W. E. Donath

Stand-Alone Wiring Program for Josephson Logic

This paper describes a channel routing wiring program and its interface to the user. Of particular interest are its interface facilities, which permit manual update of the routing, pre-routing, and incremental routing. A hierarchical organization of the logic is feasible, which permits moving of complex entities, such as latches, adders and others, as complete entities. The internal wiring of these entities could either be done manually and be fixed before layout, which would be desirable when the wiring was used as a delay line, or could be left to the wiring program, which would route them more flexibly. The features above are made possible by the special-interface organization used here. In this interface the pins on the devices can be directly addressed, relatively addressed, and indirectly addressed; a simple macrocompiler permits the hierarchical organization of the data.

Introduction

The Josephson computer project is a technology development project with relatively high-density chips. To prove the feasibility of this technology the development of a signal processor prototype has been set as an objective and the wiring of LSI chips is a requirement for successful design of such a processor.

It was considered desirable to develop a channel routing program for the Josephson technology, so that complex chip designs could be accomplished with reasonable effort. Since the effort available for this part of the project was limited, it was decided first to develop a stand-alone wiring program, and then to add a sophisticated interface to the wiring program—a direct consequence of the constraints in manpower and time. The general organization of this program is as shown in Fig. 1; the following programs are involved:

1. The facilities generation program, through which all the devices, device terminals, chip terminals, channels and tracks are represented. The input data consist essentially of array specifications of the devices.
2. A macro expander, where parameter substitution and string concatenation are possible. This expander essentially carries out text substitution.
3. The data entry program, where connections, routings, device usages, and operations on the data base (*i.e.*, connection and wiring file) are entered. In addition, this program performs serialization of nets. It also permits control over the length of the wires.
4. The termination resistor assignment program, where the termination resistors are assigned.
5. The global and local wire routing programs, which essentially do the routing.
6. Printer plotting and mask generation interfaces.

All communication *into the data base* (*i.e.*, connection and wiring file) is done via the *data entry program*; the termination resistor assignment and wire routing programs communicate their results in a format acceptable to the data entry program. This feature ensures a higher level of data integrity for the file and, at the same time, makes program development simpler. Of course, the language used for the data entry program is such as to permit updating of the data.

In the Josephson technology, circuit delays [1] as low as 13 ps have been measured (nominal 35 ps), which implies that wiring delay is, relatively speaking, a far more critical factor than in other technologies. In this program it is possible to selectively constrain individual wires to their minimum enclosing rectangles, so that wires on critical paths can be controlled as far as length is concerned. (This is of benefit only, of course, if only a fraction of the wires on a chip are on critical paths.)

In particular, we have a very flexible design methodology here regarding the design of the wires internal to the "macros," whereby we mean circuit complexes. We may develop so-called "soft" macros (where the wires are routed by the program), "hard" macros (where the internal wiring is prerouted by man through the input facility of the wiring program), "superhard" macros, which block channels in the program, and of course many varieties of "intermediate" macros. This concept permits one to choose the wiring mode according to whether technology requirements (*e.g.*, delay lines) or wirability requirements dominate.

We discuss the facilities of the data entry program in some detail, since this portion of the program has the most novelty. It should be noted that in general far more effort is devoted to manipulation of data and ensuring data integrity than to algorithmic processing, so that it is proper to devote considerable attention to this part of the program.

In the second section we discuss the image, the devices, and the wiring program; the third section deals with the input interface—*i.e.*, the macrocompiler and the data entry program, while in the fourth section we describe the input language.

