# Structured Arithmetic Tiling of Integrated Circuits*

Tony M. Carter
University of Utah
Department of Computer Science
3190a Merrill Engineering Building
Salt Lake City, Utah 84112

## Abstract

Robertson's Theory of Decomposition and Structured Tiling (an IC design technique) are combined in a structured arithmetic circuit design method. This method, extended by a set of inverse operators and a set of multiply operators, is used with computer-aided design tools to automate the design of arithmetic circuits.

## 1 Introduction

Automated design for digital integrated circuits is currently well advanced in the areas of control logic and register-transfer logic, but arithmetic circuits are still designed in a relatively ad hoc manner. In particular, little has been done to unify the theory of arithmetic with techniques for integrated circuit design. Robertson's *Theory of Decomposition* [1,2] for binary addition and subtraction, when combined with what we have learned about integrated circuit design in pursuing the development of *Path-Programmable Logic* (PPL)[3], becomes a powerful method for specifying and automatically designing arithmetic structures. The design method is known as *Structured Arithmetic Tiling* (SAT)[4] and has been used to implement a Variable Precision Processor (VPP)[7]. The Theory of Decomposition can also be implemented using the gate-array or standard-cell approaches, but would then incur the costs of those methods relative to PPL[8].

## 2 Theory of Decomposition

The Theory of Decomposition is a circuit design method for addition and subtraction. It relies on the concepts of digit-sets (a set of contiguous numbers that must include zero), diminished cardinality ($\delta$ — the number of elements in a set minus one), and offset ($\omega$ — the distance from zero of the smallest element in the set). For example, the digit-set $\{\overline{1}, 0, 1\}$ has diminished cardinality $\delta = 2$ and offset $\omega = 1$. The most interesting digit-sets (from an implementational view) are the binary digit-sets $\mathbf{a}^0 = \{0, 1\}$ and $\mathbf{a}^1 = \{\overline{1}, 0\}$, and the ternary digit-sets $\mathbf{b}^0 = \{0, 1, 2\}$, $\mathbf{b}^1 = \{\overline{1}, 0, 1\}$ and $\mathbf{b}^2 = \{\overline{2}, \overline{1}, 0\}$. The theory's main concepts are:

1. Weighted sums of binary and ternary digit-sets implement higher-order digit-sets.
2. Set-equations represent arithmetic operations.
3. If ($\delta_{in} > \delta_{out}$) the equation is unrealizable.
4. If ($\delta_{in} = \delta_{out}$) the equation is decomposable.
5. If ($\delta_{in} < \delta_{out}$) the equation is not decomposable. Add *mythical inputs* (= 0) to make the set-equation decomposable by equalizing the $\delta$s and $\omega$s of the input and output expressions.
6. All operations that transform set-equations must keep $\delta_{in} = \delta_{out}$ and $\omega_{in} = \omega_{out}$.
7. Lose information as rapidly as possible.

A simple example will serve to illustrate this style of design. Let us represent the addition of two three-bit, unsigned binary numbers as a set equation. A three-bit unsigned binary number is represented as $4a^0+2a^0+a^0$. The result of adding two three-bit numbers is a four-bit number, $8a^0 + 4a^0 + 2a^0 + a^0$. Thus, the set-equation

$$8a^0+4a^0+2a^0+a^0 \Leftarrow (4a^0+2a^0+a^0)+(4a^0+2a^0+a^0).$$

represents the addition of two three-bit numbers. For the left-hand side of the equation $\delta_{out} = 8 \cdot 1 + 4 \cdot 1 + 2 \cdot 1 + 1 = 15$, and for the right-hand side $\delta_{in} = 14$. Obviously, $\omega = 0$ on both sides.

Since $\delta_{in} < \delta_{out}$, this equation is not decomposable, but we add a *mythical input* (which does not change the value of the result) of minimum information content to make it decomposable. Since $\Delta\delta = 1$ and $\Delta\omega = 0$ we

*This research was supported by DARPA through contract number DAAK11-84-K-0017.

