J.P. Gray

Rapporteurs: Dr. T. Anderson
Mr. S. Dlay

Abstracts

1. Structured Design and Unstructured Implementations

It is often stated that design in its broadest sense is implementation independent, i.e. the designer can use a set of abstractions in developing a design that allows the suppression of physical details of the implementation medium. However good engineering is also concerned with the elegant exploitation of the medium. In VLSI systems this medium is silicon. Its simplest abstraction is a two and a half dimensional interconnect/communications space. Historically, hardware design has favoured the view that structural design and physical design can be separated. This lecture explores this principle by describing a design methodology based on hierarchical structural design and unstructured physical design.

2. Structured Design and Structured Implementations

This lecture will cover the now almost classical design methodology based on hierarchical structural design and an isomorphic physical design. There will be emphasis on what cost functions are relevant in developing such a design, and how this methodology can be supported with software tools. An attempt will be made to compare characteristics of parts designed in this style against similar parts designed in the style described in lecture 1.

STRUCTURED DESIGN AND UNSTRUCTURED IMPLEMENTATIONS

1. Introduction

In the process of designing any piece of hardware in general, or a chip in particular there are usually three separate descriptions generated : a structural description of the architecture (block diagrams to logic diagrams), a physical description of the implementation (chip artwork definition) and a behavioural description (simulation model). As any of these descriptions may be hierarchical it follows that producing a design amounts to massaging some hierarchical structural description into a hierarchical (possibly one level) physical description. Verifying a design may amount to exercising a hierarchical behavioural description. As the integrity of the design is based on keeping these separate descriptions consistent it is important that any design style which helps in this respect is useful. Before going on to discuss one such style and how it helps, it is necessary both to clarify what is meant by "design style" and to make some comments on the integrated circuit design process, the representations used and the underlying descriptions they generate.

In what follows emphasis is placed on the structural and physical aspects of design and the temporal/performance aspects are not dealt with. This is not seen as placing the cart before the horse as it is not possible to discuss timing issues without some notions of architecture and implementation in the first place.

2. Representations and Descriptions

A superficial examination of the classical levels of design abstraction (Bell 1971) in hardware, from device through circuit to PMS level, can produce an impression of unnecessary complexity due to the number of levels and possible representations. Each level of abstraction seems well defined and one can generate a number of representations that provide a notation for capturing design data. A possible simplification is to note that design data can be separated into at most three classes : structural data, physical data and behavioural data. Structural data is simply the description of units/modules/components and their interconnection/intercommunication e.g. block diagrams, logic diagrams, program specifications. Structural design has a consistent meaning across a number of engineering disciplines. Physical data is the description of a physical implementation of a design e.g. the artwork definition of a chip. Behavioural data is the description of what the parts of this design do. It is usually a simulation model. At different levels of design abstraction there are different amounts of data, possibly none in these categories. Note that "designing" is usually taken to mean the production of a physical description from a structural description.

Representations may be thought of as notations that allow the capture of design data in one or more categories. Thus a Sticks representation allows both structural and physical design to be described at the lower levels of abstraction. A logic diagram representation is both a structural and behavioural description as gate symbols have "standard" semantics. A "good" representation may be one that gives maximum assistance in generating the three basic descriptions.

3. Design Style Taxonomy

It is possible to classify integrated circuit designs with respect to two features of their physical description : the type of interconnect and the variation in size of the basic units, cells, in the design. Interconnect may be thought of as coming in two flavours ; regular or random. Cells may be any size but certain design styles use cells of a standard size. Thus gate array exhibit regular cell size and random interconnect, while memory designs exhibit regular cell size and regular interconnect. Five design styles are thus classified in the figure below

