Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 장성환
redmk1025@gmail.com

```
#include <stdio.h>

typedef struct _vector_man{
   double x;
   double y;
   double z;

}vector;

void print_mat(vector mat[]){
   int i;
   for(i=0; i<3; i++){
         printf("%5.2lf ",mat[i].x);
         printf("%5.2lf ",mat[i].y);
         printf("%5.2lf ",mat[i].z);

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

void madd(vector mat1[], vector mat2[], vector (*mat_res)[] ){
   int i;

   for(i=0; i<3; i++){
         (*mat_res)[i].x = mat1[i].x + mat2[i].x;
         (*mat_res)[i].y = mat1[i].y + mat2[i].y;
         (*mat_res)[i].z = mat1[i].z + mat2[i].z;
   }
}

void msub(vector mat1[], vector mat2[], vector (*mat_res)[]){
```

```
  int i;

  for(i=0; i<3; i++){
        (*mat_res)[i].x = mat1[i].x - mat2[i].x;
        (*mat_res)[i].y = mat1[i].y - mat2[i].y;
        (*mat_res)[i].z = mat1[i].z - mat2[i].z;
  }

}

void mmul(vector mat1[], vector mat2[], vector (*mat_res)[]){
  int i,j;

  for(i=0; i<3; i++){
            (*mat_res)[i].x = mat1[i].x * mat2[0].x + mat1[i].y * mat2[1].x + mat1[i].z * mat2[2].x;

            (*mat_res)[i].y = mat1[i].x * mat2[0].y + mat1[i].y * mat2[1].y + mat1[i].z * mat2[2].y;

            (*mat_res)[i].z = mat1[i].x * mat2[0].z + mat1[i].y * mat2[1].z + mat1[i].z * mat2[2].z;
  }
}

double det(vector mat[]){
  double tmp[3];
  tmp [0] = mat[0].x * (mat[1].y*mat[2].z - mat[2].y*mat[1].z);
  tmp [1] = mat[0].y * (mat[1].z*mat[2].x - mat[1].x*mat[2].z);
  tmp [2] = mat[0].z * (mat[1].x*mat[2].y - mat[2].x*mat[1].y);

  return tmp[0] + tmp[1] + tmp[2] ;
}

void mtranspose(vector mat[], vector (*mat_res)[]){
```

```
  (*mat_res)[0].x = mat[0].x;
  (*mat_res)[1].y = mat[1].y;
  (*mat_res)[2].z = mat[2].z;
  (*mat_res)[0].y = mat[1].x;
  (*mat_res)[0].z = mat[2].x;
  (*mat_res)[1].z = mat[2].y;
  (*mat_res)[1].x = mat[0].y;
  (*mat_res)[2].x = mat[0].z;
  (*mat_res)[2].y = mat[1].z;
}

void cramer(vector mat[],vector single_mat[], vector (*res_single_mat)[]){
  int i,j,k;
  vector tmp[3]={0};
  vector origin[3] = {0};
  double basic_det=0;
  double tmp_det=0;

  for(i=0; i<3; i++){ // clear initial values
        tmp[i].x = mat[i].x;
        tmp[i].y = mat[i].y;
        tmp[i].z = mat[i].z;
  }

  basic_det = det(mat);

  for(i=0; i<3; i++){
        if(i == 0){
           for(j=0 ; j<3 ; j++)
                tmp[j].x = single_mat[j].x;
        }
        else if(i == 1){
```

```
            for(j=0 ; j<3 ; j++)
                tmp[j].y = single_mat[j].x;
        }
        else{
            for(j=0 ; j<3 ; j++)
                tmp[j].z = single_mat[j].x;
        }

        tmp_det = det(tmp);
        (*res_single_mat)[i].x = tmp_det / basic_det;

        for(k=0; k<3; k++){ // clear initial values
            tmp[k].x = mat[k].x;
