I DSP,Xilinx zynq FPGA,MCU 및 Xilinx zynq FPGA 프로그래밍 전문가 과정

강사-INNOVA LEE(이상훈)
Gccompil3r@gmail.com
학생-윤지완
Yoonjw789 @naver.com

```
#include<stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void init_mat(float (*A)[3])
{
	int i, j;

	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; j++)
			A[i][j] = rand() % 4;
}

void print_mat(float (*R)[3])
{
	int i, j;

	for(i = 0; i < 3; i++)
	{
		for(j = 0; j < 3; j++)
			printf("%10.4f", R[i][j]);
		printf("\n");
	}
	printf("\n");
}

void add_mat(float (*A)[3], float (*B)[3], float (*R)[3])
{
	int i, j;

	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; j++)
			R[i][j] = A[i][j] + B[i][j];
}

void sub_mat(float (*A)[3], float (*B)[3], float (*R)[3])
{
	int i, j;

	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; j++)
			R[i][j] = A[i][j] - B[i][j];
}
```

```
void scale_mat(float scale_factor, float (*A)[3], float (*R)[3])
{
	int i, j;

	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; j++)
			R[i][j] = scale_factor * A[i][j];
}

#if 0
A[0][0] A[0][1] A[0][2]		B[0][0] B[0][1] B[0][2]
A[1][0] A[1][1] A[1][2]		B[1][0] B[1][1] B[1][2]
A[2][0] A[2][1] A[2][2]		B[2][0] B[2][1] B[2][2]

A[0][0]*B[0][0]+A[0][1]*B[1][0]+A[0][2]*B[2][0]		A[0][0]*B[0][1]+A[0][1]*B[1][1]+A[0][2]*B[2][1]		A[0][0]*B[0][2]+A[0][1]*B[1][2]+A[0][2]*B[2][2]
A[1][0]*B[0][0]+A[1][1]*B[1][0]+A[1][2]*B[2][0]		A[1][0]*B[0][1]+A[1][1]*B[1][1]+A[1][2]*B[2][1]		A[1][0]*B[0][2]+A[1][1]*B[1][2]+A[1][2]*B[2][2]
A[2][0]*B[0][0]+A[2][1]*B[1][0]+A[2][2]*B[2][0]		A[2][0]*B[0][1]+A[2][1]*B[1][1]+A[2][2]*B[2][1]		A[2][0]*B[0][2]+A[2][1]*B[1][2]+A[2][2]*B[2][2]
#endif

void mul_mat(float (*A)[3], float (*B)[3], float (*R)[3])
{
	R[0][0] = A[0][0]*B[0][0]+A[0][1]*B[1][0]+A[0][2]*B[2][0];
	R[0][1] = A[0][0]*B[0][1]+A[0][1]*B[1][1]+A[0][2]*B[2][1];
	R[0][2] = A[0][0]*B[0][2]+A[0][1]*B[1][2]+A[0][2]*B[2][2];

	R[1][0] = A[1][0]*B[0][0]+A[1][1]*B[1][0]+A[1][2]*B[2][0];
	R[1][1] = A[1][0]*B[0][1]+A[1][1]*B[1][1]+A[1][2]*B[2][1];
	R[1][2] = A[1][0]*B[0][2]+A[1][1]*B[1][2]+A[1][2]*B[2][2];

	R[2][0] = A[2][0]*B[0][0]+A[2][1]*B[1][0]+A[2][2]*B[2][0];
	R[2][1] = A[2][0]*B[0][1]+A[2][1]*B[1][1]+A[2][2]*B[2][1];
	R[2][2] = A[2][0]*B[0][2]+A[2][1]*B[1][2]+A[2][2]*B[2][2];
}

float det_mat(float (*A)[3])
{
	return A[0][0] * (A[1][1] * A[2][2] - A[1][2] * A[2][1]) +
		    A[0][1] * (A[1][2] * A[2][0] - A[1][0] * A[2][2]) +
		    A[0][2] * (A[1][0] * A[2][1] - A[1][1] * A[2][0]);
}

void trans_mat(float (*A)[3], float (*R)[3])
{
	R[0][0] = A[0][0];
	R[1][1] = A[1][1];
	R[2][2] = A[2][2];

	R[0][1] = A[1][0];
	R[1][0] = A[0][1];
```

```
	R[0][2] = A[2][0];
	R[2][0] = A[0][2];

	R[2][1] = A[1][2];
	R[1][2] = A[2][1];
}

#if 0
	R[0][1] = A[1][2] * A[2][0] - A[1][0] * A[2][2];
	R[0][2] = A[1][0] * A[2][1] - A[1][1] * A[2][0];

	R[1][0] = A[0][2] * A[2][1] - A[0][1] * A[2][2];
	R[1][2] = A[0][1] * A[2][0] - A[0][0] * A[2][1];

	R[2][0] = A[0][1] * A[1][2] - A[0][2] * A[1][1];
	R[2][1] = A[0][2] * A[1][0] - A[0][0] * A[1][2];
#endif

void adj_mat(float (*A)[3], float (*R)[3])
{

  R[0][0] = A[1][1] * A[2][2] - A[1][2] * A[2][1];
	R[0][1] = A[0][2] * A[2][1] - A[0][1] * A[2][2];
	R[0][2] = A[0][1] * A[1][2] - A[0][2] * A[1][1];

	R[1][0] = A[1][2] * A[2][0] - A[1][0] * A[2][2];
	R[1][1] = A[0][0] * A[2][2] - A[0][2] * A[2][0];
	R[1][2] = A[0][2] * A[1][0] - A[0][0] * A[1][2];

	R[2][0] = A[1][0] * A[2][1] - A[1][1] * A[2][0];
	R[2][1] = A[0][1] * A[2][0] - A[0][0] * A[2][1];
	R[2][2] = A[0][0] * A[1][1] - A[0][1] * A[1][0];
}

bool inv_mat(float (*A)[3], float (*R)[3])
{
	float det;

	det = det_mat(A);

	if(det == 0.0)
		return false;

	adj_mat(A, R);
#ifdef __DEBUG__
	printf("Adjoint Matrix\n");
	print_mat(R);
#endif
	scale_mat(1.0 / det, R, R);

	return true;
```

```
}

void molding_mat(float (*A)[3], float *ans, int idx, float (*R)[3])
{
        int i, j;

        for(i = 0; i < 3; i++)
        {
                for(j = 0; j < 3; j++)
                {
                        if(j == idx)
                                continue;
                        R[i][j] = A[i][j];
                }

                R[i][idx] = ans[i];
        }
}

void crammer_formula(float (*A)[3], float *ans, float *xyz)
{
        float detA, detX, detY, detZ;
        float R[3][3] = {};

        detA = det_mat(A);

        molding_mat(A, ans, 0, R);
#ifdef __DEBUG__
        print_mat(R);
       molding_mat(A, ans, 1, R);
#ifdef __DEBUG__
        print_mat(R);
#endif
        detY = det_mat(R);

        molding_mat(A, ans, 2, R);
#ifdef __DEBUG__
        print_mat(R);
#endif
        detZ = det_mat(R);

        xyz[0] = detX / detA;
        xyz[1] = detY / detA;
        xyz[2] = detZ / detA;
}

void print_vec3(float *vec)
{
        int i;

        for(i = 0; i < 3; i++)
                printf("%10.4f", vec[i]);
```

```
    printf("\n");
}

int main(void)
{
    bool inv_flag;

    float test[3][3] = {{2.0, 0.0, 4.0}, {0.0, 3.0, 9.0}, {0.0, 0.0, 1.0}};
    float stimul[3][3] = {{2.0, 4.0, 4.0}, {6.0, 2.0, 2.0}, {4.0, 2.0, 4.0}};
    float ans[3] = {12.0, 16.0, 20.0};
    float xyz[3] = {};

    float A[3][3] = {};
    float B[3][3] = {};
    float R[3][3] = {};

    srand(time(NULL));

    printf("Init A Matrix\n");
    init_mat(A);
    print_mat(A);

    printf("Init B Matrix\n");
    init_mat(B);
    print_mat(B);

    printf("A + B Matrix\n");
    add_mat(A, B, R);
    print_mat(R);

    printf("A - B Matrix\n");
    sub_mat(A, B, R);
    print_mat(R);
 printf("A - B Matrix\n");
    sub_mat(A, B, R);
    print_mat(R);

    printf("Matrix Scale(A)\n");
    scale_mat(0.5, A, R);
    print_mat(R);

    printf("AB Matrix\n");
    mul_mat(A, B, R);
    print_mat(R);

    printf("det(A) = %f\n", det_mat(A));
    printf("det(B) = %f\n", det_mat(B));

    printf("\nA^T(Transpose) Matrix\n");
    trans_mat(A, R);
    print_mat(R);
```

```
	printf("B^T(Transpose) Matrix\n");
	trans_mat(B, R);
	print_mat(R);

	printf("A Inverse Matrix\n");
	inv_flag = inv_mat(A, R);
	if(inv_flag)
		print_mat(R);
	else
		printf("역행렬 없다!\n");

	printf("test Inverse Matrix\n");
	inv_flag = inv_mat(test, R);
	if(inv_flag)
		print_mat(R);
	else
		printf("역행렬 없다!\n");

	printf("크래머 공식 기반 연립 방정식 풀기!\n2x + 4y + 4z = 12\n6x + 2y + 2z = 16\n4x + 2y + 4z = 
20\n");
	crammer_formula(stimul, ans, xyz);
	print_vec3(xyz);

	return 0;
}
```