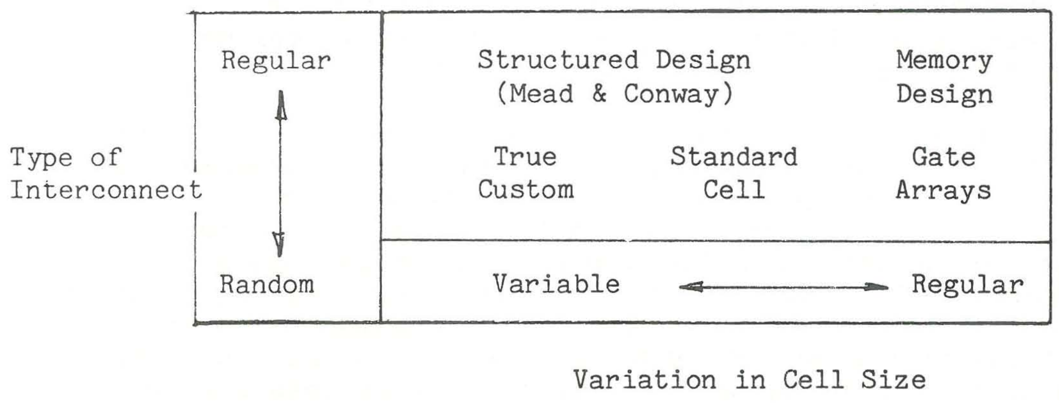

FIGURE 1 TAXONOMY

One can imagine other design metrics like functional density and design time as surfaces over this plane. The reader is encouraged to do this to convince himself/herself of the usefulness of the structured design style.

If the maturity of a design style is related to the effectiveness of tool support then there is no doubt that the gate array design style is the most mature. It is possible to work from a structural description to a physical description largely automatically by using placement and routing algorithms. Thus the gate array design style simplifies "designing" by constructing a physical description from a structural description. Other design styles "finesse" this problem in more elegant ways.

4. The Gate Array Design Style

In its simplest form a gate array can be envisaged as a two dimensional array of gates, of variable granularity, separated by wiring space. Thus a physical design can be achieved by reducing a structural design to a set of interconnected gates and mapping these onto the 2-D chip image. Two important factors must be taken into account. Firstly, as it is impossible to attach additional wires to a chip it is essential to get all the wires to fit into the given chip image. Secondly, designs may be large and complex but have to be reduced to a one level interconnection of primitive components (which will be larger).

To address the first of these problems it is possible to use wirability "theory" (Heller et al 1978) as a guide book in defining chip images. It is never possible, however, to guarantee the wirability of a specific design without incurring a large area cost. In fact designing in this style is concerned with managing a structural description so that its wiring demand is reduced beneath the wiring capacity of a particular image. Note that Heller's results show no particular limit to the size of gate array implementations, at least from a wiring point of view.

To address the second problem it is necessary to manage complexity by allowing hierarchical description. A convenient way to do this is to use a high level programming notation tailored to structural description. Figures 2, 3 and 4 show the top down refinement of an adder with full carry lookahead and Figure 5 shows a programmatic description of the same thing. Leaving aside any discussion of the design of this particular language, it is important to note the conciseness of the textual description and its formality. Because it has defined syntax and semantics it is possible to carry out extensive design debugging by compilation. Structure rules may be rigorously enforced. This, together with a physical design subsystem, allows the fully automatic design of gate arrays. It is of course sensible to exploit the designer's natural structuring of the design in layout algorithms.

5. Summary

By choosing a gate array design style it is possible to coerce hierarchical structural descriptions into one level physical descriptions. The model of programming notations and program debugging can be exploited to produce a new design environment for hardware design with real benefits in quicker and more correct designs.

References

C.G. Bell, A. Newell (1971). "Computer Structures: Readings and Examples", McGraw-Hill, 1971.

W.R. Heller, W.F. Mikhail, W.E. Donath (1978). Proceedings of 14th Design Automation Conference, pp. 32-43 (J. Design Automation and Fault-Tolerant Computing, Vol. 2, No. 2, pp. 117-144, 1978).

