Slide #1

Using VHDL Abstract Data Types to Design a High Performance 3-D Graphics Pipeline

Design **SUPERCON97**

Talcian Corporation
212 Powderhorn Trail
Broomfield, Colorado 80020
(303)466-1117 Fax: (303)466-1118
www.talcian.com

TALCIAN

Slide #2

Thomas Tessier

Talcian Corporation
TTessier@Talcian.com

Current Activities
Tom currently manages Talcian's VHDL training services, teaches VHDL training courses, keeps an eye on our recalcitrant NT, UNIX, Linux and Macintosh systems, and of course does ASIC design.

Author Background
Mr. Tessier has spent the past 13 years working as a design engineer. In January 1996 he joined with Tim Davis and Tom Bishop as founders of Talcian Corporation. Talcian supplies design services, methodology assistance and training to the electronics industry specializing in the application of Hardware Description Languages. Prior to that he worked as an electrical engineer at Ball Aerospace for 11 years where he did a lot of rocket science. Tom has a BSEE from Colorado State University.

TALCIAN

Slide #3

Thomas Bishop

Talcian Corporation
TBishop@Talcian.com

Current Activities
Tom spends his days dreaming about FPGA's and ASIC's when he isn't out delivering VHDL training or trudging through the seemingly endless business chores associated with being Talcian's Chief Financial officer.

Author Background
Mr. Bishop spent the past 12 years working as a hardware design engineer, primarily in the video and graphics industry. In January 1996 he joined with Tim Davis and Tom Tessier as founders of Talcian Corporation. Before joining Talcian he acted as an independent consultant to the Video industry for 5 years. He developed his deep interest in video systems at Ampex Corporation where he toiled day and night for 5 1/2 years.

TALCIAN

Slide #4

Timothy Davis

Talcian Corporation
TimDavis@Talcian.com

Current Activities
Tim is currently the HDL methodology geek for Talcian Corporation. When not training new recruits in VHDL, he retreats to his abstract plane of existence where he invents new design methodology guidelines to torture Tom and Tom with.

Author Background
Mr. Davis has been a hardware design engineer for 11 1/2 years since obtaining an MSEE from the University of Kansas. In January 1996 he joined with Tom Bishop and Tom Tessier as founders of Talcian Corporation. Prior to incorporation, he designed part of NCR's SCSI Protocol Processor ASIC, worked with customers as a Synthesis Apps. Engr. with Mentor Graphics and then wondered into ASIC design consulting where he stayed put for four years.

TALCIAN

Slide #5

Presentation Overview

- **Project Scope**
- **Abstract Data Types**
- **Benefits**
- **Conclusion**

Slide #7

Goals

- **Design High performance 3-D graphics system**
- **Develop chip set quickly**
- **Produce design that is...**
  - tool and technology independent
  - extendable and re-usable
- **Meet Schedule even though starting 14 months behind!**

Slide #6

Project Scope

- **Goals**
- **Setbacks: Behind the Eight Ball**
- **3-D Graphics Pipeline**

Baseline design for the 3-D graphics system was a simulation engine which fit into a refrigerator size cabinet and required 220V supplies to power it. We needed to develop a system which met or exceeded the current systems performance into a 9" by 12" PCB. The Arcade Games and Virtual Reality entertainment was the intended market.

Management wanted the ASICs and the system As Soon As Possible (ASAP). Fourteen months of an eighteen month schedule was already gone. We needed to get to the market quickly, the customers were waiting.

Management saw the value and the potential for future extensions. The consolidation of the design into a single chip was an ultimate goal. Management desired the flexibility to develop future products with the intellectual property.

Setbacks: Behind the Eight Ball

- **Given three models**
  - C language model
  - Highly structural, tool dependent, RTL VHDL
  - Gate level net list
- **Management wanted design band-aid**
- **VHDL Models**
  - Floating point precision analysis unavailable
  - Control flow decentralized and nearly invisible
  - Design hierarchy resembled bowl of spaghetti
- **Way past delivery date**

TALCIAN

Slide 8
Copyright © 1996 Talcian Corporation

3-D Graphics Pipeline

- **System Overview**
  - Block Diagram
  - Graphics Attributes
- **Render Engine**
  - Basic Mathematics
  - Block Diagram
  - Span Walker
- **Issues to Resolve**

TALCIAN

Slide 9
Copyright © 1996 Talcian Corporation

Three models of the 3-D graphics pipe were available.

