A Thesis of

Hardware Emulation Board Based on Field Programmable Gate Arrays (FPGAs) and Programmable Interconnections

by

LO Wing-yee

Presented to

Department of Electronic Engineering
and
Graduate School

in

Partial Fulfilment of the Requirements for Degree of

Master of Philosophy in Electronic Engineering

in

The Chinese University of Hong Kong

June, 1994.

Abstract

Based on Field Programmable Gate Arrays (FPGAs), an in-circuit rapid prototyping system can be built. However, just hardwiring the FPGAs together is inflexible and may waste the IO resources. One can use programmable interconnect switches to enhance the connectivity between FPGAs. Added with a microprocessor and RAMs, almost any digital systems can be realized. Moreover, automated software tools are required in circuit path analysis, FPGAs partitioning, FPGAs configuration data downloading and interconnectable switches programming. This thesis describes the hardware and software aspects of a low-cost, reconfigurable and flexible hardware emulation board. Furthermore, different bus configurations between the FPGAs are analysed to find the best configuration. This board represents a new direction in building a rapid prototyping system.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Circuit design projects can be divided into four stages: design, verification, implementation and validation. The design stage includes choosing the best alternative from all the possible initial concepts and producing a logical description such as schematic diagram, high-level description language or circuit netlists. In the verification stage, the logical description is checked if it meets the specification written by the user. Then it enters the implementation stage which entails tooling and building the prototypes by using TTL chips or ASIC chips by the vendors. The last validation stage takes place when all prototype, peripheral, PCB and software's all are brought together and are plugged into the target system. This stage is important since it puts all components which were designed and tested separately together and have to interact with each other in the target system [1].

1.1 Traditional Design Validation

Traditionally, there have been three choices available for design validation, namely breadboarding, software simulation and silicon prototyping. However, each of these alternatives has its own disadvantages.

The breadboards are implemented by using TTLs, PALs or MSI off-shelf logic chips on a board. It works quite well only for designs less than a few thousands gates. For large designs, the time taken to build such a breadboard is very long. It is not worth especially in this competitive market. Since it is very likely that the design will not function the first time, the designers need to modify the designs. However, slightest changes in designs will cause large effort of rework.

The apparently good choice of system-level simulation, indeed, cannot allow enough real-time operation to ensure correct functionality. It takes months to simulate a few seconds

of real-time operation of a moderately complex system [2]. Statistics shows that over 50% fail to work in the target system even though over 90% of test vectors pass.

The last choice is for ASIC applications -- silicon prototyping. The obvious disadvantage of silicon prototyping is that it is very difficult to probe inside the chip for debugging. Once it is in silicon, modification cannot be made to evaluate other better alternatives. The long turn around time and high Non-Recurring Engineering (NRE) cost make it no longer a good choice for design validation.

1.2 In-Circuit Rapid System Prototyping

A new technology called in-circuit hardware emulation system can alleviate the above disadvantages. It automatically produces hardware prototypes of chip designs from netlists and requires no effort in circuit design modification. The underlying key is the use of the Field Programmable Gate Arrays (FPGAs). FPGA is a device in which the final logic structure can be implemented by loading the internal RAM with configuration data, without going through the I.C. fabrication process. It combines the programmability of a PLD and the scaleable interconnection structure of mask programmable gate array (MPGA).

Figure 1 shows a conceptual diagram of a typical FPGA. As depicted, it consists of a two-dimensional array of uncommitted Configurable Logic Blocks (CLBs) in which the logic design can be resided. They contain both combinational and sequential logic. CLBs are connected by the interconnect resources. The interconnect comprises segments of wire which may be of various length. Present in the interconnect are programmable switches serve to connect CLBs to wire segments, or one wire segment to another wire segment. Logic circuits are implemented in the FPGA by partitioning the logic into the CLBs and then interconnecting the blocks as required via the switches. Around the CLBs are the

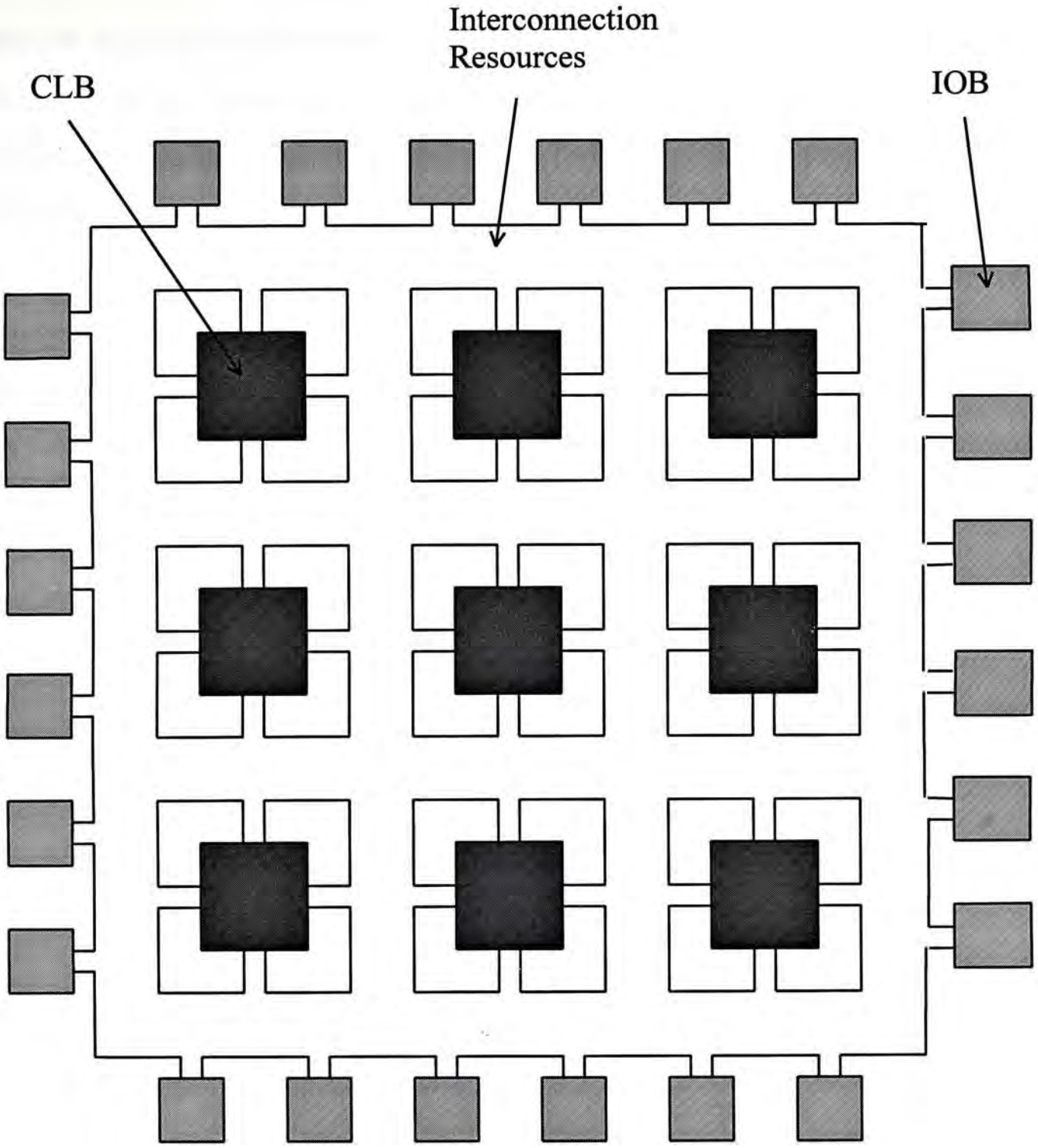


Fig.1. A conceptual FPGA.

Input/Output Blocks (IOBs). They provide interface between external package pins and the internal logic. They can be programmed to act as input, output or bidirectional.

Automated CAD tools are provided to ease the implementation of the circuit in the FPGA. Figure 2 shows all the steps involved in the implementation process. The first step is to produce a logical description. It can be a schematic diagram, a VHDL description or a

Boolean expressions specification. The logical description is then translated into a netlist format which the software tools can understand. Afterwards, the logic optimization tools optimize the area and speed of the final circuit. Technology mapping transforms the circuit into FPGA CLBs and IOBs. Upon completion of mapping the CAD system will decide where to place each CLB in the FPGA array such that total length of interconnect required is minimized. Routing involves assignment of FPGA's wire segments and choosing the programmable switches to establish the required connections. The final step is to download the configuration data to the programming unit and the circuit is realized. Since it uses actual

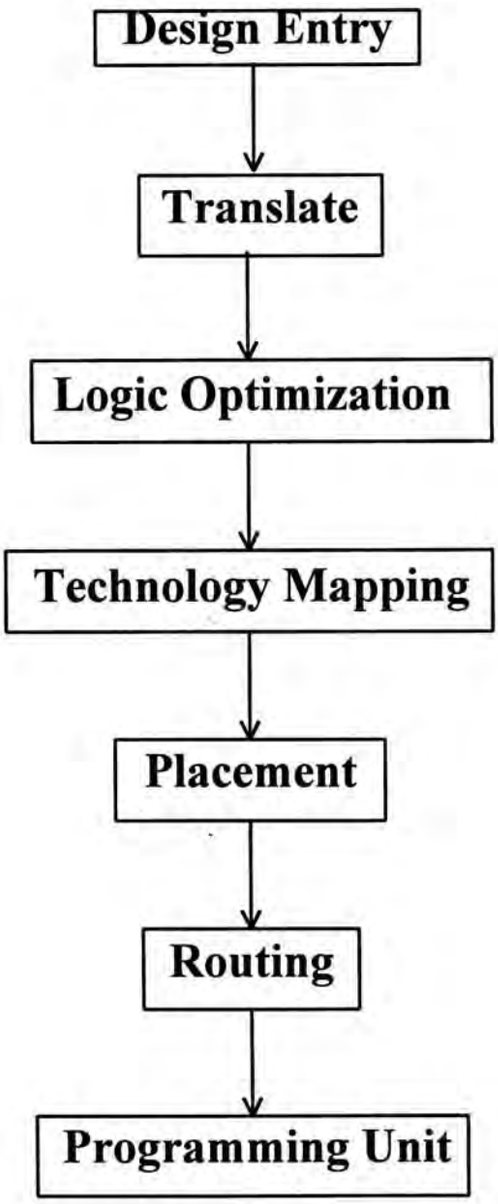


Fig.2. A CAD system for FPGA.

hardware gates to emulate the design, it offers real-time operation. Such a system offers lower cost and faster implementation. To modify and change the design, the only thing we have to do is to download another set of configuration data into the FPGA. As a result, much innovation can be obtained in designs [3].

1.3 A Summary of Prototyping Systems Available

Although using FPGAs have many advantages over using the traditional circuit validation methods, FPGAs still have limitations. First of all, it has limited amount of gate count in each FPGA. Designers can prototype small designs of several thousand gates. For designs larger than 5,000 gates, they have to use several FPGAs in which a lot of manual intervention is required. For instance, it is necessary to manually partition the design among FPGAs. Manual partitioning, actually, can be very erroneous since there are too few pins on most FPGAs compared with the number of gates inside. Hence, when the designs are partitioned across multiple FPGAs, the usable gates per FPGA goes down rapidly. Besides, wirewrapping makes it very inconvenient to deal with design changes and rework. Another limitation is that as design grows more complex, many applications require a lot of memory. Yet FPGAs can offer only a little amount of memory.

Over these few years, there are several companies and an academic institution that developed their rapid prototyping board based on FPGAs. The all try to overcome the limitations of FPGAs one way or another.

One of the products from the market is Qucikturn's RPM system. It has a multiplexed architectural rack called the Enterprise Emulation System which consists of logic emulation modules and a reprogrammable backplane. In each module, there is an array of Xilinx XC3090 FPGAs interspersed with the custom Multiplex Interconnect chips to connect the FPGAs. Up to eleven modules can be plugged into the rack. With the Interconnect Module,

the RPM can cluster at most 22 Enterprise Systems which can emulate up to six millions gates. An optional component adapter card permits the use of standard components such as memory devices. The modular and expandable capabilities of Enterprise are enhanced by the Automatic Design Partitioner (ADP). This software partitions logic into netlists fitting within a single emulation system and also automates the clustering of multiple Enterprise Systems [4]-[7].

Another competitor, PiE's Mars II has a similar architecture but they claim that they have developed the innovative timing-driven partitioning software that automates the logic-emulation process [8]. At the same time, Intel's ASIC In-Circuit-Emulation (ICE) is tailored for verifying ASIC design especially incorporating with micro-controller core and peripheral cells [9]-[10].

Research work in the North Carolina State University has developed an AnyBoard for hardware emulation. A set of FPGAs built on a single 13-by-4 inch card can be inserted into the PC slot. The FPGAs have local buses between adjacent FPGAs and global bus which connected to all FPGAs for high fan out net. Besides, each FPGA is connected with a RAM for memory intensive designs. The PC interface allows data to pass between the AnyBoard and the host PC system for downloading and readback data to and from the array [11].

1.4 Universal Prototyping Board (UPB)

It is no doubt that Quickturn's RPM system is powerful and flexible. However, it is too expensive and too large scale in gate count for most of the designs in the markets. For the AnyBoard, the interconnection between FPGAs is not flexible. As previously mentioned, there are too few pins on most FPGAs compared with the number of gates inside. Hence, partitioning the design across multiple FPGAs will bring the nets in the circuit out across the FPGAs. Just hardwiring the FPGAs may be IO wasteful and inflexible. To clarify the point,

in the AnyBoard, only local bus and global bus are available. If a net connected to two CLBs and these CLBs are assigned to FPGA 0 and FPGA 2 (non-adjacent FPGAs), intermediate

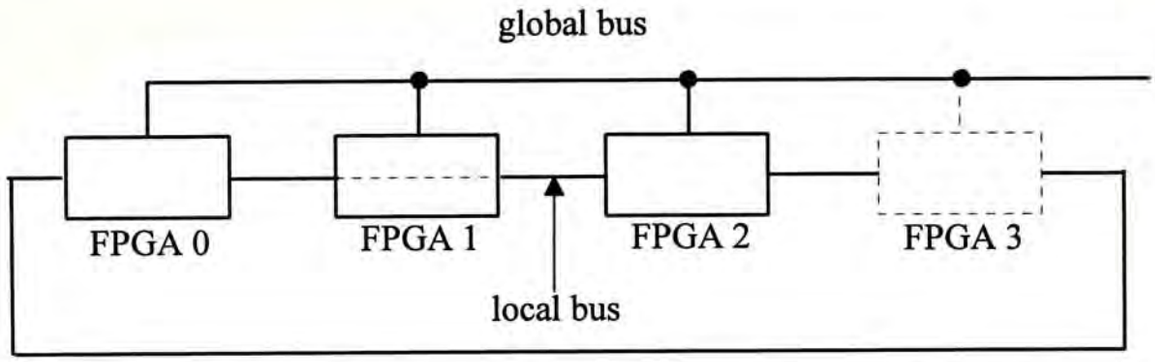


Fig. 3. Bus interconnection in AnyBoard.

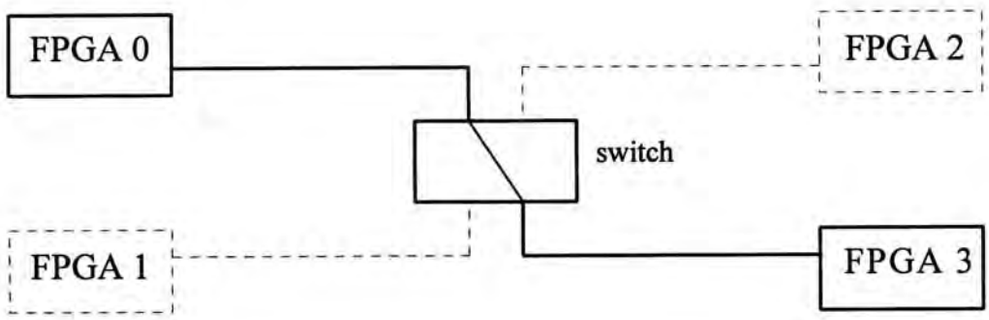


Fig. 4. Using programmable switch saves IOs in UPB.

local data path in FPGA 1 must be connected if using local bus (Figure 3). Using the global bus will also work as the global bus connected to all the FPGAs. However, four IOs are committed in both cases, wasting two IOBs in FPGAs. If an intermediate switch is used to connect the CLBs, only two IOs are used (Figure 4).

Hence, it is in practice that the interconnectivity between FPGAs should be emphasized in designing a rapid prototyping system. The Universal Prototyping Board (UPB) we developed has the flexibility of the Quickturn 's RPM system and the lower cost of the AnyBoard. This UPB is based on four FPGAs and cross-point programmable analog switches. Besides, the UPB has the following characteristics:

- FPGAs interconnected by hardwired and programmable buses,
- RAMs available for memory-intensive designs,
- microprocessor for downloading configuration data of FPGAs, testing and itself as part of emulation,
- software tools to automate design process,
- cheap and large gate count available,
- expandable hardware for more complex designs.

In this chapter, the disadvantages of the traditional design prototyping methods and new technologies of in-circuit rapid prototyping systems were discussed. In chapter 2, the hardware aspects of the UPB will be described. The software tools provided to automate the validation process will be talked in chapter 3. In order to find the best configuration of the UPB, a structure analysis was designed and will be outlined in chapter 4. In chapter 5, the results of the analysis and the best configuration of the UPB concluded from the analysis will be presented. Next, the future direction of this kind of rapid prototyping system will be talked. Lastly, a conclusion will be drawn in chapter 7.

2. HARDWARE DESIGNS

2.1 Bus Interconnection

A good interconnection structure between FPGAs in the emulation board should contain various kinds of buses for different nets so that the signal can pass across FPGAs with little delay and that no IO will be wasted. To achieve this, two kinds of buses are available in the UPB: the fixed buses and the programmable buses. The fixed buses are the hardwired buses while the programmable buses are the buses connected via programmable switches. As the fixed buses are hardwired, the delay for signal to travel along is very small. Therefore, it is mainly for critical nets, high fan-out nets, nets with high skew-rate and nets connected to CLBs which are assigned to neighbouring FPGAs. On the other hand, the programmable bus is used to saves the IOBs in the FPGAs. So the nets connected to CLBs which assigned to two next adjacent FPGAs or three consecutive FPGAs should be mapped to the programmable buses; only in this way, the signal need not to pass through an intermediate FPGA in which more IOs are used than desirable.

2.1.1 Fixed buses

Totally there exists three kinds of fixed buses in the UPB. They are local bus, global bus and board IO. It is normal to think that if the design is too large to fit on a single FPGA, the design will be partitioned and placed on the adjacent chips rather than on two widely

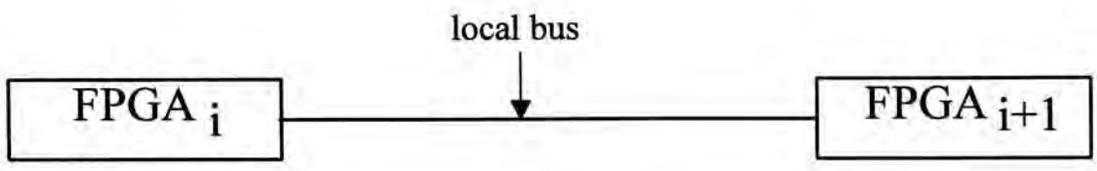


Fig. 5. Local bus between any two adjacent FPGAs in UPB.

separated chips that the signals have to travel a long way through the third FPGA. Hence, local bus is basically used to connect any two adjacent FPGAs together (Figure 5).

The global bus, on the other hand, connects all four FPGAs together. It is mainly for the high fan-out nets and the high skew-rate signals (Figure 6). At the same time, this global bus is connected to the outside for the system input and output.

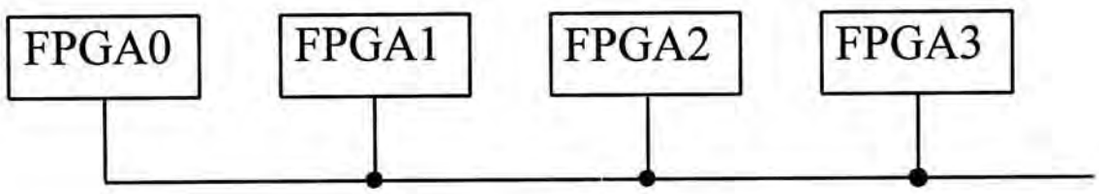


Fig. 6. Global bus in UPB.

The board IO, as the name implied, is the interface between the emulation board and the external world. Several pins of each FPGA are connected to the outside. (Figure 7). It should be noted that it is the global bus and the board IO that contribute to the system interface where the UPB can be expanded by cascading more boards together for more complex design. Combining all these buses, the overall connection of the fixed bus as shown in Figure 8.

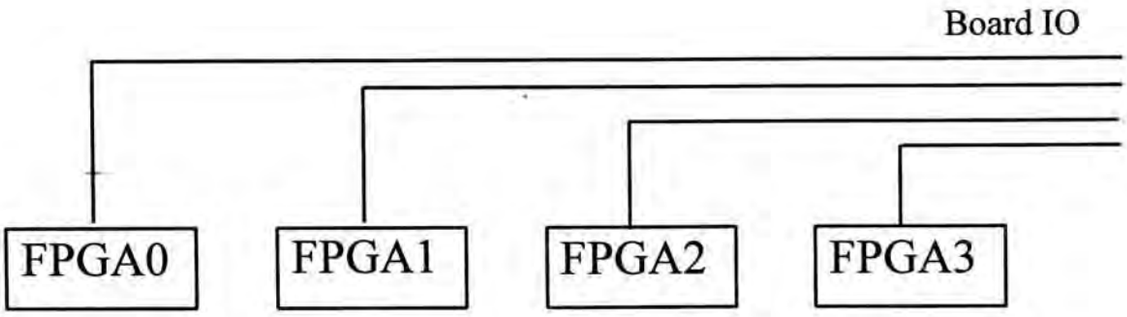


Fig. 7. Board IO in UPB.

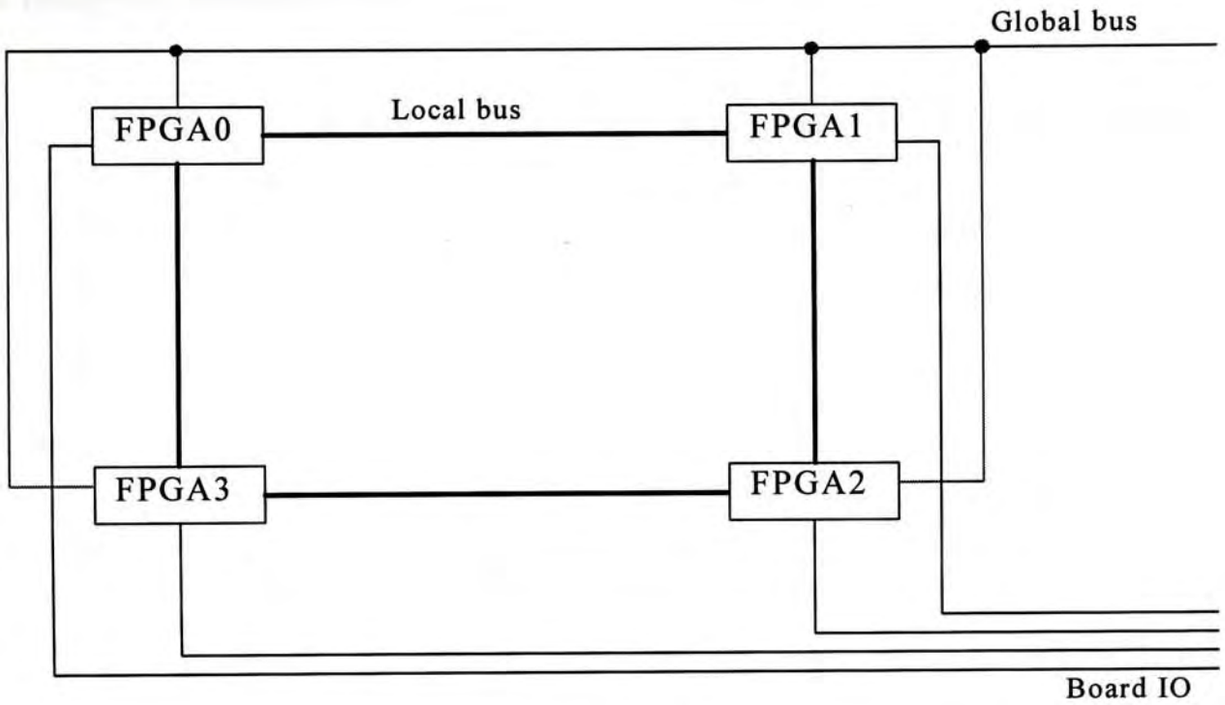


Fig. 8. Fixed bus in UPB.

2.1.2 Programmable buses

The programmable buses connect signals that span more than two FPGAs. Although we can use the reprogrammable interconnection contained in the FPGAs as the connection resources, this will mean less usable IOBs available for the designs. Instead we use several cross-point programmable analog switches (74HCT22106), then the connectivity between multiple FPGAs will increase.

Figure 9 shows one usage of the programmable switches. They connect any two next adjacent FPGAs together. Another usage of the switches is to connect any three consecutive FPGAs together. Figure 10 shows the connection method. In this method, either two out of three consecutive FPGAs are hardwired and the remaining FPGA is connected through the programmable switches.

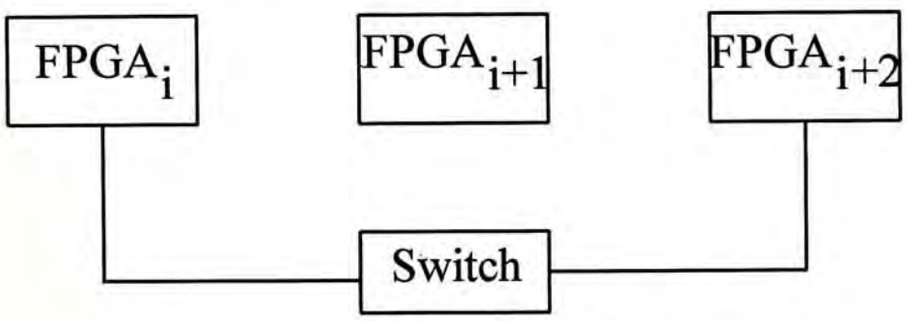


Fig. 9. Programmable buses connecting two next adjacent FPGAs in UPB.

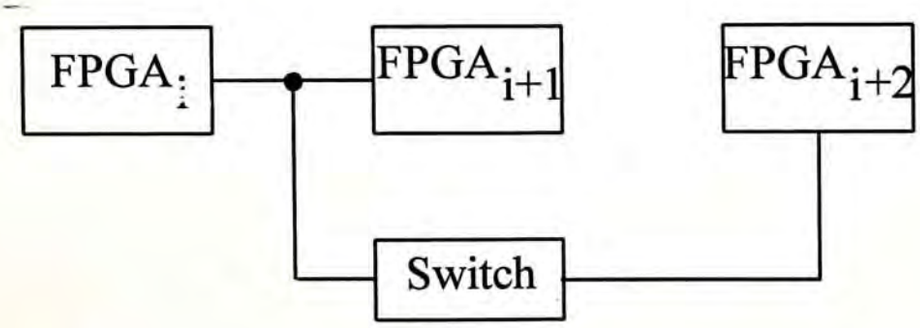


Fig. 10. Programmable buses connecting three consecutive FPGAs in UPB.

If all the FPGAs are connected in this way, a symmetrical interconnection through the programmable buses are obtained. It is illustrated in figure 11. There are 8 paths: 1 - 8 connected to the switches. The paths 1, 3, 5, 7 (odd path) are connected from each FPGA 0, 1, 2, 3 while the paths 2, 4, 6, 8 (even path) are 'T-connected' from each FPGA and its two adjacent FPGAs to the switches.

Two examples are given to illustrate the use of the programmable buses. For instance, if a net needs to connect two next adjacent FPGA 1 and FPGA 3, path 3 and path 7 (2 odd paths) are required to be programmed to connect. Similarly, if we want to connect three consecutive FPGA 1, FPGA 2 and FPGA 3, either path 3 and path 6 or path 4 and path 7 (1 odd and 1 even paths) have to be programmed. The following table shows which paths should be programmed to connect for all possible FPGAs connections.

FPGAs to be connected	Paths need to be programmed to connect
0, 2	(1, 5)
1, 3	(3, 7)
0, 1, 2	(1, 4) or (2,5)
1, 2, 3	(3, 6) or (4, 7)
2, 3, 0	(1, 6) or (5, 8)
3, 0, 1	(2, 7) or (3, 8)

Table 1. Paths need to be programmed to connect FPGAs.

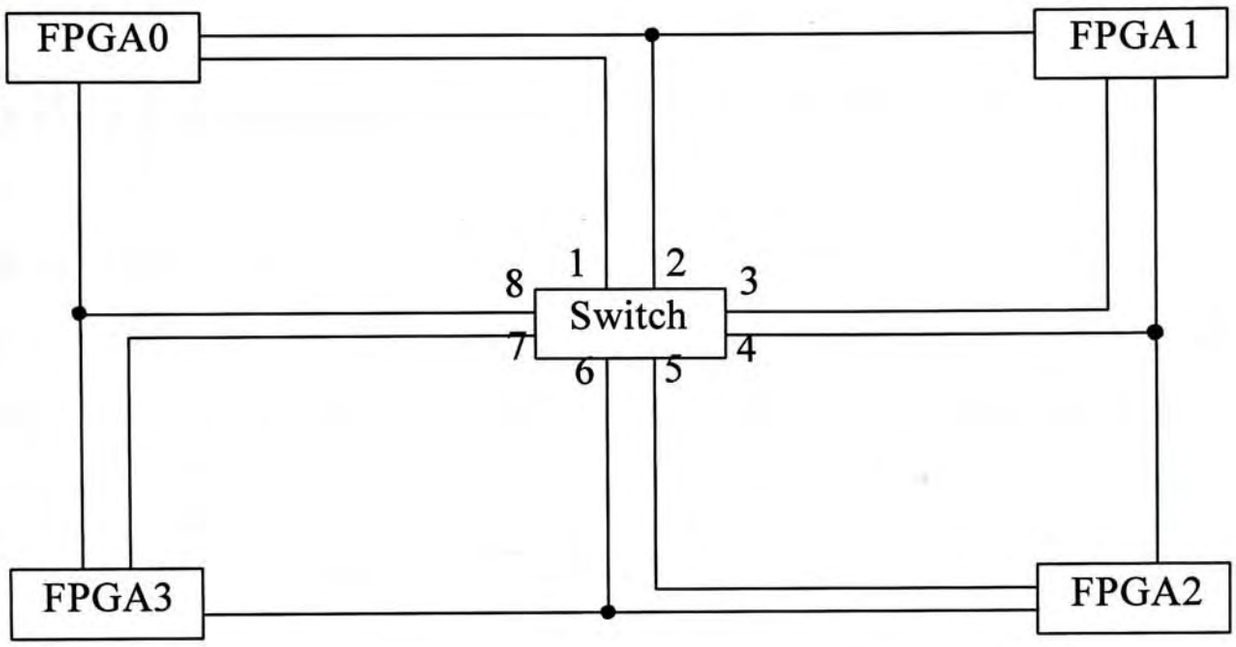


Fig. 11. The pogrammable buses in UPB.

