INT. J. ELECTRONICS, 1992, VOL. 72, NO. 1, 1-10

YAZDANI J, MASUD M, SADIQ M. SAIT•PCB LAYOUT GENERATION FROM RTL SPECIFICATIONS •INTERNATIONAL JOURNAL OF ELECTRONICS 72 (1): 1-10 JAN 1992

JAWEED YAZDANI†, MANZER MASUD† and SADIQ M. SAIT†

Automatic printed circuit board (PCB) layout generation is currently achieved through the use of several independent CAD systems. A register transfer level (RTL) based design automation system provides a viable alternative to existing methods of PCB layout generation. In this paper, the design and implementation of a software processor that generates a PCB layout using SSI/MSI components from the logical circuit provided by UAHPL, an RTL language, is presented.

1. Introduction

The design process for digital systems usually involves a number of processes—specification of system structure and functional behaviour, testing of functional behaviour of the system, and generation of the detailed circuit. The automation of the design process has led to the development of independent systems that tackle each of the design processes individually. These systems are termed computer aided design (CAD) systems. Although CAD systems achieve many of the objectives of digital design automation (DA), they fail to provide an integrated design environment for design engineers. Other efforts to integrate DA systems have led to the use of design languages—called computer hardware description languages (CHDLs)—to describe digital systems.

Present-day digital system design is supported by CAD systems that are either based on CHDLs or are independent of them. The end product of these CAD systems is the complete digital circuit that may be implemented, either as a chip or on a printed circuit board (PCB), based on the technology desired. Presently, the methods used in developing PCB layouts are confined to independent CAD systems. Another possibility is to develop a CHDL based system for the generation of PCB layouts (Yazdani 1988).

CHDL-based systems are mostly oriented towards the development of large scale integration (LSI) or very large scale integration (VLSI) systems. Very high speed integrated circuits (VHSIC) hardware description language (VHDL) is a CHDL which was recently developed and aims to support a DA system to develop VLSI chips (Waxman 1986). A more recent effort in CHDL-based systems is the development of a software processor to generate a digital system as a VLSI chip using the logical design of the digital system produced by the UAHPL compiler (Sait 1987). Universal hardware programming language (UAHPL) is an extension of AHPL, a simple and efficient hardware programming language.

The need to translate a CHDL-based design in terms of standard small scale integration (SSI) and medium scale integration (MSI) components is felt. Special-purpose digital systems in low volume may not always prove cost effective in terms of a single LSI/VLSI chip. Standard SSI/MSI components are freely available whereas

Received 12 January 1990; accepted 11 June 1991.

† College of Computer Science and Engineering, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia.

0020-7217/92 $3·00

fabrication facilities are restricted to a few locations only. A CHDL based DA system for PCB design is, therefore, an attractive alternative.

2. An overview of Universal AHPL

UAHPL is a CHDL that allows description of a circuit at an adequate level of abstraction, namely the register transfer level. UAHPL has the capability of generating large iterative combinational logic networks (CLUs). It allows concurrency, parallel processing, pipelining and a multi-module system, thereby allowing a well-structured and modular description of the digital system. UAHPL has been developed after considering the various design and test activities in different environments (Masud 1981). The latest version of UAHPL allows a better choice of register types, clocking options and bus types. It accommodates pass transistors, wired-OR gates and temporary registers. UAHPL has good software support with a multi-stage compiler that produces the logical network (Masud 1981) and simulation facilities to check the functional behaviour of the circuit (Alsharif 1983). The block diagram of UAHPL DA system is shown in Fig. 1.

