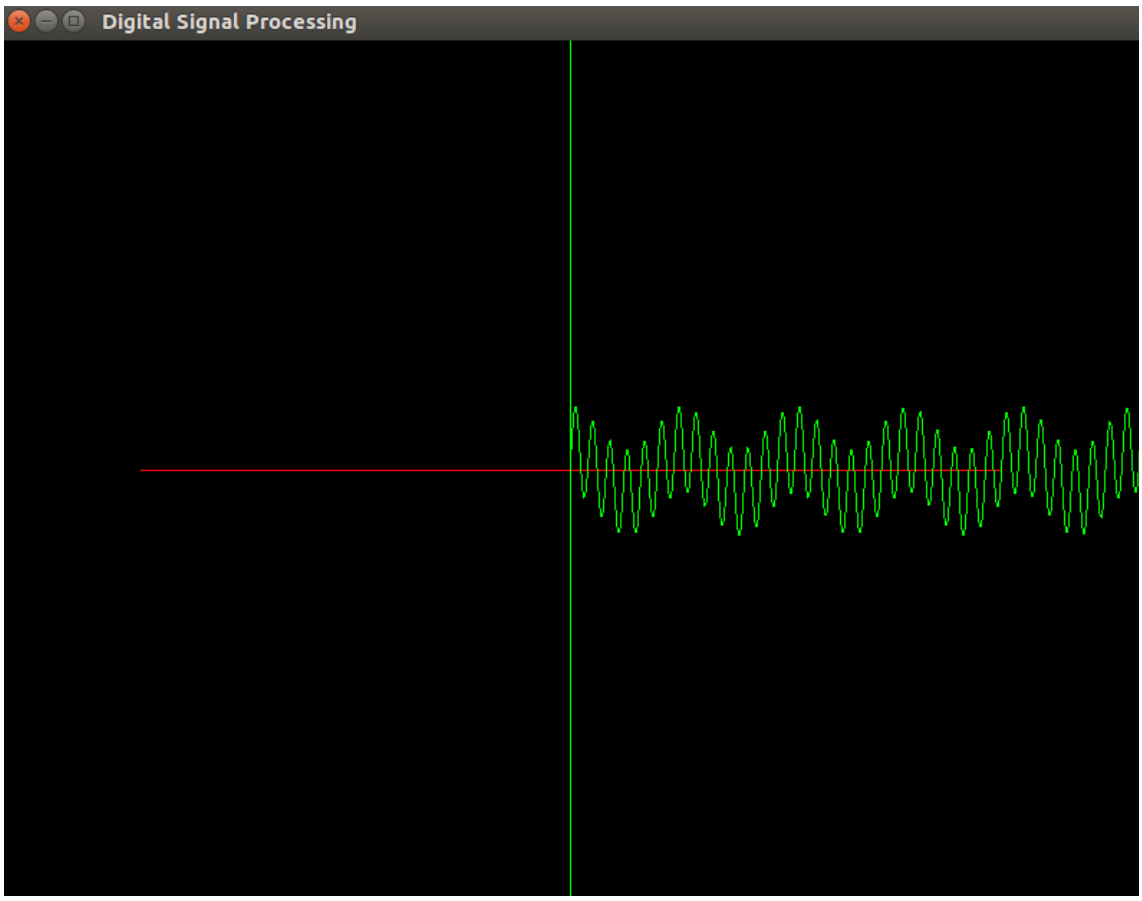
TI DSP, MCU 및 Xilinx Zynq FPGA
프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 하성용
accept0108@naver.com

73 일차
컴파일 : gcc (파일이름) -lGL -lglut -lGLU -lm



↑
와이파이- 5GHz
무선콘트롤러 - 700MHz

5G 랑 700MHz 가 합성된 신호이다
안테나로 구현해서 fpga 에 달면 그게 수신기가 됨

FFT 로 스펙트럼 분석
푸리엘트랜스폼해서 어떤 주파수가 섞여있는지 찾아야함
내가 수신하려고하거나 송신하려고하는 신호가 768MHz 인지 5GHz 인지
적절한 필터를 만들어내야함
필터만드는덴 라플라스 트랜스폼이 필요함