- The C language model was based on modules derived from the "Graphics Gem" series of books. Several knowledgeable graphics architects extended the base models to better model our hardware system.

- The RTL VHDL was provided by another design group who were having trouble completing the design. This model was very tool dependent and looked like a schematic generated with VHDL.

- The gate level net list was already converted into the foundry specific form. Upon simulation we discovered that it didn't function.

Management first insisted that we band-aid the current design. They desired closure of the product development effort.
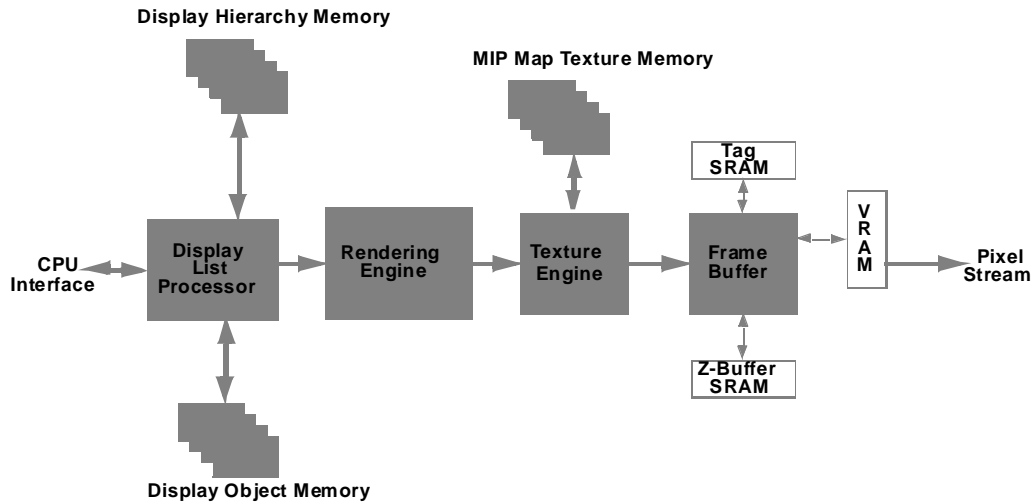
After analysis we determined that the gate level model was flawed, the RTL level model was hap-hazardously put together from very low level structural models. Neither met the specification or functionality set forth in the C language model.

Management liked the C language model and we decided to start anew from that baseline.

In the following section we give an overview of the 3-D Graphics Pipeline as we designed it. We focus our attention on the Render Engine to show the data types and mathematics involved in 3-D graphics. Finally we reduce the scope down to the span walker which highlights the issues we faced.

The span walker will be used throughout the paper to illustrate the VHDL coding styles necessary to use ADTs.

Block Diagram

Display Hierarchy Memory

MIP Map Texture Memory

Tag SRAM

V R A M

CPU Interface

Display List Processor

Rendering Engine

Texture Engine

Frame Buffer

Pixel Stream

Z-Buffer SRAM

Display Object Memory

TALCIAN

Slide 10
Copyright © 1996 Talcian Corporation

The 3-D graphics pipe consisted of 4 chips: Display List Processor, Render Engine, Texture Engine and the Frame Buffer.

The Render Engine accepted triangle data from the Display List Processor and created pixels at a peak rate of once per clock while transversal of a scan line.

Graphics Attributes

- **Triangle Based**
- **Vertex Attributes**
  - Screen/World Coordinates (X, Y: Integer)
  - Color (R, G, B: Integer)
  - Depth (1/W: Floating Point)
  - Texture Coordinates (M:Fixed point; U, V:Floating Point)
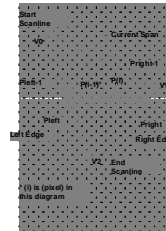 - Translucency (T: Fixed Point)

