High Level Library for fast developing pipelined data paths on FPGA: Case study of tool path computation

José Pérez Martínez, Antonio Jimeno Morenilla, Sergio Cuenca Asensi,
José Luis Sánchez Romero
{jperez,jimeno,sergio,sanchez}@dtic.ua.es
Departamento de Tecnología Informática y Computación
Escuela Politécnica Superior de Alicante, Universidad de Alicante
Crta. San Vicente, s/n, 03690 San Vicente del Raspeig, Alicante

Resumen— **The algorithms used for tool path computation demand a higher computation performance, which makes the implementation on many existing systems very slow or even impractical. Hardware acceleration is an incremental solution that can be cleanly added to these systems while keeping everything else intact. It is completely transparent to the user. The cost is much lower and the development time is much shorter than replacing the computers by faster ones.**

This paper proposes to add a reconfigurable coprocessor to improve computation, avoiding the growth of cost due to the substitution of CNC modules. To design the coprocessor, we have developed a hardware library, DTS (DataStream library). This library allows describing simply and quickly pipelined data paths. To verify the viability we have implemented a hardware prototype of a tool path computation algorithm called Virtual Digitizing. This prototype was compared with previous versions of Virtual Digitizing. Also, we use the CORDIC algorithm to consume less logic resources of the FPGA.

Palabras clave— **Virtual Digitizing, Tool-path Computation, Pipelined architecture, FPGA, CORDIC, Handel-C**

I. Introduction

In order to machine a surface by means of a cutting tool on a CNC machine tool, a series of 3D or 2D coordinates that define its motion must be supplied. These points are usually referred to as tool centre positions. In this way, the problem can be expressed as obtaining a trajectory of tool centres that defines the desired object to be machined with a given precision, in literature the problem is also known as the tool compensation problem [1]. In this case, for the sake of simplicity, the problem is presented in 2D. For 3D surfaces the problem becomes more complex.

Partial solutions to this problem use surface offsets generated by different methods [2], [3], [4]. However, these offset-surfaces are restricted to one-radius tools (e.g. spherical, cylindrical and conical) and are not valid for more complex tools, such as toroidal ones with two radix. Moreover, in most cases, self-intersection problems arise according to the surface curvature [5]. Thus, more sophisticated and higher cost computing techniques are needed to detect and solve these problems. Other solutions, based on ruled surfaces, compute the closest ruled surface to an object, although, once again, they are restricted to one-radius tools and 3-axis isoparametric machining [6].

As a case study, we present the efficient implementation of a specific tool path algorithm called Virtual Digitizing [7] (VD). The VD algorithm avoids the problem of tool-surface collision by its own definition. It was inspired by the way mechanical copiers work. These machines touch and move along the reference model's surface, while a group of arms instantly transmit that movement and distances to the cutting wheels in order to shape the workpiece. Similarly, the virtual digitizing approach "virtually touches" the reference surface with the cutting tool (actually it computes the cutting tool position), and makes sure that the machine does not remove any point within the desired object. 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 on both standard (3 or 5 axis machining) and non-standard machining (e.g. in retro-fitting machining where old machines are fitted to be computer controlled).

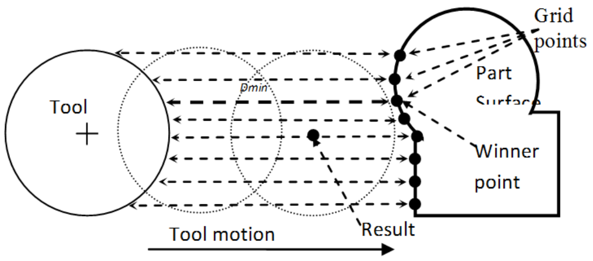

Fig. 1
Example of Virtual Digitizing process for 2D shapes.

In section II, we explain the basic method to obtain a machining trajectory by means of the VD method. It is the basis of the architecture described in section III developed onto an FPGA-based architecture, followed by some real experiments of shoe last machining in section IV. Finally, conclusions and further work are presented in section V.