3. Design process

The design process begins with the development of the RTL model of a digital circuit to be implemented. The UAHPL compiler processes the description and produces an interconnection list, in terms of hardware primitives, that represent the logic-level design of the circuit. These hardware primitives include logic gates and flip–

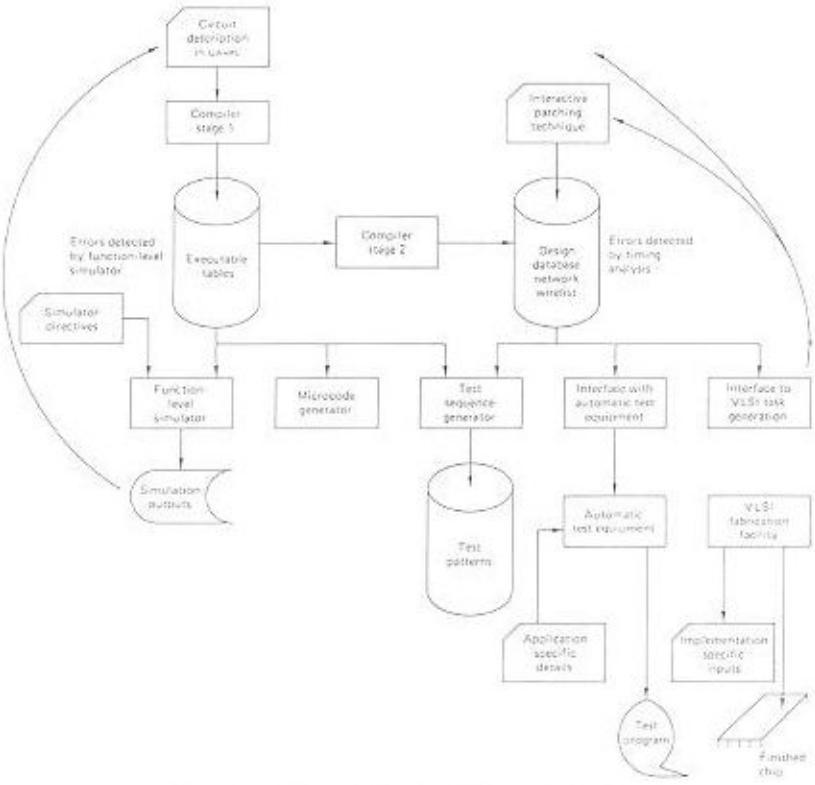

Figure 1. UAHPL design automation system.

flops (Masud 1981). Through a process of chip selection and pin assignment the interconnection list is converted into an interconnection list of SSI/MSI devices. This list is further processed by the placement and routing procedures to produce a complete circuit for PCB (Yazdani 1988).

3.1. *RTL modelling and design example*

The multiplier example given in Fig. 2 illustrates several basic constructs and features of the UAHPL language. The multiplier receives two 4-bit positive integers from an external source on an 8-bit INPUTBUS, and multiplies the two vectors by the simple shift and add technique to produce an 8-bit result. The example has been amply documented by comments enclosed in quotation marks. A reader who is unfamiliar with UAHPL may refer to Masud (1981).

In UAHPL, sequential automata are described as MODULES, iterative combinational circuits, such as adders, decoders etc., can conveniently be described as combinational logic units (CLUs), and circuits with memory but no sequential control such as shift registers, counters etc. may be described as 'functional registers' (not seen in the example). This classification allows one to model and use available digital ICs in a convenient manner. The generic CLU description is used as a template by the compiler to generate copies of the combinational circuit.

In UAHPL description of a module, only those operators are allowed which find a simple and direct hardware correspondence. UAHPL does not have primitive operators for addition, subtraction, incrementing and decoding. The reason is that depending on engineering trade-offs these devices may be designed in different ways. For example we may have a serial adder, a ripple carry adder or a carry look-ahead adder. Rather than restricting the choice to a few built-in options, UAHPL provides the facility to describe such devices as CLUs or 'functional registers'.