TALCIAN

Slide 11
Copyright © 1996 Talcian Corporation

Rendering Engine Base Equations

Equations using RED:
- red(Pleft) = red(Pleft - 1) + red(v2 - v0) / y(v2 - v0)
- red(Pright) = red(Pright - 1) + red(v2 - v1) / y(v2 - v1)
- red(pixel) = red(pixel - 1) + red(Pright - Pleft) / x(Pright - Pleft)
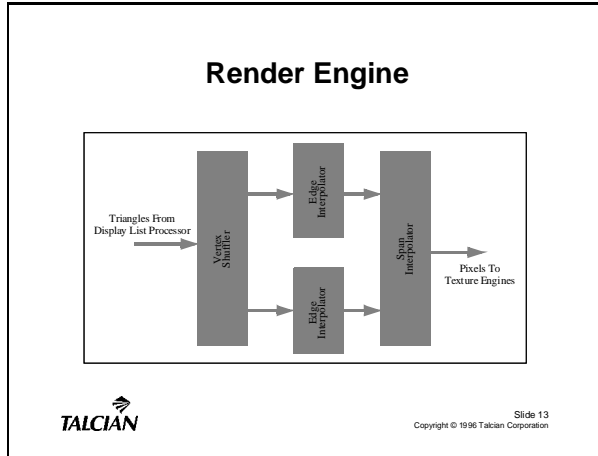

TALCIAN

Slide 12
Copyright © 1996 Talcian Corporation

The 3-D graphics pipe used triangles as its base graphic unit. Each triangle consisted of 3 vertices with each vertex containing the attributes of:

- World coordinate (in the Display List Processor) which was represented in real numbers; or Screen Coordinates which could be represented as integer numbers.

- Three color values for red, green and blue. Different markets demand different color depths, this pipe was based on 8 bits per color.

- Depth which represented perspective corrected Z was a floating point number.

- Texture coordinates are floating point values, and are accompanied by a MIP value, which sets the anti-aliasing blur.

- Translucent triangles are used for modeling glass, like cockpit canopies. A fixed point number represented the Translucency factor.

The Render Engine decomposed a triangle by traversing the left and right edges simultaneously in the Y direction thus producing a span. The span left and span right are the base points for generating pixels in the X direction.

Render Engine

Slide 13
Copyright © 1996 Talcian Corporation

Slide #14

Span Walker

Slide 14
Copyright © 1996 Talcian Corporation

The Render Engine could easily be broken down into three base operations:

- The vertex shuffler took unordered vertices from the Display List Processor and re-ordered them to a counter clockwise rotation. This rotation always guaranteed that the first vertex had the minimum Y value. The next two vertices were calculated at by the minimum X value with a secondary decision maker of the least Y value. The shuffler also determined if the pixel was minimally visible by calculating the area of the triangle.

- With the vertices ordered it was easy to compute the left and right edges. Each edge interpolator would compute a new starting and ending span values by computing the deltas and walking the edge; the render base equation was used.

- With new span values, start and end (left and right), the span interpolator would walk along in the X direction and produce the pixels. Again the rendering base equation was used.
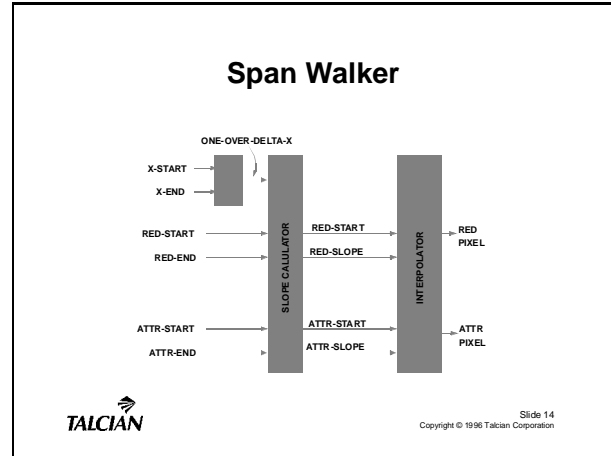
The Span Walker consisted of a reciprocal function, a slope calculator and the interpolator.

In actual implementation the reciprocal function took many clock cycles to complete. We built it as a separate component from the rest of the design so we could spend as much time as necessary optimizing it. The reciprocal was indeed the pacing item in the design.

