Reconfigurable computing for tool path computation

Antonio Jimeno[†], Sergio Cuenca[†]

† Computer Science Technology and Computation Department, University of Alicante
Carretera San Vicente del Raspeig s/n,
03690 San Vicente del Raspeig, Alicante, Spain.

Abstract. Tool path generation is one of the most complex problems in Computer Aided Manufacturing. Although some efficient strategies have been developed to solve it, most of them are only useful for 3 and 5 axis standard machining. The algorithm called *Virtual Digitising* computes the tool path by means of a "virtually digitised" model of the surface and a geometry specification of the tool and its motion, so can be used even in non-standard machining (retrofitting). This algorithm is simple, robust and avoids the problem of tool-surface collision by its own definition. However, its computing cost is high. Presented in the paper there is a Virtual Digitising optimisation that takes the advantages of reconfigurable computing (using Field Programmable Gates Arrays) in order to improve the algorithm speed. A comparative study will show the gain and precision achieved.

1. Introduction

In order to get a predefined surface by means of the cutting wheels of a machine, it is necessary to supply a series of 3D or 2D coordinates that define its motion. These points are usually referred to tool centre positions.

In this way the problem can be expressed as *obtaining a trajectory of tool centres that defines the surface to be machined with a given precision*. Figure 1 shows the trajectory (tool path) of a circle centre point in order to define a rectangle. In this case, for simplicity, the problem is presented in 2D. For 3D surfaces the problem becomes more complex. This problem can be related to the dilation process from the mathematical morphology where the object to mechanise is the shape to dilate and the tool is the structuring element, however, 3D versions of morphological operations are not efficient and techniques are still in development.

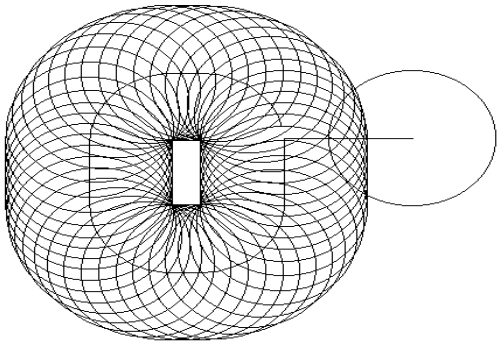

Fig.1. Circle trajectory in order to get a rectangle

The basic virtual digitising strategy

Centre tool points are obtained by virtually touching the object to mechanise. This algorithm[1, 2], typically used to compute pencil curve tracing[3], internally works as mechanical copiers do: the copying arm touches the surface and a group of arms transmitted the movement to the cutting wheels which perform the same movement and finished the copied model.

Due to the fact that all the machining processes are simulated, this algorithm has no restrictions in tool or machine specifications, so the algorithm can be used even in non-standard machining (e.g. in retro-fitting machining).

The digitalization algorithm becomes simple once the surface and tool motion are well defined. Basically, the behavior can be described as follows: For each point of the trajectory the part surface is transformed in order to face the cutting tool. Then the minimum distance from every surface point to the tool is computed in the direction of tool attack axis. This distance determines the tool center point for the current step in the virtual digitalization process. Physically, we select the point that touches the tool surface in first place when the tool is moved along the attack axis. The process is similar to that of is used for obtaining z-maps of the tool envelope surface, typically used for 3-axis CNC machining: the *inverse offsetting* method[4] and the *direct cutting* simulation[5, 6].

The basic pseudo-code algorithm can be expressed as follows:

```
For every trajectory position trpos do
 Min_distance=∞
 For u in Surface_Rows do
   For v in Surface_Columns do
    p'(u,v) = p(u,v) * TR4x4(trpos)
    Current_distance=D(p',u,v)
    If Current_distance<Min_distance
    then Min_distance=Current_distance
    Endif
   Endfor
 Endfor
 Tool_centre=Get_centre_point(MinDistance,trpos,TR4x4)
 Add_trajectory(Tool_centre)
EndFor
```

Algorithm 1. Basic virtual digitising algorithm

Analyzing algorithm, it is possible to observe up to three nested loops. One of them, the most internal one, is used to access to every surface point in the selected surface, that is, it consists into two loops, one for rows and the other for columns in fact. The most external loop goes through every trajectory position. In order to obtain a good finishing quality, it is necessary produce, at least, as many trajectory points as points the surface has.

Let assume *n* as the maximum number of surface points, and *m* as the number of trajectory positions, then the cost of the algorithm, is:

$$O(n.m) \quad (1)$$

Values for *n* and *m* depend on the model size and the precision desired for the machining. Note that *n* value consists of a grid of *Surface_Rows x Surface_Columns* in size for the Algortimh 1. As a guide, typical numbers for both *n* and *m* in shoe last machining are about twenty thousand; in this case, computing time is longer than the machining one. We will show some quantitative examples with time measures in section 3.

