KWIRE: A multiple-technology, user-reconfigurable wiring tool for VLSI

by P. C. Elmendorf

In a VLSI design environment where a range of chip technologies are available and concurrent chip designs are commonplace, it is not feasible to build a wiring program for each technology. Additionally, a chip's design methodology may demand specific abilities from a wiring program. KWIRE was developed to meet the needs of a multiple-technology, multiple-methodology VLSI design community. It has been used on a range of chips, from small designs to custom microprocessors. Modeling the users' designs and design rules in geometric terms allows KWIRE to handle such a diversity of chip designs. This paper describes the KWIRE system and router.

Introduction

VLSI chip wiring has long been recognized as an area where automated tools are required because manual methods are too error-prone and time-consuming. Due to increasing chip complexity and the need for rapid turnaround, automated wire routing is generally recognized as a necessity.

A design environment where multiple VLSI technologies and design methodologies are in use simultaneously puts additional demands on the wiring program. It is too expensive to develop a separate wiring program for each technology and methodology, and quick realization of the appropriate router is mandatory.

The KWIRE wiring system was created for a multiple-technology, multiple-methodology VLSI environment. It uses a single wire router which operates on a geometric description of the wiring problem. A family of modeling programs converts the wiring problem and its associated technology- and methodology-dependent rules into the geometric description. The results of the geometric wire router are postprocessed by other programs, restoring the technology and methodology constraints during the creation of physical data. These modeling and postprocessing programs are easily and quickly constructed, as opposed to the complex wire router itself. KWIRE permits one wire router to be used on many problems in different technologies, with differing methodologies. The investment in the router is thus spread over many projects.

KWIRE is implemented as a CMS [1, 2] program running under VM/SP [3]. It is being used in ten IBM laboratories worldwide. As of this writing, KWIRE is being used at IBM

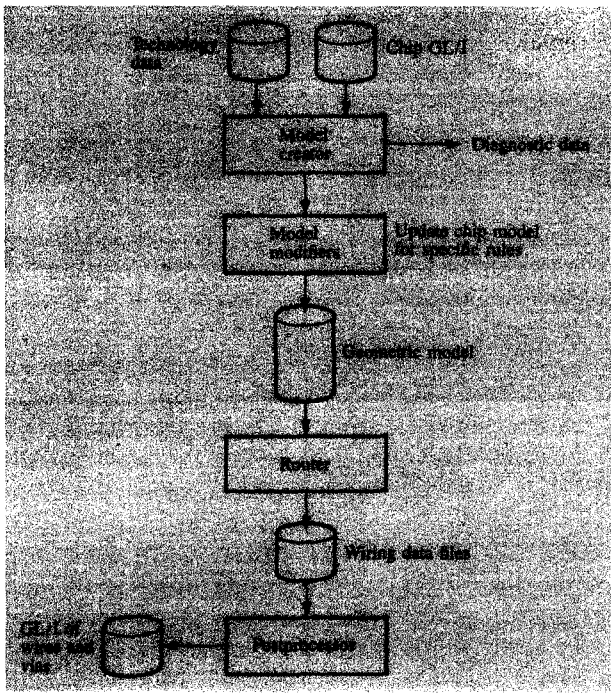

Figure 1

KWIRE process flow. Physical data input is transformed to geometric models. The resulting geometric routing is converted to physical data output.

Kingston to wire five chips. Three technologies and four methodologies are involved.

Process overview

The KWIRE wiring process may be divided into three major phases, as illustrated in **Figure 1**:

1. Creation and modification of a geometric model from chip physical data and rules.
2. Wire routing on the geometric model.
3. Postprocessing the router output into physical data.

Phase 1: Model creation

The chip physical data and technology rules are input to this phase. The chip data are expressed in IBM's corporate-wide standard graphic data format, Graphics Language/One (GL/I) [4].

Groupings of circuits on the chip are called macros. Macros may be large circuit groupings [5], smaller groupings [6], or even individual circuits. Macros of widely differing sizes and shapes can be used on the same chip. The size of the macros depends on the problem at hand. Use of large macros helps partition a complex function into easily handled pieces. KWIRE can be used to wire the internal connections of large macros.

Macros are usually represented by shadow shapes and logic service terminal (LST) shapes. Shadow shapes are polygons which represent the containing boundary of the actual internal circuit shapes. The use of shadow shapes helps reduce the volume of input data. Shadows may be manually generated, or automatically created by a macro generation program [6, 7]. The LST locations are recognized as shapes which possess netname attributes [4]. The objective of the router is to connect all the shapes which have the same netname attribute.

The GL/I input is first processed into a set of data files which represent the chip as a collection of rectangles and netnames. This intermediate format is input to a program which creates the geometric model files for the router. This geometric model may need further modification because of particular technology constraints which are not accounted for in the model builder. A large collection of modification programs has been created for this purpose. Their functions are too specific to warrant inclusion in the model builder.

To keep the geometric model file and the model builder simple, only rectangles, orthogonal polygons, and orthogonal lines are accepted as input. Any shapes not meeting these criteria are flagged and rejected during the model creation process. Orthogonal polygons and lines are fractured into rectangles.

