

Sookmyung Women's university

BPS LAB

- Process through Software on a universal Central Processing Unit(CPU) instead of using Hardware or Digital Signal Processor(DSP)



BPS LAB

- SIMD is something that can handle multiple data with one instruction which stands for Single Instruction, Multiple Data.
- Multiple data is a structure that has size of 4byte boundary like 64 bit (for MMX), 128bit(for SSE), 256 bit(for AVX). The data must be aligned on the memory and they can be processed by SIMD instruction at a time.



BPS LAB

Sookmyung Women's university

BPS LAB

Sookmyung Women's university

BPS LAB

Sookmyung Women's university

BPS LAB

- Necessary to implement a low-cost FFT algorithm that can perform OFDM symbol in real time



BPS LAB

Sookmyung Women's university

BPS LAB

Sookmyung Women's university

BPS LAB

- Changed **bflyNbit** function, which was generalized in functions 64-point FFT, 256-point FFT, and 1024-point FFT according to N.



BPS LAB

Improved FFT Algorithm- (2)

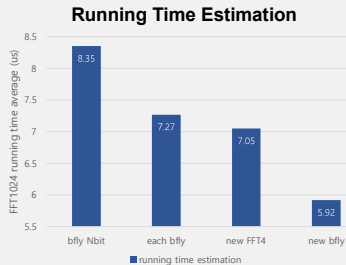
- Implemented a new 4-point FFT algorithm to reduce the use of `mm madd epi16` as much as possible. Since 4-point FFT is called repeatedly in 1024-point FFT, the time is reduced.

Bfly Nbit :FFT4  New FFT4 :FFT4

31 **12**

SIMD instruction	Estimation time [ns]
_mm_madd_epi16	0.1304
_mm_add_epi32	0.0887

Performance Analysis



Running time estimation				
	Bfly Nbit	Each bfly	New FFT4	New bfly
1024-point FFT (us)	8.35	7.27	7.05	5.92
Reduction Ratio(%)	Ref.	12.86	15.5	29.16

Improved FFT Algorithm- (3)