미분방정식 푸리엘, 라플라스를 알아야 필터를 만들수있음

이건 MCU, FPGA, DSP 다 필요함
푸리엘라인 참고
센서가 잡음을 타는데 잡음방지하려면 필터를 달아줘야함

5G_768M_signal.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <GL/glut.h>

#define G5_PERIOD           1.0 / 5000000000.0
#define CALC_5G_2PI         10000000000 * M_PI
#define CALC_NOISE_2PI      1536000000 * M_PI
```

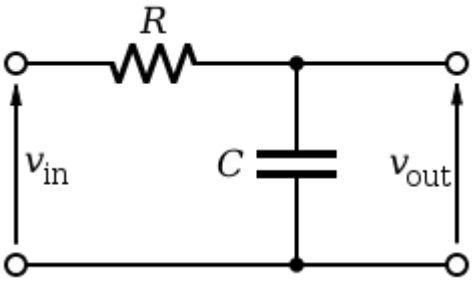
```
void draw_omega_sin(void);

float common_angles[5] = {15.0, 30.0, 45.0, 60.0, 75.0};
float freq_table[5] = {1000.0, 2400.0, 5000.0, 24000.0, 77000.0};

float theta = 0.0;

void display(void)
{
		glClearColor(0.0, 0.0, 0.0, 1.0);
		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
		glLoadIdentity();

		//gluLookAt(0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

		glColor3f(1, 0, 0);

		glBegin(GL_LINE_LOOP);
		glVertex3f(100.0, 0.0, 0.0);
	glVertex3f(-100.0, 0.0, 0.0);
	glEnd();

	glColor3f(0.0, 1.0, 0.0);

	glBegin(GL_LINE_LOOP);
	glVertex3f(0.0, 100.0, 0.0);
	glVertex3f(0.0, -100.0, 0.0);
	glEnd();

		draw_omega_sin();
		glutSwapBuffers();
}

#if 0
void reshape(int w, int h)
{
		glViewport(0, 0, (GLsizei)w, (GLsizei)h);
		glMatrixMode(GL_PROJECTION);
		glLoadIdentity();
		gluPerspective(60, (GLfloat)w / (GLfloat)h, 0.1, 100.0);
		glMatrixMode(GL_MODELVIEW);
}
#endif

void reshape(int w, int h)
{
	GLfloat n_range = 100.0f;

	if(h == 0)
		h = 1;

	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	if(w <= h)
		glOrtho(-n_range, n_range, -n_range * h / w, n_range * h / w, -n_range,
n_range);
	else
```

```
            glOrtho(-n_range * w / h, n_range * w / h, -n_range, n_range, -n_range,
n_range);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{
            switch(key)
            {
                        case 27:
                                    exit(0);
                                    break;
            }
}

void set_rand_amplitude(float *amp)
{
            *amp = rand() % 3 + 3;
}

void set_angle_with_common_angles(float *angle)
{
            *angle = common_angles[rand() % 5];
}

void angle2radian(float *angle, float *radian)
{
            *radian = *angle * M_PI / 180.0;
}

void radian2angle(float *angle, float *radian)
{
            *angle = *radian * 180.0 / M_PI;
}

void set_rand_frequency(float *freq)
{
            *freq = freq_table[rand() % 5];
}

void calc_period(float *freq, float *period)
{
            *period = 1 / (*freq);
}

void calc_angular_velocity(float *freq, float *ang_vel)
{
            *ang_vel = 2 * M_PI * (*freq);
}

float get_step(float slice, float period)
{
            return period / slice;
}

void draw_omega_sin(void)
{
            float amp, angle, period, freq, rad, omega, t, step = 0.0;
```

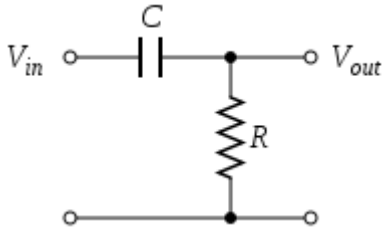
```
		float radius = 3.0;
		float x = 0, x2 = 0, y2, cx, cy;
		float tmp;
		int cache = 0;

		srand(time(NULL));

#if 0
		set_rand_amplitude(&amp);
		set_angle_with_common_angles(&angle);
		angle2radian(&angle, &rad);
		set_rand_frequency(&freq);
		calc_period(&freq, &period);
		calc_angular_velocity(&freq, &omega);
#endif

#if 1
		amp = 10;
		angle = 45.0;
		freq = 100.0;

		angle2radian(&angle, &rad);
		calc_period(&freq, &period);
		calc_angular_velocity(&freq, &omega);
#endif

#if 0
		printf("amplitude = %f\n", amp);
		printf("angle = %f degree\n", angle);
		printf("radian = %f\n", rad);
		printf("frequency = %f\n", freq);
		printf("period = %f\n", period);
		printf("angular_velocity = %f\n", omega);
#endif

		//t = step = get_step(SLICE, period);
		step = G5_PERIOD / 32.0;
		t = 0.0;

