# 3 Silicon Compilers for Graphics Hardware Design?

*Oliver Renz*
*Alwin Groene*

ABSTRACT Experiences with the realization of an object processor using a silicon compiler will be described. Object processors are parts of the object oriented display processor architecture PROOF (Pipeline for Rendering in an Object Oriented Framework; [9] and [8]). Placed in an object processor pipeline the object processors perform the scan conversion, the interpolation of the depth values and the normal vectors of the primitive objects of a scene to be rendered. The suitability of the silicon compiler GENESIL [1] for the development of graphics hardware will be examined using the object processor as an example.

## 3.1 Introduction

The goal of each chip development is to achieve a high quality design in a short design cycle. High quality in this context means high performance and compact layout. Silicon compilers claim to be a tool for a fast development of chips that can compete with full custom designs. The hardware realization of some typical graphical algorithms allowed us to test the suitability of the silicon compiler GENESIL in this context. The experiences were gained during the following activities:

- the development of an object processor (OP) [4]. Object processors are an essential part of the display processor PROOF. An OP is responsible for the scan conversion and shading of a single primitive object, which is in our case a triangle or a vector. The algorithms and their realization in GENESIL will be described in the following sections.
- the development of an ASIC [2] [7], [5] for the scan conversion of triangles (using an algorithm described in [3]) and of vectors. Antialiasing is performed using subpixel masks.
- Test of the conversion of an existing logical design (a filter stage) into a GENESIL design. This filter stage is like the OP a part of PROOF and performs antialiasing and the handling of transparency.

The experiences with the three architectures were quite similar. Therefore only a few important components of the first mentioned design will be described further.

[1]GENESIL is a trademark of MENTOR GRAPHICS Silicon Design Division.
[2]This work has been partly supported by the Commission of the European Communities through the ESPRIT II-Project SPIRIT, Project No. 2484.

## 3.2 Structure of the OP

In this section a short overview of the parts of PROOF and the essential components of an OP will be given.

PROOF consists of three stages being placed in pipeline manner (see figure 3.1):

1. The object processor pipeline (OPP): The OPP consists of OPs arranged in a pipeline. The task of the OPP is to perform the scan conversion and the hidden surface removal. Further the OPP generates data which is necessary in the next stages. Each OP is responsible for the processing of a single primitive object (triangle or vector). Generally one OP exists for each object in the OPP. The computations of an OP are sequential for each pixel. For each pixel a list of depth sorted objects is maintained. These objects cover the pixel partly (The last object in such a pixel list covers the pixel totally). Because of these pixel lists the filter stage is able to perform the hidden surface removal, the antialiasing and the handling of transparency. In the z-buffer mode the pixel list contains only one object per pixel. This object is next to the viewer and has a coverage different from zero.

2. The shading pipeline: The following shading algorithms are provided: Flat shading, Gouraud shading and Phong shading. The color components (RGB) will be computed with 8 bits of accuracy. In the case of Gouraud shading the OPP calculates the color components or in case of Phong shading the components of the normal vectors. A possible implementation of the shading pipeline is described in [1].

3. The filter stage: If filtering or antialiasing should be performed the OPP generates depth sorted pixel lists. These lists contain geometrical information about the objects, so the filter stage can compute the final pixel color (The filter stage determines the pixel area, that is covered by each object and blends the pixel color corresponding to this area.). This will reduce aliasing effects. A description of this stage is given in [6].

The bottom of figure 3.1 shows the principle structure of an OP. A FIFO receives data and control signals from the predecessor OP and passes the signals to a delay stage and to seven processing units. Three of the processing units (shown as Int. R, Int. B, Int. G) generate the shading data; one computes the z-value (Int. z); another three processing units (Int. d1, d2, d3) perform the geometrical computations on which the coverage unit can decide whether the considered pixel lies inside the pixel. The comparator performs the mentioned management of the pixel lists.