2.2 Architectural Features

2.2.1 Field programmable gate array

There are four Xilinx XC3042 FPGAs in the UPB as the prototype implementation technology. Although not all the gates provided are usable because of the incomplete routability of the internal blocks in each FPGA (see 3.2.2), it is estimated that the usable gate count is approximately 10, 000 gates. To download the configuration data, FPGA 0 is configured in the peripheral mode, FPGA 1 - 3 are configured in the slave mode. There is a microprocessor (see 2.2.2) in the UPB to download the data to the FPGAs. It gets the data from the serial port of a PC and then downloads to FPGA 0. The FPGA 0 is in fact acting as the leading device in the daisy chain. When it receives excess configuration data, it will pass to other slave FPGAs and so on [12] - [13] (figure 12).

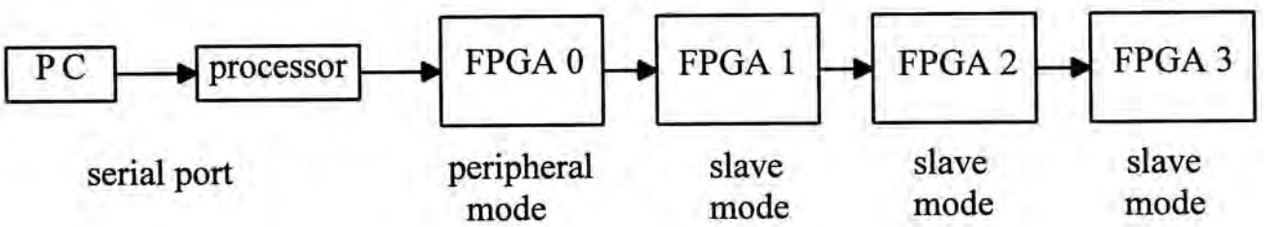


Figure 12. Downloading configuration data to FPGAs in UPB.

2.2.2 Microprocessor

In the UPB, there is a Motorola 68000 (68000μp) microprocessor [14] - [15]. With this microprocessor, two modes: testing mode and emulation mode are provided. These two modes are controlled by a toggling mode-select switch. At any time, the two modes is interchangeable by toggling the switch.

In the testing mode, the operation of UPB is controlled by the monitor program. The microprocessor can download the configuration data into the FPGAs. Besides, it can also read and write data from and to the RAMs. It is very useful when the design is in its developing stage since it replaces part of the design involving the control of accessing RAMs. Therefore the user can validate the control logic of accessing RAMs after the rest of the system is proved to work as expected. Moreover, apart from the normal operation (called normal mode) of the system, the whole system can debug the design in the trace mode. Designers can trace or single-step their designs in this mode for debugging and then verify them in normal mode after correcting their designs. The last but not the least, designers can program any switches they like. It does not mean that designers are required to program the switches for the interconnections between the FPGAs after partitioning. The software tools provided will program it automatically (see 3.4). Rather, designers can program the switches at any time for their own purposes.

In the emulation mode, the operation of UPB is controlled by the instructions given by the designers. The firmware are located in another sets of EPROM. By toggling the mode-select switch, this firmware is selected. In other words, the microprocessor itself can be a part of the emulation system.

2.2.3 Memory

Three 32 x 8 RAMs each of which is connected to FPGA 0 - 2 to provide storage elements for memory-intensive design. Figure 13 shows how the memory design may be realized in the UPB. In this example, it is assumed that only RAM 1 is used in the design. FPGA 3 contain logic of an address generator, such as a counter, to drive all RAMs while FPGA 1 contains logic to read, write and control the RAM data. At least one intermediate local path is required to connect both logic blocks to provide communication between them.

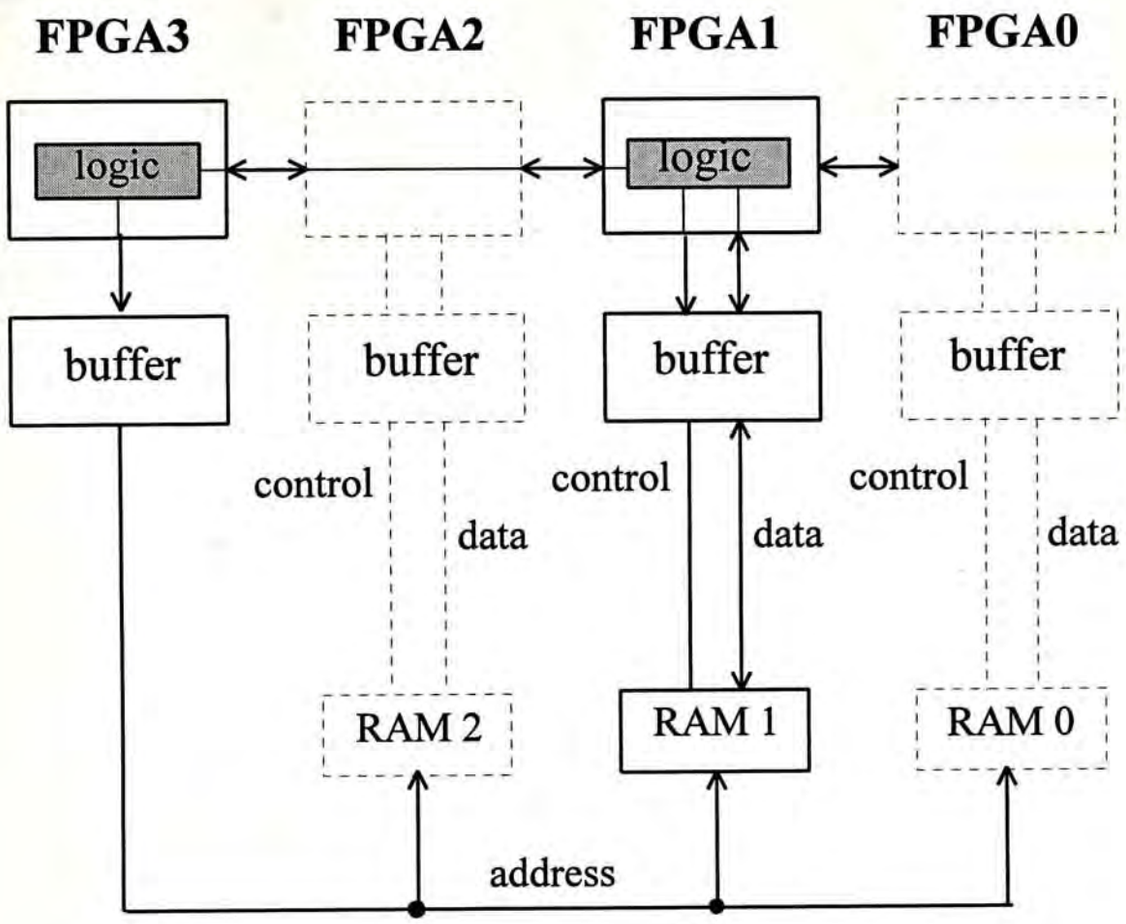


Fig. 13 Memory in UPB.

It should be noted that some of the programmable buses serve two purposes. Part of the address and data buses come from these programmable buses. If the RAMs are used in the UPB, these buses are used as the address buses and data buses for the RAMs. If not, they are used as normal programmable buses. This arrangement can maximize the number of IO pins on each FPGA for interconnection between the FPGAs. The buffers between the FPGAs and RAMs are provided to prevent bus contention. It is because the 68000μp can also access the RAMs. So the data and address buses in RAM are physically connected to both FPGAs and 68000μp. Bus contention would occur if both parties intended to access at the same time while there was no buffers between them. (see 2.2.4)

2.2.4 Buffers

As stated before, RAM can be accessed by both FPGAs and 68000μp, buffers must be provided to prevent bus contention. Apart from the RAMs, FPGA 0 can also be accessed by both 68000μp (for downloading configuration data) and RAM 0, buffer is required to connect between them. Otherwise, bus contention may occur if both parties access FPGA 0 at the same time. Figure 14 and Figure 15 show all the buffers required in isolating address and data lines between the corresponding two parties connected to them.

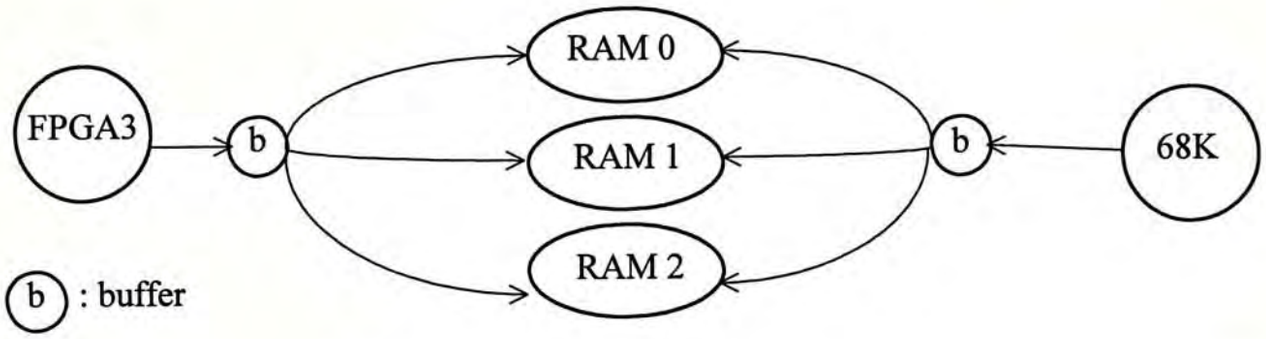


Fig. 14. Buffers to prevent address bus contention.

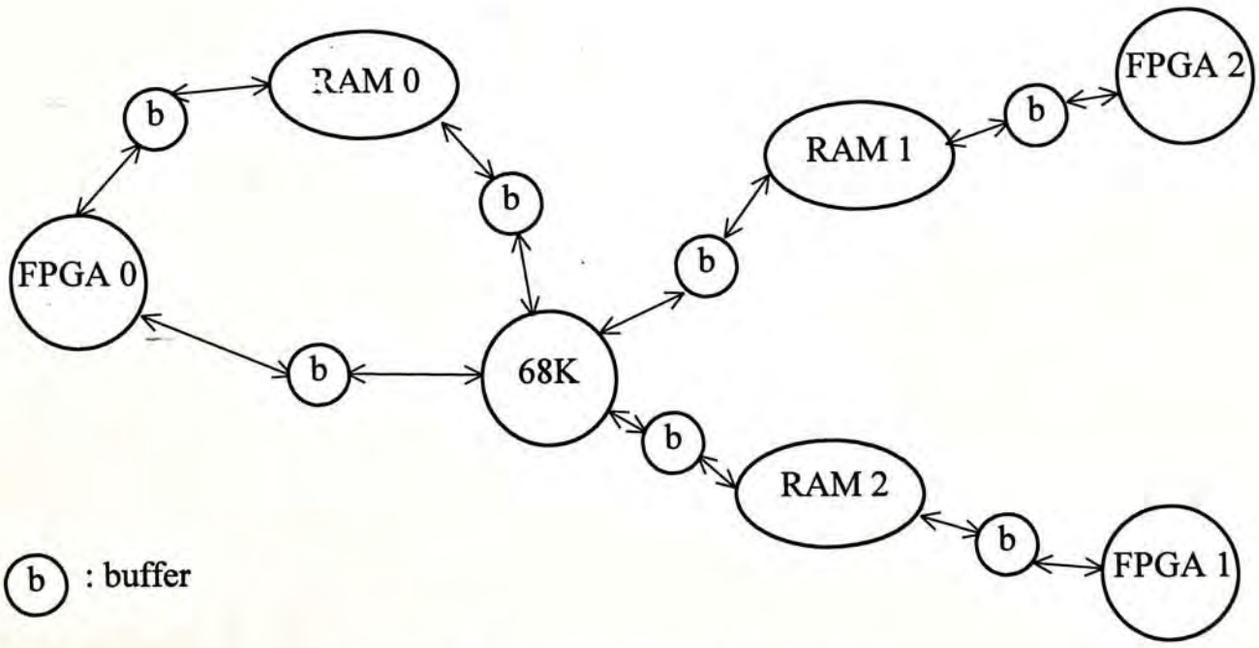


Fig. 15 Buffers to prevent data bus contention.

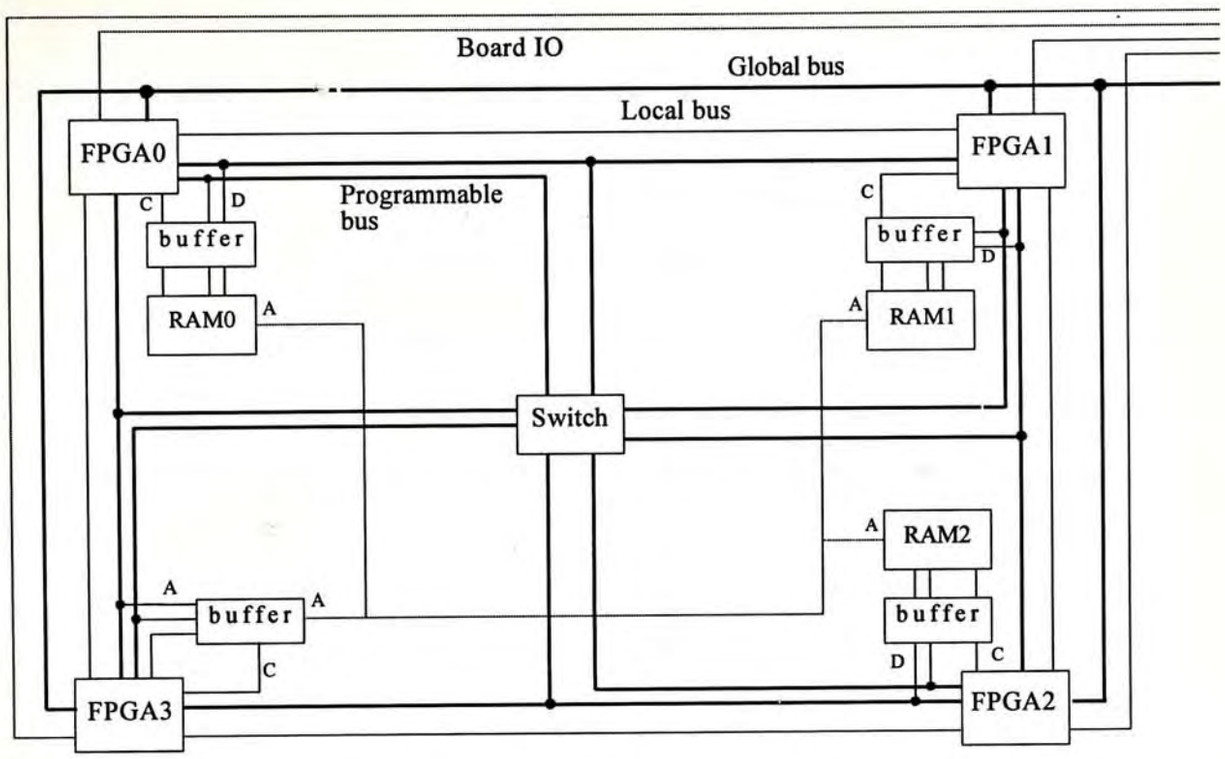


Fig. 16. Overall structure of UPB.

Figure 16 shows the overall structure of the UPB including the FPGAs, RAMs buffers, switches and all kinds of buses [26].

3. SOFTWARE TOOLS

3.1 Critical Path Analysis

High-speed high-performance digital systems require a careful timing analysis of all critical signal paths to establish the maximum usable system clock frequency in sequential circuit designs, or to establish that critical path delays are compatible with the timing specification in combinational circuit designs. Very often the design will still not work even though the circuit passes the functional test. This is mainly because the designers do not consider the worst-case component delays and the signal wiring propagation delays [16]. This is important in UPB since there are programmable switches between the FPGAs, it is obvious that the propagation delays of a signal which leaves a FPGA, then proceeds through the switches and re-enter into another FPGA is longer than that propagates within a FPGA. Hence, a critical path analysis is performed before partitioning the circuit across the FPGAs. All the components in the critical path found will be assigned to a single FPGA.

For a realistic estimation of the operation speed of a system, the propagation delays of each components in each signal path plus the interconnection propagation delays must be determined. A data file (delay.dat) which specifies all the primitive gates delays and the path delay can be modified by the user. The format and the content of the delay.dat file is shown in Appendix A. The default propagation delays of all primitive gates and paths are one. i.e. unit-delay model. For combinational circuit, the critical path is the longest delay of a path originates from the UPB input to the UPB output. On the other hand, the operation speed of a sequential circuit is determined by the paths between DFFs and system input and output. So the critical path is the longest delay of one path in the following: from the UPB input to the input of a D-flip flop (DFF, the primitive clocked device in FPGA), or from the input of a DFF to the input of another DFF, or from the input of a DFF to the UPB output in the sequential circuit. An option that the critical path analysis will treat all the components in a

sequential circuit as combinational elements is provided for analysis. In other words, if this option is chosen, the program will find the longest path from the UPB input to the UPB output disregarding the DFFs.

3.1.1 Algorithm of critical path analysis

The algorithm of the critical path analysis is very simple. It just iteratedly searches all paths in the circuit from the system input to the system output in the combinational circuit and from the system input to the input of a DFF, or from the input of a DFF to the input of another DFF, or from the input of one DFF to the system output in the sequential circuit.

First of all, a GATE variable array which records all the primitive gates is constructed. For each gate, a linked netlist of its succeeding gates, and the information of the type of the gate such as AND, DFF, etc., and that whether it is an input gate or output gate are recorded. Input gate is the gate whose signals incident on it is system input wile the output gate is the gate whose signals incident on it is system output. At any time, two paths are stored in memory. One stores the longest path currently found so far while the other stores the current path it is searching. At the same time, two critical path delays correspond to the two paths are also noted. It continuously compares the current path with the stored longest path. If the current path is longer than the longest path stored, the longest path recorded will store the current path. Otherwise, the current path will be deleted.

3.1.2 Computation time

It seems that the algorithm is quite simple and straightforward. However, if it is not carefully done, the computation time will be terrible. One naive approach is deleting the whole path after comparing with the longest path stored. This is not necessary since within a path, there may be a gate which have two or more succeeding gates. In such case, there exists another

path in the circuit in which the path preceding that gate is the same. From figure 17, gates 1, 3 & 4 are input gates and gate 7 is output gate. Gates 2 and 4 have two succeeding gates. We can see that path 2 preceding gate 2 is the same as path 1 preceding gate 2. Similarly, path 3

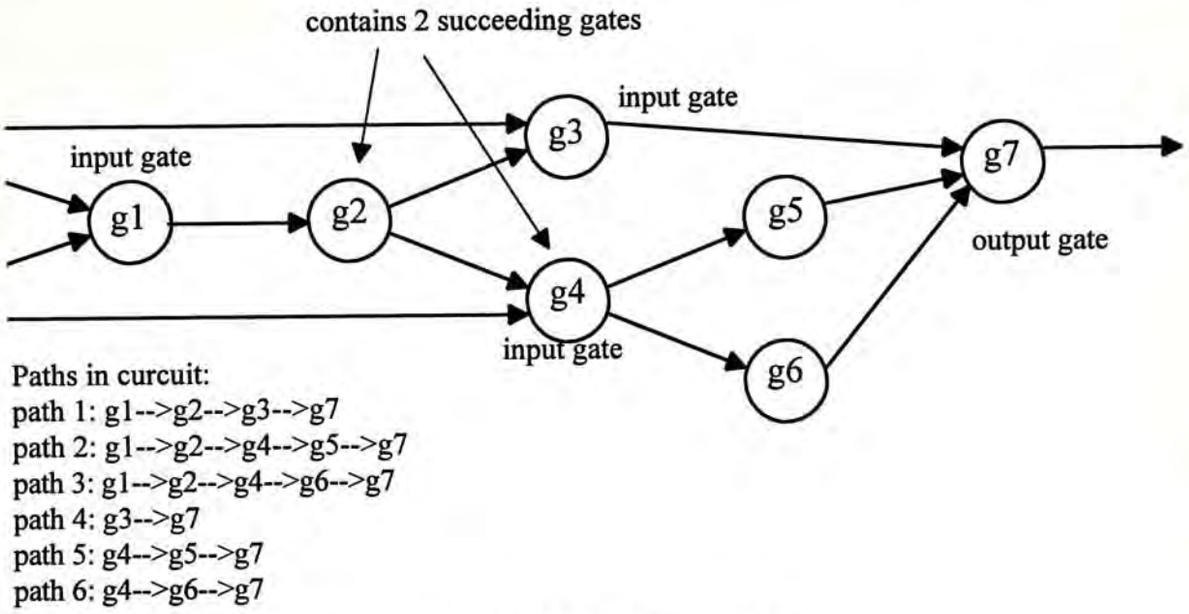


Fig. 17 All paths in a circuit.

preceding gate 4 is the same as path 2 preceding gate 4. Therefore, a stack (not queue) is used to store the gates with two or more succeeding gates. Every time after comparing with the longest path stored, we only delete the path proceeding the gate in the stack.

Another time-consuming task is to check for feedback path. A simple approach is to check if the current gate found already exists in a path. This wastes a lot of time for searching from the start of a path to the current gate if the circuit is very large because the path obtained will be very long. The problem can be solved by adding a pointer in each entry in the GATE variable array pointed to the gate in the path. Every time it adds a gate in the path, we only check if the pointer of the gate is NULL. If so, it does not exist on the path. Otherwise, the gate does exist. i.e. it is the feedback path. The existed gate found is disregarded and not added in the path. It then continues to find the next succeeding gates in the current path.

3.2 Circuit Partitioning

In order to automate the design process, a partitioning tool is provided to partition the netlist into multiple sets of netlist each of which fits into a FPGA. There are two kinds of partitioning techniques: pre-mapping partitioning and post-mapping partitioning. Pre-mapping partitioning

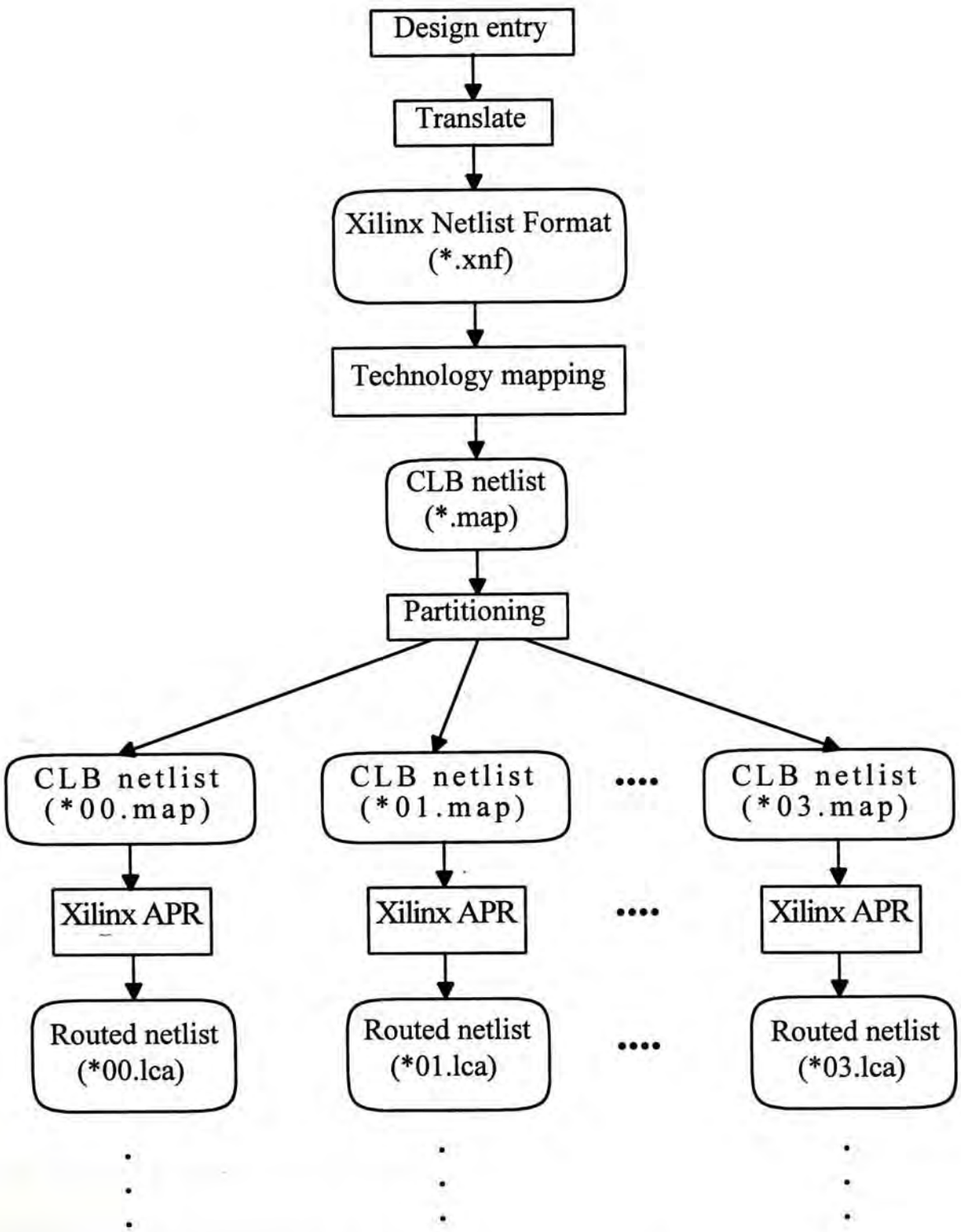


Fig. 18. Post-mapping partitioning.

means the circuit partitioning is performed before the technology mapping process whereas in the post-mapping partitioning, partitioning is done after the technology mapping process. Clearly, an advantage of post-mapping partitioning is that it occurs at the actual implementation of the design process. The physical information specific to the FPGA is available [17]. So the partitioner here adopts post-mapping partitioning. It accepts the netlist file with extension *.map (mapping) and produces the partitioned netlists that can be fitted into individual FPGA for Xilinx Automatic Place and Route (APR) (see figure 18).

3.2.1 Partitioning algorithm

Logic partitioning is NP-complete. In practice, the size of the partitioning problems makes it impossible to perform an exhaustive search to find an optimal partition. Hence, algorithms based on heuristic rationales that give good results in a reasonable amount of time have been developed [18]-[20].

3.2.1.1 Kernighan and Lin

One of the well known heuristic methods of circuit partitioning was developed by Kernighan and Lin (K&L) [21]. It became the basis for most of the iterative improvement partitioning algorithms generally used. One example is the linear-time heuristic partitioning algorithm developed by Fiduccia and Mattheyses (F&M) [22] (see 3.2.1.2). Another example is the level-gain model developed by Balakrishnan Krishnamurthy [23]. This algorithm deals with the problem of dividing a set of cells into two blocks A and B (bi-partitioning) so that the cut size between two blocks is minimized. The cut size is the number of nets connected simultaneously to cells in both blocks of the partition. It starts with an initial partition of cells into A and B, and improves it by choosing one cell from each block and swaps them. The cell pair chosen is the pair that gives the most improvement in cut size. The algorithm contains a number of passes. In each pass, potential gains achieved from all possible cell pairs if they are

swapped are calculated. After choosing the cell pair with the best potential gain, they are swapped and locked in place. The cut size of the current partition is recorded and the potential gains are updated. The same process continues among the unlocked cells until all the cells are locked. At the end of the pass, the cells in the two partitioned blocks and the cut size will be exactly the same as at the beginning since all the cells in block A are moved to block B and vice versa. The algorithm looks back at the sequence of gains that have been recorded, and undoes swaps that are made after the greatest improvement has been seen. The same process is repeated in subsequent passes, and the whole algorithm stops until the gain cannot be improved. It was shown that the running time per pass is $O(c^2 \log c)$, where c is the total number of cells in the network, and that the process usually converges after a few passes.

3.2.1.2 Fiduccia and Mattheyses

With the proper model from reference [24], Fiduccia and Mattheyses modified the algorithm of K&L such that the worst case running time per pass is O(clogc) i.e. grows linearly with the size of the network, and proved that it also converges in several passes. Instead of swapping the best cell pair from each block, the basic approach is to move one cell at a time, from one block to the other, in an attempt to reduce the number of nets which have cells in both blocks. Each cell is assigned a gain value defined as the number of nets by which the cut size would decrease if it was moved from its current block to its complementary blocks. This MOVE operation consists of selecting the best cell (highest gain) to be moved, moving it and then adjusting the gains of its free (unlocked) neighbouring cells. In a pass, all the cells are moved and locked, and then some later moves are undone until the greatest improvement is obtained as in K&L algorithm. Since it moves one cell to the other block at a time, there is a possibility that all the cells in one block may be moved to the other. A balancing criterion is established to prevent the MOVE during a pass.

If this MOVE operation was implemented naively, it would require $O(c^2)$ operation. Two tricks in implementing this operation are proposed that will reduce the total work to perform one pass to be linear. The first one is the introduction of a data structure: BUCKET variable

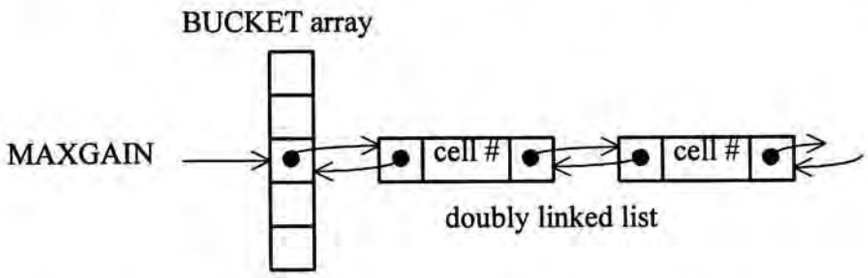


Fig. 19. BUCKET array structure in K & L algorithm.

array, as shown in Figure 19, whose k^{th} entry contains a doubly-linked list of free cells with gains currently equal to k. Two such arrays are needed, one for block A and one for block B. For each BUCKET array, a MAXGAIN index is maintained which is used to keep track of the bucket having a cell of highest gain. This data structure quickly returns a cell of highest gain and allows recomputed cell gains to be re-entered into the structure.

The second one is to update the gains of its free neighbouring cells by an appropriate sequence of simple gain increments or decrements of the current gains rather than recomputing the gains of all neighbouring cells each time the best cell moves. Define the distribution of a net n, relative to the blocks A and B, as A(n) and B(n) which represents the number of cells the net n has in blocks A and B respectively. The net n is said to be cut if it has at least one cell in each block and uncut if otherwise. This refers to the cut-state of the net. The net is critical if it is connected to a cell which if moved will change the net's cut-state. It is easy to see that n is critical iff: either A(n) or B(n) is equal to 0 or 1. More importantly, a net which is not critical either before or after a move cannot possibly influence the gains of any of its cells. Therefore, only those nets, connected to the best cell, that are critical before or after the move have to be considered. Let F(n) (From) be the distribution of a net n relative to the block in which the best cell is resided, and T(n) (To) be the distribution of the net n relative to the complimentary block. We have to check the critical nets before and after the move. Besides, the net

distribution of the net to which the best cell is connected has changed to reflect the move. Before the move, the critical nets in To-side i.e. T(n) = 0 and T(n) = 1 are checked since the best cell is going to be moved to it. On the other hand, after the move, critical nets in From-side i.e. F(n) = 0 and F(n) = 1 are checked as the best cell have already been moved from it. During the move, both F(n) and T(n) are updated. In the following, all cases before and after the move are illustrated. Assume C_1 is the best cell to be moved.

Case 1: Before the move, T(n) = 0, F(n) ≥ 2 (Figure 20).
Result: gain of all free cells on net n incremented.