2. The reconfigurable approach

The high computational cost of the algorithm does not allow implanting efficient tool paths generators in most of computers. For this reason, we are encouraged to develop efficient strategies in order to reduce the tool path computing time. If we have a look to the Basic Virtual Digitising algorithm, we notice that the most of the complexity resides inside of the third loop. Accelerating the functions called in this part, we will be able to reduce the total computation time significantly.

Most of computer systems are based on simple instruction set microprocessors. Instructions have been chosen and optimized according to their frequency of occurrence in programs. For high-intensity computations, an additional improvement can be if we use special-purpose architectures.

One way of dealing with special-purpose algorithms as Virtual Digitizing is to use application-specific integrated circuits (ASIC), this kind of circuits is unable to execute any other task but the one it was constructed for, however they offer the best speed rate. One of the least expensive technologies for implementing these circuits is the field-programmable gate array (FPGA). Using this technology, a single device can be reprogrammed (reconfigured) to perform different tasks at hardware speed. An algorithm or a part of it is represented in a hardware-description language, compiled into a netlist, and then transferred to an FPGA chip. Hence, a highly specialized device is constructed. FPGA technology is being successfully used in genetic algorithms, neural networks, or fuzzy neural networks (a complete state-of-the-art in FPGAs is presented in [7]).

There are three different operations inside the third loop of the Virtual Digitizing algorithm that are excellent candidates to be implemented in a FPGA:

Point transformation: $p'(u,v) = p(u,v) * TR_{4x4}$

A 3D transformation is applied on every surface point, so the tool faces the surface. This operation is made by means of a 4x4 transformation matrix. From a generic stand point, the process consists of a row x matrix post multiplying.

Distance computing: *D(p',u,v)*

This function computes the distance between a surface point and the tool in the tool attack direction. Depending on the complexity of the tool geometry - sphere, torus, cone, and so on - the function becomes more complex.

Comparison and assignment: *If Cur_dist<Min_dist then Min_dist = Cur_dist*

Finally, the most internal loop makes a comparison and an assignment if the computed distance is lower than the current minimum distance.

These three different operations are carried out on every point of the tool trajectory and for every surface point. Any optimization made at this level will improve the total computation time significantly. As expressed above, distance computing implementation varies on tool geometry and point transformation depends on tool path strategy. So if we create hardware circuits (as ASICs) in order to speed up the algorithm for each function, we will need as many circuits as different strategies or tools we are going to use, that is, an expensive and complex architecture. A smart solution would be the use of reconfigurable circuits. Figure 2 shows a static reconfigurable architecture used to perform tool trajectories using virtual digitizing. Different machines and tool path strategies will imply different functions in FPGA 1 and 2. So CPU will choose the proper task involved at a time and prepare FPGA for the computation by reprogramming the static configuration memory. Surface data resides in main memory and can be accessed both CPU and reconfigurable circuits.

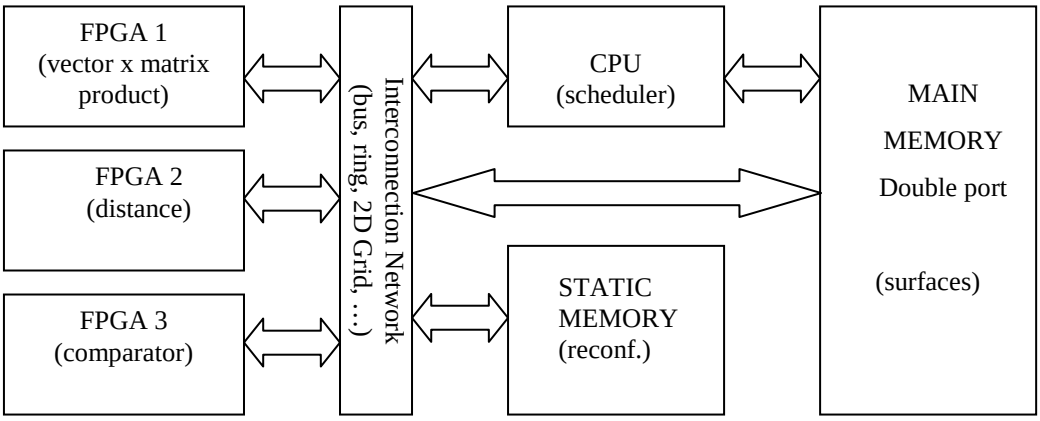

Fig. 2. General Scheme. Reconfigurable architecture for Virtual Digitizing algorithm.