		//printf("t = %f\n", t);
#if 1
		if(t > 40 * G5_PERIOD)
			t = 0.0;
#endif

		glBegin(GL_LINES);
		for(; ; t += step)
		{
			if(t > 40 * G5_PERIOD)
			{
				break;
				t = 0.0;
			}

			//float rad_angle = angle * (M_PI / 180.0);
			//x2 += x;			// time += step;
			//x2 += 0.1;
			//y2 = amp * sin(CALC_5G_2PI * t);
			y2 = 10 * sin(CALC_5G_2PI * t) + 5 * cos(CALC_NOISE_2PI * t);
			//y2 = radius * sin((double)rad_angle);
```

```
#if 1
                        if(cache)
                        {
                                glVertex2f(cx * 20000000000.0, cy);
                                glVertex2f(t * 20000000000.0, y2);
                        }
#endif

#if 0
                        glVertex2f(t * 40000000000.0, y2);
#endif
                        cache = 1;
                        cx = t;
                        cy = y2;
                        //printf("t = %f, y2 = %f\n", t * 4000, y2);
                }
                glEnd();
}

int main(int argc, char **argv)
{
                float amplitude, angle, period, frequency, radian, angular_velocity;
                float step = 0.0;

                glutInit(&argc, argv);
                glutInitDisplayMode(GLUT_DOUBLE);
                glutInitWindowSize(800, 600);
                glutInitWindowPosition(0, 0);
                glutCreateWindow("Digital Signal Processing");

#if 0
                srand(time(NULL));

                set_rand_amplitude(&amplitude);
                set_angle_with_common_angles(&angle);
                angle2radian(&angle, &radian);
                set_rand_frequency(&frequency);
                calc_period(&frequency, &period);
                calc_angular_velocity(&frequency, &angular_velocity);

                printf("amplitude = %f\n", amplitude);
                printf("angle = %f degree\n", angle);
                printf("radian = %f\n", radian);
                printf("frequency = %f\n", frequency);
                printf("period = %f\n", period);
                printf("angular_velocity = %f\n", angular_velocity);

                cos_sim(amplitude, angular_velocity, period);
                sin_sim(amplitude, angular_velocity, period);
#endif

                glutDisplayFunc(display);
                //glutIdleFunc(display);
                glutReshapeFunc(reshape);
                //glutKeyboardFunc(keyboard);
                glutMainLoop();

