A Wiring Tool for Ultracomputer

Guoying Chen *

Ultracomputer Note #176
December, 1990
Revision 2.0, January 1992

Abstract

During the final stage of computer system design and implementation, using some flexible and convenient tools can save a lot of time and efforts. For example, when all subcomponents of a design are carefully placed on a board, and the design schematics and some necessary data are given, a wiring tool is strongly needed so that the wiring can be done easily and automatically. However, it is a practical question how to route each signal wire for *many* components on *any* given board under various constraints and complicated situations.

This report describes a general-purpose wiring tool which meets such a kind of demand, and is used particularly for Ultracomputer prototyping and implementations. The most important part of the wiring tool is how to solve the routing problem for a signal net. We reduce it to a Traveling Salesman Problem (TSP) with various constraints. The wiring tool employs a heuristic and fast TSP algorithm that is able to produce optimal results in high probabilities.

The wiring tool program consists of three parts: *Preprocessor*, *wiring-man*, and *output generator*. The tool package offers a flexible interface to users, and is able to handle any new coming boards, chips, and other technologies *semi*-automatically with minimal manual intervention.

In addition to describing the features and the method of implementation of the tool, we also provide useful information and discuss several issues for its future upgradings.

*Supported in part by U.S. Department of Energy under grant number DE-FG02-88ER25052 and in part by U.S. National Science Foundation under grant number MIP-8915488

1 Introduction

In the process of designing large computer systems (like Ultracomputer [6] that is to interconnect up to hundreds and thousands of processors and memory modules), many chips are selected and used in building the components of processor elements (PE), memory modules, network interfaces and I/O interfaces. In these components, different signals (such as data signals, address signals and control signals) generally traverse not only from one chip to another chip, but also from one component to another component.

When the design work is finished(e.g., the schematic layouts are generated) and the chips are carefully placed[1], how to route the wires of these signals under various constraints and requirements becomes a practical question. We may have to handle a number of problems. First of all, the chips that we use often have various shapes with various technologies. For example, possible chip types are: Dual Input Package (DIP) type, Pin Grid Array (PGA) type, Adapters (ADP), Connectors (CON), and so forth. Secondly, the boards on which chips are placed also have variety of types. Chips can be of various form[2], big or small, regular or irregular. Thirdly, signals may have many physical characteristics. On one hand, signals may be either control, data or address; on the other hand, they could be input, output or bidirectional, either two-state, tri-state or open-collector, and so on. Thirdly, chips are classified in different categories by manufacturing technology (e.g., TTL, or ECL).

A flexible wiring tool package is thus developed to meet the growing needs in the stage of computer system prototyping and implementation. The most important part of the wiring tool is how to solve the routing problem for a net of signals. One of the most important objectives is to route wires of each signal with minimum length, as well as under other conditions and constraints. In this report, it is explained how an approximate and heuristic solution to Traveling Salesman Problem is applied to the routing problem. It is shown by experiments that the algorithm that we employ can produce optimal results in high probabilities.

The wiring tool program consists of three parts (see Figure 1: *Preprocessor*, *wiring-man*, and *output generator*. The Preprocessor analyzes the given pin list files and retrieves information about chip, chip types, pins, and signal connections. Other information is supplied from board description files, chip description files and placement files so that the data are collected and organized in the form of tables and linked lists. The Wiring-man takes the organized data from the preprocessor and applies the heuristic routing algorithm to obtain the minimum length, along with the special care with power and ground pins and signals. The Output Generator postprocesses the results of wiring and generates several files for design verifying and final checking. The final wiring data is then supplied to vendor for *real* wiring.

The tool package offers a flexible interface to users, and is able to handle any new coming boards, chips, and other technologies *semi*-automatically with minimal manual intervention. The features that the tool provides are:

[1]Assuming it is optimized to some degree with respect to areas and other conditions.
[2]Two often used boards are COMBION board and MUPAC type, we we build Ultracomputers.

- Different chip types can be described in a text file. One chip type description is sharable by many chips with the same type.
- Various boards can also be given with a mapping file in text rather than usually by a binary mapping file as usual. Thus, there is no limitation on board type.
- Flexible handling to adaptors.
- Power and ground signals are fairly handled and evenly distributed.
- Solder wash information is provided.
- Connectors are treated flexibly, too.
- Many user command options are available.
- The programs of wiring tool are portable and extendible.

This paper is organized as follows: The features and implementation methods of the tool are fully described in section 3 and 4, after the routing algorithm is presented in section 2. An example demonstrating how the wiring tool works is given in section 5. In section 6, additional information is provided and several issues are discussed for future upgradings of wiring tool.

2 Routing Algorithm

Wiring a net for each signal in the circuit is basically a routing problem. Such a routing problem can be reduced to a graph problem [5]. Given a strongly connected graph of $G =< E,V >$ which represents the connections of a signal, where V is a set of nodes and each node corresponds to a pin of a chip which carries the signal; E is a group of edges, each of which is a pair of (v_i, v_k) for $1 \leq i, j \leq \|V\|$; the edges between a pair of nodes with their distances can be represented by an $m \times m$ matrix; the problem is to find the shortest serial (no branch) connection between $\|V\|$ nodes of the net. At first glance, one might think it is a minimum spanning tree problem. However, it is not really such a case because some other requirements and constraints must be satisfied, which cannot be solved simply by the algorithm of minimum spanning tree problem. For instance, we may need to define some node to be the starting point and some node to be the end point; Other situations like nonsymmetric and non-Euclidean graphs must be required when we assume some paths can only flow in one direction but not in the other (particularly for those nets of signals with ECL technology). Also, some cyclic wirings are required to reduce the transmission latencies (actually our wiring tool has such a function). Thus, our attention is paid so as for the algorithm to be general enough to deal with various routing situations.

Our algorithm is a slightly modified version of the Fencl's [5]. For a given $m \times m$ matrix d of distances (with zero diagonal and not necessarily symmetric) between all pairs of nodes, and a given vector p containing n nodes numbers to be connected referring to appropriate entries in the matrix d, Fencl's algorithm was to construct various connections for one net (not necessarily exhausting all m nodes, hence $n \leq m$) with r trials (the value of r will be discussed later) and to select the optimal one among them. We may have different outcomes depending on different needs. If sn (starting node) = en (ending node) is allowed, the result is a Hamiltonian circuit — traveling-salesman problem; and if the input en is restricted to 0, the result will be a connection from start to any nodes.

Since it is well known that the traveling-salesman problem is an NP-C problem [8] [3], heuristic approaches must be resorted to. Conventional methods are dynamic programming [7], nearest neighbor , and farest addition and insertion approaches [1] [8] [2]. The difference between these methods is how to find the next node to add into the tour and how to use heuristics information to connect it with those nodes in the existing tour. The dynamic programming approach has its drawbacks in recursion since it requires huge amount of CPU power and memory storage (Held and Karp's algorithm requires of the order os $(n-1)(n-2)2^{(n-3)}$ calculations and more than $(n-1)2^{(n-2)}$ storage locations for an n–node problem). The extension to this method is the fast recursive partition algorithm. Such an extension makes it fast in producing *some* relatively long tours. Some other tours with fast implementations include Rosenkrantz et al's nearest insertion and farest insertion algorithms [8]. The nearest addition and farest addition are similar to those, but employs a slightly different rule for adding a new node into the tour. Bently[2] in his experiment also proposes the random method (insertion and addition) which add nodes into the tour in a random order. One may use other heuristics information to speed up the algorithms, such as the minimum spanning tree heuristic and an approximate version of Christofides' heuristic (reviewed by [8]). Supporting these heuristics is beneficial for those special operations such as greedy matching, 2-opting a matching , and computing a degree-constrained minimum spanning tree (in which each vertex can only have a limited number of links).

Fencl's algorithm, like other heuristic algorithms, employs a kind of tour-building technique combined with tour-to-tour improvement. Basically, it performs in two phases: in the first phase, the tour (a sequence of nodes) is built up by successfully inserting not-yet-involved nodes into the tour. For example, if in the middle of the tour-building the nodes $p_1, p_2, \cdots, p_k$ are included in the tour, the next nodes $p_{k+1}, p_{k+2}, ...p_n$ and the arc among the arcs $p_1p_2, p_2p_3, ...p_kp_1$ are chosen so that the tour increment will be minimum, i.e., i, j (for $1 \leq i \leq k, k \leq j \leq n$) are chosen such that $d(p_i, p_j) + d(p_j, p_{i+1}) - d(p_i, p_{i+1})$ is minimum (where d is the matrix of distances). Tour building starts from the arc p_1p_1, where p_1 can be any node, and ends with all n nodes are included in the tour.

It can be imagined that the tour thus built is hardly optimal and for large nets it is far more from the optimum. The second phase of the algorithm is to improve the tour of n nodes by so called k-opt method, where $k \geq 2$, and $n \geq 3$. Improvements includes exchanging k arcs, or links of the given connection by k other links (usually $k = 3$ is good enough). If there are no more k links to exchange for tour improvement, the tour is said to be k-optimal. For example, when $k = 3$, the tour is 3-optimal. The implication of the 3-link exchange is essentially in reinsertions. Consecutive node chains of length l ($1 \leq l \leq n$) are successively tried to be reinserted into remaining links for tour improvements, which actually represent 3-link exchanges (and also 2-link at the same time). A 3-opt tour shows a certain probability to be an optimal one in relation to n. Different 3-opt tours can be achieved if different initial nodes are chosen, which allows us to increase the probabilities of achieving optimal solution.

Thus, by trying r trials with different initial nodes to obtain several possible solutions of the improved tour, we may select the best one as an approximately optimal solution. The result of an optimal solution relies on the empiric probability. If $r > n$, there is little chance for further improvement, since initial nodes repeat and the tour development can be affected only by previous contents of the vector p on which the tour is built. Probability that the tour is 3-optimal is somewhat high (over 95 percent) when obtaining an optimum if $r = 2$ to 5 for $n \leq 10$ and $r = 5$ to 15 for $5 \leq n \leq 30$. In contrast to finding a 3-opt solution from a given random sequence of nodes, the fast building of an appropriate tour in the first phase considerably reduces the amount of work of reinsertions. Our modification to the algorithm is to generalize it to asymmetric problems by setting the distance of the prohibitive path to be ever largest value. For example, if the signal net contains some nodes (pins) which is associated with ECL chips, the run path must visit the various types of pins in the following specified sequence:

```
I/O Output pins (signal coming onto board) --> output pins --> input
pins --> I/O input pins (signal leaving board)
```

Any part of the reverse path is prohibited, thus the distance of them is set to the largest positive numbers.