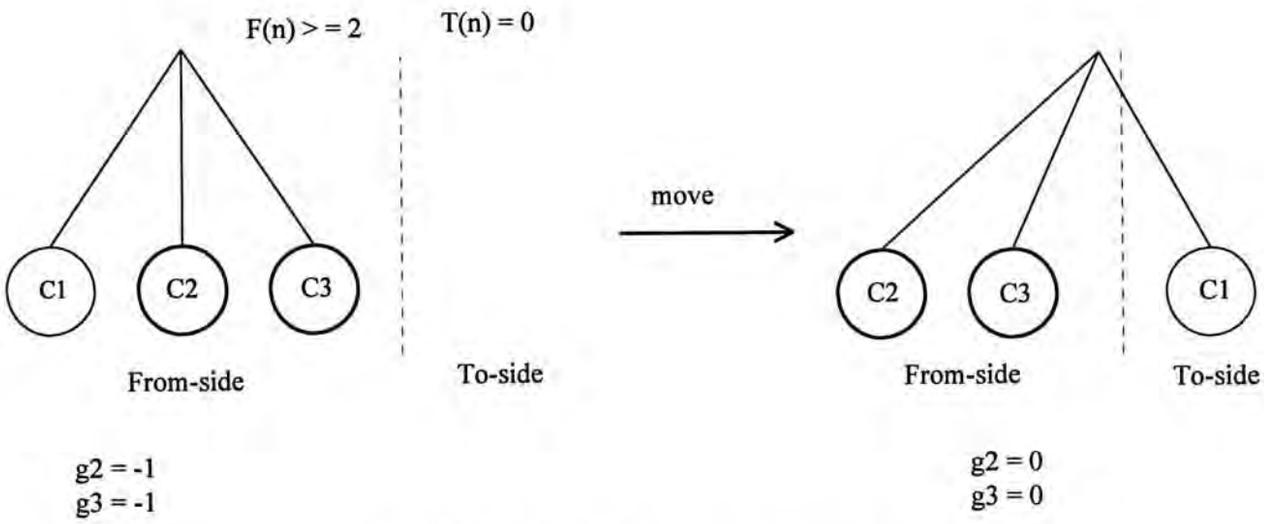


Fig. 20. Check critical nets before the move: T(n) = 0.

Case 2: Before the move, T(n) = 1, F(n) ≥ 2 (Figure 21).
Result: gain of the only cell in To-side decremented, if it is free.

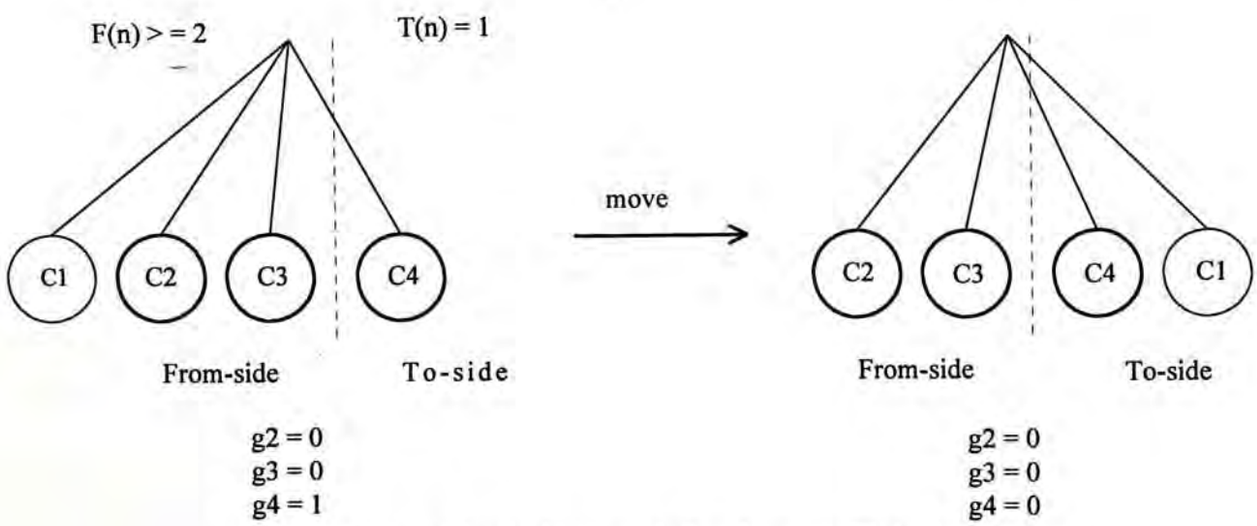


Fig. 21. Check critical nets before the move: T(n) = 1.

Case 3: After the move, F(n) = 0, T(n) ≥ 2 (Figure 22).
Result: gains of all free cells on net n decremented.

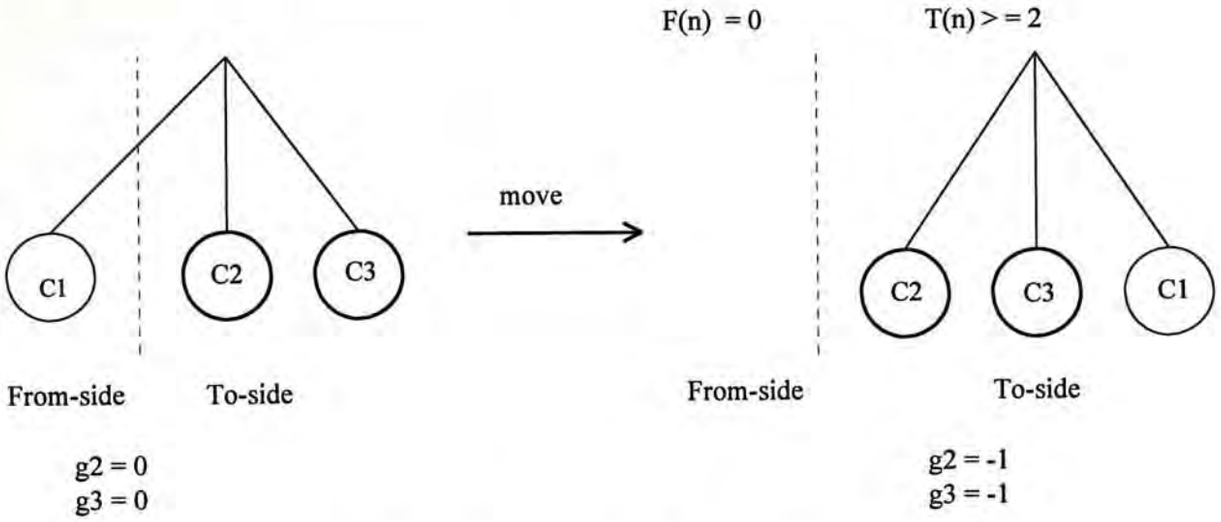


Fig. 22. Check critical nets after the move: F(n) = 0.

Case 4: After the move, F(n) = 1, T(n) ≥ 2 (Figure 23).
Result: gain of the only cell in From-side incremented, if it is free.

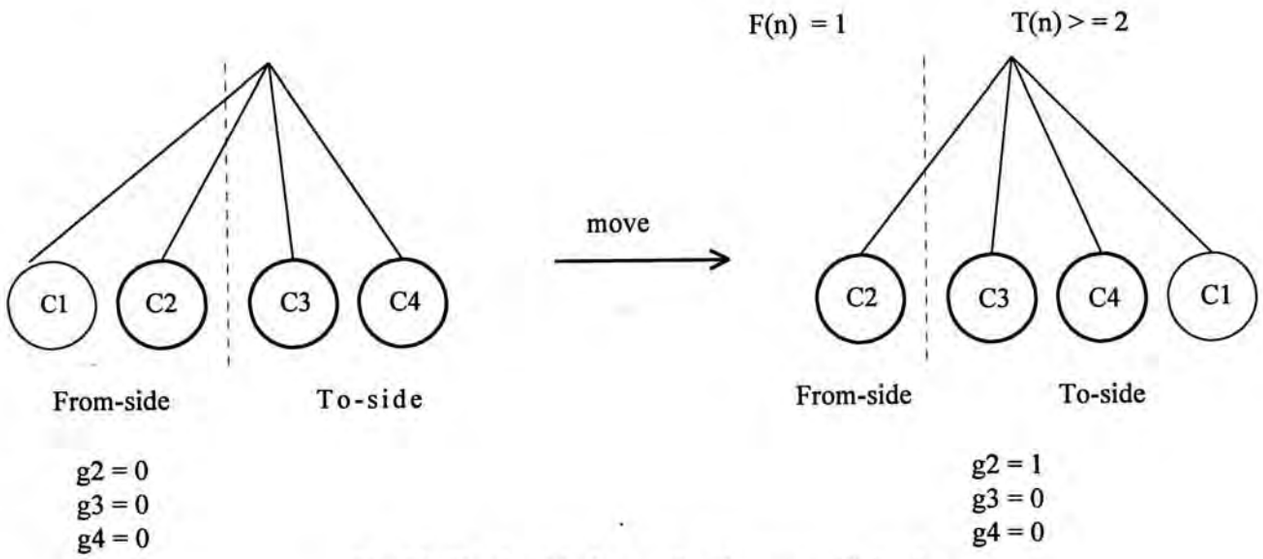


Fig. 23. Check critical nets after the move: F(n) = 1.

Combining all the cases together, updating the gains of the free neighbouring cells of the best cells requires only an appropriate sequence of gain increment or decrement of the current gains. The process can be written as follows:

```
FOR each net n on the best cell DO
    IF T(n) = 0 THEN                          /* check critical nets before move */
        increment gains of all free cells on n;
    ELSE IF T(n) = 1 THEN
        decrement gain of the only T cell on n, if it is free;
    END IF
    decrement F(n);                           /* change net distribution to reflect */
    increment T(n);                           /* the move */
    IF F(n) = 0 THEN                          /* check critical nets after move */
        decrement gains of all free cells on n;
    ELSE IF F(n) =1 THEN
        increment gain of the only F cell on n, if it is free;
    END IF
END FOR
```

3.2.1.3 UPB Partitioner

As the computation time of F&M is much faster than K&L for large circuits, F&M algorithm is used to partition the netlist in the UPB. However, both K&L and F&M algorithms deal with bi-partition, there are totally four FPGAs (blocks) in the UPB, it is necessary to adapt the bi-partitioning F&M algorithm to the multi-partitioning algorithm. There are two ways to adapt the bi-partitioning algorithm to multi-partitioning. One way successively chooses pairs of blocks and applies the bi-partitioning to these pairs until all pairs are chosen. However, eliminating a net across a given pair does not necessarily remove it in the multiple block partition. Another way hierarchically uses bi-partitioning algorithm. It initially uses bi-partitioning to partition the whole network into two blocks, and then partition each blocks into two more blocks and so on. Obviously, the first level partitioning will limit the scope of improvement in the next level partitioning. Besides, the first level partitioning will try to minimize the number of connections between the first two blocks. This tends to maximize the connections inside these blocks which will make it harder to get a good result in the next level partitioning. Both ways are obviously not satisfactory. A better approach is to expand the scope at each step in the algorithm rather than applying the whole bi-partitioning algorithm to block pairs one by one. In this approach, all possible moves to any of the other blocks are considered and the best move is chosen and recorded [25].

It is different from bi-partitioning that in the multi-partitioning, the cells can be moved to all other blocks rather than only the complimentary block. So there are several gains associated with each cell. Define the gain $g_{ft}(C)$ of a cell C as the number of nets by which the cut size would decrease if it was moved from f-block to t-block. In the UPB, each cell has 3 gain values since there are totally 4 FPGAs in the UPB and each cell can move from its current FPGA to the other 3 FPGAs. The net n has k cuts if it has the cells in k+1 blocks and the cut size of the partition in the UPB is defined as the total number of cuts it has.

The tricks proposed by F&M were also adopted and modified to make running time per pass is linear. First of all, the BUCKET variable arrays are also used in the UPB partitioner. However, there are totally 4 x 3 = 12 such arrays kept. For each block, 3 arrays are kept to represent the possible movement to the other 3 blocks. A MAXGAIN index is also maintained in each array to point to the cell with the maximum gain. In order to ease the process of choosing the best cell among those cells pointed by the MAXGAIN pointers, a HEAP variable array is used to keep a sorted list of the MAXGAIN pointers and the corresponding move directions. The pointer with the highest gain value can be found at the bottom of the HEAP and will be chosen and moved.

Secondly, updating the gains of the free neighbouring cells is also modified such that it can be done by an appropriate sequence of gain increments or decrements of the current gains when the best cell is moved. Apart from the F(n) and T(n), let O(n) (Other) be the distribution of a net n relative to the block which the best cell are not moved from or moved to. The following 8 cases illustrates how the process of updating gains can be implemented.

Case 1: Before the move, T(n) = 0, O(n) = 0, F(n)≥ 2 (Figure 24).
Result: gains g_{ft} of all free cells on n incremented.

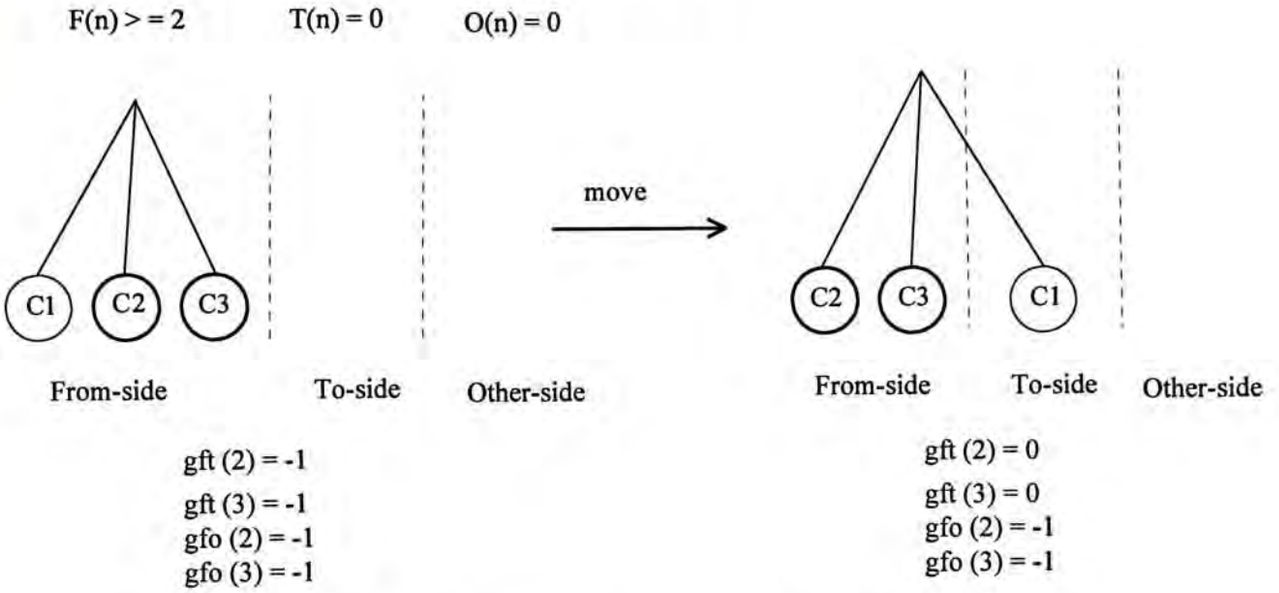


Fig. 24. Check critical nets before the move: T(n) = 0, O(n) = 0.

Case 2: Before the move, T(n) = 0, O(n) ≥ 1, F(n) ≥ 2 (Figure 25).
Results: gains g_{ft} of all free cells on n incremented and
gains g_{ot} of all free cells on n incremented.

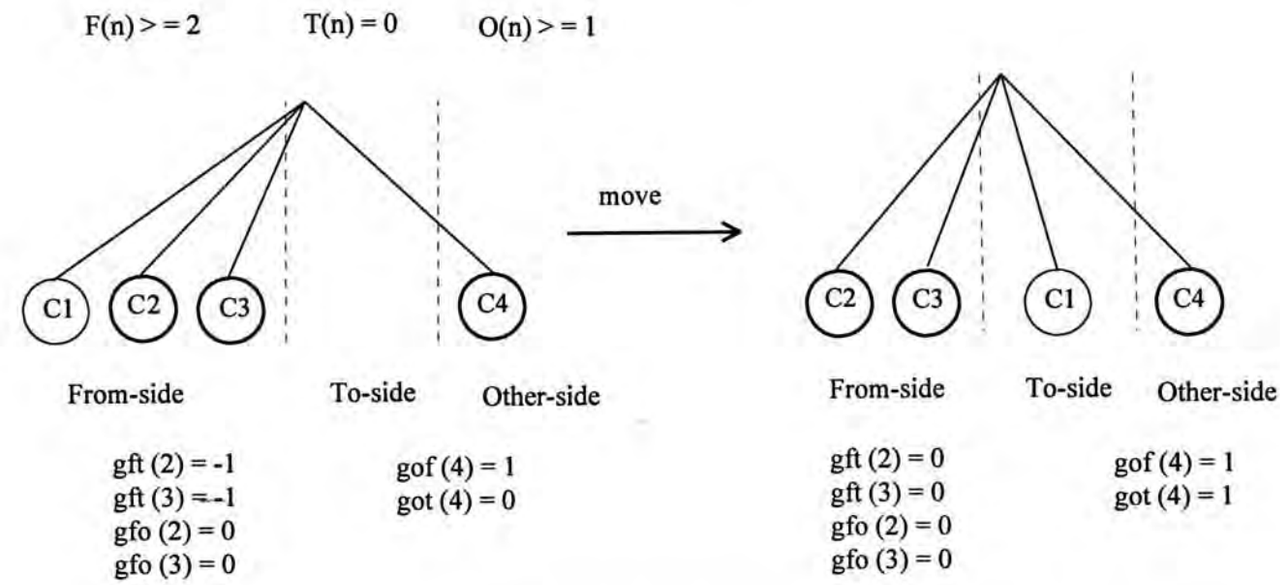


Fig. 25. Check critical nets before the move: T(n) = 0, O(n) > = 1.

Case 3: Before the move, T(n) = 1, O(n) = 0, F(n) ≥ 2 (Figure 26).
Results: gain g_{tf} of the only cell in To-side decremented, if it is free and gain g_{to} of the only cell in To-side decremented, if it is free.

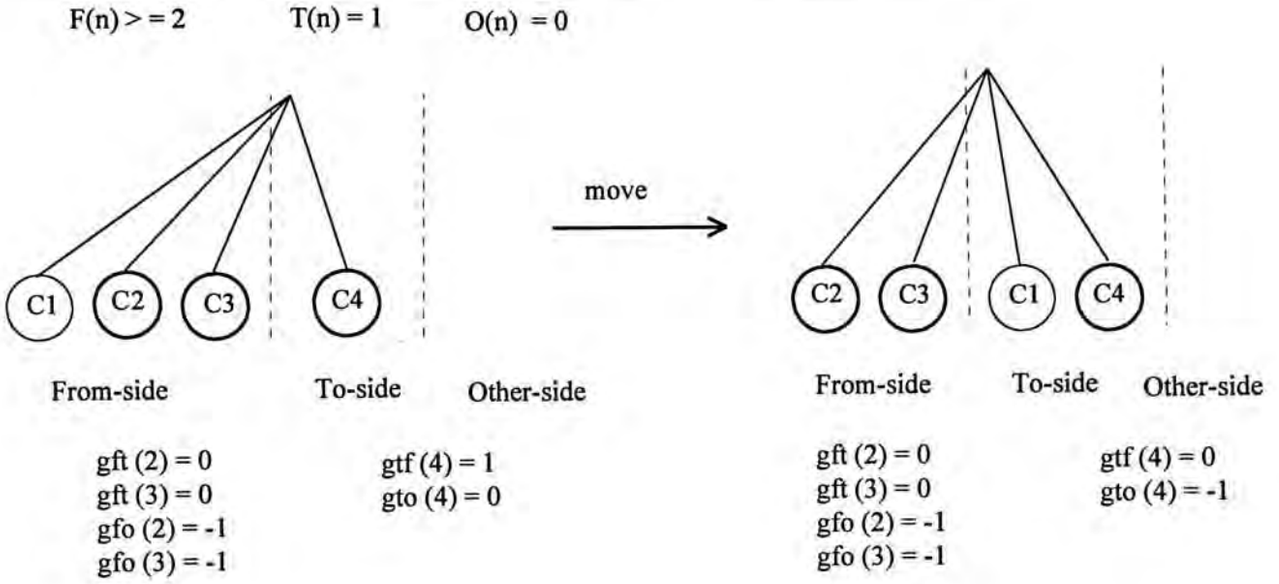


Fig. 26. Check critical nets before the move: T(n) = 1, O(n) = 0.

Case 4: Before the move, T(n) = 1, O(n) ≥ 1, F(n) ≥ 2 (Figure 27).
Results: gain g_{tf} of the only cell in To-side decremented, if it is free and gain g_{to} of the only cell in To-side decremented, if it is free.

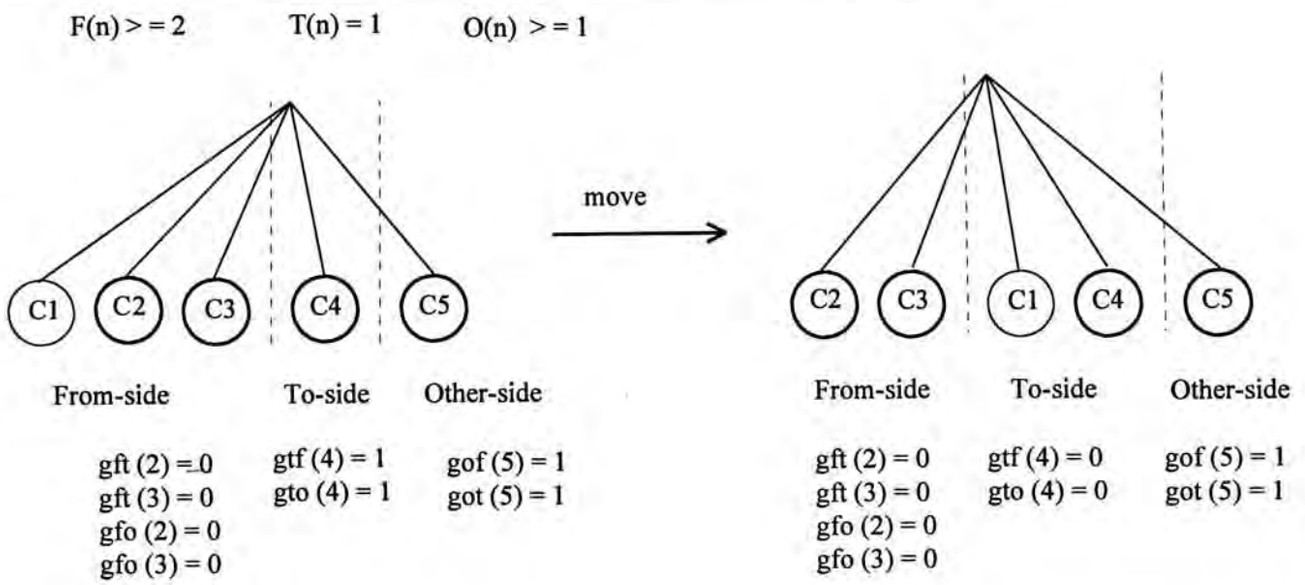


Fig. 27. Check critical nets before the move: T(n) = 1, O(n) > = 1.

Case 5: After the move, F(n) = 0, O(n) = 0, T(n) ≥ 2 (Figure 28).
Result: gains g_{tf} of all free cells on n decremented.

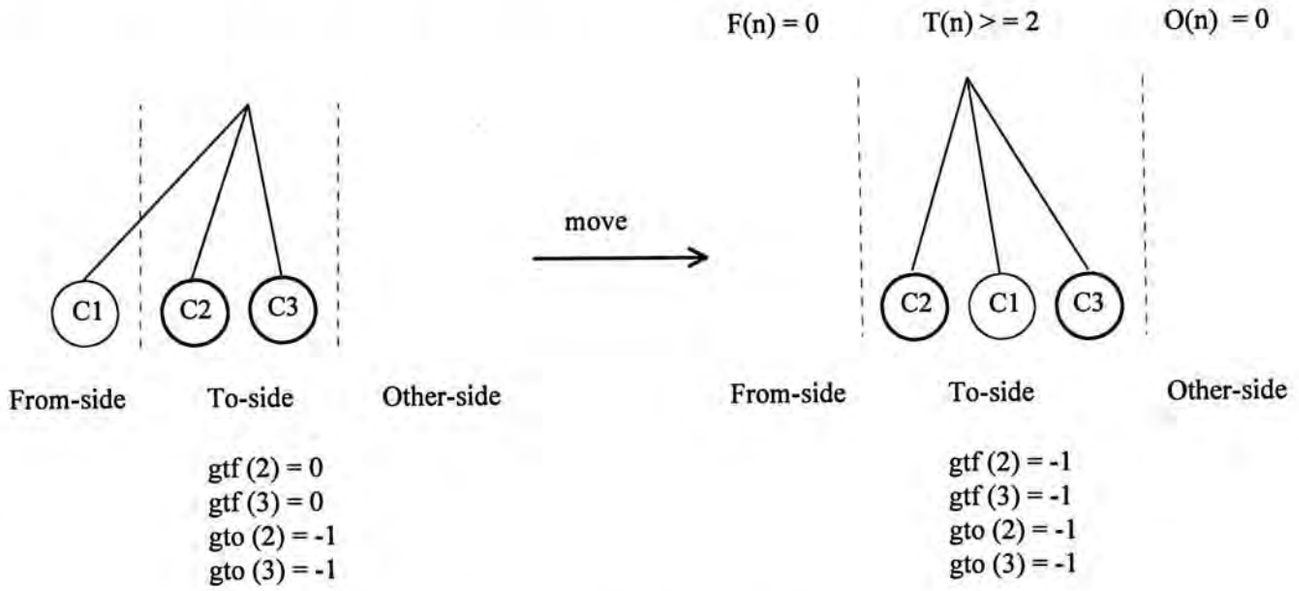


Fig. 28. Check critical nets after the move: F(n) = 0, O(n) = 0.

Case 6: After the move, F(n) = 0, O(n) ≥ 1, T(n) ≥ 2 (Figure 29).
Results: gains g_{tf} of all free cells on n decremented and
gains g_{of} of all free cells on n decremented.

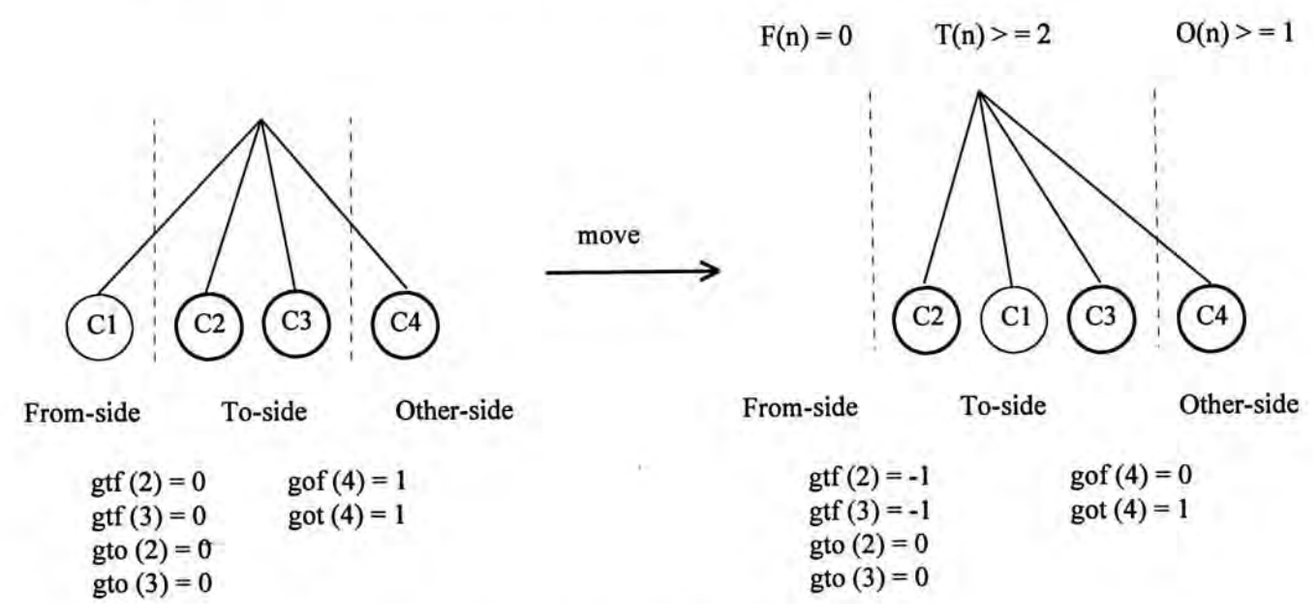


Fig. 29. Check critical nets after the move: F(n) = 0, O(n) > = 1.

Case 7: After the move, F(n) = 1, O(n) = 0, T(n) ≥ 2 (Figure 30).
Results: gain g_{ft} of the only cell in From-side incremented, if it is free and gain g_{fo} of the only cell in From-side incremented, if it is free.

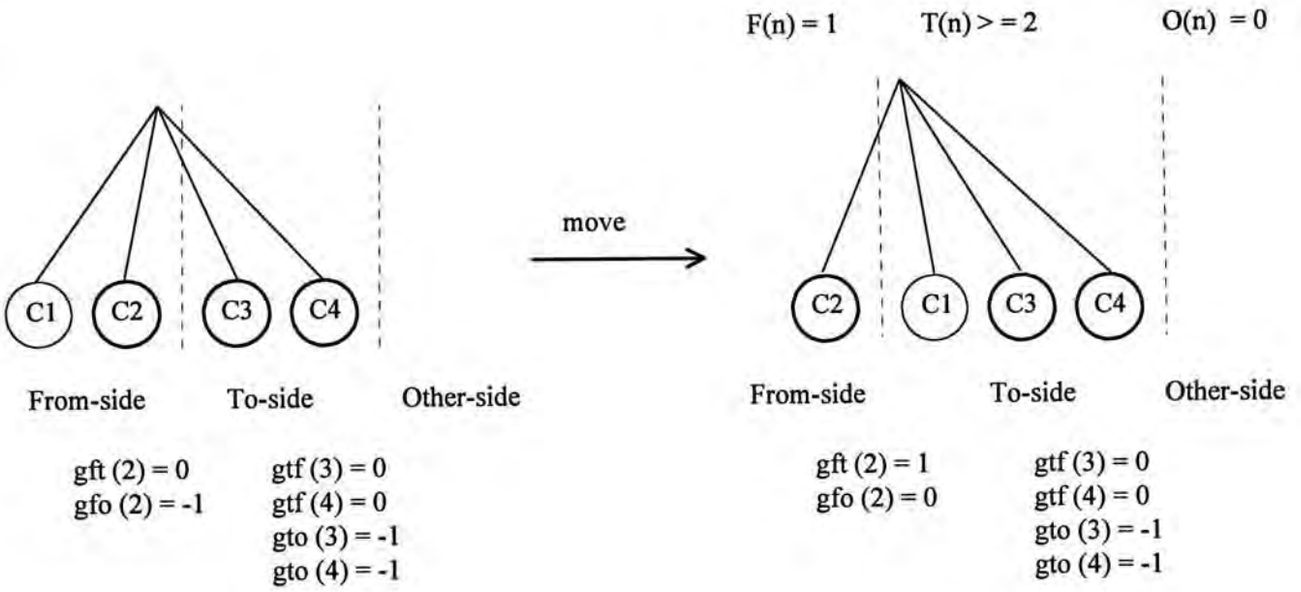


Fig. 30. Check critical nets after the move: F(n) = 1, O(n) = 0 .

Case 8: After the move, F(n) = 1, O(n) ≥ 1, T(n) ≥ 2 (Figure 31).
Results: gain g_{ft} of the only cell in From-side incremented, if it is free and gain g_{fo} of the only cell in From-side incremented, if it is free.