                return 0;
}
```

< Low pass filter >



< High pass filter >



low pass filter 에
C(캐퍼시터)와 R(저항기)의 자리를 바꿔주면
High-pass filter 가 된다.
반대의 경우도 마찬가지

신호를 해석하는데 푸리에트랜스폼이 필요함
시스템자체를 해석하려면 라플라스 트랜스폼이 필요함

이것의 시스템자체를 해석하려면 라플라스가 필요함

식을 아래와 같이 쓸 수 있다.

$$v_{in}(t) - v_{out}(t) = R\, i(t)$$
$$Q_c(t) = C\, v_{out}(t)$$
$$i(t) = dQ_c/dt$$

V = IR 에서 직렬 회로에 해당하므로 전류가 커패시터에 흐르는 점을 응용하는 식이다.

$$x_i - y_i = RC\, y_i - y_{i-1} / \triangle_T$$

그리고 위의 식은 아날로그 값을 디지털 형태로 바꾼 것이다.

$$y_i = \overbrace{x_i\,(\triangle_T/RC+\triangle_T)}^{\text{Input contribution}} + \overbrace{y_{i-1}\,(RC/RC+\triangle_T)}^{\text{Inertia from previous output}}$$

이전에 작업했듯이 현재 y 값에 대해 정리를 해보면 위와 같이 정리된다.
현재값이 기여한 값과 이전 값이 기여한 값에 가중치가 붙는 형태라고 볼 수 있다.

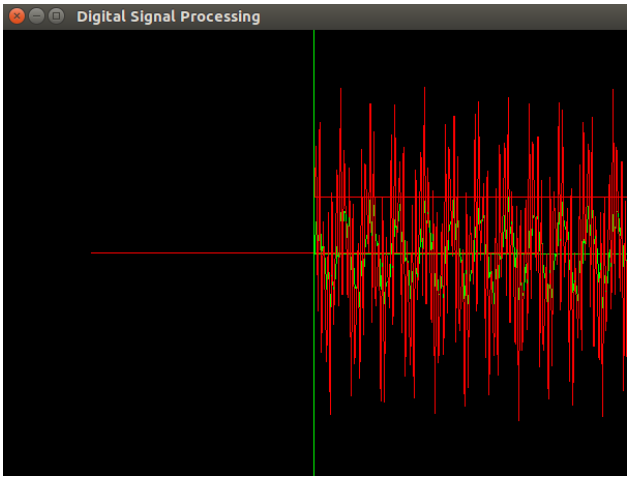
$$y_i = ax_i + (1 - a)y_{i-1} \qquad \text{where} \qquad a \overset{\triangle}{=} \triangle t/RC + \triangle t$$

알파라는 값을 위와 같이 지정하여 위의 식과 같이 변환한다.
그리고 RC 에 대해 정리하면 아래와 같이 정리된다.

$$RC = \triangle_T(\, 1\text{-}a/a)$$

lpf_signal.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>

#include <GL/glut.h>

int glob = 2;
int count = 1;

#define SLICE
        (1024)
#define HALF_SLICE
        (SLICE >> 1)
#define QUAD_SLICE
        (SLICE >> 2)
#define CALC_ORDER
        ((HALF_SLICE) + 1)
#define CALC_HEIGHT
        (SLICE - 3)

#define SAMPLE_FREQ             (12000000000.0)
#define SAMPLE_PERIOD           (1.0 / SAMPLE_FREQ)

#define CALC_5G_2PI             (10000000000 * M_PI)
//#define CALC_2_4G_2PI         (4800000000 * M_PI)
#define CALC_NOISE_2PI          (1536000000 * M_PI)

#define G5_PERIOD               (1.0 / 5000000000.0)

typedef struct complex
{
    double re;
    double im;
} c;

void draw_omega_sin(void);
void draw_spectrum(void);
void low_pass_filter(double *);
void spectrum_analysis(double *);

float common_angles[5] = {15.0, 30.0, 45.0, 60.0, 75.0};
float freq_table[5] = {1000.0, 2400.0, 5000.0, 24000.0, 77000.0};

float theta = 0.0;

void display(void)
{
        double lpf_signal[SLICE] = {0};

        glClearColor(0.0, 0.0, 0.0, 1.0);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();

        //gluLookAt(0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

        glColor3f(1, 0, 0);
```



```
		glBegin(GL_LINE_LOOP);
		glVertex3f(100.0, 0.0, 0.0);
	glVertex3f(-100.0, 0.0, 0.0);
	glEnd();

	glColor3f(0.0, 1.0, 0.0);

	glBegin(GL_LINE_LOOP);
	glVertex3f(0.0, 100.0, 0.0);
	glVertex3f(0.0, -100.0, 0.0);
	glEnd();

		//draw_omega_sin();
		//draw_spectrum();
		low_pass_filter(lpf_signal);
		spectrum_analysis(lpf_signal);
		glutSwapBuffers();
}

void reshape(int w, int h)
{
	GLfloat n_range = 100.0f;

	if(h == 0)
		h = 1;

	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	if(w <= h)
		glOrtho(-n_range, n_range, -n_range * h / w, n_range * h / w, -n_range,
n_range);
	else
		glOrtho(-n_range * w / h, n_range * w / h, -n_range, n_range, -n_range,
n_range);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{
		switch(key)
		{
			case 27:
				exit(0);
				break;
		}
}

void set_rand_amplitude(float *amp)
{
		*amp = rand() % 3 + 3;
}

void set_angle_with_common_angles(float *angle)
{
		*angle = common_angles[rand() % 5];
```

```
}

void angle2radian(float *angle, float *radian)
{
	*radian = *angle * M_PI / 180.0;
}

void radian2angle(float *angle, float *radian)
{
	*angle = *radian * 180.0 / M_PI;
}

void set_rand_frequency(float *freq)
{
	*freq = freq_table[rand() % 5];
}

void calc_period(float *freq, float *period)
{
	*period = 1 / (*freq);
}

void calc_angular_velocity(float *freq, float *ang_vel)
{
	*ang_vel = 2 * M_PI * (*freq);
}

float get_step(float slice, float period)
{
	return period / slice;
}

static double rt_hypotd_snf(double u0, double u1)
{
	double y;
	double a;