3. Optimising tool path for a Turning Lathe Machine

We are developing a reconfigurable system that is able to perform tool path computation using the Virtual Digitising algorithm efficiently.

The surface to be machined consists of a discrete model of a free-form NURB surface. The more grid points we use to represent that surface, the more accurate trajectory we get. For an object of 10 x 10 x 200 mm^3 approximately we use a grid of 130 x 120 points what implies a distance of 2 mm between points in each dimension.

The machine selected is a traditional turning lathe, consisting of three different axes, as shown in figure 3. All of them perform a spiral movement around the object to be mechanised. The tool selected is a 3D torus, which simulates the double cutting wheel in movement.

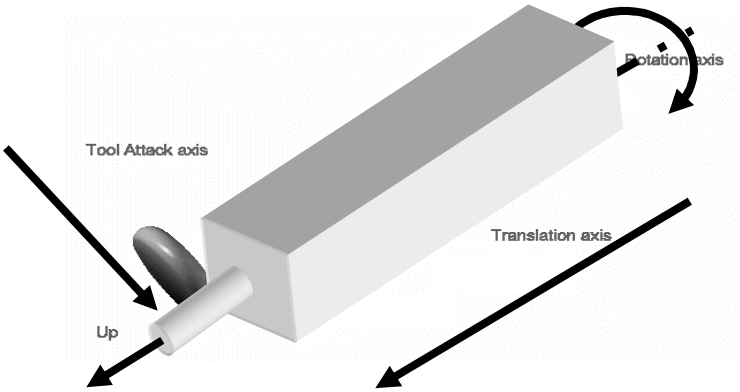

Fig. 3. Tool motion definition for a turning lathe machine

The Algorithm 1 is slightly transformed into:

```
For every_tool_translation_position_x do
 Min_distance=∞
 For u in Surface_Rows do
  For v in Surface_Columns do
    For degrees = 0, 360 step inc_angle do
      TR4x4 = Rotation_Matrix_X(degrees)
      p'(u,v) = p(u,v) * TR4x4
      Current_distance=D(p'u,v)
      If Current_distance<Min_distance
      then Min_distance=Current_distance
      Endif
    Endfor
  Endfor
 Endfor
 Tool_centre=Get_centre_point(MinDistance,trpos,TR4x4)
 Add_trajectory(Tool_centre)
EndFor
```

Algorithm 2. Changes introduced in algorithm 1

For simplicity, we assume the X axis as the rotation axis (always is possible to find a 3D-transformation matrix that translates any rotation vector *v* to the *X*-axis).

Algorithm 2 consists of four nested loops, first and last ones simulate the tool movement around the object. Second and third loops analyse every surface point in order to find the nearest one to the tool. The transformation matrix consists of a basic 3D rotation matrix around the X axis that rotates every point an increment angle up to complete a round. The distance function computes the distance between a 3D point and a 3D torus in the tool attack direction (Y axis) and can be expressed in equation (2).

$$D(x,y,z) = T_y - y - \sqrt{\left(R + \sqrt{r^2 - (x - T_x)^2}\right)^2 - z^2} \quad \textbf{(2)}$$

Where:

T_x, T_y : Are the x,y coordinates of the torus centre.
x,y,z : are the 3D point coordinates
R, r : are the major and minor torus radii

For this example, we must to select the best FPGA configuration for each task involved in the Virtual Digitising process. In the inner loop we distinguish three different tasks: *task 1* performs point rotation, *task 2* computes the distance function and *task 3* makes a conditional assignment. We have made software simulations in order to evaluate the cost of every task in the inner loop and the best time we can achieve by using the field programmable logic technology.

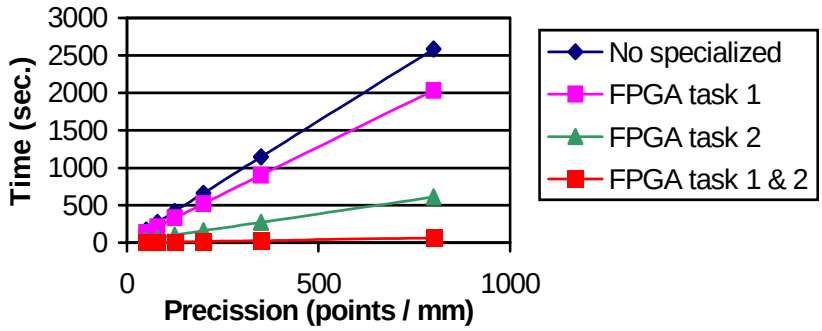

Fig. 4. Software simulation. *Task 1* performs point transformation, in the case of study, consists of a 3D rotation around the X axis. *Task 2* computes 3D distance between a 3D grid point and a torus (the tool).