II. The Virtual Digitizing strategy

This algorithm can be divided into four phases:

1. Definition of the tool motion
2. Obtain a discrete model of the part surface
3. Simulate the tool motion
4. Virtual digitization process

The digitization algorithm becomes simple once the surface and tool motion are well-defined. Basi-

```
1:  $t \leftarrow 0$
2:  while $t \leq 1$ do
3:    $mindist \leftarrow \infty$
4:    for $surface_i \in Object$ do
5:      for $p_{ij} \in Surface$ do
6:        $p'_{ij} \leftarrow p_{ij} \cdot TR(t)$
7:        $d \leftarrow d_y(p'_{jk}, Tool)$
8:        if $d < mindist$ then
9:          $mindist \leftarrow d$
10:       end if
11:     end for
12:   end for
13:   $centre \leftarrow$ computecentre $(mindist, TR(t))$
14:   addtrajectory$(centre)$
15:   increment$(t)$
16: end while
```

TABLA I

Basic virtual digitizing algorithm

cally, the behaviour can be described as follows: For each point of the trajectory, every part surface is transformed in order to face the cutting tool according to the machining strategy.

Once the part is positioned, the minimum distance from every grid point to the tool is computed in the direction of tool attack axis. This minimum distance determines the tool centre point for this step in the virtual digitization process. Physically, we select the point that touches the tool surface first when the tool is moved along the attack axis.

Shoe lasts are machined using turning lathes that use a toroidal tool. Figure I presents the case of shoe last machining. The figure includes pictures with the initial and final shapes of shoe lasts, as well as the helix-shaped trajectory of the workpiece and the tool.

On analysing the algorithm presented in table I, up to three nested loops can be observed. The innermost one is used to access every grid point on the selected surface, that is, it consists of two loops, one for rows and the other for columns. The outermost loop goes through every trajectory position. From experience, in order to obtain a good-quality finish, it is necessary to produce at least as many trajectory points as grid points on the surface. The distance function computes the distance between a 3D point and a 3D torus in the tool attack direction (Y axis) and is expressed in equation 1.

$$D(x,y,z) = T_y - y - \sqrt{(R + \sqrt{r^2 - (x - T_x)^2}^2 - z^2} \quad (1)$$

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

The computational cost of this algorithm is high in some machining scenarios, such as traditional industrial fields, where there are no high-performance computers with regard to design and manufacture. The use of low-performance computers and standard operating systems is therefore a restriction, since they share both the management tasks and those of the CAD/CAM.

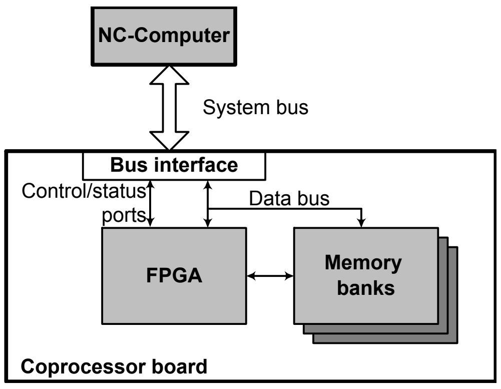

Fig. 2

Generic system architecture

III. System architecture

Figure 2 shows the system architecture proposed to implement the VD algorithm. A coprocessor board with an FPGA works jointly with the software to run into a NC-Computer. NC-Computer chooses the tasks needed during the VD algorithm execution and reconfigures the FPGA using the proper bitstream (through FPGA configuration files). Using the reconfiguration capability of the FPGA allows loading the architecture to run the VD algorithm. To communicate the coprocessor with the NC-Computer a high-performance bus (e.g. PCI, PCIe) is used to provide the bandwidth in order to support bidirectional data transmissions. In addition, multiple memory banks are used to store data temporally, constants, machine parameters, and results. Moreover, there are some ports dedicated to debug HW side from the SW side.

