max(max(real(y4)),max(imag(y4)));
y4=floor(y4/normal*2^order);
for p=1:num;
for k=1:num;
t2(p,k)=floor(2^order*exp(2*j*pi*p*k/num))/16;
end
end
out4=(t2*y4)/2^(3*order)*normal;
plot(n,abs(out0)-n','c^',n,abs(out1)-n','d',n,abs(out2)-n','gs',n,abs(out3)-n','k*',n,abs(out4)-n','ro');
legend('order=8','order=10','order=12','order=14','order=16');
```

In the simulation system, all data flowed are restricted to a specified number of bits, which can be modeled as the following figure: $\odot$ means inner product



And the simulation result is as follows:



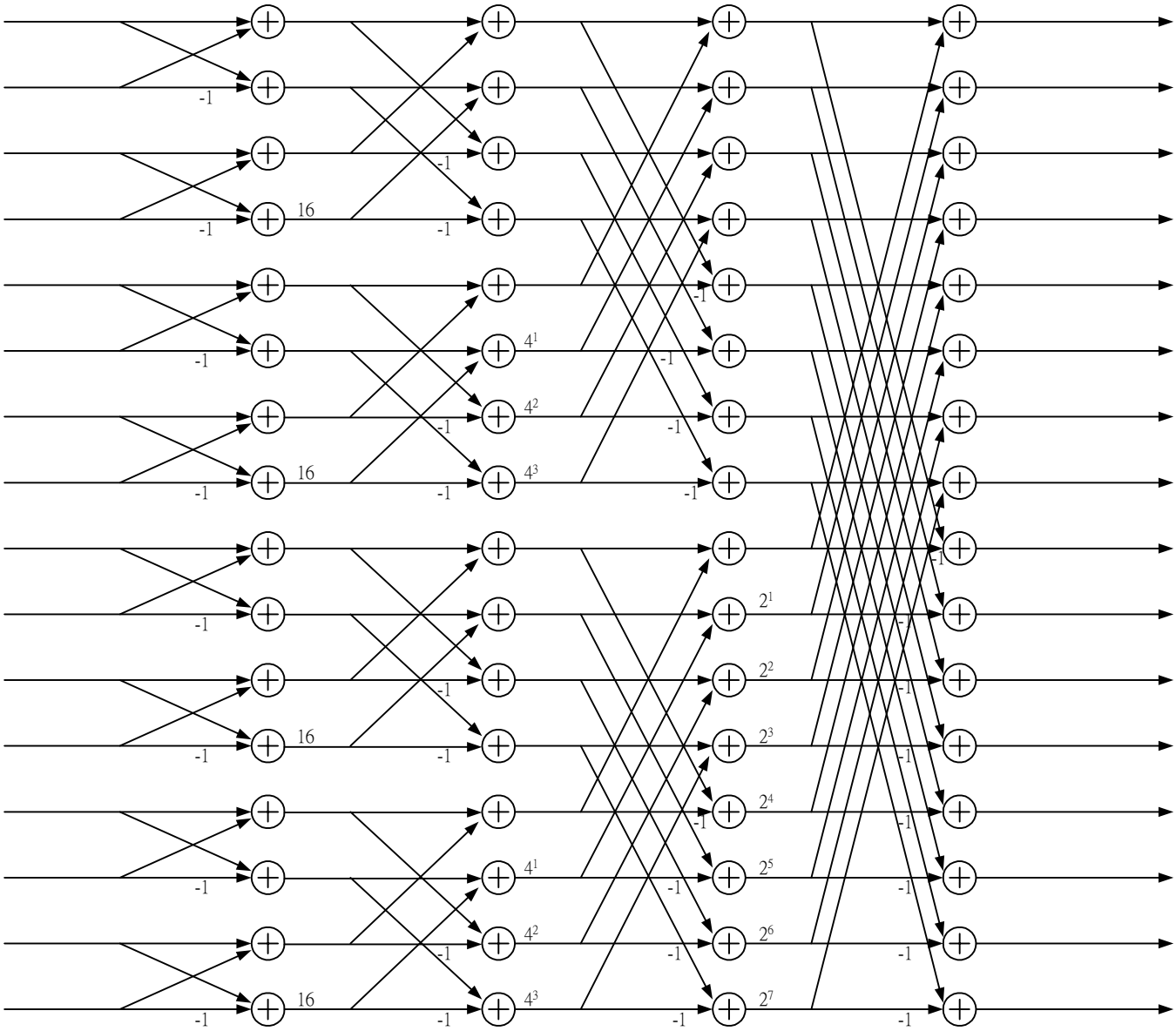
The x-axis is input data from 1 to 16, the y-axis is the output difference between finite precision system and ideal system. From this figure we can see when the word length is

longer, the output is more approach the ideal one, which makes sense. But compare the FNT system with the above one, we can find we only use 8-bit word length, but the above must use more than 14-bit.