Figure 4 shows the improvement achieved using FPGAs against the no specialized algorithm (without using FPGAs). These lines have to be taken as optimal, since no overhead has been considered in simulations for the use of FPGAs. Figure 5 represents the computing cost of each task as a percentage.

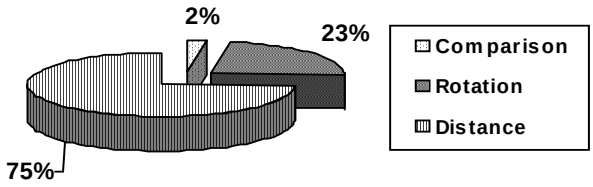

Fig. 5. Inner loop computing time

In next section, a reconfigurable architecture specialized in the rotation task is presented.

4. The Multi-Rotator architecture

In the virtual digitising algorithm rotations are always computed in the same way: every point is rotated a fixed angle around the rotation axis for the whole round (e.g. for an increment of 4 degrees the successive rotations will be: 0, 4, 8, 12 …, 352 and 356 degrees using a configuration of 90 points per round).
In field programmable logic, CORDIC algorithm[8] is used to perform generic spatial rotations. CORDIC provides an iterative method of performing vector rotations by arbitrary angles using only shifts and adds[9]. As opposite, our algorithm (called Multi-Rotator[10] or MR), computes the rotation directly, achieving the maximum precision allowed by the binary representation of the geometric points. MR implementation could be carried out by just using Constant Coefficient Multipliers (KCMs) and/or distributed arithmetic[11,12], achieving a response time equal or even smaller than the unrolled and bit parallel versions of CORDIC algorithms[13,14,15], using similar resources and keeping the maximum precision.

The approach consist of producing *r* rotations of the same point 3D *p* with coordinates *(x,y,z)*. Every rotation step will be constant *(Δθ)*. For the *i* rotation, the point *p* will be rotated as:

$$\theta_i = \theta_{i-1} + \Delta\theta \quad (3)$$

For this study, we assume every rotation is made over the z-axis. The coordinates of the *p* point after for *i-th* rotation are shown in (3). Note that *z* component will keep unchanged due to the fact that rotation is made over the z-axis):

$$x_i' = x\cos\theta_i - y\sin\theta_i = xC_i - yS_i$$
$$y_i' = y\cos\theta_i + x\sin\theta_i = yC_i + xS_i \quad (4)$$

Where $C_i=\cos\theta_i$, $S_i=\sin\theta_i$ .These equations could be expressed as a function of the previous iteration angle as:

$$x_i' = x\cos(\theta_{i-1}+\Delta\theta) - y\sin(\theta_{i-1}+\Delta\theta) \quad (5)$$

$$y_i' = y\cos(\theta_{i-1}+\Delta\theta) + x\sin(\theta_{i-1}+\Delta\theta)$$

Using trigonometric properties of the angle addition for the sine and cosine:

$$\begin{aligned} \sin(a+b) &= \sin a \cdot \cos b + \cos a \cdot \sin b \\ \cos(a+b) &= \cos a \cdot \cos b - \sin a \cdot \sin b \end{aligned} \quad (\mathbf{6})$$

we obtain:

$$\begin{aligned} x_i' &= xC_i - yS_i = [x\,C_{i-1}\,C_\Delta - x\,S_{i-1}\,S_\Delta] - [y\,S_{i-1}\,C_\Delta + y\,C_{i-1}\,S_\Delta] \\ y_i' &= yC_i + xS_i = [y\,C_{i-1}\,C_\Delta - y\,S_{i-1}\,S_\Delta] + [x\,S_{i-1}\,C_\Delta + x\,C_{i-1}\,S_\Delta] \end{aligned} \quad (\mathbf{7})$$

where $C_\Delta = \cos\Delta\theta$, $S_\Delta = \sin\Delta\theta$, $C_{i-1} = \cos\theta_{i-1}$, $S_{i-1} = \sin\theta_{i-1}$.

The four products: xC_i, yS_i, yC_i, xS_i , constitute a linear combination of the previous rotation xC_{i-1}, yS_{i-1}, yC_{i-1}, multiplied by the C_Δ and S_Δ constants. Because of these transformations, and once the first rotation is calculated, for the rest of calculations it is only necessary to do four products by constant coefficients (C_Δ y S_Δ). Even the first rotation is computed by means of constant products taking the initial angle equal to zero degrees.

$$\begin{aligned} x_1' &= x\,C_\Delta - y\,S_\Delta \\ y_1' &= y\,C_\Delta + x\,S_\Delta \end{aligned} \quad (\mathbf{8})$$

The KCMs are well suited to FPGA designs. They need only look up tables (LUTs), shifts and adders. However, the LUTs size grows exponentially with n (where n represents the binary precision selected for coordinate representation). This number should be kept low to achieve a good FPGA spatial occupation. In this way, we should compute the rotation using a 4 or 8 bit precision that is not a good value for most of applications. An alternative consists of using the Serial Distributed Arithmetic in order to calculate products and additions at the same time; in this case the algorithm architecture is modified as figure 6 shows.