In cooperation with the original algorithm of Fencl's, our algorithm can handle an open connection — start-to-end and start-to-any node. The maintenance of this property is ensured in the algorithm by assigning the end-to-start or each-to-start distance with the largest negative values (e.g, $-n * max_{ij}[d_{ij}]$) which firmly attach to the end or any of n nodes to the start node permitting a circuit to form. In fact, the algorithm works on a net as if it were a closed circuit and keeps the node configuration by modifying the distance matrix. But this modification is transparent to the application of the algorithm. That is to say, users are not aware of the largest negative values.

As the core part of the routing algorithm, two subroutines are listed below (`route_each()` and `routing()`). Some structures are defined in the beginning part. Two subroutines are self-explaintory. `Route_each` is to enforce constraints and conditions of routing and then call `routing` to route one net (i.e., a linked list of a signal). `Routing` is to find a shortest path with r trials, given an initial node and an end node.

```
#define  LARGEST    1000000  /* for impossible route distance: see dummy node*/

extern unsigned sig_cnt;               /* length of the sigtable*/
extern unsigned multitech;             /* contains multiple technology eg ECL*/
int **Distance;                        /* distance matrix d */
short optimize_option, optimize_number;/* optimization? and optimizing level*/

extern struct sigtable sigtable[];     /* signal list table */
struct link * (sigptrs[MAXSIGSIZE]),*sigf; /* temporal list */

/* ********************* route_each() ******************
 *
 *    the routing algorithm routine
```

```
 *    call this routine will route one run path
 *
 */
void
route_each(index,l)       /* route one entry in sigtable */
int index;                /* index of the entry */
int *l;                 /* length and runs */
{
  /* main task is to create the distance matrix
   * be aware that all matrices sending to the
   * routing algorithm starts from index 1
   */
   int     i,n,j,r,p[MAXSIGSIZE];
   int     terminate_node = 0;

   /* create the list */
   sigf = sigtable[index].linkptr;  /* points to the first */
   if(sigf == NULL) {
     *l = 0;
     return;
   }

   for (i = 1; sigf != NULL; i++) {
     sigptrs[i] = sigf;
     /* what is the termination node */
     if(sigf->attrib == DASH_TERMINATE)
       terminate_node = i;
     sigf = sigf->linknext;
   }
   n = i-1;      /* number of entries */
   sigtable[index].connections = n;
   if (n == 1) {  /* only one pin in signal */
     *l = 0;
     return;
   }

   if(sigtable[index].linetype == ECL && terminate_node == 0)
     printf("Warning: no terminate node in ECL sigline: %s\n", sigtable[index].signame);

   /* now let's create the distance matrix */
   for (i = 1; i <= n; i++) {
     for (j = 1; j <= n; j++)   /* change form j < n */
       if( i != j) {
          /* ordering enforcement if ECL or when multitech applies */
          if(multitech)  {
            if(!(sigptrs[i]->attrib==BIDIRECTION || sigptrs[j]->attrib==
                  BIDIRECTION) && (sigptrs[i]->attrib > sigptrs[j]->attrib))
              Distance[i][j] = LARGEST;
            else
              Distance[i][j] = calcu( sigptrs[i]->x,sigptrs[i]->y,sigptrs[j]->x,sigptrs[j]->y);
          }
```

```
      else
        Distance[i][j] = calcu( sigptrs[i]->x,sigptrs[i]->y,sigptrs[j]->x,sigptrs[j]->y);
    }
  Distance[i][i] = 0;
}

for(i = 1; i <= n; i++) {
  Distance[n+1][i] = LARGEST;
  /* attrib with IO_OUTPUT or OUTPUT_PIN is output pins ... */
  if(!multitech)
    Distance[n+1][i] = 1;
  else
    /* if((sigtable[index].linetype != ECL) || (sigptrs[i]->attrib ==
     * OUTPUT_PIN) || (sigptrs[i]->attrib == IO_OUTPUT))
     */
    if((sigptrs[i]->attrib == OUTPUT_PIN) || (sigptrs[i]->attrib == IO_OUTPUT))
      Distance[n+1][i] = 1;

  Distance[i][n+1] = LARGEST;
}
Distance[n+1][n+1] = 0;

for(i = 1; i <= n+1; i++)
  p[i] = i;

if (n < 99/(2*n) + 1 )
  r = n;
else  {
  if (optimize_option)
    r = (int) n/optimize_number;
  else
    r = 99/(2*n) + 1;
}

/*    if(sigtable[index].linetype != ECL)
 */
if(!multitech)
  terminate_node = 0;

/* now, call the routing algorithm */
routing(n+1,p,n+1,terminate_node,MAXSIGSIZE,Distance,1,r);

for(i = 1; i <= n; i++)
  p[i] = p[i+1];

/* now we have to finish the wiring link */
/* optimize(p,n);  */

sigtable[index].wirefirst = sigptrs[p[1]];
for (i = 1; i < n; i++)
  sigptrs[p[i]]->wirenext = sigptrs[p[i+1]];
```

```
   sigptrs[p[n]]->wirenext = NULL; /* the last one */
 }



/******************  routing()  ********************************
 *
 *     shortest path routing procedure
 *     modified version from Zdenek Fencl, CACM, Vol. 16, Page. 572 (1973).
 *
 */
int
routing(n,p,sn,en,m1,d,ll,r)
int     n,      /* number of nodes to be connected */
         p[],    /* node number vector, on input; optimal connection out */
         sn,      /* start node number */
         en,      /* end node number */
         m1,     /* size of the distance matrix */
        **d,    /* distance matrix */
         *ll,     /* shortest connection length (output) */
         r;      /* number of runs */

{
  int    id[MAXSIGSIZE];
  int    q[100],
         i,i1, j,j1, js,je,jp, ip,ip1,
         mininc,  inc,
         n1, irs, k,k1,
         icount, icor,
         ir,ir1,  l1, l, lr,
         jp1, ln, large;

  /* get large number (= n * max d(i,j))  */
  large = 0;
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++)
      if (d[i][j] > large)
         large = d[i][j];
  }
  if (large == 0 ) {
    printf("ERROR: in wiring algorithm -- large = 0\n");
    exit(0);
  }

  large = large*n*2;
  /* define non-existing arcs by assigning */
  /* their distances large negative values */
  if (en == 0) {
    for (i = 1; i <= m1; i++) {
      id[i] = d[i][sn];
      d[i][sn] = -large;
```

```
      d[sn][sn] = 0;
    }
  }

  if (!(sn == en  ||  en == 0)) {
    id[1] = d[en][sn];
    d[en][sn] = -large;
  }

  /* run r trials */
  l = large;
/*280 -- the number is the label derived from the original algorithm
 *        written in Fortran.   So do the rest occurrences.
 */
  for (irs = 1; irs <= r; irs++) {
    /* build tour by  successive inserting */
    /* not-yet-involved nodes */
    /* intiate tour is considered as  */
    /* arc p(1) to p(1) */
/*90*/
    for ( js = 2; js <= n; js++) {
      mininc = large;
      /* trace all not-yet-involved nodes */
      /* to choose the one with minimum increment */
/*70*/
      for (j = js; j <= n; j++) {
         jp = p[j];
         je = js-1;
         /* for each not-yet-involved node trace already   */
         /* built-up tour to choose the minimum increment arc */
/*60*/
         for (i = 1; i <= je; i++) {
           ip = p[i];
           ip1 = p[i+1];
           if (i == je)
             ip1 = p[1];
           inc = d[ip][jp] + d[jp][ip1] - d[ip][ip1];
           if (!(inc >= mininc)) {
             j1 = j;
             i1 = i;
             mininc = inc;
           }
         }
      }
      /* stretch tour by inserting the chosen node p(j1)  */
      /* between the nodes p(j1) and p(i1+1)  */
      while ((j1=j1-1) != i1) {
         ip = p[j1];
         p[j1] = p[j1+1];
         p[j1+1] = ip;
      }
```

```
    }
    /* correct tour by 3-opt method */
    /* vary consecutive chain length k  */
    n1 = n - 1;
    if (n < 3)   goto L210;
/*200*/
    for (k = 1; k <= n1; k++) {
      icount = 0;
      /* shift consecutive chain  */
      /* throughout sequence of n nodes */
L100:
      icor = 0;
/*190*/
      for (j = 1; j <= n; j++) {
        /* calculate chain length in forward */
        /* and backward direction */
        l1 = 0;
        lr = 0;
        if (k != 1) {
          i = j;
          k1 = 1;
          do {
            if (i > n)
              i = i - n;
            ip = p[i];
            ip1 = i + 1;
            if (ip1 > n)
              ip1 = 1;
            ip1 = p[ip1];
            l1 = l1 + d[ip][ip1];
            lr = lr + d[ip1][ip];
            i = i + 1;
            k1 = k1 + 1;
          } while (k1 < k);
        }
        /* for each positioned chain (as is and inverted) */
        /* check all arcs to see if insertion improves tour */
        mininc = large;
        j1 = j + k - 1;
        if (j1 > n)
          j1 = j1 - n;
/*150*/
        for (i = 1; i <= n; i++) {
          if (j <= j1  &&  (i >= j  &&  i <= j1))   goto L150;
          if (j > j1  &&  (i <= j1  ||  i >= j))    goto L150;
          ip = p[i];
          jp = p[j];
          jp1 = p[j1];
          ip1 = i + 1;
          if (ip1 > n)
            ip1 = 1;
```

```
          je = ip1;
          if (ip1 == j)
            ip1 = j1 + 1;
          if (ip1 > n)
            ip1 = 1;
          ip1 = p[ip1];
          ln = l1;
          ir = 0;
L130:
         inc = d[ip][jp] + ln + d[jp1][ip1] - d[ip][ip1];
          if (inc > mininc  ||
               (inc == mininc  &&  (je != j  ||  je == j  &&  ir == 1)))
            goto L140;
          i1 = i;
          ir1 = ir;
          mininc = inc;
L140:
         if (ir == 1)   goto L150;
          ir = 1;
          ln = lr;
          js = jp;
          jp = jp1;
          jp1 = js;
          goto L130;
L150:  ;
        }
        i = i1 + 1;
        if (i > n)
          i = 1;
        if (i == j  &&  ir1 == 0)   goto L190;
        /* reinsert chain of length k starting in j */
        /* between nodes p[i1] and p[i1+1] */
        icor = 1;
        js = j;
        je = 0;
        if (ir1 == 0)   goto L160;
        js = j1;
        je = -1;
L160:
        k1 = 0;
L170:
        k1 = k1 + 1;
        if (k1 > k)     goto L190;
        i = js;
        js = js + je;
        if (js < 1)
          js = n;
L180:
        ip = i + 1;
        if (ip > n)
          ip = 1;
```

```
        jp = p[i];
        p[i] = p[ip];
        p[ip] = jp;
        i = i + 1;
        if (i > n)
          i = 1;
        if ((ip-i1) == 0)  goto L170;
        else goto L180;
L190:  ;
      }
      if (icor == 0)      goto L200;
      icount = icount + 1;
      if (icount < n)     goto L100;
L200: ;
    }
L210:
    /* orient tour with sn in p(1) */
/*230*/
    for (i = 1; i <= n; i++) {
      if (p[1] == sn)
        goto L240;
      js = p[1];

/*220*/
      for(j = 1; j <= n1; j++) {
        p[j] = p[j+1];
L220: ;
      }
      p[n] = js;
L230: ;
    }

    /* calculate tour length */
L240:
    l1 = 0;
/*250*/
    for (i = 1; i <= n1; i++) {
      ip = p[i];
      ip1 = p[i+1];
      l1 = l1 + d[ip][ip1];
L250: ;
    }
    ip = p[1];
    if (sn == en)
      l1 = l1 + d[ip1][ip];

    /* save the solution, if better, and set new initiate node */
    if (l1 >= l)
      goto L270;
    l = l1;
/*260*/
```

```
    for (i = 1; i <= n; i++) {
      q[i] = p[i];
L260: ;
    }
L270:
    j = irs + 1;
    if (j > n)
      j = j - n;
    js = p[1];
    p[1] = p[j];
    p[j] = js;
L280: ;
  }
  /* restore p and dummy distances  */
/*290*/
  for (i = 1; i <= n; i++) {
    p[i] = q[i];
L290: ;
  }
  if (en != 0)
    goto L310;
/*300*/
  for (i = 1; i <= m1; i++) {
    d[i][sn] = id[i];
L300: ;
    }
L310:
  if (sn == en  ||  en == 0)
    goto L320;
  d[en][sn] = id[1];
L320:
  *ll = 1;
  return;
}

int
calcu(x1,y1,x2,y2)      /* calculate distance between two points */
int x1,x2,y1,y2;
{
  int x,y;
  float d;

  x = x1 - x2;
  y = y1 - y2;
  d = (float)(100 * (x*x + y*y));
  d = sqrt(d);
  return (int) d;
}
/* --- end --- */
```