Phase 2: Geometric router

The KWIRE router operates on the geometric data. The LSTs have become locations of netnames, and the shadows have become blocked areas. Wires, contacts [8], and vias [9] are laid down according to the rules implicit in the structure of the model files and the explicit router options selected by the user. (For the purposes of discussion, *via* means a level change from one wiring layer to an adjacent layer.) Wires and vias are abstract: zero-width lines and points, respectively, with no overlaps. KWIRE assumes that wires can be placed in adjacent tracks and that vias can be placed adjacent to wires. These assumptions simplify the problem but may result in a loss of wiring density. KWIRE does not permit 45-degree wiring, further simplifying the problem.

The router is built around a heavily modified Lee Method [10]. Most of the router parameters are under user control. A weighted wiring search gives the user a lot of control over router behavior. The weights specify the relative cost of moves in the various directions. The weights and regions of differing weights are assigned by the user. An iterative rip-up-and-reroute wiring approach, user-selectable via adjacency rules, and several search restriction methods under user control round out the major router capabilities and features. LSTs can be given specialized treatment. All these features are described in more detail below.

Phase 3: Postprocessing

In the postprocessing phase, technology rules and methodology requirements are restored to the purely

geometric results from the router. In general, each technology (and sometimes each methodology) requires its own postprocessor. KWIRE's library of postprocessor subroutines provides a set of building blocks which makes postprocessor construction easy.

Line widths, material overlaps, selection and rotation of vias, and which layer names to assign are some of the physical requirements which the postprocessors restore to the router results.

Process description

The nature of the geometric model is discussed below. The creation of the model, routing on the model, and postprocessing the router results are then discussed in more detail.

• *Geometric model*

The grid

The grid is a discrete Cartesian space. The X and Y spacing of the grid remain constant throughout the levels of wiring, but the X spacing need not be equal to the Y spacing. The grid must be chosen so wires may run adjacently without breaking minimum spacing dimensions for the technology in every wiring level.

All shapes present in the input data are mapped into the grid points. The rules for shape construction are liberal; that is, LSTs must be positioned so that they contain at least one grid intersection and the edges of shapes need not fall on the grid. During model construction, the technology spacing and width rules are applied to determine whether an off-grid shape edge will shrink or expand to the nearest grid point.

Chip map

A storage image of the chip is used as the geometric model. A block of contiguous storage represents a mapping of the macro and wiring blockages and the space that is available for wiring into grid intersection points. Each grid intersection point is represented by a one-byte code. The image is organized such that successive storage locations correspond to increasing values in the X direction of the chip. When the right edge of the chip is reached, the next storage location wraps to the left edge ($X = 0$) and increases in the Y direction.

Multiple contiguous chip images as described above correspond to the multiple levels of wiring. Each image reflects blockages only in that level of wiring.

Each of the eight bits in the byte is utilized. The weight region type, presence of blockage, and presence of wire are recorded. Three bits are used during the wiring scan cycle to remember the direction the search took.

The images are stored in a single map file.

Via map

Storage images of the chip for via blockages are built in the same manner as the wiring levels. There is one less contiguous via map in storage than there are wiring maps. A blockage code at a point in the Mth via plane will prohibit a via from being placed between wiring levels M and $M + 1$ at the same point.

The via planes are used to enforce specific via rules. Each via has a "domain of influence" in its own via plane, the via plane below (if existing), and the via plane above (if existing). These domains and their overlaps are recorded in the via maps. This permits the enforcement of, for example, contact-to-via adjacency rules.

Certain areas in a chip may be restricted from vias. Shapes can be included in the input GL/I which represent via blockages in regions where wires are allowed. A common use of this feature is the prohibition of vias on LSTs.

The images are stored in a single map file.

LST, wire, via models

The LSTs, wires, and vias are stored in separate files. LSTs are represented as rectangles, wires as lines, and vias as points. Polygonal LSTs are represented by "continued" rectangles, commonly called an *LST family*. LST families need not be physically contiguous; this property is useful with *feedthroughs*, discussed later.

• *Model construction*

The model building program is a general modeling process which does not take into account technology or methodology vagaries. Unique requirements are handled by two approaches: alteration of the input to model building, and alteration of the output of model building.

Alteration of input—example

Several tools exist which allow easy alteration of the rectangle files which are input to the model builder. Generally, alteration of input data is used when a methodology rule imposes a restriction which is not present in the technology rules.

Example: Suppose we have a technology in which diffusion is a better conductor than polysilicon (poly), and buried contacts (diffusion-poly ohmic connections) have adverse electrical characteristics. Thus, an artificial rule may be imposed: The wire router is not allowed to wire in poly, nor use buried contacts, but may use diffusion, contacts, and metal. The circuit macros use diffusion *and* poly LSTs. It would seem that we could get by with modeling diffusion and poly on the same wiring plane. However, the diffusion and poly LSTs are the same width and are on the poly-diffusion shadow edge **(Figure 2)**. Simple modeling would let diffusion wiring touch the poly LST. To implement the artificial rule, the poly LSTs are shrunk to the enclosed grid points before modeling starts **(Figure 3)**. Thus, they are