3.2. *Interconnection list generation and its preprocessing*

The basic kernel of the UAHPL DA system is the multi-stage UAHPL compiler shown in Fig. 1. The first stage of the compiler processes the UAHPL circuit description, performs syntax and semantic analysis and produces an internal tabular representation. The second stage produces a logic-level interconnection list. The Stage-2 output is in the form of a doubly linked list consisting of a GATELIST and an IOLIST. All Stage-3 processors are application dependent and therefore require specific details depending on the target technology. Presently, Stage-3 processors include a function-level simulator, a VLSI chip layout generator and the PCB layout generator. Suitable modifications to the interconnection list based on technology specifications are required prior to its use by a Stage-3 processor. These modifications are as follows.

(i) The fan-out requirements are checked and in case this criterion is not satisfied, the interconnection list is modified by introducing new gates. The fan-in criterion is met at the time of selection of ICs for the circuit.

(ii) The signal sets are generated by the preprocessor. All electrically common points in the circuit are assigned a negative number. The Stage-2 compiler assigns -1 to V_{cc} and -2 to ground. The rest of the electrically common points are assigned numbers starting from -5.

(iii) Single input NAND/NOR gates are replaced by inverters.

3.3. *Chip selection and pin assignment*

The modified interconnection list requires conversion to a list consisting of SSI/MSI devices. This list, called the ICLIST, is then used by the placement and routing procedures to generate the PCB layout. The generation of the ICLIST requires procedures that address the following issues:

(i) selection of ICs; and

(ii) pin assignment.

The ICs are selected by matching the hardware primitives in the interconnection list with elements of a chip library containing description of such devices. Pin assignment is done based on the assignment of circuit elements to the elements in the selected ICs. The UAHPL generated logical circuit provides a list of hardware primitives forming the circuit. The hardware primitives include logic gates, memory elements, etc. The process of generating a list of SSI/MSI devices that replace the primitives involves an efficient mapping of the primitives to the available SSI/MSI devices. The list of available devices is provided by the chip library. The library also contains a listing of the pins on the IC in a predetermined format. The ICs in the present library belong to the TTL family.

The complete description of an IC in the library is provided by unique entries in two tables, the chip selection table (CST) and the pin assignment table (PAT). A record in CST contains the UAHPL generated gate type, the IC number, the number of inputs to each logic element in the IC, the number of elements per IC, and the length and width of the IC. Each record of PAT, contains the IC number, the number of valid pins on the IC, the pin number(s) representing the input(s) to a gate in the IC, and the output pin number(s) for the gate. The input and output pin numbers to the other gates in the IC are similarly assigned. V_{CC} and ground are the last two pins specified in this record.

The ICLIST is generated by the selection of ICs from the library which match with the hardware primitives in the interconnection list and assignment of these primitives to the selected ICs. Some of the ICs contain more than one hardware primitive of the same type. An efficient mapping can be achieved by the grouping of elements of the interconnection list into a minimum number of ICs and assignment of those elements having common connections to the same IC. The procedures required in selecting ICs for memory elements differ from those required for logic gates. CLUs do not have corresponding ICs in the library and are therefore assigned black boxes.

Memory elements are represented by flip–flops in the UAHPL generated interconnection list. The flip–flops are present in both the control and data logic for the circuit. The selection of ICs for the memory elements proceeds with the grouping of flip–flops having the same enable, set and reset inputs. These groups of flip–flops are assigned to ICs containing registers or D-flip–flops. The objective is to minimize the number of ICs by assigning as many flip-flops as possible to one IC. The present library contains ICs with 8-bit and 4-bit registers and dual D-flip–flops. The data and control flip–flops are handled separately in the selection process. All the control flip–flops in the interconnection list have the same enable, set and reset lines except for the reset control flip–flop. The reset control flip–flop is specified as part of the UAHPL description. The control flip–flops are grouped into ICs containing registers/D-flip–flops depending on their number. A separate IC is assigned to the reset control flip–flop.

Not all data flip–flops (unlike the control flip–flops) may have the same enable input. Data flip–flops having the same enable signal are clustered into one or more ICs depending on their number. The assignment of data flip–flops having the same enable

to an IC is dependent on the clock and flip–flop input. The assignment procedure puts those flip–flops with common clock input in the same IC. The selection procedure has been designed so as to generate a minimum number of ICs. For example, if three flip–flops having the same enable signal exist then instead of two ICs of dual D-flip–flops, an IC containing a 4-bit register is selected. Similarly, in the case of five flip–flops having the same enable signal, a single 4-bit register and a D-flip–flop is selected.