            tmp[k].y = mat[k].y;
            tmp[k].z = mat[k].z;
        }
    }
}

void adj(vector mat[], vector (*mat_res)[]){
    int i;
    vector tmp[3]={0};
    vector res[3]={0};

    tmp[0].x = (mat[1].y*mat[2].z - mat[2].y*mat[1].z);
    tmp[0].y = (mat[1].z*mat[2].x - mat[1].x*mat[2].z);
    tmp[0].z = (mat[1].x*mat[2].y - mat[2].x*mat[1].y);

    tmp[1].x = (mat[0].y*mat[2].z - mat[2].y*mat[0].z);
    tmp[1].y = (mat[2].z*mat[0].x - mat[2].x*mat[0].z);
    tmp[1].z = (mat[0].x*mat[2].y - mat[2].x*mat[0].y);

    tmp[2].x = (mat[0].y*mat[1].z - mat[1].y*mat[0].z);
```

```
  tmp[2].y = (mat[0].z*mat[1].x - mat[0].x*mat[1].z);
  tmp[2].z = (mat[0].x*mat[1].y - mat[1].x*mat[0].y);

  mtranspose(tmp, &res);

  for(i=0; i<3; i++){
    (*mat_res)[i].x = res[i].x;
    (*mat_res)[i].y = res[i].y;
    (*mat_res)[i].z = res[i].z;
  }
}

void inverse_mat(vector mat[], vector (*mat_res)[]){
  int i;
  vector tmp[3]= {0};
  double m_det = det(mat);

  if(det != 0){
    adj(mat, &tmp);

    for(i=0; i<3; i++){
      (*mat_res)[i].x = (1/m_det)*tmp[i].x;
      (*mat_res)[i].y = (1/m_det)*tmp[i].y;
      (*mat_res)[i].z = (1/m_det)*tmp[i].z;
    }
  }
  else{
    printf("No det value !! result!!\n");
  }
}
```

```
void gauss_inverse_mat(vector mat[], vector (*mat_res)[]){
  int i;
  vector imat[3] = {{1,0,0},{0,1,0},{0,0,1}};
  vector bmat[3] = {0};
  double scale =0;
  double sub_scale[3]={0};

  if(mat[1].x != 0){
    scale = (mat[1].x/mat[0].x);

      imat[1].x = imat[1].x - (scale * imat[0].x);
      imat[1].y = imat[1].y - (scale * imat[0].y);
      imat[1].z = imat[1].z - (scale * imat[0].z);

      mat[1].x = mat[1].x - (scale * mat[0].x);
      mat[1].y = mat[1].y - (scale * mat[0].y);
      mat[1].z = mat[1].z - (scale * mat[0].z);
  }

  if(mat[2].x != 0){
    scale = (mat[2].x/mat[0].x);

      imat[2].x -= (scale * imat[0].x);
      imat[2].y -= (scale * imat[0].y);
      imat[2].z -= (scale * imat[0].z);

      mat[2].x -= (scale * mat[0].x);
      mat[2].y -= (scale * mat[0].y);
      mat[2].z -= (scale * mat[0].z);
  }
```

```
//////////////////////////////////////

  if(mat[0].y != 0){
    scale = (mat[0].y/mat[1].y);
      imat[0].x -= scale * imat[1].x;
      imat[0].y -= scale * imat[1].y;
      imat[0].z -= scale * imat[1].z;

      mat[0].x -= scale * mat[1].x;
      mat[0].y -= scale * mat[1].y;
      mat[0].z -= scale * mat[1].z;
  }

  if(mat[2].y != 0){
    scale = (mat[2].y/mat[1].y);
      imat[2].x -= scale * imat[1].x;
      imat[2].y -= scale * imat[1].y;
      imat[2].z -= scale * imat[1].z;

      mat[2].x -= scale * mat[1].x;
      mat[2].y -= scale * mat[1].y;
      mat[2].z -= scale * mat[1].z;
  }



//////////////////////////////////////
  if(mat[0].z != 0){
    scale = (mat[0].z/mat[2].z);
```