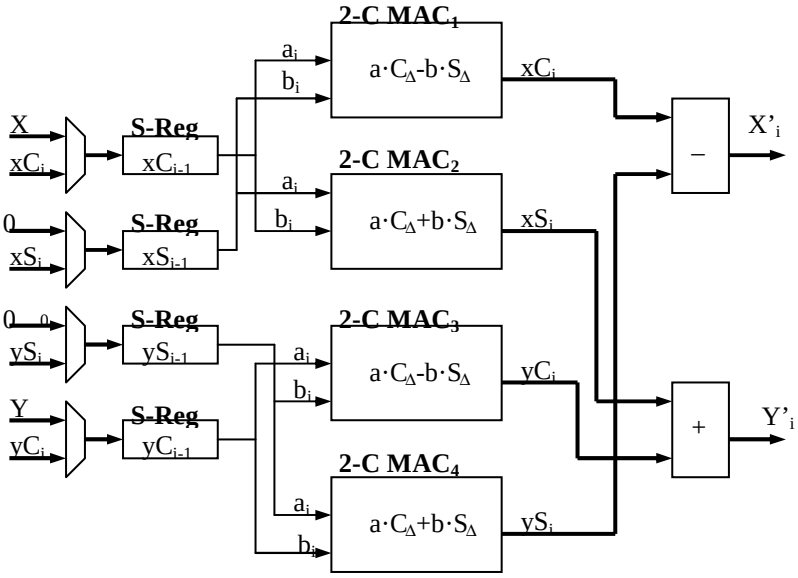

Fig.6.. MR architecture SDA based

Four *n*-bit shift registers are used to store products of *i-1* rotation. In the first one (θ_0=0), the input multiplexers update the shift registers (S-Regs) with the point coordinate values (*x*,*y*) or zero if the product includes the sine factor. The two coefficients MAC blocks (2-C MAC) compute the products $a \cdot C_\Delta - b \cdot S_\Delta$ and $a \cdot C_\Delta + b \cdot S_\Delta$ working on a serial mode. After *n* cycles the four components of the (7) addition are obtained (*n+3* bits). In the last cycle, S-Regs are updated and both *x* and *y* coordinates are computed at the same time. In order to reduce the resources, is possible to use just Adder/Substracter (Add/Sub) blocks in the final phase, by means of increment a cycle in the computing time. As a resume, the architecture will consist of 1 Add and 1 Sub for a *n+1* cycles computing time, or 1 Add/Sub block for a total time cost of *n+2* cycles.

Every 2-C MAC module consists of a LUT that computes the partial products, and a Scaling accumulator which computes partial sums (see figure 7). The LUT data, referenced in figure 7, is composed of all partial sums of the two coefficients (C_Δ and S_Δ). The least significant bit of the shift registers addresses the LUT. Because the address of the LUT contains all possible combinations of one or zero, based on the two inputs, the LUT outputs contains all four possible sums of the coefficients. The LUT is four *n+1*-bit words in size.

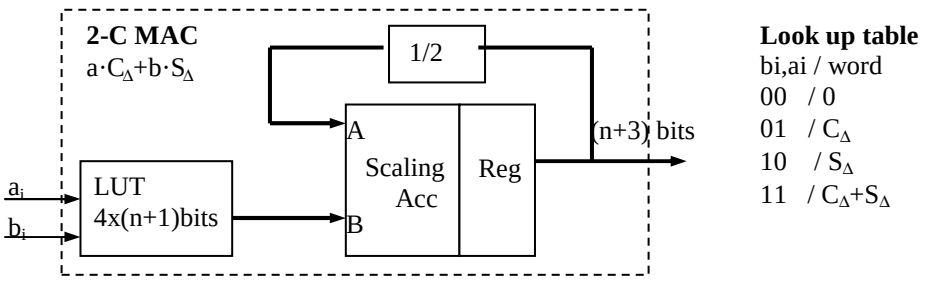

Fig. 7. Structure of 2-C MAC based on SDA

All the LUTs are similar for products made in 2-C MAC (1, 3) and (2, 4) respectively, for this reason only two double port memory LUTs will be needed. The whole architecture will consist of: 4 multiplexers, 4 shift registers of n-bits, 2 LUTs of *4x(n+1)* bits, 4 Scaling accumulators of (*n+2)*-bits, 1 adder of (*n+2)*-bits and 1 substracter of (*n+2)*-bits.