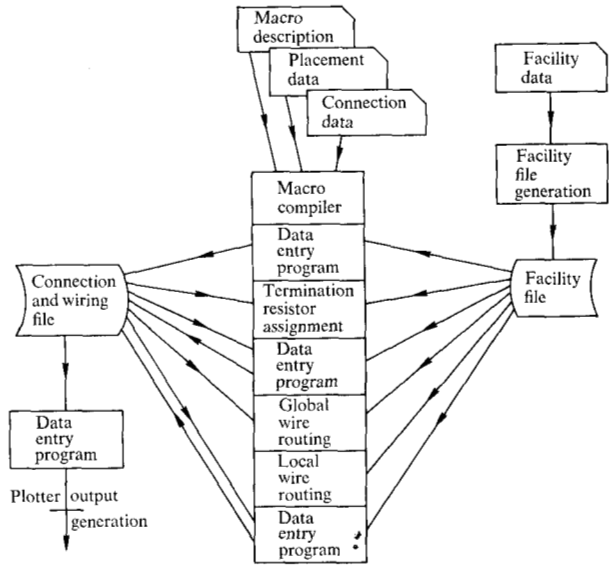

Figure 1 Data flow in the Josephson wiring program. Note that the data entry program is used at many places—it is the only place from which the connection and wiring file is updated and it is also used in interfacing to the printer plotter.

Devices, image, and the wiring program

The image is assumed to consist of an array of cells in a rectangular pattern, where each cell has, on its periphery, its logic service terminals. Superimposed on this array of cells are the wiring channels; typically these may be in the channels between the cells as conduits for the long connections between cells, or they may serve as accesses from the channels to the terminals of the cells. Occasionally, connections internal to the cell may be made through such tracks or the access may be to a service terminal outside the cell area (such facilities are also made available through the facility generation program). Figure 2 shows a portion of the image as represented by the printer-plotter program. Basically, there are two kinds of circuit devices: a three-junction interferometer, which acts mainly as an OR-gate, and a two-junction device, which acts mainly as an AND-gate (both these devices serve other functions, too, and may be parts of latches and inverters). There are two control lines crossing the three-junction devices; a current on either of these two lines will cause the device to switch and emit a current. The two-junction device requires two current sources on its two inputs. If just one of the two sources is on, the device will be conducting and the current will be grounded; if both sources are on, the device will also emit a current, since it will then switch into its voltage state. An OR-AND device is built by connecting two three-junction devices to a two-junction device, a four-way OR-AND device is built by connecting two such OR-AND devices to another two-junction device. The four-way OR-AND device makes an ideal multiplexor if inverse boolean notation is used. These devices are defined as macros.

Since the devices are current sources, the devices they drive must be ordered in a serial chain. Termination is provided by special termination resistors, which connect to ground. It is to be noted that the four-way OR-AND device drives two serial chains, and that two three-junction interferometers can be put together to form a four-way OR device, which also drives two chains.

Other devices available to the program include power resistors as well as termination resistors. These the program may connect up through device personalization instructions, or their I/O pins may connect to other pins—*i.e.*, the pins of the devices.

The wiring program works in two stages: a channel routing program followed by a line-packing procedure. The channel router routes two-point connections, using a restricted-rectangle maze-running procedure with a cost function for using wiring space, which becomes very steep at the point where the capacity of the space is being fully used. This means that the global wiring program tries to avoid situations where the wiring space is used to