Table 1: **Logic Implementations of the Operators**

| Name | Set Equation | Logic Implementation | | |
|---|---|---|---|---|
| HA0 | $b^0 \Leftarrow a_1^0 + a_2^0$ | $b_g^0 = a_1^0 \cdot a_2^0$ | $b_e^0 = a_1^0 \oplus a_2^0$ | |
| HA1 | $b^1 \Leftarrow a^1 + a^0$ | $b_g^1 = a^1$ | $b_e^1 = a^1 \oplus a^0$ | |
| HA2 | $b^2 \Leftarrow a_1^1 + a_2^1$ | $b_g^2 = a_1^1 \cdot a_2^1$ | $b_e^2 = a_1^1 \oplus a_2^1$ | |
| CG0 | $2a^0 + a_2^0 \Leftarrow b^0 + a_1^0$ | $2a^0 = b_g^0 + b_e^0 \cdot a_1^0$ | $a_2^0 = b_e^0 \oplus a_1^0$ | |
| CG1a | $2a^0 + a_2^1 \Leftarrow b^0 + a_1^1$ | $2a^0 = b_g^0 + b_e^0 \cdot a_1^1$ | $a_2^1 = b_e^0 \oplus a_1^1$ | |
| CG1b | $2a^0 + a^1 \Leftarrow b^1 + a^0$ | $2a^0 = \overline{b_e^1} \cdot a^0 + \overline{b_g^1} \cdot b_e^1$ | $a^1 = b_e^1 \oplus a^0$ | |
| CG2a | $2a^1 + a^0 \Leftarrow b^1 + a^1$ | $2a^1 = \overline{b_e^1} \cdot a^1 + b_g^1 \cdot b_e^1$ | $a^0 = b_e^1 \oplus a^1$ | |
| CG2b | $2a^1 + a_2^0 \Leftarrow b^2 + a_1^0$ | $2a^1 = b_g^2 + b_e^2 \cdot a_1^0$ | $a_2^0 = b_e^2 \oplus a_1^0$ | |
| CG3 | $2a^1 + a_2^1 \Leftarrow b^2 + a_1^1$ | $2a^1 = b_g^2 + b_e^2 \cdot a_1^1$ | $a_2^1 = b_e^2 \oplus a_1^1$ | |
| RCG0 | $2a^0 + b_3^0 \Leftarrow b_1^0 + b_2^0$ | $2a^0 = b_{g1}^0 + \overline{b_{g2}^0}$ | $b_{g3}^0 = b_{e1}^0 \cdot b_{e2}^0 + b_{g1}^0 \cdot b_{g2}^0$ | $b_{e3}^0 = b_{e1}^0 \oplus b_{e2}^0$ |
| RCG1 | $2a^0 + b_2^1 \Leftarrow b^0 + b_1^1$ | $2a^0 = b_g^0 + \overline{b_{g1}^1} \cdot b_e^0 \cdot b_{e1}^1$ | $b_{g2}^1 = b_{g1}^1 \cdot b_{e1}^1$ | $b_{e2}^1 = b_e^0 \oplus b_{e1}^1$ |
| RCG2a | $2a^1 + b^0 \Leftarrow b_1^1 + b_2^1$ | $2a^1 = b_{g1}^1 \cdot b_{e1}^1 + b_{g2}^1 \cdot b_{e2}^1 \cdot \overline{b_{e1}^1}$ | $b_g^0 = b_{e1}^1 \cdot \overline{b_{g2}^1} \cdot b_{e2}^1$ | $b_e^0 = b_{e1}^1 \oplus b_{e2}^1$ |
| RCG2b | $2a^0 + b^2 \Leftarrow b_1^1 + b_2^1$ | $2a^0 = \overline{b_{g1}^1} \cdot b_{e1}^1 + \overline{b_{g2}^1} \cdot b_{e2}^1 \cdot \overline{b_{e1}^1}$ | $b_g^2 = b_{g2}^1 \cdot b_{e2}^1 \cdot \overline{b_{e1}^1}$ | $b_e^2 = b_{e1}^1 \oplus b_{e2}^1$ |
| RCG2c | $2a^1 + b_2^0 \Leftarrow b_1^0 + b^2$ | $2a^1 = b_g^2 + \overline{b_{g1}^0} \cdot \overline{b_{e1}^0} \cdot b_e^2$ | $b_{g2}^0 = b_{g1}^0 \cdot \overline{b_e^2}$ | $b_{e2}^0 = b_{e1}^0 \oplus b_e^2$ |
| RCG2d | $2a^0 + b_2^2 \Leftarrow b^0 + b_1^2$ | $2a^0 = b_g^0 + b_e^0 \cdot \overline{b_{g1}^2} \cdot b_{e1}^2$ | $b_{g2}^2 = b_g^2 \cdot \overline{b_e^0}$ | $b_{e2}^2 = b_{e1}^0 \oplus b_e^2$ |
| RCG3 | $2a^1 + b_2^1 \Leftarrow b^2 + b_1^1$ | $2a^1 = b_g^2 + b_{g1}^1 \cdot b_{e1}^1$ | $b_{g2}^1 = b_e^2 + b_g^2 \cdot b_{g1}^1$ | $b_{e2}^1 = b_e^2 \oplus b_{e1}^1$ |
| RCG4 | $2a^1 + b_3^2 \Leftarrow b_1^2 + b_2^2$ | $2a^1 = b_{g1}^2 + b_{g2}^2$ | $b_{g3}^2 = b_{g1}^2 \cdot b_{g2}^2 + b_{e1}^2 \cdot b_{e2}^2$ | $b_{e3}^2 = b_{e1}^2 \oplus b_{e2}^2$ |

must choose a single $a^0$ of weight 1. If we view the $8a^0$ term of the left-hand side of the equation as a carry-out, we note that there is no corresponding carry-in and we can rewrite the set-equation as

$$8a^0 + 4a^0 + 2a^0 + a^0 \Leftarrow$$
$$(4a^0 + 2a^0 + a^0) + (4a^0 + 2a^0 + a^0) + a^0$$

where the last $a^0$ can be mythical, or can serve as a carry-in! Note that by so doing we have equalized the $\delta$s (the $\omega$s were already equal) and thereby rendered the equation *decomposable*. There is a set of seventeen operators that can be applied to set-equations, transforming the input expression into the output expression, to implement any addition/subtraction operation: three generalized half-adders (denoted HA$n$) take two binary digit-sets and produce a ternary digit-set of the same weight, six carry-generators (denoted CG$n$) take a binary digit-set and a ternary digit-set and produce a binary digit-set of the same weight and a binary digit-set at twice the weight, and eight redundant carry-generators (denoted RCG$n$) take two ternary digit-sets and produce a ternary digit-set of the same weight and a binary digit-set at twice the weight. Carry-generators and redundant carry-generators serve as information losing devices and half-adders reconfigure an equation so that information may be lost. Table 1 contains the (previously published) logic designs for the seventeen operators. Note that some of the logic designs are identical; the seventeen logical operators map onto only eleven physical implementations. In the logic equations, ternary digit-sets are implemented as two bits denoted by subscripts $_g$ (Robertson's Greek variables) and $_e$ (Robertson's English variables). The superscripts represent the digit-set offset. The sub-subscripts are enumerative and serve to disambiguate logic variables. Therefore, $b^0$ would be implemented as $b_g^0$ and $b_e^0$.

## 2.1 Inverse Operators

In the design of the VPP, Robertson saw the need to change a $d^2$ digit-set into the sum of two $b^1$ digit-sets to simplify the design of the multiplier, and to eliminate all outputs of weight 8 in the multiplier-level. This resulted in the design of one inverse redundant-carry-generator and the concept of *fanout* circuits that only modify circuit interconnection without adding any hardware. The fanout module generates an input to a half-adder, and this combination performs exactly the function of an inverse carry-generator. In fact, each of the operators in the Theory of Decomposition has an inverse. The designs for the inverse half-adders are from Robertson (who calls them converters) and are included in table 2 with the rest of the inverse operators.