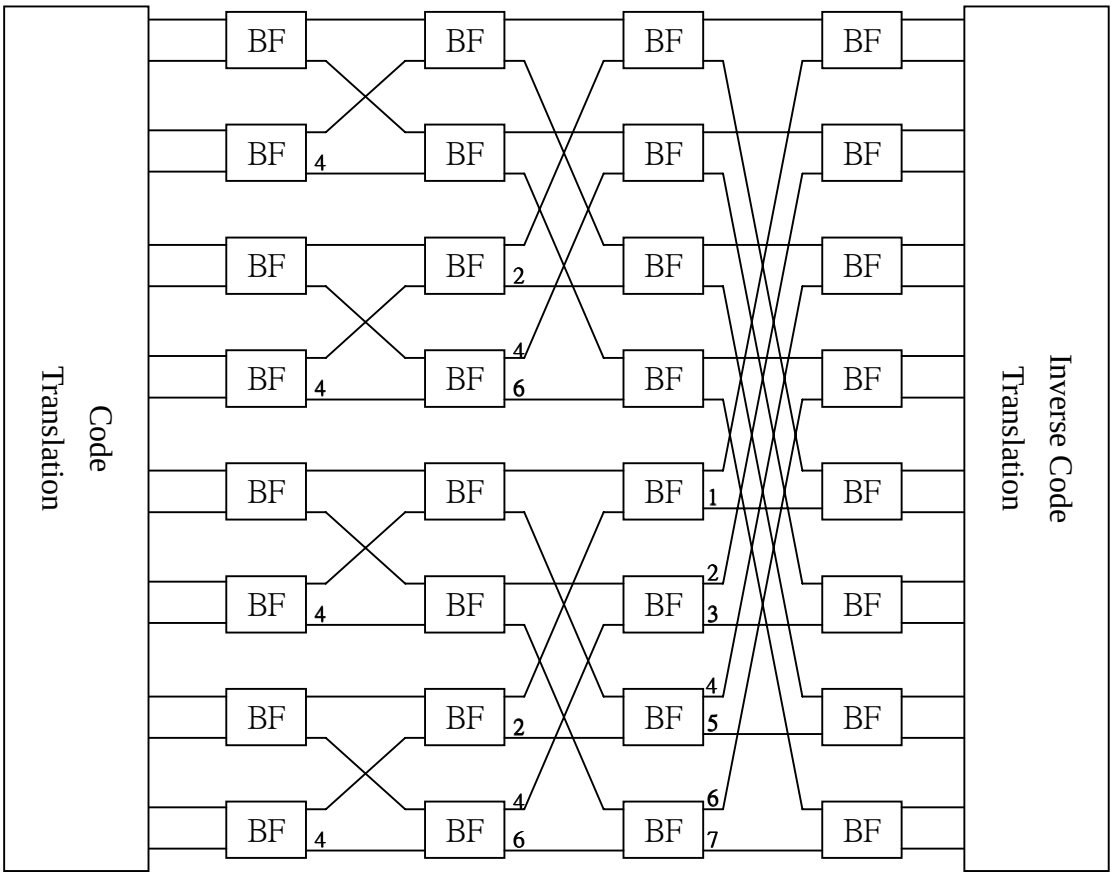
Although FNT can gain the less usage of word length in such a case, it doesn't mean FNT is also better in other cases. Because there's restriction between the selection of transform length, word length, and the kernal α. When we want a longer transform length, the word length will grow linearly with it, as discussed before.

Hardware implementation

The following FFT-like structure is the FNT I want to implement with F_t=257, transform length=16, wordlength=9bits.