<가오스 소거법 >

```
void gauss_elimination(float (*A)[3], float *ans, float *xyz)
 {
	float R[3][4] = {};
-	float scale;

	create_3x4_mat(A, ans, R);
 #if __DEBUG__
@@ -276,6 +275,81 @@ void gauss_elimination(float (*A)[3], float *ans, float *xyz)
	finalize(R, xyz);
 }

+void create_3x6_mat(float (*A)[3], float (*R)[6])
+{
+	int i, j;
+
+	for(i = 0; i < 3; i++)
+		for(j = 0; j < 3; j++)
+		{
+			R[i][j] = A[i][j];
+
+			if(i == j)
```

```
+                               R[i][j + 3] = 1;
+                       else
+                               R[i][j + 3] = 0;
+               }
+}
+
+void print_3x6_mat(float (*R)[6])
+{
+       int i, j;
+
+       for(i = 0; i < 3; i++)
+    {
+        for(j = 0; j < 6; j++)
+            printf("%10.4f", R[i][j]);
+        printf("\n");
+    }
+    printf("\n");
+}
+
+void adjust_3x6_mat(float (*A)[6], int idx, float (*R)[6])
+{
+    int i, j;
+    float div_factor, scale;
+
+       scale = A[idx][idx];
+
+    for(i = idx + 1; i < 3; i++)
+    {
+        //div_factor = -A[idx][idx] / A[idx + 1][idx];
+        //div_factor = -A[idx + 1][idx] / A[idx][idx];
+        //div_factor = -A[i][0] / A[idx][0];
+        div_factor = -A[i][idx] / A[idx][idx];
+        printf("div_factor = %f\n", div_factor);
+
+               if(div_factor == 0.0)
+                       continue;
+        for(j = 0; j < 6; j++)
+            R[i][j] = A[idx][j] * div_factor + A[i][j];
+    }
+
+       for(j = 0; j < 6; j++)
+               R[idx][j] = A[idx][j] / scale;
+}
+
+void gauss_elim_mat(float (*A)[3], float (*R)[3])
+{
+       float mid[3][6] = {};
+
+       create_3x6_mat(A, mid);
+#if __DEBUG__
+       print_3x6_mat(mid);
+#endif
```

```
+
+       adjust_3x6_mat(mid, 0, mid);
+#if __DEBUG__
+    print_3x6_mat(mid);
+#endif
+
+       adjust_3x6_mat(mid, 1, mid);
+#if __DEBUG__
+    print_3x6_mat(mid);
+#endif
+}
+
 int main(void)
 {
        bool inv_flag;
@@ -348,5 +422,9 @@ int main(void)
        gauss_elimination(stimul, ans, xyz);
        print_vec3(xyz);

+       printf("가우스 소거법으로 역행렬 구하기!\n");
+       gauss_elim_mat(test, R);
+       print_mat(R);
+
        return 0;
 }
```