The gates in the interconnection list produced by the Stage-2 and modified by the preprocessor have corresponding ICs in the library. A count of each set of gates is made. A set of gates is identified by its type and the number of inputs. The number of ICs needed for each set of gates is computed based on their count. The assignment procedure assigns those gates having common connections to the same IC in each set.

3.4. *Partitioning*

The selection procedures generate the required number of ICs for the system described in UAHPL. It may not be possible to accommodate all the ICs onto a single board of convenient size. In such cases, the circuit is partitioned into a set of boards. Rather than partitioning the interconnection list produced by the Stage-2, we partition the ICLIST. The partitioning problem is dependent on certain parameters that restrict the number of elements (ICs) on a single board. These parameters are the size of the board and external connection limits on the board. The 'partitioning problem' is to find a minimum number of partitions such that each partition satisfies the space and external connection limits. The partitioning algorithm is a modified version of an algorithm given by Hanan and Kurtzburg (1972 a).

Initially the seed elements for each partition are generated. A seed element is identified by the maximum number of connections from other elements. Once a seed element is selected, the 'neighborhood' of that element is generated. The 'neighborhood' of an element is a list of elements that have a certain degree of connectivity with the element. The list is sorted on the number of connections each element has with the seed element. The elements in the sorted list that satisfy the external connection limit and the seed element are removed from the candidate list for the selection of other seed elements. This process is continued until all seed elements are selected. Each partition at this stage contains only the seed element. The candidate elements at this stage are all elements except the seed elements. The partition problem is now reduced to a sequence of linear assignment problems wherein the candidate elements have to be assigned to the partitions based on the space and external connection constraints. The candidate elements are assigned to the partitions using the row scan method for solving the linear assignment problem.

It is possible that not all candidate elements may be placed in the partitions because of the space and external connection constraints. In such a case, the number of partitions is incremented by a factor and the partitioning process is repeated. The factor is computed by dividing the number of candidate elements not assigned to any partitions and the average number of external lines contributed by each element assigned to the partitions. The repetition continues until all the elements find place in the partitions. Each of the partitions are stored as individual circuits to be further processed by the placement and routing software.

3.5. *Placement*

An important step on the generation of the PCB layout is the placement of modules on the board. The placed modules should satisfy certain criteria defined on the pin

connections between the modules, for example, reduction in overall wirelength. The placement of modules also influences the subsequent process of wire routing. The process of finding an efficient placement is a difficult combinatorial problem. Exhaustive evaluation of all possible solutions, which yields an optimal solution, is costly in terms of computing time for a large number of modules.

Any attempt to get a solution close to the optimal must also be computationally efficient. A two-step approach that converges to a near-optimal solution in a relatively short time is as follows.

Step 1. Generation of a good initial placement using a constructive placement algorithm.

Step 2. Improvement on the initial placement through the use of an iterative placement–improvement algorithm.

A number of constructive initial placement algorithms are available in the literature (Haman and Kurtzburg 1972 b). These approaches involve the selection and placement of modules closely connected to the already placed modules. This process is continued until all modules are placed on the board. The advantage in using these methods is that the amount of computation time is comparatively small for an acceptable initial placement. A heuristic that generates an initial placement is the 'cluster development method' (Haman and Kurtzburg 1972 b). Suitable modifications to this method have been made to produce a good initial placement. Iterative methods seek to improve on the placement by repeated modification of it. Pairwise interchange, an iterative placement method in which a module is exchanged with another module only if a reduction in wire length is achieved, produces a better layout although at considerably increased costs in terms of computation time when compared with the constructive initial placement method. Iterative methods may get stuck at a local optimal. A more globally optimized solution, therefore, might not be possible unless a good initial placement is used. The combination of the two methods discussed above, an efficient initial placement method and an iterative improvement method, has produced acceptable results.