For the sake of modularity, I redraw the above structure as the following, which has only local connected of regular butterfly structures:



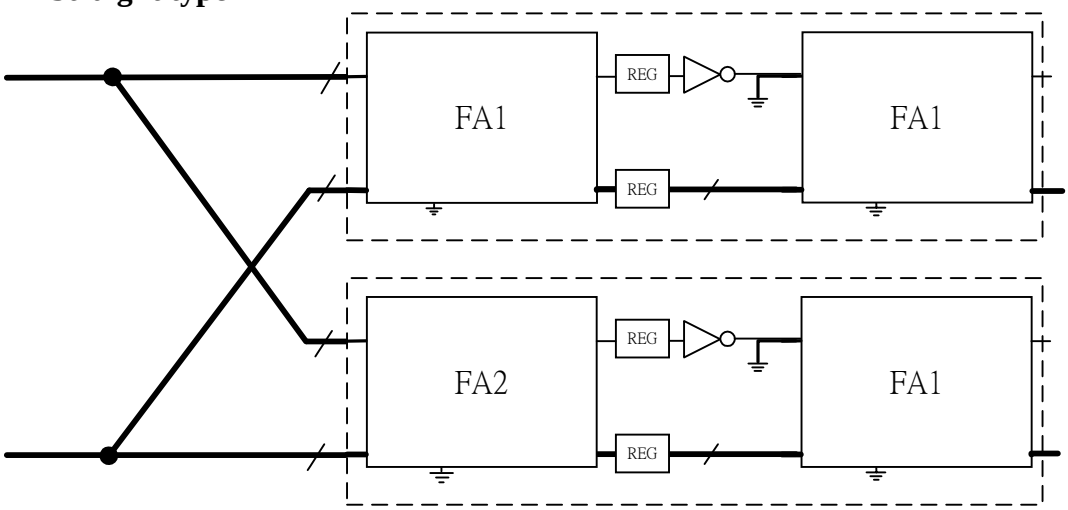
The number above the datapath means right shift the number of bits

Since there exists the problem of doing the modulo of a Fermat number as mentioned, the butterfly structure is not as easy as only two adders.

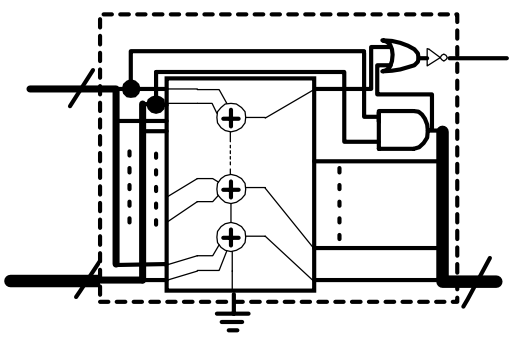
Butterflies

Using the diminish 1 notation when doing addition, it is different to the normal one. So I propose 2 structures to implement the butterflies, as the following figures show:

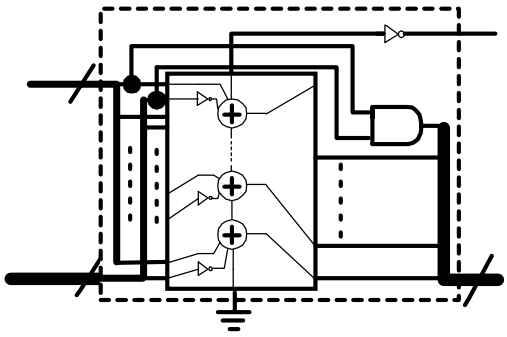
Straight type



FA1 has an adder doing addition, and FA2 has an adder doing subtraction as the following figures:



FA1



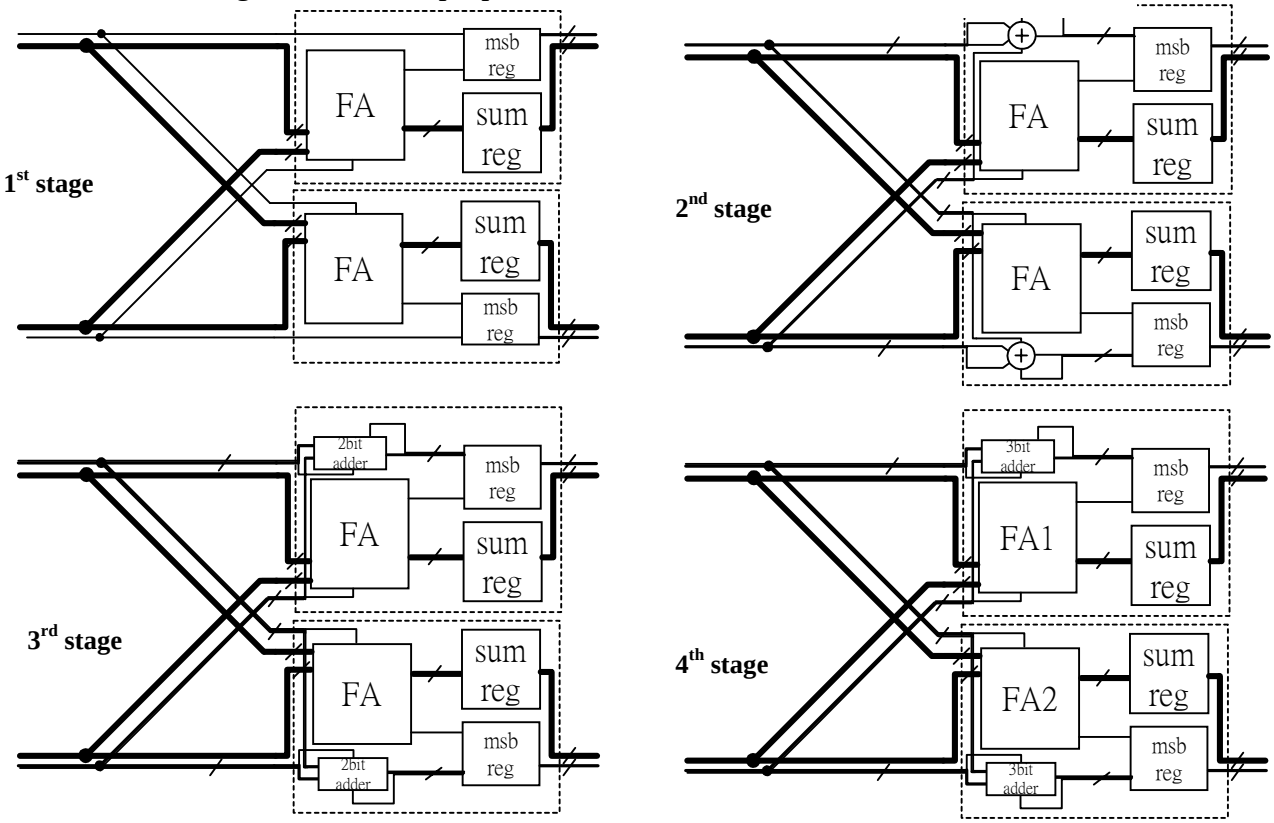
FA2

The reason I design FA1 and FA2 like these is from the diminish-1 representation:
1. Only one MSB=1 $\Rightarrow$ MSB=0
2. Both MSB=1(addition of two zeros) $\Rightarrow$ MSB=1
3. Both MSB=0 $\Rightarrow$ MSB=0

Using Karnaugh map, we can get $MSB_{out}=MSB_{in1} \bullet MSB_{in2}$. The corrective addition is one or zero can be easily derived as the figures of FA1 and FA2 show.
The reason I call the first one “straight type” is that it is derived straight forward from the diminish-1 representation. This type has a big problem of its low throughput and complexity. Each data is computed completely after 2 clock cycles. And each butterfly structure needs 4 adders, which is a number double than the normal one. We can promote the utilization rate by a factor of two by interleaving the input data with two independent data sequences like I-channel and Q-channel. Probably for the sake of its so straight forward, I saw this structure was discussed in some papers.

