# 图像旋转方案

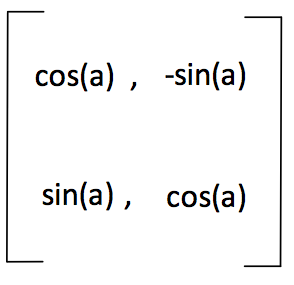
## 一，图像旋转原理

图像旋转的本质利用的是向量的旋转，而在MATLAB等算法工具中向量的计算往往转换成相应矩阵的计算，向量是几何中的概念，因此在算法的编译中常常不直接进行向量的运算，而是将其转换成在极坐标中的对应坐标矩阵来进行算法的构建。

矩阵乘法的实质是进行线性变换，因此对一个向量进行旋转操作也可以，通过矩阵和向量所对应的特征矩阵相乘的方式进行，而这在大多数的计算机语言中是通用的方法。正是因为这一点，在图像旋转的这个模块中，采用了构建特征矩阵进行坐标转化这个思路。

具体思路如下：

假设有二维向量v = [x ; y]，其中x,y是原图的像素点的横轴和纵轴坐标。若要进行逆时针旋转角度a。则旋转矩阵R为：



旋转后的向量 Ro = R \* v。 Ro =[Xo,Yo]; 其中Xo, Yo 是输出图像的坐标值。

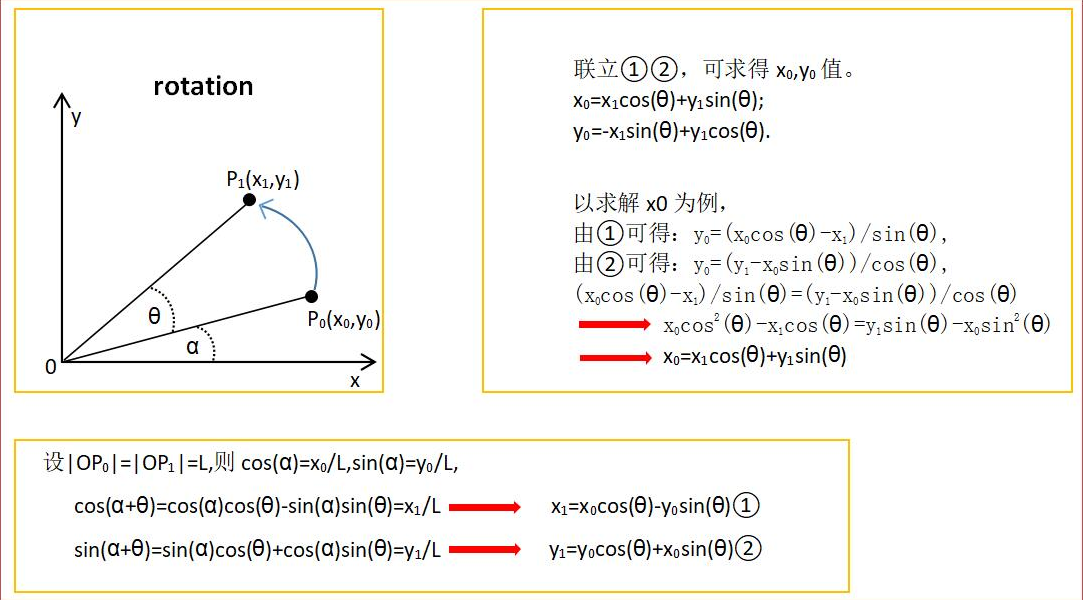
在正式处理过程中可以这么表示，原像素位置记为p，中心点记为c，旋转后像素位置记为pp。则有(pp - c) = R\*(p - c)，即:

pp = R\*(p-c) + c

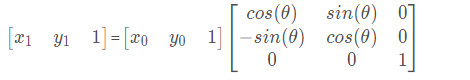
## 输入输出图像坐标的方案选择

方案一：

在此方案中，实现代码的方式是正向的思路，将原图中的像素点的坐标进行坐标的旋转，然后直接幅值到输出的图像中，此方案旨在找到输入坐标与输出坐标之间的代数对应关系，以此来进行Verilog代码的编写。

但在实际的分析的过程中，先采用极坐标系进行分析，得到了对应的坐标对应关系，如下图所示：

在该方法中，首先将原始坐标以及目标坐标放入了极坐标中，并且通过在极坐标中的关系，找到了同时满足X0,Y0,X1,Y1四个参量的方程组，以此来解出对应的坐标关系，并以此为基础得到了输入与输出之间的矩阵运算关系如下：



