

---

Logic synthesis allied to a hardware-description language can break through the design bottleneck posed by random logic

---

# Logic synthesis

## a faster route to silicon

The short history of integrated-circuit (IC) design has witnessed a potentially mind-boggling explosion in complexity. A little over 20 years ago, the gate count on a digital IC was typically a few tens of gates; today, gate counts of the order of tens of thousands are common, and chips with hundreds of thousands of gates are not unknown.

To cope with the enormous amount of detail inherent in modern digital ICs, designers have adopted a top-down design style, starting the design process at a level far removed from the detail of individual gates. Top-down design requires the support of computer-aided-design (CAD) tools that allow the initial design to be expressed at the required broad level of detail most obviously represented by the various parts libraries provided by application-specific IC (ASIC) vendors. Although such libraries once concentrated on basic logic gates and the equivalent of standard TTL functions, the trend is towards increasingly complex functions, including memory arrays, processor cores, arithmetic logic units etc., allowing design capture at a relatively high level.

As an alternative to designing in terms of a parts library, logic designers are making increasing use of hardware-description languages (HDLs). As their name implies, these are special-purpose languages designed to support the design of digital hardware. Examples include Ella, Helix, Verilog and VHDL. Ella,

which was developed at RSRE Malvern, is the Ministry of Defence's preferred HDL, and is widely used throughout the UK IC-design community.

---

by  
**Roger Dettmer**  
Technical Editor

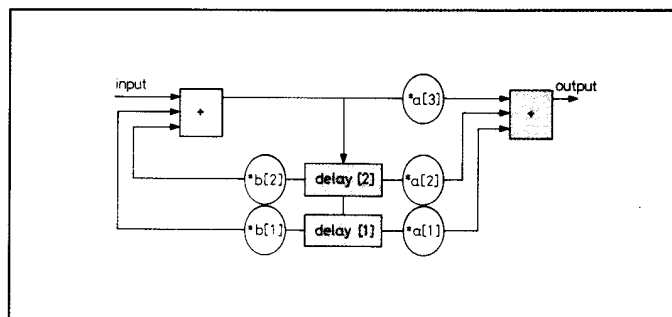
---

As a means of capturing IC designs, HDLs have a number of distinct advantages over the library-based approach. In particular, a good HDL will allow a design to be captured at a more succinct and abstract level than is possible using parts libraries. Succinct and abstract descriptions make for fast simulations, so that, in general, use of an HDL allows a designer to evaluate a number of different hardware architectures before becoming committed to the details of a

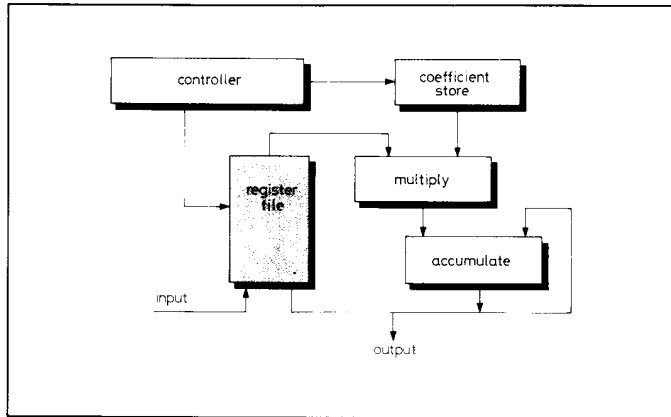
design. In this way, potentially unsatisfactory architectural choices can be spotted early, rather than revealed by a gate-level simulation, at which stage it will probably be too late to make more than minor changes to the hardware design.

Using an HDL to evaluate a hardware architecture is all very well, but it does presuppose that a satisfactory architecture is available. In practice, devising appropriate hardware architectures is the most important part of digital design, and is the one area where CAD tools have the least to offer. A simple example will help illustrate some of the issues.

Any digital signal-processing function can be created out of three basic elements: unit delays, adders and multipliers. A second-order infinite-impulse-response (IIR) filter, for example, can be defined using two unit delays, five multipliers and two adders (Fig. 1). Although Fig. 1



I IIR filter, behavioural model



## 2 IIR architecture

is a perfectly correct behavioural definition of a second-order IIR filter, it does not provide a suitable architectural model for implementing an IIR filter in silicon.

The most obvious problem is the abundance of multipliers. Hardware multipliers require large areas of silicon and one multiplier would be the absolute maximum. Given this constraint, something on the lines of Fig.2 would provide a basis for a

suitable hardware design. The register file holds new input values, intermediate results and output values; the coefficient store holds the three 'a' coefficients and the two 'b' coefficients; the accumulator replaces the two adders; while the controller provides all the necessary control signals to ensure that the various functional blocks work together as required.