Note that the parameter r depends on n. When $n <= 5$, r is set to n. Otherwise, it will depend on the option of optimization level. The larger the level, the less optimized it is (our default optimization level is set to the smallest to achieve the best result).

3 Features

The wiring tool attempts to be of general-purpose in the sense that the applications are not limited by any particular form of boards, chips etc — provided that their descriptions are given (which can be prepared by text files). The tool can be easily adapted to another board, other chip types with other technologies which are not available now but may be available in the long run.

One goal of the tool is to hide miscellaneous details from users so that users are only required to learn a little how to use the tool. Currently, the wiring tool has following features.

- **No Limitation to Board Type.** Originally, only one type of board was considered for wiring. It was very dependent and not capable of handling new board types. For example, the PE module for NYU Ultracomputer Prototype II uses a COMBION board, while the current PE module for Ultracomputer Prototype III uses a MUPAC board and future machine implementations for large size system will use a much bigger sized board. Our tools handles this technology generations generally with "pixel-mapping" each board (the term used in computer graphics). In other words, instead of traditionally using binary form to represent board information (1 for a position for a pin — a hole usually, and 0 otherwise), the board description in textural form is used. For example, in addition to that the position is "1" in the description if it is a correspondent hole on the board and "0" with no hole, other possible situations may arise: if the position on the board is a ground (GND), or a power (+5V), or for solder wash, we must be able to specify it in the board description explicitly ("G", "P", and "S" respectively). The use of pixel-mapping technique instead of binary mapping is to provide an easy interface for one to modify or reconfigure the board description. If only "1" or "0" is limited to use in binary file (often called *bit-mapped*), other features such as power, ground and solder wash cannot be implemented. Furthermore, pixel-mapping offers more dimensions to deal with new characteristics of each position on the board. As we can see, it only requires one to supply a different board description if a different board is used. When a new board is to be used, the tool provides a very efficient way to upgrade from the old board to the new one. The only thing to do is to supply the new board descriptions.
- **Variety of Chip Types** There are many chip types which are used frequently in computer system design and implementations. Often-used chip types are DIP type, PGA type, and adapters (often in irregular forms). We may want to include connectors as well in the chip type category since it looks almost same as a chip in the point view of wiring tool although it is used to connect to the outside of the components. All of these chip types are different not only for placement (physical shapes, initial placement point), but also for their pin arrangements and pin numbering scheme. DIP pins are counted numerically (e.g. 1, 2, 3), but PGA pins are usually numbered in the form of <letter><digit> (e.g. A1, C32), the pins of adapter and connector, on the other hand, can be counted in either way. All these complicate the determination of physical positions for any given pin with different chip types.

  Our wiring tool provides an interface for users to specify the chip type in a commonly accessible file called "GENERIC.cdf" (cdf means *chip description file*), which contains descriptions of different chip-type groups and numerous chip types in each group. For example, in group DIP (identified by [DIP], see Table 1), there are a number of chip types with their specifications (where ';' begins a comment line).

  All chip types in Table 1 belong to DIP. For example, 68000 is a type of Intel 68000 CPU which has width of 9 and 64 pins in total, manufactured with TTL compatible technology; 7400 is a type of

```
[DIP]
      ;  chip_type      technology     width     pins
         2764               TTL          6        28
         68000              TTL          9        64
         7400               TTL          3        14
         AM29841A           TTL          3        24

         ......
```

Table 1: DIP Chip Description

NAND gate chip which has width of 3 and 14 pins totally. We may call chips with the same type differently in each instance when using them, but only one entry is put in the description file. For instance, we may name "NAND1" for the first chip used, and "NAND2" for the second and so on.

Other group of chip type such as PGA (identified by [PGA]) is specified with little difference as in Table 2).

The first PGA type in this example is an AMD 29000 RISC processor type, which uses CMOS technology with TTL-compatible; and the second PGA is a XILINX type. Similarly, chips with the same type can also have their different chip names.

- **Any Form of Adapters**

  Adapters are a really big problem in implementations. Our first attempt was to define the adapter by specifying the number of pins column by column. But unfortunately, adapters are often in various forms and sometime very irregular. For instance, it may look like PGA form but not completely if you look at it carefully (see table refadaptor. The best way of generalization is to define the adapter type in so called "fully-mapped" way, which means every pin must be specified in the descriptions. Take an adapter type — PGA132-03, for example, it can be specified as in Table 3 This type of adapter looks almost regular, but in the first row the pin arrangement is different from the rest. So, care must be taken to number them.

- **Fair Handling to Power and Ground**

  The distribution of power and ground pins is board-dependent. But they have something in common, i.e., they are intended to be distributed evenly on the board so that power and ground pins of the chips can be assured to be available in an area close to them.

  The wiring tool takes advantage of this property and route power and ground signals with the nearest corresponding board power and ground pins. We will describe the algorithms of searching the nearest in section 4.

- **Capability for Solder Wash**

  When PGA type chips are used, it is often required in the PGA area to solder the power or ground pins rather than route them. The wiring tool is capable of knowing which area can be solder-washable from the board description. From this information, It can produce a list of solder-wash pins in detail along with other wiring information.

- **Flexibility for Connectors**

  Flexible handling to connectors are extremely important for generalization of the tool. Originally, connectors were considered only one line on top of the board and one line on the bottom of the board, with each line being several rows. But in practice, it is not the case. Very often, connectors can be put in arbitrary place around or within the board, and sometimes, multiple lines of connectors are

```
[PGA]
  ; PGA chip_type  technology   number-of-rows   number-of-columns
       AM29000        CMOS           17                 17
          U T R P N M L K J H G F E D C B A
       1  * * * * * * * * * * * * * * * * *
       2  * * * * * * * * * * * * * * * * *
       3  * * * * * * * * * * * * * * * * *
       4  * * * * - - - - - - - - - - * * *
       5  * * * - - - - - - - - - - - * * *
       6  * * * - - - - - - - - - - - * * *
       7  * * * - - - - - - - - - - - * * *
       8  * * * - - - - - - - - - - - * * *
       9  * * * - - - - - - - - - - - * * *
       10 * * * - - - - - - - - - - - * * *
       11 * * * - - - - - - - - - - - * * *
       12 * * * - - - - - - - - - - - * * *
       13 * * * - - - - - - - - - - - * * *
       14 * * * - - - - - - - - - - - * * *
       15 * * * * * * * * * * * * * * * * *
       16 * * * * * * * * * * * * * * * * *
       17 * * * * * * * * * * * * * * * * *

       XILINX             TTL             16                16
          T R P N M L K J H G F E D C B A
       1  - * * * * * * * * * * * * * * *
       2  * * * * * * * * * * * * * * * *
       3  * * * * * * * * * * * * * * * *
       4  * * * * - - - - - - - - * * * *
       5  * * * * - - - - - - - - * * * *
       6  * * * * - - - - - - - - * * * *
       7  * * * * - - - - - - - - * * * *
       8  * * * * - - - - - - - - * * * *

       9  * * * * - - - - - - - - * * * *
       10 * * * * - - - - - - - - * * * *
       11 * * * * - - - - - - - - * * * *
       12 * * * * - - - - - - - - * * * *
       13 * * * * - - - - - - - - * * * *
       14 * * * * * * * * * * * * * * * *
       15 * * * * * * * * * * * * * * * *
       16 * * * * * * * * * * * * * * * *

            ......
```

Table 2: PGA Chip Descriptions

```
;  type-name          width   length
   PGA132-03           17       14
  01 18 - - 35 52 - -  - -  - -  -   -
  02 19 - - 36 53 - - 69 85 - - 101 117
  03 20 - - 37 54 - - 70 86 - - 102 118
  04 21 - - 38 55 - - 71 87 - - 103 119
  05 22 - - 39 56 - - 72 88 - - 104 120
  06 23 - - 40 57 - - 73 89 - - 105 121
  07 24 - - 41 58 - - 74 90 - - 106 122
  08 25 - - 42 59 - - 75 91 - - 107 123
  09 26 - - 43 60 - - 76 92 - - 108 124
  10 27 - - 44 61 - - 77 93 - - 109 125
  11 28 - - 45 62 - - 78 94 - - 110 126
  12 29 - - 46 63 - - 79 95 - - 111 127
  13 30 - - 47 64 - - 80 96 - - 112 128
  14 31 - - 48 65 - - 81 97 - - 113 129
  15 32 - - 49 66 - - 82 98 - - 114 130
  16 33 - - 50 67 - - 83 99 - - 115 131
  17 34 - - 51 68 - - 84 100 - - 116 132
```

Table 3: Adaptor Chip Description

required. Moreover, connectors can be available with any shape and any forms Pin arrangement and numbering of connectors also varies (PGA-like, DIP-like etc). To make it general, what we do is to treat connectors with a special chip type (since it also has its placement information and pin connection information, as many other chips do). The tool provides enough flexibility so that connectors can be placed anywhere around the board and in any orientation. Besides, users do not have to worry about the problem for any particular shape of connectors, no matter whether it is regular or irregular.