VD algorithm has two nested loops, the inner loop reads each point of the surface and the outer point goes through all trajectory positions. To obtain good quality is necessary to produce at least as many points as trajectory points have the surface. Assuming n is the maximum number of surface points and m is number of positions in the path, then the temporal cost of the algorithm is $O(nm)$. Variables n and m depend on the model size and precision of the machining process. Note that the value on n is a matrix of order ixj, determining the size and type representation for NURBS surface.

There are three different operations in the VD algorithm: point transformation (rotation), tool distance computation (which includes estimation of the area of influence, and computation of T_y) and a comparison. These operations are performed for each point of the tool path and each point of the original surface. Any optimization at these operations will significantly improve the overall computation time. Initially we implement a software version of the VD algorithm, then we develop a software/hardware version which is finally transformed into a hardware version implemented onto an FPGA. Due to the modularity of the operations, this can be done in small steps, which implies slight changes in the hardware. This strategy simplifies and speeds up the debugging process.

The design space has several freedom degrees. Changing the parameters associated with them will give different trade-offs between performance, cost and accuracy. The first trade-off is called "*Software/Hardware partition*", which determines the distribution of functionality between hardware (FPGA) and software. From a software implementation of the algorithm, it has been observed that different functionalities can be migrated to the hardware, each group defines a partition between hardware and software. In [8] we made a comparison between hardware and software partition and an only hardware partition (only FPGA). This paper continues to explore the solution space in purely hardware designs using DTS, a hardware library developed to describe quickly datapaths in Handel-C .

Data types and word length are other design factors to consider. Typically they have a major impact on hardware/software implementations. A powerful CPU has hardware units dedicated to operate with floating and fixed point. A reduction on the number of bits only has effect on memory or performance. The hardware design is much more flexible and allows any number of bits for data word length. A smaller number of bits involves less use of resources in the FPGA. The free resources in the FPGA can be used for greater acceleration, extending any functionality or allowing the use of a smaller (or cheaper) device. Unfortunately, a reduction in the word length also increases the error in the result. FPGA architectures are well suited to fixed-point operations while floating-point calculations require a significant amount of resources. In this paper we focus on fixed-point arithmetic.

Parallelism is another design factor to consider. The main loop in VD consists of functions that are executed sequentially and can be mapped completely in a hardware datapath. The computation can be divided into subtasks to build a pipeline of k stages. A single iteration of the loop will consume k clock cycles. However, the algorithm has no data dependency between iterations and, therefore, after the initial latency of k cycles, the pipeline will produce one result each clock cycle.

A. DTS development library

To help the developing of hardware and exploring various trade-offs between the designing parameters, a hardware library called DTS (Data Stream) was coded. DTS is fully coded in Handel-C [9]. This language is a high level HDL developed in the University of Oxford (and now owned by Mentor Graphics). It is based on C, making it an ideal tool to rewrite the original VD algorithm. The similarity between these two languages provides a short time to migrate the application. Once the first implementation was obtained, it was very easy and fast improving iteratively the development. The only required actions are recompiling and testing our design. Handel-C provides a good balance between output quality and development time. By contrast the classical HDL (Verilog, VHDL) allow greater control over the details of implementation but at the expense of a more lengthy and costly development.

DTS aims at the processing of continuous streams of data (streams). A stream is a container that includes multiple streams of data and flags for synchronization and validation. A stream carries data from a source component, which produces the data, to a destination component that uses them. Thus the configuration of a processing pipeline is defined as a set of components interconnected by streams. The library includes a variety of processing primitives, which consist of components of arithmetic type (Add, Sub,Abs, etc.), relational (Lt, Gt, etc.), operator of memories (RamRead,RamWrite, RAMPL2Read, RAMPL2Write, etc.), logical (And, Or, Not,Xor, etc.), generator constant (constant) and flow control (Delay, Cond, etc.). The main feature of the DTS components is its polymorphism, namely the ability of operators to suit the type of data contained in the streams with which they are connected. This property allows the reuse of code to generate datapaths with different size and format. Once defined and debugged the basic structure of the pipeline, it is only necessary to change the declarations of the streams. Then, the synthesizer will generate, at compilation time, the corresponding hardware instances without user intervention. This will greatly facilitate the study of the various trade-offs between design parameters and accelerate the achievement of optimal solutions.