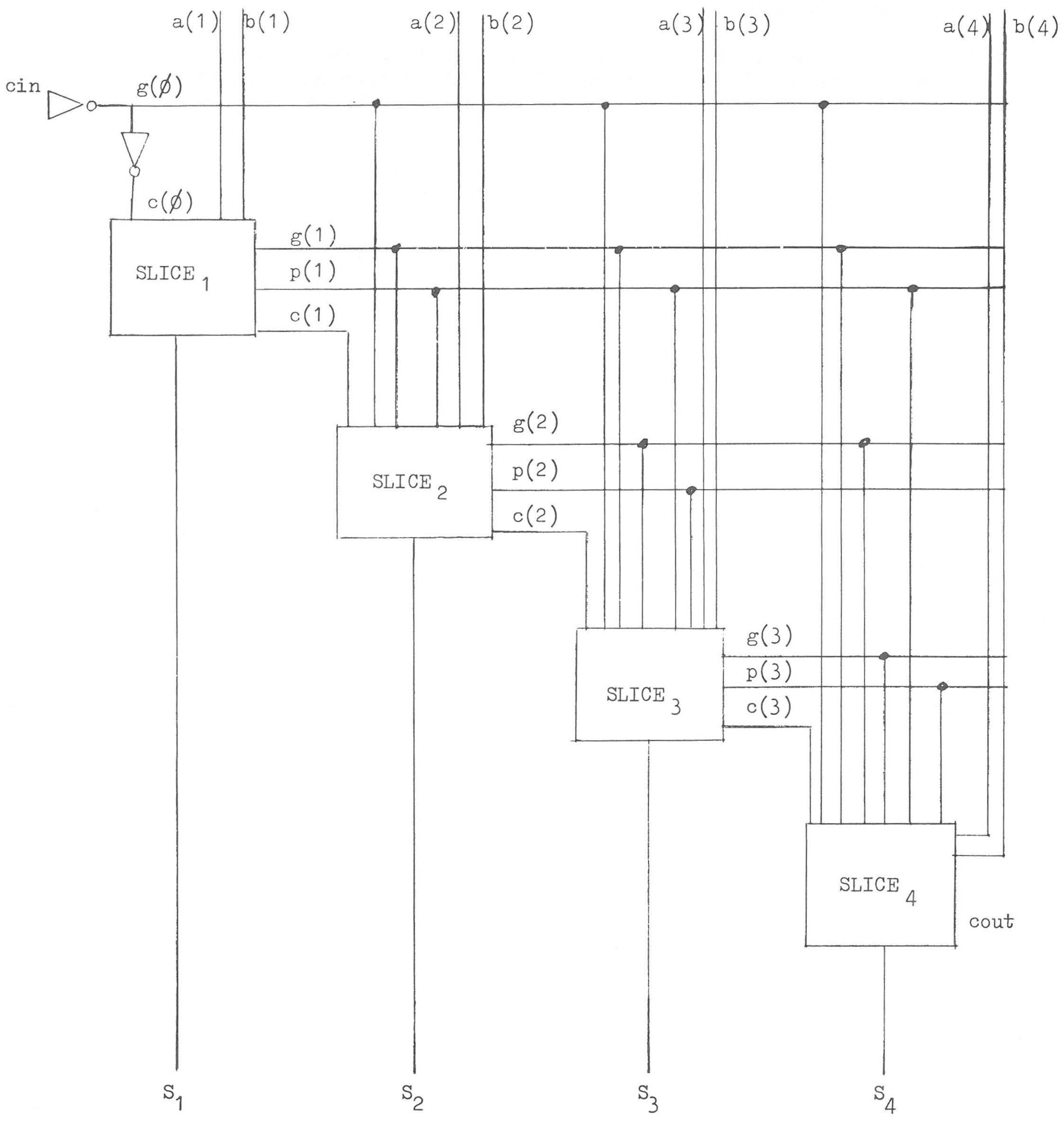

Figure 2 ADDER

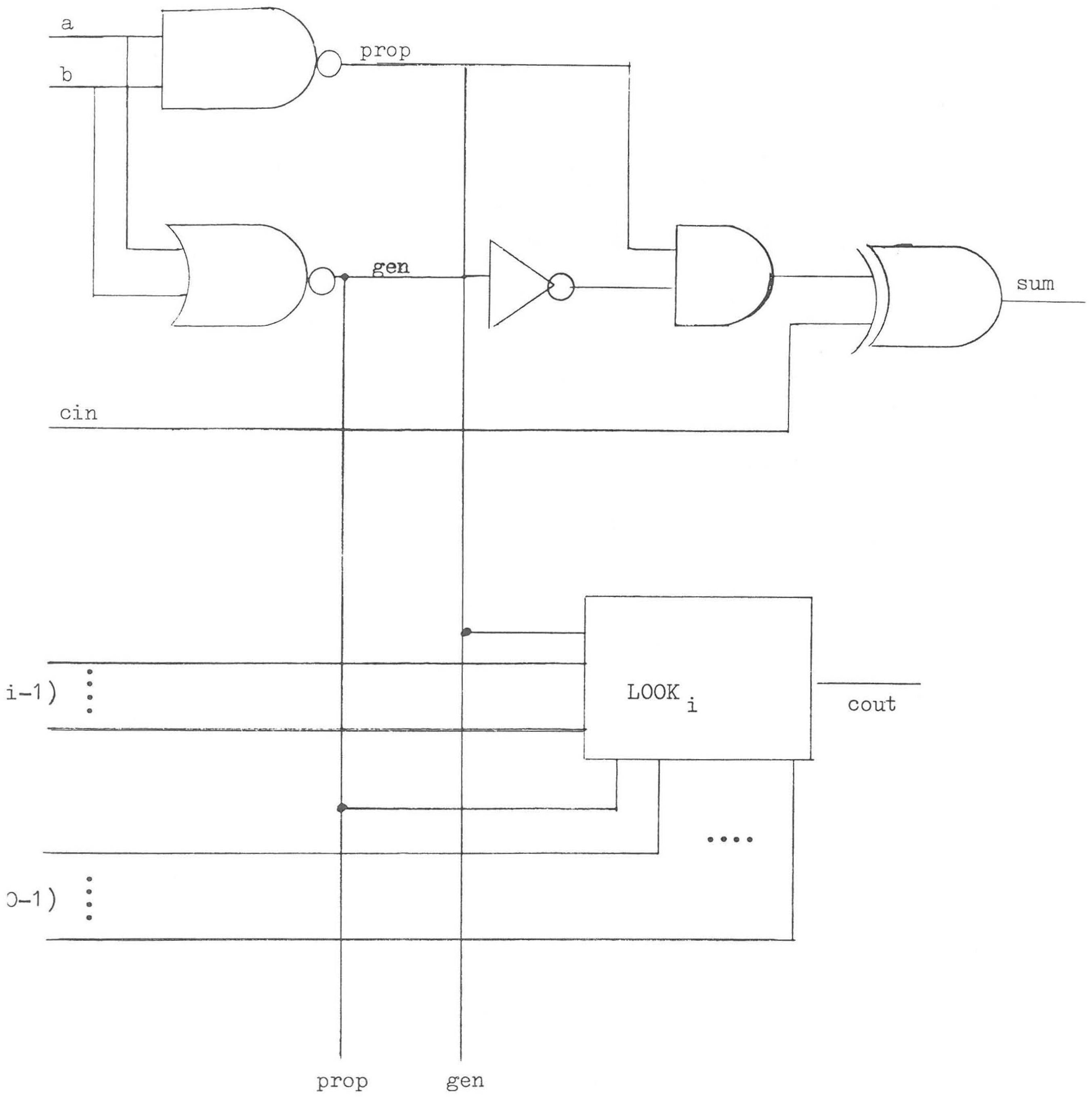

Figure 3 SLI$\acute{}$CE$_i$

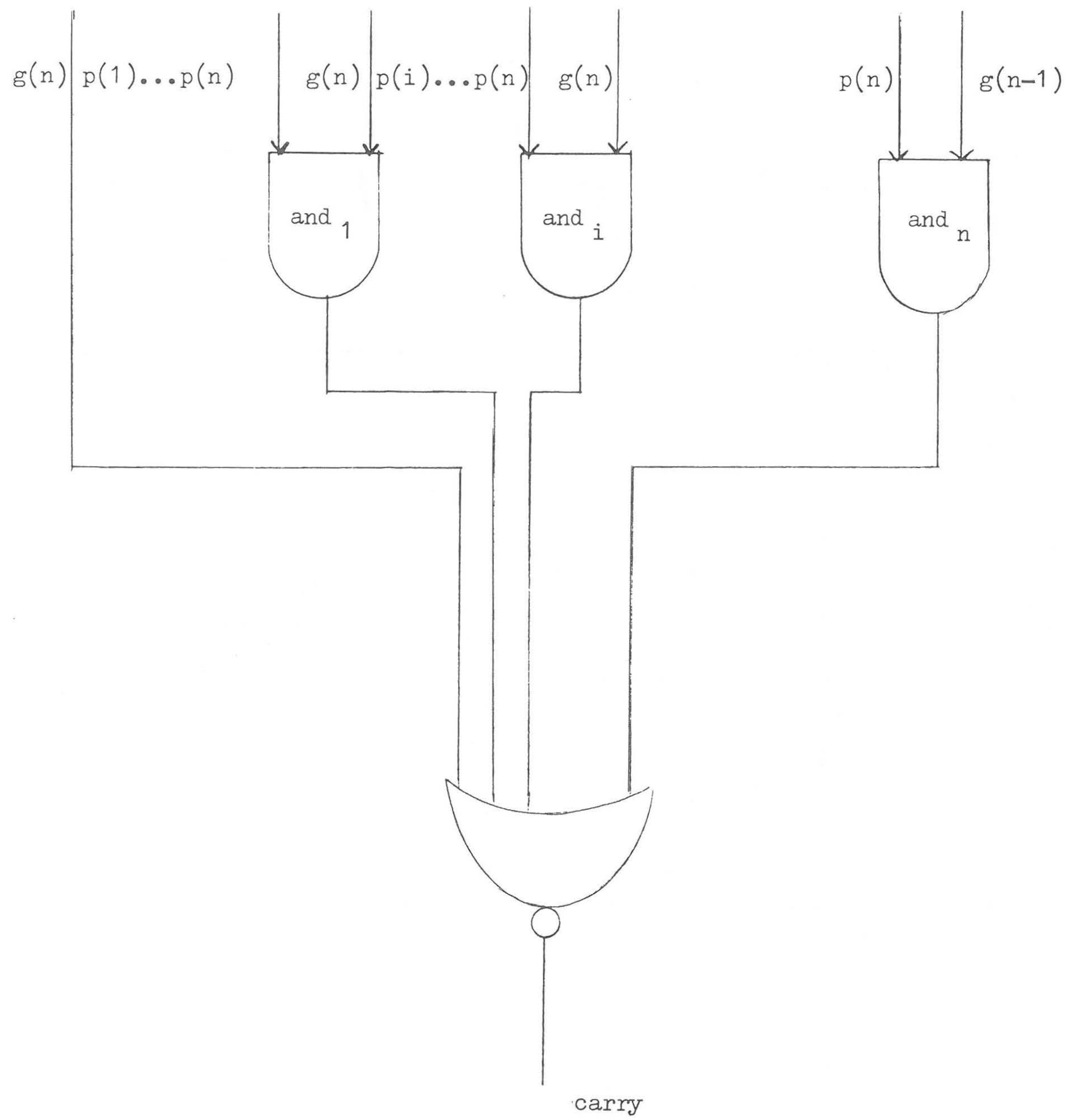

Figure 4 LOOK$_i$

```
{N-bit binary full adder with fast carry, similar to TI part SN74LS283}

Include "modlib:library.inc"

Constant bits=4

Input Pad a(1:bits),b(1:bits),cin

Output Pad s(1:bits),cout

Part look [n] (p(1:n),g(0:n)) --> carry
   Signal temp(1:n)
   Integer j
   For j=1:n Cycle
      and(p(j:n),s(j-1)) --> temp(j)
   Repeat
   nor(s(n),temp(1:n)) --> carry
End

Part slice [n] (a,b,p(1:n-1),g(0:n-1),cin) --> sum,cout,prop,sen
   nand(a,b) --> prop
   nor(a,b) --> gen
   xor(cin,and(prop,not(gen))) --> sum
   look [n ](p(1:n-1),prop,g(0:n-1),gen) --> cout
End

Part adder [n] (a(n:1 By -1),b(n:1 By -1),cin) --> cout,s(n:1 By -1)
   Integer j
   Signal c(0:bits),p(1:bits),g(0:bits)
   not(cin) --> g(0)
   not(g(0)) --> c(0)
   c(n) --> cout
   For j=1:n Cycle
      slice [j ](a(j),b(j),p(1:j-1),g(0:j-1)) --> s(j),c(j),p(j),g(j)
   Repeat
End

adder [bits] (a(bits:1 By -1),cin) --> cout,s(bits:1 By -1)

Endoffile
```