## 3.3 Usefulness of GENESIL Components for the OP Design

The silicon compiler GENESIL combines some advantages of full custom design and the facilities of macro and standard cell design. Both a macro cell library (FIFO, PLA, ROM, RAM, multiplier, ...) and a standard cell library (gates, adders, counters, decoders,latches, multiplexers, ...) [2] are available. In addition GENESIL provides two powerful datapath

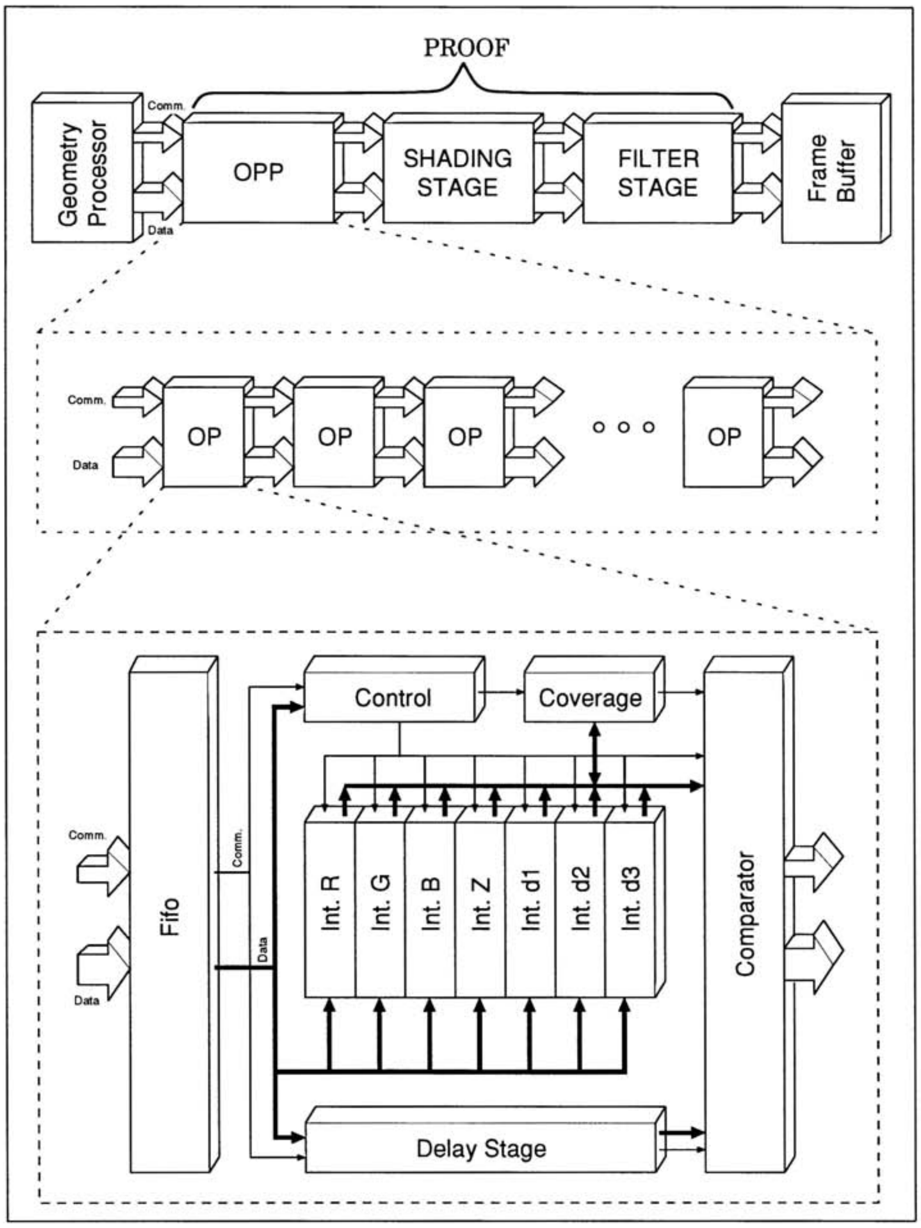


FIGURE 3.1. Proof

libraries[3] for controlling and processing parallel data. This enables a design to use efficient bus operations with high bit widths.

GENESIL forces the designer to use a 2-phase non-overlapping complementary clock for the chip development (But several independent clock regions are allowed.). This clocking strategy prevents unclocked feedbacks. GENESIL carries out automatic consistency checks [4] and accepts a circuit only, if there are no unclocked feedbacks. Therefore an edge triggered design is not possible. The designer cannot built his own latches or flipflops.

The following sections examine the usability of GENESIL circuit elements for the OP design and similar graphics hardware.