CSA-like type

the following structures is I propose to accomplish the butterflies in FNT:



The reason why I called this one “CSA-like type” is that it doesn’t do the diminish computation at each stage, but save all the MSBs of each stage and do the computation at the latter stage. This structure works as follows: when data are computed at the end of the 1st clock, the “original MSB” is saved in the msb reg and the sum data is saved in the sum reg. During the 2nd clock, two sums of the 1st stage are processed in butterflies at 2nd stage, and the MSBs can also be processed at the same time. At the end of the 2nd clock, there are two “original MSBs” from the 1st stage and the 2nd stage. So we need a 2 bits register to save them. The same procedure is done at the end of the 3rd clock. Finally we add the sums and corrective addition at the last stage. Because the corrective operation is done parallel at latter stages, it retains the throughput and not as the straight one.

This structure promotes the throughput by a factor of two of a single data sequence. There’s good pipeline between each stage. The difference between blocks of different stages is only the length of msb reg and adders for saving the correction terms. Because of its high regularity and modularity, we can build the structure easily, except the output stage. Blocks of the last stage have additional adders.

Shifters

The shift here is not only the normal shift, but also the ‘diminish-1 shift’. The operation is shown as follows:

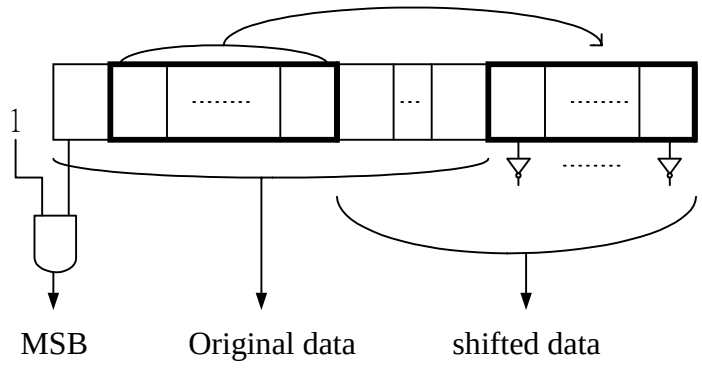
if we want to left shift 6 by 2 00101 6

L-shift 1: 01010+00001=01011 12

L-shift 1: 00110+00000=00110 7

By this diminish-1 shift, if we want left shift m bits, we must circular shift 1 bit those bits except MSB and add the correction 1 for m times. It will take a lot of time, and not as easy as we think as previous mentioned.

I propose a method that can do diminish-1 shift easily:



Actually the copy of data in the above figure can be done by only shift the data path. Using the same example: left shift 6 by 2

00101 $\overline{01}$

0 0110 which is the same

Code translator(CT) & inverse code translators(ICT)

Because we use the diminish-1 representation, we must do code translation for continuing arithmetic. The function of the block is directly derived from the following example:

CT: 01100 normal 12

01111

11011

0

01011 diminish-1 12

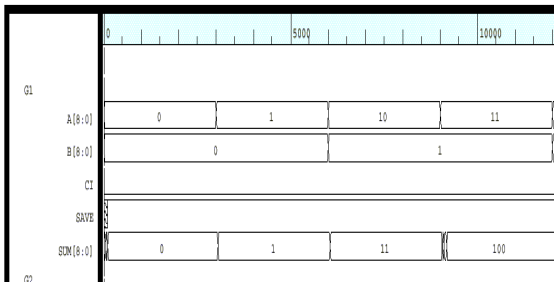
ICT: 01011 diminish-1 12

1

01100 normal 12

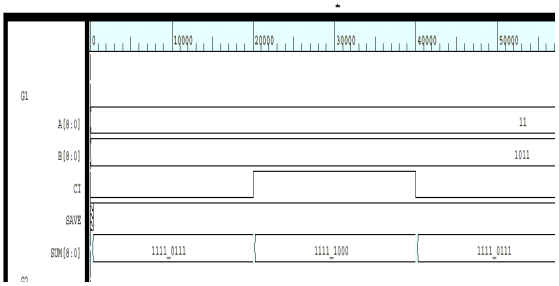
Simulation by verilog

D1-adder



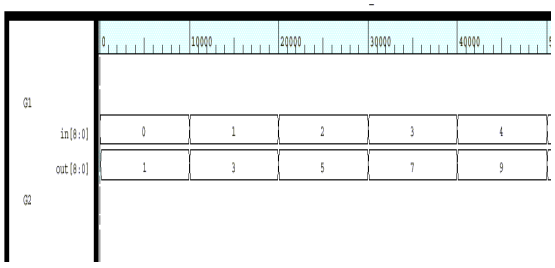
From the sum and two inputs, the D1-adder looks like normal adder, but it needs to save the MSB to be the corrective term. Notice that the pin “save” is the complement of the MSB of the sum in the left figure.