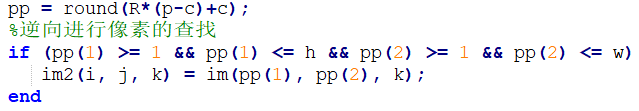
这样就解决了坐标的代数关系。但在实际的测试中发现，这种方法所旋转得到的图像有着较为严重的失真现象，具体情况如下图所示：

很明显可以看到，在旋转之后这两张图片出现了较大的差别，首先是原图像被裁减了，其次是目标图像中有较多的瑕点(杂点)。究其原因在于，从原图旋转后得到的目标图像的像素位置在原图中找不到。另外就是边缘被裁剪的问题，由于在这个方案中约束了显示区域，因此在旋转的过程中，部分像素点就会由于超出边界而被裁剪。针对以上的两个问题，进行了如下改进。

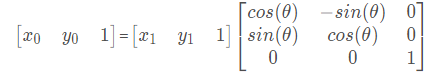
方案二：

由于在之前的方案中出现了杂点以及图像边缘裁剪的问题，因此在本方案中，我们采用了逆向思维，用目标图像的坐标去与原图的坐标进行坐标匹配，若在原图像中能找到匹配的图像，就显示该点旋转后的点坐标，若在原图中找不到该点，则不显示该点，通过这样就解决了杂点的问题，具体所限定的原图的查找区域代码如图所示：



其中，pp为旋转在后的坐标对应矩阵，在if语句中限定了原图的区域，用此区域则可以到原图中的坐标点，以此来排除不在区域中的坐标点，这样就可以解决杂点的问题。(以上是前期在MATLAB中的仿真代码截取)

在这种方案下，坐标的对应关系如下：



这样，该旋转后的图像就有了较好的还原度，达到了相应的题目要求，具体的方案的效果如下图所示：

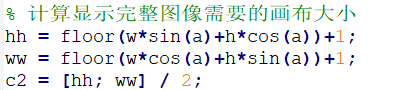


如图所示，相对方案一而言，图像的效果就好了很多，但图像边缘仍然存在边缘被切割的现象，因此在第三种方案中，对代码进一步进行优化。

方案三：

考虑到未对旋转后的图像进行显示区域的划分，因此此类旋转只是对单一像素点的旋转，然后在原图像的显示区域上进行坐标点的重新组合，得到显示的图像。在解决的方法的思路上，采用目标显示区域的重新划分来解决该问题。

具体思路是，采用原图像的长宽作为基准，再用坐标转换的关系，将长和宽转换到旋转后的坐标系中，得到目标图像在旋转后坐标系中的显示区域，具体如下：



这样，就解决了图像边缘被裁剪的问题，是整个图像得以完整的显示，实际的效果如下：



从图示的效果可以看出，边缘区域被裁剪的问题被解决了，但问题是图片加阴影的区域面积比原图大很多。

综合以上三种方案，结合实际需求，由于我们的显示是在一块固定大小的屏幕上进行显示，整个图像的显示范围有限，因此，最终选择方案二进行图像的旋转坐标对应。

## 旋转参数的提取和表示

在该设计中，要求图像拥有0到360的任意角度的旋转，坐标变换需要角度的正弦和余弦值。

1. 该正弦，余弦通过MATLAB计算得到，并预先储存到FPGA的片上储存空间中，在进行坐标变换时，读取对应角度的正弦，余弦值，进行坐标变换。由于计算得到的正弦和余弦值为浮点数，而FPGA擅长于进行整数运算。故要进行浮点数到整数的转换，具体的实现方法是，将计算得到的浮点正弦，余弦值乘上 256 后再取整，计算得到的结果于原结果相比被扩大了256倍，而在数字电路中，除法操作可以用移位来进行。结果右移8位即等效于除于256 。
2. 坐标变换需要0到360度的正弦，余弦值。一般的方案时直接储存 0到360的正弦，余弦值，需要时直接读取即可，但这种方法过于浪费储存资源。根据数学上的原理，正弦，余弦函数是周期变换的，在一个周期内有多个重复的函数。只需要储存四分之一周期的正弦函数值，再根据角度关系，确定其他四分之三周期角度的正弦，余弦值以及符号值。
3. 正负数的表示方法。在数字电路中，无符号数计算较为简单，电路设计也比较简单。但在坐标变换中，有符号位负的参数。在工程设计中，采用储存各个参数的绝对值，并另外设置一个符号标志位来标志参数的正负。用1表示参数是正数，用0表示参数是负数。在坐标变换计算中，根据标识位的值（0,1）来确定进行进行加法计算还是减法计算。

4，将坐标变换计算模块封装为一个子模块，输入输出图像的坐标和旋转角度后，即可计算出对应的输入图像对应的像素的坐标。然后读取该坐标的像素值，写入到旋转重建的图像对应的坐标位置即可。