### 3.3.1 Datapath Module

All pixel calculations in the OP can be done by interpolating a bilinear equation of type

$$\mathbf{F(x,y) = ax + by + c}$$

(x, y are the screen coordinates of a pixel).

These calculations include

- three equations to determine the distance between a pixel and a primitive object,
- three equations to interpolate the colors/normal vectors in a triangle, and
- one equation to calculate the z-value of pixels covered by a triangle.

The incremental calculation of values **F(x,y)** during a frame is done similar to the following algorithm:

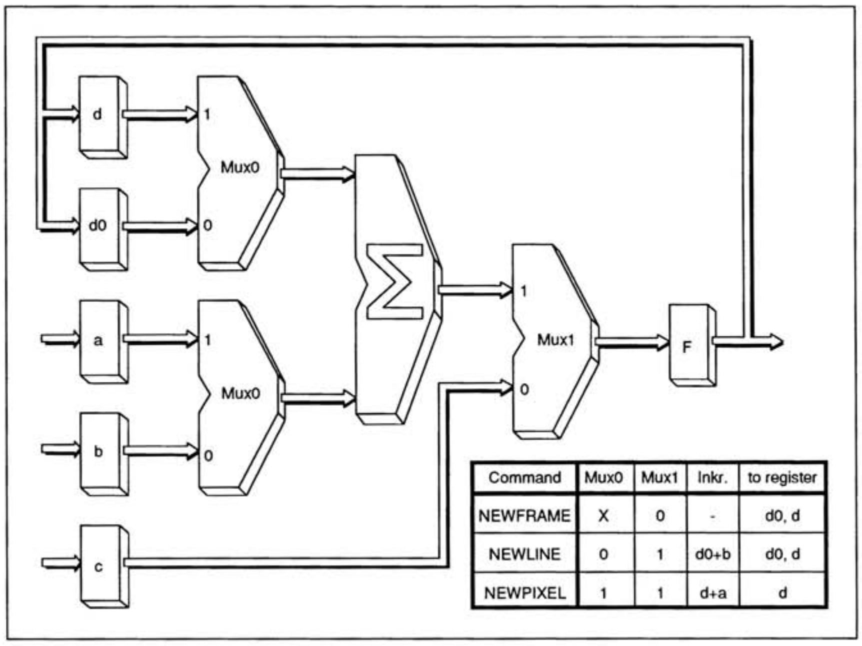
```
F(0,0):= c;                                  /* New Frame */
for y=0 to max_scanline-1 do
begin
      for x=0 to max_pixel_per_scanline-1
            F(x+1,y) := F(x,y) + a;          /* New Pixel */
      F(0,y+1) := F(0,y) + b                 /* New Scanline */
end
```

This algorithm can be implemented in each of the seven interpolation units as shown in figure 3.2.

The initializations and calculations are controlled by three multiplexers (and of course by some load signals for the latches which are not shown in figure 3.2). The command NEWFRAME loads the c-values into the registers 'd' and 'd0'. The NEWPIXEL command writes the output of the adder (d+a) into the register 'd'. Last but not least the NEWLINE command writes the output of the adder (d0+b) into the registers 'd' and 'd0'.

[3]The Parallel Datapath Library for 1.5 $\mu m$ or 2.0 $\mu m$ fablines (supporting IOTA1/1.5 design rules) and the so called New Datapath Library for 1.0 $\mu m$ fablines (IOTA2 design rules).

[4]This is done by timing attributes attached to the circuit elements and their signals. The timing attributes express the relationship between the clocks and the signal timing.



| Command | Mux0 | Mux1 | Inkr. | to register |
|---|---|---|---|---|
| NEWFRAME | X | 0 | - | d0, d |
| NEWLINE | 0 | 1 | d0+b | d0, d |
| NEWPIXEL | 1 | 1 | d+a | d |

FIGURE 3.2. The structure of an interpolation unit

The interpolation units are almost identical. The main difference is the number of bits. The distance calculations are done with 30 bit, the color values (or components of normal vector resp.) with 39 bit, and the z-value calculations with 47 bit fixed-point numbers. Because of these large bit widths these units can be implemented only with GENESIL datapath modules. Usually a GENESIL datapath module consists of several functional units (called datapath blocks). The following operations can be implemented with datapath blocks:

- addition, subtraction (complement of 2)
- comparison operations
- ALU functions (a complete ALU module is available)
- shift operations
- random logic operations
- latch operations
- control of data flow (e.g. multiplexer, I/O-modules, change bus functions)

All operations are also available for *tristate* and *precharge* signals. The datapath blocks receive their input data via I/O-ports or by two global data busses. In addition neighboring blocks can be linked by two local busses.