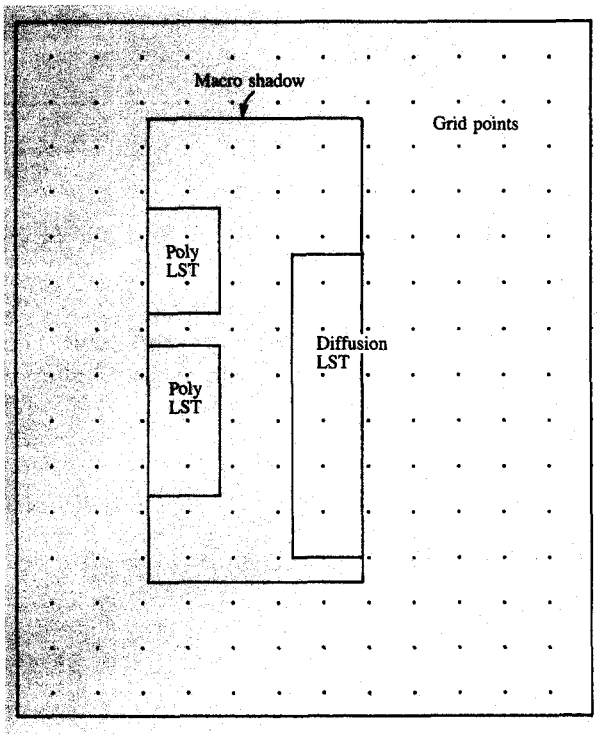

Figure 2

Original data showing large poly LSTs.

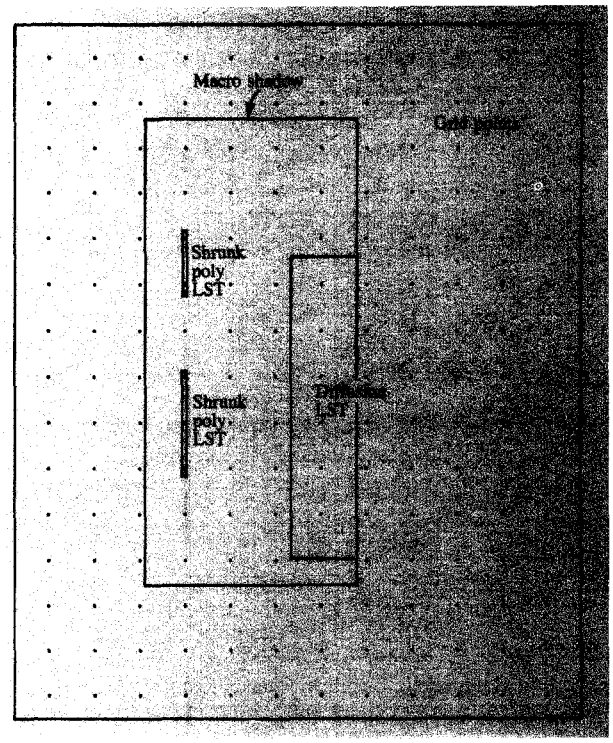

Figure 3

Modified data with shrunk poly LSTs.

isolated from the diffusion wiring by the shadow, but are available for metal wires to drop contacts. The diffusion LSTs model normally: Diffusion wires *or* contacts may touch them.

Alteration of model—example

In a situation where vias are not permitted unless they are greater than N microns away from macro shadows, block-via shapes could be designed which were N microns larger than the macro shadows. However, this can be avoided by taking all the macro shadow rectangles, expanding them by the N microns, and using those data to block vias in the via rules map. This kind of processing has been used on many chips.

- *KWIRE router*

The heart of the KWIRE system is the KWIRE router. The router implements general geometric concepts which are used in specific ways to model wiring problems. The search algorithm is based on Lee's Method, although heavily modified for faster run times. The most often used features of the router are discussed below.

Weight regions and costs

Each grid point on each wiring plane is associated with a weight region. A weight region is an area in which a certain set of wiring costs is in effect. The weight region type is an inherent property of each grid point and is encoded in the bit pattern of each byte in the chip map.

Up to four weight region types may be used in a wiring plane, with any number of regions of a given type. The size of a weight region can be as small as a single grid point or as large as the entire wiring plane. The wiring costs assigned to the same type regions in different wiring planes can be different.

The user assigns wiring costs to the different weight region types. These costs are integers ranging from 1 to 31. They represent the penalty for movement horizontally, vertically, up, and down. A cost of 1 represents no penalty. Increasing the cost increases the penalty. Zero cost represents infinite penalty and acts as a directionally dependent blockage. This feature is used when wrong-way moves must be prohibited.

Use of the weight regions and their associated costs gives the user great flexibility in determining the router behavior. The double-level-metal chip shown in **Figure 4(a)** is a typical example. The user wanted most of the global connections to reside in the T-shaped path, and be on first-level metal. This required a strong bias against wrong-way moves in the T. Some slight bias against wiring in the local areas would help keep global nets in the T path, while allowing local connections to wire normally. In addition, the local power

Table 1 Wiring costs assigned to weight regions used on the layout shown in Fig. 4(a).