3.6. *Routing and editing*

The immediate step following the placement of modules on the PCB is routing—the generation of precise conductor paths necessary to properly interconnect the modules on the board. Although PCBs exist with more than two layers, only those with two layers are considered in this work. One layer contains horizontal paths and the other layer consists of vertical paths. This kind of routing is also referred to as two layer H–V routing. Although routing is a difficult problem in itself, the many and varied constraints further complicate the routing process. These include limiting the width of the conductor paths, maintaining a minimum distance between two paths, minimizing feedthroughs, etc.

The objective of a routing algorithm is to generate a complete set of defined connections between the various elements of the circuit such that the total wire length is minimized, the various routing criteria are satisfied and no conductor paths intersect on the same layer. Generally, the circuit is not completely connected in the first attempt. The routing process must include methods that guarantee a completely routed circuit. The routing algorithms used by Sait (1987) attempt to achieve complete routing. If the circuit is not completely routed, the paths are reordered. The reordering is done such

that the paths that were blocked are connected first. In the case where reordering produces further blocked paths, the modules on the board which are blocking the path are shifted such that the blocked paths are routed. This process of reordering the paths and shifting of modules is continued until all paths have been routed. Procedures used by Sait (1987), with minor modifications, have been used in this work to route the circuit on the PCB. The routing algorithms attempt to connect any two points with the shortest path using the two available layers. A grid router based on Lee's algorithm (1961) is used for this purpose.

The placement of modules on the PCB and the subsequent routing is done automatically. This does not necessarily guarantee the best result from the design engineer's viewpoint. A layout editing facility has been provided such that manual editing of the layout would improve the layout to the designer's satisfaction. A layout editor, LUCIE, has been interfaced with the PCB layout generator. Interface programs that convert the output of the router to the LUCIE description format have been developed. A detailed description of the LUCIE system is available (Guyot *et al.* 1985).

4. Results

A number of examples have been used to test the various procedures developed for the proposed system. These include generation of PCB layout for a subset of INTEL

```
MODULE    :MULTIPLIER.
  MEMORY :    AC1[4];AC2[4];COUNT[2];EXTRA[5];BUSY.
  EXBUSES    : INPUTBUS[8].
  EXINPUTS   : DATAREADY;CLOCK;RESET.
  OUTPUTS    : RESULT[8];DONE.
  CLUNITS    : ALU[5].
BODY
  SEQUENCE      : CLOCK.
  1    AC1,AC2 <= INPUTBUS;   "registers receive data"
       EXTRA  <= 5$0;
       => ('DATAREADY) / (1).
  2    BUSY   <= \1\;       "shift if LSB is 0"
       => ('AC1[3] )/ (4). "add then shift if LSB is 1"
  3    EXTRA <= ALU[0:4] (\1\;EXTRA [1:4]; AC2). "add"
  4    EXTRA,AC1 <= \0\,EXTRA,AC1[0:2];
       COUNT <= ALU[0:1](\0\;COUNT);    "increment COUNT"
       =>'(&/COUNT)/(2).
  5    RESULT = EXTRA[1:4],AC1;  "output result"
       DONE = \1\;
       BUSY <= \0\;
       =>  (1).
   ENDSEQUENCE
   CONTROLRESET (RESET) / (1).
END.
```

Figure 2. UAHPL description of a 4-bit multiplier.

8085, a 4-bit multiplier, and a 16-bit microcomputer. For the sake of brevity, we consider the 4-bit sequential multiplier whose UAHPL description is given in Fig. 2. The multiplier operands are received from the INPUTBUS and stored in registers AC1 and AC2 in the control step 1. The actual multiplication takes place in steps 2–4. An arithmetic logic unit (ALU) is used with its first argument specifying the operation. If the first argument is a 1, the operation is ALU and is executed on the next two arguments. Else if the first argument is a 0, then the operation is INC (increment) and is executed on the next argument. The result of the multiplication is placed on the output lines RESULT in step 5.