Figure 3.3 shows the internal datapath structure of the interpolation unit for z-values.

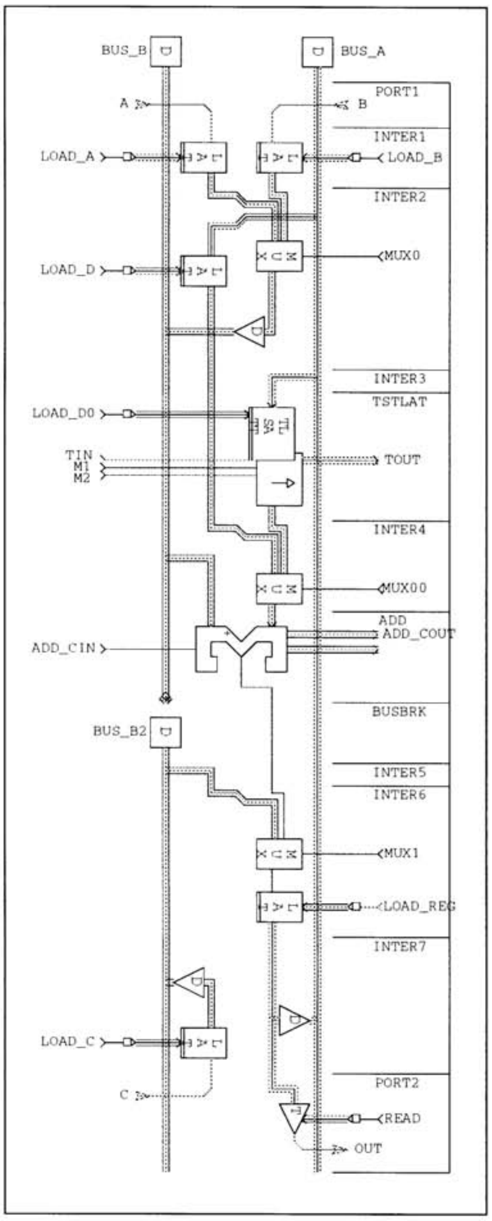


FIGURE 3.3. Datapath of a z-interpolation unit

The four busses allow quite complex connections between the blocks. Feedbacks are possible and I/O-ports can be attached in each place of the datapath modules. Especially the possibility to extend datapath registers to testability latches is remarkable. These testability latches are able to shift data into datapath registers as part of a scanpath during a test mode. Analogously data can be examined by shifting out. Though testability latches are four times as large as normal latches, therefore it's reasonable to use them sparingly.

The designed datapath modules consist of a great number of datapath blocks. This is the reason why they are large and slow (up to 60 ns cycle time). Especially the adders (width up to 47 bits) and many long busses connecting various datapath blocks cause these high signal delay times.

Compared with the other library elements GENESIL datapath modules reach a high integration density. The density varies depending on bit slices including just interface operations or more complex ALU operations. Figure 3.4 shows the layout of a d-interpolation unit.[5] The bit slices run vertical in groups of four. The adders in the middle of the slices are quite compact (approx. 100 $\mu m^2$ per transistor). On the other hand they increase the distances of multiplexers and registers between slices significantly. The resulting area per transistor of the whole datapath amounts to about 500 $\mu m^2$ and with New Datapath Blocks (IOTA2) about 200 $\mu m^2$. Unfortunately we could not use IOTA2 design rules.

### 3.3.2 Random Logic Modules

GENESIL random logic modules are user assembled macro cells which consist of standard cells (so called random logic blocks). The following random logic blocks are available:

- gates
- transparent latches and flipflops (1-16 bits)
- more complex functional blocks like multiplexers, decoders and adders
- blocks with internal states like counters
- buffers (e.g. tristate or precharge drivers)

The blocks vary in their width but have a uniform height. The width depends on the specified function and the number of bits. Data enter and leave the cells always on the upper and lower sides of a cell. The power supply and clock connectors are on the left and right sides of a cell. During floorplanning it is reasonable to place the RL blocks in one row (see figure 3.5).