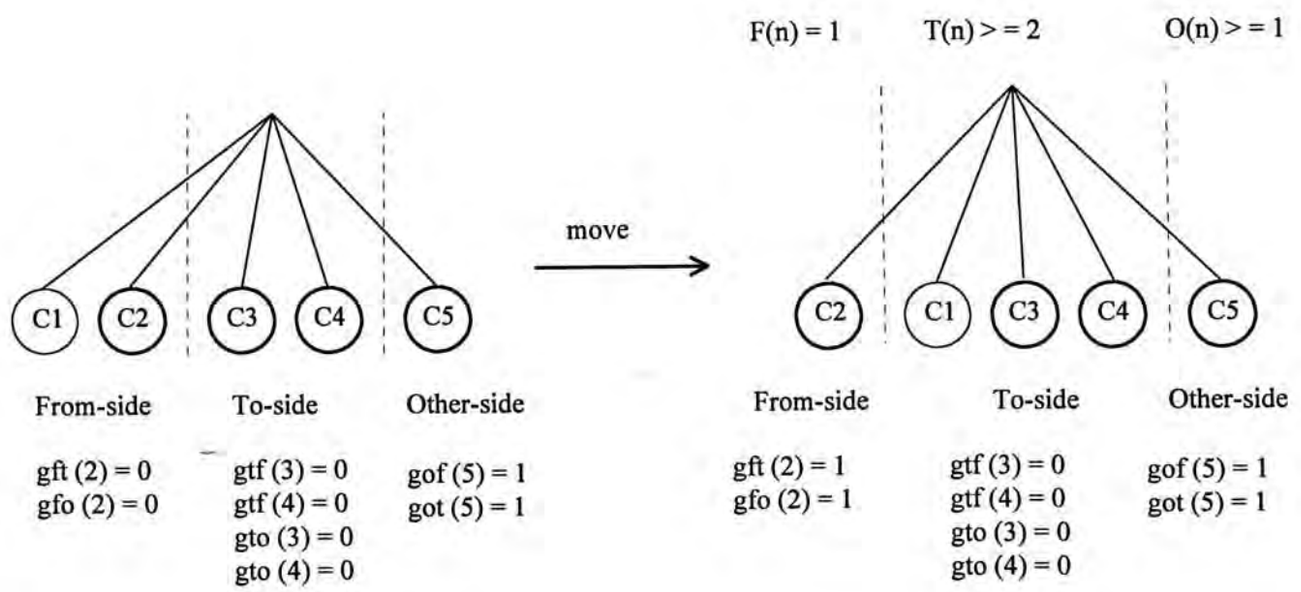


Fig. 31. Check critical nets after the move: F(n) = 1, O(n) > = 1.

Combining all the cases above, the following codes can be developed to update the gates of the neighbouring cells when the best cell moves:

```
FOR each net n on the best cell DO
    IF T(n) = 0 THEN                                    /* check critical nets before the */
        increment gains g_ft of all free cells on n;    /* move */
        increment gains g_ot of all free cells on n, if any;
    ELSE IF T(n) = 1 THEN
        decrement gain g_tf of the only T cell, if it is free;
        decrement gain g_to of the only T cell, if it is free;
    END IF
    decrement F(n);                                     /* change net distribution to */
    increment T(n);                                     /* reflect the move */
    IF F(n) = 0 THEN                                    /* check critical nets after the */
        decrement gains g_tf of all free cells on n;    /* move */
        decrement gains g_of of all free cells on n, if any;
    ELSE IF F(n) = 1 THEN
        increment gain g_ft of the only cell in From-side, if it is free;
        increment gain g_fo of the only cell in From-side, if it is free;
    END IF
END FOR
```

Before generating the initial partition, the CLBs which contain gates on the critical path are assigned to an FPGA so that the number of interconnections between critical path and the fixed neighbouring cells. The fixed neighbouring cells come from the fact that some logic cells that drive fixed pins such as the address lines for the RAM chips must be assigned to the FPGA connected to those fixed pins. Also, the algorithm allows users to assign any signals to a specific FPGA for their own purposes. For example, in figure 32, there are three

interconnections between the critical path and the fixed neighbouring cells assigned to FPGA 1, six interconnections between the critical path and the fixed neighbouring cells assigned to FPGA 2, and two interconnections between the critical path and the fixed neighbouring cells

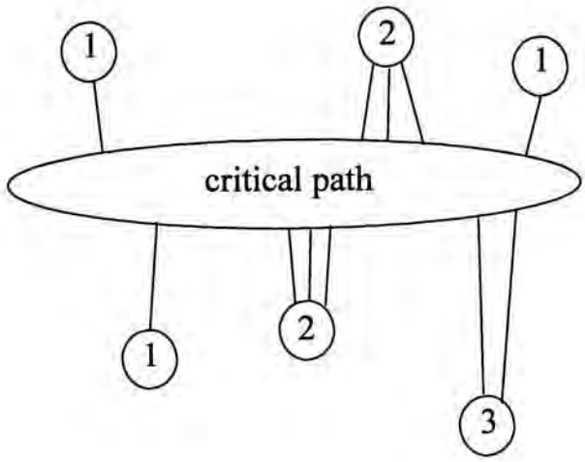


Fig. 32. Assigning cells on critical path to FPGA.

assigned to FPGA 3. So the cells on the critical path are assigned to FPGA 2. On the other hand, whenever no fixed neighbour cells exists, the cells on the critical path are arbitrarily assigned to an FPGA. Once the critical path and fixed cells are taken care of, cells remained are then assigned to a FPGA which currently contains the most number of cells. If the design is too large to fit into the FPGA, the cells are assigned to the adjacent FPGA and the next adjacent FPGA and so on until all the cells are assigned.

3.2.2 Effects of partitioning

As discussed above, the delays of a signal routed off one FPGA and back onto another will increase. It is better if the signal is mapped to the hardwired bus. In contrast, if it is mapped to the programmable bus, extra delay that the programmable switches caused may be very serious. That's why the critical path analysis is needed before partitioning the circuit. Of course, assigning all gates on the critical path into a single FPGA may have a disadvantage that the cut size between FPGAs will be higher since it forces some CLBs into a particular FPGA. Luckily, this will affect the non-critical-part of the circuit.

The fixed gate count of the FPGA always limit the number of the CLBs assigned to each FPGA. Theoretically, each XC3042 FPGA can handle 4, 200 gates, totally 16, 800 gates in the UPB. However, locking the IOs by the partitioner can severely restrict the ability of the Xilinx APR to place and route the design. There are only at most 50% of CLBs, around 10,000 gates, usable to obtain 100% routability of each FPGA. The CLB utilization rate of the UPB is defined as the maximum ratio of CLBs in each FPGAs can be used. Hence the higher the CLB utilization rate of FPGAs was set by the partitioner, the lower the routability of the FPGAs will be obtained and vice versa.

Two ways of partitioning methods can be chosen by the user. The first one is the balanced partitioning. It partitions the circuit into any number (2-4) of FPGAs in a balanced way. It has the advantage that the logic after partitioning in each FPGA is smaller. So the Xilinx APR most probably can route the circuit completely. However, balanced partitioning increases the cut size between the FPGAs. This results in greater delay in signals and congestion in interchip routing. The second one is the non-balanced partitioning. It first partitions the circuit in such a way that the lowest number of FPGAs is used to accommodate the circuit. In this case, the cut size between FPGA will be smaller and hence it has the greater possibility to assign all IOs between FPGAs. The delay will be smaller as less signals will pass through the switches and the interchip buses. Yet, the logic assigned to each FPGA will increase. This may lead to the result that the APR is not able to route all the nets in FPGAs. Table 2 compares the effects of balanced partitioning and non-balanced partitioning.

	Balanced partitioning	Non-balanced partitioning
Cut Size	↑	↓
Assigning IO success rate	↓	↑
Delay	↑	↓
Logic in each FPGA	↑	↓
Routability	↓	↑

↑ : increase
↓ : decrease

Table 2. Balanced and Non-balanced partitioning comparison.

3.2.3 Partitioning parameters

As the partition influence several aspects of the design implementation, there are several parameters that can be varied by the user to tradeoff various effects. The user can choose not performing the critical path analysis for moderately low-speed applications. Or he can choose the option that the partition treats all primitive gates as combinational gates for analysis purpose. The user can vary the CLB utilization ratio of FPGA from 0.1 to 0.9. Besides, he can choose from balanced partitioning into any number (2-4) of FPGAs or non-balanced partitioning. With all these parameters varied at the same time, many partitioned sets of netlist can be obtained. From those sets, one may be able to get a partitioned netlist which has minimum cut size, minimum propagation delay between interchips, 100% IO assignment and 100% routability of all FPGAs used. The default options of the partitioner is no critical path analysis, CLB utilization rate = 0.5 and non-balanced partitioning. All the command line argument options are listed in Appendix B.

3.2.4 Pseudo-code of partitioning algorithm

1. *Network initialization;*
2. *All user-defined cells and the dedicated cells which drive fixed pins of FPGA are assigned to appropriate FPGAs and locked;*
3. *Find and return the critical path;*
4. *Obtain a starting partition after assigning and locking the cells on the critical path and neighbour fixed cells, if any;*
5. *REPEAT*
    1. *Initialize starting partition;*
    2. *Initialize BUCKET gain array;*
    3. *Initialize HEAP;*
    4. *WHILE(a move chosen preserving 2 constraints is possible && not all the cells locked) DO*
        1. *Gains of all possible moving of each cell from its home FPGA to all other FPGAs are computed and noted. The highest gain move is returned;*
        2. *The cell returned from above is locked and moved to the assigned FPGA;*
        3. *Update the gains of its neighbouring cells;*
        4. *Update the MAXGAIN pointer in BUCKET array;*
        5. *Update HEAP;*
        6. *The move and the current system gain are recorded in MOVE record;*
    5. *'Unlock' and 'unmove' the cells according to the MOVE record until the highest system gains obtained.*

UNTIL there is no improvement in system gain.

A move is possible only when it satisfies two constraints. First, a finite number of combinational logic blocks (CLBs) limits the amount of logic that can be assigned to a single

FPGAs. This number is limited by the CLB utilization rate (see later). Second is the limited number of interconnect (IOs) on each FPGA.

3.3 IO Assignments

Since all types of buses in the UPB are fixed to particular pins of the FPGAs, IO assignment is needed after partitioning. The pin numbers corresponds to all types of buses are specified in a data file iosource.dat which is first read in before IO assignments is performed. The formats and the content of the iosource.dat is shown in Appendix C. Two kinds of IO are needed to be assigned: the system IO and the interconnection nets between FPGAs. Actually for each system IO and the interconnection nets, there are several assignments possible. Some assignment use or even waste more IOs while others use less and save IOs. The assignment depends mainly on the number of FPGAs the nets connected to. The next subsections outlines all possible assignments and the number of IOs used and wasted for different kinds of nets.

3.3.1 Connect 4 FPGAs

3.3.1.1 Global bus: 4 IOs used

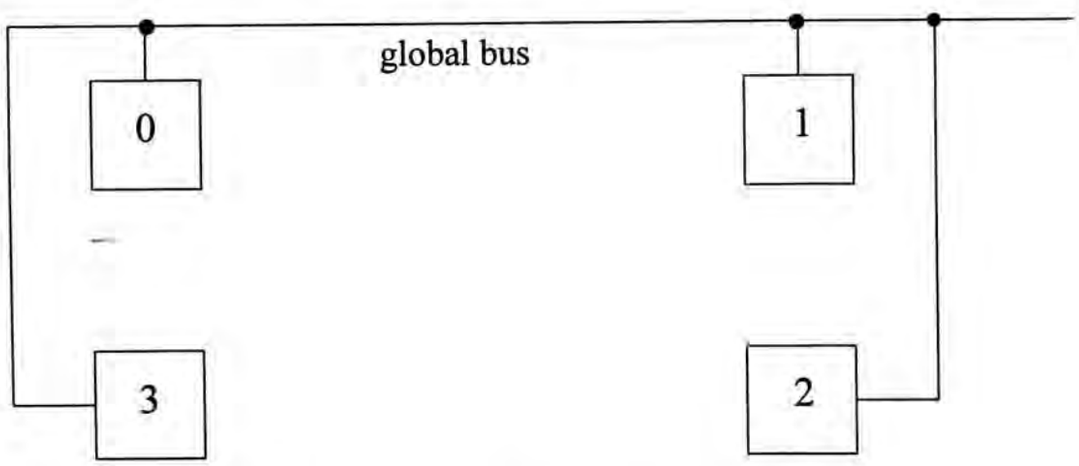


Fig. 33. Connect 4 FPGAs using global bus.

3.3.1.2 Local + programmable bus (+board IO): 5 (6) IOs used

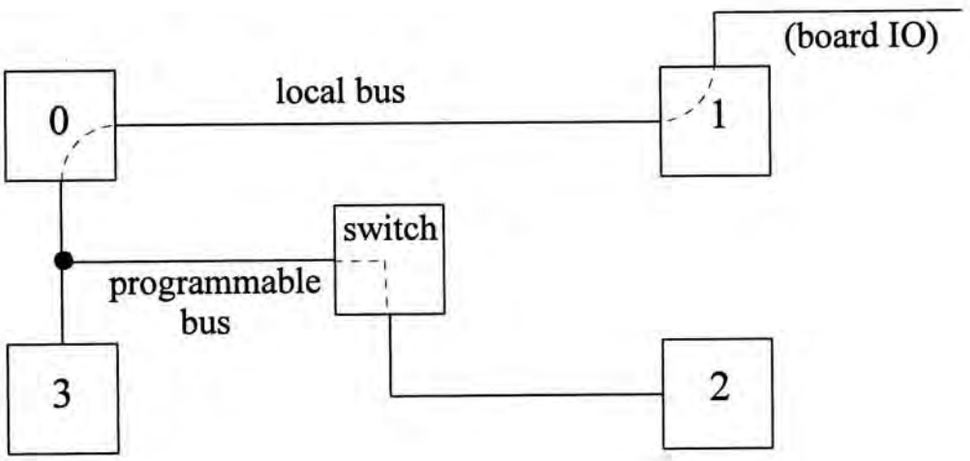


Fig. 34. Connect 4 FPGAs using local & programmable bus (& board IO).

3.3.1.3. Local bus (+board IO): 6 (7) IOs used

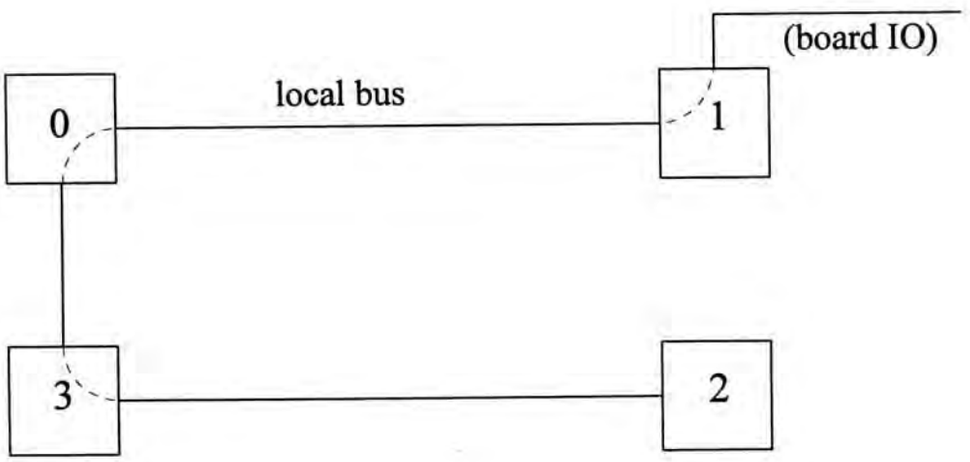


Fig. 35. Connect 4 FPGAs using local bus (& board IO).

3.3.2 Connect 3 FPGAs

3.3.2.1 Global bus: 4 IOs used, 1 IO wasted

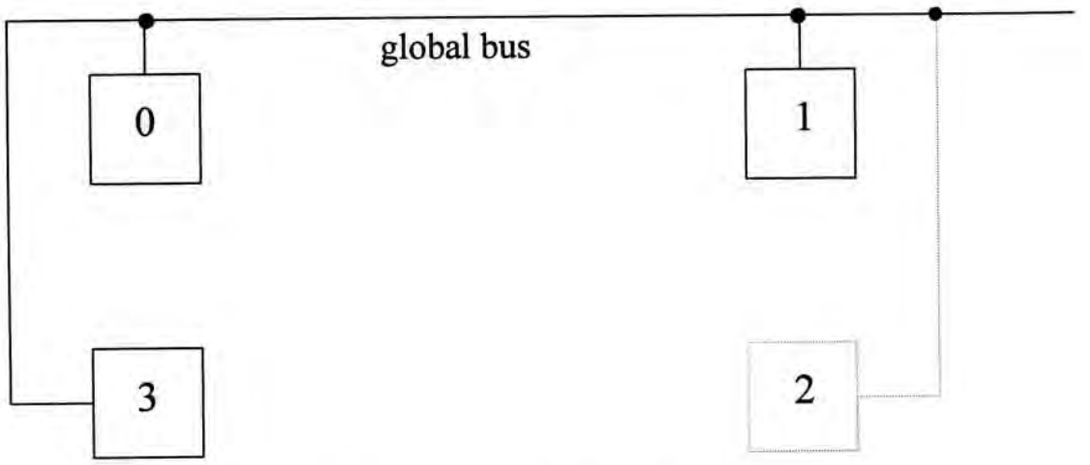


Fig. 36. Connect 3 FPGAs using global bus.

3.3.2.2 Programmable bus (+board IO) : 3 (4) IOs used

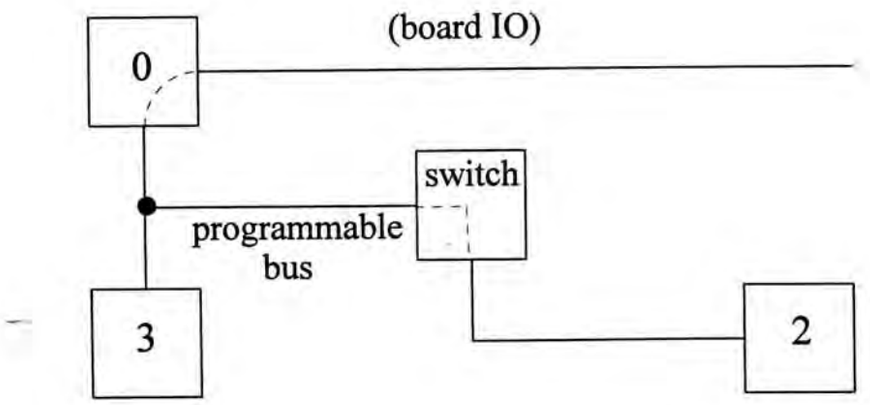


Fig. 37. Connect 3 FPGAs using programmable bus (& board IO).

3.3.2.3 Local bus (+board IO): 4 (5) IOs used

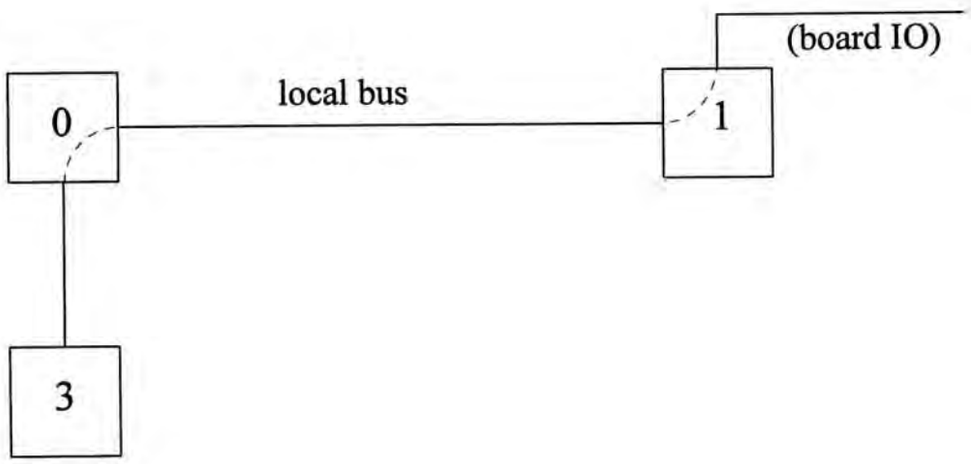


Fig. 38. Connect 3 FPGAs using local bus (& board IO).

3.3.2.4 Local + programmable bus (+board IO): 4 (5) IOs used

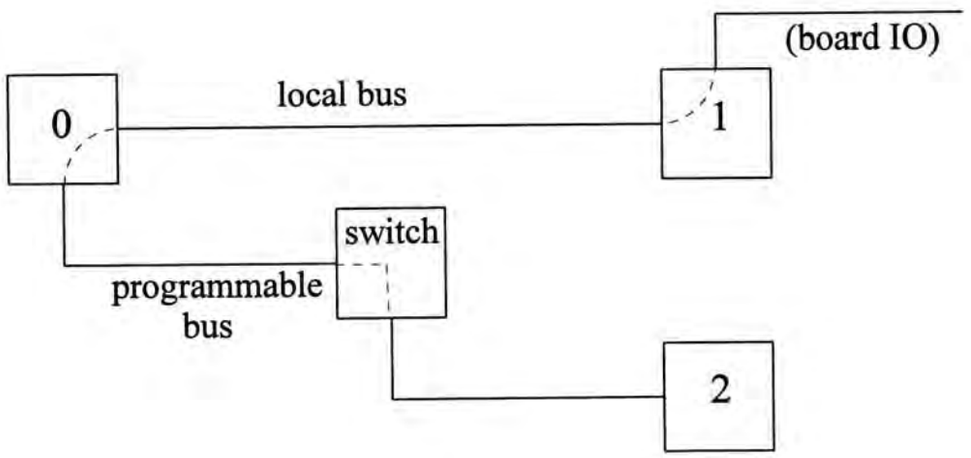


Fig. 39. Connect 3 FPGAs using local & programmable bus (& board IO).

3.3.3 Connect 2 FPGAs

3.3.3.1 Adjacent FPGAs

-- local bus (+board IO): 2 (3) IOs used.

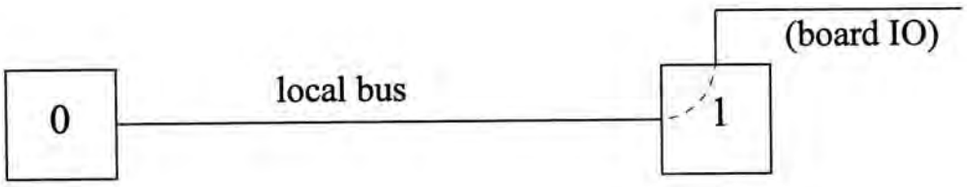


Fig. 40. Connect 2 adjacent FPGAs using local bus (& board IO).

3.3.3.2 Next adjacent FPGAs

-- programmable bus (+board IO): 2 (3) IOs used.

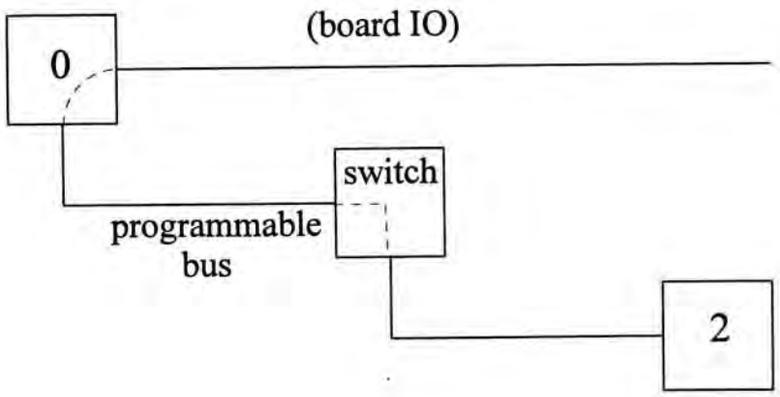


Fig. 41. Connect 2 next adjacent FPGAs using programmable bus (& board IO).

3.3.3.3 Adjacent FPGAs

-- Programmable bus (+board IO): 3 (4) IOs used, 1 IO wasted.

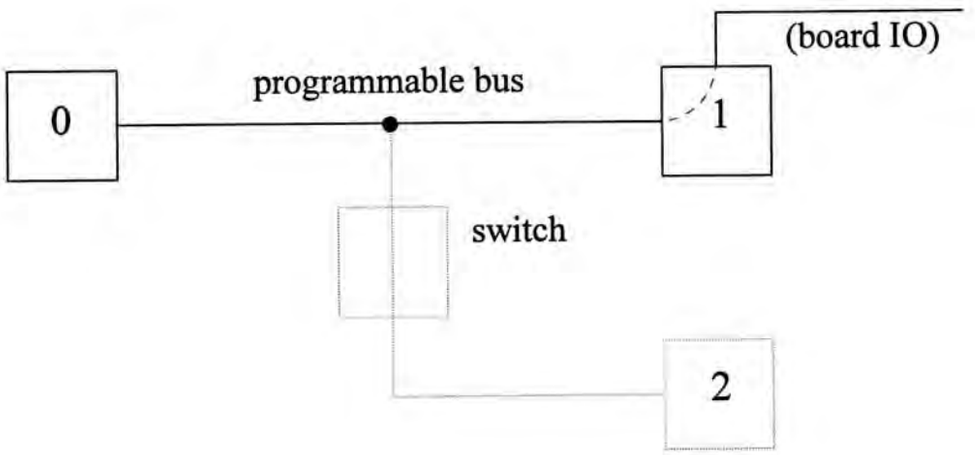


Fig. 42. Connect 2 adjacent FPGAs using programmable bus (& board IO).

3.3.3.4 Next adjacent FPGAs

-- programmable bus (+board IO): 3 (4) IOs used, 1 IO wasted.

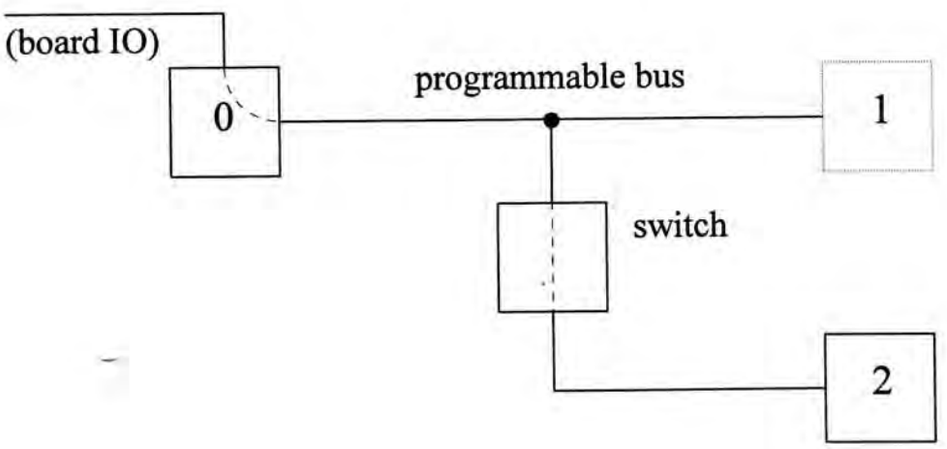


Fig. 43. Connect 2 next adjacent FPGAs using programmable bus (& board IO).

3.3.3.5 Global bus: 4 IOs used, 2 IOs wasted

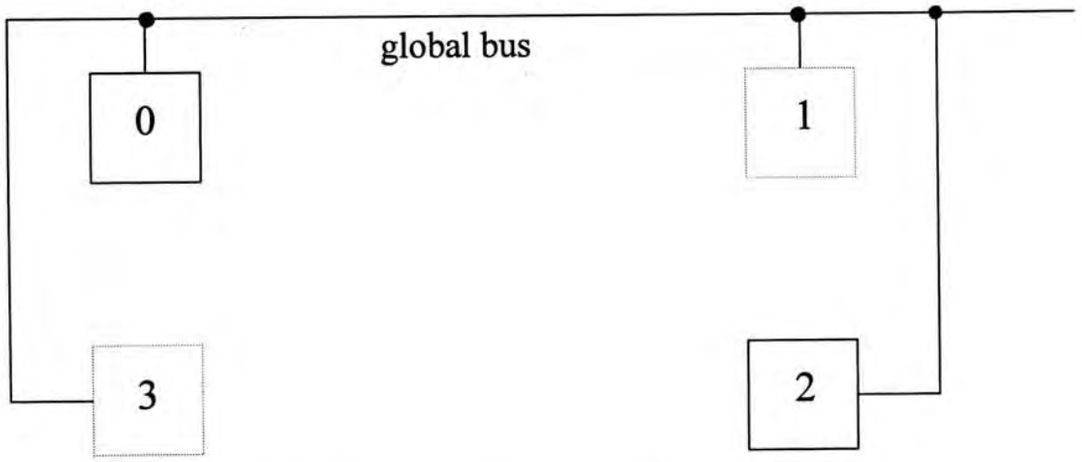


Fig. 44. Connect 2 FPGAs using global bus.

3.3.4 System IO (connect 1 FPGA)

3.3.4.1 Board IO: 1 IO used

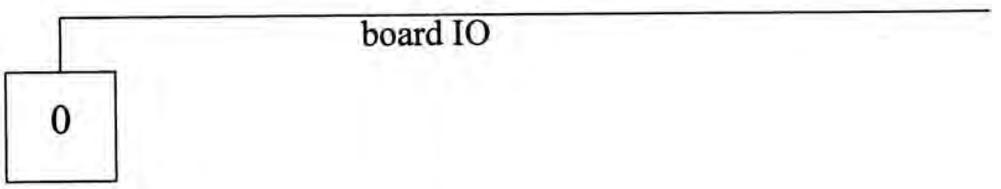


Fig. 45. System IO using board IO.

3.3.4.2 Global bus: 4 IOs used, 3 wasted

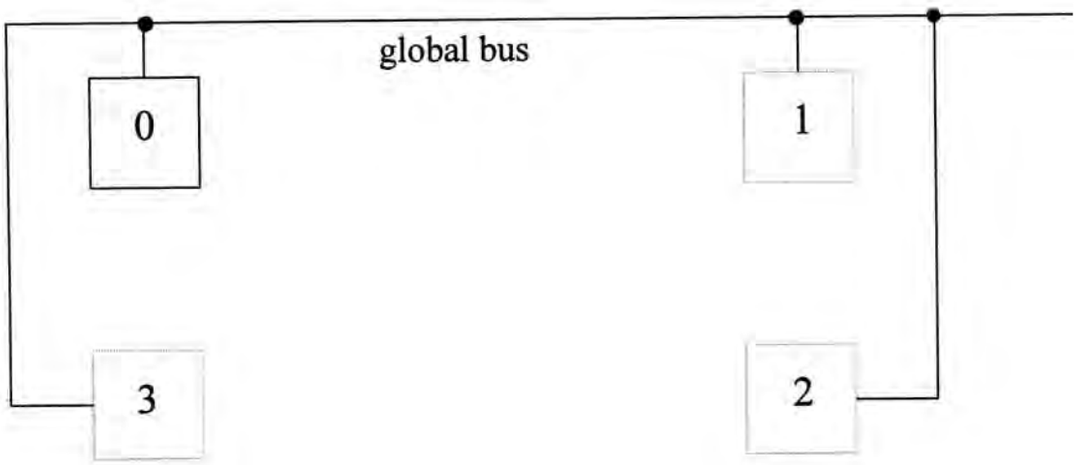


Fig. 46. System IO using global bus.