If the inclusion of any other unavailable operator were necessary, DTS allows adding IP modules (IP cores) from different manufacturers to the design. The user should only define the interface (inputs and outputs) with the external module and instantiate specific DTS macros to communicate with the IP module. These macros automatically perform the conversion between the streams and the signals and buses of the external element.

Table II shows a simple example of using the library DTS. First, define the different streams that interconnect the various blocks of the pipeline as well as its type. In this example streams of type unsigned 32-bit have been defined (lines 1 - 2). In lines 6 to 8 inputs and outputs of the pipeline have been defined by the components *DtsOutput* and *DtsInput* respectively. Then all the elements are arranged to form the pipeline (line 10 - 14). The structure `par{...}`, typical of Handel-C, defines that all these elements are synthesized in separate circuits and work in parallel.

Pipeline components can be in turn pipeline. For example, *DtsCordic* is segmented into 36 stages.

```
 1: DTS_UINT32 (A); DTS_UINT32 (B);
 2: DTS_UINT32 (C); DTS_UINT32 (D);
 3: par {
 4:  /* pipeline */
 5:
 6:  DtsInput (&A); DtsInput (&B);
 7:  DtsInput (&C); DtsInput (&D);
 8:  DtsOutput (&res);
 9:
10:  DtsDelayer (&C, &delay_C, 1);
11:  DtsDelayer (&D, &delay_D, 36);
12:  DtsSub (&A, &B, &resta_AB);
13:  DtsCordic (&resta_AB, &C, &cordic);
14:  DtsLt (&cordic, &D, &res);
15:
16:  /* controlblock */
17:  seq {
18:  par {
19:  DtsWrite (&A, 55);
20:  DtsWrite (&B, 1);
21:  DtsWrite (&C, 0);
22:  }
23:  par {
24: DtsWrite (&A, 37);
25:  DtsWrite (&B, 3);
26:  // ...
27:  }
28: #ifdef USE_SIM
29:  DtsDebug(&res);
30: #else
31:  DtsRead (&res);
32:  }
33: }
```

TABLA II
DTS SAMPLE CODE.

This decreases the depth of the design logic and achieves higher frequency operation. Thus, each component of the library has an associated latency which defines the number of cycles required to produce the first result. When several branches meet together in the same processing component, the user must balance the latencies of each of them so as to synchronize data arrival. To achieve this *DtsDelayer* component is available for introducing no calculation stages in the pipeline. In the example we can see that the CORDIC stream comes from a branch with a latency of 36 (1 of the DtsSub + 35 of *DtsCordic*), while the *D* stream comes from a branch with a latency of 0 (directly *DtsInput*). Therefore for both data streams to arrive synchronously *DtsLt* component is necessary to introduce a delay of 36 cycles in the stream *D*. Also a set of sequential instructions that correspond to the control block are used (lines 17 -32). From this block, data is entered into the pipeline with DtsWrite macro (lines 25 - 30) and the generated results will be read by *DtsRead*. This example shows the simplicity of using or understanding a design developed with the library.

DTS library is designed to simplify debugging using the facilities of IDE "DK Design Suite". This IDE for Handel-C is similar to any other IDE for software development and allows the inspection and modification of the value of the structures and variables during a simulation of the design cycle by cycle (cycle-accurate simulation).

Lines 28-31 show the utilization of the precompiler directives in order to use the *DtsDebug* macro to extract data from the res stream and to route them to the debug console.

B. Prototype proposed

In order to validate the library and explore its possibilities, several prototypes of the VD algorithm have been developed. For co-simulation of the whole system, we have used the DSM library (Data Stream Manager). This API allows the connection of the software side and hardware side without user intervention. It is also possible to co-simulate the whole system to ensure their correct functionality and performance. Thanks to DSM, the system check was carried out with the original test bench of the software version of the algorithm.

