

**USING MATLAB FOR ALGORITHM DEVELOPMENT:
A COORDINATE MAPPING FOR A RAPID PROTOTYPING SYSTEM**

Orlando J. Hernandez, Sr.

Wilfrido A. Moreno

Center for Microelectronics Research

ABSTRACT

In this paper we present how MATLAB was used to develop an algorithm to compensate for rotational misalignment between a Very Large Scale Integration (VLSI) chip and the table where it mounts, in a rapid prototyping system. After a basic mathematical solution to the problem was formulated, MATLAB was used to enhance and test this solution; such that a robust algorithm emerged. After the algorithm was validated, it was implemented in the controlling software of the system. This resulted in the addition of a real-time misalignment compensation capability for the rapid prototyping system.

University of South Florida
Tampa, Florida 33620

I. INTRODUCTION

One of the fundamental components of a rapid prototyping system is a high precision linear induction motor x-y table with laser interferometer position feedback (position absolute accuracy is better than 0.5 μm from the commanded position over the entire range of travel)[1]. This table is used to position VLSI circuits under a computer controlled laser system; such that restructuring, and hence rapid prototyping, can be achieved. Restructuring takes place by using the pulsed laser beam to create conductive links and cut conductors, thus interconnecting electronic circuitry as required by the designer[2].

During the laser operation, it is critical that the x-y coordinate system of the chip (the circuitry) coincides with the table's coordinate system. This implies that there should be no rotation of the chip coordinate system with respect to the table's; because if there is no rotation between the chip and the table, their coordinate systems can agree by setting the table's origin at the chip's origin. Although a manual rotation compensation stage is in place; achieving the required alignment manually is nearly impossible, and the time required grows exponentially as the angle between the coordinate systems becomes smaller. The tolerated rotation angle of the chip with respect to the table is well below

0.1°. Since dimensions of less than 1 μm are involved in this process, even a very small rotation angle will result in a considerable error between the commanded position to the table and the position on the chip over a typical range of travel between 0.25 and 4.00 inch.

II. BASIC SOLUTION

In its simplest form, the problem can be stated as follows: "There is an inherent error, introduced by a misalignment between the chip and the table, such that the commanded position to the table does not agree with the position on the chip." It is required that when the table is commanded to move to position (x_1, y_1) of the chip's design coordinate system, the laser beam hits this position on the chip. Instead, the laser beam will hit some other $(x_1 + \epsilon_x, y_1 + \epsilon_y)$ position. This problem can be identified as one of error correction or cancelling, and solved as such.

In matrix format, let \mathbf{P} denote the commanded position $[x \ y \ 1]^T$ (e.g. (x_1, y_1)) and \mathbf{P}' denote the actual position $[x' \ y' \ 1]^T$ (e.g. $(x_1 + \epsilon_x, y_1 + \epsilon_y)$); where everything is being referenced to the chip's coordinate system. The error introduced by the misalignment between the chip and the table can be described as a matrix \mathbf{T} such that $\mathbf{P}' = \mathbf{T} \mathbf{P}$; where the matrix \mathbf{T} has the form

$$\mathbf{T} = \begin{bmatrix} A_2 & A_1 & A_0 \\ B_2 & B_1 & B_0 \\ 0 & 0 & 1 \end{bmatrix}$$

This \mathbf{T} matrix is the well known two dimensional composite translation and rotation matrix of graphics theory[3].

The solution to the problem consists of finding a matrix $\mathbf{t} = \mathbf{T}^{-1}$, and then pre-multiplying the desired

position \mathbf{P} by \mathbf{t} before commanding it to the table. Then

$$\mathbf{P}' = \mathbf{T} (\mathbf{t} \mathbf{P})$$

Since $\mathbf{t} = \mathbf{T}^{-1}$;

$$\mathbf{P}' = \mathbf{T} (\mathbf{T}^{-1} \mathbf{P})$$

$$\mathbf{P}' = \mathbf{T} \mathbf{T}^{-1} \mathbf{P}$$

$$\mathbf{P}' = \mathbf{P}$$

Thus, we have had cancelled the effects of the error matrix \mathbf{T} .

Finding \mathbf{t} ,

$$\mathbf{t} = \begin{bmatrix} a_2 & a_1 & a_0 \\ b_2 & b_1 & b_0 \\ 0 & 0 & 1 \end{bmatrix}$$

and hence \mathbf{T} , is equivalent to characterizing the misalignment present between the chip and the table. Thus, it only needs to be done once; as long as the chip is not replaced in the table.