Obviously the routing of neighboring blocks is unfavorable because it cannot be done via the adjacent left and right sides. Instead U-turns have to be used. This long routing results in large chips and higher signals delays. Usually an equivalent design with PLA blocks or data path modules is more compact and therefore faster. The integration density of RL modules is 2-3 times less than that of datapath modules.

[5]This layout was built in a fabline supporting only IOTA1.5 design rules. The same datapath built with New Datapath Blocks (IOTA2 design rules) was about 2 times smaller.

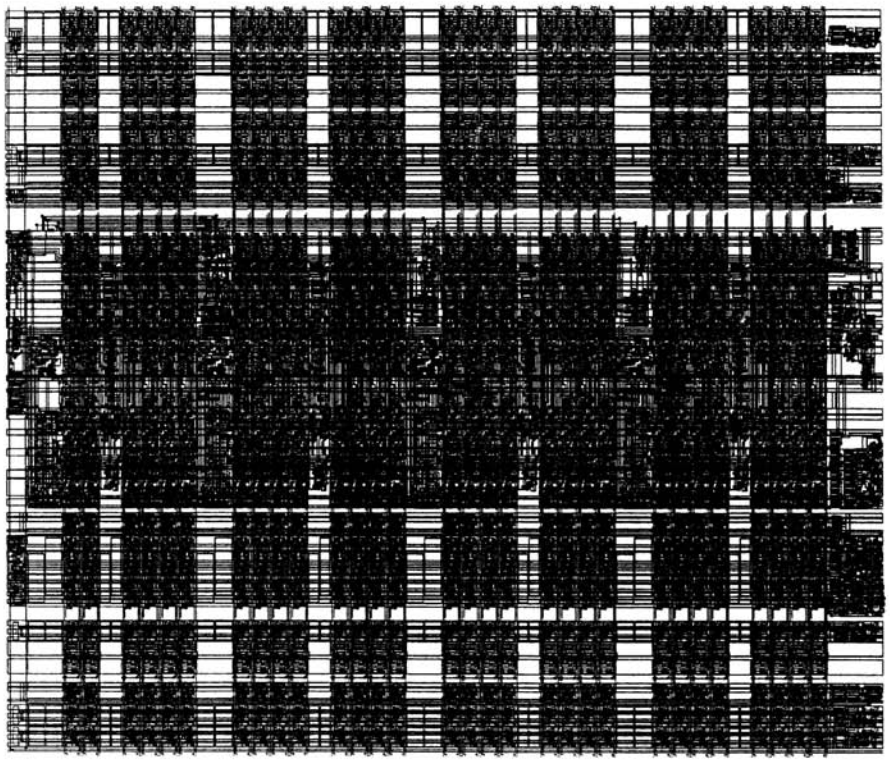


FIGURE 3.4. Layout of a d-interpolation unit

### 3.3.3 Independent Blocks Library

The independent blocks library consists of macro cell elements which occupy more chip area than the standard cell elements in the random logic library. The following independent blocks are available:

- FIFO-blocks (2-64 bits in depth and width)
- Multiplier (4-32 bits (old) or 8-80 bits (new) ; both 2-complement)
- Data/power/clock/test-pads, clockprocessor-pads
- PLA-blocks (max 512 minterms and max. 256 inputs and outputs)
- RAM-blocks (2-128 bit depth, 2-16 address bits)
- ROM-blocks (2-128 bit depth, 4-12 address bits)

All blocks support *tristate* and *precharge* signals.
The PLA blocks are quite remarkable. The AND and OR planes can be specified either as object code (minterms) or in a Pascal like PLA description language called PLAEQ (PLA