```
      imat[0].x -= scale * imat[2].x;
      imat[0].y -= scale * imat[2].y;
      imat[0].z -= scale * imat[2].z;

      mat[0].x -= scale * mat[2].x;
      mat[0].y -= scale * mat[2].y;
      mat[0].z -= scale * mat[2].z;
}

if(mat[1].z != 0){
   scale = (mat[1].z/mat[2].z);

      imat[1].x -= scale * imat[2].x;
      imat[1].y -= scale * imat[2].y;
      imat[1].z -= scale * imat[2].z;

      mat[1].x -= scale * mat[2].x;
      mat[1].y -= scale * mat[2].y;
      mat[1].z -= scale * mat[2].z;
}


//////////////////////////////////////

sub_scale[0] = mat[0].x;
sub_scale[1] = mat[1].y;
sub_scale[2] = mat[2].z;

for(i =0; i<3; i++){
   if(sub_scale[i] != 0){
      imat[i].x = (1/sub_scale[i])*imat[i].x;
```

```
        imat[i].y = (1/sub_scale[i])*imat[i].y;
        imat[i].z = (1/sub_scale[i])*imat[i].z;
      }
   }

   mat[0].x = (1/sub_scale[0])*mat[0].x;
   mat[1].y = (1/sub_scale[1])*mat[1].y;
   mat[2].z = (1/sub_scale[2])*mat[2].z;



   for(i=0; i<3; i++){
      (*mat_res)[i].x = imat[i].x;
      (*mat_res)[i].y = imat[i].y;
      (*mat_res)[i].z = imat[i].z;
   }

}

void simultaneous_eq(vector mat[],vector single_mat[], vector (*res_single_mat)[]){

   int i,j,k;
   vector res[3]={0};
   vector single_res[3] = {0};

   gauss_inverse_mat(mat, &res); //  A^-1

   mmul(res, single_mat, &single_res); // A^-1 * [result...] = [x;y;z]

   for(i=0; i<3; i++){
      (*res_single_mat)[i].x = single_res[i].x;
```

```
    (*res_single_mat)[i].y = single_res[i].y;
    (*res_single_mat)[i].z = single_res[i].z;
  }
}



int main (void){
  int i;
  vector mat1[3]={0};
  vector mat2[3]={0};
  vector mat_res[3]={0};
  vector single_mat[3]={0};
  vector res_single_mat[3]={0};


  for(i=0; i<3; i++){
        printf("input first matrix %dst row\n", i+1);
        scanf("%lf", &(mat1[i].x));
        scanf("%lf", &(mat1[i].y));
        scanf("%lf", &(mat1[i].z));
  }


  for(i=0; i<3; i++){
        printf("input second matrix %dst row\n", i+1);
        scanf("%lf", &(mat2[i].x));
        scanf("%lf", &(mat2[i].y));
        scanf("%lf", &(mat2[i].z));
  }

  printf("input single column matrix \n");
```

```
scanf("%lf", &(single_mat[0].x));
scanf("%lf", &(single_mat[1].x));
scanf("%lf", &(single_mat[2].x));



//printf("%lf\n",det(mat)); //det ok

//madd(mat1, mat2, &mat_res);
//print_mat(mat_res);

//msub(mat1, mat2, &mat_res);
//print_mat(mat_res);

//mmul(mat1, mat2, &mat_res);
//print_mat(mat_res);

//mtranspose(mat1, &mat_res);
//print_mat(mat_res);

//cramer(mat1, single_mat, &res_single_mat);
//print_mat(res_single_mat);

//inverse_mat(mat1, &mat_res);
//print_mat(mat_res);

//gauss_inverse_mat(mat1, &mat_res);
//print_mat(mat_res);

//simultaneous_eq(mat1, single_mat, &res_single_mat);
//print_mat(res_single_mat);
```

```
   return 0;
}
```