Among all the possible IO assignments for each net, the one uses and wastes the least IOs will be chosen to assign first. If the best choice in the IO resource is used up, the next best IO assignment will be chosen and so on until there are IO resources available for the net. If all the IO resources are used up and there is still nets that are not yet assigned, then IO assignment fails. The user has to change the partitioning parameters to obtain an alternative partitioned netlists for IO assignments.

3.4 Other Tools

Apart from the partitioning, there are other tools to automate the validation of the circuit in the UPB. The configuration data for the programmable switches connection is generated automatically upon successful completion of IO assignments. The format and the content of the <filename>.con is shown in Appendix D. Both the configuration data of FPGAs and the programmable switches connection are in a format ready for downloading. At any time of operation, the data of RAMs can be read and written from and to the RAMs to ease the validation process.

4. STRUCTURE ANALYSIS

Having decided the types of the buses and IO assignments in the UPB, the number of each kind of buses have to be determined. They were determined by performing a structure analysis in order to obtain the best configuration of the UPB. The best configuration is in the sense that the UPB can realize a wide spectrum of different circuits with all the IOs among the FPGAs ideally assigned and with the least number of IOs used and wasted.

Benchmark circuits selected from the Partitioning benchmark suite of the MCNC Centre for Microelectronic Systems Technologies were used to carry out the structure analysis. They are in the form of Xilinx netlists including both *.xnf and *.map files. Since post-mapping partitioning was adopted to partition the design, *.map files of the benchmark circuits were used. The numbers of CLBs and IOs of the circuits are different from one and others. However, only some were selected in the structure analysis. The maximum CLB utilization rate of the UPB was assumed to be 0.6. There are 144 CLBs in each XC3042 FPGA in the UPB, only benchmark circuits with CLBs less than 144 x 4 x 0.6 = 345 were used. Table 3 shows all the benchmark circuits used and their number of CLBs and IOs in the circuits.

The following outlines the structure analysis method. First of all, the selected benchmark circuits are both balanced partitioned into two to four FPGAs and non-balanced partitioned. In each case, all the CLB utilization rate of 0.3, 0.4, 0.5 and 0.6 are performed. After partitioning, all the IOs including IOs among the FPGAs and system IOs are assigned. Each IO assignment in the circuit is recorded so that the bus ratio of board IO, global bus, local bus, programmable bus to connect 2 FPGAs and programmable bus to connect 3 FPGAs can be found. There are two assumptions in carrying out the structure analysis. It is assumed that the number of IOs in the FPGAs is infinite. This is acceptable because only the ratio of the various kinds of buses have to be found rather than the exact number of the buses. Although there are several IO assignment methods for each nets available, only the one with the least

number of IOs used and wasted is selected. Figure 47 shows all the ideal IO assignment methods. These assignment methods are noted in a data file strana.dat and will be read in each time the structure analysis is performed. The format and the content of the strana.dat is shown on Appendix C.

no.	Benchmark	CLB no.	IO no.	no.	Benchmark	CLB no.	IO no.
1	c17xc3.map	2	7	14	s400xc3.map	32	11
2	c432xc3.map	50	43	15	s420xc3.map	50	23
3	c499xc3.map	66	73	16	s444xc3.map	32	11
4	c880xc3.map	84	86	17	s510xc3.map	68	26
5	c1355xc3.map	70	73	18	s526xc3.map	55	11
6	c1908xc3.map	116	58	19	s526nxc3.map	55	11
7	c3540xc3.map	283	72	20	s820xc3.map	91	39
8	s27xc3.map	3	7	21	s832xc3.map	91	39
9	s208xc3.map	25	15	22	s838xc3.map	102	39
10	s298xc3.map	26	11	23	s953xc3.map	107	41
11	s344xc3.map	20	22	24	s1196xc3.map	143	30
12	s349xc3.map	20	22	25	s1238xc3.map	158	30
13	s382xc3.map	31	11	26	s1423xc3.map	112	24

Table 3. Benchmark circuits selected from MCNC.

As the bus ratio depends on the IO assignment methods which in turn depends on the number of FPGAs the net connected, it is not accurate to take all the results obtained from the structure analysis. For example, if a circuit size is so small that the number of the FPGAs used is two when it is non-balanced partitioned, only board IO, local bus and programmable bus (connect 2 FPGAs) are used while no global bus and programmable bus (connect 3 FPGAs) are assigned because the maximum number of FPGAs connected to the nets is two. This would seriously affect the bus ratio determined if this result was taken into account. Similarly, it is the same case in considering the results obtained from balanced partitioning

into two FPGAs. Hence, only those results of benchmark circuits which have at least three FPGAs in the UPB after non-balanced partitioning are considered as valid results.

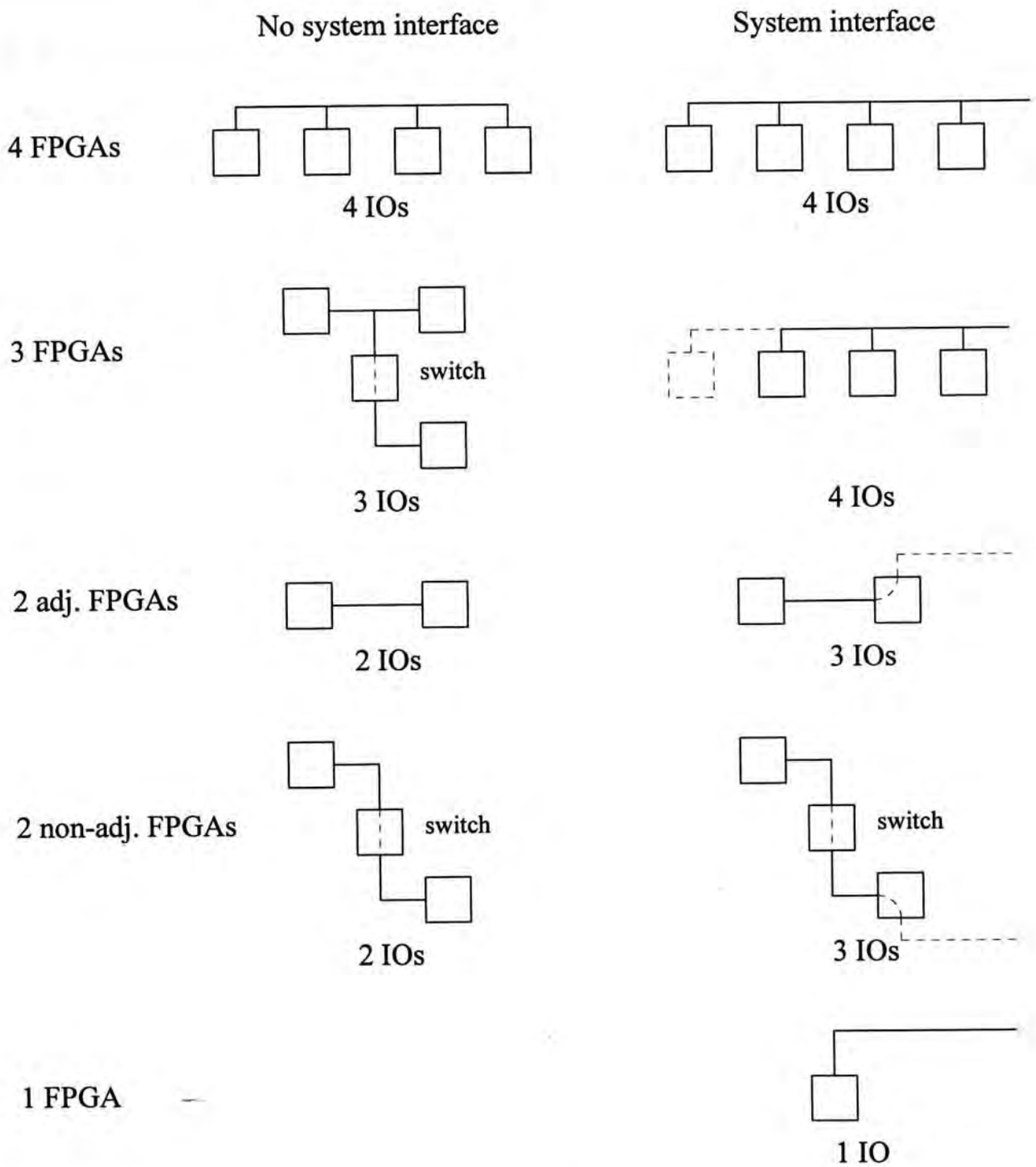


Fig. 47. Ideal IO assignment.

5. RESULTS

Table 4 - 19 shows all the results obtained from the structure analysis. The CLB distribution is the number of CLBs assigned to the FPGA 0, 1, 2, 3 respectively after partitioning. The CLB distribution and the cut size are also shown here just for completeness and reference. The bus ratios are the ratio of board IO to global bus to local bus to programmable bus (connect 2 FPGAs) to programmable bus (connect 3 FPGAs). The bolded numbers of the bus ratio in the tables signify the valid results used to determine the best bus configuration of the UPB. Some entries are empty because the benchmark circuits concerned are so small that they can be fitted into 1 FPGA to obtain 100% routability. In order to provide a clear view of the effects of the partitioning methods on the cut size, the cut size is plotted against each benchmark circuit used in the structure analysis graphically. From the graphs (Figure 48 - 51), one can find the results are reasonable.. The cut size of the circuit is generally smaller when it is non-balanced partitioned, and increases with the number of FPGAs finally obtained from balanced partitioning as explained in the previous section.

Table 4. Non-balanced partitioning, utilization=0.3.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43	{43, 7, 0, 0}	12	43:0:12:0:0
3	c499xc3.map	66	73	{38, 28, 0, 0}	17	72:0:17:0:0
4	c880xc3.map	84	86	{40, 43, 0, 0}	21	81:0:25:0:0
5	c1355xc3.map	70	73	{41, 29, 0, 0}	16	73:0:16:0:0
6	c1908xc3.map	116	58	{36, 41, 0, 39}	54	**41:1:52:12:0**
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23	{9, 41, 0, 0}	7	21:0:7:0:0
16	s444xc3.map	32	11			
17	s510xc3.map	68	26	{25, 43, 0, 0}	33	27:0:33:0:0
18	s526xc3.map	55	11	{42, 13, 0, 0}	11	9:0:11:0:0
19	s526nxc3.map	55	11	{42, 13, 0, 0}	11	9:0:11:0:0
20	s820xc3.map	91	39	{43, 5, 43, 0}	29	**31:0:0:19:5**
21	s832xc3.map	91	39	{43, 13, 35, 0}	35	**32:3:4:15:5**
22	s838xc3.map	102	39	{16, 43, 43, 0}	50	**35:1:33:15:0**
23	s953xc3.map	107	41	{33, 43, 31, 0}	84	**28:5:49:17:4**
24	s1196xc3.map	143	30	{40, 22, 41, 40}	99	**15:13:46:16:4**
25	s1238xc3.map	158	30	{43, 42, 39, 34}	110	**16:13:48:21:6**
26	s1423xc3.map	112	24	{35, 43, 34, 0}	52	**20:3:39:3:2**

Table 5. Non-balanced partitioning into 2, utilization=0.3.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43	{22, 28, 0, 0}	15	43:0:15:0:0
3	c499xc3.map	66	73	{29, 37, 0, 0}	18	72:0:18:0:0
4	c880xc3.map	84	86	{41, 42, 0, 0}	21	78:0:21:0:0
5	c1355xc3.map	70	73	{39, 31, 0, 0}	18	73:0:18:0:0
6	c1908xc3.map	116	58			
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23	{24, 26, 0, 0}	12	21:0:12:0:0
16	s444xc3.map	32	11			
17	s510xc3.map	68	26	{37, 31, 0, 0}	29	25:0:29:0:0
18	s526xc3.map	55	11	{29, 26, 0, 0}	21	8:0:21:0:0
19	s526nxc3.map	55	11	{29, 26, 0, 0}	21	8:0:21:0:0
20	s820xc3.map	91	39			
21	s832xc3.map	91	39			
22	s838xc3.map	102	39			
23	s953xc3.map	107	41			
24	s1196xc3.map	143	30			
25	s1238xc3.map	158	30			
26	s1423xc3.map	112	24			

Table 6. Balanced partitioning into 3, utilization=0.3.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43	{15, 19, 16, 0}	34	43:0:19:7:4
3	c499xc3.map	66	73	{22, 26, 18, 0}	31	65:0:19:12:0
4	c880xc3.map	84	86	{30, 22, 31, 0}	32	75:3:13:9:2
5	c1355xc3.map	70	73	{24, 24, 22, 0}	32	73:0:16:16:0
6	c1908xc3.map	116	58	{43, 39, 34, 0}	64	**49:2:23;11:13**
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23	{15, 16, 19, 0}	22	19:2:13:5:0
16	s444xc3.map	32	11			
17	s510xc3.map	68	26	{21, 23, 24, 0}	59	26:0:25:16:9
18	s526xcc3.map	55	11	{22, 16, 17, 0}	40	8:3:9:7:9
19	s526nxc3.map	55	11	{22, 16, 17, 0}	40	8:3:9:7:9
20	s820xc3.map	91	39	{34, 33, 24, 0}	40	**29:3:20:4:5**
21	s832xc3.map	91	39	{31, 26, 34, 0}	41	**31:3:16:9:5**
22	s838xc3.map	102	39	{35, 40, 27, 0}	39	**35:3:25:6:1**
23	s953xc3.map	107	41	{43, 35, 29, 0}	90	**28:5:41:23:8**
24	s1196xc3.map	143	30			
25	s1238xc3.map	158	30			
26	s1423xc3.map	112	24	{36, 36, 40, 0}	42	**18:3:20:12:2**

Table 7. Balanced partitioning into 4, utilization=0.3.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43	{9, 16, 14, 11}	32	43:3:13:10:0
3	c499xc3.map	66	73	{19, 21, 12, 14}	47	65:0:22:13;6
4	c880xc3.map	84	86	{25, 16, 16, 26}	51	73:3:26:15:2
5	c1355xc3.map	70	73	{19, 16, 22, 13}	48	62:0:27:5:8
6	c1908xc3.map	116	58	{21, 32, 26, 37}	75	**44:3:45:14:5**
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23	{15, 9, 10, 16}	28	19:3:8:12:1
16	s444xc3.map	32	11			
17	s510xc3.map	68	26	{13, 14, 21, 20}	70	25:3:28:21:6
18	s526xc3.map	55	11	{11, 16, 11, 17}	37	8:3:10:6:6
19	s526nxc3.map	55	11	{11, 16, 11, 17}	37	8:3:10:6:6
20	s820xc3.map	91	39	{24, 21, 29, 17}	59	**31:8:21:16:0**
21	s832xc3.map	91	39	{28, 28, 18, 17}	63	**32:8:23:18:0**
22	s838xc3.map	102	39	{25, 21, 27, 29}	43	**36:3:21:13:0**
23	s953xc3.map	107	41	{34, 26, 20, 27}	113	**29:12:46:22:10**
24	s1196xc3.map	143	30	{37, 32, 43, 31}	102	**15:14:41:18:6**
25	s1238xc3.map	158	30	{43, 36, 36, 43}	106	**17:10:61:10:4**
26	s1423xc3.map	112	24	{21, 35, 21, 35}	46	**20:3:19:14:2**

Table 8. Non-balanced partitioning, utilization=0.4.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73	{57, 9, 0, 0}	15	65:0:15:0:0
4	c880xc3.map	84	86	{56, 27, 0, 0}	16	83:0:16:0:0
5	c1355xc3.map	70	73	{56, 14, 0, 0}	19	65:0:197:0:0
6	c1908xc3.map	116	58	{2, 57, 57, 0}	46	**45:1:41:3:0**
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26	{11, 57, 0, 0}	23	28:0:23:0:0
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{57, 34, 0, 0}	24	33:0:24:0:0
21	s832xc3.map	91	39	{57, 34, 0, 0}	28	32:0:22:0:0
22	s838xc3.map	102	39	{0, 47, 55, 0}	19	36:0:19:0:0
23	s953xc3.map	107	41	{0, 57, 50, 0}	53	29:0:53:0:0
24	s1196xc3.map	143	30	{50, 0, 56, 37}	66	15:3:35:21:2
25	s1238xc3.map	158	30	{54, 0, 53, 51}	75	18:5:32:25:4
26	s1423xc3.map	112	24	{49, 57, 6, 0}	33	20:3:24:3:0

Table 9. Balanced partitioning into 2, utilization=0.4.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73	{29, 37, 0, 0}	18	72:0:18:0:0
4	c880xc3.map	84	86	{41, 42, 0, 0}	21	78:0:21:0:0
5	c1355xc3.map	70	73	{39, 31, 0, 0}	18	73:0:18:0:0
6	c1908xc3.map	116	58			
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26	{37, 31, 0, 0}	29	25:0:29:0:0
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{40, 51, 0, 0}	23	33:0:23:0:0
21	s832xc3.map	91	39	{45, 46, 0, 0}	22	34:0:22:0:0
22	s838xc3.map	102	39	{54, 48, 0, 0}	22	36:0:22:0:0
23	s953xc3.map	107	41	{57, 50, 0, 0}	52	32:0:52:0:0
24	s1196xc3.map	143	30			
25	s1238xc3.map	158	30			
26	s1423xc3.map	112	24			

Table 10. Balanced partitioning into 3, utilization=0.4.

Benchmark no.	Bench Mark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73	{22, 26, 18, 0}	31	65:0:19:12:0
4	c880xc3.map	84	86	{30, 22, 31, 0}	32	75:3:13:9:2
5	c1355xc3.map	70	73	{24, 24, 22, 0}	32	73:0:16:16:0
6	c1908xc3.map	116	58	{39, 46, 31, 0}	58	**39:2:29:5:10**
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26	{21, 23, 24, 0}	59	26:0:25:16:9
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{34, 33, 24, 0}	40	29:3:20:4:5
21	s832xc3.map	91	39	{31, 26, 34, 0}	41	31:3:16:9:5
22	s838xc3.map	102	39	{35, 40, 27, 0}	39	35:3:25:6:1
23	s953xc3.map	107	41	{43, 35, 29, 0}	90	28:5:41:23:8
24	s1196xc3.map	143	30	{54, 51, 38, 0}	64	**15:5:29:21:2**
25	s1238xc3.map	158	30	{53, 48, 57, 0}	84	**17:7:40:24:3**
26	s1423xc3.map	112	24	{33, 34, 45, 0}	37	**19:3:18:11:1**

Table 11. Balanced partitioning into 4, utilization=0.4.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73	{19, 21, 12, 14}	47	65:0:32:13:6
4	c880xc3.map	84	86	{25, 16, 16, 26}	51	73:3:26:15:2
5	c1355xc3.map	70	73	{19, 16, 22, 13}	48	62:0:275:8
6	c1908xc3.map	116	58	{21, 32, 26, 37}	75	**44:3:45:14:5**
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26	{13, 14, 21, 20}	70	25:3:28:21:6
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{24, 21, 29, 17}	59	31:8:21:16:0
21	s832xc3.map	91	39	{28, 28, 18, 17}	63	32:8:23:18:0
22	s838xc3.map	102	39	{25, 21, 27, 29}	43	36:3:21:13:0
23	s953xc3.map	107	41	{34, 26, 20, 27}	113	29:12:46:22:10
24	s1196xc3.map	143	30	{42, 27, 45, 29}	96	**15:10:39:26:5**
25	s1238xc3.map	158	30	{48, 33, 29, 48}	106	**16:10:55:16:6**
26	s1423xc3.map	112	24	{21, 35, 21, 35}	46	**20:3:19:14:2**

Table 12. Non-balanced partitioning, utilization=0.5.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73			
4	c880xc3.map	84	86	{11, 72, 0, 0}	15	78:0:15:0:0
5	c1355xc3.map	70	73			
6	c1908xc3.map	116	58	{72, 44, 0, 0}	35	50:0:35:0:0
7	c3540xc3.map	283	72	{69, 70, 72, 72}	126	**50:10:50:37:8**
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26			
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{72, 19, 0, 0}	17	32:0:17:0:0
21	s832xc3.map	91	39	{72, 19, 0, 0}	16	33:0:16:0:0
22	s838xc3.map	102	39	{33, 69, 0, 0}	18	35:0:18:0:0
23	s953xc3.map	107	41	{35, 72, 0, 0}	51	28:0:51:0:0
24	s1196xc3.map	143	30	{71, 72, 0, 0}	40	22:0:40:0:0
25	s1238xc3.map	158	30	{72, 14, 72, 0}	88	**17:5:20:54:2**
26	s1423xc3.map	112	24	{40, 72, 0, 0}	17	22:0:17:0:0

Table 13. Balanced partitioning into 2, utilization=0.5.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73			
4	c880xc3.map	84	86	{41, 42, 0, 0}	21	78:0:21:0:0
5	c1355xc3.map	70	73			
6	c1908xc3.map	116	58	{66, 50, 0, 0}	34	48:0:34:0:0
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26			
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{40, 51, 0, 0}	23	33:0:23:0:0
21	s832xc3.map	91	39	{45, 46, 0, 0}	22	34:0:22:0:0
22	s838xc3.map	102	39	{54, 48, 0, 0}	22	36:0:22:0:0
23	s953xc3.map	107	41	{51, 56, 0, 0}	53	30:0:53:0:0
24	s1196xc3.map	143	30	{71, 72, 0, 0}	40	19:0:40:0:0
25	s1238xc3.map	158	30			
26	s1423xc3.map	112	24	{50, 62, 0, 0}	19	20:0:194:0:0

Table 14. Balanced partitioning into 3, utilization=0.5.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73			
4	c880xc3.map	84	86	{30, 22, 31, 0}	32	75:3:13:9:2
5	c1355xc3.map	70	73			
6	c1908xc3.map	116	58	{39, 46, 31, 0}	58	39:2:29:5:10
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26			
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{34, 33, 24, 0}	40	29:3:20:4:5
21	s832xc3.map	91	39	{31, 26, 34, 0}	41	31:3:16:9:5
22	s838xc3.map	102	39	{35, 40, 27, 0}	39	35:3:26:6:1
23	s953xc3.map	107	41	{43, 35, 29, 0}	90	28:5:41:23:8
24	s1196xc3.map	143	30	{54, 51, 38, 0}	64	15:5:29:21:2
25	s1238xc3.map	158	30	{63, 42, 53, 0}	76	**16:8:31:21:4**
26	s1423xc3.map	112	24	{33, 34, 45, 0}	37	19:3:18:11:1

Table 15. Balanced partitioning into 4, utilization=0.5.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73			
4	c880xc3.map	84	86	{25, 16, 16, 26}	51	73:3:26:15:2
5	c1355xc3.map	70	73			
6	c1908xc3.map	116	58	{21, 32, 26, 37}	75	44:3:45:14:5
7	c3540xc3.map	283	72	{69, 72, 70, 72}	132	**48:8:64:41:4**
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26			
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{24, 21, 29, 17}	59	31:8:21:16:0
21	s832xc3.map	91	39	{28, 28, 18, 17}	63	32:8:23:18:0
22	s838xc3.map	102	39	{25, 21, 27, 29}	43	36:3:21:13:0
23	s953xc3.map	107	41	{34, 26, 20, 27}	113	29:12:46:22:10
24	s1196xc3.map	143	30	{42, 27, 45, 29}	96	15:10:39:26:5
25	s1238xc3.map	158	30	{48, 33, 29, 48}	106	**16:10:55:16:6**
26	s1423xc3.map	112	24	{21, 35, 21, 35}	46	20:3:19:14:2

Table 16. Non-balanced partitioning, utilization=0.6.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73			
4	c880xc3.map	84	86			
5	c1355xc3.map	70	73			
6	c1908xc3.map	116	58	{85, 31, 0, 0}	23	57:0:23:0:0
7	c3540xc3.map	283	72	{73, 86, 86, 38}	113	**53:10:59:22:5**
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26			
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{86, 5, 0, 0}	5	39:0:5:0:0
21	s832xc3.map	91	39	{86, 5, 0, 0}	5	39:0:5:0:0
22	s838xc3.map	102	39	{16, 86, 0, 0}	13	37:0:13:0:0
23	s953xc3.map	107	41	{21, 86, 0, 0}	40	32:0:40:0:0
24	s1196xc3.map	143	30	{86, 57, 0, 0}	38	20:0:38:0:0
25	s1238xc3.map	158	30	{86, 72, 0, 0}	46	22:0:0:46:0
26	s1423xc3.map	112	24	{26, 86, 0, 0}	17	20:0:17:0:0

Table 17. Balanced partitioning into 2, utilization=0.6.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73			
4	c880xc3.map	84	86			
5	c1355xc3.map	70	73			
6	c1908xc3.map	116	58	{66, 50, 0, 0}	34	48:0:34:0:0
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26			
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{40, 51, 0, 0}	23	33:0:23:0:0
21	s832xc3.map	91	39	{45, 46, 0, 0}	22	34:0:22:0:0
22	s838xc3.map	102	39	{54, 48, 0, 0}	22	36:0:22:0:0
23	s953xc3.map	107	41	{51, 56, 0, 0}	53	30:0:53:0:0
24	s1196xc3.map	143	30	{75, 68, 0, 0}	39	22:0:39:0:0
25	s1238xc3.map	158	30	{86, 72, 0, 0}	45	22:0:45:0:0
26	s1423xc3.map	112	24	{50, 62, 0, 0}	19	20:0:19:0:0

Table 18. Balanced partitioning into 3, utilization=0.6.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73			
4	c880xc3.map	84	86			
5	c1355xc3.map	70	73			
6	c1908xc3.map	116	58	{39, 46, 31, 0}	58	39:2:46:5:0
7	c3540xc3.map	283	72			
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26			
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{34, 33, 24, 0}	40	29:3:20:4:5
21	s832xc3.map	91	39	{31, 26, 34, 0}	41	31:3:16:9:5
22	s838xc3.map	102	39	{35, 40, 27, 0}	39	35:3:25:6:1
23	s953xc3.map	107	41	{43, 35, 29, 0}	90	28:5:41:23:8
24	s1196xc3.map	143	30	{54, 51, 38, 0}	64	15:5:29:21:2
25	s1238xc3.map	158	30	{63, 42, 53, 0}	76	16:8:31:21:4
26	s1423xc3.map	112	24	{33, 34, 45, 0}	37	19:3:18:11:1

Table 19. Balanced partitioning into 4, utilization=0.6.

Benchmark no.	Benchmark	CLB no.	IO no.	CLB dist.	Cut size	Bus ratio
1	c17xc3.map	2	7			
2	c432xc3.map	50	43			
3	c499xc3.map	66	73			
4	c880xc3.map	84	86			
5	c1355xc3.map	70	73			
6	c1908xc3.map	116	58	{21, 32, 26, 37}	75	44:3:45:14:5
7	c3540xc3.map	283	72	{84, 67, 77, 55}	134	**53:13:76:24:2**
8	s27xc3.map	3	7			
9	s208xc3.map	25	15			
10	s298xc3.map	26	11			
11	s344xc3.map	20	22			
12	s349xc3.map	20	22			
13	s382xc3.map	31	11			
14	s400xc3.map	32	11			
15	s420xc3.map	50	23			
16	s444xc3.map	32	11			
17	s510xc3.map	68	26			
18	s526xc3.map	55	11			
19	s526nxc3.map	55	11			
20	s820xc3.map	91	39	{24, 21, 29, 17}	59	31:8:21:16:0
21	s832xc3.map	91	39	{28, 28, 18, 17}	63	32:8:23:18:0
22	s838xc3.map	102	39	{25, 21, 27, 29}	43	36:3:21:13:0
23	s953xc3.map	107	41	{34, 26, 20, 27}	113	29:12:46:22:10
24	s1196xc3.map	143	30	{42, 27, 45, 29}	96	15:10:39:26:5
25	s1238xc3.map	158	30	{48, 33, 29, 48}	106	16:10:55:16:6
26	s1423xc3.map	112	24	{21, 35, 21, 35}	46	20:3:19:14:2

Fig. 48. Cut size of benchmark circuits with utilization rate = 0.3.

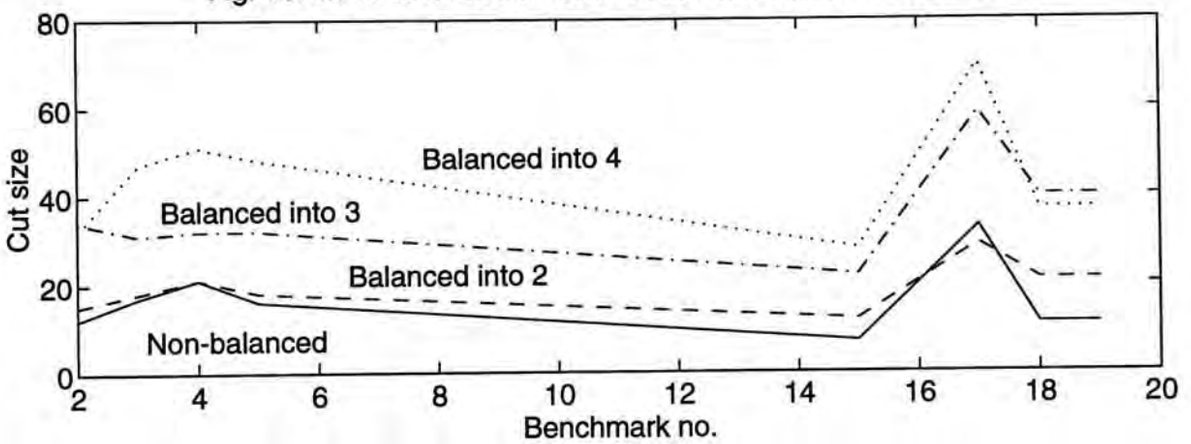


Fig. 49. Cut size of benchmark circuits with utilization rate = 0.4.

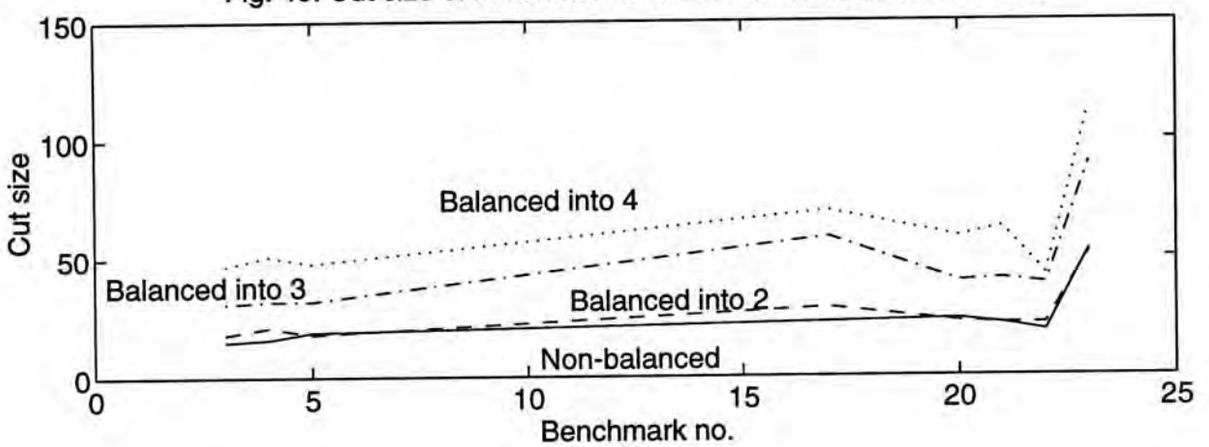


Fig. 50. Cut size of benchmark circuits with utilization rate = 0.5.