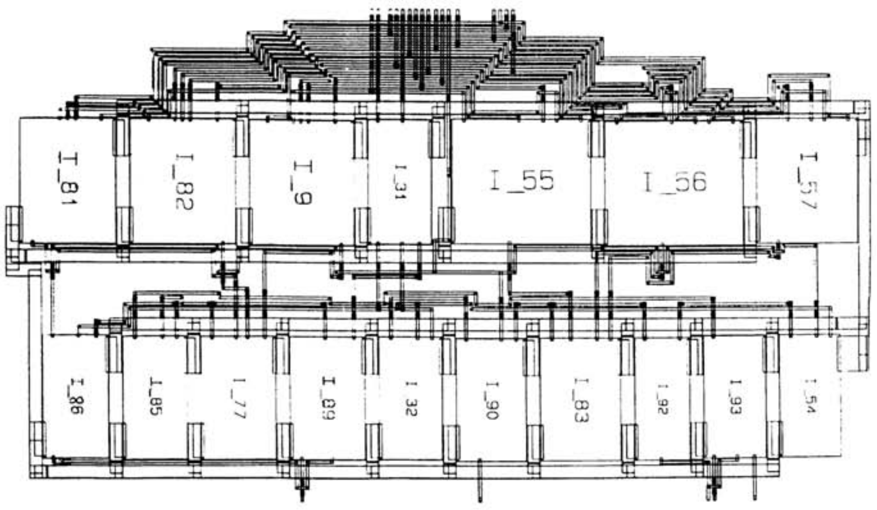


FIGURE 3.5. Routing of random logic blocks

Equations). With PLAEQ the PLA can be defined with logical equations or as actions of a finite state machine. In all cases the definition can be optimized. The generated PLA layout is as dense as the layout of a parallel datapath.

In order to control the interpolation units a gate version and a PLA version were developed. The PLA version was much more compact and faster. The following figure 3.6 shows the layout of a PLA with 47 minterms [6]. and about 1000 transistors.

### 3.3.4 Logic Compiler

With the Logic Compiler tool all mentioned library elements can be optimized (with regard to aspect ratio, connector positions, additional routing channels ...). Totally new blocks can be defined via functional models. These can be transformed into GENESIL blocks. Unfortunately the Logic Compiler tool is optional and was not available for the OP design.

### 3.3.5 Floorplanning, Routing, Timing-Analysis

The GENESIL floorplanning of a chip or a module consists of the following activities:

- Placement of blocks and modules (orientation to each other).

[6]There were 60 minterms before GENESIL made the automatic optimization

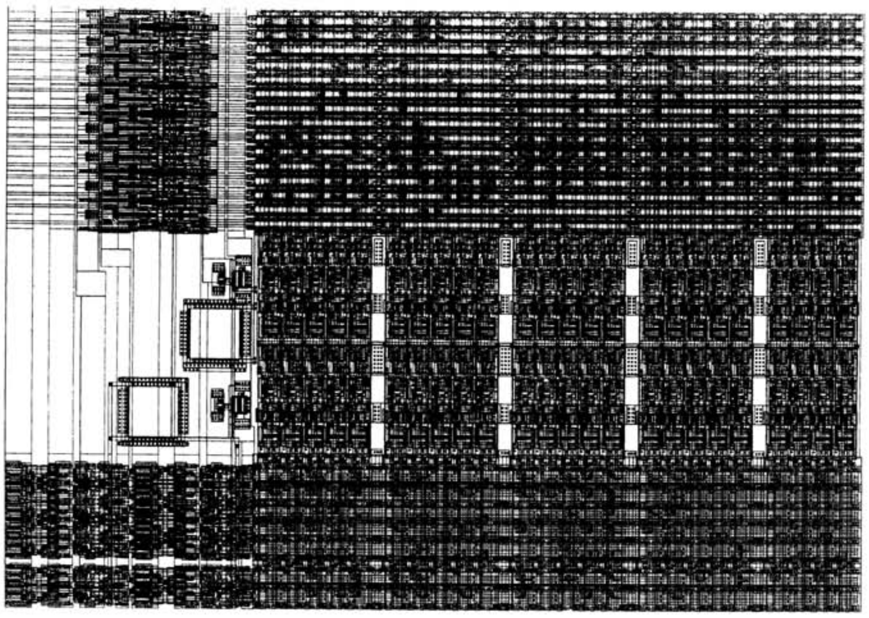


FIGURE 3.6. Layout of a PLA

- Slicing and Alignment: Specification of the routing channels between the blocks (slicing) and definition of a growing direction of these channels or a direction for moving the blocks, if this is necessary during the routing (alignment).
- Pinout: Definition on which sides the connectors of a module or the pads of a chip shall be located.

After these three steps the floorplanning is finished and the module or the chip can be routed.