- **Portability and Extendability**

  The tool itself is written with C programming language. It is easy to port to other machines. Currently, the tool can be run on any machine in Ultracomputer Laboratory. The porting to IBM/PC MS-DOS systems exhibits very successful (due to Jiarui Wang's effort with DOS gcc, gnus' C compiler version on MS-DOS). With little difficulty, additional features can be added it into the current system when necessary. More issues will be addressed in section 6.

- **Multiple User Command Options**. The user command to run the wiring tool has a few useful options. The command usage for wiring DASH version 4 is:

  ```
  wirepv4 -d <descript-file-dir> -c -m -D -L pinfile
  ```

  where –d <descript-file-dir> is to specify the directory of description files,
  –c is for cyclic-wiring (to reduce transmission latency),
  –m for multiple technologies (such as ECL technology),
  –D for debugging purpose, and
  –L for generating landscape <file>.lsw output (default: portrait).

Users are required only a couple of basic files before running the tool:

(1) A pin list file, which is the output file by DASH post processing [4].

(2) A placement file, which contains placement information of each chip. The file can be created manually, or generated by a simple program, from a DASH schematic which contains solely all chips used and the board on which the chips are placed. The board being used must be explicitly specified in the placement file.

(3) The board description file, which describes the board row by row, and column by column. Characteristics of each location of the board is specified by recognizable characters (the meaning od each character is explained as follows):

```
1  -- there is a pin in the location, which can be wirable,
0  -- no pin in this place, thus cannot be wired,
E  -- the location is part of edge connectors,
G  -- the location is a ground pin,
P  -- the location is a power pin, and
S  -- the location can be soldered wash.
```

(4) The chip description file (it is named as "GENERIC.cdf"), which is accessed by all applications. As we described above, each entry must belong to a group of chip types, and each entry itself is a member of that group. The information required for each chip type is a tuple of <chip type name, technology of the chip, geometrical data (like width, length, number of pins or just layout of the pins)>.

The wiring tool reads DASH [4] pin list files as input (the syntax of pin list file will be described later). Pin list file is extracted from DASH schematic layout. The pictorial information with their attributes is translated into the textual pin lists by DASH's post-processing. The pin list file provides all necessary information about the net of each signals. Since the DASH software package itself is upgraded, the format and syntax of pin list file is somehow changed, our tool must be compatible with their changes. The only way to do so is provide an optional pin list parser so that users can select as they wish.

After the information about the board, chips and the placement is properly read and processed, the wiring tool is to convert logic position of each pin to its internal representation (physical pin location). Each pin is associated with its chip and each chip is associated with a chip type. The wiring tool then generates the signal table, which contains initial pin connection orders of each signal. Each pin's physical location is obtained at this stage. The router performs all rest jobs to optimize the routing with the minimum distance and other required constraints (like technology constraints and direction limitations). The option "-m" is used to indicate multiple technologies are used (not only TTL, but also others like ECL, CMOS etc). Their constraints are appropriately taken care of by the wiring tool (see man page appendixed at the end of this report).

The results generated by the wiring tool are outputs in several files (where <filename> is the user's application file name without file attribute name):

```
<filename>.lsw -- statistic report of chips and their types, and
                  the number of chips each type; information about
                  connections of each run (line) of signals (containing
                  how many points of the connections, wire length, and
                  the wire path each connection);
<filename>.csb -- a rough diagram report about the board on which each
                  chip is placed. A diagram of the signal arrangement
                  on the board is also given.
<filename>.wls -- final output to manufactures, containing only pin
                  positions (logical) or pin names of wire connections.
```

```
                  The chip name or chip types does not care;
```

There are other features and functions that the wiring tool can provide, such as those options along with user commands. For example, option "-D" can be used to debug in detail and locate easily where an error takes place. With this option, three extra files are produced,

```
EACHPIN.dbg -- output of each pin with its internal (physical) location.
CHIPS.dbg   -- output of each chip with its initial pin's internal
               (physical) location.
SIGNALS.dbg -- output of all signals and their initial (unrouted thus not
               optimal) connection orders.
```

Another example is that it can produce cyclic wirings (or loop wiring) to reduce transmission latencies if necessary. But it is limited for those nets with a small number of connected nodes.

4 Implementation

The tool consists mainly of three components: Preprocessor, Wiring Programs (*Wiring-man*) and Output Facilities (*Output Generator*). The relationship between them can be described in Figure 1.

The Preprocessor analyzes the given pin list files and retrieves information about chip, chip types, pins, and signal connections. Other information is supplied from board description files, chip description files and placement files so that the data are collected and organized. The Wiring-man takes the organized data from the preprocessor and applies the heuristic routing algorithm to achieve the objective of minimum length. The Output Generator postprocesses the results of wiring and generates several files for design verifying, final checking and data supplying to vendors.

4.1 Pinlist Parsing

DASH pinlist was a flat form before version 4. Since version 4, the list structure has been changed to hierarchical (nested) form. Following are the pin list group nesting provided by DASH Manual [4].

```
PINLIST,4                 Header Record
  DATE,...                Run Date
DCM,...                   DCM File Definition Group
  DATE,...                DCM File Creation Date
  SET,...                 Drawing Set Definition Group
    FILE,...              Drawing Number and Filename
      DATE,...            Drawing File Creation Date
D,...                     Design Data Record
  T,...                   Design-Related Text
DWG,...                   Drawing Group
  PATH,...                Path Record
  D,...                   Drawing Data Record
    T,...                 Drawing-Related Text Record
```

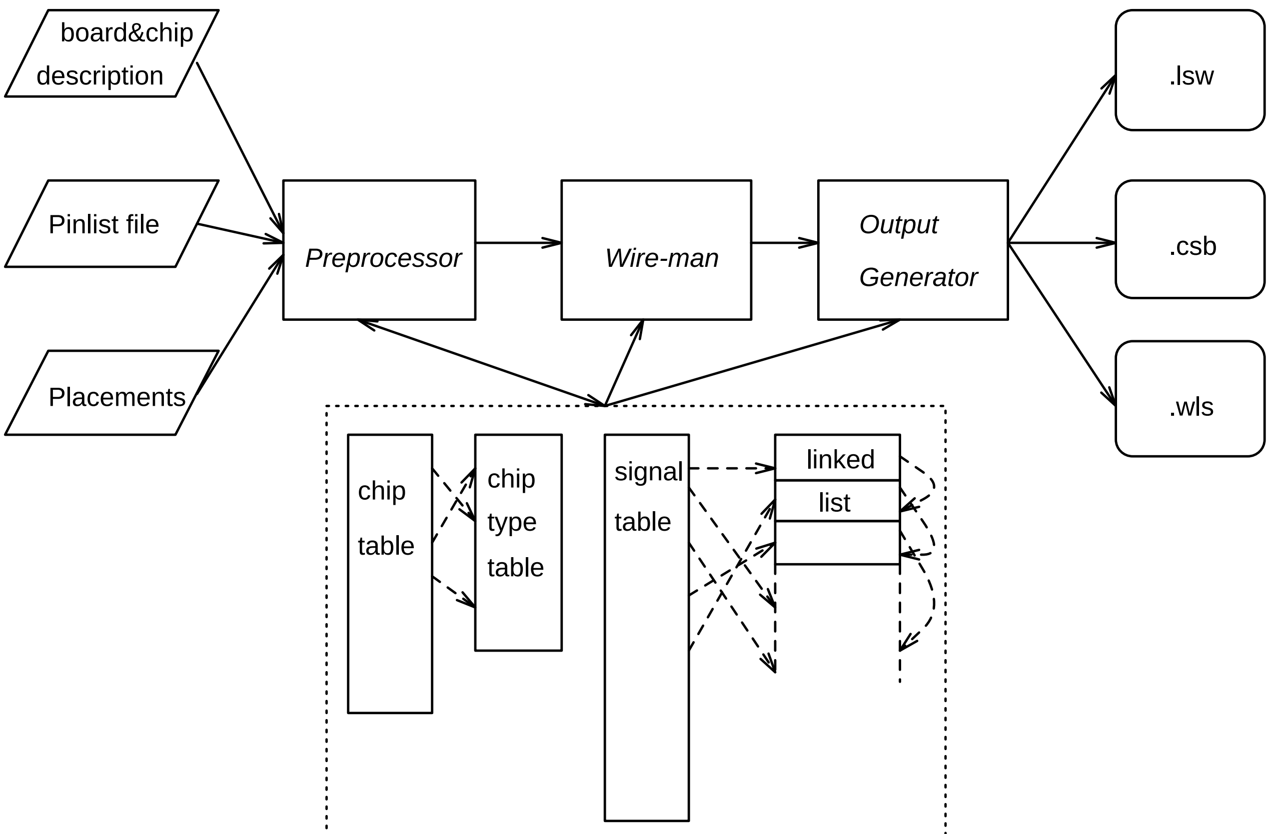


Figure 1: Wiring Components and Their Relationship

```
SYM,...                 Symbol Record
  D,...                 Symbol Data Record
    T,...               Symbol-Related Text Record
  PIN,...               Pin Record
    D,...               Pin Data Record
      T,...             Pin-Related Text Record
BUS,...                 Bus Record
  D,...                 Bus Data Record
    T,...               Bus-Related Text Record
  BSIG,...              Bused Signal Record
    EQU,...             Equate Record
    D,...               Bused Signal Data Record
      T,...             Bused Signal-Related Text Record
SIG,...                 Signal Record
  EQU,...               Equate Record
  D,...                 Signal Data Record
     T,...              Signal-Related Text Record
DWG,...                 Drawing Group
```