	double b;
	a = fabs(u0);
	b = fabs(u1);
	if (a < b) {
		a /= b;
		y = b * sqrt(a * a + 1.0);
	} else if (a > b) {
		b /= a;
		y = a * sqrt(b * b + 1.0);
	//} else if (rtIsNaN(b)) {
	} else if(b == 0.0){
		y = b;
	} else {
		y = a * 1.4142135623730951;
	}

	return y;
}

void find_frequency(c X[SLICE], double R[CALC_ORDER], double f[HALF_SLICE])
{
	double P2[SLICE];
	c y[SLICE];
	int k;
```

```
		for (k = 0; k < SLICE; k++) {
#if 0
			if (X[k].im == 0.0) {
				y[k].re = X[k].re / 256.0;
				y[k].im = 0.0;
			} else if (X[k].re == 0.0) {
				y[k].re = 0.0;
				y[k].im = X[k].im / 256.0;
			} else {
#endif
			y[k].re = X[k].re / SLICE;
			y[k].im = X[k].im / SLICE;
//			}

			P2[k] = rt_hypotd_snf(y[k].re, y[k].im);
			//printf("P2[%d] = %lf\n", k, P2[k]);
		}

		memcpy(&R[0], &P2[0], CALC_ORDER * sizeof(double));
		for (k = 0; k < HALF_SLICE - 1; k++) {
			R[1 + k] = 2.0 * P2[1 + k];
			//printf("R[%d] = %lf\n", k + 1, R[k + 1]);
		}

		for(k = 0; k < CALC_ORDER; k++)
		{
			f[k] = SAMPLE_FREQ * k / SLICE;
			//printf("f[%d] frequency = %lf\n", k, f[k]);
		}
}

void low_pass_filter(double *lpf)
{
		int i;
		double t = 0.0;
		double signal[SLICE] = {0};
		double fc = 800000000.0;
		double rc = 1.0 / (2 * M_PI * fc);

		printf("Original Signal\n");
		for(i = 0; i < SLICE; t += SAMPLE_PERIOD, i++)
		{
		 signal[i] = 10 * sin(CALC_5G_2PI * t) + 5 * cos(CALC_NOISE_2PI * t);
			printf("signal[%d] = %lf\n", i, signal[i]);
		}

		printf("RC Low Pass Filter\n");
		for(i = 1; i < SLICE; i++)
		{
			lpf[i] = (rc * lpf[i - 1] + SAMPLE_PERIOD * signal[i]) / (rc +
SAMPLE_PERIOD);
			printf("lpf[%d] = %lf\n", i, lpf[i]);
		}

#if 0
		for(i = 0; i < SLICE; i++)
			fout[i] = signal[i + 1] + fc * (signal[i + 1] - signal[i]) /
SAMPLE_PERIOD;
#endif
}
```

```
void spectrum_analysis(double *lpf)
{
	double t = 0, period, freq = SLICE, step = 0.0;
	double temp_re, temp_im, twid_re, twid_im;
	double x = 0, x2 = 0, y2, cx, cy, orig_y;
	double dv0[CALC_ORDER] = {0};
	double dv1[CALC_ORDER] = {0};
	double rf[CALC_ORDER] = {0};
	double f[CALC_ORDER] = {0};
	double signal[SLICE] = {0};
	int cache;

	c y[SLICE] = {0};
	int ix = 0, ju = 0, iy = 0, tst, iheight, istart, ihi, i, j;

	//step = 2 * M_PI / SLICE;
	step = G5_PERIOD / SLICE;

	for(i = 0; i < SLICE; i++)
		printf("lpf[%d] = %lf\n", i, lpf[i]);

	for(i = 0; i < SLICE; t += SAMPLE_PERIOD, i++)
	 signal[i] = 10 * sin(CALC_5G_2PI * t) + 5 * cos(CALC_NOISE_2PI * t);

	t = 0.0;
	i = 0;

	glBegin(GL_LINES);
	for(; ; t += step)
	{
#if 1
#if 0
			if(t > 40 * G5_PERIOD)
			{
				t = 0.0;
				break;
			}
#endif
			if(i > 1023)
			{
				t = 0.0;
				break;
			}

			if(cache)
			{
			glColor3f(0.0, 1.0, 0.0);
				glVertex2f(cx * 4000000000000.0, cy * 5);
				glVertex2f(t * 4000000000000.0, lpf[i] * 5);

			glColor3f(1.0, 0.0, 0.0);
				glVertex2f(cx * 4000000000000.0, orig_y * 5);
				glVertex2f(t * 4000000000000.0, signal[i] * 5);
			}

			cache = 1;
			cx = t;
			cy = lpf[i];
```

```
                        orig_y = signal[i];

                        //printf("lpf[%d] = %lf\n", i, lpf[i]);
                        i++;
#endif
                }
                glEnd();
                glob = 2;
                i = 0;
}

int main(int argc, char **argv)
{
                float amplitude, angle, period, frequency, radian, angular_velocity;
                float step = 0.0;

                glutInit(&argc, argv);
                glutInitDisplayMode(GLUT_DOUBLE);
                glutInitWindowSize(400, 200);
                glutInitWindowPosition(0, 0);
                glutCreateWindow("Digital Signal Processing");

                glutDisplayFunc(display);
                glutReshapeFunc(reshape);
                glutMainLoop();

                return 0;
}
```