GENESIL offers an automatic placement which is only acceptable if the placement is done with a small number of modules. Placement is an iterative and therefore time-consuming process because GENESIL offers none of the common aids (like e.g. simulated annealing). It should be optimized to achieve both small chip areas and short wire length. Therefore the designer is forced to compromise between these two demands.

Routing can be done automatically. However usually it is necessary for the designer to influence the router by additional instructions. Each data/clock-signal can be controlled with these additional router instructions [7]. Unfortunately this controllability is restricted to instructions telling the router which slicing channels to use and which to avoid. U-turns

[7]However it is very difficult to influence the routing of VCC and VSS.

and other useless passing actions of the router caused by signal crossings or unfavorable slices are difficult to control. In many cases they are not at all controllable.

During design optimization a timing analysis can help to find critical paths. Afterwards the floorplan may have to be modified. The timing behavior can be improved by redoing critical paths or by a different placement. Especially if the chip size is already quite large, such a *floorplaning-routing-timing-cycle* is very time-consuming and therefore an optimum is hardly ever reached.

The timing analysis of any module can be done immediately after defining the circuit and is independent of any test vectors. Therefore the designer gets the circuit performance (setup/hold/delay times, cycle times including phase-high times, circuit power consumption) almost instantaneously.

Due to the large databus bit-width connecting the interpolation units the routing area is enormous (almost half of the core area).

Figure 3.7 shows the routed OP.

## 3.4 Some OP Performance Data

The following table shows some performance data of the OP:

| | | |
|---|---|---|
| Technology | CMOS-1 | |
| Bonding Pads / Pins | 113 | |
| Number of transistors | 87691 | |
| Process | ES2 CMOS 1.5 $\mu$ | VTC CMOS 1.0 $\mu$ |
| High | 634.9 mils ( = 16.1 mm ) | 505.4 mils ( = 12.6 mm ) |
| Width | 553.2 mils ( = 14.0 mm ) | 429.3 mils ( = 10.9 mm ) |
| Area per transistor | 4.0 mils$^2$ ( = 2586 mm$^2$) | 2.4 mils$^2$ ( = 1596 mm$^2$) |
| CORE routing | 46 % | 53 % |
| CHIP routing | 37 % | 42 % |
| CORE area | 80 % | 80 % |
| PADRING area | 19 % | 19 % |
| Ratio PADs/PADRING | 76 % | 80 % |
| Symmetric cycle time | 165.0 ns ( = 6.0 MHz ) | 96.2 ns ( = 10.4 MHz ) |
| Minimum cycle time | 146.5 ns ( = 6.8 MHz ) | 96.2 ns ( = 10.4 MHz ) |
| Power dissipation | 3.7 W | 3.9 W |