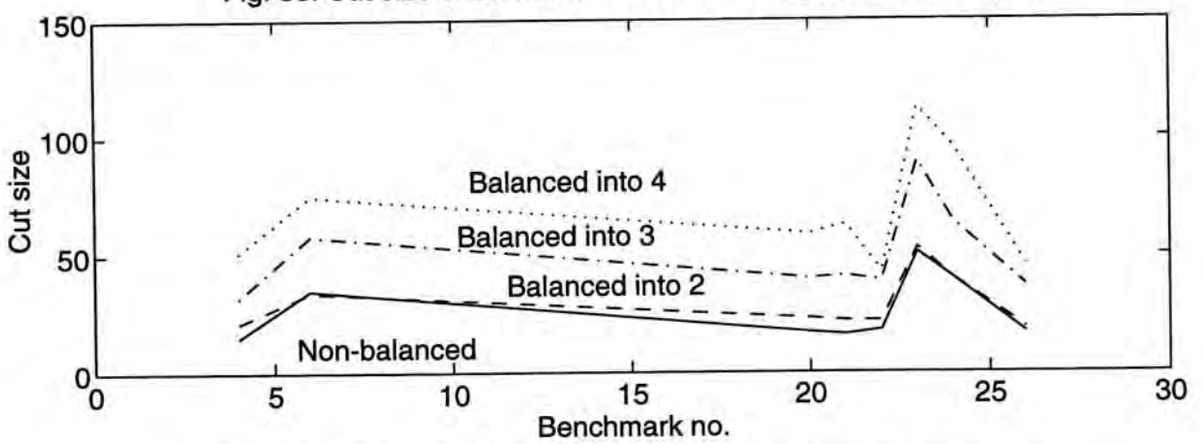
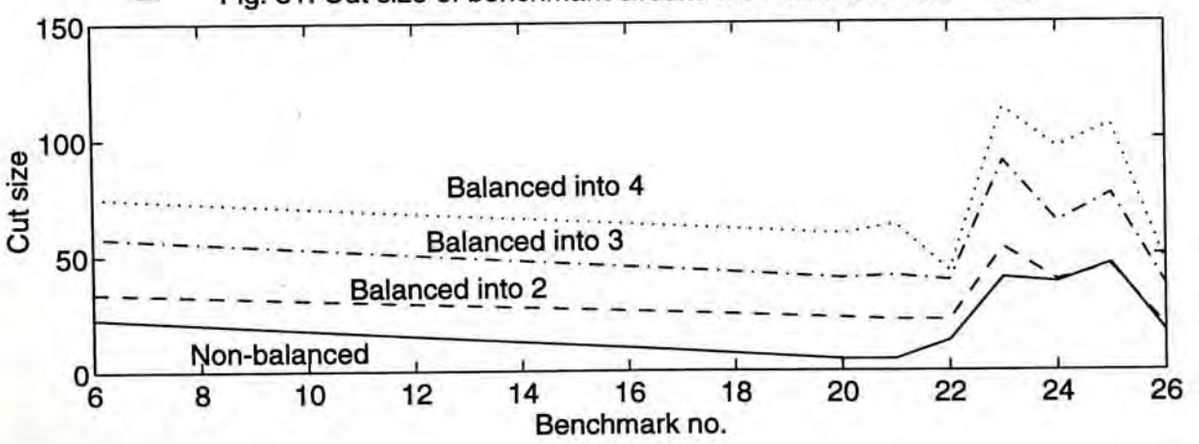


Fig. 51. Cut size of benchmark circuits with utilization rate = 0.6.



The average number of each type of buses from the valid data is calculated and the final average bus ratio is

$$7.34 : 1.49 : 8.70 : 4.16 : 1.$$

We can see that more local buses are used than the programmable bus, the ratio is 8.70 : 4.16 for nets connected to 2 FPGAs. The reason is obvious because the partitioner assigns circuits which are too large to fit into a single FPGA, into two adjacent chips rather than two widely separated FPGAs.

Since board IO, global bus, local bus, programmable bus connect to 2 FPGAs and programmable bus connected to 3 FPGAs require 1, 4, 2, 2, 3 IOs respectively, the final ratio of IOs is

$$7.34 : 5.97 : 17.41 : 8.33 : 3.$$

The total IOs available for the buses in the UPB is 282. According to the ratio above, the number of different kinds of buses allocated is as follow:

IOs for Board IO	=	*49.20*	~	*14 + 14 + 14 + 8 = 50;*
IOs for Global bus	=	*40.07*	~	*10 x 4 = 40;*
IOs for Local bus	=	*116.75*	~	*16 x 2 x 4 = 112;*
IOs for Prog. bus (2)	=	*55.86*	~	*14 x 4 = 56;*
IOs for Prog. bus (3)	=	*20.12*	~	*2 x (4 x 2) + 2 x (4) = 24.*

The bus allocation is shown in Figure 52. The number of board IO comes from FPGA 3 is 6 less than that from FPGA 0 - 2 because a total of 16 IOs is used in FPGA 3 for the address bus and the control bus of the address generator while only 10 IOs are used in FPGA 0 - 2 for the data and control bus of the RAM 0 - 2. Such board IO allocation can make the bus structure more symmetrical. It should also be noted that the programmable bus to connect 3 FPGAs requires 1 odd path and 1 even path, totally 3 IOs. Hence, it was chosen to allocate 2 IOs for the even paths and 2 IOs for the odd path.

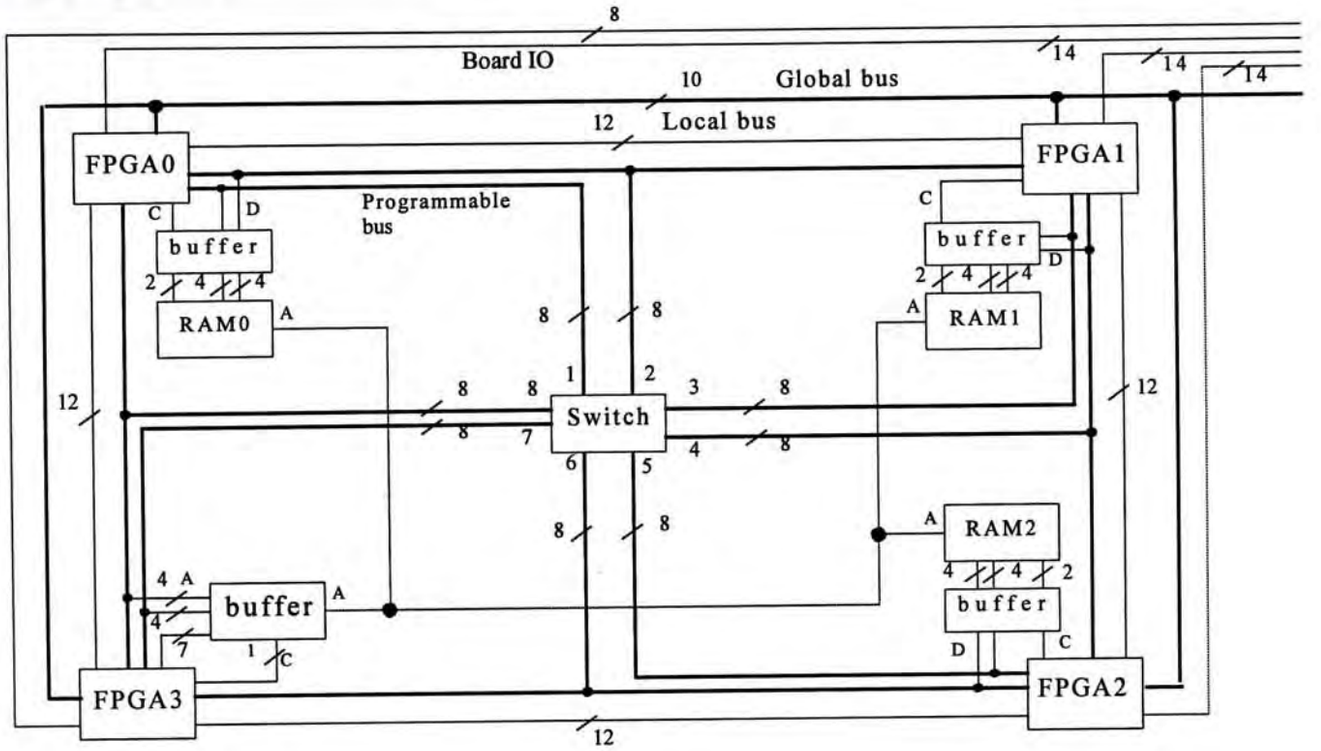


Fig. 53. Final bus allocation in UPB.

6. FUTURE DIRECTION

6.1 Other Possible Configurations

The UPB uses four Xilinx XC3042 FPGAs as the prototype implementation technologies. It is estimated the maximum CLB utilization rate is 0.6. There are around 10,000 gate count available. Actually, it has not been investigated that the best number of FPGAs in the UPB is 4. May be it would be better to use larger capacity FPGAs and more number of FPGAs in the UPB because at least more gate count will be contributed for the larger and more complex designs. The structure analysis used in the UPB can be adopted to find the best bus interconnections for emulation boards with different number of FPGAs provided that the connection method for each type of nets is specified. This structure analysis is suitable for the emulation board which have 4 - 8 FPGAs.

The final bus ratio depends greatly on the structure analysis. During the structure analysis, the circuit is first partitioned. So the partitioning algorithm has great effect on the final bus structure. The F & M algorithm is quite fast (almost linear with the circuit size) and simple to implement. However, there are other algorithms available to partition the circuit. Although its computation time is higher and the algorithms are much more sophisticated, the cut size between the partitioned component is smaller. Perhaps great enhancement would be obtained from a better algorithm.

6.2 Programmable Interconnection

In the UPB, 8 x 8 cross-point switches are used as the routing resource between the FPGAs. There is a limitation here. If the number of buses in either dimension is greater than 8 (such as 15 x 2 in the UPB), it is very difficult to realize it. Even though it can be realized by cascading more switches, the delay would be very large. A better approach for future UPB

development is to determine the best bus structure first. After knowing the dimensions of the required switches, use either an intermediate FPGA or custom-made programmable switches as the routing resources. Then any configuration of bus structure could be implemented as the FPGA provides a large number of IOs and the custom programmable switches is tailored for the board designed.

6.3 Expandability of UPB

The global bus and board IO in the UPB provide the interface to expand the UPB by cascading more UPBs together. The UPBs can be interconnected by just using the hardwired buses. Another option is to scale up the UPBs by the programmable interconnection with the same bus interconnection structure between the FPGAs in a single UPB. In other word, the structure analysis can find the final bus structure not only between the FPGAs bus also between UPBs. And all the buses including Board IO, global bus, local bus and programmable bus are scaled up in the same manner. The last option is that the UPBs can be connected by both the hardwired buses and the programmable bus designed differently. However, no one knows which one is better. The best configuration requires thorough study and research.

7. CONCLUSION

An Universal Prototyping Board (UPB) was developed to overcome the disadvantages of traditional design prototyping. It provides a fast, flexible, reconfigurable and in-circuit validation for any digital designs. Although such an emulation board exits commercially and academically, they are either too expensive and too large capacity for general applications or too inflexible. The UPB was successfully implemented which has the flexibility for design modification and requires lower cost.

This UPB is based on the Field Programmable Gate Array (FPGAs) and programmable interconnections. Both hardwired and programmable buses are used in between the FPGAs. These buses were carefully designed so that each net will have little delays after net assignment. Besides, external RAMs are available for the memory-intensive designs. Furthermore, a microprocessor acts both as the controller of the UPB and one of the emulated hardware.

In order to automate the validation process, partitioning tool was developed to partition the circuit into multiple sets of netlists target for each FPGA. It performs the critical path analysis before partitioning to ensure the operation speed or timing specification requirement. Different parameters can be adjusted by the user to obtain the most desirable partitioned netlists.

The ratio of various kinds of buses found was proved to be the optimum through the structure analysis. Although, the optimum number of FPGAs used in the UPB has not been studied, this structure analysis can still be used to find the optimum bus ratio in UPBs which have 4 to 8 FPGAs.

In short, with this emulation board, software tools and structure analysing tools, almost all digital circuits can be realized in the UPB, and one can design a different sized UPB with the optimum interconnect structure.

BIBLIOGRAPHY

[1] N. Zafar, "Managing Risk in ASIC Design Cycle," *Proc. Third Annual IEEE ASIC Seminar & Exhibit,* Sep. 1990.

[2] S. Walters, "Reprogrammable Hardware Emulation Automates System-Level ASIC Validation," *WESCON/90 Conference Record,* Nov. 1990.

[3] S.D. Brown, R.J. Francis, J. Rose & Z.G. Vranesic, *Field-Programmable Gate Array,* Kluwer Academic Publishers, 1992.

[4] S. Walters, "Reprogrammable Hardware Emulation for ASICs Makes Thorough Design Verification Practical," *COMPCON Spring '89,* Thirty-Fourth IEEE Computer-Society International Conference, 1989.

[5] S. Walters, "Computer-Aided Prototyping for ASIC-Based Systems," *IEEE Design and Test of Computers*, vol. 8, iss. 2, 1991, pp.4-10.

[6] R. J. Hasslen & N. Zafar, "A Validation Strategy for Embedded Core ASICs," *Proc. Third Annual IEEE ASIC Seminar & Exhibit*, Sep. 1990.

[7] B. Tuck, "Prototyping System Emulates Up To Six Million Gates," *Computer Design*, Nov.1992, pp.119.

[8] L. Maliniak, "Logic Emulation Promotes Parallel Design Methods," *Electronic Design*, Apr. 1992, pp.104-105.

[9] S. J. Cravatta, "Logic Cell Emulation for ASIC In-Circuit Emulators," *Proc. Third Annual IEEE ASIC Seminar & Exhibit*, Sept. 1990.

[10] D. Pasternak & T. Hike, "In-Circuit-Emulation in ASIC architectural Core Designs," *Proc. Second Annual IEEE ASIC & Exhibit*, Sep. 1989.

[11] D.E.V.D. Bout, J.N. Morris, D. Thomae, S. Labrozzi, S. Wingo & D. Hallman, "AnyBoard: An FPGA-Based, Reconfigurable System," *IEEE Design & Test of Computers*, Sept. 1992, pp. 22-30.

[12] *The Programmable Gate Array Data Book,* Xilinx Inc., 1991.

[13] *User Guide and Tutorial,* Xilinx Inc., 1991.

[14] A. Clements, *Microprocessor Systems Design 68000 Hardware, Software, and Interfacing,* PWS-KENT Publishing Co., 1987.

[15] A.D. Wilcox, *68000 Microprocessor Systems: Designing and Troubleshooting,* Prentice-Hall Inc., 1987.

[16] J.Buchanan, *System Timing in CMOS/TTL Digital Systems Design,* McGraw-Hill Publishing Co., 1990, pp. 199-215.

[17] K. Perry, "Partitioning Logic Designs into FPGAs," in *Electronic Engineer,* May 1993, pp. 86-91.

[18] S. Goto & T. Matsuda, "Partitioning, Assignment and Placement," in *Layout Design and Verification,* T. Ohtsuki ed., Elsevier Science Publishers B. V. (North-Holland), 1986, pp. 55-64.

[19] W.E. Donath, "Logic Partitioning, " in *Physical Design Automation of VLSI Systems,* B. Preas & M. Lorenzetti ed., The Benjamin. Cummings Publishing Co., Inc., 1988, pp. 65-86.

[20] T. Lengaure, "Circuit Partitioning," in *Combinatiorial Algorithms,* John Wiley & Sons, 1990, pp. 251-273.

[21] B.W. Kernighan & S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Syst. Techn. J.,* vol. 49, Feb. 1970, pp. 291-307.

[22] C. M. Fiduccia & R.M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *Proc. 19th Design Automation Conference,* 1982, pp. 175-181.

[23] B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks," *IEEE Transaction on Computers,* vol. c-33, no. 5, May 1984, pp.438-446.

[24] D.G. Schweikert & B.W. Kernighan, "A Proper Model for the Partitioning of Electrical Circuits," *Proc. 9th Design Automation Workshop,* Dallas, Jun. 1979, pp. 57-62.

[25] L.A. Sanchis, "Multiple-Way Network Partitioning," *IEEE Transaction on Computers,* vol. 38, no. 1, Jan. 1989, pp. 62-81.

[26] W.Y. Lo, C.S. Choy & C.F. Chan, "Hardware Emulation Board Based on FPGAs and Programmable Interconnections," Fifth International Workshop on Rapid System Prototyping, Jun. 1994, pp. 126-130.

APPENDICES

APPENDIX A

Delay.dat

The content of the delay.dat is shown as follows:

```
AND 1
OR 1
XOR 1
NAND 1
NOR 1
XNOR 1
INV 1
DFF 1
ACLK 1
GCLK 1
OSC 1
PATH 1
```

On each line, the name and the delay of the primitive gate is specified. It is no need to put the names of the gates in alphabetical order. However, the names of the gates must be in capital letter, and the spellings of the primitive gates should be the same as that in *.map files. PATH means the wiring path.

APPENDIX B

Command line argument options

Particulars	Argument	Options	Description	Defaults
Critical path analysis	-f	--	normal critical path analysis	no critical path analysis
	-c	--	critical path analysis, treat all clocked elements as combinational elements	
Partitioning	-b	2, 3 or 4	balanced partitioning into 2, 3 or 4 FPGAs	non-balanced partitioning
Utilization rate	-u	1 - 9	utilization rate of CLB in each FPGA, 1 - 9 means utilization rate = 0.1 - 0.9 respectively	5

Example 1: parti <filename>[.map] -f -b/2 -f <ENTER>

Meanings: normal critical path analysis, balanced partitioning into 2 FPGAs, utilization rate = 0.6

Example 2: parti <filename>[.map] -u/4 <ENTER>

Meanings: no critical path analysis, non-balanced partitioning, utilization rate = 0.4

APPENDIX C

Iosource.dat

The content of iosource.dat is shown as follows:

```
BOARD IO
LCA0..2: 25..27, 30, 33..40, 49, 76;
LCA3: 25..27, 30, 33..35, 76;
END
GBL BUS
LCA0..3: 42, 44..48, 50, 53, 61, 66;
END
PROG INTC
P1: 28, 56..60, 62..63;
P3: 29, 65, 67..71, 75;
P5: 29, 65, 67..71, 75;
P7: 29, 65, 67..71, 75;
P2,LCA0: 29, 65, 67..71, 75;
  ,LCA1: 77..84;
P4,LCA1: 28, 56..60, 62..63;
  ,LCA2: 77..84;
P6,LCA2: 28, 56..60, 62..63;
  ,LCA3: 77..84;
P8,LCA3: 28, 56..60, 62..63;
  ,LCA0: 77..84;
END
LOC BUS
LCA0,1: 2..9, 16, 24, 41, 73:72;
LCA1,2: 10..11, 13..15, 17..21, 23, 73:72;
LCA2,3: 2..9, 16, 24, 41, 73:72;
LCA3,0: 10..11, 13..15, 17..21, 23, 73:72;
END
ADD BUS
LCA3,A0: 36..40, 49, 52;
    ,S8: 56, 58, 60, 62;
    ,S7: 65, 67, 70, 75;
END
DATA BUS
LCA0,S2: 75, 70, 67, 65;
    ,S1: 62, 60, 58, 56;
LCA1,S3: 75, 70, 67, 65;
    ,S4: 62, 60, 58, 56;
LCA2,S5: 75, 70, 67, 65;
    ,S6: 62, 60, 58, 56;
END
EOF
```

All alphabets are written in capital letters. The iosource.dat is divided into a number of records which is delimited by the record name and END record. The order of placement of the records are fixed as follows: Board IO, Global bus, Programmable interconnect, Local bus, Address bus and Data bus. The last record must be the EOF (end of file) record.

C-1 Numerical value

The numerical values include the LCA numbers and the pin numbers. They can be listed one by one separated by a comma ','. Also, two full stop '..' can be used to list a range of number. For example, both 3, 4, 5, 6 and 3..6 means pin 3, pin 4, pin 5 and pin 6. Similarly, both LCA0..2 and LCA0, LCA1 and LCA2 means the FPGA 0, FPGA 1 and FPGA 2 and FPGA 3. It should be noted that a semicolon ';' instead of the comma is used to illustrate the last pin number or last pin range.

C-2 Records

In the following, the formats of all records are illustrated. i, j and k are the substitute of the numerical value.

Board IO

Record Name: BOARD IO

Format: *LCAi[..j]: <pin number or pin range>;*

Description: LCAi means FPGA i.

Global bus

Record Name: GBL BUS

Format: *LCAi[..j]: <pin number or pin range>;*

Programmable bus

Record Name: PROG INTC

Format: *Pi: <pin number or pin range>;* for odd paths,

Pi, LCAj: <pin number or pin range>; for even paths

Description: Pi means path number i. For the even paths, it has to tell which two FPGAs the even path connect to.

Local bus

Record name: LOC BUS

Format: *LCAi, j: <pin number or pin range>;*

Description: It has to state which FPGA (i and j) the local bus connect to. If the local bus connects to the same pin number in both FPGA, just write the pin number as other numerical values. If not, use colon to state them. For instance, in the iosource.dat shown before, 73:72 in the line 1 of LOC BUS record means the local bus connects to pin 73 on FPGA 0 and pin 72 on FPGA 1.

Address bus

Record name: ADD BUS

Format: *LCAi,Aj:<pin number or pin range>;* or

,Sk: <pin number or pin range>;

Description: In the UPB, 8 address buses are shared with the programmable bus while 4 are not, Sk states the shared address bus and Aj states the non-shared address bus. Pin numbers should be listed in an order correspond to A1 - A15.

Data bus

Record name: DATA BUS

Format: *LCAi,Sj: <pin number or pin range>;* or

,Sk: <pin number or pin range>;

Description: Pin numbers should be listed in an order corresponds to D0 - D15.

APPENDIX D

<filename>.con

The <filename>.con is the Hex-character data file for programming the switch connection between the FPGAs. In the UPB, the switch address starts from $140000 and ends $14007F. Hence, there are totally 128 data and 8 lines in the file. The first data corresponds to the address $140000 and the second corresponds to the address $140001 and so on.

Example:

```
00030000000000000000000000000004
00000003000000000000000800000000
00000000000300000000000000000000
00000000000000030000000000000000
00000010000000000002000000000000
00000000000000000000000200000000
00200000000000000000000000020000
00000000000000000000000000000002
```

APPENDIX E

Table of pin number for different buses

Legend:

1. Board IO
2. Global bus
3. Local bus
4. Programmable interconnect (path)

A. Address bus

D. Data bus

C. Control bus

*** Permanent function on such pin

Pin #	Function	FPGA 0	FPGA 1	FPGA 2	FPGA 3
1	GND	***	***	***	***
2	A13 - I/O	3	3	3	3
3	A6 - I/O	3	3	3	3
4	A12 - I/O	3	3	3	3
5	A7 - I/O	3	3	3	3
6	I/O	3	3	3	3
7	I/O	3	3	3	3
8	A11 - I/O	3	3	3	3
9	A8 - I/O	3	3	3	3
10	A10 - I/O	3	3	3	3
11	A9 - I/O	3	3	3	3
12	PWRDN*	***	***	***	***
13	TCLKIN - I/O	3	3	3	3
14	I/O	3	3	3	3
15	I/O	3	3	3	3
16	I/O	3	3	3	3
17	I/O	3	3	3	3
18	I/O	3	3	3	3
19	I/O	3	3	3	3
20	I/O	3	3	3	3
21	I/O	3	3	3	3
22	VCC	***	***	***	***
23	I/O	3	3	3	3
24	I/O	3	3	3	3

25	I/O	1	1	1	1
26	I/O	1	1	1	1
27	I/O	1	1	1	1
28	I/O	4 (1)	4 (4)	4 (6)	4 (8)
29	I/O	4 (2)	4 (3)	4 (5)	4 (7)
30	I/O	1	1	1	1
31	M1 - RDATA*	***	***	***	***
32	M0 - RTRIG	***	***	***	***
33	M2 - I/O	1	1	1	1
34	HDC - I/O	1	1	1	1
35	I/O	1	1	1	1
36	LDC* - I/O	1	1	1	1
37	I/O	1	1	1	A
38	I/O	1	1	1	A
39	I/O	1	1	1	A
40	I/O	1	1	1	A
41	I/O	3	3	3	3
42	INIT* - I/O	2	2	2	2
43	GND	***	***	***	***
44	I/O	2	2	2	2
45	I/O	2	2	2	2
46	I/O	2	2	2	2
47	I/O	2	2	2	2
48	I/O	2	2	2	2
49	I/O	1	1	1	A
50	I/O	2	2	2	2
51	I/O	C	C	C	C
52	I/O	C	C	C	A
53	XTL2(IN) - I/O	2	2	2	2
54	RESET*	***	***	***	***
55	DONE/PG*	***	***	***	***
56	D7 - I/O	4 (1) , D	4 (4), D	4 (6), D	4 (8), A
57	XTL1(OUT) - BCLKIN -I/O	4 (1)	4 (4)	4 (6)	4 (8)
58	D6 - I/O	4 (1), D	4 (4), D	4 (6), D	4 (8), A
59	I/O	4 (1)	4 (4)	4 (6)	4 (8)
60	D5 - I/O	4 (1), D	4 (4), D	4 (6), D	4 (8), A
61	CS0* - I/O	2	2	2	2

62	D4 - I/O	4 (1), D	4 (4), D	4 (6), D	4 (8), A
63	I/O	4 (1)	4 (4)	4 (6)	4 (8)
64	VCC	***	***	***	***
65	D3 - I/O	4 (2), D	4 (3), D	4 (5), D	4 (7), A
66	CS1* - I/O	2	2	2	2
67	D2 - I/O	4 (2), D	4 (3), D	4 (5), D	4 (7), A
68	I/O	4 (2)	4 (3)	4 (5)	4 (7)
69	I/O	4 (2)	4 (3)	4 (5)	4 (7)
70	D1 - I/O	4 (2), D	4 (3), D	4 (5), D	4 (7), A
71	RDY/BUSY* - RCLK* - I/O	4 (2)	4 (3)	4 (5)	4 (7)
72	D0 - DIN - I/O	3	3	3	3
73	DOUT - I/O	3	3	3	3
74	CCLK	***	***	***	***
75	A0 - WS* - I/O	4 (2), D	4 (3), D	4 (5), D	4 (7), A
76	A1 - CS2 * - I/O	1	1	1	1
77	A2 - I/O	4 (8)	4 (2)	4 (4)	4 (6)
78	A3 - I/O	4 (8)	4 (2)	4 (4)	4 (6)
79	I/O	4 (8)	4 (2)	4 (4)	4 (6)
80	I/O	4 (8)	4 (2)	4 (4)	4 (6)
81	A15 - I/O	4 (8)	4 (2)	4 (4)	4 (6)
82	A4 - I/O	4 (8)	4 (2)	4 (4)	4 (6)
83	A14 - I/O	4 (8)	4 (2)	4 (4)	4 (6)
84	A5 - I/O	4 (8)	4 (2)	4 (4)	4 (6)

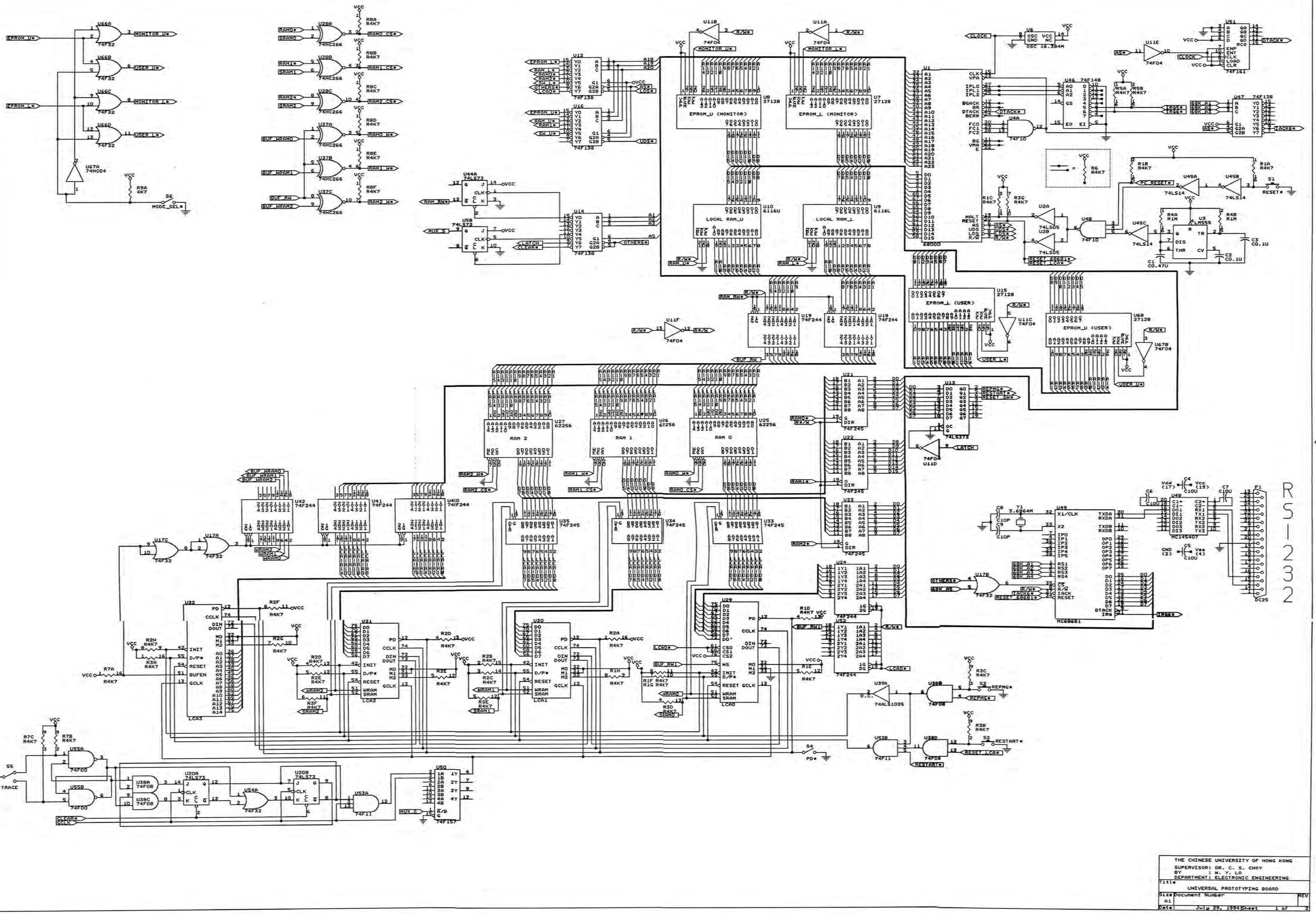
APPENDIX F

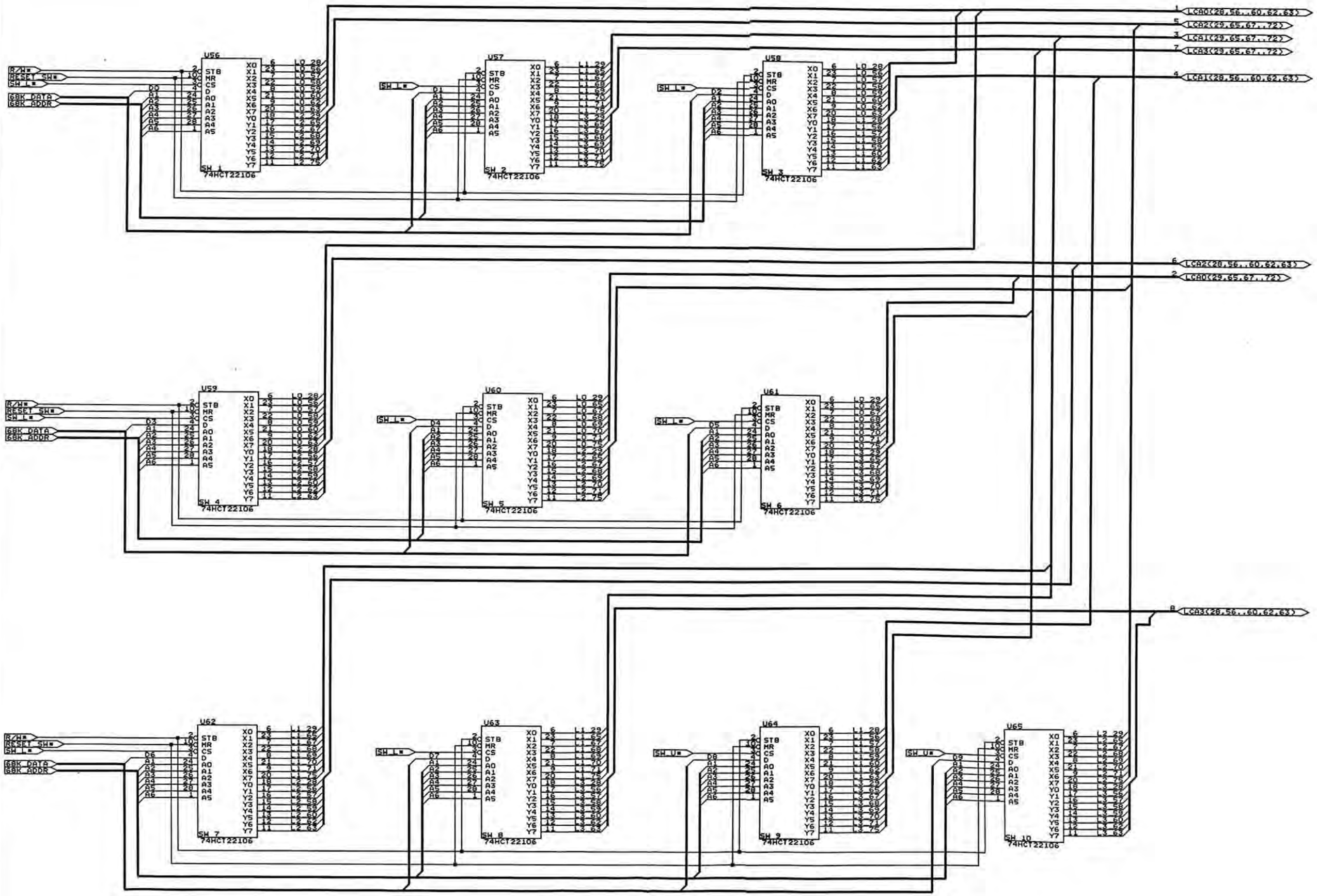
Memory map

Device	Address Space ($)	Notes
EPROM (monitor program)	000000 - 007FFF	user unaccessible
EPROM (user program)	040000 - 047FFF	user accessible
Local RAM	080000 - 080FFF	80000 - 807FF user unaccessible 80800 - 80FFF user accessible
RAM 0, 1, 2	0C0000 - 13FFFF	for memory-intensive designs
Programmable switch	140000 - 14007F	interconnection between FPGAs
MC68681	180000 - 18001F	serial interface connected PC
LCA 0	1C0000	downloading FPGAs config. data