Issues To Resolve

- **Attribute data representation**
  - Understand math precision
  - Keeping track of data types in the VHDL
- **Large, geographically diverse team**
- **Time-To-Market**
  - Prepare one VHDL model
  - Little time for math analysis
  - Simulate many image databases

TALCIAN

Slide 15
Copyright © 1996 Talcian Corporation

Solution:
Abstract Data Types (ADT)

- **Improve "readability" of VHDL**
- **Simulate with high level data types**
- **Rapid switch to synthesis - no change to architecture required**

TALCIAN

Slide 16
Copyright © 1996 Talcian Corporation

As with any numerical problem represented by digital logic, one has to understand the precision required to get the desired results. We needed time to understand the math operations to derive the required precision.

Another key was to develop data representations that have meaning. This was doubly important because in order to complete this design ASAP we needed to employ a diverse team in geographically separate locations. Clear information transfer was necessary and descriptive data representations aided that requirement.

Time to market was key. We could not afford to develop more than one model or solution. KISS (Keep It Simple Stupid) was our goal and with simplicity comes elegance.

We needed to get up and running as quickly as possible thus one model (which could easily represent both behavioral and synthesis styles) was desirable. Because of the time-to-market pressures we had little time to analyze the math precision required. We needed images quickly to get subjective results about quality instead.

Finally, management demanded that we simulate images as much as possible. The previous team could never produce images using the VHDL models —only the C language model. Management wanted warm fuzzies.

Abstract Data Types

- **Design Capture Paradigms**
- **Definition of ADT**
- **VHDL Support For ADT's**
- **ADT's In Our Design**

TALCIAN
Slide 17
Copyright © 1996 Talcian Corporation

- We start by setting the stage for ADT's by discussing the different methods for capturing design data.

- A definition for ADT's provides a foundation for understanding what VHDL must support to utilize their benefits.

- VHDL has a number of features which support ADT's as opposed to merely enabling their use. VHDL provides convenient (easy, safe and efficient) facilities to support the creation of ADT's as opposed to merely enabling them.

- Lastly, we will examine how ADT's were used in the 3-D graphics pipe design through some simplified examples of real code used in the project.

Design Capture Paradigms

- **Procedural**
  - Focus solely on processing
  - Example: SQRT function
- **Modular**
  - Focus on data organization
  - Example: FIFO module
- **Data abstraction**
  - Focus on type of data
  - Set of operations
  - Example: Floating point, +, -, *, /

TALCIAN
Slide 18
Copyright © 1996 Talcian Corporation

There are a multitude of ways to capture a design. The focus of the designer however tends towards one of several general approaches.

Abstract Data Type Definition

- **"Type" represents abstract set of values**
  - Example: Integers represented as bit vectors
- **Compiler must create objects of given type**
  - Signals, variables, constants, generics, etc.
- **Set of operations**
  - +, -, *, /
  - =, <, >
  - is_negative, is_zero, is_positive
- **Goal: Insulate user from underlying representation of abstract values**

TALCIAN
Slide 19
Copyright © 1996 Talcian Corporation

An abstract type is defined as a set of values and the operations that can be performed on those values. The specification of how the values are represented is what is "abstracted". The details are hidden from us. If the details of the value implementation are hidden then the operation details must be hidden as well. Hence, the user of the ADT doesn't have to concern him/herself with anything but "how can I use this type given this set of operations?" Consider the abstract set of integers. We could represent each integer by counting electrons or perhaps by distinct frequencies of light. Typically, in a hardware system the integers are represented in binary vectors. (Even then there are multiple ways to do that).

The compiler or synthesizer must be capable of automatically creating things of a given type based on its specification. This is critical, otherwise we are back to modular design.

Once an implementation for the data is selected then the operations specified by the ADT can be implemented. In the case of the integers, we would probably want multiply, divide, add and subtract at a minimum. Since the integers are ordered, relational operations might be necessary too.

Ultimately, we don't want the user of the type to know (or care) about how the values are represented. That way the types can be changed to better representations at a later date without requiring user intervention to consider possible side effects.

—Key requirement for our design is floating point precision may change very late in the design cycle.

VHDL Support

- **Type Declarations**
- **Sub-programs and overloading**
- **Strict type checking**
- **Packages**

TALCIAN

Slide 20
Copyright © 1996 Talcian Corporation

- VHDL directly supports the creation of abstract data types through type declaration statements.

- The operations for those types can be specified with sub-programs (procedures or functions) -- overloading them if necessary.

- Strict type checking is a hallmark of VHDL and makes the creation of ADT's possible.

- Packages are special design units in VHDL that allow an engineer to capture frequently used declarations and their definitions in one place to facilitate re-use.