<가우스 소거법>

$$\begin{pmatrix} 1 & 0 & 0 \\ 3/2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3/2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 8 \\ -11 \\ -3 \end{pmatrix}$$

즉,

$$\begin{pmatrix} 2 & 1 & -1 \\ 0 & 1/2 & 1/2 \\ 0 & 2 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3/2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}\begin{pmatrix} 8 \\ -11 \\ -3 \end{pmatrix}$$

이므로, 이 두 과정은 같은 과정임을 알 수 있다.

이제 벡터 (x, y, z) 와 (8, −11, −3) 을 계속 가지고 다닐 필요가 없으므로 위의 식을 간단하게

$$\left(\begin{array}{ccc:ccc} 2 & 1 & -1 & 1 & 0 & 0 \\ 0 & 1/2 & 1/2 & 3/2 & 1 & 0 \\ 0 & 2 & 1 & 1 & 0 & 1 \end{array}\right)$$

와 같이 쓰기로 한다. 이제 마지막 식에서 y 를 소거하기 위하여 $R_3 - 4\,R_2$ 를 새로운 R_3 로 취하자.

$$\left(\begin{array}{ccc:ccc} 2 & 1 & -1 & 1 & 0 & 0 \\ 0 & 1/2 & 1/2 & 3/2 & 1 & 0 \\ 0 & 0 & -1 & 5 & -4 & 1 \end{array}\right)$$

<그래머 소거법>

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \longrightarrow \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} e \\ f \end{bmatrix}$$

우리는 간단히 행렬로 나타낼 수 있다.

$$x = \frac{\begin{vmatrix} e & b \\ f & d \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}} = \frac{ed - bf}{ad - bc} \qquad y = \frac{\begin{vmatrix} a & e \\ c & f \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}} = \frac{af - ec}{ad - bc}$$

일반화하면

$$\det(A) = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} \qquad \det(A1) = \begin{vmatrix} b_1 & a_{12} & \cdots & a_{1n} \\ b_2 & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ b_n & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

$$\det(A2) = \begin{vmatrix} a_{11} & b_1 & \cdots & a_{1n} \\ a_{21} & b_2 & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & b_n & \cdots & a_{nn} \end{vmatrix} \qquad \det(Ai) = \begin{vmatrix} a_{11} & a_{12} & \cdots & b_1 \\ a_{21} & a_{22} & \cdots & b_2 \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & b_n \end{vmatrix}$$

$$X = \frac{\det(A1)}{\det(A)} \qquad Y = \frac{\det(A2)}{\det(A)}$$

$$Z = \frac{\det(A3)}{\det(A)} \qquad Xi = \frac{\det(Ai)}{\det(A)}$$