The chips selected for the multiplier and their types are given in Fig. 3. The PCB layout for the multiplier example is given in Fig. 4. The external connection lines originate from the connector on the bottom right. The ALU (type 7000) is shown in the centre (IC #8). Comparison based on chip count and gate utilization was made between manual hardware implementation of the multiplier and the generated layout. The design for manual implementation was extracted from the UAHPL description of the multiplier. A total number of 27 ICs have been generated for the PCB layout of the multiplier example compared with 29 ICs used in the manual implementation. The gate utilization was 90·4% for the layout compared with 80·2% for the manual implementation. This may be due to provisions in the layout generator such as the grouping of

NUM	TYPE
1	74173
2	74173
3	74173
4	74173
5	7474
6	7474
7	7474
8	7000
9	7400
10	7400
11	7404
12	7404
13	7404
14	7408
15	7408
16	7408
17	7408
18	7408
19	7408
20	7408
21	7408
22	7408
23	7408
24	7432
25	7432
26	7432
27	7432

Figure 3. Chip types for multiplier.

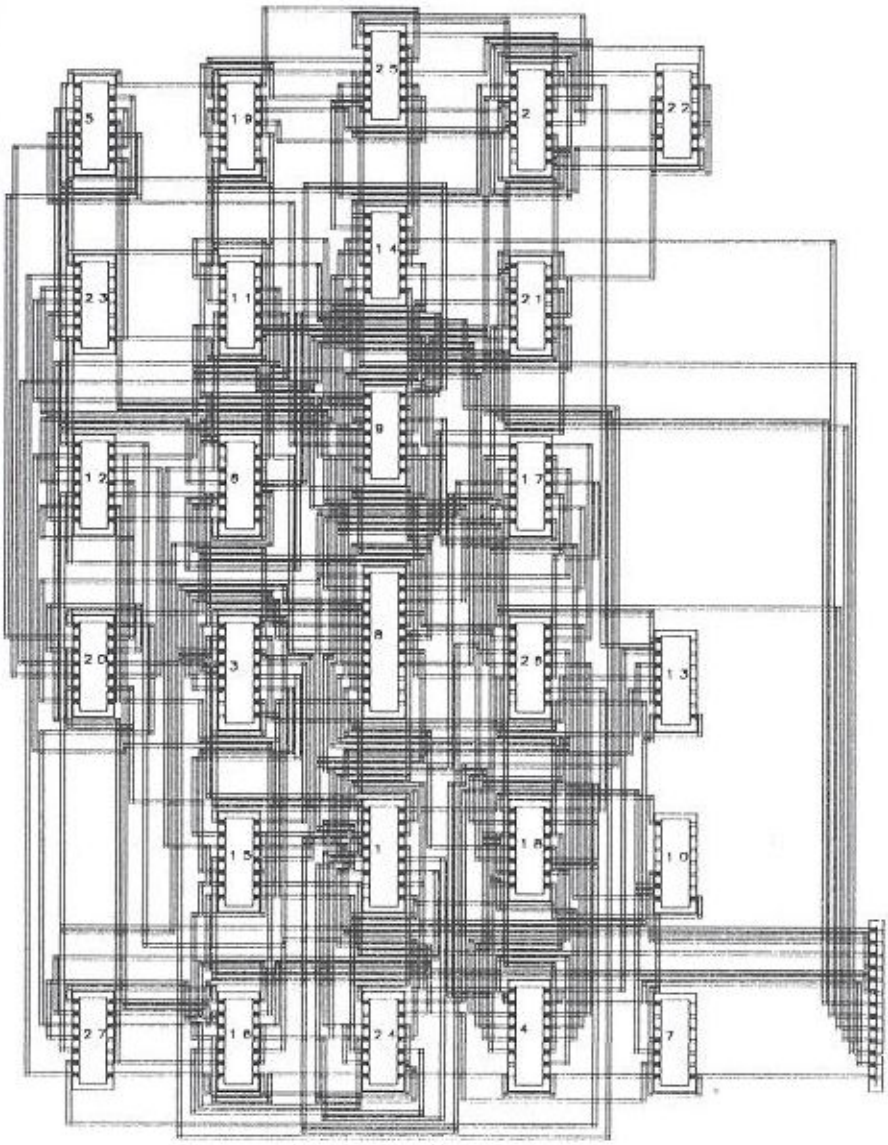

Figure 4. PCB layout for multiplier.

data and control filp–flops into registers, sharing of chips by both data and control units, etc. Similar results were produced while testing the system with other examples.

5. Conclusions

PCB layout generation from RTL specifications is a new approach to the design of PCB based digital systems. The system in its present state can generate PCB layouts for digital circuits described in UAHPL. However, considerable sophistication can be achieved by the expansion of existing tools and the addition of certain other tools.

The immediate step following the generation of the PCB layout is to interface the system with an artwork generation facility. Such a facility combined with other CAM

tools would result in the physical implementation of layouts generated by the layout generator.

TTL chips in the chip library containing registers and flip–flops do not support a uniform pin assignment strategy. This has resulted in a limited choice of TTL chips for the selection procedures. A similar problem is faced in handling CLUs or ALUs defined in the UAHPL description. Conventions could be developed to map the various inputs and outputs of the CLU/ALU with chips in the library. For example, the symbol table generated by the UAHPL Stage-2 compiler can be used to identify an ALU. The UAHPL generated order of pin connections can then be mapped to the ALU pin structure stored in the chip library. Considerable modifications to the preprocessor, chip selection and pin assignment procedures are required to implement this facility.