First metal					
		Vertical	*Horizontal*	*Up*	*Down*
Weight region	0	05	02	07	00
	1	06	01	08	00
	2	02	05	07	00
	3	01	06	08	00
Second metal					
		Vertical	*Horizontal*	*Up*	*Down*
Weight region	0	02	05	00	07
	1	01	06	00	08
	2	05	02	00	07
	3	06	01	00	08

buses were on first-level metal and ran vertically in the top portion of the chip, but horizontally in the bottom portion. Local first-level metal should be parallel to the power buses. The user wanted to limit the number of vias used, and allow wrong-way wiring at higher penalty. The preferred direction for second-level metal was always perpendicular to that of first-level metal.

Figure 4(b) shows the weight regions which were employed. The user outlined the weight regions by using a graphical aid which displays the chip data on a 3277 Graphics Attachment [11]. The wiring costs used are shown in **Table 1**.

If desired, the wiring costs associated with the weight regions may be assigned on a per-net basis. For example, in a technology using diffusion and two metal layers of wiring, it is desirable to keep critical nets from ever using diffusion, while allowing other nets to use it. The costs-per-net feature is used to implement such a restriction.

Restricted search

A Lee router tends to search too thoroughly for a path. It spends much time searching in the wrong direction. To speed up the search, a restricted-search feature is offered. It makes use of a *search box*. The size of the search box is initially the least enclosing rectangle which contains the portion of the net that is already wired and the LST to be wired. The initial coordinates of the search box are calculated by the router prior to wiring the LST. This initial search box is then expanded. The expansion is controlled by three user-specified router inputs: a minimum expansion amount, a maximum expansion amount, and an expansion multiplier. The expanded search box is then used to limit the search area **(Figure 5)**. This approach results in small search areas for LSTs which are close together, and larger search areas for LSTs which are more spread out.

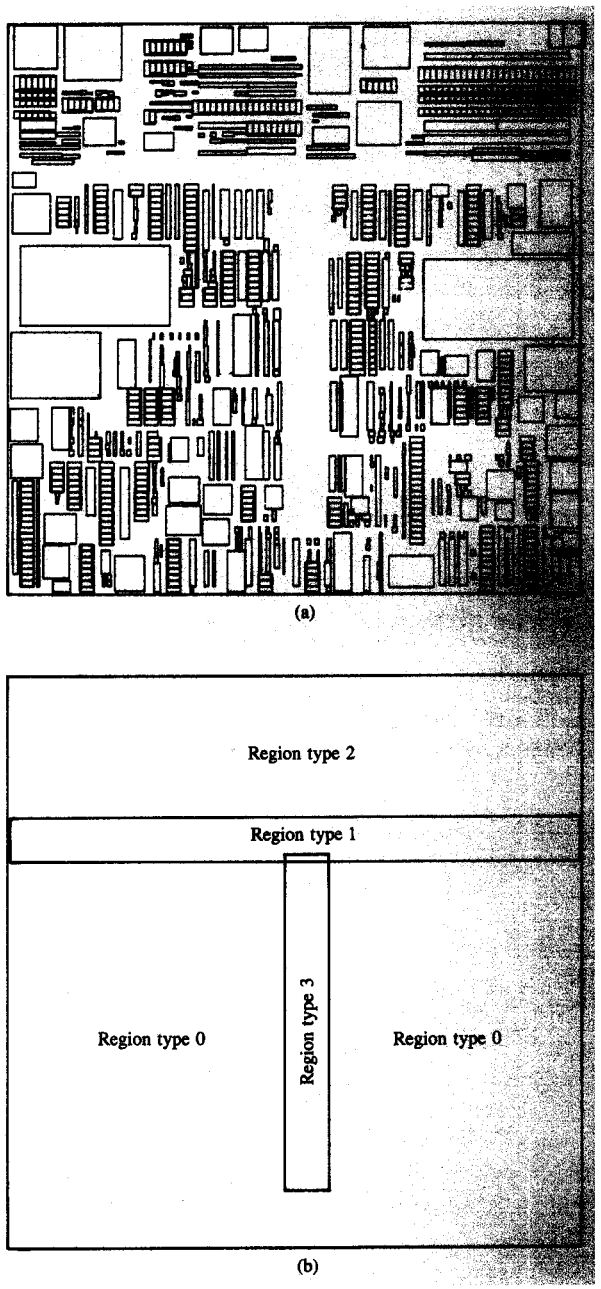

Figure 4

(a) Layout which requires specific routing behavior. (b) Weight regions used on the layout in Fig. 4(a).

The search box is dynamically placed into the chip map so that it appears to be a blockage and limits the search. The search algorithm cannot differentiate between blockages and the search box. After the LST has been processed, its search box is removed from the chip map.

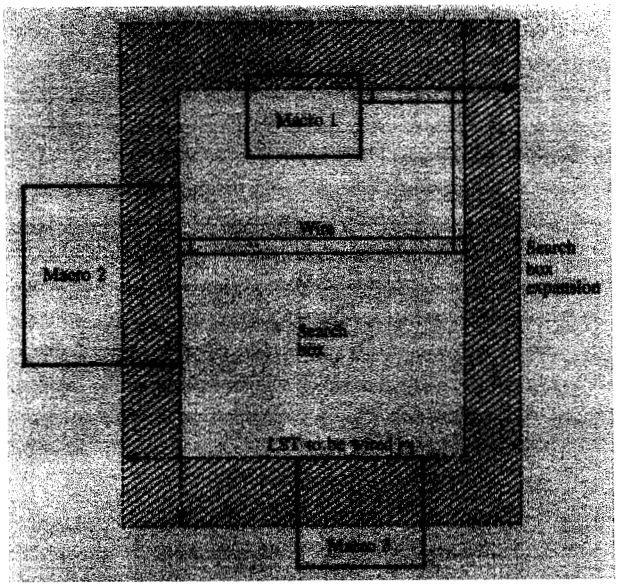

Figure 5

Initial search box and expansion.