Architecture	Number of cycles	Resources	n/w relation
Bit-serial CORDIC	nxw	3 serial S-Reg 3 serial Add/Subs 1 serial ROM	n>w
Bit –parallel CORDIC iterative	w	3 Reg. 3 Barrel Shift. 3 Add/Subs	n>w
Bit –parallel fully segmented	w+1	(3 Add/Subs)•(n+1)	$n=w+2+\log_2(w)$
SDA MultiRotator	n+1	4 S-Reg 2 LUT, 4x(n+1) bits 4 Scaling Acc 1 Add + 1 Subs	-
2-bit PDA Multirotator	n/2+1	8 S-Reg 4 LUT, 4x(n+1) bits 8 Scaling Acc 1 Add + 1 Subs	-

Table 1. Algorithms Comparative

In order to increase the processing speed of the MR architecture, it is possible to use 2-C MACs based on Parallel distributed arithmetic (PDA). For instance, it is possible to process odd and even bits of each S-Reg (2-bit PDA) at the same time, reducing the number of cycles in a half. In that case, the number of LUTs is doubled and it is necessary to include an additional Scaling accumulator to each module.

Table 1 shows a performance and resource comparison between MR and usual CORDIC implementations. This kind of devices uses angle approximation in order to reach the desired angle. Every coordinate is represented by an n-bit number, CORDIC implementations use w approximation steps (called iterations) to obtain a valid result. In this table, the *number of cycles* column means the clock cycles necessaries to produce a single valid rotation (remember that virtual digitizing algorithm produces this kind of rotations multiple times for every surface point).

Bit serial CORDIC implementation offers the worst performance index so that it needs n cycles (n shifts) per iteration, although it allows the highest work frequencies for the FPGAs. In the bit-parallel implementation the n shifts are carried out in a cycle, using Barrel shifters, so that the number of cycles is reduced up to n. On the other hand, the cycle period is about five times bigger[9]. Both of CORDIC implementations achieve a maximum precision that depends on the number of iterations (w), that is, in any case, always lower than the precision reached with n-bit numbers. In MR implementations the number of cycles is equal or lower that the fastest CORDIC implementation and the simplicity of the resources needed does

feasible clock frequency rates similar to the bit-parallel. Moreover, the precision reached is the highest we could obtain using n-bits.

5. Experiments

The MR architecture has been simulated and analysed in terms of accuracy. The target platform we have used was the XC4000E FPGA family. The simulation tool was the Xilinx Foundation Series 2.1i software. Experiments simulate surface point rotations using the virtual digitizing algorithm. Every point is rotated many times using different angle steps up to complete a whole round (360 degrees). In this context, we call *iteration* to each single rotation (do not confound with the term *iteration* used in CORDIC literature as *approximation*).

Due to the fact of architecture scalability, we made experiments for both 16 and 32 bits in data word length. As showed in Table 1, the more bits are used for data representation, the more hardware resources are needed. However, for 16-bit simulations, the results were no good in terms of application requirements since we obtain a relative error up to 6% (1% is the maximum allowed relative error in shoe last machining). For this reason, we must to double architecture resources in order to improve precision. In this section we only analyze 32-bit results.

Figure 8 shows relative error evolution for the 32-bit MR architecture. A whole round was made using three different steps. Note that relative error is lineally dependent with the number of iterations due to error accumulation. In order to compute a tool path, we will obtain better precision if we use small steps; on the other hand, we will achieve worse angle approximation since the iteration number will grow. Figure 9 illustrates the linear dependency phenomena since there is no significant difference between 0'5, 2'5 and 5º steps. In this graphic, we observe that the relative error is much better than the 16-bit one. As an example, after five iterations an error of 0'001632 % is obtained. For a high iteration number, the relative error values keep under 0'1%, that is, ten times more accurate than maximum application allowed error.

Figures 10 and 11 translate the previous results into number of erroneous bits in data representation. Note that this number increases logarithmically with the iteration number. After 200 iterations we obtain near nine erroneous bits, that is, a bit more of 25% in data length, even in the case the relative error is under 0'075%.

In absolute terms, the error obtained after a whole rotation (360º) for the 16-bit MR could reach the ± 0'5 mm that is no good for machining purposes, however could be suitable for another kind of applications as graphic representation. On the other hand, after 720 iterations to complete a whole round, the 32-bit MR obtains an absolute error of ± 7'8 microns, which is excellent for a shoe last mechanization process.