- We found some unnecessary calculations in the code and changed them to be calculated only once.

```

for(int k=0; k<N; k++)
{
    x218 = (_mm256i_t*)0;

    //list vnew's (15th output) q_4
    for(int j=0; j<N; j++) //-----
    {
        p16 = (_mm256i_t*)k*16+16*j;
        x218[0] = __mmask8_k16*16+16*j, x218, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+16;
        x218[1] = __mmask8_k16*16+16*j+16, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+32;
        x218[2] = __mmask8_k16*16+16*j+32, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+48;
        x218[3] = __mmask8_k16*16+16*j+48, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+64;
        x218[4] = __mmask8_k16*16+16*j+64, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+80;
        x218[5] = __mmask8_k16*16+16*j+80, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+96;
        x218[6] = __mmask8_k16*16+16*j+96, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+112;
        x218[7] = __mmask8_k16*16+16*j+112, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+128;
        x218[8] = __mmask8_k16*16+16*j+128, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+144;
        x218[9] = __mmask8_k16*16+16*j+144, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+160;
        x218[10] = __mmask8_k16*16+16*j+160, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+176;
        x218[11] = __mmask8_k16*16+16*j+176, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+192;
        x218[12] = __mmask8_k16*16+16*j+192, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+208;
        x218[13] = __mmask8_k16*16+16*j+208, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+224;
        x218[14] = __mmask8_k16*16+16*j+224, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+240;
        x218[15] = __mmask8_k16*16+16*j+240, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+256;
        x218[16] = __mmask8_k16*16+16*j+256, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+272;
        x218[17] = __mmask8_k16*16+16*j+272, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+288;
        x218[18] = __mmask8_k16*16+16*j+288, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+304;
        x218[19] = __mmask8_k16*16+16*j+304, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+320;
        x218[20] = __mmask8_k16*16+16*j+320, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+336;
        x218[21] = __mmask8_k16*16+16*j+336, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+352;
        x218[22] = __mmask8_k16*16+16*j+352, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+368;
        x218[23] = __mmask8_k16*16+16*j+368, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+384;
        x218[24] = __mmask8_k16*16+16*j+384, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+400;
        x218[25] = __mmask8_k16*16+16*j+400, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+416;
        x218[26] = __mmask8_k16*16+16*j+416, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+432;
        x218[27] = __mmask8_k16*16+16*j+432, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+448;
        x218[28] = __mmask8_k16*16+16*j+448, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+464;
        x218[29] = __mmask8_k16*16+16*j+464, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+480;
        x218[30] = __mmask8_k16*16+16*j+480, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+496;
        x218[31] = __mmask8_k16*16+16*j+496, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+512;
        x218[32] = __mmask8_k16*16+16*j+512, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+528;
        x218[33] = __mmask8_k16*16+16*j+528, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+544;
        x218[34] = __mmask8_k16*16+16*j+544, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+560;
        x218[35] = __mmask8_k16*16+16*j+560, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+576;
        x218[36] = __mmask8_k16*16+16*j+576, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+592;
        x218[37] = __mmask8_k16*16+16*j+592, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+608;
        x218[38] = __mmask8_k16*16+16*j+608, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+624;
        x218[39] = __mmask8_k16*16+16*j+624, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+640;
        x218[40] = __mmask8_k16*16+16*j+640, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+656;
        x218[41] = __mmask8_k16*16+16*j+656, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+672;
        x218[42] = __mmask8_k16*16+16*j+672, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+688;
        x218[43] = __mmask8_k16*16+16*j+688, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+704;
        x218[44] = __mmask8_k16*16+16*j+704, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+720;
        x218[45] = __mmask8_k16*16+16*j+720, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+736;
        x218[46] = __mmask8_k16*16+16*j+736, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+752;
        x218[47] = __mmask8_k16*16+16*j+752, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+768;
        x218[48] = __mmask8_k16*16+16*j+768, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+784;
        x218[49] = __mmask8_k16*16+16*j+784, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+800;
        x218[50] = __mmask8_k16*16+16*j+800, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+816;
        x218[51] = __mmask8_k16*16+16*j+816, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+832;
        x218[52] = __mmask8_k16*16+16*j+832, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+848;
        x218[53] = __mmask8_k16*16+16*j+848, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+864;
        x218[54] = __mmask8_k16*16+16*j+864, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+880;
        x218[55] = __mmask8_k16*16+16*j+880, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+896;
        x218[56] = __mmask8_k16*16+16*j+896, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+912;
        x218[57] = __mmask8_k16*16+16*j+912, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+928;
        x218[58] = __mmask8_k16*16+16*j+928, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+944;
        x218[59] = __mmask8_k16*16+16*j+944, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+960;
        x218[60] = __mmask8_k16*16+16*j+960, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+976;
        x218[61] = __mmask8_k16*16+16*j+976, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+992;
        x218[62] = __mmask8_k16*16+16*j+992, p16[0];
        p16 = (_mm256i_t*)k*16+16*j+1008;
        x218[63] = __mmask8_k16*16+16*j+1008, p16[0];
        p16 = (_mm256i_t*)k*
```

```

static inline __m128i _mm_xrsl_epi32(__m128i a, __m128i b)
{
    __m128i r;

    pH++;
    __r20b++;

    //final results, cumulation
    for(int i=0; i<4; i++)
    {
        r = _mm_add_epi32(_mm_srli_epi32(a, i), _mm_srli_epi32(b, i));
        r = _mm_add_epi32(r, _mm_srli_epi32(a, i+1));
        r = _mm_add_epi32(r, _mm_srli_epi32(b, i+1));
        r = _mm_add_epi32(r, _mm_srli_epi32(a, i+2));
        r = _mm_add_epi32(r, _mm_srli_epi32(b, i+2));
        r = _mm_add_epi32(r, _mm_srli_epi32(a, i+3));
        r = _mm_add_epi32(r, _mm_srli_epi32(b, i+3));

        //bit control
        r = _mm_packs_epi32(_mm_srli_epi32(r, 16), _mm_srli_epi32(r, 16));
        r = _mm_packs_epi32(r, _mm_srli_epi32(r, 16));

        y128[i] = _mm_unpacklo_epi16(_mm_srli_epi32(r, 16), _mm_srli_epi32(r, 16));
    }

    y128++;
    // = and for int32_t[i]; i++)
    // = and for int32_t[i]; i++)

    return(r);
}

//if defined, 64bit =

```

Conclusion

- **Design and implement an FFT algorithm** that can be processed in real time using the parallel processing function of the CPU.
- Confirmed that **our FFT algorithm can achieve the LTE synchronization process** in real time.
- By applying this, the 5G synchronization process will also be implemented in software, making it a **key technology in V2X**.