The prototype platform selected is the Celoxica RC2000 board. This is a device with sufficient capacity to accommodate the hardware solution, that is, the algorithm with multiple pipelines. It includes a Virtex-II FPGA (XC2V6000) with six banks of ZBT Synchronous RAM 512x32. This family of FPGA is characterized by their high level of integration. Moreover, a large amount of internal memory blocks (BRAM) for fast data access, and a number of embedded multipliers to implement complex algorithms are available. These multipliers use wired logic and do not consume any logic block.

With DTS library, a fixed point (78 stages) pipelined datapath has been developed. The hardware architecture can be seen in figure 5. Three banks of external memory store the surface points (x_i, y_i, z_i) and internal memory blocks (BRAM) store rotation matrices. A fourth external memory bank stores the results of the trajectory computation. For each cycle a surface point is sent to the pipeline and processed as shown in the algorithm in table I. A sequential control block controls the entire VD algorithm execution in the pipeline. It controls all transfers between memories and the pipeline, communications to/from the NC-Computer, generates read/write addresses for memory and updates the value of the parameters before carrying out a new process.

As mentioned in Section III, iterations in the VD algorithm are completely independent one of each other. So they can be executed simultaneously in different pipelines (see Figure 3, right side).

The pipelines are working fully synchronized, sharing memories X, Y, Z. Each pipeline has its own internal memories containing the rotation matrices. In this memories, a portion of the rotation matrices are available, depending on the iterations that

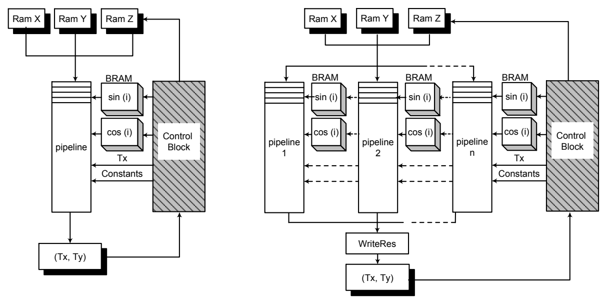

Fig. 3

One pipeline design (left) and multi-pipeline design (right).

each pipeline performs. WriteRes block is used to sequence the n writings of the n results generated by the pipeline on each iteration of the outer loop.

Figure 4 shows the detailed design of each pipeline. All blocks drawn with a white square are developed with the library DTS. The striped blocks and marked as “delayer” are also found in the library DTS, and they are delay elements used for balancing between different branches of the pipeline.

In equation 1, used for calculating the distance between a 3D point and a 3D torus in the tool attack direction, two hard to compute operations appear:

$$partial_{dist}(a,b) = \sqrt{a^2 - b^2} \quad (2)$$

Instead of performing each one of these operations by means of a sequence of two squares, a subtraction, and a square root, they have been performed by including a module that implements CORDIC. This algorithm was originally developed in 1959 for approximating 2D rotations and trigonometric functions using simple binary shift and add operations throughout several iterations. In 1971, the algorithm was extended to support other functions (logarithm, exponential, hyperbolic tangent, and so on). CORDIC can work with circular, hyperbolic, or linear coordinates.

With regard to the calculations required in equation 2, CORDIC with hyperbolic coordinates from now on, CORDIC-h produces a fairly good approximation to the partial distance calculation, as shown in the following expression:

$$Cordich(x_0, y_0, z_0) = \left(\frac{K_h}{\sqrt{x_0 - y_0}}, 0, z_0 + tanh^{-1}\frac{y_0}{x_0} \right) \quad (3)$$

The first component of the resulting tuple shown in 3 coincides with the required calculation, with the difference that the former is scaled by the factor K_h. Therefore, this result must be multiplied by K_h^{-1}. The utilization of CORDIC reduces the consumption of resources per pipeline at the expense of a slight reduction in precision. An optimized CORDIC module for Virtex-II (CORDIC v3.0 Xilinx IP-Core) was generated by means of the coregen tool and integrated within the architecture.