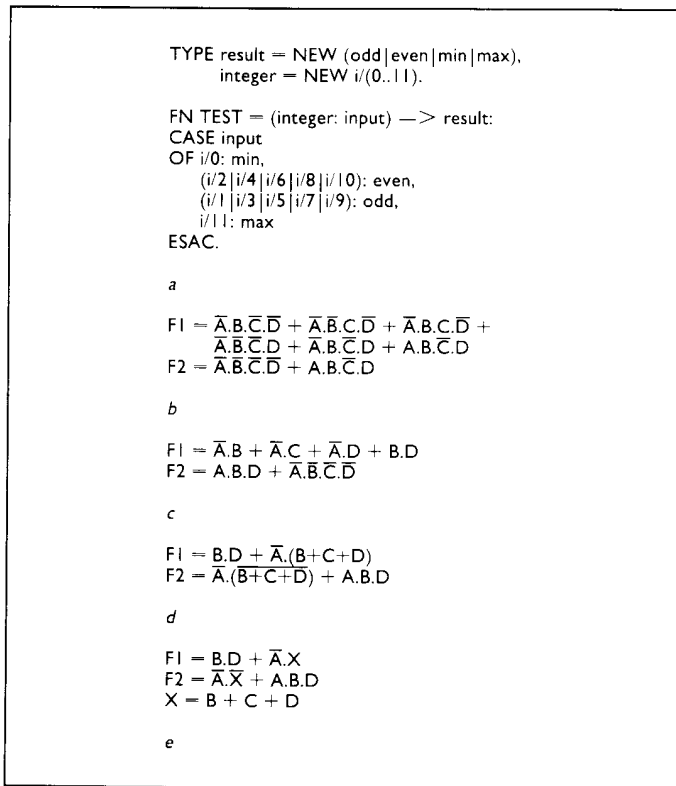
Given a suitable high-level des-

cription, possibly written in ELLA, of a basic architecture of the form shown in Fig.2, simulation tools can be used to explore architectural variations, examining, for example, the timing implications of various implementation options for the multiplier, or whether it might be possible to replace the multiplier by an arithmetic logic unit. While HDLs can do much to support the refinement of architectural details in this way, they do not overcome the need to have a good basic architecture to start with. That task remains very much the responsibility of the designer, and as chips have grown more complex, architectural design has become increasingly difficult.

Faced with the growing complexity of digital systems, designers have responded by adopting a highly disciplined approach to the whole question of devising system architectures. The most important feature of this design style, termed the algorithmic state machine (ASM) method, is a clear separation between the controller (Fig.2) and the remaining functional blocks of a chip, the latter generally referred to as the architecture. Digital design then falls into two distinct but closely related activities: choosing the appropriate functional blocks for the architecture, and devising a controller to make the functional blocks work together as required.

In principle, the role of the controller is very simple — it must provide control signals to the architectural part of the chip in the required sequence. Under the ASM approach, this sequence is defined by a form of flow chart (an ASM chart), in which the controller is viewed as being clocked synchronously between a number of finite 'states'. For a given controller state, and a given set of controller inputs (status signals from the architecture to the controller), the ASM chart defines the controller output signals (control signals to the architecture) and the next state. The basis of the ASM method is thus to represent the controller as a finite-state machine, and the use of a convenient notation, i.e. the ASM chart, for capturing this representation.

Once the ASM chart of a controller has been produced, there are a variety of relatively straightforward, if generally tedious, ways of generating a logically equivalent set of Boolean equations. Unfortunately, such Boolean descriptions



3 Locam optimisation process: (a) ELLA text; (b) initial Boolean equations; (c) two-level minimisation; (d) factorisation; (e) decomposition

## A user's tale

Racal Research provides a centre for VLSI design expertise for the whole Racal Group. One of the first users of Ella outside RSRE, Racal has adopted a design methodology based firmly on a top-down approach coupled with the use of Ella for design capture and evaluation. David Orton, Racal's technical manager, VLSI design development, is firmly committed to the use of Ella.

'We're currently developing a very complex chip design for a radar application. The initial hardware architecture for this chip was described at a high level in Ella, and immediately that enabled us to identify deficiencies in parts of the original specification — either resulting from incompleteness, or from a basic inconsistency with a hardware realisation. That's one of the important advantages of Ella — you can describe something very rapidly and then activate it, and it's activation that's so informative to the designer.'

Some 4 or 5 years ago, Racal began to run into problems arising from the increasing complexity of the state machines being generated as chip controllers. Translating these into equivalent gate-level structures was, in David Orton's words, 'an extremely long-winded, error-prone and tedious process'. In response to this situation, Racal acquired an early public-domain logic-synthesis tool from the USA, and

set about adapting it to its own needs. This took about 1 man-year of effort, and necessitated writing an interface to the Racal cell library.