Search boxes are not the best solution, but they have been effective in practice and entail very little overhead.

Continued LST (feedthrough)

The router permits a "single" LST to consist of multiple pieces. Such a multiple-piece LST is called a *continued LST* or *LST family*. A continued LST can be used to represent a polygonal LST. (Remember that all polygons are fractured into rectangles during the first phase.) However, the router permits the pieces of a continued LST to be discontiguous and/or on different wiring planes. This property of continued LSTs is used to advantage in the concept of a *feedthrough*.

As was previously stated, the router tries to connect all like-named LSTs. Often, a particular net can be connected to a macro in more than one location. Thus, there could be many LSTs on a macro which have the same netname and which are all electrically common. It would be advantageous to let the router decide which LST it prefers. Wiring one of these LSTs would wire all, and the net could continue out the other LSTs, if need be. At best, this offers the advantage of using the (invisible) macro internal connections to help wire some nets, at a saving in global wiring space. Nets could "feed through" the macro instead of going around it. At least the router would get an alternate way to achieve a connection. However, since the LSTs would be (obviously) disjoint, the router would try to connect them all together **[Figure 6(a)]**. A convention was defined in which internal continuity between LSTs of the same net is designated by their connection with a line on a special, nonmanufacturable level. The model builder traps these lines during model creation, and generates a continued LST from the disjoint LSTs. This type of continued LST is called a feedthrough. The router then uses the continued LST in its regular way, and the benefits are realized **[Figures 6(b), 6(c)]**.

A problem occurs when the internal path between feedthroughs has poor electrical characteristics. It would then be incorrect to use the internal path in continuing the net. The router supports a type of continued LST which may be wired at any point, but which is not used to continue the net. These are called "soft" feedthroughs.

Ports (wirethroughs)

The router supports an extension of the continued LST, called a *port family*. A port family consists of a set of LSTs which belong to *no* net, which provide continuity among all the LSTs in the set, and which may be assigned to any net as part of the wiring for that net. These properties are used in the concept of a *wirethrough*.

Some circuit macros may have some available wiring paths inside. One way to use these paths is to create a fractured shadow shape for the macro and allow the router to lay the wire **[Figure 7(a)]**. This approach has two main disadvantages: The creation of disjoint shadows is not easy, and the available free path must fall on the wiring grid.

An alternative approach uses the port family in a form called a wirethrough. During macro creation, real wire is placed in the free path. This wire connects to LSTs (with *no* netname) which are placed on the macro boundary. A line on another special, nonmanufacturable level is used to connect the LSTs. The model builder traps this construct and models it as a port family. The router uses this prepared path through the macro when it wires nets **[Figure 7(b)]**. The port family becomes the property of the net which uses it.

Iterative mode

If the router cannot find a path for an LST, it attempts to find a path by allowing the new wire to cross (short circuit) other wires at very high cost. If a path is now found, any crossed nets are deleted from the model, but the deleted data are saved for a while. After the deletions, the router finds a new path for the new wire, taking advantage of the additional free space provided by the deleted wires. (The path found by allowing shorts is not necessarily used, because it may have too much "wrong-way" wire. The additional path search helps maximize the number of useful wiring channels by limiting "wrong-way" wiring.)

Once the new wire is placed, the saved data from the deleted nets are checked against the new wire. Wherever a crossing (short) would occur, a "super dot" is created and assigned to the new wire. Super dots prohibit other wires from ever crossing the point which they protect. They

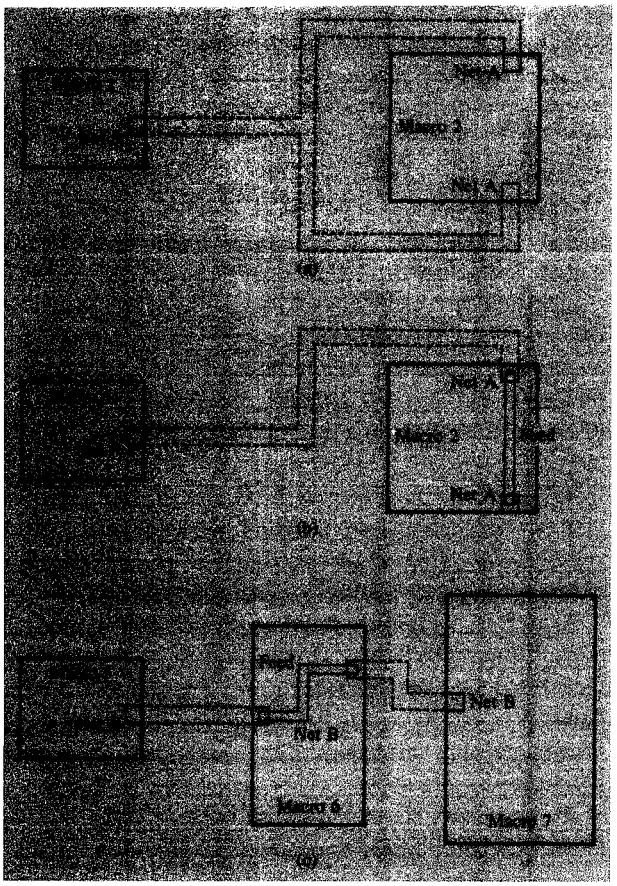

Figure 6

(a) Without feedthroughs, the router connects all like-named LSTs together. (b) Alternate LST specified by feedthrough. (c) Using internal macro continuity to help wire a net.