Type Declarations

- **TYPE statement**
- **Constrain built-in types**
- **Build composite structures**

TALCIAN

Slide 21
Copyright © 1996 Talcian Corporation

Type declarations are made with the "type" statement in VHDL. The statement can declare a particular identifier (name) to represent a type, and additionally define a set of values for that type.

The values may be brand new in the case of an enumerated type or consist of a constrained range of one of the existing built in types like REAL or INTEGER.

Composite structures in VHDL are either ARRAYS or RECORDS. The former consisting of a homogeneous set of elements while the later is heterogeneous.

We see some type examples on the next few slides.

Type Examples

```
TYPE float IS RECORD
    sign : std_logic;
    exponent : integer RANGE -16 TO 15;
    -- implicit 1. here
    mantissa : integer RANGE 0 TO (2**19) - 1;
END RECORD float;
```

```
TYPE float IS std_logic_vector(24 DOWNTO 0);
```

TALCIAN

Slide 22
Copyright © 1996 Talcian Corporation

The record definition shows three fields. A sign indication for the number as a whole, the signed exponent and the fractional portion of the number.

Ultimately, in hardware, a signal of this type will be implemented as a 25 bit bus. The person who assembles this particular specification of a floating point number benefits from the use of the built in type integer. It will simulated quicker then a bit representation and yet is also synthesizable. Later, if necessary, the exponent could be changed to a vector representation if a synthesis tool didn't support integers for example. The user of the type would be insulated from that, however.

Sub-Programs

- **Functions, Procedures**
  - Algorithmic processing
  - Hides specific details of data representation
  - Full power of language available
- **Overload functions for math operators**
  - Arithmetic expressions easy to read
  - Eliminates type specific names

TALCIAN

Slide 23
Copyright © 1996 Talcian Corporation

Type Checking

- **19 bit integer distinguished from 19 bit real**
- **Strong typing required for overloading**
- **Increased readability**
  - Type name expresses object values/range
  - Makes data management explicit

TALCIAN

Slide 25
Copyright © 1996 Talcian Corporation

There are two ways in VHDL to represent sub-algorithms. Functions return single values. Procedures can return multiple values. Both perform algorithmic processing of some type. Sub-programs are the interface to the specific implementation of a given type. It is here that a multiply algorithm for "float" type numbers knows whether to act on individual bits of a bit vector or on the fields of a record.

VHDL overloading is a means for accumulating several function definitions under the same name. The most common use is to overload the names of the built-in math operations so that expressions with abstract data types can look just like expressions with built-in types.

Type checking permits the compiler to correctly identify objects (signals, variables, constants) which seem to have the same external structure but that properly represent different "abstract" things.

Overloading relies on the type names to decide which of multiple sub-programs to select for a particular operation. (This makes sense right -- overloading permits multiple sub-programs with the same name, it is only the operand "types" that distinguishes them)

By declaring two different type names with the same definition the user explicitly instructs the compiler (and the reader) of their different purposes and meaning. This directly increases the amount of information in the description and hence the readability.

Sub-Program Example

```
FUNCTION " * " (left, right : float) RETURN float IS
   VARIABLE result : float;
BEGIN
   result.exponent := left.exponent + right.exponent;
   result.mantissa := left.mantissa * right.mantissa;
   result.sign := left.sign XOR right.sign;
   normalize(result);
   RETURN result;
END FUNCTION " * " ;
```

TALCIAN

Slide 24
Copyright © 1996 Talcian Corporation

Here, the built-in multiply operation is overloaded to work on the float type we saw a few slides earlier.

In many ways the representation of the type values is specified by the algorithmic steps found in the sub-program.

Packages

- **Shared, common declarations**
- **ADT source librarian**
- **Separate Compilation possible**
  - Package header contains declarations
  - Package body contains definitions
  - Multiple bodies allowed

TALCIAN

Slide 26
Copyright © 1996 Talcian Corporation

Packages are VHDL design units that permit the sharing of common declarations. For this reason they make ideal receptacles for abstract data types.

One person can generally take responsibility for managing the ADT source. Since the package can be in a separate source file, the user of the ADT's and the librarian won't be in conflict over editing a single common source.

The separate compilation capability of the package header and package body insulates the design unit which references the package contents from frequent compilations.