To parse the pinlist, we define our parser rules as follows[3]:

```
  Start ==:  pinlist_file
  pinlist_file ==: header_grp drawing_grp

/* HEADER GROUP:  contains a header record, a run-time date record,
 * DCM file definition group, set definition GROUPS, and design
 * data GROUPS.
 */
  header_grp   ==: header_rec  dcm_file_def_grp  design_data_grps
  header_rec   ==: head_rec  date_rec
  head_rec     ==: PINLIST COMMA NUM NL

/* A header record is followed by a date record which specifies the date
 * on which the pin list report was generated
 */
  date_rec     ==: DATE COMMA  text NL

/* Followed by is a DCM file definition group, which contains two records:
 * a DCM file record and
 * a file creation date record
 */
  dcm_file_def_grp ==: <epsilon>     /* where <epsilon> is an empty */
                | dcm_file_def_grp  dcm_file_rec file_create_date_rec set_defs
  dcm_file_rec ==: DCM COMMA FILEPATH COMMA NUM NL
  file_create_date_rec ==: date_rec

/* Set defs is a series of set definition groups that document the drawing
```

[3]The rules are implemented in an yacc file “pinlist.y”; a lex file “pinlist.lex” (not shown in this report) is created to provide its recognized tokens to the parser.

```
  sets that were used in creation of the dcm file
 */
 set_defs       ==:  <epsilon>   /* nothing */
               | set_defs  set_def_grp

/* Set definition group consists of a set record followed by a series of
  file and file date record pairs
 */
 set_def_grp    ==: set_def_rec  file_date_pairs
 set_def_rec    ==: SET COMMA NUM NL
 file_date_pairs ==:  <epsilon>      /* nothing */
                | file_date_pairs  file_date_rec   date_rec
 file_date_rec : FFILE COMMA NUM COMMA FILEPATH NL

/* Followed by the set definition group is a # of design data groups,
  which document the contents of the design-related alpha fileds
  contained within the design
 */
 design_data_grps ==: /* nothing */
                 | design_data_grps   design_data_grp

/* Each design data group consists of A Data Record followed by Text
  RecordS
 */
 design_data_grp ==: data_rec text_recs
 data_rec   ==: D COMMA NUM COMMA text NL
 text_recs  ==:  /* nothing */
           | text_recs data_texts
 data_texts ==: T COMMA text NL
           | T COMMA id COMMA text NL

/* ========================= DRAWING GROUPS ======================= */
/* Drwaing groups contains all of the data that relates to an instance
 * of a particular drawing file in the design. In structured designs,
 * drawing groups may be generated several times for the same drawing if
 * that drawing file is used multiple times in the structured design.
 *
 * Six types of records can appear in drawing groups.
 *   1st -- DWG  header record (always present).
 *   2nd -- PATH
 *   3rd -- D    (drawing data groups)
 *   4th -- SYM
 *   5th -- BUS
 *   6th -- SIG
 *
 */

 drawing_grp  ==:  /* nothing */
           | drawing_grp  drawing_rec  path_rec
                 draw_data_grps  symbol_grps  bus_grps  signal_grps;
 drawing_rec ==: DWG COMMA FILEPATH COMMA OCC_CNT COMMA num NL
```

```
 path_rec    ==:   /* nothing */
           | path_rec PATH COMMA FILEPATH COMMA OCC_CNT COMMA num COMMA id NL
 text       ==: /* nothing */
 occ_cnt    ==: /* nothing */
           | OCC_CNT
 num        ==: /* nothing */
           | NUM

/* Follwed by the path group is the drawing data group, which
    contains all the drawing-related alpha fields and
    associated layered text for the drawing file.
 */
 draw_data_grps ==: /* nothing */
               | draw_data_grps  draw_data_grp
 draw_data_grp  ==: draw_data_rec  data_text_rec
 draw_data_rec  ==:  D COMMA num COMMA text NL
 data_text_rec ==: /* nothing */
              | T COMMA text NL
              | T COMMA id COMMA text NL
 symbol_grps  ==: /* nothing */
              | symbol_grps symbol_grp
 symbol_grp   ==: symbol_rec
               sym_data_grp
               sym_pin_grp
 symbol_rec   ==: SYM COMMA num COMMA text NL
 sym_data_grp ==: /* nothing */
             | sym_data_grp  sym_data_rec  data_text_rec
 sym_data_rec ==:  D COMMA NUM COMMA id NL
 sym_pin_grp ==: /* nothing */
            | sym_pin_grp  sym_pin_rec  pin_data_rec
 sym_pin_rec==: PIN COMMA bus_id COMMA bus_attr COMMA bus_occ_cnt
                 COMMA sig_id COMMA sig_attr COMMA sig_occ_cnt COMMA pin_attr
                 COMMA pin_id NL
 pin_data_rec ==:  D COMMA num COMMA text NL
 bus_id      ==:  id ;
 bus_attr    ==: num ;
 bus_occ_cnt ==: occ_cnt ;
 sig_id      ==: id ;
 sig_attr    ==: num ;
 sig_occ_cnt ==: occ_cnt;
 pin_id      ==: id ;
 pin_attr    ==: num ;
 id    ==: /* nothing */
      | ID
      | NUM
      | OCC_CNT

 /* BUS group consists of a bus record, bus data groups, and bused
  * signal groups
  */
 bus_grps    ==: /* nothing */
```

```
          | bus_grps  bus_rec bus_data_grp bus_sig_grp
bus_rec   ==: BUS COMMA  bus_id  COMMA  num  COMMA  occ_cnt NL
bus_data_grp ==:  /* nothing */
          |  bus_data_grp bus_data_rec
bus_data_rec  ==: D COMMA num COMMA id NL
bus_sig_grp ==: /* nothing */
          | bus_sig_grp   bus_sig_rec  bus_sig_data_grp
bus_sig_rec ==:  BSIG COMMA sig_id COMMA num COMMA occ_cnt NL
equate_rec ==: EQU COMMA bus_id  COMMA alias COMMA num NL
alias           ==: id
bus_sig_data_grp ==: /* nothing */
              | bus_sig_data_grp   bus_sig_data_rec
bus_sig_data_rec ==: D COMMA num COMMA id NL

/* Non-bused signal groups. A signal group cinsists of a signal
 * Record, equate records, and signal data groups
 */
signal_grps     ==: /* nothing */
           | signal_grps signal_rec  sig_data_grp
signal_rec      ==: SIG COMMA sig_id COMMA num COMMA occ_cnt  NL
sig_equate_rec  ==:  EQU COMMA COMMA alias COMMA num NL
sig_data_grp    ==: /* nothing */
                  | sig_data_grp  sig_data_rec  data_text_rec
sig_data_rec    ==:  D COMMA num COMMA id  NL
```

4.2 Data Structures

The data structure for implementations is mainly tables and linked lists. Three tables are important through the program running: chip table (chiptab), chip type table (chiptype), and signal table (sigtable). Their data structures are defined below:

```
/*
 * Chip Table
 */
struct chiptable {
  char           *chipname;     /* chip name */
  struct chiptype *typeptr;     /* point to chip type table */
  int            initrow;       /* place of the first pin  -- row */
  int            initcol;       /*                         -- col */
  char           **sig_array;   /* if CNTTYPE, store the signals */
  struct     gv   *glink;       /* for power and ground link */
  struct     gv   *vlink;
  char           orient;        /* V = vertical, H = horizontal */
  union {
    struct XY1 {
         int *loc_x;          /* pin's x relative position to the first pin*/
         int *loc_y;          /* pin's y relative position to the first pin*/
    } XY1;
```

```
    struct XY2 {
      int **loc_x;           /* pin's relative position to the first pin */
      int **loc_y;
    } XY2;
  } cu;
 } ;

/*
 * Chip Type Table
 */
struct chiptype {
  char *typename;               /* chip typename */
  short pactype;                /* package type(i.e, DIP,PGA,etc) */
  short tech;
  union {                       /* union for each package type */
    struct DIP {  /* DIP type */
        unsigned width;
        unsigned pins;
      } DIP;
    struct PGA {          /* PGA type */
      int **pa;                 /* int pa[PGAROW][PGACOL]; */
      char alp[26+1];           /* alphabet, 26 letters a,b,c... */
      int *num;                 /* store number of row/col */
      short tag;                /* for pinname determine whether A1 or 1A */
      unsigned rows,cols;
     } PGA;
    struct ADP {       /* Adaptor type */
      int *width_array;           /* for storing info of widths */
      char alp[26+1];             /* pga mapping */
      int index[26][30];
      short pga;
      unsigned rows,cols;        /* number of rows and columns */
     } ADP ;
    struct CNT {     /* Connector type */
      unsigned width,pins;
      short pga;
      char *col_char;            /* wid-col chars A,B,C... */
      int **layout;              /* the layout of its chiptype, from chiptype description */
      int *offset_x;
      int *offset_y;
    } CNT;
    struct FMP {            /* Full-Map type */
      unsigned width,pins;
      char alp[26+1];
      int index[26][30];
      char *col_char;
      int **layout;
      int *offset_x;
      int *offset_y;
    } FMP;
  } u;
```

```
};

/*
 * Signal Table
 */
struct sigtable {
  char        *signame;        /* signal name */
  int         connections;     /* how many connections */
  int         linetype;        /* signal line type: TTL, ECL, other new one? */
  float       length;          /* total length */
  struct link *lastlink;       /* pointing to the last one on list */
  struct link *wirefirst;      /* wired order for that signal */
  struct link *linkptr;        /* point to link chain(one by one in an entry)*/
};

/* link chain for sigtable
 * store signal information.
 */
struct link {
  struct chiptable *chipptr;       /* point to which chip it belongs */
  char             *pinname;       /* pin name */
  int              x;              /* its physical position in board */
  int              y;              /*   (x,y)  */
  int              attrib;         /* input,output,bidirectional,tristate?*/
  struct link      *wirenext;      /* point to the pin in the wired path */
  struct link      *linknext;      /* next pin connected with same sig*/
};
```

Each chip has an entry in chip table, in which a pointer (`typeptr`) is directed to its associated chip type (in chip type table). Each chip table contains the chip name, the initial placement locations, orientation flag, and each pin's relative positioning to its first pin. Each chip type table includes information of type name, package type (e.g., DIP, PGA), technology used, and geometrics of the chip type. Signal table is constructed from the pinlist data and provides all information for wiring. Each signal entry consists of signal name, number of connections, associated chip technology (`linetype`), linked list pointers to pins of the signal net and total length of the net, An important feature is that each pin in the linked list points back to its chip table entry so that its associated chip can be easily found.

4.3 Input Interface

As illustrated in Figure 1, some input files are required: board descriptions, chip descriptions and placement data. Two procedures (functions in C) are responsible for reading and processing inputs: `read_cdf()`, and `read_plc()`. `Read_cdf()` reads chip descriptions one by one from file "GENERIC.cdf". As we described before, each entry of chip description contains chip type name, technology used (TTL, ECL etc), and the geometrical information (width, length, or even just layout). Each entry belongs to one group of chip types. These groups are DIP, PGA, Adaptors, Connectors, and the Full-Mapped. `Read_cdf()` is to store the information of each chip type in chip type table. At this time, two chip types are specially treated: Connectors and the Full-Mapped, because the relative positions of their pins are known from the chip descriptions.