## 2.2 Non-Adding Operators

In the design of Chow's Variable Precision Processor [5], Robertson notes the need for two additional computational modules that are not strictly part of the

Table 2: Logic Implementations of the Inverse Operators

| Name | Set Equation | Logic Implementation | | | |
|---|---|---|---|---|---|
| IHA0 | $a_1^0 + a_2^0 \Leftarrow b^0$ | $a_1^0 = b_g^0$ | $a_2^0 = b_g^0 + b_e^0$ | | |
| IHA1 | $a^0 + a^1 \Leftarrow b^1$ | $a^1 = b_g^1$ | $a^0 = b_g^1 \oplus b_e^1$ | | |
| IHA2 | $a_1^1 + a_2^1 \Leftarrow b^2$ | $a_1^1 = b_g^2$ | $a_2^1 = b_g^2 + b_e^2$ | | |
| ICG0 | $b^0 + a_1^0 \Leftarrow 2a^0 + a_2^0$ | $b_g^0 = 2a^0$ | $b_e^0 = 0$ | $a_1^0 = a_2^0$ | |
| ICG1a | $b^0 + a_1^1 \Leftarrow 2a^0 + a_2^1$ | $b_g^0 = 2a^0$ | $b_e^0 = 0$ | $a_1^1 = a_2^1$ | |
| ICG1b | $b^1 + a^0 \Leftarrow 2a^0 + a^1$ | $b_g^1 = \overline{2a^0}$ | $b_e^1 = 1$ | $a^0 = \overline{a^1}$ | |
| ICG2a | $b^1 + a^1 \Leftarrow 2a^1 + a^0$ | $b_g^1 = 2a^1$ | $b_e^1 = 1$ | $a^1 = \overline{a^0}$ | |
| ICG2b | $b^2 + a_1^0 \Leftarrow 2a^1 + a_2^0$ | $b_g^2 = 2a^1$ | $b_e^2 = 0$ | $a_1^0 = a_2^0$ | |
| ICG3 | $b^2 + a_1^1 \Leftarrow 2a^1 + a_2^1$ | $b_g^2 = 2a^1$ | $b_e^2 = 0$ | $a_1^1 = a_2^1$ | |
| IRCG0 | $b_1^0 + b_2^0 \Leftarrow 2a^0 + b_3^0$ | $b_{g1}^0 = b_{g3}^0$ | $b_{e1}^0 = b_{e3}^0$ | $b_{g2}^0 = 2a^0$ | $b_{e2}^0 = 0$ |
| IRCG1 | $b^0 + b_1^1 \Leftarrow 2a^0 + b_2^1$ | $b_g^0 = \overline{b_{g2}^1} \cdot b_{e2}^1$ | $b_e^0 = \overline{b_{e2}^1}$ | $b_{g1}^1 = \overline{2a^0}$ | $b_{e1}^1 = 1$ |
| IRCG2a | $b_1^1 + b_2^1 \Leftarrow 2a^1 + b^0$ | $b_{g1}^1 = b_g^0$ | $b_{e1}^1 = \overline{b_e^0}$ | $b_{g2}^1 = 2a^1$ | $b_{e2}^1 = 1$ |
| IRCG2b | $b_1^1 + b_2^1 \Leftarrow 2a^0 + b^2$ | $b_{g1}^1 = b_g^2$ | $b_{e1}^1 = \overline{b_e^2}$ | $b_{g2}^1 = \overline{2a^0}$ | $b_{e2}^1 = 1$ |
| IRCG2c | $b_1^0 + b^2 \Leftarrow 2a^1 + b_2^0$ | $b_{g1}^0 = b_{g2}^0$ | $b_{e1}^0 = b_{e2}^0$ | $b_g^2 = 2a^1$ | $b_e^2 = 0$ |
| IRCG2d | $b^0 + b_1^2 \Leftarrow 2a^0 + b_2^2$ | $b_g^0 = \overline{b_{g2}^2} \cdot b_{e2}^2$ | $b_e^0 = b_{e2}^2$ | $b_{g1}^2 = \overline{2a^0}$ | $b_{e1}^2 = 0$ |
| IRCG3 | $b^2 + b_1^1 \Leftarrow 2a^1 + b_2^1$ | $b_g^2 = b_{g2}^1 \cdot b_{e2}^1$ | $b_e^2 = b_{e2}^1$ | $b_{g1}^1 = 2a^1$ | $b_{e1}^1 = 0$ |
| IRCG4 | $b_1^2 + b_2^2 \Leftarrow 2a^1 + b_3^2$ | $b_{g1}^2 = b_{g3}^2$ | $b_{e1}^2 = b_{e3}^2$ | $b_{g2}^2 = 2a^1$ | $b_{e2}^2 = 0$ |

method[6]: a *conditional complementer* (for subtraction by addition of the complement) and an *elementary multiplier*. Conditional complementers require symmetric digit-sets as inputs and thus are restricted to digit-sets whose diminished cardinality is even. There is a whole amily of elementary multipliers that may involve higher order digit-sets. Indeed, in the Variable Precision Processor we use a $\mathbf{b^1}$ by $\mathbf{d^2}$ multiplier. The logic designs for the first two conditional complementers and the basic set of binary and ternary elementary multipliers, as well as the $\mathbf{b^1}$ by $\mathbf{d^2}$ multiplier are presented in table 3. It should be noted that the complexity of coupled don't-cares in higher-order operators renders the task of designing a minimal logic implementation very difficult, and that layout considerations occasionally warrant using a non-minimal logic implementation.