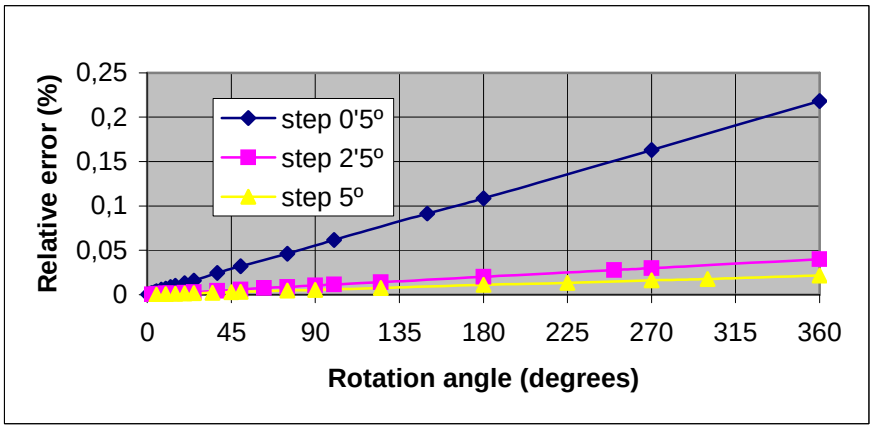

Fig. 8. Relative error versus goal angle for the 32-bit MR architecture.

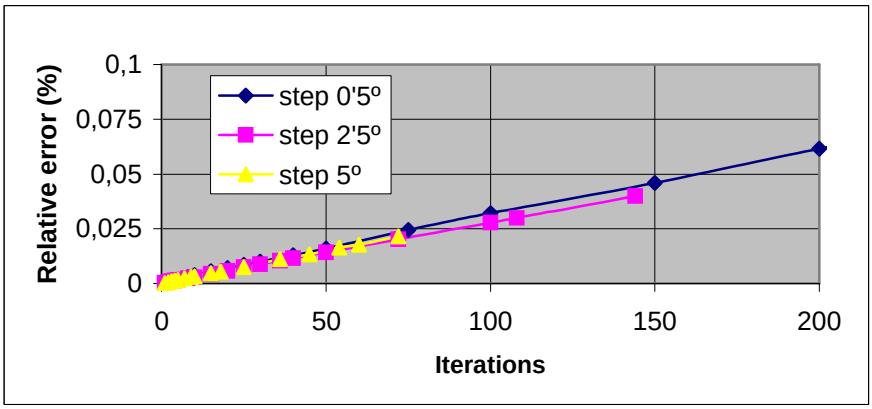

Fig. 9. Relative error versus number of iterations for the 32-bit MR architecture.

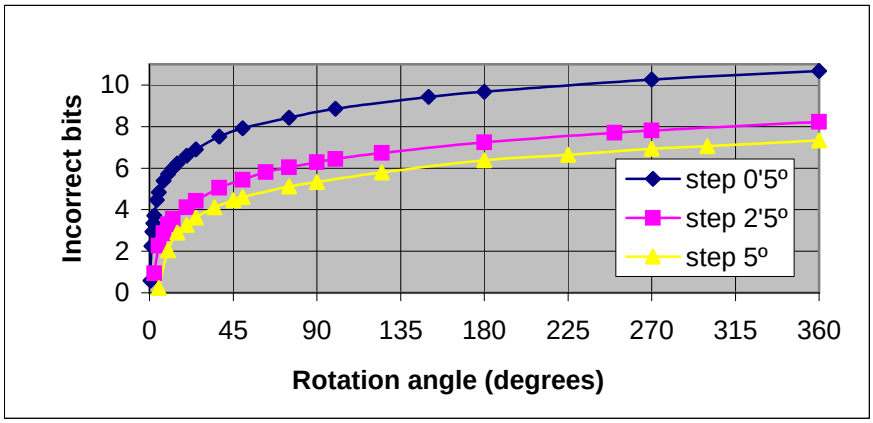

Fig. 10. Erroneous bits versus goal angle for the 32-bit MR architecture.