This public-domain software worked reasonably well, but had the serious disadvantage of requiring input in the form of a truth table. At this time, Racal had adopted the practice of specifying controller designs as Ella case statements, and the manual translation of these Ella-based descriptions into truth tables remained error prone and boring. The arrival of Locam, able to accept high-level Ella as input, was seen as an obvious boon, and Racal is now using Locam as the primary means of generating gate-level controller designs. Three chips have now been designed using Locam, and early results indicate a 20-30% improvement in silicon efficiency over the public-domain software.

Aside from silicon efficiency, the other great advantage of Locam is that it speeds up the whole design process, reducing time to market and allowing greater effort to be devoted to the crucial matter of architectural evaluation. David Orton again: 'In VLSI design we are constantly trying to get further and further away from silicon, so that we can think more about the architecture and data-flow operations within the system as a whole. Locam is an important contributor to this process.'

are not normally a suitable basis for defining a gate-level implementation of the controller. First, they will contain many redundant terms, representing a potential waste of silicon area, and secondly they will require efficient mapping onto the particular range of gates supported by specific silicon vendors.

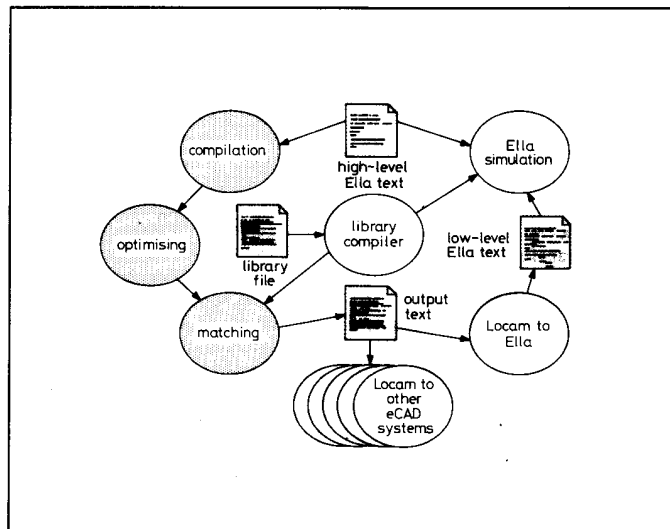
Traditional methods of gate-count minimisation, e.g. Karnaugh maps, are unable to cope with the complexity that characterises the controllers of modern VLSI circuits, so that the task of creating an efficient gate-level description of the controller can represent a major bottleneck in the design cycle. The same problem does not apply to the architecture. In contrast to the random logic that generally forms the controller, the architecture tends to be highly structured, comprising specific functional blocks whose gate-level implementations are predefined by the silicon vendors. To achieve fast, efficient gate-level implementations of the random, unstructured parts of chips (e.g. controllers), designers are turning increasingly to a range of novel CAD tools known collectively as logic synthesis. At their best, such tools can minimise the area occupied by random logic as well as an experienced designer can, and can significantly reduce the overall time to market for a design.

The starting point for the use of

logic synthesis is a description of the circuit to be synthesised. In many of the early applications of synthesis, this took the form of a netlist (a list of structural elements and their interconnections) of an existing circuit, the logic-synthesis tools simply being used to generate an alternative and possibly more efficient implementation. More recently, the emphasis has shifted towards logic-synthesis tools that can accept descriptions of new circuit designs at a level far removed

from the details of gates and their interconnections. The form of such descriptions varies: VLSI Technology, for example, supports graphical or text-based inputs equivalent to ASM charts, but particular interest centres on the use of HDLs as the input to logic synthesis. Examples include the Design Compiler from Synopsis, which accepts Verilog, and Locam from Praxis Electronic Design, which accepts Ella.

The initial development of the algorithms behind logic synthesis



4 Locam design flow

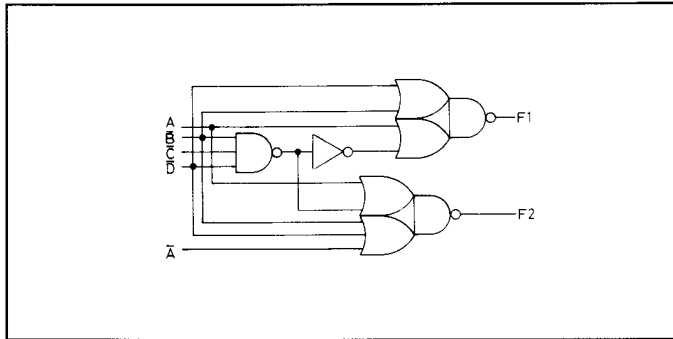
was concentrated in the United States, and US companies have figured prominently in the early application of logic synthesis. IBM developed its own logic-synthesis tool, the Logic Synthesis System (LSS), in the early 1980s, which was used on more than 100 chips in the 3090 computer. Other US users of logic synthesis include Sun Microsystems, Ardent Computer, Stellar Computer and Intel. Locam is very much a European product, and is based on a logic-synthesis system developed within Philips. All logic-synthesis tools follow the same broad principles, and a closer look