## 2.3 Algorithms for Decomposition

We are currently implementing a CAD system that transforms a set of adder specifications (set-equations) into a silicon implementation. This object-oriented system, which is based on operator (such as IIA0), digit-set-term (a weighted digit-set), adder-block (defined by a set-equation), and information-loss-table objects, can check the validity of a set-equation, decompose adder-blocks (backtracking if necessary by using inverse operators), generate a logic-level emulation program, and generate input to simulators such as MOSSIM or SPICE. The hard part remains: to automatically place and interconnect the circuit modules that implement the desired arithmetic function.

In the specifications for adder units, input and output set-expressions must represent digit-sets. Robertson's method for verifying set-expressions (based on the radix and diminished-cardinality of successive terms in the set-expression) does not detect all set-expressions that represent digit-sets. Sometimes it is necessary to do the set additions indicated by the set-expression and check the resulting set to see if it includes all integers between its least and most significant elements.

Inverse operators are not necessary for decomposition to minimum information content when redundant carry-generators are used, but are used when the desired output set-expression does not have minimum information content. In the cases we have studied, the inclusion of this extra operator incurs no performance penalty because it does not change the number of gates in the worst-case delay path. Redundant carry-generators are necessary in the decomposition of some modules (such as the one discussed later in this paper) if inverse operators are not used.

The order of operator applications must be considered since, in some cases, it can cause decomposition to succeed in achieving minimum information content while not matching the desired output set-expression. In general, there usually is not a single unique decomposition that results in minimum information content, but some operator orderings will result in a shorter worst-case delay path from input to output. One can achieve

Table 3: **Logic Implementations of the Non-Adding Operators**

| Name | Set Equation | Logic Implementation | | |
|---|---|---|---|---|
| CC2 (on $\sigma$) | $b_2^1 \Leftarrow \pm b_1^1$ | $b_{g2}^1 = b_{g1}^1 \oplus \sigma$ | $b_{e2}^1 = \overline{b_{e1}^1}$ | |
| CC4 (on $\sigma$) | $2a_2^1 + b_2^0 \Leftarrow \pm(2a_1^1 + b_1^0)$ | $2a_2^1 = \sigma \oplus 2a_1^1$ | $b_{g2}^0 = \overline{b_{e1}^0} \cdot (\sigma \oplus b_{g1}^0)$ | $b_{e2}^0 = b_{e1}^0$ |
| MPY1.0a | $a_3^0 \Leftarrow a_1^0 * a_2^0$ | $a_3^0 = a_2^0 \cdot a_1^0$ | | |
| MPY1.0b | $a^0 \Leftarrow a_1^1 * a_2^1$ | $a^0 = a_1^1 \cdot a_2^1$ | | |
| MPY1.1 | $a_2^1 \Leftarrow a_1^1 * a^0$ | $a_2^1 = a_1^1 \cdot a^0$ | | |
| MPY2.0a | $b^0 \Leftarrow b^0 * a^0$ | $b_g^0 = a^0 \cdot b_g^0$ | $b_e^0 = a^0 \cdot b_e^0$ | |
| MPY2.0b | $b^0 \Leftarrow b^2 * a^1$ | $b_g^0 = a^1 \cdot b_g^2$ | $b_e^0 = a^1 \cdot b_e^2$ | |
| MPY2.1a | $b_2^1 \Leftarrow b_1^1 * a^0$ | $b_{g2}^1 = \overline{b_{g1}^1}$ | $b_{e2}^1 = a^0 \cdot b_{e1}^1$ | |
| MPY2.1b | $b_2^1 \Leftarrow b_1^1 * a^1$ | $b_{g2}^1 = b_{g1}^1$ | $b_{e2}^1 = a^1 \cdot b_{e1}^1$ | |
| MPY2.1c | $b_3^1 \Leftarrow b_2^1 * b_1^1$ | $b_{g3}^1 = b_{g1}^1 \oplus b_{g2}^1$ | $b_{e3}^1 = b_{e1}^1 \cdot b_{e2}^1$ | |
| MPY2.2a | $b_2^2 \Leftarrow b_1^2 * a^0$ | $b_{g2}^2 = a^0 \cdot b_{g1}^2$ | $b_{e2}^2 = a^0 \cdot b_{e1}^2$ | |
| MPY2.2b | $b^2 \Leftarrow b^0 * a^1$ | $b_g^2 = a^1 \cdot b_g^0$ | $b_e^2 = a^1 \cdot b_e^0$ | |
| MPY4.0a | $2a^0 + b_3^0 \Leftarrow b_1^0 * b_2^0$ | $2a^0 = b_{g1}^0 \cdot b_{g2}^0 + b_{g1}^0 \cdot b_{e2}^0$ | $b_{g3}^0 = b_{g1}^0 \cdot b_{g2}^0 + b_{g2}^0 \cdot b_{e1}^0$ | $b_{e3}^0 = b_{e1}^0 \cdot b_{e2}^0$ |
| MPY4.0b | $2a^0 + b^0 \Leftarrow b_1^2 * b_2^2$ | $2a^0 = b_{g1}^2 \cdot b_{g2}^2 + b_{g1}^2 \cdot b_{e2}^2$ | $b_g^0 = b_{g1}^2 \cdot b_{g2}^2 + b_{g2}^2 \cdot b_{e1}^2$ | $b_e^0 = b_{e1}^2 \cdot b_{e2}^2$ |
| MPY4.4a | $2a^1 + b_2^2 \Leftarrow b^0 * b_1^2$ | $2a^1 = b_g^0 \cdot b_g^2 + b_g^0 \cdot b_e^2$ | $b_{g3}^2 = \overline{b_g^0} \cdot b_g^2 + b_g^2 \cdot b_e^0$ | $b_{e3}^2 = b_e^0 \cdot b_e^2$ |
| MPY4.2a | $2a_2^1 + b_2^0 \Leftarrow b^1 * (2a_1^1 + b_1^0)$ | $2a_2^1 = b_e^1 \cdot (2a_1^1 \oplus b_g^1)$ | $b_{g2}^0 = \overline{b_{e1}^0} \cdot b_e^1 \cdot (b_{g1}^0 \oplus b_g^1)$ | $b_{e2}^0 = b_e^1 \cdot b_{e1}^0$ |
| MPY4.2b | $2a^1 + b_2^0 \Leftarrow b^1 * b_1^0$ | $2a^1 = \overline{b_g^1}$ $b_{g2}^0 = \overline{b_g^1} \cdot \overline{b_e^1} + \overline{b_{g1}^0} \cdot \overline{b_{e1}^0} \cdot b_g^1 + b_{g1}^0 \cdot \overline{b_g^1} \cdot b_e^1$ | | $b_{e2}^0 = b_{e1}^0 \cdot b_e^1$ |
| MPY4.2c | $2a^1 + b^0 \Leftarrow b^1 * b^2$ | $2a^1 = \overline{b_g^1}$ $b_g^0 = \overline{b_g^1} \cdot \overline{b_e^1} + \overline{b_g^1} \cdot \overline{b_g^2} \cdot \overline{b_e^2} + b_g^1 \cdot b_e^1 \cdot b_e^2$ | | $b_e^0 = b_e^1 \cdot b_e^2$ |