APPENDIX G

Circuit diagram


EPROM_U#
EPROM_L#
MONITOR_U#
USER_U#
MONITOR_L#
USER_L#
MODE_SEL#
RAM0#
RAM1#
RAM2#
BUF_WRAM0
BUF_WRAM1
BUF_WRAM2
BUF_RW
EPROM_U (MONITOR)
EPROM_L (MONITOR)
LOCAL RAM_U
LOCAL RAM_L
EPROM_L (USER)
EPROM_U (USER)
RAM 2
RAM 1
RAM 0
CLOCK
OSC 16.384M
DTACK#
BERR
BGACK
BR
FC0
FC1
FC2
VMA
AS#
IPL0
IPL1
IPL2
MALT
RESET
UDS#
LDS#
R/W#
IACK1#
IACK2#
IACK3#
IACK4#
IACK5#
IACK6#
IACK7#
RESET#
PC_RESET#
OTHERS#
LOCAL RAM_U#
LOCAL RAM_L#
EPROM_U#
EPROM_L#
SW_U#
SW_L#
LOAD#
LATCH
CLEAR#
MUX_S
REPMC#
RESTART#
RESET_LCA#
PD#
TRACE
INIT
D/P#
RESET
BUFEN
GCLK
CCLK
DIN
DOUT
M0
M1
M2
WRAM
SRAM
LCA0
LCA1
LCA2
LCA3
RS232
THE CHINESE UNIVERSITY OF HONG KONG
SUPERVISOR: DR. C. S. CHOY
BY : W. Y. LO
DEPARTMENT: ELECTRONIC ENGINEERING
Title
UNIVERSAL PROTOTYPING BOARD
Size A1
Document Number
REV
Date: July 29, 1994 Sheet 1 of 2



THE CHINESE UNIVERSITY OF HONG KONG		
SUPERVISOR: DR. C. S. CHOY		
BY : W. Y. LO		
DEPARTMENT: ELECTRONIC ENGINEERING		
Title: INTERCONNECTABLE SWITCH		
Size: A1	Document Number:	REV:
Date: July 29, 1994	Sheet 2 of 2	

APPENDIX H

Program listings

Parti.h

```
/*****************************************************************************/
/* NET_LMT & CELL_LMT calculated from equations in comments are only suggestion. */
/* CLB_LMT,IO_LMT & CUT_LMT ARE fixed by emulation board hardware.           */
/* UTIL is suggested by Xilinx data book.                                    */
/*****************************************************************************/

#define SYM_NM_LEN 30                   /* max. symbol name length */
#define RCD_NM_LEN 8                    /* length of first entry of line in .map */
#define FILENM_LEN 13                   /* max. filename length */
#define CHAR_NO 200                     /* max. no. of charactor on line in .map */

#define NO_CP_ANA 0                     /* no critical path analysis */
#define FLIP_FLOP 1                     /* critical path analysis */
#define ALL_COMB 2                      /* treat all cells in CLB as combinational cells */

#define FPGA_LMT 4                      /* max. no. of FPGAs in emulation board */
#define CLB_LMT 100
#define REAL_CLB_LMT_D 80               /* real max. no to ensure 100% routability=CLB_LMT*0.8 */
#define CLB_IN_FPGA_D 20                /* max. no. of CLB in each FPGA=REAL_CLB_LMT*0.8/4 */
#define NET_LMT 200                     /* max. no. of NET incident on CLB in emulation board=CLB_LMT*2 */
#define CELL_LMT 400                    /* max. no. of CELL_LMT in CLB in emulation board=NET_LMT*2 */
#define IO_LMT 100                      /* max. no. of IO in emulation board (=CLB_LMT, for testing only) */
#define CUT_LMT 100                     /* max. no. of cut size between FPGAs */

#define NO_NEIBOR_CLB 20                /* max. no of CLB which has no neighbour CLB */

#define INIT_CLB_LMT_D 64               /* inital max. no. of CLBs for partition=CLB_LMT*0.8*0.8 */
#define INIT_CLB_IN_FPGA_D 20           /* initial max. no. of CLB in each FPGA for initial partitioning */
                                        /*   = CLB_LMT*UTIL*0.8/4 */

#define SYS_LMT 50
#define PROG_PIN_LMT 50                 /* PROG_LMT*total_path_no/2 */
#define LCA_IO_LMT 100

struct BLOCK
{
  char blk_name[SYM_NM_LEN];
  int fpga_id, final_fpga_id;
  unsigned locked;
  struct LIST *sig_first;
  struct DL_LIST *gain_bukt[3];
};

struct SIGNAL
{
  char sig_name[SYM_NM_LEN];
  unsigned ip_net, op_net, bi_net, connected;
  struct LIST *blk_first;
};

struct LIST
{
  unsigned lst_id, lst_type;
  char lst_name[SYM_NM_LEN];
  struct LIST *next_lst;
};

struct CELL
{
  char cell_name[SYM_NM_LEN], cell_type[SYM_NM_LEN], op_net_name[SYM_NM_LEN];
  unsigned ip_cell, op_cell, d_ff;
  int clb_id;
  struct SUCCESSOR *suc_cell_first, *in_path;
};

struct SUCCESSOR
{
  unsigned suc_cell_id;
  struct SUCCESSOR *next_suc_cell;
};

struct HEAD
{
  struct SUCCESSOR *first, *last;
};

struct DL_LIST
{
  int mag;
  struct DL_LIST *prevs, *next;
};

struct NO_NEIBOR
{
  char name[SYM_NM_LEN];
  int fpga_id;
```

```
  struct LIST *net_first;
};

struct IOB
{
  unsigned fpga_id, iob_type, pin_id;
  char net_name[SYM_NM_LEN];
};

struct IO_NET
{
  char net_name[SYM_NM_LEN], no_neibor_name[SYM_NM_LEN];
  unsigned net_id, i, o, pin_id, fpga_id, lca_con_no;
  int clb;
  struct IO_NET *next_io_net;
};

struct SYS_INTF
{
  unsigned pin[SYS_LMT], index;
};

struct CON
{
  unsigned lca_con_no, lca_id[4], op_clb;
};

struct LCA_IO
{
  unsigned lca_con_no, adj_lca, bd_intf;
};

struct PRI_GATE
{
  char gate[SYM_NM_LEN];
  unsigned gate_delay;
};

struct SW
{
  unsigned lca_con_no, sw_id, path_id[2], index[2];
};
```

Parti.c

```
/*********************************************************************/
/* cp_ana = 0: no critical path analysis.                            */
/*          1: critical path analysis.                               */
/*          2: critical path analysis, treat all cells in CLBs as    */
/*             combinational cells in sequectial circuit.            */
/*********************************************************************/

#include<stdio.h>
#include"parti.h"

extern unsigned cp_ana;

FILE *f_ptr;

main(argc, argv)
int argc;
char *argv[];
{
  GetCmd(argc, argv);                    /* get command line */
  MakeList();                            /* make linked list for partition */
  switch(cp_ana)
  {
    case 0:                              /* no critical path analysis */
      Partition(0);                      /* partitioning */
      break;
    case 1:
      CPAna(1);                          /* normal critical path analysis */
      Partition(1);                      /* partitioning */
      break;
    case 2:
      CPAna(0);                          /* combinational critical path analysis */
      Partition(1);                      /* partitioning */
      break;
  }
  AssignIO();                            /* IO assignment */
}
```

Getcmd.c

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include"parti.h"

extern FILE *f_ptr;

unsigned cp_ana, bal_parti, TtlFpgaNo;
unsigned REAL_CLB_LMT, CLB_IN_FPGA, INIT_CLB_LMT, INIT_CLB_IN_FPGA;
float UTIL, LOW_BAL_LMT, HIGH_BAL_LMT;
char infile[FILENM_LEN];
char *c_ptr, *strchr(), *strcat();

unsigned GetCmd(argc, argv)
int argc;
char *argv[];
{
  int key;
  unsigned i, cpchked;
  unsigned char para;

  bal_parti=0;                                  /* initialization */
  cpchked=0;                                    /* initialization */
  cp_ana=NO_CP_ANA;                             /* default: no critical path analysis */
  UTIL=0.5;                                     /*        : utilization of CLB in FPGA = 0.5 */
  TtlFpgaNo=4;                                  /*        : no. of FPGAs used = 4 */
  if(argc==1)
  {
    printf("\nEnter the file[.map] for partitioning...\n");
    scanf("%s", infile);
    ChkExt(infile);
    do
    {
      printf("\nCritical path analysis (y/n)? ");
      getchar();
      key=getchar();
    }
    while(key!='y' && key!='Y' && key!='n' && key!='N');
    if(key=='y' || key=='Y')
    {
      do
      {
          printf("\nNormal critical analysis (y/n)? ");
          getchar();
          key=getchar();
      }
      while(key!='y' && key!='Y' && key!='n' && key!='N');
      if(key=='y' || key=='Y')
          cp_ana=FLIP_FLOP;
      if(key=='n' || key=='N')
          cp_ana=ALL_COMB;
    }
    do
    {
      printf("\nBalanced partitioning (y/n)? ");
      getchar();
      key=getchar();
    }
    while(key!='y' && key!='Y' && key!='n' && key!='N');
    if(key=='y' || key=='Y')
    {
      bal_parti=1;
      do
      {
        printf("\nHow many FPGAs used (2-4)? ");
        scanf("%d", &TtlFpgaNo);
      }
      while(TtlFpgaNo<2 || TtlFpgaNo>4);
    }
    do
    {
      printf("\nUtilization of FPGA (0.1-0.8)? ");
      scanf("%f", &UTIL);
    }
    while(UTIL<0.1000000 || UTIL>0.80001);
  }
  else if(argc==2)
    ChkExt(argv[1]);
  else
  {
    ChkExt(argv[1]);
    for(i=2; i<argc; i++)
    {
      if(*argv[i]!='-')
          WrongPara();
      switch(*(argv[i]+1))
      {
          case 'c':
          case 'C':
            if(!cpchked)
            {
              cp_ana=ALL_COMB;
              cpchked=1;
```

```
          }
          else
          {
            printf("\nPlease specify clearly which option of critical path analysis!\n");
            exit(0);
          }
          if(*(argv[i]+2)!='\0')
            WrongPara();
          break;
        case 'f':
        case 'F':
          if(!cpchked)
          {
            cp_ana=FLIP_FLOP;
            cpchked=1;
          }
          else
          {
            printf("\nPlease specify clearly which option of critical path analysis!\n");
            exit(0);
          }
          if(*(argv[i]+2)!='\0')
            WrongPara();
          break;
        case 'b':
        case 'B':
          bal_parti=1;
          ChkBalParti(argv[i]);
          break;
        case 'u':
        case 'U':
          ChkUtil(argv[i]);
          break;
        default:
          WrongPara();
          break;
      }
    }
  }
  if(argc>5)
  {
    printf("\nToo many parameters entered!\n");
    help();
  }
  REAL_CLB_LMT=(unsigned)(CLB_LMT/4*TtlFpgaNo*UTIL);
  REAL_CLB_LMT=(unsigned)(REAL_CLB_LMT/TtlFpgaNo*TtlFpgaNo);
  CLB_IN_FPGA=(unsigned)(REAL_CLB_LMT/TtlFpgaNo);
  INIT_CLB_LMT=(unsigned)(REAL_CLB_LMT*0.8);
  INIT_CLB_IN_FPGA=(unsigned)(INIT_CLB_LMT/TtlFpgaNo);
  LOW_BAL_LMT=(float)1/(float)TtlFpgaNo-0.07;
  HIGH_BAL_LMT=(float)1/(float)TtlFpgaNo+0.07;
}

ChkExt(filename)                                    /* check file extension (.map) */
char filename[FILENM_LEN];
{
  strcpy(infile, filename);
  c_ptr=strchr(infile, '.');
  if(!c_ptr)
    strcat(infile, ".map");
  else
  {
    if(strcmp(c_ptr, ".map") && strcmp(c_ptr, ".MAP"))
    {
      printf("\nIncorrect file extension!\n");
      exit(0);
    }
  }
  f_ptr=fopen(infile, "r");
  if(!f_ptr)
  {
    printf("\nFile %s not found!\n", infile);
    exit(0);
  }
}

help()
{
  printf("\n");
  printf("\nHARDWARE EMULATION BOARD -- PARTITIONING");
  printf("\n============================================================");
  printf("\nCommand: ");
  printf("\nparti <filename[.map]> [-<para> [-<para> [-<para>]]] <ENTER>");
  printf("\n");
  printf("\nParameter:");
  printf("\nf        : critical path analysis.");
  printf("\nc        : critical path analysis, treat all cells in CLBs as ");
  printf("\n           combinational cells");
  printf("\nb[/no.]  : balanced partitioning into [no.] of FPGA.");
  printf("\nu[/ratio]: utilization ratio (e.g. 2 => ratio=0.2) of FPGA.");
  printf("\n============================================================");
  printf("\n");
  printf("\ndefault: no critical path analysis, not balanced partitioning & ");
  printf("\n         utilization ratio = 0.6.");
  printf("\n");
  exit(0);
```

```
}

WrongPara()
{
  printf("\nWrong parameter entered!\n");
  help();
}

ChkBalParti(temp)
char *temp;
{
  if(*(temp+2)=='\0')
    TtlFpgaNo=4;
  else
  {
    if(*(temp+2)!='/')
      WrongPara();
    switch(*(temp+3))
    {
      case '2':
          TtlFpgaNo=2;
          break;
      case '3':
          TtlFpgaNo=3;
          break;
      case '4':
          TtlFpgaNo=4;
          break;
      default:
          WrongPara();
    }
    if(*(temp+4)!='\0')
      WrongPara();
  }
}

ChkUtil(temp)
char *temp;
{
  char temp1;

  if(*(temp+2)=='\0')
    UTIL=0.8;
  else
  {
    if(*(temp+2)!='/')
    WrongPara();
    temp1=*(temp+3);
    UTIL=(float)(atoi(&temp1))/10;
    if(UTIL<0.1 || UTIL>0.80001)
      WrongPara();
    if(*(temp+4)!='\0')
      WrongPara();
  }
}
```

Makelist.c

```
/*****************************************************************************/
/* clb_id in cell[] = -1: cell embedded in CLB which has no neighbour CLB. */
/* no_neibor_clb[].fpga_id = -1: default value, CLB which has no neighbour */
/*                               CLBs not assigned to any FPGA.              */
/* lst_type in net_list = 1: input net.                                      */
/*                      = 0: output net.                                     */
/*                      = 2: bidirectional net.                              */
/*****************************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include"parti.h"

extern FILE *f_ptr;
extern unsigned bal_parti, CLB_IN_FPGA, REAL_CLB_LMT, TtlFpgaNo, cp_ana;

char line[CHAR_NO], rcd_nm[RCD_NM_LEN], sym_nm[SYM_NM_LEN], pin_nm[3];
char sym_type[SYM_NM_LEN], entry_nm[RCD_NM_LEN], temp_sig_nm[SYM_NM_LEN];
char pin_type[2], sig_nm[SYM_NM_LEN];
unsigned net_no, clb_no, cell_no, same_net_nm, temp_net_no, cell_count;
unsigned non_inv_cell, inv_cell_no, total_cell_no, total_clb_no;
unsigned total_net_no, TtlNoClb, MaxPin, fb_net, misc, blk_nm;
long cur_f_ptr;
struct LIST *net_list, *clb_list;
struct BLOCK clb[REAL_CLB_LMT_D];
struct SIGNAL net[NET_LMT];
struct CELL cell[CELL_LMT];
struct SUCCESSOR *suc_cell_list;
struct IO_NET *head_io_net, *io_net;
struct NO_NEIBOR no_neibor_clb[NO_NEIBOR_CLB];

void MakeList()                                 /* make lists for partitioning */
{
  unsigned count, io_no;

  printf("\nMaking netlists...\n");
  ChkLCAFile();                                 /* check 1st & 2nd lines of .map file format */
  misc=0;                                       /* initialization */
  clb_no=0;                                     /* initialization */
  net_no=0;                                     /* initialization */
  cell_no=0;                                    /* initialization */
  io_no=0;                                      /* initialization */
  MaxPin=0;                                     /* initialization */
  blk_nm=2;                                     /* initialization */
  strcpy(entry_nm, "SYM,");
  SkipLines(entry_nm, 4, f_ptr);                /* skips line until 1st record encountered */
  sscanf(line, "%s %s %s", rcd_nm, sym_nm, sym_type);
  while(strncmp(sym_type, "CLB", 3) && strncmp(sym_type, "IOB", 3))
  {
    MiscRcd();
    misc=0;
  }
  while(strncmp(sym_type, "IOB", 3)==(int)NULL)   /* count IOB records */
  {
    if(blk_nm==2)
    {
      if(strstr(line, "BLKNM="))                /* check if 'BLKNM=<block name>' exists in SYM records */
          blk_nm=1;
      else
          blk_nm=0;
    }
    strcpy(entry_nm, "EXT,");
    SkipLines(entry_nm, 4, f_ptr);
    io_no++;
    if(io_no+1>IO_LMT)                          /* check too many IO in cirkt. */
    {
      printf("\nCircuit too many IO to fit in emulation board!");
      exit(0);
    };
    cur_f_ptr=ftell(f_ptr);
    ReadSYM();
  };
  while(!strncmp(line, "EXT,", 4))
  {
    cur_f_ptr=ftell(f_ptr);
    ReadSYM();
  }
  while(strncmp(sym_type, "CLB", 3)==(int)NULL    /* CLB records */
        && strncmp(line, "EOF", 3)!=(int)NULL)    /* and not end of file */
    CLBRcd();
  total_clb_no=clb_no;
  if(bal_parti==0 && total_clb_no<CLB_IN_FPGA)
  {
    printf("\nCircuit can be fit into ONE FPGA, no need to partition.");
    printf("\n");
    exit(0);
  }
  if(bal_parti==1 && total_clb_no<TtlFpgaNo)
  {
    printf("\nCircuit too small to fit into %d FPGAs, no need to partition.",
            TtlFpgaNo);
    printf("\n");
```

```
    exit(0);
  }
  total_net_no=net_no;
  for(count=0; count<total_clb_no; count++)
  {
    clb[count].fpga_id=-1;
    clb[count].final_fpga_id=-1;
  }
  IONetLst();                                   /* make list of io net */
  DelIONetArray();                              /* delete 1-cell net */
  if(cp_ana)
  {
    WriteInvCell();                             /* search for and write INV cells */
    SucCellLst();                               /* make successor cell list for critical path analysis */
  }
/*  printf("\nclb%d, net%d.", total_clb_no, total_net_no);*/
  DelNoNeiborCLB();                             /* delete CLB which has no neighbour CLB */
  for(count=0; count<NO_NEIBOR_CLB; count++)
    no_neibor_clb[count].fpga_id=-1;            /* inititalization */
/*  for(count=0; count<NO_NEIBOR_CLB; count++)
    printf("\nNo_neibor_clb[%d]=%s", count, no_neibor_clb[count].name);
  printf("\nclb:%d, net:%d.", total_clb_no, total_net_no);*/
}

ChkLCAFile()                                    /* check .map file format */
{
  fgets(line, CHAR_NO, f_ptr);
  if(strncmp(line, "LCANET,", 7)!=(int)NULL)    /* first line -- LCANET, <version> */
    NotLCA();                                   /* not .map file format */
  fgets(line, CHAR_NO, f_ptr);
  if(strncmp(line, "PROG,", 5)!=(int)NULL)      /* second line -- PROG, <prog. name>, <version>... */
    NotLCA();                                   /* not .map file format */
}

NotLCA()                                        /* not .map file format */
{
  printf("\nNot [.map] file format!");
  exit(0);
}

MiscRcd()
{
  misc=1;
  CLBRcd();
}

SkipLines(entry_nm, str_len, t_ptr)             /* skip line until record of entry_nm encountered */
char *entry_nm;                                 /* and read entries of 1st line in record */
unsigned str_len;
FILE *t_ptr;
{
  while(strncmp(line, entry_nm, str_len)!=(int)NULL)
    fgets(line, CHAR_NO, t_ptr);
}

ReadSYM()
{
  fgets(line, CHAR_NO, f_ptr);
  sscanf(line, "%s %s %s", rcd_nm, sym_nm, sym_type);
}

CLBRcd()
{
  unsigned clb_count;
  struct LIST *temp_net_list, *temp;

  strcpy(clb[clb_no].blk_name, sym_nm);         /* write CLB block name */
  strcpy(entry_nm, "PIN,");                     /* skip all CFG lines */
  SkipLines(entry_nm, 4, f_ptr);
  sscanf(line, "%s %s %s %s", rcd_nm, pin_nm, pin_type, sig_nm);
  while(strncmp(line, "PIN,", 4)==(int)NULL)    /* make list of NETs for each CLB */
  {
    same_net_nm=0;
    clb_count=0;
    do                                          /* search for same net name */
    {
      temp_net_list=clb[clb_count].sig_first;
      while(temp_net_list && strcmp(temp_net_list->lst_name, sig_nm))
          temp_net_list=temp_net_list->next_lst;
      if(temp_net_list && !strcmp(temp_net_list->lst_name, sig_nm))
          SameNetNm(temp_net_list->lst_id);
      clb_count++;
    }
    while(!same_net_nm && clb[clb_count-1].sig_first);
    fb_net=0;
    temp_net_list=clb[clb_no].sig_first;
    while(temp_net_list && strcmp(temp_net_list->lst_name, sig_nm))
      temp_net_list=temp_net_list->next_lst;
    if(temp_net_list && !strcmp(temp_net_list->lst_name, sig_nm))
      fb_net=1;
    if(!fb_net)
    {
      net_list=(struct LIST *)calloc(1, (unsigned)(sizeof(struct LIST)));
      ChkMemLst(net_list);                      /* check memory space */
      net_list->lst_id=net_no;                  /* write net id. no. in net_list */
      strcpy(net_list->lst_name, sig_nm);       /* write net name in net_list */
```

```
    if(!strncmp(pin_type, "I", 1))
        net_list->lst_type=1;                    /* input net */
    else
        net_list->lst_type=0;                            /* output net */
    if(!clb[clb_no].sig_first)                   /* add net to net_list */
    {                                            /* empty list */
        net_list->next_lst=(struct LIST *)NULL;
        AddlNet();
    }
    else
    {
        net_list->next_lst=clb[clb_no].sig_first;
        AddlNet();
    }
  }
  else
  {
    temp_net_list->lst_type=2;
    net[temp_net_list->lst_id].ip_net=0;
    net[temp_net_list->lst_id].op_net=0;
    net[temp_net_list->lst_id].bi_net=1;
    temp=net[temp_net_list->lst_id].blk_first;
    while(temp && temp->lst_type!=2)
    {
        if(temp->lst_id==clb_no)
          temp->lst_type=2;
        temp=temp->next_lst;
    }
  }
  if(same_net_nm==1)
    net_no=temp_net_no;
  else
  {
    net_no++;
    if(net_no>NET_LMT)
    {
        printf("\nCircuit too many nets to fit in emulation board!");
        printf("\n");
        printf("\nType \"parti -h <ENTER>\" for help...");
        printf("\n");
        exit(0);
    }
  }
  ReadPIN();
 };
 clb_no++;
 if(clb_no>REAL_CLB_LMT-1)                       /* check too large cirkt. */
 {
   if(TtlFpgaNo==4)
     printf("\nCircuit too large to fit in emulation board.");
   else
     printf("\nCircuit too large to fit in only %d FPGAs.", TtlFpgaNo);
   exit(0);
 }
 if(!misc)
 {
   if(cp_ana)
     WriteCell();                                /* write cells for critical path ananlysis */
   else if(!cp_ana)
   {
     strcpy(entry_nm, "ENDMOD");                 /* skip all MODEL records */
     SkipLines(entry_nm, 6, f_ptr);
   }
   fgets(line, CHAR_NO, f_ptr);
 }
 ReadSYM();
}

ChkMemLst(list)                                  /* check for sufficient memory allocation */
struct LIST *list;
{
  if(list==(struct LIST *)NULL)
  {
    printf("\nOut of memory! (making list)");
    exit(0);
  }
}

SameNetNm(cell_id)                               /* search for same NET name */
unsigned cell_id;
{
  temp_net_no=net_no;
  net_no=cell_id;
  same_net_nm=1;
}

AddlNet()                                        /* add 1 net to net_list */
{
  clb[clb_no].sig_first=net_list;
  MakeCLBLst();                                  /* makes list of CLBs for each net */
}

MakeCLBLst()                                     /* make list of CLBs for each net */
{
  while(net[net_no].blk_first==(struct LIST *)NULL ||
```

```
        net[net_no].blk_first->lst_id!=clb_no)
  {
    clb_list=(struct LIST *)calloc(1, (unsigned)(sizeof(struct LIST)));
    ChkMemLst(clb_list);                          /* check memory space */
    clb_list->lst_id=clb_no;
    strcpy(clb_list->lst_name, clb[clb_no].blk_name);
    strcpy(net[net_no].sig_name, sig_nm);         /* write net name */
    if(strcmp(pin_type, "I,")==(int)NULL)         /* write pin type */
      net[net_no].ip_net=1;
    if(strcmp(pin_type, "O,")==(int)NULL)
      net[net_no].op_net=1;
    clb_list->lst_type=net_list->lst_type;
    if(!net[net_no].blk_first)                    /* empty list */
    {
      net[net_no].blk_first=clb_list;
      clb_list->next_lst=(struct LIST *)NULL;
    }
    else
    {
      clb_list->next_lst=net[net_no].blk_first;
      net[net_no].blk_first=clb_list;
    }
  }
}

ReadPIN()
{
  fgets(line, CHAR_NO, f_ptr);
  sscanf(line, "%s %s %s %s", rcd_nm, pin_nm, pin_type, sig_nm);
}

WriteCell()                                       /* read in and write cell name for critical path analysis */
{
  ReadSYM();
  while(strncmp(line, "ENDMOD", 6)!=(int)NULL)
  {
    strcpy(cell[cell_no].cell_name, sym_nm);      /* writes cell name */
    strcpy(cell[cell_no].cell_type, sym_type);    /* writes cell type */
    if(cell[cell_no].clb_id!=-1)
      cell[cell_no].clb_id=clb_no-1;              /* writes CLB no in which the cell resides */
    if(strcmp(sym_type, "DFF")==(int)NULL)        /* D flip flop cell */
      cell[cell_no].d_ff=1;
    do
    {
      ReadPIN();                                  /* skip lines until output pin encountered */
    }
    while(strcmp(pin_type, "O,")!=(int)NULL);
    strcpy(cell[cell_no].op_net_name, sig_nm);
    cell_no++;
    TestCellLmt();
    strcpy(entry_nm, "END");                      /* skip lines until end of current SYM */
    SkipLines(entry_nm, 3, f_ptr);
    ReadSYM();
  };
}

TestCellLmt()
{
  if(cell_no>CELL_LMT)
  {
    printf("\nCircuit too many cells in CLB to fit in emulation board!");
    printf("\n");
    printf("\nType \"parti -h <ENTER>\" for help...");
    printf("\n");
    exit(0);
  }
}

IONetLst()                                        /* find net with only 1 cell incident on */
{
  struct LIST *temp;
  unsigned i, o;

  net_no=0;
  while(net[net_no].sig_name[0])
  {
    i=0;
    o=0;
    temp=net[net_no].blk_first;
    if(!net[net_no].blk_first->next_lst && !net[net_no].bi_net)
    {
      io_net=(struct IO_NET *)calloc(1, (unsigned)(sizeof(struct IO_NET)));
      ChkMemIONet(io_net);
      io_net->net_id=net_no;
      io_net->clb=(int)(net[net_no].blk_first->lst_id);
      io_net->lca_con_no=1;
      strcpy(io_net->net_name, net[net_no].sig_name);
      if(net[net_no].ip_net==1)
          io_net->i=1;
      if(net[net_no].op_net==1)
          io_net->o=1;
      if(!head_io_net)
      {
          head_io_net=io_net;
          io_net->next_io_net=(struct IO_NET *)NULL;
      }
```