`Read_plc()` obtains the board description data and chip placement data from the corresponding files. A `Read_board()` function is called to process board description. It first checks the top and bottom area beyond of the actual board (in case of connectors connected). Then, logical and physical location mappings of each pin on the board are created, which considers both the edge connector area and the top connector area. After that, ground and power pins are specially handled so that a linked list is formed for them. Solder wash information is also recorded. The physical location for a given pin is obtained by the function of `get_pin_position(chiptab,pname,&x,&y)`, where `chiptab` is the pointer to the chip table, `pname` is pin name, and `x` and `y` are the returned physical location of the given pin. Logical and physical address translation is performed by `physic_to_logic(), logic_to_physic(), x_physic_to_logic(), y_physic_to_logic(), x_logic_to_physic(), and y_logic_to_physic()` .

`Read_plc()` reads the placement data (usually the first pin position of each chip) from `.plc` file, and place each pin in relative to the first pin by the geometrical information obtained from chip descriptions. Only then can physical location of a given pin be calculated and translated. The linked list of each signal thus can then be filled with sufficient data for wiring algorithm.

4.4 Power and Ground Localizations

The Power and Ground are two very special signals for any designs. To reduce the complexity and length of wires, most boards provide many power and ground positions across it. Sometimes, these power and ground positionings may not be evenly distributed. One of our objectives is to reduce each wire length as much as possible. The search for the closest power or ground pin on the board is very crucial for each power or ground signal net. We adopt the approach of localizations. That is, each power or ground pin on the board builds its own territory as "aggressively" as possible. But, at the same time, their competition should also be as fair as possible. As a result, each pin "radiates" in eight directions (North, South, East, West, NE, NW, SE and SW). The algorithm for localization can be well parallelized in the future since each process of "radiation" is independent except that it meets the boundary or other pin's territory. The algorithm is implemented in the function of `init_gv()` in a file called `power_ground.c`

4.5 Wiring Program

Wiring program (in `route.c`) is straightforwardly written as long as the signal net is formed and the routing algorithm is designed (see section 2). The only thing that complicates the wiring function is power and ground signals, in particular, the SGND (signal ground) and SHTTL (signal power). The function of `do_wiring()` is written to deal with SGND and SHTTL. Before a ground or power pin is found, we must be sure whether or not it is in the solder wash area. If so, we may not have to wire it since the pin is solder-washed. Then, we call power and ground localization function to evenly distribute them for signal nets. After that, for each power (ground) signal net, a power (ground) pin on the board are connected with a shortest path (by looking up localization information) At this point, all power and ground pins are connected (it is possible to have only one power or ground pin in the net – it is the board pin itself). We skip this power or ground net with no connections, and join the rest of power and ground nets together with other signals in the signal table.

As everything is done, the routing algorithm is called to route each signal net with the minimum length under some conditions ans constraints. The wire length is returned upon the completion of routing.

4.6 Output Facilities

From the illustration of Figure 1, three outputs are generated: the tt .lsw file, the `.csb` file and the `.wls` file. The `.lsw` file gives a report of chips, their types and the number of their occurrences. It also contains detail connection information for each run (line) of signals (such as how many connections, wire length and paths of each connection); The `.csb` file displays diagrams about chips placed on the board and the area occupied. Diagrams for signals are also produced. The `.wls` file contains the final output for vendors (point-to-point connection each line).

Chip, chip types and the number of their occurrences are immediately available upon the pinlist file being parsed and analyzes. The collection of this data is made by function `statistics()` . The `.csb` file is obtained by functions of `chip_plc()` and `signal_on_board()` in file `print_on_board.c`. The `.wls` result is produced by `siglines()` in `sigline.c`. The output printing programs are straightforward. We don't want to repeat here.

5 An Example

To illustrate how the wiring tool is working, we give an example of IOP board, an IO Processor for Ultracomputer.

Building the IOP interface card is a typical computer hardware design. The schematic of one design is shown in appendix A. In order to run wiring program, first, we must have the IOP's DASH schematic postprocessing file — `iop.pv4`, which is the pin lists generated from the drawing files of schematics. The format and syntax of `iop.pv4` is the same as described in section 4. Second, we need IOP's placement file — `iop.plc`, which is included in appendix B. Other files (like the GENERIC chip description file, and the board description file) must also be supplied before running the applications. Appendix C and D show both two description files respectively.

When all data files are ready, we may start running the program. A typical running proceeds as follows (by UNIX script):

```
Script started on Sun Jan 26 03:07:38 1992
> wirepv4 iop
Wiring Soft, Version 1.3
(c)Copyright, Ultracomputer Lab, NYU, 1989-1991. All rights reserved.(G. Chen)

** Reading and parsing pinlist file: iop.pv4 ...
** Parse successful!
** Output the report of chips and chip_types to file: iop.lsw ...
** Loading placement info from file iop.plc...
** Loading board information from file PCAT.bod...

** Wiring... (205 signals, 44 of them are Power/Ground signals)
Wiring:     1 IOW-
Wiring:     2 IOR-
Wiring:     3 SA.3
Wiring:     4 SA.4
......
```

```
** Output the map of chips-on-board to file iop.csb...
** Output signal wiring info also to file iop.csb...
** Output result to file iop.wls...
Warning: length 0 in wiring: chip=CN1AB, pin=B10
Warning: length 0 in wiring: chip=CN1AB, pin=B31
Warning: length 0 in wiring: chip=CN1CD, pin=D18
Successfully Done!
> exit
exit
script done on Sun Jan 26 03:08:24 1992
```

Each message above tells which stage the program has reached. Notice that

- *iop.lsw*[4] contains two parts: the first part is statistic information about chips, chip types and placements; the second part lists all runs of signals (after having wired, of course) with final summary about number of runs of each wired length, total number of runs and total length of wires.
- *iop.wls* lists all real wires line by line, with each line being an individual wire.
- We do not show iop.csb in the appendix because it is very long. *Csb* is named by us, which means *chip signals on the board.* In literally meaning, iop.csb contains rough diagrams of chip placements and signal distributions on the board.

6 Discussion

Our wiring tool has been used for about 2 and half years for wiring various boards of Ultracomputers. Experiments show that our routing algorithm can achieve optimal results in the probability over 95%.

We note that as the speed of computation increases, in particular on newly fast RISC machines, the parameters of our algorithms may have to change dramatically. The algorithm given in section 2 is just an old sample running on relatively low-speed IBM-PC/RT machines. We may want to change the parameter r to be larger value if it is run on a newly fast RISC machines (e.g., HP RISC 7000 series or the MIPS-based machines).

Actually, on a fast RISC machine, we can take an exhaustive approach for a net with r nodes. The value of the r can be as large as 20 if the computing power is available and allowed. Such an approach will be taken in later version of wiring tool, hoping to obtain better results.

Extensions to the tool can be made in several ways:

- by adjusting parameters defined in header file `newpac.h` if capacity of wiring program is beyond control. Most important parameters are `MAXCHIPS` — maximum of chips it can handle (300 in default), `MAXTYPE` — maximum chip types allowed in wiring (200 in default), `MAXSIG` — maximum signals allowed in sigtable (3000 in default), and `MAXSIGSIZE` — maximum size for an entry of sigtable (500 in default).

[4]Historically, the file attribute name of *.lsw* was inherited from previous generation developers. We are not sure what it stands for. But we guess it means *list wired*. Similarly, *.wls* maybe means *wired list*, which is supplied to vendors for wiring.

Other useful parameters are board sizes (`BODROW, BODCOL`), limited number of power and ground pins (`POW_GND_LIMIT`), solder wash limits (`SOLDER_WASH_LIMIT`).

- Interactive wiring. One may prefer runingn the wiring prgram interactively. For example, if a chip type or a chip placement was not given, the wiring program may ask one to provide in place. Such a feature can be easily added in current tool.
- Dynamic allocations. Most of our data structures are dynamically allocated as needed by the program. However, there are a few structure that are not. For instance, chip table and signal table. If the program can determine how large of them for a given design at run time, some statically reserved space for them would not be wasted.
- Automating wiring tool. Currently, what we provided is not an automated tool. Fundamental description files must be supplied. If there are some errors in these files, the wiring tool cannot detect them and may produce misleading results. Worse more, it may not work at all. An ambitious plan for us is to automate the wiring process. That is, given only a drawing file and a pinlist file, the wiring tool can retrieve all necessary data for wiring without any manual intervention.

Final note: all programs are written with clear and suffcient comments. It is definitely useful in future upgradings.

7 Summary

The wiring tool we described is of general-purpose and flexible to use. It is transparent to users of those technical details such as types of boards, types of chips, and technologies of chips and so on. The transparency is accomplished by supplying data of board descriptions, chip descriptions, and then abstracting and generalizing associated objects (such as power and ground pins, adaptors, connectors, and solder-wash).

The routing algorithm employed in this tool has proved empirically that it can produce optimal results in very high probabilities.

The output of the application caontains much detail and useful information. In addition to that, the debug mode is provided and extra information is extracted during the execution so that the application error can be detected very easily.

The tool has been used for about two and half years at NYU Ultracomputer Laboratory. Many PE, memory, interconnected network components and I/O components were routed out by the wiring tool.

Some human interface of the tool needs to be improved and it seems that the more fairly interactive, the more user friendly. Automating such a tool will definitely enhance its usability and generality.

Acknowledgements: This project was supported by Ultracomputer Laboratory. Thanks go to Allan Gottlieb for his consistent support. I also would like to thank Jiarui Wang for his many suggestions and ideas during my implementing the system.

References

[1] M. Bellmore and G.L. Nemhauser. The traveling salesman problem: A survey. *Operations Research*, 16:pp538 – 558, 1968.

[2] Jon Louis Bentley. Experiments on geometric traveling salesman heuristics. Computing Science TR 151, AT&T Bell Laboratory, Murray Hill, NJ 07974, August 1990. it collects for papers.

[3] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge , MA, 1990.