still better performance and space efficiency by considering the dependence of circuit layout on decomposition. Since the goal is to lose information, we currently apply redundant carry-generators and carry-generators before half-adders, but we do not yet have enough design experience to predict which operator ordering will result in the fastest or smallest circuit.

# 3 Physical Implementation

Path-Programmable Logic (PPL) began as a folded PLA, where the *and* and the *or* planes are superimposed. This gives the designer the ability to include memory and other logical elements directly in the array, and to arbitrarily segment the large array into smaller, independent sections connected by a few signal wires. Extensions made to the PPL concept have led to a methodology in which it is not the folded *and-or* plane that is important, but rather a conceptual array of wires, some being used to generate and carry logical values and others interspersed at regular intervals to carry power and ground (see figure 1). The grid pitch in one direction is determined by the pitch of the power/ground busses and in the other direction by the layout size of the simplest (or unit) logic tile. Logic functions are physically implemented as tiles that can be superimposed on the wiring array. Furthermore, the size and shape of each logic tile is quantized by the underlying grid and signal *ports* are constrained to be at locations where wires cross the tile boundary. Since the logical tiles are conceptually superimposed on a wiring array, tile adjacency implies the automatic interconnection of physically corresponding signal ports. If interconnection is not desired, the wire must be explicitly *broken* (by the user) between two adjacent tiles.

The wiring array abstraction, automatic (and hidden) inclusion of power and ground routing in the array, and a symbolic notation for functional tiles yield a very powerful circuit design paradigm in which many time-consuming details are more or less automatically handled (a feature of equal value to human circuit designers and CAD tool writers alike). The generic methodology is called *structured tiling*. With respect to physical implementation, the fundamental elements of *structured tiling* are: 1) modules are constructed by placing functional tiles on a wiring array (or grid), 2) in general, power and ground routing should be automatic and hidden, 3) tile adjacency should imply signal interconnection unless an explicit break is specified, 4) tile areas and shapes must be quantized by the underlying wiring grid, 5) the tileset must provide for signal routing.

## 3.1 The NMOS Tileset

Structured Arithmetic Tiling was first developed using an NMOS technology with a single level of metal interconnect. In NMOS circuits, a single load transistor

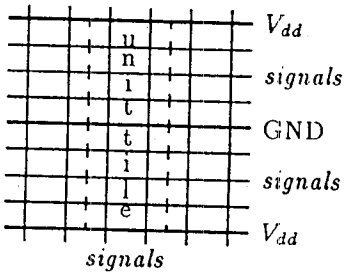
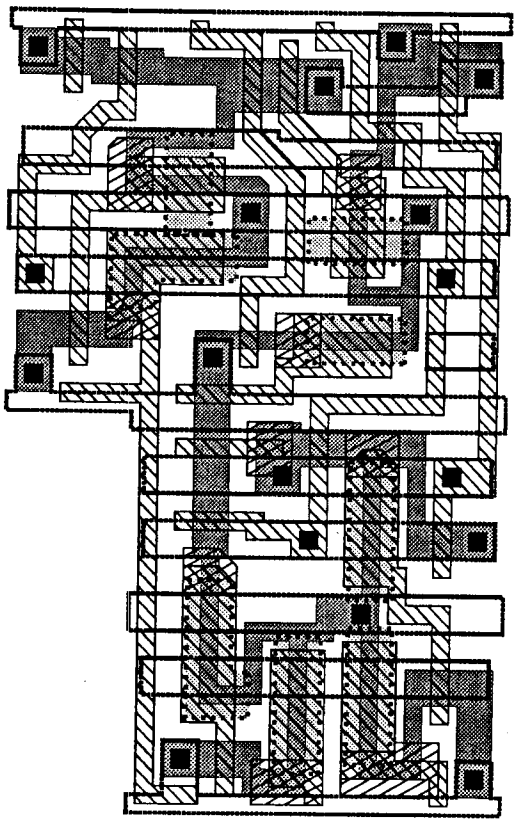



Figure 1: Wiring Array

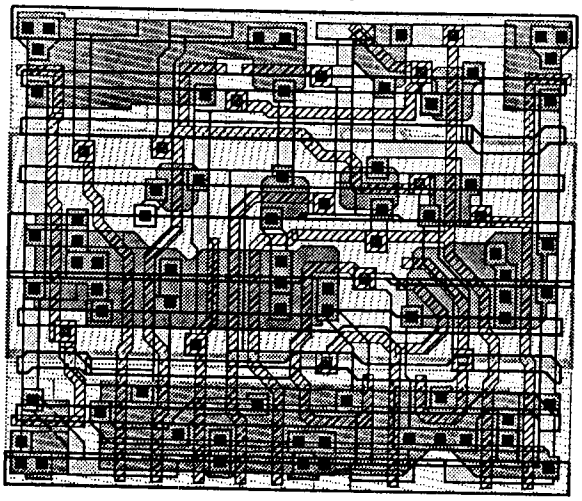
is attached to each gate output and pulls the output high when the input conditions don't require pulling it low. With NMOS logic using gates having more pulldown transistors than pullup transistors, and structured arithmetic tiles generally losing information (that is, they have fewer output signals than input signals and therefore fewer grid locations at the output edge than at the input edge), the NMOS tileset carry-generators have an upside-down "L" shape for optimal space efficiency. Our CAD systems that deal with hierarchical tile-based designs handle non-rectangular hierarchical tiles (since limiting them to be rectangular would be too costly in terms of area), and non-rectangular leaf tiles can be handled in the same manner.