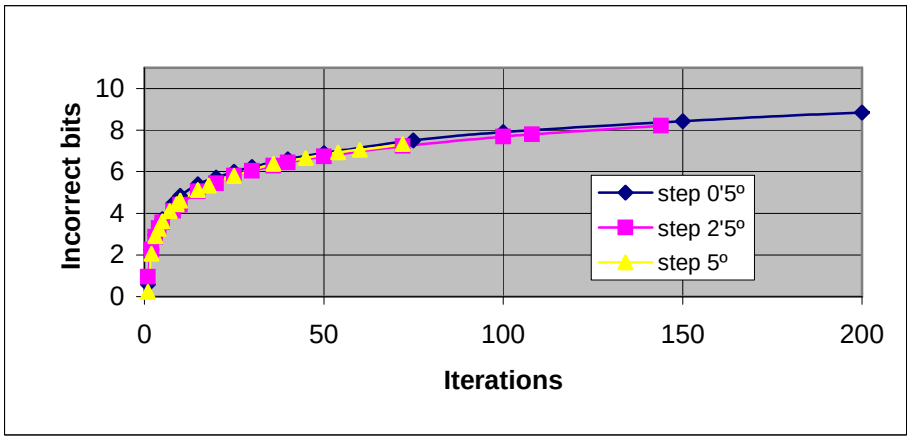

Fig. 11. Erroneous bits versus number of iterations for the 32-bit MR architecture.

6. Conclusions and future investigation

The procedure of virtual digitizing it is simple to implement, offers good results and avoid the problem of tool collision by its own definition. On the other hand, the algorithm is not suitable for general purpose machining algorithms since it is too slow versus other types of tool path generation algorithms. The use of reconfigurable computing may be useful to solve high cost problems coming from the CAD/CAM world. In particular, it can be applied to the virtual digitising algorithm achieving good results in tool path generation.

The general scheme proposes new configurations in order to perform efficiently the most of complex tasks as the distance computation. Future studies will allow developing an intelligent scheduler able to choose the best FPGAs configurations for every kind of trajectory achieving better computing times.

The MR architecture appears as a scalable and powerful FPGA design that accelerates the speed of tool path computation for helicoidal trajectories. Simulations have shown that 32-bit MR is suitable for machining purposes, being more accurate and faster than traditional CORDIC schemes. In this study we are discarded 16-bit implementations, which consume fewer resources; however they could be used for other purposes that require less precision as computer graphics (circle or ellipse drawings).

As future work, we are going to study different MR implementations on real FPGA chip-cards. Best performance chips will be integrated in a CAD/CAM platform in order to test hardware acceleration in tool path generation algorithms.

7. References

1. Jimeno A, García J., Salas F. “Shoe lasts machining using Virtual Digitising”. International Journal of Advanced Manufacturing Technology. Vol 17 No.10 (2001), pp 744-750.
2. Jimeno A, García J., Salas F. “Shoe lasts and computer systems”. (Spanish) Tecnología del Calzado. No. 178 (2000), pp. 73-79.
3. Jung W Park et al **“**Pencil Curve Tracing via Virtual Digitizing” Proc. of IFIP CAPE Conference, (1991), pp.97-104.
4. Takeuchi, Y et al. ‘Development of a personal CAD/CAM system for mold manufacturing’ Annals of CIRP, Vol 38 No 1 (1989), pp.429-432.
5. Jerard, R B, Drysdale, R L and Hauck, K ‘Geometric simulation of numerically controlled machining’ Proc. of ASME Int’l Conf. on Computers in Engineering, ASME, New York (1988), pp.129-136.
6. Farin, G., “Curves and surfaces for CAGD”, Academic Press. 1993.
7. Radunović B., Milutinović V., "A Survey of Reconfigurable Computing Architectures," Eighth International Workshop on Field Programmable Logic and Applications, Tallinn, Estonia, 1998.
8. Volder, J.E.; The CORDIC Trigonometric Computing Technique, IRE Transactions on Electronic Computers, V. EC-8, No. 3, (1959), pp. 330-334.
9. Andraka R. J.; A survey of CORDIC algorithms for FPGAs, FPGA '98. Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays. Monterey, CA (1998). pp 191-200.
10. Jimeno, A., Cuenca, S. “Reconfigurable logic for CAM” (Spanish). Proceedings of the 2001 JCRA. National Conference on Reconfigurable Computing and Applications. Alicante - Spain, (2001). pp 243-252.
11. Stanley A. W. “Applications of Distributed Arithmetic to Digital signal Processing : A tutorial review, IEEE A DSP Magazine, (1989), vol. 6, no. 3.
12. Goslyn G. “The role of DA in FPGA-based signal processing”. Xilinx Inc., 1995.
13. Wang, S. and Piuri, V., “A Unified View of CORDIC Processor Design”, Application Specific Processors, Kluwer Academic (1996), pp. 121-160.
14. Andraka R. J; “Building a High Performance Bit-Serial Processor in an FPGA,” Proceedings of Design SuperCon (1996) sections 5.1 - 5.21.
15. H.X. Lin and H.J. Sips, “On-Line CORDIC Algorithms,” IEEE Trans. Computers, vol. 39, no. 8 (1990), pp. 1,038-1,052.