Like any human designer, Locam next proceeds to generate more efficient forms of these two-level Boolean equations. This optimisation process takes place in three phases. In phase 1, Locam generates equivalent two-level expressions, but reduces the number of literals (Fig.3c). In phase 2, common factors are introduced to create multi-level logic (Fig.3d), and in phase 3, decomposition identifies common factors between logic expressions and creates cascaded logic (Fig.3e). The net effect of optimisation in this example is to reduce the initial number of literals

inherent efficiency, including the matcher's ability to start by choosing the most complex cells, and the optional preference for the use of exclusive-OR gates, which can give significant area savings in arithmetic circuits.

A schema of the output of Locam's matcher for the equations of Fig.3e is shown in Fig.5. The key features include the use of cascaded logic (the three-input NAND acts as an input function to two other cells) and large complex cells, and the fact that the circuit does not require the inversion of input signals (i.e. although B, C and D appear as literals in Fig.3e, Fig.5 accepts their inverses as inputs, saving on inverters).

The output from Locam is a text-based description of the cells used and their interconnections. This can be translated automatically into a format consistent with the requirements of a range of other electronic CAD systems. Formats currently available include Edif (electronic design interchange format), ES2, LSI Logic, Mentor, Valid, Altera and Xilinx. In addition, Locam-to-Ella translation allows the output of Locam to be simulated as low-level Ella, providing an optional check on the integrity of the whole logic-synthesis cycle (Fig.4).



5 Schema of Locam output

at Locam will serve to indicate what is involved.

#### Locam

The starting point for Locam is a piece of Ella text describing either a complete circuit or a function, within a circuit. Fig.3a is an Ella description of a simple function which is reasonably intelligible even without any prior knowledge of Ella. The function, called TEST, examines an integer input in the range 0 to 11, and then returns a result defined by a CASE statement. If the input is 0, then the result is min; if the input is even (but not 0), the result is even; if the input is odd (but not 11), the result is odd; and if the input is 11, the result is max.

Locam transforms Ella text into gate-level circuit descriptions in three distinct stages (Fig.4). The first stage, compilation, results in a set of two-level (the literals are ANDed together and then ORed) Boolean equations (Fig.3b), automatically assigning the correct number of bits to the input and output signals, i.e. two bits (F1 and F2) to give the four values required for result, and four bits (A, B, C and D) to give the 12 values required for integer.

(A, B, C and D) from 32 in Fig.3b to 13 (A, B, C, D and X) in Fig.3e.

A key feature of optimisation by Locam, and other logic-synthesis tools, is that it is based on techniques that place no theoretical limit on the size of circuit that can be optimised. This is in contrast to traditional approaches to optimisation, e.g. Karnaugh maps, which, even when automated, can only handle circuits of restricted size. Within the broad objectives of minimising gate count, Locam offers the user a range of options. For example, a designer can set limits on the number of levels of logic produced, thereby trading silicon area for a potentially faster circuit. Additionally, limits can be set on the cascading of logic (sharing of a common input function), trading silicon area for wireability.

In Locam's third and final stage, the optimised Boolean expressions are matched against the gate functions provided in the parts library of the target silicon vendor. The Locam matching function is fast and highly efficient, giving results that outperform the human designer. By way of an example, Orbitel has reported the synthesis of a 20 000-gate circuit in 32 minutes. A number of factors contribute to this

#### A silicon compiler?

If the constant concern of VLSI designers is to get as far away from silicon as they can, the question naturally arises: how far can they go? Ultimately, one can envisage an electronic CAD system that will accept a behavioural description of a chip and then, at the press of a button, translate this description into a correct, silicon-efficient, gate-level implementation — the so-called silicon compiler. On this strict definition, Locam is not a silicon compiler as it leaves responsibility for architectural development firmly in the hands of the designer, where it will probably always be.

Nevertheless, Locam, and other similar tools, can automate the gate-level design of what would otherwise be large areas of manually generated random logic. As such, they are likely to play an increasingly important role in VLSI design and development.

#### Acknowledgment

The author would like to thank Peter Claydon of GEC Avionics, Paul McLellan of VLSI Technology and John Saunders of Praxis Electronic Design for their help in the preparation of this article.