Figure 2(a) presents the layout of the RCG2a tile. This particular tile is four grid units wide at the top (one for each input signal), three grid units wide at the bottom (one for each output signal) and is two grid units high. The unit tile is 66 microns high and 21 microns wide in a 3-micron technology. There are three horizontal ground busses, one at the top, one at the bottom and one in the center of the tile. The top and bottom busses are shared with adjacent tiles. Running horizontally through the centers of the top and bottom halves of the tile are the power busses. Between the power and ground busses are horizontal routing wires that may carry signals through the tile. The four input signals enter at the top of the tile, and the three output signals leave at the bottom, all on polysilicon. Note that the polysilicon wires do not extend to the edge of the tile. The CAD tools automatically insert small rectangles (actually connection tiles placed between functional tiles) of the appropriate material to connect signal wires between adjacent tiles, unless a wire break is specified.

All tiles that lose information (carry-generators and redundant carry-generators) produce a *carry* (either $2a^0$ or $2a^1$) that is the leftmost output and is therefore available at the bottom edge or at the top of the notch. The notch greatly reduces the height of hierarchical tiles designed using the NMOS tileset since often only half a functional tile height is needed to connect carry signals. Tile modifiers are used to connect inputs and outputs to the horizontal (metal) routing wires.



(a) NMOS Layout



(b) CMOS Layout

Figure 2: The SAT RCG2a Tile

### 3.2 The CMOS Tileset

The CMOS tileset implements the same logic function as the NMOS tileset, but its tiles are all rectangular because the number of pullup devices equals the number of pulldown devices. Figure 2(b) shows the CMOS layout for the RCG2a tile. Power is present along the top and bottom edges of the tile, and ground is in the middle. The minimum routing pitch is such that a tile requiring four columns (having four inputs) cannot be contained in the area of four routing tiles. Instead, we chose to make each input and output signal available at at least two adjacent grid locations. This permits tiles to slide one grid location relative to each other while still assuring that tile adjacency implies correct signal interconnection. Each CMOS tile contains well and substrate contacts (to reduce the chance of latchup) in the leftmost and/or rightmost column. While only one signal wire is present in each column, six horizontal signal routing wires exist at minimum routing pitch and each tile has an associated set of tile modifiers to connect any input to any of the top three wires and any output to any of the bottom three wires. There are also modifiers to connect any input to ground (logical zero) which permits mythical inputs to be implemented simply by grounding the input wire. The grid size for CMOS is 135 microns high by 13.5 microns wide for a 3-micron technology. Each CMOS half-adder takes 1.97 times the area of its NMOS counterpart, each carry-generator 2.31 times the area, and each redundant carry-generator 1.75 times the area. In addition, to obtain equal routing capability in each technology, CMOS takes about 0.85 times the area needed in NMOS. CMOS SAT designs that are functionally equivalent to an NMOS design whose area is 70% functional and 30% routing should be about 1.7 times the size of the NMOS circuit.

## 4 A Comparative Example

The Variable Precision Processor furnishes us with several interesting but not too complex modules: Input-Level-2 will serve as an example. This module is designed from the following set-equation:

$$16b^1 + (8a^1 + 4a^0 + 2a^0 + b^0) \Leftarrow$$
$$4b^1 + (8a^1 + 4b^0 + 2a^1 + b^0) + (8a^1 + 4b^0 + 2a^1 + b^0).$$

This circuit adds two numbers with $\delta = 20$ and $\omega = 10$, with the $4b^1$ input being set to 0 for addition and subtraction (but not for multiplication). Running the decomposer generates the information-loss chart in table 4, which can be drawn as a block diagram as in figure 3. This module has been designed in NMOS PPL, CMOS PPL, NMOS SAT, and CMOS SAT. Table 5 contains some comparative data on these various designs (all dimensions are in units of $\lambda$, which would be 1.5 for a 3-micron technology). Note that NMOS SAT is nearly three times as area efficient as PPL for this specific task, while the CMOS ratio is greater than four. This should hold for large arithmetic circuits in general. It is also important to note that CMOS SAT is more area efficient than NMOS PPL. The PPL and SAT designs are presented in figures 4 and 5. The layouts of the PPL and SAT tiles were done by hand and made as small as possible. The tile-level designs were done using an editor (INSTED [9]) especially designed for handling hierarchical, non-rectangular tiles. If the results cited by Israelsen[8] hold, SAT circuits should range from being roughly comparable in area and performance to a full-custom implementation to consuming twice the area and incurring some small performance penalty. While no actual comparisons of NMOS circuits versus CMOS circuits have been made using SAT, it is assumed that the normal tradeoffs between the two technologies will hold (i.e. CMOS will be roughly twice as large as NMOS, will consume virtually no power, and will operate between five and ten times as fast as NMOS).