The present library can be further expanded by introducing new IC modules. The intermixing of logic families or the replacement of the entire library by modules belonging to another logic family is also possible. This will require modifications in the chip selection and pin assignment procedures and the chip library.

In spite of the suggested improvements which might be incorporated, the DA system in its present form is unique and powerful. It provides an integrated environment for design and testing and has an extremely fast turn-around time compared with other manual or graphic-based CAD systems. To our knowledge, no other systems exist that generate PCB layouts of digital systems directly from its algorithmic (register transfer level) specifications. Furthermore, the PCB layouts generated by the system compare well with manual implementations.

Acknowledgment

The authors acknowledge the support of King Fahd University of Petroleum and Minerals.

References

Alsharif, M. M., 1983, Functional level simulator for universal AHPL. M.S. thesis, University of Arizona.

Guyot, A., Jerraye, A., and Raymond, J., 1985, Langage universitaire conception de curcuits integrés pour l'enseignement. IMAG report.

Hanan, M., and Kurtzburg, J. M., 1972 a, Partitioning and card selection. *Design Automation of Digital Systems: Theory and Technique*, Vol. I, edited by M. A. Breuer (New York: Prentice Hall), Chap. 4, pp. 173–212; 1972 b, Placement techniques. *Design Automation of Digital Systems: Theory and Technique*, Vol. I, edited by M. A. Breuer (New York: Prentice Hall), Chap. 5, pp. 213–282.

Lee, C. Y., 1961, An algorithm for path connections and its applications. *I.R.E. Transactions on Electronic Computers*, **10**, 346–365.

Masud, M., 1981, Modular implementation of a digital hardware design automation system. Ph.D. dissertation, Department of Electrical Engineering, University of Arizona.

Sait, S. M., 1987, VLSI mask generation from register transfer level descriptions: an automated approach. Ph.D. dissertation, Department of Electrical Engineering, King Fahd University of Petroleum and Minerals.

Waxman, R., 1986, Hardware design languages for computer design and test. *I.E.E.E. Computer*, April.

Yazdani, J., 1988, A software tool to generate PCB layouts from RTL specifications using SSI/MSI devices. M.S. thesis, College of Computer Science, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.