III. ALGORITHM DEVELOPMENT

In general,

$$\begin{bmatrix} x'_1 & x'_2 & x'_3 & x'_N \\ y'_1 & y'_2 & y'_3 & y'_N \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 - x'_1 & x_2 - x'_2 & x_3 - x'_3 & x_N - x'_N \\ y_1 - y'_1 & y_2 - y'_2 & y_3 - y'_3 & y_N - y'_N \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(1)

where N is the number of points used initially to characterize the misalignment between the chip and the table. Taking the transpose of both sides of equation (1);

$$\begin{bmatrix} x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 \\ x'_3 & y'_3 & 1 \\ - & - & - \\ x'_{N-1} & y'_{N-1} & 1 \\ x'_N & y'_N & 1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ - & - & - \\ x_{N-1} & y_{N-1} & 1 \\ x_N & y_N & 1 \end{bmatrix} \mathbf{T}^T \quad (2)$$

Then, it is clear that

$$\mathbf{T} = \left(\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ - & - & - \\ x_{N-1} & y_{N-1} & 1 \\ x_N & y_N & 1 \end{bmatrix}^{\#} \begin{bmatrix} x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 \\ x'_3 & y'_3 & 1 \\ - & - & - \\ x'_{N-1} & y'_{N-1} & 1 \\ x'_N & y'_N & 1 \end{bmatrix} \right)^T \quad (3)$$

In equation (3), # denotes the inverse (-1) if $N = 3$, and the **generalized inverse (Moore-Penrose inverse)** if $N > 3$. If $N > 3$, the rank of the matrix being inverted is the number of columns N ; and for any column ranked matrix A , $A^{\#} = (A^T A)^{-1} A^T$. It should be noted that N should be at least 3, so that equation (2) is not an under-determined system. Once \mathbf{T} is known, \mathbf{t} can be found easily;

$$\mathbf{t} = \mathbf{T}^{-1} \quad (4)$$

The last step before formulating the algorithm, is to find the minimum N that will provide a good error correction. It is obvious that the larger N is (i.e. the more points are used to characterize the misalignment \mathbf{T}) the better \mathbf{T} and \mathbf{t} are going to be in

the least squared error sense. Notice that equation (3) is a formulation of a least squares problem, except that the right hand side is transposed. MATLAB was used, and it was fundamental, to experiment with different methods and N 's; thus allowing us to find the best approach to calculating \mathbf{t} . A copy of a MATLAB program used is provided in **APPENDIX A**. Experimenting with these types of programs, showed us that the misalignment was non-linear and not purely rotational, and that the best way to implement the algorithm was to use two \mathbf{t} 's, each one found with $N=3$. Since the misalignment is non-linear, and it is not purely rotational in nature (i.e. the components of \mathbf{T} and \mathbf{t} include rotation and translation terms, even though the origins of the coordinate systems for the chip and the table coincide), it was found that the best compensation was obtained by dividing the x-y coordinate system of the chip into two triangular regions; $x > y$ and $x < y$. A different \mathbf{t} was used to pre-multiply \mathbf{P} depending on which region it laid. The points used to calculate the \mathbf{t} 's, were four significant points near each corner of the chip (Figure 1). For the \mathbf{t} corresponding to $x > y$, the lower-left, lower-right, and upper-right points were used; for the \mathbf{t} corresponding to $x < y$, the lower-left, upper-right, and upper-left points were used. This is counting on that the (0, 0) point has been set somewhere near the lower-left corner, but it does not have to coincide with the lower-left point used in the calculation of the \mathbf{t} 's.

A formulation of the algorithm in C-like pseudo-code follows:

```
/* At beginning of restructuring
session */
set table's (0, 0) point to coincide
with chip's (0, 0) point;
move to lower-left point;
move to lower-left mark;
```

```

record position from table  $P_T$ ;
Refer  $P_T$  to the chip's coordinate
system;
move to upper-right point;
move to upper-right mark;
record position from table  $P_T$ ;
Refer  $P_T$  to the chip's coordinate
system;
move to lower-right point;
move to lower-right mark;
record position from table  $P_T$ ;
Refer  $P_T$  to the chip's coordinate
system;
move to upper-left point;
move to upper-left mark;
record position from table  $P_T$ ;
Refer  $P_T$  to the chip's coordinate
system;
calculate  $t$ 's according to equations
(3) and (4);
/* This portion is done for the rest
of the restructuring session in real
time each time the table moves */
while(in session){
    if(x>y){
        pre-multiply  $P$  by the  $t$ 
        corresponding to the
        lower triangular region;
    }
    else{
        pre-multiply  $P$  by the  $t$ 
        corresponding to the
        upper triangular region;
    }
    move table;
}

```

To refer the points in the coordinate system of the table to the chip's, the following operation was performed:

```

Point_in_Chip = Commanded_Point +
                (Commanded_Point -
                 Point_Recorded_from_the_Table)
so;
Point_in_Chip = 2
Commanded_Point
                - Point_Recorded_from_the_Table

```

The complete algorithm was implemented in the C code of the rapid prototyping system controlling software.