Figure 5 MODEL Description

DISCUSSION (1)

Professor Sequin enquired if a programming notation had any advantages over logic diagrams? **Dr. Gray** considered that his programming notation was more consisted and checkable. Some of this checking could be automated, and as a result designs could more easily be debugged. Professor Sequin observed that tools such as SCALD were available to represent and manipulate the relevant information in diagrammatic form. Such techniques could be economic in the near future. However, Dr. Gray claimed to have worked with similar tools for many years and now considered them to have been oversold. **Professor Hoare** challenged Professor Sequin to read and understand the details of a 3000 gate circuit diagram when presented on a CRT display. **Professor Coulouris** noted that another advantage of a programming notation is that iteration can be made explicit whereas in a large diagram repetition would be obscured.

Professor Lewin felt that the lecturer had been somewhat disparaging in his remarks on testing and simulation - surely these were essential in order to have confidence that a design would actually work as intended. **Dr. Gray** suggested that the programming language techniques of compiler diagnostics, debugging and program verification could all contribute to confirming the validity of a design expressed in a programming notation.

Professor Aspinall felt that a hardware description language should not be tied to a particular implementation technique such as gate arrays, since this limited the freedom of the designer. **Dr. Gray** approved of limitations on a designer's freedom and claimed that gate arrays were an excellent technique for exploiting silicon technology. Appropriate stress is placed on the problems of interconnecting modules rather than the modules themselves. Professor Aspinall suggested that the management of complexity was the vital issue, with which Dr. Gray agreed. The problems of interconnecting modules can be alleviated by a hierarchical appproach.

Professor Kung wondered if any experimental results on the structured design approach were available. **Dr. Gray** replied that the work of Heller et al (1978)was relevant here but that as larger designs were attempted design approaches were likely to change (for example, wiring metrics were of increasing importance). **Professor Kinniment** pointed out that when a structured approach is adopted in a large custom design the outcome can be very close to that resulting from the gate array approach.

DISCUSSION (2)

Professor Kung asked how old was the multiplier design, and at what speed did it operate? **Dr. Gray** thought that the chip had been designed in 1978.

Professor Randell wondered if routing algorithms should be eschewed in favour of designing so as to identify structural and physical descriptions. **Dr. Gray's** view was that routing algorithms often produce inelegant designs. Professor Randell suggested that obvious analogies could be drawn between board layout and text composition - perhaps Knuth's TEX system could be used to set up a floor plan for a chip.

Professor Whitfield expressed concern that tracks leading from one cell might have to be permuted before connecting to a destination cell. **Dr. Gray** asserted that cells were usually in the correct order and the only problem frequently encountered in making track connections was a pitch mismatch, and this was easily dealt with. **Professor Kinniment** observed that a PLA is a nice example of a cell in which input and outputs can be arbitrarily permuted.

Professor Rem asked if Dr. Gray was disparaging the top-down approach to design. **Dr. Gray** denied this and claimed he was attempting to link a top-down style with concern for the standard of wiring achieved. Even with the top-down approach there are always problems with wiring. **Professor Randell** felt that low-level wiring issues conflicted with the notion of having a large library of general purpose cells. **Dr. Gray** agreed, and pointed out that his approach meshed well with the structure provided by soft cells.

Professor Kung asked why Dr. Gray did not recommend the Cal-Tech approach based on an assortment of tools. **Dr. Gray** replied that the only method of eliminating errors was by checking.