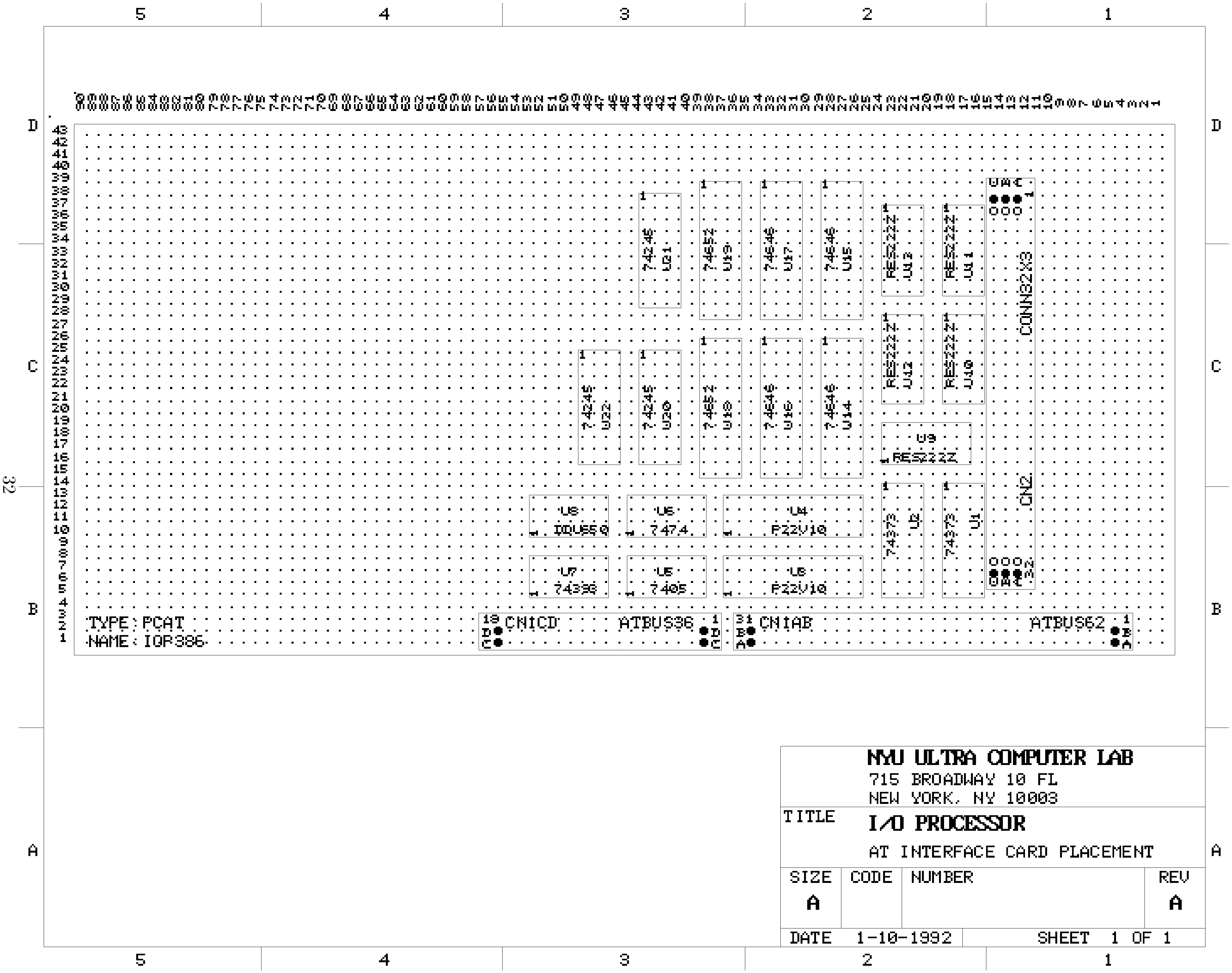
[4] DASH:. *DASH Manual*. Data I/O, 1991.

[5] Zdenek Fencl. Routing problem. *CACM*, 16(9):572 – 574, September 1973.

[6] Allan Gottlieb. An overview of the nyu ultracomputer project. Ultracomputer Note 100, New York University, Courant Institute, 715 Broadway, 10th Floor, New York, NY 10003, April 1987.

[7] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of Society of Industrial Applied Math.*, 10(1):196–210, March 1962.

[8] Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis. An analysis of several heuristic for traveling salesman probelm. *SIAM Journal of Computing*, 6(3):563–581, September 1977.



Appendix A: IOP Schematics

Appendix B: IOP Placement file

```
IOP386     PCAT
;Total 25 chips
;ChipName Orient  Col  Row
;
CN1AB        H      5    1
CN1CD        H     39    1
CN2          V     13   38
U1           V     19   14
U10          V     19   28
U11          V     19   37
U12          V     24   28
U13          V     24   37
U14          V     29   26
U15          V     29   39
U16          V     34   26
U17          V     34   39
U18          V     39   26
U19          V     39   39
U2           V     24   14
U20          V     44   25
U21          V     44   38
U22          V     49   25
U3           H     37    5
U4           H     37   10
U5           H     45    5
U6           H     45   10
U7           H     53    5
U8           H     53   10
U9           H     24   16
;***** END *******
```

Appendix C: Chip Description File — IOP related chip types

```
[DIP]
; for comments only
; type name     tech  width    pins
... ...

2764          TTL      6        28
...

68000                  TTL      9        64
68451                  TTL      9        64
6850          TTL      6        24
7400          TTL      3        14
7404          TTL      3        14
7405          TTL      3        14
...

74245                  TTL      3        20
...

74373                  TTL      3        20
...

74393                  TTL      3        14
74646                  TTL      3        24
74652                  TTL      3        24
7474                   TTL      3       14
...

AM29841A      TTL      3        24

DDU650                 TTL      3        14
...

P22V10                 TTL      3        24
RES222Z                TTL      3        16
```

```
[CONNECTOR]
;typename    row-len  width (either number (e.g. 2) or letter(e.g. C)
CONN3X32     32       C
C 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01
B 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01
A 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01
CONN25           25      2  ;assuming T25HTBLR
49 47 45 43 41 39 37 35 33 31 29 27 25 23 21 19 17 15 13 11 09 07 05 03 01
50 48 46 44 42 40 38 36 34 32 30 28 26 24 22 20 18 16 14 12 10 08 06 04 02

[PGA]
ATBUS62   TTL   2    31

  31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
B * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
A * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
ATBUS36   TTL   2    18
  18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
D * * * * * * * * * * * * * * * * * *
C * * * * * * * * * * * * * * * * * *

;
CONN32  TTL   32   3
     C   B   A
01      *   *   *
02      *   *   *
03      *   *   *
04      *   *   *
05      *   *   *
06      *   *   *
07      *   *   *
08      *   *   *
09      *   *   *
10      *   *   *
11      *   *   *
12      *   *   *
13      *   *   *
14      *   *   *
```

```
15      *   *   *
16      *   *   *
17      *   *   *
18      *   *   *
19      *   *   *
20      *   *   *
21      *   *   *
22      *   *   *
23      *   *   *
24      *   *   *
25      *   *   *
26      *   *   *
27      *   *   *
28      *   *   *
29      *   *   *
30      *   *   *
31      *   *   *
32      *   *   *


; ADPtypename          rows    cols-seen
; each width of column
[ADP]
SW1A169 34   5
0 3 3 3 3
SW2A169      34      5
0 3 3 3 3

[FULLMAP]
;name              row-num  col-num
PGA132-03           17      14
01 18 - - 35 52 - -  - -  - -  -   -
02 19 - - 36 53 - - 69 85 - - 101 117
03 20 - - 37 54 - - 70 86 - - 102 118
04 21 - - 38 55 - - 71 87 - - 103 119
05 22 - - 39 56 - - 72 88 - - 104 120
06 23 - - 40 57 - - 73 89 - - 105 121
07 24 - - 41 58 - - 74 90 - - 106 122
08 25 - - 42 59 - - 75 91 - - 107 123
```

```
09 26 - - 43 60 - - 76 92 - - 108 124
10 27 - - 44 61 - - 77 93 - - 109 125
11 28 - - 45 62 - - 78 94 - - 110 126
12 29 - - 46 63 - - 79 95 - - 111 127
13 30 - - 47 64 - - 80 96 - - 112 128
14 31 - - 48 65 - - 81 97 - - 113 129
15 32 - - 49 66 - - 82 98 - - 114 130
16 33 - - 50 67 - - 83 99 - - 115 131
17 34 - - 51 68 - - 84 100 - - 116 132
```

Appendix D: The Board Description for IOP — (abridged)

The board used for IOP wiring is a very simple board. It is called 'PCAT.bod'.

```
[TOPAREA]   ; Top connector area: x1,y1 --> x2,y2
X: 45 45
Y: 5 56
[EDGEAREA]  ; edge connector area
X: 1 1
Y: 5 56
[ROW_PHYSICAL] ; physical row mapping (your numbering is logic row)
1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
[COL_PHYSICAL] ; physical column mapping (your numbering is logic column)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 \
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 \
86 87 88 89 90
[MAPPING]  ; the mapping of the board
 1 0   000011111111111111111111111111111110001111111111111111110000000000000000000000000000000000
 2 1   0000111111111G11111111111111111111G000111111111111111P1G0GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
 0 2   000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
 3 3   000000000000000000000000000000000000000000000000000000000PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
 4 4   0000000GG0PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP11111111111111111111111111111111
 5 5   0000000GG0P1111111111111111111111111111111111111111111111111111111111111111111111111111111
 6 6   0000000GG0P1111111111111111111111111111111111111111111111111111111111111111111111111111111
 7 7   0000000GG0P1111111111111111111111111111111111111111111111111111111111111111111111111111111
 8 8   0000000GG0P1111111111111111111111111111111111111111111111111111111111111111111111111111111
 9 9   0000000GG0P1111111111111111111111111111111111111111111111111111111111111111111111111111111
10 10 0000000GG0P1111111111111111111111111111111111111111111111111111111111111111111111111111111

...... ......


40 40  0000000GG0P1111111111111111111111111111111111111111111111111111111111111111111111111111111
41 41  0000000GG0PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
42 42  000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
43 43  0000000GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
44 44  000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Appendix E: IOP's lsw file – (abridged)

```
; Wiring Output Part 1/3: statistics, wiring runs and/or solder_wash
; NYU Ultracomputer Lab
; Date: 01/26/92

Statistics of Circiut Chips and Types
=======================================
    25  Chips
    13  ChipTypes

CHIP_NAME             CHIP_TYPE
----------            -------------
CN1AB                 ATBUS62
CN1CD                 ATBUS36
CN2                   CONN32
U1                    74373
U10                   RES222Z
U11                   RES222Z
U12                   RES222Z
U13                   RES222Z
U14                   74646
U15                   74646
U16                   74646
U17                   74646
U18                   74652
U19                   74652
U2                    74373
U20                   74245
U21                   74245
U22                   74245
U3                    P22V10
U4                    P22V10
U5                    7405
U6                    7474
U7                    74393
U8                    DDU650
U9                    RES222Z

-----------------------------------
CHIP_TYPE     COUNT
----------    -----
7405            1
74245           3
74373           2
74393           1
74646           4
74652           2
7474            1
ATBUS36         1
ATBUS62         1
```