D1-subtractor



Like the D1-adder the pin “save” is the complement of the sum.

D1-shift1

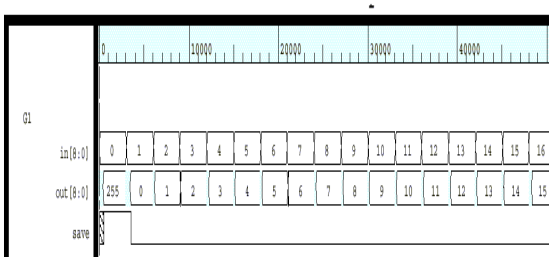


Here only illustrates D1-shift1. Using the formula:

D1 representation=2*normal representation+1

We can find the left figure is right.

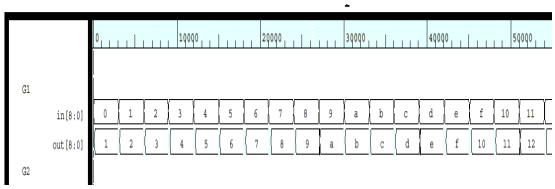
Code translator



The D1 representation=normal-1

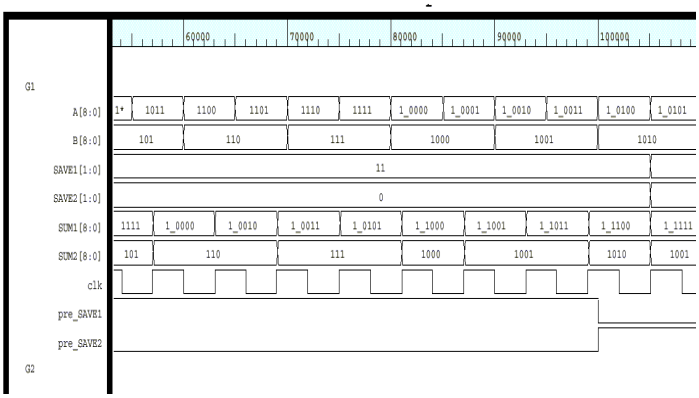
We can find the left figure is right

Inverse code translator



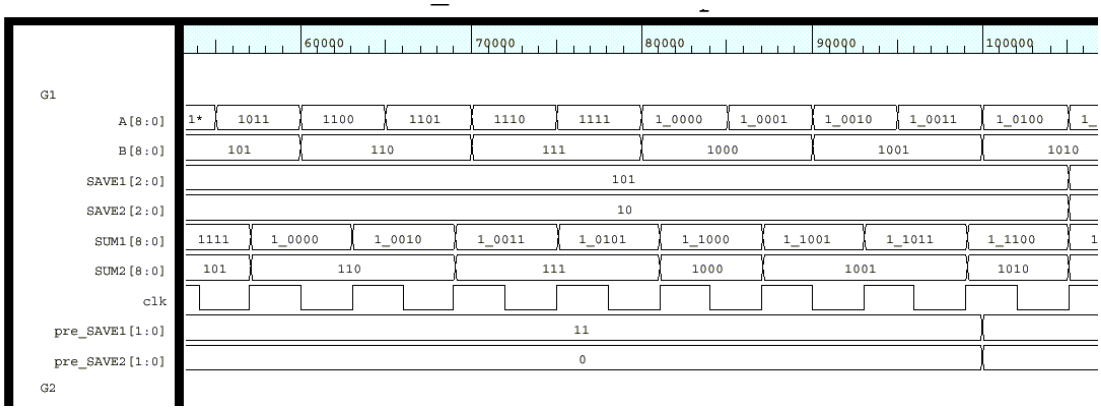
The normal = D1 representation +1
We can find the left figure is right

BF1

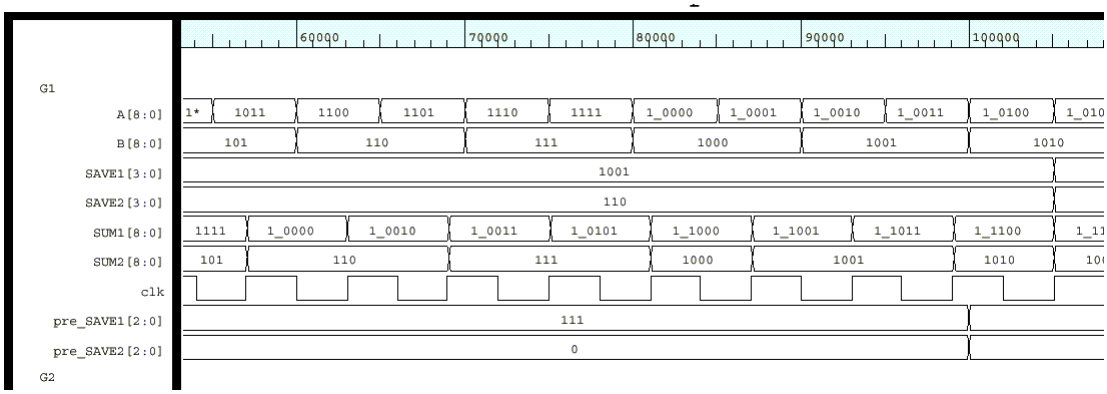


Sum1=A+B+1bit pre_save2
Sum2=A-B+1bit pre_save1
The data saved in save register can refer to the architecture above.

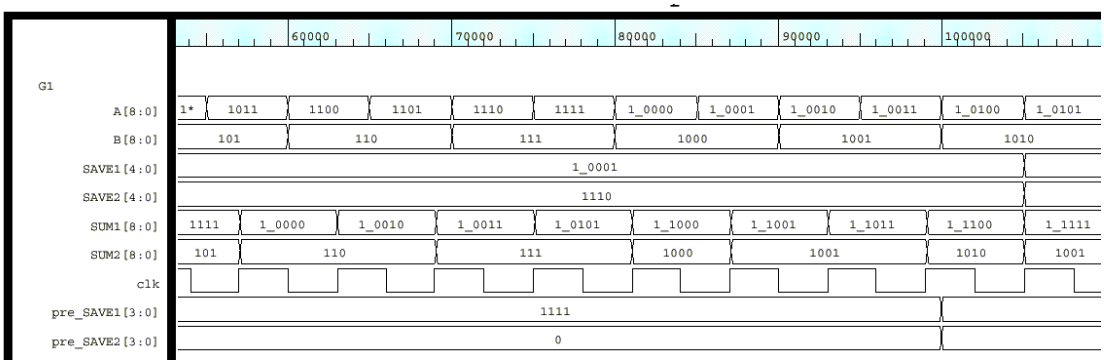
BF2



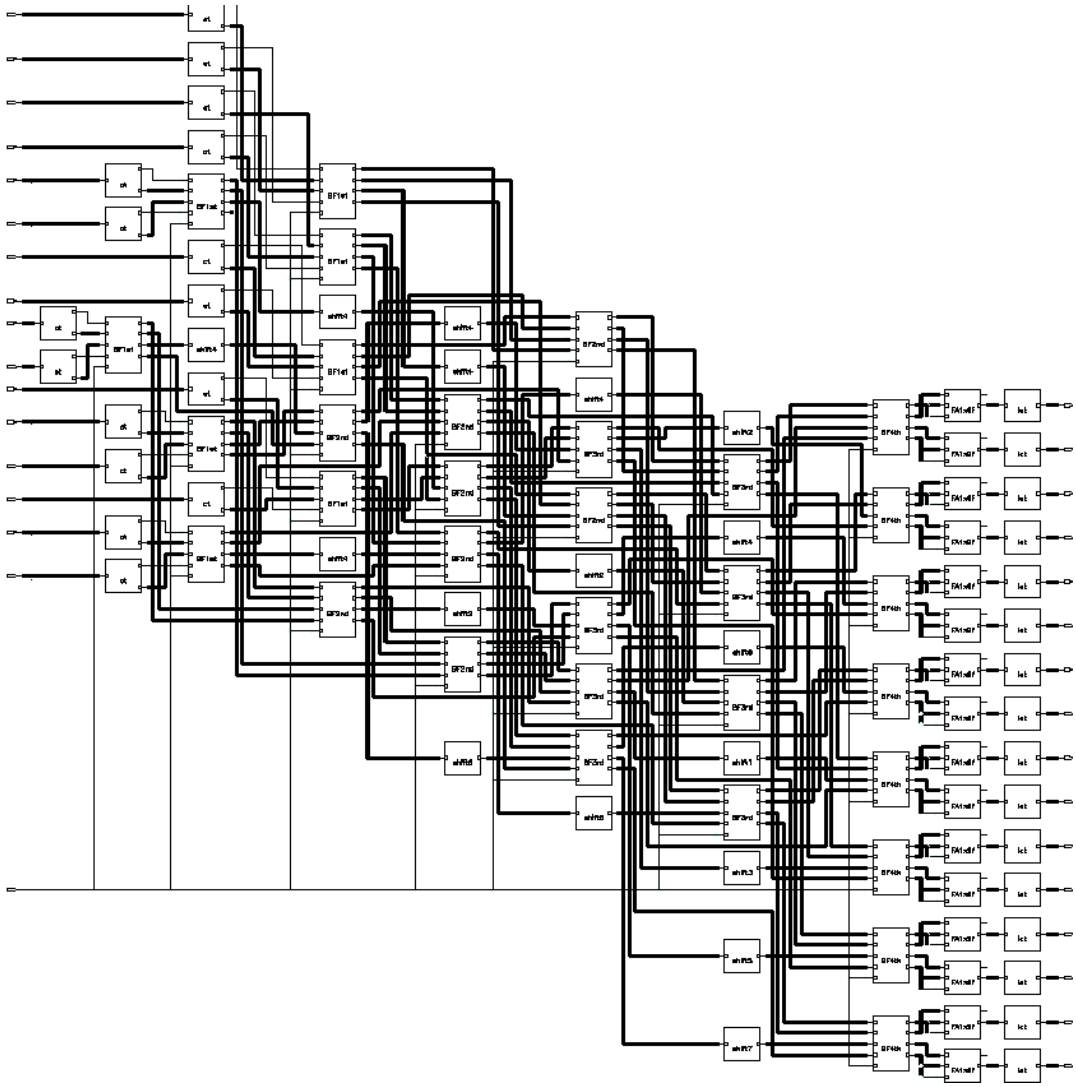
BF3



BF4



Whole structure