## 5 Conclusions

The Structured Arithmetic Tiling (SAT) circuit design methodology is one member of a family of methodologies that relies on an entirely tile-based approach to circuit design. The unification of SAT, PPL, and other related methodologies through a set of common design tools will give us the capability to design complex integrated systems that include innovative arithmetic architectures such as the Variable Precision Processor. The size comparison between SAT and PPL implementations of the same circuit indicates, not surprisingly, that design methodologies that are specifically tailored to a task will be more efficient than more general methods. SAT strives to keep the advantages gained in using a structured technique like PPL while improving over PPL by taking into account more domain-specific knowledge.

The automatic tools associated with SAT permit the easy design and implementation of complex arithmetic modules that would be difficult if not impossible to design otherwise. Although no SAT circuits have been returned from fabrication at this point, two initial circuits implemented using PPL have been fabricated, tested, and shown to be completely functional at first silicon.

We believe that SAT can play an important role

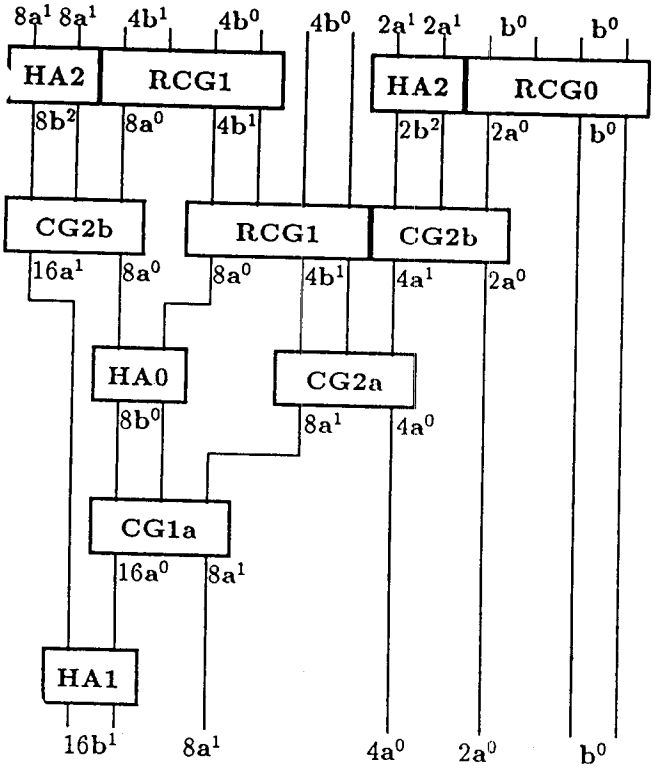


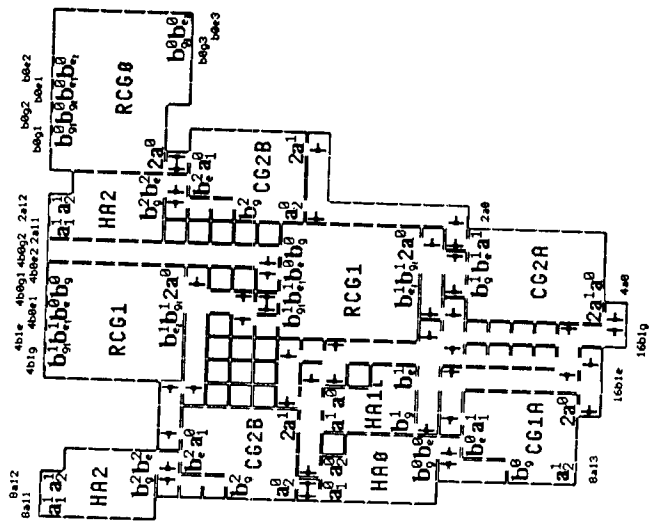

Figure 3: Block Diagram of Input-Level-2

Table 4: Information Loss Chart for Input-Level-2

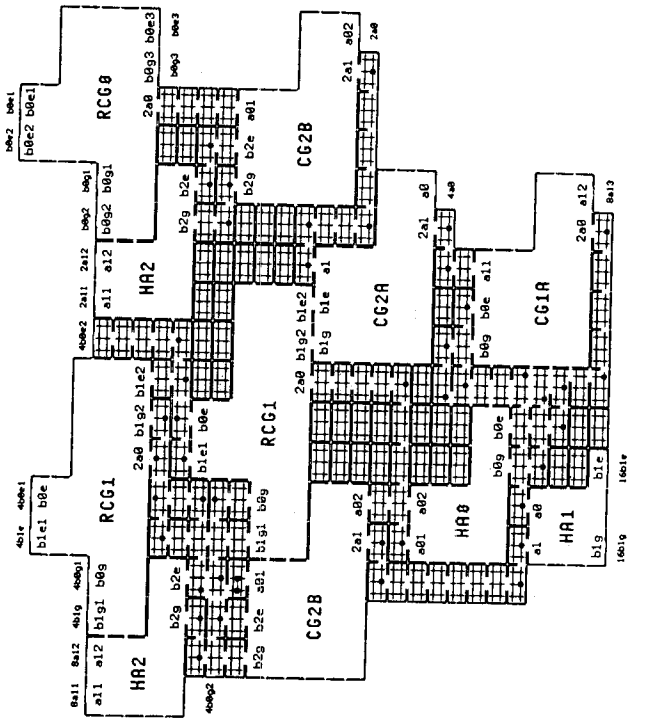
| Lev. | Weight Set | 16 $b^1\ a^1\ a^0$ | 8 $b^2\ b^0\ a^1\ a^0$ | 4 $b^1\ b^0\ a^1\ a^0$ | 2 $b^2\ a^1\ a^0$ | 1 $b^0$ |
|---|---|---|---|---|---|---|
| | Input | | 2 | 1 2 | 2 | 2 |
| 1 | RCG0 | | 2 1 | 1 1 | 2 1 | 1 |
| 1 | HA2 | | 1 1 | 1 1 | 1 1 | 1 |
| 2 | CG2B | 1 | 1 | 1 1 1 | 1 | 1 |
| 2 | RCG1 | 1 | 2 | 1 1 | 1 | 1 |
| 3 | HA0 | 1 | 1 | 1 1 | 1 | 1 |
| 3 | CG2A | 1 | 1 1 | 1 | 1 | 1 |
| 4 | CG1A | 1 1 | 1 | 1 | 1 | 1 |
| 5 | HA1 | 1 | 1 | 1 | 1 | 1 |