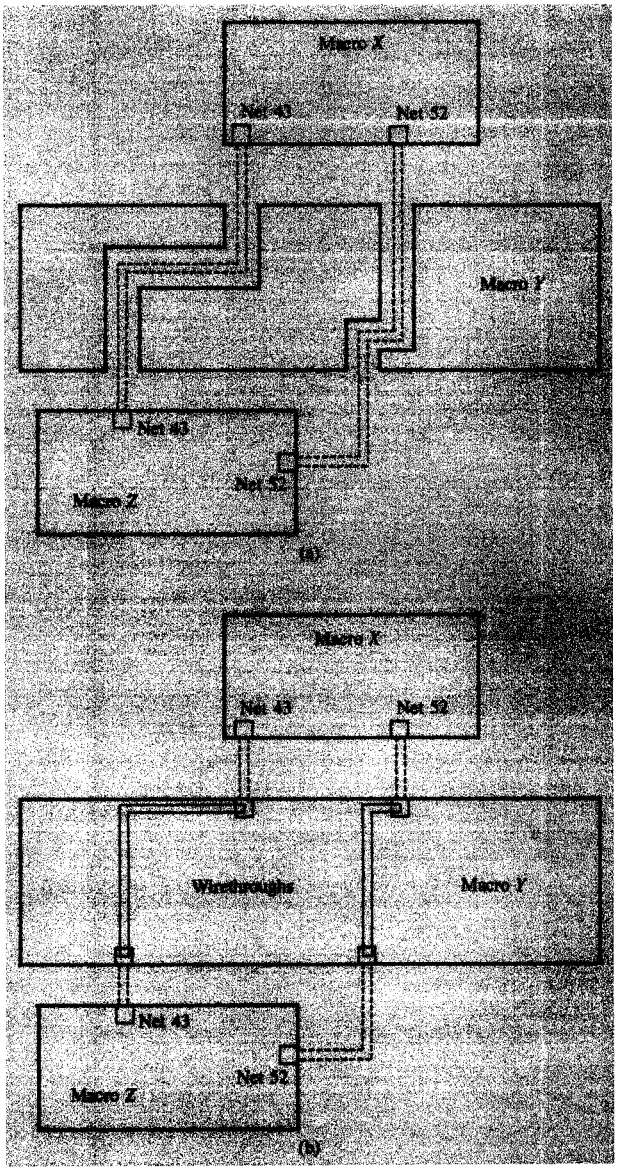

Figure 7

(a) Wiring space through a macro specified by fractured shadow. (b) Wiring space through a macro specified by wirethroughs. The router assumes internal continuity.

prevent loops from occurring by forcing subsequent wires to take alternate paths.

Those portions of nets which are not crossed may qualify for being replaced intact into the model, without further path searching. After these "put-backs" are determined and placed, wiring continues. Alternate paths for the deleted wires are found before new wires are attempted.

Probes and ghosts

When the router fails to find a path for an LST, it can flag that LST as a "fail" and ignore it during the rest of the run. The user has no way of knowing how much of the path the router could complete. This approach would have dire consequences if the fails had to be manually imbedded. However, the router can be run in a mode which does provide the necessary information.

When a fail would occur, the router tries to wire the path by permitting shorts to occur—at high cost to maximize the length of path taken in legal material (the "probe" portion) and minimize the length of path causing shorts (the "ghost" portion). "Probe" and "ghost" segments are placed on different layers than "real" segments so they can be easily identified (this strategy is especially vivid on color graphics systems).

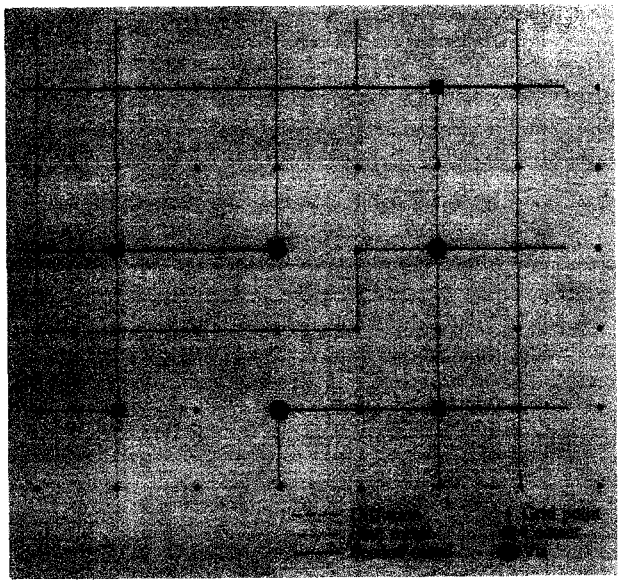

Figure 8

Router results: wires are lines without width, and vias are points.