IV. CONCLUSIONS

We have described an algorithm to compensate for misalignment between chip and table in a rapid prototyping system, and how MATLAB was used to develop this algorithm. Implementation of this algorithm in the controlling software, provided the system with a real-time position error correction capability.

One of the important features of our algorithm is that it is independent of the device being prototyped. Also, it is expandable to provide compensation, not only in the x and y directions, but also in the z direction. Since the system has a z stepper motor, it is possible to implement a compensation for deviations in the surface coplanarity of the VLSI chip. This feature is planned to be added in the future to the system controlling software.

APPENDIX A

An experimental MATLAB program used to identify the best method and the minimum N that yielded acceptable results:

```

disp('Procedure to correct for
mismatches between design
coordinates and');
disp('real coordinates due to
horizontal translation and
rotation. ');
disp('(5 ways) ');
disp(' ');
disp('Want to enter physical
coordinates as rows? (0 for NO, 1
for YES) ');
n=input(' ');
disp(' ');
if n
    xd=[29.2; 2828.2; 2883.2; 2832.2;
        73.2; 34.7];
    yd=[101.2; 74.2; 1460.2; 2824.2;
        2824.2; 1496.7];
    XYd=[xd yd ones(6,1)];
%

```

```

bl=input('Bottom left ');
br=input('Bottom right ');
op=input('Point at one ');
tr=input('Top right ');
tl=input('Top left ');
pp=input('Point at pad ');
%
xyp=[bl 1; br 1; op 1; tr 1; tl 1;
pp 1];
XYp=2*XYd-xyp;
%
%METHOD 1
XYd1L=XYd;
XYd1L(3:3, : )=[];
XYd1L(4:5, : )=[];
XYd1U=XYd;
XYd1U(2:3, : )=[];
XYd1U(4:4, : )=[];
XYp1L=XYp;
XYp1L(3:3, : )=[];
XYp1L(4:5, : )=[];
XYp1U=XYp;
XYp1U(2:3, : )=[];
XYp1U(4:4, : )=[];
AB1L=inv(XYd1L)*XYp1L;
AB1U=inv(XYd1U)*XYp1U;
%
%METHOD 2
XYd2=XYd;
XYp2=XYp;
AB2=pinv(XYd2)*XYp2;
%
%METHOD 3
XYd3L=XYd;
XYd3L(5:6, : )=[];
XYd3U=XYd;
XYd3U(2:3, : )=[];
XYp3L=XYp;
XYp3L(5:6, : )=[];
XYp3U=XYp;
XYp3U(2:3, : )=[];
AB3L=pinv(XYd3L)*XYp3L;
AB3U=pinv(XYd3U)*XYp3U;
%
%METHOD 4
XYd4=[[xd .^ 2] [xd .* yd] [yd .^
2] [xd] [yd] ones(6,1)];
XYp4=XYd4;
XYp4( : ,4:6)=XYp;
AB4=inv(XYd4)*XYp4;
%
%METHOD 5
XYd5=[[xd .^ 2] [xd .* yd] [yd .^
2] [xd] [yd] ones(6,1)];