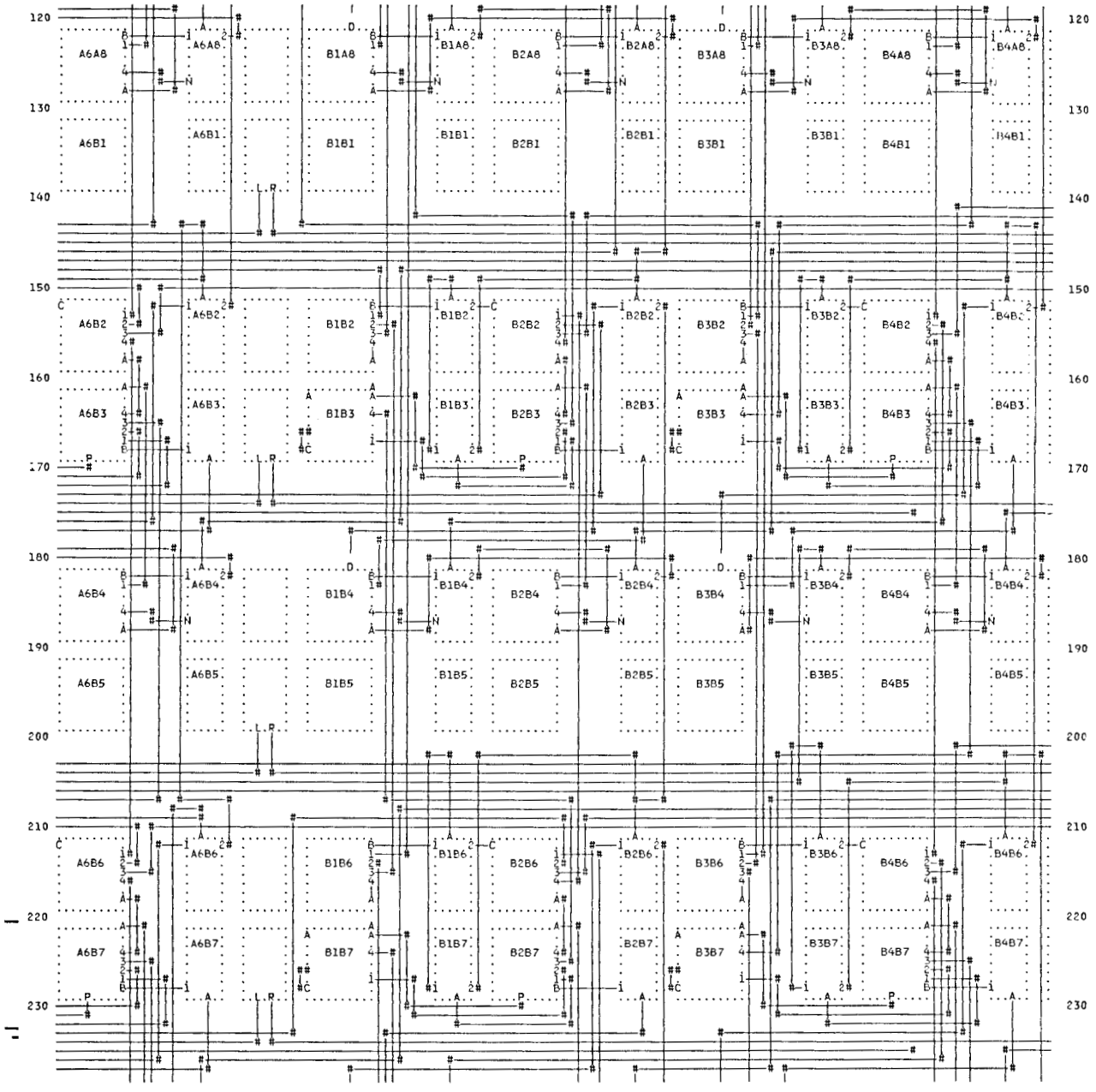

Figure 2 Printer plotter outprint of wired image; dots outline cell spaces, four letters inside cells denote area name (note that each name covers two areas), characters on periphery denote I/O pins of areas, and # symbol denotes connections between planes. The small areas hold the two-junction interferometers, while the large areas contain the three-junction interferometers.

excess, but does not necessarily guarantee that this is the case. The algorithm is described in more detail in Appendix C of Ref. [2]. The major difference between the program described here and the program developed by Chen *et al.* [3] is that the Chen program counts crossings of lines, while this program counts the usages of a channel segment.

The global wiring program is followed by a simple line-packing program, which does the detailed assignment of wires to tracks.

Input facilities

An essentially new feature of this program is its input facility, which permits the definition and flexible movement

of "macro devices," whereby a macro device consists of several devices connected together. The language developed here—which is presented in its essential features—permits a pre-routing of these intra-macro connections (*i.e.*, if the connection is to be a delay line) or allows the routing to be carried out by the wiring program. The latter is more efficient in terms of wiring space, because it allows the wiring program more flexibility.