By declaring names in the header and definitions for those names in the body we can have multiple bodies -- each containing different implementations of our abstract data types. Multiple implementations are used in the graphics pipe description to trade off simulation time versus synthesizability.

Package Example

```
PACKAGE example IS
    TYPE float;
END PACKAGE example
```

```
PACKAGE BODY example IS
    TYPE float IS real;
END PACKAGE BODY example;
```

```
PACKAGE BODY example IS
    TYPE float IS std_logic_vector(24 DOWNTO 0);
END PACKAGE BODY example;
```

TALCIAN

Slide 27
Copyright © 1996 Talcian Corporation

The one on the left doesn't synthesize but simulates quickly. The opposite is true for the body on the right.

ADT's in Our Design

- **Abstractions**
- **Design Unit Structure**
- **RTL Description**
- **Some Subtleties**

TALCIAN

Slide 28
Copyright © 1996 Talcian Corporation

In this sub-section we cover:

- The information that we hid in the pipeline description,

- How the various VHDL design units for the render engine were associated,

- Actual VHDL illustrating how the ADT's were used,

- Some subtle issues with the use of ADT's in our design.

Slide #29

Abstraction

- **Data type Synthesizability hidden**
  - Used built-in types (reals, integers)
  - Used Std_logic_vector
- **Numeric range/precision hidden**
- **Class of data not hidden**
  - Color
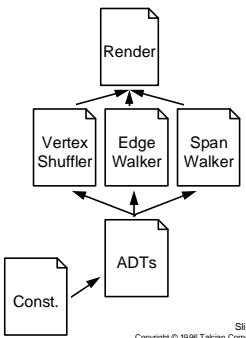 - Texture coordinate
  - Translucency, etc...

TALCIAN

The primary bit of information hidden from the architect of the Render module was the implementation of synthesized vs. non-synthesized data types. All the architect worried about was how to implement the mathematics of 3-D graphics -- not how floating point numbers were represented or manipulated.

This allowed us to simulate much earlier then we could have otherwise. Many problems relating to visual artifacts present in various test images were eliminated quickly by simulating with "real" based ADT's. Later, on when the synthesizable ADT's were finished, the slower simulation speed was tolerable because there was much less debugging to do.

Slide #30

Design Unit Structure

- **ADT's in single pkg**
- **Constants used to define numerical precision**
- **Type for each distinct floating point, integer and fixed point number required**

TALCIAN

There were dozens of number types in the design description. Some of them were local to the render ASIC and some shared by the other ASICs as well.

Next, we will look at how the RTL description for the architecture utilized the ADT concept.

Slide #31

RTL Description

```
ENTITY span_walker IS
  PORT MAP( red_left, red_right : IN color;
            X_left, X_right : IN coordinate;
            red_attribute : OUT color);
END ENTITY span_walker;

ARCHITECTURE rtl OF span_walker IS
  -- local signal declarations
BEGIN
  walk_span_fsm: PROCESS (Clock, Reset)
  BEGIN ... END;
  -- Calculation Setup
  new_red: PROCESS (Clock)
  BEGIN ... END;
END ARCHITECTURE rtl;
```

TALCIAN

Register transfer level (RTL) is frequently used synonymously with "synthesizable". Behavioral is frequently taken to mean "non-synthesizable". Well, the render ASIC was certainly described using an RTL style but with the ADT package implemented with real values the description was anything but synthesizable!

The slide illustrates how the span-walker entity ports use the ADT's. No mention is made of "std_logic_vector" or other synthesizable types. Hence, from this point of few the reader can't tell if the model is synthesizable or not.

The architecture will be described over the next several slides. We will skip the state machine, it isn't directly relevant to our discussion of the ADTs.

Slide #32

Local Signal Declarations

```
-- local signal declarations
SIGNAL dx : delta_coordinate;
SIGNAL dred : delta_color;
SIGNAL dred_over_dx : color_slope;
SIGNAL reciprocal : one_over_delta_coordinate;
SIGNAL enable_attribute : boolean;
SIGNAL set_initial_attribute : boolean;
```

Here, just like with the entity ports, local signal objects are declared to be of user defined types. Take the "dx" signal for example. The user gets more information from the type name "delta_coordinate" than from std_logic_vector(11 downto 0).