The simulation of the whole structure doesn’t work correctly. After debbuging I found it’s the result of the correction term doesn’t pass through the shifters. It should pass through shifter because if the correction term is added first then shift, the correction term is shifted. But this means the butterflies passed by the datapaths which are shifted must modified. The modified butterflies will need more registers to save the lengthened correction term. I want to design a filter originally, but its too complicated so I’ve just done the FNT.

Conclusion

In this final project, I study the NTT and FNT. And I verify the algorithm of FNT by Matlab. Finally I try to use verilog to implement the FNT in gate level. In the architecture design, I propose two schemes to improve the architecture of FNT:

(1) CSA-like butterfly architecture

(2) fast shifting

From the practical design I find in the small size transform, the FNT will not cost less than FFT. Because of the special representation method, it will need more gate count to accomplish the addition, subtraction, shifting, and code translation. But when transform size is large, the word length may grow to cost more gate count than the multipliers in FFT.

Reference:

[1] R. C. Agarwal and C. S. Burrus, "Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering," *IEEE. Trans. ASSP*, vol. 22, pp87-97, April. 1974.

[2] R. C. Agarwal and C. S. Burrus, "Number Theoretic Transform to Implement Fast Digital Convolution," *Proc. IEEE.*, vol. 63, pp550-560, April. 1975.

[3] L. M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," *IEEE Trans. ASSP.*, vol. 24, pp356-359, Oct. 1976.

[4] K. H. Rosen, "*ELEMENTARY NUMBER THEORY AND ITS APPLICATIONS*," Addison-Wesly, 1993.

[5] D. F. Elliott and K. R. Rao, "*FAST TRANSFORMS Algorithms, Analysis, Applications*,"