IV. Results

A software implementation of the VD algorithm was used as a test bench. It was coded in ANSI-C and compiled with Microsoft Visual Studio 2005 Express at the highest optimization level (similar to level 3 in the gcc compiler). The hardware used for development and validation was an Intel Core2 Duo at a frequency of 2.33 GHz and equipped with 2 GB RAM. The CORDIC module was implemented by means of Xilinx IP-cores Cordic v3.0 and Multiplier v10.0. The environment for cosimulation and development was Agility DK Design Suite v5, ModelSim VHDL 6.1f, and Xilinx ISE v10.1.

Experiments were carried out by using a shoe last model represented as a 15,270 points surface (131 sections of 120 points each one). Results about FPGA resources occupation are shown in Table III. It can be observed that the DTS approach provides a significant reduction in terms of logic consumption (slices) and multipliers. With regard to pipeline replication, the DTS approach allows up to 10 pipelines, while in previous works [8] only up to 5 pipelines were allowed.

Component	Results in [8]	Results with DTS
Slices	11.263	5.111
BRAM 16Kb	12	1
Mult18x18	28	3

TABLA III

FPGA resources nedded per pipeline (Virtex-II XC2V6000)

V. Conclusions and future work

Tool path computation is an important task within the manufacturing processes. The algorithms required for this stage involve a high computational cost. In this work, we propose the addition of hardware components to the CNC so as to accelerate the algorithms for tool path computation. As a case study, we have implemented a completely hardware version of the Virtual Digitizing algorithm.

For this purpose, a hardware library was coded in Handel-C for designing pipelined data paths. A pipelined data path was developed on the prototype and experiments were carried out so as to validate the design. The Celoxica RC2000 board was selected as the prototyping platform due to the fact that it has enough capacity to accommodate the hardware solution since it includes a Virtex-II (XC2V6000) with six ZBT RAM Synchronous 512x32 banks.

The results obtained show a significant reduction in terms of FPGA resources occupation in comparison to previous works. The accuracy obtained in the experiments is fairly good, although research work must continue to get better accuracy.

Future research will focus on making several improvements in the work presented. It is required to perform a study to find the best compromise between accuracy and performance and to extend the possibilities of the DTS library. The implementation of a