Yes, this could be covered by a comment but comments are rarely maintained if they are written at all. With good type names the writer is forced to convey useful information from the very start!

Slide #33

Calculation Setup

```
-- Calculation Setup

dx <= x_right - x_left;
reciprocal <= 1 / dx;
dred <= red_right - red_left;
dx_over_dred <= dred * reciprocal;
```

Notice how easy this is to read? The computation to be performed by the span walker is directly evident since we are very comfortable with reading arithmetic expressions. Procedure or function calls certainly could express the same concepts but the direct overloading of the math operators increases readability significantly.

Pure modular design would not be quite as readable though the functionality would almost certainly come out to be the same.

Slide #34

New Red Process

```
new_red: PROCESS (Clock)
BEGIN
  IF rising_edge(Clock) THEN
    IF set_initial_attribute THEN
      red_attribute <= red_left;
    ELSIF enable_attribute THEN
      red_attribute <= red_attribute + dred_over_dx;
    END IF;
  END IF;
END PROCESS new_red;
```

Notice the synthesizable template for a clocked process is unchanged by the use of abstract data types.

Slide #35

Some Subtleties

- **Number Precision**
  - Reals more precise then vectors
  - Expected some visual differences
- **Signed vs Unsigned numbers**
  - Delta_color was signed
  - Color was unsigned
- **Can't pass a clock into a function**

One problem with the different implementations of the ADT's was that built-in real numbers are naturally of higher precision (32 bit) then the floating point number precision we required. As a result we saw some virtually imperceptible differences in the images we tested the render ASIC on.

In some cases the abstract types hid information that might be potentially useful for the reader to know. Delta values for color could be signed. Delta values for coordinates would never be signed because they X_left was always guaranteed to be smaller the X_right by virtue of the vertex shuffler operation. The meaning of "delta" was thus hidden in some cases. We covered this with comments in the definition of the ADT's for delta coordinate and in the vertex shuffler.

Slide #36

Benefits

- **Rapid Simulation**
- **Concurrent Development**
- **Single Architect/Designer**
- **Other Benefits**

Slide #37

Rapid Simulation

- **Render 512 x 512 frame**
  - REAL ADT's ~ 20 minutes
  - STD_LOGIC_VECTOR ADT's ~ 2 1/2 hours

Management desired images simulated by the VHDL models. We did better than that —we ran 30-frame animations.

The animations showed all sorts of visual effects which weren't apparent in still images. Edge conditions were found to which the eye was very sensitive.

The fast simulation times and hence the ability to generate animations quickly, allowed us to improve the product quality.

Slide #38

Concurrent Development

- **ADT package**
  - Created in about 2 weeks
  - ADT's implemented with VHDL REALs
- **Render**
  - Up and running in 3 weeks
  - Used "real" based ADT's
  - ~ 7 weeks extra simulation as a result
- **Synthesizable ADT's ~ 8 weeks**

Each chip had its own ADT package. The chip architect developed this package rapidly from the base equations for each chip. The initial packages were constructed with real number representations. The use of real numbers reduced simulation time.

After analysis of the bit precision specialist were used to implement the synthesizable math functions. These specialist were concerned only with the implementation of the math, not with how the math was to be used.

Slide #39

Design Task Division

- **Single Architect/Designer Per Chip**
- **Task Boundary Contrast**
  - Specialist implemented synthesizable math operations
  - Architect focused on 3-D graphics process
  - Rapid design iterations possible

Traditional approaches to breaking down a chip is by module. Each module is given to an engineer who completes each level of abstraction down to the gates.

We chose a methodology of one designer per ASIC. The designer could focus on the problem and solution at hand without worrying about the lower levels of abstraction. A separate specialist handled the implementation of the math.

Single Architect/Designer

- **Reduced design communication overhead**
- **No functional boundaries to negotiate**
- **Easier to manage feature changes later**
- **Changes made with less consequence**
- **Early problem detection**
- **Minimal component structure - fewer files to edit**

TALCIAN

Slide 40
Copyright © 1996 Talcian Corporation

Task Boundary Contrast

By *Module*

System

ASIC A	ASIC B ■	ASIC C

Mod A ▲	Mod B ●	Mod D ■
	Mod C ●	

By *ADTs*

System

ASIC A	ASIC B ■	ASIC C

Mod A ■	Mod B ■	Mod D ■
	Mod C ■	

■ Architect ● Engineer 2 ▲ Engineer 3

TALCIAN

Slide 41
Copyright © 1996 Talcian Corporation

Notice the overhead of the traditional approach.