```

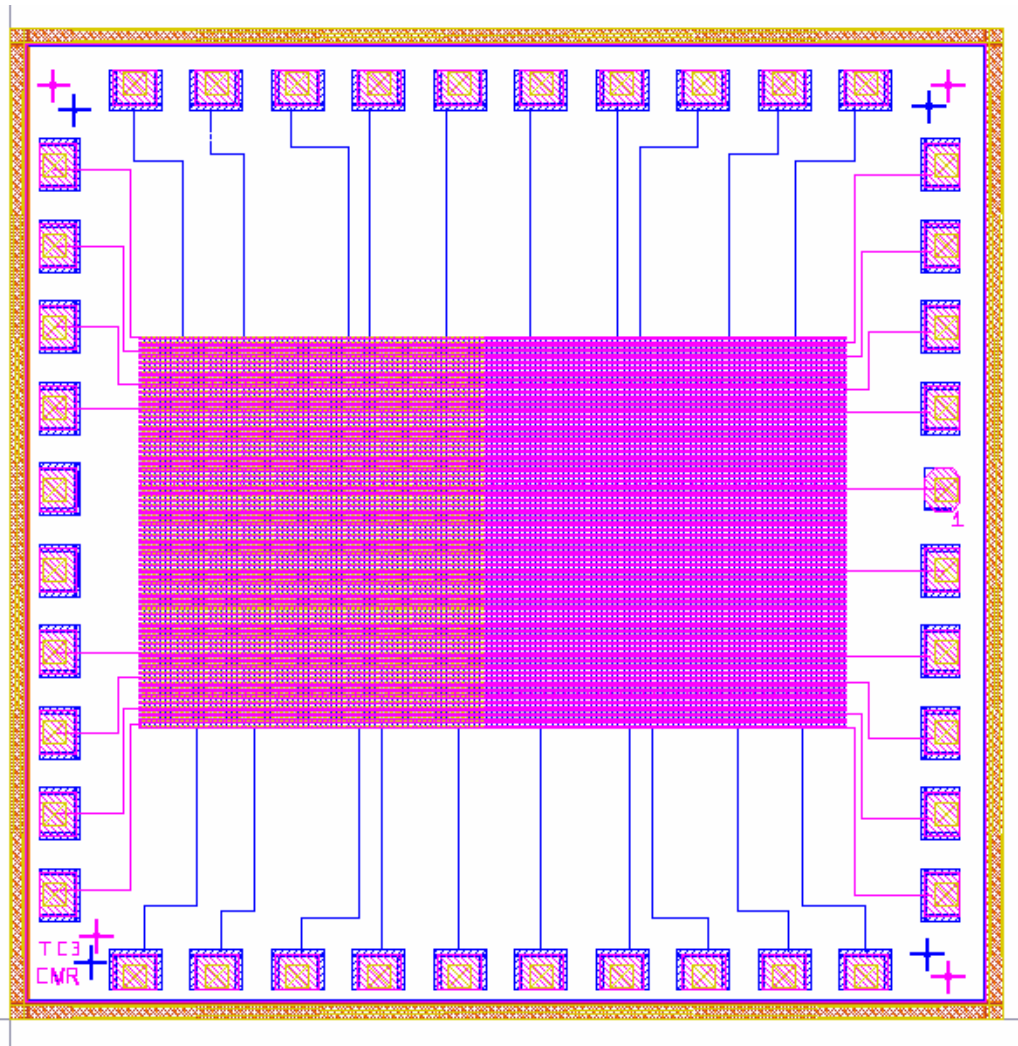
```

Xp=XYp( : ,1:1);
Yp=XYp( : ,2:2);
XYp5=[[Xp .^ 2] [Xp .* Yp] [Yp .^
2] [Xp] [Yp] ones(6,1)];
AB5=inv(XYd5)*XYp5;
end
xyc=input('Check point as column ');
XYc=[xyc; 1];
%
%RESULTS
if XYc(1,1)>XYc(2,1)
XY1=inv(AB1L)*XYc;
else
XY1=inv(AB1U)*XYc;
end
XY2=inv(AB2)*XYc;
if XYc(1,1)>XYc(2,1)
XY3=inv(AB3L)*XYc;
else
XY3=inv(AB3U)*XYc;
end
xc=xyc(1,1);
yc=xyc(2,1);
XY4=inv(AB4)*[xc^2; xc*yc; yc^2;
xc; yc; 1];
XY5=inv(AB5)*[xc^2; xc*yc; yc^2;
xc; yc; 1];
%
%DISPLAY
XY=[XY1 XY2 XY3 XY4(4:6, : )
XY5(4:6, : )];
format bank;
disp(' ');
XY
disp(' ');
d;

```

REFERENCES

- [1]R. Lee, W. Moreno, O. Hernandez, E. Harrold, and D. Whittaker, "Rapid Prototyping Using Laser Restructurable VLSI Circuits," Proceedings of the 4th International Workshop on Rapid System Prototyping, June 1993.
- [2]O. J. Hernandez, "A Gate Array Chip Using Laser Restructurable VLSI," In preparation.
- [3]R. C. Gonzalez, and R. E. Woods,



Digital Image Processing
(book). Addison-Wesley, 1992

Figure 1 An integrated circuit (IC). The cross shaped marks can be seen at the corners.