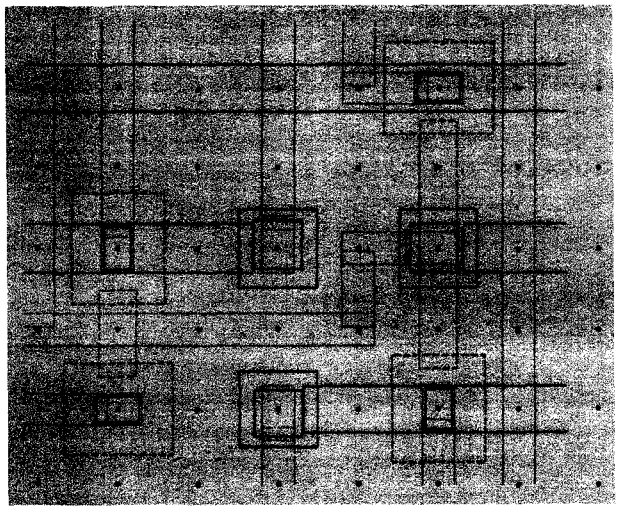

Figure 9

Postprocessor output: all materials have the correct width and overlap. Via cells have been selected and correctly rotated.

The probe-ghost paths are used as a guide in the manual imbedding process. Such use drastically cuts down imbed time because the person doing the imbedding has a path to follow and does not have to laboriously find one. Usually, only a small portion of the probe-ghost path is ghost material. It has been found that the users can merely "stitch" the probe segments together to complete the path, with minimal impact on surrounding wires.

• *Postprocessing methods*

The purely geometric router results must be transferred back into physical constructs to be useful. The postprocessors provide technology-dependent ways to achieve this.

Like materials can be made to overlap at wire-to-wire, wire-to-LST, and wire-to-via junctions. The amount of overlap in different layers at different kinds of junctions is programmable. Such overlap rules are required in some technologies to allow for a margin of error in mask preparation.

Adjacency rules between via shapes and wires, or between vias, can be satisfied by the design of vias which are appropriate for given situations [12, 13]. A postprocessor can select the correct via and the orientation for it.

Figure 8 shows some router results from a three-wiring-level problem. The contact and via locations have been rendered as large dots to make them visible. **Figure 9** depicts the result of postprocessing the data of Fig. 8. Note the material overlaps at rotations of contacts and vias **(Figure 10)**.

The postprocessors are built from a standard subroutine library. Via selection and orientation is table-driven. Each postprocessor has its own set of via tables.

• *Some results*

Since 1979, KWIRE has been successfully used on 17 chips at IBM Kingston, and several more company-wide. These chips range from small designs to custom bipolar [6] and FET microprocessors using both IBM and vendor technologies.

Statistics from eight custom chip designs are presented in **Table 2**. They appear in chronological order, and span three versions of KWIRE.

The first three chips were bipolar components with two metal layers available for wiring. Analysis of these results led to large performance improvements. The number of weight region types was increased from one region type per wiring level to four region types per design. User-selectable via adjacency rules were implemented.

The next two designs took KWIRE into a new realm: FET, single-metal chips. The four weight regions just developed were of immediate use. The first complex postprocessor was built. Model-building utilities were created to handle situations such as those shown earlier in Figs. 2 and 3. The results from these designs suggested improvements in the iteration strategy, performance enhancements, the restricted search box, probe and ghost wiring for imbed assist, and implementation of four weight region types per wiring level.

The next design, an FET processor of over 21 000 circuits, utilized all the features of KWIRE at that time. Table 2 only shows statistics for the global wiring of this chip; many of the macros themselves were wired with KWIRE. Continued LSTs were implemented for this chip. This design was done at IBM Boca Raton.

The last two chips were processed with the latest version of KWIRE. "Soft" feedthroughs and costs-per-net were implemented for these chips.

To summarize, user experiences with each design suggested improvements in KWIRE, KWIRE utilities, and the way the KWIRE system is applied to a problem.

• *Associated tools*

A 3277 Graphics Attachment [11] program which can display the router output and most of the input and model files is part of the KWIRE tool box and is heavily slanted towards KWIRE and its users. It provides typical graphics viewing functions and special functions related to KWIRE (e.g., a weight regionalization mode and super-dot viewing).

KWIRE runs both interactively and in batch mode. The interactive mode is supported by a comprehensive set of menus. The commonly used utilities and model modification programs are included on the menus. The KWIRE router has its own interactive environment and a large set of wiring commands. On-line help menus can be called upon when the router is run interactively.

Batch execution is supported by rules files and preset procedures which are created for each chip design. The user need only remember the name of the appropriate rules set to run a complete KWIRE job. This has become the preferred technique.

The KWIRE router has an interactive debugging mode which is used by the program authors during debugging and new feature verification. The debugger has been a great help in tracking down program errors which users have found in KWIRE.