Table 5: Comparison of Input-Level-2 (size in $\lambda$) Designs

| Technology | Unit Hgt | Unit Wdt | Bounding Box Row | Bounding Box Col | Bounding Box % Use | Relative Size BBox | Relative Size Used |
|---|---|---|---|---|---|---|---|
| nmos SAT | 44 | 14 | 13 | 8 | 62 | 1.00 | 1.00 |
| cmos SAT | 90 | 9 | 34 | 3 | 88 | 1.29 | 1.80 |
| nmos PPL | 14 | 22 | 25 | 22 | 60 | 2.65 | 2.67 |
| cmos PPL | 21 | 50 | 30 | 18 | 68 | 8.85 | 9.54 |

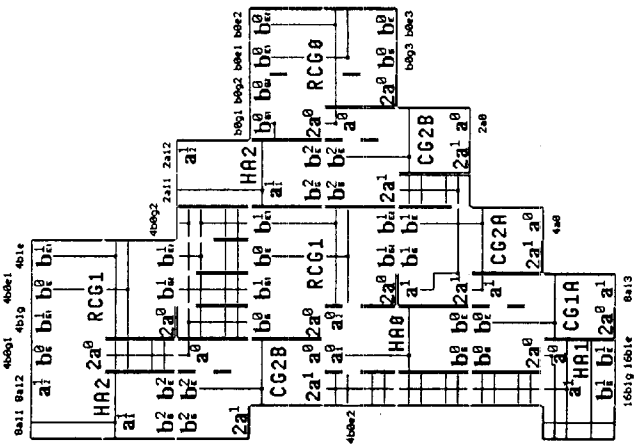



(a) NMOS Design

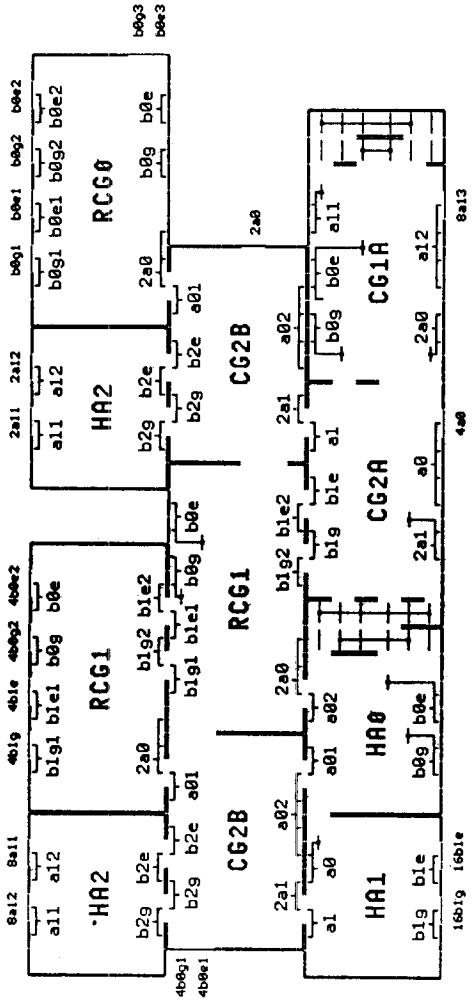



(b) CMOS Design

Figure 4: PPL Designs of Input-Level-2



(a) NMOS Design




(b) CMOS Design

Figure 5: SAT Designs of Input-Level-2

in the development of complex integrated systems. We are continuing our investigations into influencing decomposition with physical constraints (including area and performance), automatic placement and interconnection of modules implemented using SAT, and integration of SAT with other tile-based, circuit-design methodologies such as PPL.

# References


[1] J. E. Robertson, "A Theory of Decomposition of Structures for Binary Addition and Subtraction", Tech. Report UIUCDCS-R-81-1004, Univ. of Ill. at Urbana-Champaign, Jan. 1983.

[2] J. E. Robertson, "A Systematic Approach to the Design of Structures for Arithmetic", *Proceedings of the 5$^{th}$ Symp. on Computer Arithmetic*, May 1981, pp. 35-41.

[3] K. F. Smith, T. M. Carter, and C. E. Hunt, "Structured Logic Design of Integrated Circuits Using the Stored Logic Array," *IEEE Trans. on Electron Dev.*, Vol. ED-29, No. 4, Apr. 1982, pp. 765-776.

[4] T. M. Carter, *Structured Arithmetic Tiling of Integrated Circuits*, Ph.D. Diss., Univ. of Utah, Dept. of Computer Science, Dec. 1983.

[5] C. Y. F. Chow, *A Variable Precision Processor Module*, Ph.D. Diss., Univ. of Ill. at Urbana-Champaign, Dept. of Comp. Science, July 1980.

[6] J. E. Robertson, "Design of the Combinational Logic for a Radix-16 Digit-Slice for a Variable Precision Processor Module", *Proceedings of ICCD 1983*, Nov. 1983, pp. 696-699.

[7] T. M. Carter and L. A. Hollaar "The Implementation of a Radix-16 Digit-Slice Using a Cellular VLSI Technique", *Proceedings of ICCD 1983*, Nov. 1983, pp. 688-691.

[8] P. D. Israelsen and K. F. Smith, "Comparison of the Path Programmable Logic Design Methodology with Other Custom and Semicustom Approaches", *Proceedings of ICCD 1985*, Oct. 1985, pp. 73-76.

[9] R. M. Neff, *INSTED: An Integrated Structured Tiling Editor*, M.S. Thesis, Univ. of Utah, Dept. of Comp. Science, Aug. 1986.