The essential features of the language for the data entry program include its different addressing capabilities for the I/O terminals of the primitive devices—*i.e.*, direct, relative, and indirect—as well as the addressing capabilities for the wire locations (direct and relative). Also important is its capability for giving each device a function, where the device is addressed by a pin. Several devices may be associated with each cell. Examples are given in the section on the input language. Furthermore, wire may be routed which acts only as a blockage, permitting devices to extend into the wiring area and block channels. Another capability of the data entry program, namely automatic serialization of nets, is dependent on the way the interferometers work—*i.e.*, that current enters the loop of each interferometer and leaves on the same side (discussed in more detail in the previous section). The program used here gives an approximate solution to the traveling salesman problem. The algorithm is described in detail in Ref. [4]. It uses an assignment procedure to generate several tours, which are then merged by a heuristic procedure.

Of considerable interest is the interaction between the macroexpander and the data entry program, which permits the separate description of a placement list and the logical net list. In the placement portion the user provides the system with the macro function (which implies a placement of the components of that macro and a description of connections internal to the macro), the logical name of the block, and a physical pin position. The physical pin position is used as a reference position for the internal connections of the macro as well as the I/O pins of the macro; the logical block name for the macro is concatenated with pin identification characters and used to generate indirect addresses for the I/O pins of the macro, which are used later in the logical net and connection lists. In this way placement information and the logical connection information are separated and it is possible to move complete macros about.

Finally, control over the length of the connection is made possible by giving a special control entry for each wire whose length is to be controlled.

Input language

We describe the structure of the input language in this section: first, the notation for addressing pins, devices, and cells; second, the operational features of the language and the method for describing routing; and last, the macro features.

• *Addressing features*

As described previously, the image consists of an array of cells which either contain devices or are used for wiring. In the images used here, even-numbered cells contain devices, while the other cells are used for routing. Typically, a four-character name is used to address the device cell, with the first two characters referring to the column and the second two characters to the row. A fifth character is used to identify the device, and a sixth character the pin. An address such as B2C3A1 refers to column B2, row C3, device A, pin 1. Let us assume that the reference address is B2C3A1; a relative address might be *4 -2 BA where 4 is the number of columns one moves over, -2 the number of rows, and the last two characters of the reference address are replaced by BA. This relative address might refer then to pin B4C2BA (remember that the cell counts include wiring channels). The general notation then is

*Δx Δy *final characters* for defining a relative address.

These instructions are provided for setting the reference address:

LOC *address* (direct, relative, or indirect)

PUSH *address*

POP

LOC sets or changes the address currently on top of the stack, which is the current reference address. PUSH adds another address on top of the stack and makes it the current reference address. POP removes an address from the top of the stack. Indirect addresses are generated by the command

$ *name pin-address;*

the address *name* preceded by a $ sign becomes then the address of the pin.

• *Operational features*

Devices are given a personality by a two-character personality specification; a command

DEV *pin-name personality;*

assigns the device with the pin on it the personality implied by *personality*. Information regarding this personality assignment is fed to the mask generation interface and the specified personality is generated for this device.

Wires are specified by either the SER, SERZ, or M command; these are used as follows:

SER (SERZ) *source pin* $pin_1\ pin_2 \cdots pin_n$;

where the command SERZ will ensure that pin_n stays as the last pin. The command

M $pin_1 pin_2$ [Other information, routing]

makes a connection between pin_1 and pin_2 and allows a routing to be described for that connection. The instructions given in the routing section consist of R and //, where R begins routing and // ends it, and in between is a set of wiring positions of the cell image. These are as follows:

A—refers to the first pin.
B—refers to the second pin.
P *x y tx ty p*—route a wire to cell *x y* and track *tx*, *ty* on plane *p*.
/—change plane in routing to next position.
X *x tx*—route a wire to channel *x*, tracks *tx* in horizontal direction route.
* Δx *tx*—route Δx channels over to *tx* track.
SP—is used like P, but starts a new connection.

Under other information can be a T command, which, followed by a "0," limits the wire to the minimum rectangle.