- Interface specification must be developed by the architect and communicated to the other engineers. Decreasing the time available for doing the important -- 3-D graphics architecture.

- With ADTs the architect works on what he does best.

Ultimately, this reduced the amount of explicit structure in our design. This made changes easier with out sacrificing our need for hierarchy for place and route and floor planning.

Other Benefits

- **Enhanced readability**
  - Type names add extra "descriptiveness"
  - Readable by non-VHDL literate people
  - Increased maintainability
  - Type implementations easy to document
- **Mentor Autologic I synthesized hierarchy for functions and block statements**
- **Intellectual property development facilitated**

TALCIAN

Slide 42
Copyright © 1996 Talcian Corporation
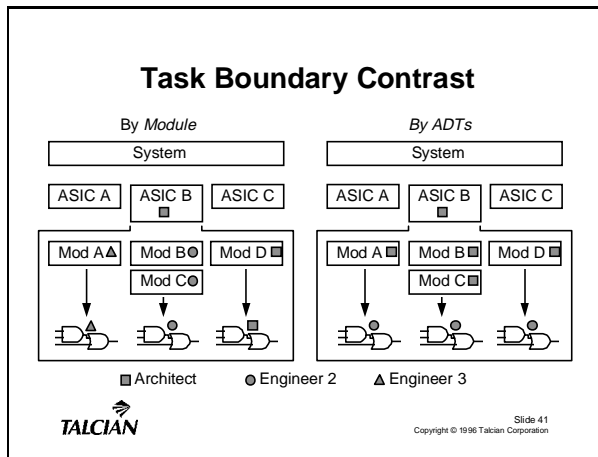
A mathematician was brought on board to precision analysis and queue analysis. With minimal training he was able to read the VHDL, understand its intent and suggest a modification. This modification gave the system ~ 15% performance gain.

Slide #43

Conclusion

- **Success**
- **Other Factors**
- **Motto**

Slide #44

Engineering Success!

- **First pass silicon success**
- **System operational in 2 weeks**
- **Render completed in ~11 months**

Three of the four chips completed and integrated into the PPB in less than 2 weeks. Management was amazed.

Render Engine ~292K gates completed in ~11 months with ~2 person average loading. Other chips were smaller and completed sooner with similar man loading.

One chip which didn't use this approach was never delivered due to factors beyond our control.

Slide #45

Success Factors Not Covered

- **Documentation Guidelines**
- **Configurations Management**
- **Verification methodology**
- **Management bought lunch.**

With sixteen engineers at the peak of the project, documentation guidelines were essential. Communication needed to be reduced or we would never have finished.

Sixteen engineers, four locations, four chips, three models per chip, two simulations systems, two synthesis systems; Yes, configuration management was necessary.

A verification strategy which supported still images, animations, multiple memory structures and a system wide view was necessary for the success of this project.

Slide #46

Motto

- **Make it work, *then* make it faster, smaller, cheaper,...**

Too often design teams focus on the details early. We propose that teams:

- Understand the problem

- Explore the implementations available

- Decompose into manageable blocks

- Verify

Recommended Resources

- **Equipment and software**
  - **Mentor Graphics**
    - **QuickVHDL for Simulation**
    - **Autologic I for Synthesis**
    - **LSI LCA300K for Silicon**
  - **Utah Raster Tool-kit - via the Internet**
  - **XV - the picture viewer also via the Internet**

TALCIAN

Slide 47
Copyright © 1996 Talcian Corporation

Recommended Resources (cont.)

- **Other Resources and Consultant Services**
  - **Tim Davis, Timothy Davis Consulting**
  - **Tom Bishop, Golden Systems Design**
  - **Idealogy Inc. - Boulder, Colorado**
  - **First Pass Inc. - Castle Rock, Colorado**
  - **Pinpoint Solutions - Boulder, Colorado**
  - **Silicon Arts - San Diego, California**
- **Mentor Graphics Hotline was <u>invaluable</u> in the completion of this project.**

TALCIAN

Slide 48
Copyright © 1996 Talcian Corporation