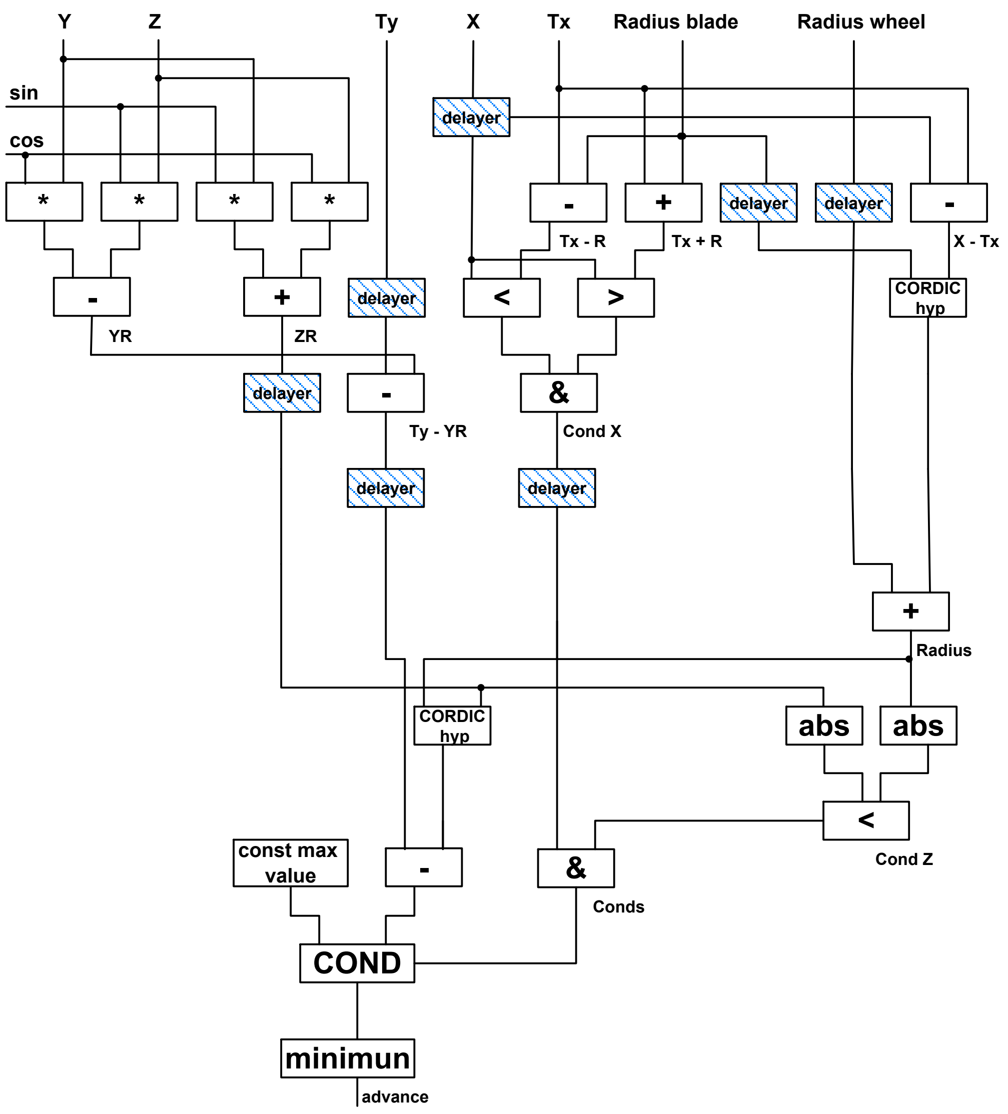

Fig. 4
VD Algorithm solved with a pipelined datapath.

more generic architecture for the VD algorithm must be also studied so as to incorporate a higher number of models, cutting tools and strategies for tool path computation. Another area of future research work is related to the protection of intellectual property. Hardware solutions provide a higher level of protection than that of software, and reverse engineering -especially on ASIC- is a very difficult task. Future designs should take advantage of the protection mechanisms available in new FPGAs.

VI. Acknowledgements

This work has been supported by the *Conselleria de Educacion de la Comunitat Valenciana* under grant GV/2011/043.

Referencias

[1] G. Farin, "Curves and surfaces for computer aided geometric design. a practical guide," *Academic Press Inc.*, 1993.
[2] Y. Wang, "Intersection of offsets of parametric surfaces," *Computer Aided Geometric Design*, vol. 13, pp. 456 — 465, 1996.
[3] B. K Choi, "Surface modeling for cad/cam," *Elsevier Science Publishers*, pp. 263 — 272, 1991.
[4] M. Held, "Voronoi diagrams and offset curves of curvilinear polygons," *Computer-Aided Design*, vol. 30, no. 4, pp. 287 — 300, 1998.
[5] Sakkalis T. Wallner, J., "Self-intersections of offset curves and surfaces," *Journal of Shape Modeling*, , no. 7, pp. 1 — 21, 2001.
[6] Fish R. Elber, G., "5-axis freeform surface milling using piecewise surface approximation," *ASME Journal of Manufacturing Science and Engineering*, vol. 119, no. 3, pp. 383 — 387, 1997.
[7] F.; García-Chamizo J. Jimeno, A.; Maciá, "Trajectory-based morphological operators: a morphological model for tool path computation," pp. 352—357, 2004.
[8] Sergio Cuenca, Antonio Jimeno-Morenilla, Antonio Martínez, and Rafael Maestre, "Hardware approach to tool path computation for step-nc enabled cnc: A case study of turning lathe machining," *Computers in Industry*, vol. 62, no. 5, pp. 509—-518, June 2011.
[9] Mentor Graphics Inc., "Handel-c laguage reference manual: http://www.mentor.com/products/fpga/handel-c/upload/handelc-reference.pdf," Ago 2011.