• *Macro instructions*

The keyword *macro* is reserved for the macro compiler, which precedes the data entry program. It is used as follows:

MACRO *name* $(p_1\ p_2\ p_3 \cdots p_n)$

Text (with references to $p_1 \cdots p_n$)

ENDMAC

and defines that *name* must be replaced by the *Text*. Concatenation of two characters strings *x* and *y* is denoted by

x | | *y*.

• *Usage*

The following describes a two-way OR-gate:

```
MACRO ORZ (N,P) /*N-name of logical function, P-position /*;
LOC P;
DEV     * 0 0 0A; /* 3-j interferometer used as 0A device */
$N | | A  * 0 0 AA; /* OUTPUT PIN */
$N | | 1  * 0 0 A1; /* INPUT PIN 1 */
$N | | C1 * 0 0 A4; /* CONTINUE ON PIN 1*/
$N | | 2  * 0 0 A2; /* INPUT PIN 2*/
$N | | C2 * 0 0 A3; /* CONTINUE ON PIN 2*/
ENDMAC
```

The following is the placement of three OR-gates:

```
ORZ(GA, AAAJAA);
ORZ(GB, AABJAA);
ORZ(GC, AACJAA);
```

OR-gate GA now feeds gates GB and GC:

```
SER $GAA $GB1 $GC1;
```

Note that only logical addresses are used here; one could also make the serialization manually:

```
M $GAA $GB1; M $GBC1 $GC1;
```

Large-scale study and conclusion

To eliminate errors from the program and test the program for a large test case, logic for a 16 × 16-bit cross bar switch was designed together with a demonstration image; the logic consisted of 64 latches, which held the information for the switching of the signals, plus 16 decoder-selection networks, each of which selected one of the 16 input signals on the basis of the decoded signal from four latches. The latches each required six three-junction and six two-junction interferometers; the total logic required 1100 three-junction and 600 two-junction interferometers and used about 4300 connections. The demonstration image held 1536 three-junction and 1536 two-junction interferometers in a 24 × 64 array. About 250 wiring tracks were provided in the vertical direction and 320 in the horizontal direction. There were 13 connections not routed after running the program. Pre-routing the internal connections of the latch increased that count to 25. In general, it seems that pre-routing of the macros has a negative impact on the wirability of the chip since it limits the degree of flexibility available to the routing program.

Computing time for a run was on the order of eight minutes on the IBM 370/168 computer under VM/CMS; however, the program required on the order of two megabytes of storage. While this is acceptable, the heavy storage requirement was caused by two separate aspects of the program. First, the facility file had a separate entry for every device and logic service terminal, which made for easy programming, but could be avoided with some thought; second, the routing programs used the facility file (in addition to their internal files), which could also be avoided with some effort.

Acknowledgments

I want to thank G. Koppelman, M. Ullner, and B. Stahl for their efforts in developing certain parts of the system

described here: G. Koppelman for the interface to manufacturing and the Gerber plotter, M. Ullner for the traveling salesman program, and B. Stahl for optimizing the program on computing time performance. I wish to thank my management for encouraging and supporting this work, L. Kugel for many suggestions regarding the requirements for this program, and R. Partridge for interesting discussions regarding this program.

References

1. T. Gheewala, "Design of 2.5-Micrometer Josephson Current Injection Logic (CIL)," *IBM J. Res. Develop.* **24,** 130–142 (1980).
2. W. E. Donath, W. R. Heller, and W. F. Mikhail, "Prediction of Wiring Space Requirements for LSI," *Proceedings of the 14th Annual Design Automation Conference*, New Orleans, LA, June 20–22, 1977, pp. 32–42.
3. K. A. Chen, M. Feuer, K. H. Khokhani, N. Nan, and S. Schmidt, "The Chip Layout Problem: An Automatic Wiring Procedure," *Proceedings of the 14th Design Automation Conference*, New Orleans, LA, June 20–22, 1977, pp. 298–302.
4. N. Christofides, "Worst Case Analysis of a New Heuristic for the Travelling Salesman Problem," *Management Sciences Research Report No. 388*, Carnegie Mellon University, Pittsburgh, PA, 1976.

Received April 20, 1979; revised September 5, 1979

The author is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.