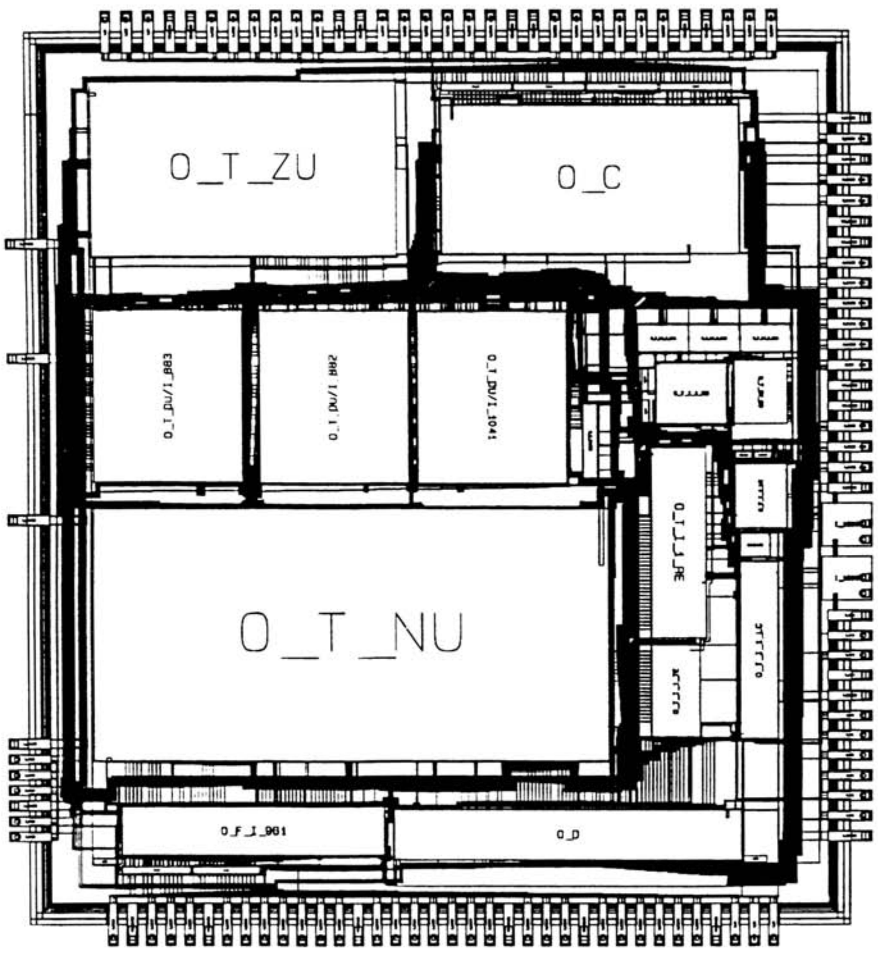


FIGURE 3.7. Routed OP

## 3.5 Some General Experiences with GENESIL

Fast GENESIL chips (with 20 Mhz and more) can (probably) be developed on the following conditions:

- The die size of the chip should not be larger than 300 x 300 mils$^2$ (7.5 x 7.5 mm)$^2$.
- The number and length of the nets should be low. The best circumstances are provided by directly connected neighboring modules. Otherwise the routing control is very time-consuming.
- If possible do not use greater bus width than 30 bits. Bus operations should only be implemented as Parallel Datapath Modules or New Datapath Modules.
- The modules shouldn't be too large in order to get a good floorplan. If the differences of the modules are too great, this leads nearly always to wasted chipspace.
- All modules should be optimized with the Logic Compiler tool.

Many graphics hardware applications are more complex than the considered designs concerning required computations. If similar databus bitwidth is used, the use of GENESIL for a hardware implementation can only be recommended if the control logic can be designed much easier than the OP control.

## Acknowledgements:

We appreciate the guidance of Prof. Wolfgang Straßer and the cooperation with our colleagues at the Graphics Department of the University of Tübingen.

## 3.6 References

[1] U. Claussen: Verfahren zur schnellen Beleuchtung u. Schattierungsberechnung. Dissertation der Fakultät für Physik der Universität Tübingen, 1990

[2] GENESIL-Designer: Compiler Library Volume I, II, III. Silicon Compiler Systems (now MENTOR GRAPHICS), Sep. 1989

[3] J. Pineda: Parallel Algorithm for Polygon Rasterization. *Computer Graphics*, 22(4):17-20, Aug. 1988

[4] O. Renz: Entwurf eines Objektprozessors mit Hilfe eines Silicon-Compilers. Diplomarbeit am Wilhelm-Schickard-Institut für Informatik der Universität Tübingen, Nov. 1990

[5] O. Renz: Implementierung des SPIRIT ASIC. Interner Bericht des Wilhelm-Schickard-Instituts für Informatik der Universität Tübingen, Mai 1991

[6] Cl. Romanova and U. Wagner: A VLSI-Architecture for Anti-Aliasing. In: R.L.Grimsdale, W.Strasser (Eds.):*Advances in Computer Graphics Hardware IV*, EurographicSeminars. Springer-Verlag, Berlin, 1991, p.75-90

[7] A. Schilling: Some Practical Aspects of Rendering. In: R.L.Grimsdale, A. Kaufman (Eds.):*Advances in Computer Graphics Hardware V*,EurographicSeminars. Springer-Verlag, Berlin, 1992, p.53-66

[8] B.-O. Schneider: Eine objektorientierte Architektur für Hochleistungs- Display-Prozessoren. Dissertation der Fakultät für Physik der Universität Tübingen, Feb. 1990

[9] B.-O. Schneider and U. Claussen: PROOF: An Architecture for Rendering in Object Space. In: A.A.M.Kuijk (Ed.): *Advances in Computer Graphics Hardware III*, EurographicSeminars. Springer-Verlag, Berlin, 1991, p.121-140