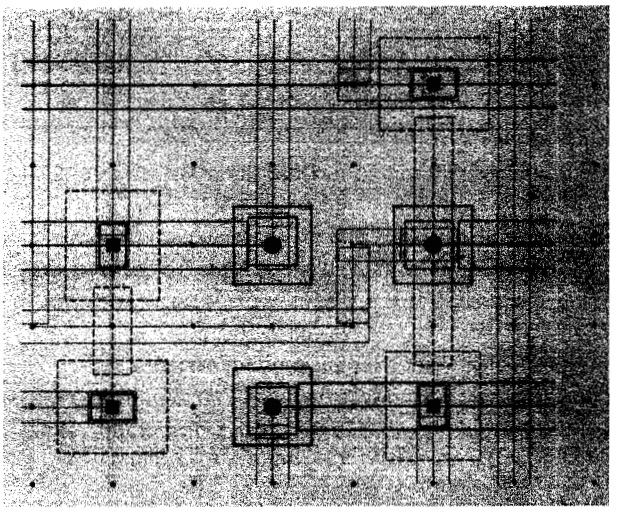

Figure 10

Overlay of Figs. 8 and 9.

Conclusions

KWIRE's success has demonstrated that the removal of technology and methodology constraints from a wire router is a good strategy when multiple technologies and methodologies must be accommodated. When enhancements are required in the router, they represent general concepts (e.g., port families) which benefit not only their requestor, but also subsequent users. This spreads the development cost of general router features over many users.

As technology advances, new tools will take KWIRE's place. Presently, hierarchical tools built around a central database are being studied. The ideas of user

Table 2 Statistics for eight chips wired with KWIRE. Circuit counts assume 3.5 devices per circuit.

Chip	*Levels*	*Metals*	*Circuits*	*Circuit area (%)*	*Nets*	*LSTs*	*Fails*	*Imbed time (days)*
Bipolar processor component	2	2	4100	40	1383	3689	27	14
Bipolar processor component	2	2	3975	40	1147	3095	19	3
Bipolar processor component	2	2	4025	40	1141	3003	0	0
FET processor	2	1	6772	43	816	2558	46	14
FET data channel	2	1	5659	40	971	2878	26	7
FET processor	2	2	21428	57	1000	3910	0	0
FET display driver	2	2	5000	40	1315	2150	0	0
FET communications adapter	3	2	8473	47	2034	6521	0	0

reconfigurability and removal of dependencies—KWIRE's toolbox approach—will become an important feature of these new tools.

Acknowledgments

The author wishes to acknowledge the contribution of C. B. Killen in the initial development of KWIRE and associated tools while at Kingston, and the subsequent developments and contributions he made while at Boca Raton. Over the years, many users of KWIRE have made suggestions and comments which resulted in substantial program improvements. Though space does not permit comprehensive acknowledgment of all these individuals, the author wishes to thank them for their contribution. Thanks go to G. J. Tuma of IBM Kingston for Figs. 6 and 7, and to all those who reviewed the manuscript and suggested changes, especially T. A. Elmendorf and S. S. Hultquist for reviewing several drafts.

References and notes

1. *IBM Virtual Machine/System Product: CMS Command and Macro Reference,* Order No. SC19-6209; available through IBM branch offices.
2. *IBM Virtual Machine/System Product: CMS User's Guide,* Order No. SC19-6210; available through IBM branch offices.
3. *IBM Virtual Machine/System Product: System Programmer's Guide,* Order No. SC19-6203; available through IBM branch offices.
4. *GL/I Language Specification,* Order No. GE45-1127-0; available through IBM branch offices.
5. Anthony Correale, "Physical Design of a Custom 16-Bit Microprocessor," *IBM J. Res. Develop.* **26,** 446–453 (1982).
6. K. F. Mathews and L. P. Lee, "Bipolar Chip Design for a VLSI Microprocessor," *IBM J. Res. Develop.* **26,** 464–474 (1982).
7. W. C. Finger et al., *Macro Assembler Process for Automated Circuit Design,* U. S. Patent 4,377,849, March 22, 1983.
8. For this discussion, a "contact" is an ohmic connection between the first layer of metal and diffusion or polysilicon.
9. A "via" is a ohmic connection between two layers of metallization. In this paper, the word "via" is used to mean "vias or contacts" unless stated otherwise.
10. C. Y. Lee, "An Algorithm for Path Connections and Its Applications," *IRE Trans. Electron. Computers* **10,** 346–365 (1961).
11. *IBM 3277 Display Station, Graphics Attachment RPQ 7H0284, Custom Feature Description,* Order No. GA33-3039; available through IBM branch offices.
12. A versatile set of contacts for a single-metal technology was designed by R. McClurg.
13. A versatile set of contacts and vias for a double-metal technology was designed by A. Correale and G. J. Tuma.

Received December 8, 1983; revised April 26, 1984

Peter C. Elmendorf *IBM Communication Products Division, Neighborhood Road, Kingston, New York 12401.* Mr. Elmendorf is a staff engineer in the Physical Design Development Department. He joined IBM in 1979. He received the B.S. in electrical engineering from Rensselaer Polytechnic Institute, Troy, New York. He is currently enrolled in the graduate work study program at Syracuse University. Mr. Elmendorf is a member of the Association for Computing Machinery and an affiliate member of the Computer Society of the Institute of Electrical and Electronics Engineers.