```
CONN32         1
DDU650         1
P22V10         2
RES222Z        5

Chips and their Placement           BOARD:          IOP386
                                    BOARD PART NO.: PCAT
=================================================================
CHIP_NAME                                   ROW(X)       COL(Y)
-----------                                 ------       ------
CN1AB                 , ASSIGNED TO SOCKET :   1 5
CN1CD                 , ASSIGNED TO SOCKET :   1 39
CN2                   , ASSIGNED TO SOCKET :   38        13
U1                    , ASSIGNED TO SOCKET :   14        19
U10                   , ASSIGNED TO SOCKET :   28        19
U11                   , ASSIGNED TO SOCKET :   37        19
U12                   , ASSIGNED TO SOCKET :   28        24
U13                   , ASSIGNED TO SOCKET :   37        24
U14                   , ASSIGNED TO SOCKET :   26        29
U15                   , ASSIGNED TO SOCKET :   39        29
U16                   , ASSIGNED TO SOCKET :   26        34
U17                   , ASSIGNED TO SOCKET :   39        34
U18                   , ASSIGNED TO SOCKET :   26        39
U19                   , ASSIGNED TO SOCKET :   39        39
U2                    , ASSIGNED TO SOCKET :   14        24
U20                   , ASSIGNED TO SOCKET :   25        44
U21                   , ASSIGNED TO SOCKET :   38        44
U22                   , ASSIGNED TO SOCKET :   25        49
U3                    , ASSIGNED TO SOCKET :   5 37
U4                    , ASSIGNED TO SOCKET :   10        37
U5                    , ASSIGNED TO SOCKET :   5 45
U6                    , ASSIGNED TO SOCKET :   10        45
U7                    , ASSIGNED TO SOCKET :   5 53
U8                    , ASSIGNED TO SOCKET :   10        53
U9                    , ASSIGNED TO SOCKET :   16        24

SIGNAL        CHIPS CONNECTED
=======       =================
LINE A0                      2 CONNECTIONS, WIRE LENGTH =  1.0 IN.
     10  27    13  21    0.7 IN  U4-P22V10-11            U2-74373-19
LINE A1                      3 CONNECTIONS, WIRE LENGTH =  1.0 IN.
     10  21    08  27    0.6 IN  U2-74373-16             U3-P22V10-14
     08  27    10  28    0.2 IN  U3-P22V10-14            U4-P22V10-10
LINE ACK-                    3 CONNECTIONS, WIRE LENGTH =  2.5 IN.
     08  28    08  40    1.2 IN  U3-P22V10-15            U5-7405-9
     08  40    20  46    1.3 IN  U5-7405-9               U22-74245-15
LINE B0OE-                   3 CONNECTIONS, WIRE LENGTH =  2.9 IN.
     08  36    23  31    1.6 IN  U3-P22V10-23            U16-74646-21
     23  31    36  31    1.3 IN  U16-74646-21            U17-74646-21
LINE B1OE-                   3 CONNECTIONS, WIRE LENGTH =  3.0 IN.
     08  35    23  26    1.7 IN  U3-P22V10-22            U14-74646-21
```

```
     23  26    36  26    1.3 IN  U14-74646-21           U15-74646-21
LINE BALE                    4 CONNECTIONS, WIRE LENGTH =  2.3 IN.
     05  16    05  21    0.5 IN  U1-74373-11            U2-74373-11
     05  21    02  32    1.2 IN  U2-74373-11            CN1AB-ATBUS62-B28
     02  32    05  37    0.6 IN  CN1AB-ATBUS62-B28      U3-P22V10-1
LINE BD.0                    4 CONNECTIONS, WIRE LENGTH =  1.2 IN.
     29  29    29  34    0.5 IN  U15-74646-11           U17-74646-11
     29  34    29  39    0.5 IN  U17-74646-11           U19-74652-11
     29  39    29  41    0.2 IN  U19-74652-11           U21-74245-11
LINE BD.1                    4 CONNECTIONS, WIRE LENGTH =  1.2 IN.
     30  29    30  34    0.5 IN  U15-74646-10           U17-74646-10
     30  34    30  39    0.5 IN  U17-74646-10           U19-74652-10
     30  39    30  41    0.2 IN  U19-74652-10           U21-74245-12


......        ......   ......   ......
......        ......   ......   ......


LINE WCK                     3 CONNECTIONS, WIRE LENGTH =  2.6 IN.
     13  36    26  39    1.3 IN  U4-P22V10-23           U18-74652-1
     26  39    39  39    1.3 IN  U18-74652-1            U19-74652-1
LINE WEN                     2 CONNECTIONS, WIRE LENGTH =  1.0 IN.
     08  26    13  34    0.9 IN  U3-P22V10-13           U4-P22V10-21
LINE WRD-                    3 CONNECTIONS, WIRE LENGTH =  2.3 IN.
     13  35    23  36    1.0 IN  U4-P22V10-22           U18-74652-21
     23  36    36  36    1.3 IN  U18-74652-21           U19-74652-21

CONNECTORS: LINE ASSIGNMENT ON BOARD IOP386

NUMBER OF RUNS        LENGTH IN INCHES
       99                      1
       83                      2
        9                      3
        8                      4
        4                      5
        1                      6

TOTAL NUMBER OF RUNS = 205
TOTAL LENGTH = 350
TOTAL NUMBER OF WIRES = 353
```

Appendix F: IOP's wls file — (abridged)

```
10    27     13    21    A0               IOP386     000001
10    21     08    27    A1               IOP386     000002
08    27     10    28    A1               IOP386     000003
08    28     08    40    ACK-             IOP386     000004
08    40     20    46    ACK-             IOP386     000005
08    36     23    31    B0OE-            IOP386     000006
23    31     36    31    B0OE-            IOP386     000007
08    35     23    26    B1OE-            IOP386     000008
23    26     36    26    B1OE-            IOP386     000009
05    16     05    21    BALE             IOP386     000010
05    21     02    32    BALE             IOP386     000011
02    32     05    37    BALE             IOP386     000012
29    29     29    34    BD.0             IOP386     000013
29    34     29    39    BD.0             IOP386     000014
29    39     29    41    BD.0             IOP386     000015
30    29     30    34    BD.1             IOP386     000016
30    34     30    39    BD.1             IOP386     000017
30    39     30    41    BD.1             IOP386     000018

......   ......   ......
......   ......   ......


13    36     26    39    WCK              IOP386     000349
26    39     39    39    WCK              IOP386     000350
08    26     13    34    WEN              IOP386     000351
13    35     23    36    WRD-             IOP386     000352
23    36     36    36    WRD-             IOP386     000353
```

Appendix G: Man Page (see next page)

NAME

dowire, wirepv4 – wiring tools

SYNOPSIS

dowire [**–d** <**descript-dir**>] [**–c**] [**–m**] [**–D**] [**–L**] pinfile
wirepv4 [**–d** <**descript-dir**>] [**–c**] [**–m**] [**–D**] [**–L**] pinfile

DESCRIPTION

Dowire or *wirepv4* reads "pinlist" file produced by *DASH* schematics post processors (see Reference (1)), and based on the information of specified environment files such as chip description file (usually with the name of GENERIC.cdf), board description file (usually with the name of <boardname>.bod) and placement file (with the name of <filename>.plc, produces out the wiring lists for all signals of the pinlist with the objectives of minimizing the length of the wires and under the constraints imposed by some conditions, such as signal flow directions for those chips with TTL and ECL technology. For example, for TTL lines, the optimal routing is simply the shortest path. However, for ECL lines, the run path must visit the various types of pins in the specified sequence:

I/O output pins (signals coming onto board) --> output pins --> input pins --> I/O input pins (signal leaving board).

The only distinction between *dowire* and *wirepv4* is in treating differently their input pinlist files. *Dowire* only processes the pinlist format below version 4 (the extension of such pinlist file name is .pin); while *wirepv4* (from the command name) can understand pinlist format version 4 and above only (with filename extension of .pv4). It is worthwhile to notice that the version 4 format of pinlist (also called enhanced pinlist) is of hierarchical structure and the one below version 4 is flat.

At least three output files are produced when *dowire* or *wirepv4* is executed (in following, *fname* is derived from pinfile name of the command line):

fname.lsw — give a report of chips, their types and the number of their occurrences. It also contains detail connection information for each run (line) of signals (such as how many connections, wire length and pathes of each connection).

fname.csb — displays diagrams where chips are placed on the board and the area occupied. Diagrams for signals are also produced.

fname.wls — contains the final output for manufactures (point-to-point connection each line).

The **–d** option specifies the directory of environment files mentioned above and listed below in ENVIRONMENT section. If this option is not provided, the current working directory is the default.

The **–c** option informs the system to produce cyclic lists for some connections (currently, only for those wires with only two nodes, which means that two opposite direction wires are generated rather than one direction in the usual case).

The **–m** option tells the tool to be aware of multi-technology used in the design (not only TTL, but also ECL and so on).

The **–D** option will force the system to enter into the debugging mode. Under this mode, several additional outputs are generated for debugging. They are

EACHPIN.dbg — contains each pin with its physical location on the board;
CHIPS.dbg — contains each chip with its initial(1st pin) location on the board; and
SIGNALS.dbg — all signals and their connections.

The **–L** option is used for generating <*fname*>.lsw output in landscape mode. The default mode is portrait.

ENVIRONMENTS

Dowire or *wirepv4* cannot run unless following environment files are available :

GENERIC.cdf — chip description file. It must be used for all applications (or rather designs).

PINLIST (<*fname*>.<pin>) pinlist file, the data file generated from *DASH*; one application must have its own pinlist file.

<*fname*>.pin — for pinlist format below version 4 only.

<*fname*>.pv4 — for enhanced pinlist (version 4 and above).
<boardname>.bod — board description file; different boards must have different description files. It has nothing to do with applications.
<*fname*>.plc — chip placement description file; one application must have its own placement file.

The description format of above environment files and other technical details can be found in Reference (2).

REFERENCES

(1) Manual of DASH Schematic Layout Editor, DATA I/O Inc. 1991

(2) Guoying Chen, ‘‘A Wiring Tool for Ultracomputer", UCR#176, Ultracomputer Lab Tech Note, NYU, April, 1991.

FILES

GENERIC.cdf and
other files listed in the ENVIRONMENT section.

AUTHORS

Guoying Chen, the main developer of wiring tool.

The parser for pinlist below version 4 (i.e. for dowire program) was provided by Jiarui Wang, who also provided many useful comments and suggestions when wiring programs were written.

BUGS

Some inappropriate placement will cause ‘‘core dump".
Also, irregular boards must be carefully and precisely
described in its board description file.