```
    else
    {
        io_net->next_io_net=head_io_net;
        head_io_net=io_net;
    }
  }
  else
  {
    i=0;
    o=0;
    temp=net[net_no].blk_first;
    while(temp)
    {
        if(temp->lst_type==1)
          i=1;
        if(temp->lst_type==0)
          o=1;
        temp=temp->next_lst;
    }
    if((i==1 && o==0) || (i==0 && o==1))
    {
        io_net=(struct IO_NET *)calloc(1, (unsigned)(sizeof(struct IO_NET)));
        ChkMemIONet(io_net);
        io_net->net_id=net_no;
        io_net->clb=(int)(net[net_no].blk_first->lst_id);
        io_net->lca_con_no=0;
        strcpy(io_net->net_name, net[net_no].sig_name);
        if(i==1 && o==0)
          io_net->i=1;
        if(i==0 && o==1)
          io_net->o=1;
        if(!head_io_net)                          /* empty list */
        {
          head_io_net=io_net;
          io_net->next_io_net=(struct IO_NET *)NULL;
        }
        else
        {
          io_net->next_io_net=head_io_net;
          head_io_net=io_net;
        }
    }
  }
  net_no++;
 }
}

ChkMemIONet(ionet)                                /* check for sufficient memory allocation */
struct IO_NET *ionet;
{
 if(ionet==(struct IO_NET *)NULL)
 {
   printf("\nOut of memory! (making io net list)");
   exit(0);
 }
}

DelIONetArray()                                   /* search & delete io net in net array */
{
 unsigned i, j;
 int count;

 j=0;
 for(count=0; count<total_net_no; count++)
 {
   if(!net[count].blk_first->next_lst)
   {
     DelIONetLst(count+j);                        /* delete io net in net-list */
     j++;
     if(count!=total_net_no-1)
     {
         for(i=count+1; i<total_net_no; i++)
         {
           strcpy(net[i-1].sig_name, net[i].sig_name);
           net[i-1].ip_net=net[i].ip_net;
           net[i-1].op_net=net[i].op_net;
           net[i-1].bi_net=net[i].bi_net;
           net[i-1].blk_first=net[i].blk_first;
         }
         count--;
     };
     total_net_no--;
   };
 }
 ChangeIONetId();                                 /* renumber net id. after deleting io net */
}

DelIONetLst(io_net_id)                           /* search & delete io net in net-list */
unsigned io_net_id;
{
 unsigned count1;
 struct LIST **temp, *temp1;

 for(count1=0; count1<total_clb_no; count1++)
 {
   if(clb[count1].sig_first!=(struct LIST *)NULL)
```

```
    {
      temp=&(clb[count1].sig_first);
      while((*temp)->next_lst && (*temp)->lst_id!=io_net_id)
        temp=&((*temp)->next_lst);
      if((*temp)->lst_id==io_net_id)              /* io net encountered */
      {
        temp1=(*temp)->next_lst;                  /* delete io net */
        free((struct LIST *)(*temp));
        *temp=temp1;
      };
    };
  }
}

ChangeIONetId()                                   /* change net id. in net-list & in io-net-list */
{
  unsigned count, count1;
  int i;
  struct LIST *temp;
  struct IO_NET *temp1;

  for(count=0; count<total_net_no; count++)
  {
    count1=0;
    while(count1!=total_clb_no)
    {
      if(clb[count1].sig_first)
      {
          i=0;
        temp=clb[count1].sig_first;
          while(temp)
          {
            if(!strcmp(temp->lst_name, net[count].sig_name))
              temp->lst_id=count;
            i++;
            temp=temp->next_lst;
          }
          if(i>(int)MaxPin)                       /* max. gain = max. no. of nets on CLB */
            MaxPin=(unsigned)i;
      }
      count1++;
    }
    temp1=head_io_net;
    while(temp1 && temp1->net_id!=count)
    {
      if(!strcmp(temp1->net_name, net[count].sig_name))
          temp1->net_id=count;
      temp1=temp1->next_io_net;
    }
  }
}

WriteInvCell()                                    /* read in and write INV cell name */
{
  non_inv_cell=cell_no;
  clb_no=0;
  fseek(f_ptr, cur_f_ptr, 0);
  ReadSYM();
  while(strncmp(line, "EOF", 3))                  /* search for and write INV cell */
  {
    strcpy(entry_nm, "MODEL");
    SkipLines(entry_nm, 5, f_ptr);
    fgets(line, CHAR_NO, f_ptr);
    while(strncmp(line, "ENDMOD", 6))
    {
      ReadPIN();
      if(!strcmp(pin_type, "O,"))
          ReadPIN();
      do
      {
          if(strstr(line, ", INV"))
            InvCell();                            /* INV cell encountered */
          ReadPIN();
      }
      while(!strcmp(pin_type, "I,") && strncmp(line, "END", 3));
      strcpy(entry_nm, "END");                    /* Skip lines until end of current SYM */
      SkipLines(entry_nm, 3, f_ptr);
      ReadSYM();
    };
    clb_no++;
    fgets(line, CHAR_NO, f_ptr);
    ReadSYM();
  };
  total_cell_no=cell_no;
}

InvCell()                                         /* write INV cell */
{
  strcpy(temp_sig_nm, sig_nm);
  strcat(temp_sig_nm, "NOT");
  cell_count=non_inv_cell;
  while(strcmp(cell[cell_count].cell_name, temp_sig_nm) &&
          cell[cell_count].cell_name[0])
    cell_count++;
  if(!cell[cell_count].cell_name[0])
  {
```

```
    strcpy(cell[cell_no].cell_type, "INV");
    strcpy(cell[cell_no].cell_name, sig_nm);
    strcat(cell[cell_no].cell_name, "NOT");
    strcpy(cell[cell_no].op_net_name, sig_nm);
    strcat(cell[cell_no].op_net_name, "INV");
    cell[cell_no].clb_id=clb_no;
    cell_no++;
    TestCellLmt();
  };
}

SucCellLst()                                        /* make list of successor cell for critical path
analysis */
{
  fseek(f_ptr, cur_f_ptr, 0);
  ReadSYM();
  cell_no=0;
  while(strncmp(line, "EOF", 3))
  {
    strcpy(entry_nm, "MODEL");
    SkipLines(entry_nm, 5, f_ptr);
    fgets(line, CHAR_NO, f_ptr);
    while(strncmp(line, "ENDMOD", 6))
    {
      ReadPIN();
      while(!strncmp(line, "PIN,", 4))
      {
          if(!strcmp(pin_type, "O,"))
          {
            ChkIOCell();                            /* check if the cell connected to output pin is io cell
*/
            fgets(line, CHAR_NO, f_ptr);
            if(strncmp(line, "PIN,", 4))
              fgets(line, CHAR_NO, f_ptr);
            else
              sscanf(line, "%s %s %s %s", rcd_nm, pin_nm, pin_type, sig_nm);
          }
          else
          {
            FindSucCell();
            if(strncmp(line, "PIN,", 4))
              fgets(line, CHAR_NO, f_ptr);
            else
              sscanf(line, "%s %s %s %s", rcd_nm, pin_nm, pin_type, sig_nm);
          }
      }
      sscanf(line, "%s %s %s", rcd_nm, sym_nm, sym_type);
      cell_no++;
    }
    fgets(line, CHAR_NO, f_ptr);
    ReadSYM();
  }
  fgets(line, CHAR_NO, f_ptr);
  ReadSYM();
}

DelNoNeiborCLB()                                    /* record & delete no-net-on-CLBs */
{
  unsigned i;
  int count;
  struct IO_NET *temp;
  struct LIST *temp1;

  for(count=0; count<total_clb_no; count++)
  {
    if(clb[count].sig_first==(struct LIST *)NULL)
    {
      ChangeBlkId(count);                           /* change CLB id. in cell & in io net */
      strcpy(no_neibor_clb[TtlNoClb].name, clb[count].blk_name);
      TtlNoClb++;
      if(TtlNoClb>NO_NEIBOR_CLB)
      {
          printf("\nToo many CLBs which has no neighbour CLBs!\n");
          printf("\nType \"parti -h <ENTER>\" for help...");
          printf("\n");
          exit(0);
      }
      if(count!=total_clb_no-1)                     /* shift CLB in CLB array after deleting */
      {
          for(i=count+1; i<total_clb_no; i++)
          {
            strcpy(clb[i-1].blk_name, clb[i].blk_name);
            clb[i-1].sig_first=clb[i].sig_first;
            temp=head_io_net;
            while(temp)                             /* change CLB id. in IO net */
            {
              if(temp->clb==i)
                temp->clb=i-1;
              temp=temp->next_io_net;
            }
          }
          count--;
      };
      total_clb_no--;
      for(i=0; i<total_net_no; i++)                 /* change CLB id. in CLB list */
      {
```

```
        temp1=net[i].blk_first;
        while(temp1)
        {
          if(temp1->lst_id>=count+1)
            (temp1->lst_id)--;
          temp1=temp1->next_lst;
        }
      }
    };
  }
}

ChangeBlkId(j)
unsigned j;
{
  unsigned count1;
  struct IO_NET *temp;

  for(count1=0; count1<total_cell_no; count1++)
  {
    if(cell[count1].clb_id==j)
      cell[count1].clb_id=-1;
  }
  temp=head_io_net;
  while(temp)
  {
    if(temp->clb==j)
    {
      temp->clb=-1;
      strcpy(temp->no_neibor_name, clb[j].blk_name);
    }
    temp=temp->next_io_net;
  }
}

ChkIOCell()
{
  struct IO_NET *temp_io_net;

  temp_io_net=head_io_net;
  while(temp_io_net->next_io_net && strcmp(temp_io_net->net_name, sig_nm))
    temp_io_net=temp_io_net->next_io_net;
  if(!strcmp(temp_io_net->net_name, sig_nm))
  {
    if(!strstr(line, ", INV"))
    {
      if(temp_io_net->i)                              /* system input net */
          cell[cell_no].ip_cell=1;
      else if(temp_io_net->o)                         /* system output net */
          cell[cell_no].op_cell=1;
      else
      {
          printf("\nError! (IO_net)\n");
          exit(0);
      }
    }
    else
    {
      if(temp_io_net->i && cell[inv_cell_no].ip_cell==0)
          cell[inv_cell_no].ip_cell=1;
      if(temp_io_net->o && cell[inv_cell_no].op_cell==0)
          cell[inv_cell_no].op_cell=1;
    }
  }
}

FindSucCell()
{
  do
  {
    if(!strstr(line, ", INV"))
    {
      ChkIOCell();
      cell_count=0;
      while(strcmp(cell[cell_count].op_net_name, sig_nm) &&
              cell_count<non_inv_cell)
          cell_count++;
      if(!strcmp(cell[cell_count].op_net_name, sig_nm))
          AddSucCell(&(cell[cell_count].suc_cell_first), cell_no);
    }
    else
    {
      strcpy(temp_sig_nm, sig_nm);
      strcat(temp_sig_nm, "INV");
      cell_count=non_inv_cell;
      while(cell_count<total_cell_no &&
              strcmp(cell[cell_count].op_net_name, temp_sig_nm))
          cell_count++;
      inv_cell_no=cell_count;
      ChkIOCell();                                    /* check if INV cell is io cell */
      AddSucCell(&(cell[cell_count].suc_cell_first), cell_no);
      cell_count=0;
      while(cell_count<non_inv_cell &&
              strcmp(cell[cell_count].op_net_name, sig_nm))
          cell_count++;
      if(!strcmp(cell[cell_count].op_net_name, sig_nm))
```

```
        if(!ChkSucCell(cell[cell_count].suc_cell_first, inv_cell_no))
          AddSucCell(&(cell[cell_count].suc_cell_first), inv_cell_no);
    };
    ReadPIN();
  }
  while(!strcmp(pin_type, "I,") && strncmp(line, "END", 3));
}

AddSucCell(first, suc_cell_no)                    /* adds successor cell */
struct SUCCESSOR **first;
unsigned suc_cell_no;
{
  suc_cell_list=(struct SUCCESSOR *)calloc(1, (unsigned)(sizeof(struct SUCCESSOR)));
  ChkMemSuc(suc_cell_list);
  suc_cell_list->suc_cell_id=suc_cell_no;
  if(*first==(struct SUCCESSOR *)NULL)
  {
    *first=suc_cell_list;
    suc_cell_list->next_suc_cell=(struct SUCCESSOR *)NULL;
  }
  else
  {
    suc_cell_list->next_suc_cell=*first;
    *first=suc_cell_list;
  };
}

ChkMemSuc(suc)
struct SUCCESSOR *suc;
{
  if(suc==(struct SUCCESSOR *)NULL)
  {
    printf("\nOut of memory! (making successor cell list)");
    exit(0);
  }
}

ChkSucCell(first, cell_id)                        /* check exist successor cell id. in list *first */
struct SUCCESSOR *first;
unsigned cell_id;
{
  struct SUCCESSOR *temp_suc_cell;
  temp_suc_cell=first;
  if(!temp_suc_cell)                              /* empty list */
    return(0);
  else
  {
    while(temp_suc_cell->suc_cell_id!=cell_id && temp_suc_cell->next_suc_cell)
      temp_suc_cell=temp_suc_cell->next_suc_cell;
    if(temp_suc_cell->suc_cell_id==cell_id)
      return(1);
    else
      return(0);
  };
}
```

Cpana.c

```
/*******************************************/
/* Path cp0 stores the final critical path. */
/* Path cp1 stores the current path.        */
/*******************************************/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include"parti.h"

#define ADD 1
#define SUB 0
#define PRI_GATE_LMT 30

extern unsigned total_cell_no, cell_count, cp_ana, total_clb_no;
extern char line[CHAR_NO];
extern struct BLOCK clb[REAL_CLB_LMT_D];
extern struct CELL cell[CELL_LMT];
extern struct SUCCESSOR *suc_cell_list;

unsigned cnt0, cnt1, flip_flop, total_gate_no, dff_delay, path_delay;
unsigned long max_delay[2];
FILE *rf_ptr;
struct SUCCESSOR *head_cp1, *cp[2], *suc_cell_ptr[CELL_LMT];
struct SUCCESSOR *head_stack, *stack_lst;
struct HEAD head_cp0;
struct PRI_GATE pri_gate[PRI_GATE_LMT];

CPAna(ff)                                       /* find critical path & lock CLB */
unsigned ff;
{
  printf("\nCritical pass analysis...\n");
  flip_flop=ff;
  max_delay[0]=0;                               /* initialization */
  CPInit();
  for(cell_count=0; cell_count<total_cell_no; cell_count++)
  {
    if(cell[cell_count].suc_cell_first)
      if(flip_flop==1)
      {
          if(cell[cell_count].ip_cell==1 || cell[cell_count].d_ff==1)
            FindNextPath();                     /* find all paths in cirkt. */
      }
      else
      {
          if(cell[cell_count].ip_cell==1)
            FindNextPath();
      }
  }
  printf("\nCritical path delay: %d units.\n", max_delay[0]);
}

CPInit()
{
  unsigned i;

  i=0;
  rf_ptr=fopen("delay.dat", "r");
  if(!rf_ptr)
  {
    printf("\nFile delay.dat not found!\n");
    exit(0);
  }
  fgets(line, CHAR_NO, rf_ptr);
  do
  {
    sscanf(line, "%s %d", pri_gate[i].gate, &(pri_gate[i].gate_delay));
    if(!strcmp(pri_gate[i].gate, "DFF"))
      dff_delay=pri_gate[i].gate_delay;
    else if(!strcmp(pri_gate[i].gate, "PATH"))
      path_delay=pri_gate[i].gate_delay;
    i++;
    if(i==PRI_GATE_LMT)
    {
      printf("\nError! Too much primitive gate type.\n");
      exit(0);
    }
    fgets(line, CHAR_NO, rf_ptr);
  }
  while(strlen(line)-1);
  total_gate_no=i;
  fclose(rf_ptr);
}

FindNextPath()                                  /* find all paths in cirkt. */
{
  unsigned count;

  max_delay[1]=0;
  AddPathLst(cell_count);
  for(count=0; count<total_cell_no; count++)    /* initialize pointers */
    suc_cell_ptr[count]=cell[count].suc_cell_first;
  do
```

```
{
  if(!head_stack)
    FindPath(cell_count);                          /* find all paths for each input cell */
  else
    FindPath(head_stack->suc_cell_id);
  if(flip_flop==1)
    if(cell[head_cp1->suc_cell_id].d_ff)
        UpdateDelay(SUB, dff_delay);
  DelShorterPath();                                /* delete shorter path after comparing current with stored
longest path */
  if(flip_flop==1)
    if(cell[head_cp1->suc_cell_id].d_ff)
        UpdateDelay(ADD, dff_delay);
  if(head_stack)
    DelSucCells(head_stack->suc_cell_id);
  else
    DeAllocPath(&(head_cp1), 1);
 }
 while(head_stack);
}

AddPathLst(cell_id)                                /* add each successor cell in path list */
unsigned cell_id;
{
 CalMaxDelay(ADD, cell_id);
 AddSucCell(&(head_cp1), cell_id);
 cell[cell_id].in_path=suc_cell_list;
}

CalMaxDelay(cal, cell_id)                          /* find cell type for calculating path delay */
unsigned cal, cell_id;
{
 unsigned i;

 i=0;                                              /* initialization */
 while(strcmp(cell[cell_id].cell_type, pri_gate[i].gate))
   i++;
 if(i<total_gate_no)
 {
   UpdateDelay(cal, pri_gate[i].gate_delay);
   UpdateDelay(cal, path_delay);
 }
 else
 {
   printf("\nElement %s not specified in file delay.dat.\n",
            cell[cell_id].cell_type);
   exit(0);
 }
}

UpdateDelay(cal, cell_delay)                       /* update path delay */
unsigned cal, cell_delay;
{
 if(cal==ADD)
   max_delay[1]=max_delay[1]+cell_delay;
 else
   max_delay[1]=max_delay[1]-cell_delay;
}

FindPath(count)                                    /* find whole path from ip. to op. */
unsigned count;
{
 if(count!=cell_count)                             /* check if count is output cell or d_ff */
 {
   if(flip_flop==1)
   {
     if(cell[count].d_ff==1 || cell[count].op_cell==1)
         return(0);
   }
   else
   {
     if(cell[count].op_cell==1)
         return(0);
   };
 };
 cnt0=count;
 if(cell[cnt0].suc_cell_first->next_suc_cell)
 {
   cnt1=suc_cell_ptr[cnt0]->suc_cell_id;
   suc_cell_ptr[cnt0]=suc_cell_ptr[cnt0]->next_suc_cell;
   if(suc_cell_ptr[cnt0] && !ChkSucCell(head_stack, cnt0))
     AddSucCell(&(head_stack), cnt0);              /* put cell in stack which has more than 2 successor cells */
   if(head_stack)
     if(!suc_cell_ptr[head_stack->suc_cell_id])/* delete cell in stack after analyzing all successor cells */
       DeAllocStackFirst();
 }
 else
 {
   cnt1=cell[cnt0].suc_cell_first->suc_cell_id;/* cnt1: successor cell */
 };
 if(!cell[cnt1].in_path)                           /* check feedback path */
   AddPathLst(cnt1);
 else
   return(0);                                      /* feedback path */
 cnt0=cnt1;
 FindPath(cnt0);
```

```
}

DeAllocStackFirst()                                  /* delete 1 cell in stack */
{
  struct SUCCESSOR *temp;
  unsigned k;

  k=head_stack->suc_cell_id;
  suc_cell_ptr[k]=cell[k].suc_cell_first;
  temp=head_stack->next_suc_cell;
  free((struct SUCCESSOR *)head_stack);
  head_stack=temp;
}

DelShorterPath()                                     /* delete shorter CP after comparing with currect CP */
{
  if(flip_flop==1)
  {
    if(cell[head_cp1->suc_cell_id].d_ff==0 &&
       cell[head_cp1->suc_cell_id].op_cell==0)
      return(0);                                     /* feedback path */
  }
  else
  {
    if(cell[head_cp1->suc_cell_id].op_cell==0)
      return(0);                                     /* feedback path */
  };
  if(max_delay[1]>max_delay[0])
  {
    DeAllocPath(&(head_cp0.first), 0);
    CopyPath();
    max_delay[0]=max_delay[1];
  };
}

DeAllocPath(first, x)                                /* delete path */
struct SUCCESSOR **first;
unsigned x;
{
  struct SUCCESSOR *temp;

  while(*first)
  {
    if(x)
      cell[(*first)->suc_cell_id].in_path=(struct SUCCESSOR *)NULL;
    temp=(*first)->next_suc_cell;
    free((struct SUCCESSOR *)(*first));
    *first=temp;
  };
}

CopyPath()                                           /* copy path 1 to path 0 */
{
  struct SUCCESSOR *temp, *temp_path_lst;

  temp=head_cp1;
  while(temp)
  {
    temp_path_lst=(struct SUCCESSOR *)calloc(1, (unsigned)(sizeof(struct SUCCESSOR)));
    ChkMemSuc(temp_path_lst);
    if(!head_cp0.first)
    {
      head_cp0.first=temp_path_lst;
      head_cp0.last=temp_path_lst;
    }
    else
    {
       head_cp0.last->next_suc_cell=temp_path_lst;
       head_cp0.last=temp_path_lst;
    };
    head_cp0.last->suc_cell_id=temp->suc_cell_id;
    temp=temp->next_suc_cell;
  };
}

DelSucCells(cell_id)                                 /* delete all one-successor-cells until */
unsigned cell_id;                                    /* which more than 2 successors encountered */
{
  struct SUCCESSOR *temp;

  while(head_cp1->suc_cell_id!=cell_id)
  {
    CalMaxDelay(SUB, head_cp1->suc_cell_id);
    cell[head_cp1->suc_cell_id].in_path=(struct SUCCESSOR *)NULL;
    temp=head_cp1->next_suc_cell;
    free((struct SUCCESSOR *)head_cp1);
    head_cp1=temp;
  }
}
```

Partitio.c

```
/*************************************************************************/
/* fpga_id in clb[i]=-1: default value, not assigned to any FPGA.        */
/* clb_id in cell[i]=-1: cell[i] is in CLB which has no neighbour CLB.   */
/* locked in cell[i]= 0: free cell.                                      */
/*                   = 1: locked cell, locked within a pass.             */
/*                   = 2: fixed cell, locked forever.                    */
/*************************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include"parti.h"

#define INC 1
#define DEC 0
#define CR 0x0d

extern unsigned total_clb_no, total_net_no, real_clb_lmt, total_cell_no;
extern unsigned total_no_net_clb, MaxPin, TtlNoClb, bal_parti, CLB_IN_FPGA;
extern unsigned INIT_CLB_LMT, REAL_CLB_LMT, INIT_CLB_IN_FPGA, TtlFpgaNo;
extern float LOW_BAL_LMT, HIGH_BAL_LMT;
extern struct BLOCK clb[REAL_CLB_LMT_D];
extern struct CELL cell[CELL_LMT];
extern struct SIGNAL net[NET_LMT];
extern struct HEAD head_cp0, *head_cp1, *suc_cell_ptr[CELL_LMT];
extern struct NO_NEIBOR no_neibor_clb[NO_NEIBOR_CLB];
extern struct IO_NET *head_io_net;

struct GAIN
{
  int clb_id, gain_value;
};

struct MOVE
{
  unsigned fm_fpga, to_fpga, base_clb;
  int move_gain;
  struct DL_LIST *best_move;
};

unsigned clb_cnt[FPGA_LMT], total_heap_no, min_cut_size;
unsigned cut_size, total_gain_no, clb_dist[NET_LMT][FPGA_LMT];
unsigned move_no, total_move_no, cnt, no_valid_move, final_cut, pass;
unsigned min_clb_no, max_clb_no, temp_clb_cnt[FPGA_LMT];
struct GAIN gain[FPGA_LMT][FPGA_LMT][CLB_IN_FPGA_D];
struct DL_LIST *gain_lst, gain_ptr[FPGA_LMT][FPGA_LMT];
struct DL_LIST *bucket[FPGA_LMT][FPGA_LMT][NET_LMT];
struct MOVE heap[12], move_rcd[REAL_CLB_LMT_D];

Partition(cp_ana)
unsigned cp_ana;
{
  unsigned count, count1;
  int x, y;
  float z;

  printf("\nPartitioning...\n");
  for(count=0; count<TtlFpgaNo; count++)          /* initialization */
    clb_cnt[count]=0;                             /* no. of CLBs currently in each FPGA */
  final_cut=CUT_LMT;                              /* initialization */
  min_cut_size=CUT_LMT;
  pass=0;                                         /* initialization */
  while(count<FPGA_LMT)                           /* initialization */
  {
    clb_cnt[count]=0;
    count++;
  }
  if(bal_parti)
  {
    z=(float)total_clb_no*(float)LOW_BAL_LMT;
    x=(int)z;
    if(z>(float)x)
      min_clb_no=(unsigned)x+1;
    else
      min_clb_no=(unsigned)x;
    z=(float)total_clb_no*(float)HIGH_BAL_LMT;
    max_clb_no=(unsigned)z;
    if(max_clb_no>CLB_IN_FPGA)
    {
      count=max_clb_no-CLB_IN_FPGA;
      max_clb_no=max_clb_no-count;
      min_clb_no=min_clb_no+count;
    }
  }
  else
  {
    if(total_clb_no>INIT_CLB_LMT)
      count=total_clb_no/(int)CLB_IN_FPGA;
    else
      count=total_clb_no/(int)INIT_CLB_IN_FPGA;
    if(count==FPGA_LMT)
      TtlFpgaNo=count;
    else
```

```
    TtlFpgaNo=count+1;
  }
  count=total_clb_no/TtlFpgaNo;
  count1=total_clb_no%TtlFpgaNo;
  for(x=TtlFpgaNo-1; x>(TtlFpgaNo-1-(int)count1); x--)
    clb_cnt[x]=count+1;
  for(y=x; y>((int)count1+x-(int)TtlFpgaNo); y--)
    clb_cnt[y]=count;
  if(cp_ana==0)
    NetwkInit(TtlFpgaNo-1, 0);                    /* obtain initial partition */
  else
  {
    count=LockCP();                               /* lock CLBs in critical path in 1 FPGA */
    NetwkInit(count, cnt);                        /* obtain initial partition by assigning other CLBs */
    ClearCP();                                    /* delete all paths created in critical path analysis */
  };
  do
  {
    final_cut=min_cut_size;
    PartiInit();                                  /* initialization for partitioning */
    printf("\nPartitioning pass %d...\n", pass+1);
    Pass();
    pass++;
  }
  while(min_cut_size<final_cut);
  RearrangeCLBId();
  printf("\nFinal Assigment with cut size %d:", final_cut);
  printf("\nCLB distribution: {%d, %d, %d, %d}.\n", clb_cnt[0], clb_cnt[1],
           clb_cnt[2], clb_cnt[3]);
  AsgnNoNeiborCLB();                              /* assign CLB which has no neighbour CLB */
}

NetwkInit(i, j)                                   /* assign CLBs randomly */
unsigned i, j;
{
  unsigned count;

  for(count=0; count<total_clb_no; count++)
    if(clb[count].fpga_id==-1)                    /* CLB not assigned to any FPGA */
    {
      if(j==clb_cnt[i])
      {
          i--;
          j=0;
      }
      clb[count].fpga_id=(int)i;
      j++;
    }
}

LockCP()                                          /* lock CLBs in critical path in 1 FPGA */
{
  unsigned i, k;
  int j;
  struct SUCCESSOR *temp;

  i=TtlFpgaNo-1;
  k=0;
  temp=head_cp0.first;
  do
  {
    if(k==clb_cnt[i])
    {
      i--;
      k=0;
    }
    j=cell[temp->suc_cell_id].clb_id;
    if(j!=-1)                                     /* CLB which is no neighbour CLB */
    {
      if(clb[j].locked==0)
      {
          clb[j].fpga_id=(int)i;
          clb[j].locked=2;
          k++;
      };
      temp=temp->next_suc_cell;
    };
  }
  while(temp!=(struct SUCCESSOR *)NULL);
  cnt=k;
  return(i);
}

ClearCP()                                         /* delete all paths created in critical path analysis */
{
  unsigned count;

  DeAllocPath(&(head_cp0.first));
  for(count=0; count<total_cell_no; count++)
  {
    if(cell[count].suc_cell_first!=(struct SUCCESSOR *)NULL)
      DeAllocPath(&(cell[count].suc_cell_first));
  }
}

ChangeCLBId()
```

```
{
  unsigned count, count1, i;

/*  i=0;
  for(count=0; count<FPGA_LMT; count++)
    temp_fpga[count]=-1;
  for(count=0; count<FPGA_LMT; count++)
  {
    if(clb_cnt[count]>0)
    {
      temp_fpga[i]=(int)count;
      if(i!=count)
      {
          for(count1=0; count1<total_clb_no; count1++)
          {
            if(clb[count1].fpga_id==(int)count)
              clb[count1].fpga_id=(int)i;
          }
          clb_cnt[i]=clb_cnt[count];
      }
      i++;
    }
  }
  while(i<FPGA_LMT)
  {
    clb_cnt[i]=0;
    i++;
  }*/
}

PartiInit()                                          /* initialization for partitioning */
{
  unsigned n, a, f, t, b, cnt[FPGA_LMT], count, count1, count2;
  int i, j, k;
  struct LIST *temp;
  struct DL_LIST *temp1;

  for(count=0; count<TtlFpgaNo; count++)
    cnt[count]=0;
  for(n=0; n<total_net_no; n++)
    for(a=0; a<TtlFpgaNo; a++)
      clb_dist[n][a]=0;                              /* initialize no. of CLBs incident on net i in FPGA j */
  for(f=0; f<TtlFpgaNo; f++)
    for(t=0; t<TtlFpgaNo; t++)
      for(count=0; count<CLB_IN_FPGA; count++)
      {
          gain[f][t][count].clb_id=-1;               /* initialize gain of CLB after moving to another FPGA */
          gain[f][t][count].gain_value=0;
      }
  for(n=0; n<total_net_no; n++)                      /* initialize CLB distribution */
  {
    temp=net[n].blk_first;
    while(temp!=(struct LIST *)NULL)
    {
      a=(unsigned)clb[temp->lst_id].fpga_id;
      clb_dist[n][a]++;
      temp=temp->next_lst;
    };
  }
  cut_size=0;
  for(n=0; n<total_net_no; n++)                      /* calculate cut size between FPGAs */
  {
    for(a=0; a<TtlFpgaNo; a++)
      if(clb_dist[n][a]>0)
          cut_size++;
    cut_size--;
    min_cut_size=cut_size;
  }
  for(a=0; a<TtlFpgaNo; a++)                         /* calculate gain of CLB after moving to another FPGA */
  {
    for(b=0; b<total_clb_no; b++)
    {
      if(clb[b].fpga_id==(int)a)
      {
          for(count=0; count<TtlFpgaNo; count++)
          {
            if(count!=a)
            {
              count2=cnt[a];
              gain[a][count][count2].clb_id=(int)b;
            };
          }
          temp=clb[b].sig_first;
          while(temp!=(struct LIST *)NULL)
          {
            n=temp->lst_id;
            for(count=0; count<TtlFpgaNo; count++)
            {
              if(count==a)
              {
                if(clb_dist[n][count]==1)
                {
                    for(count1=0; count1<TtlFpgaNo; count1++)
                    {
                      if(count1!=a)
                        gain[a][count1][count2].gain_value++;
```

```
                }
              };
            }
            else
            {
              if(clb_dist[n][count]==0)
                  gain[a][count][count2].gain_value--;
            };
          }
          temp=temp->next_lst;
        };
        cnt[a]++;
    };
  }
}
total_gain_no=MaxPin*2+1;
for(count=0; count<TtlFpgaNo; count++)
 for(count1=0; count1<TtlFpgaNo; count1++)
  {
    if(count!=count1)
    {
        i=(int)MaxPin;
        for(count2=0; count2<total_gain_no; count2++)
        {                                          /* make gain structure */
          bucket[count][count1][count2]=(struct DL_LIST *)calloc(1,
                                          (unsigned)(sizeof(struct DL_LIST)));
          ChkMemDL(bucket[count][count1][count2]);
          bucket[count][count1][count2]->mag=i;
          bucket[count][count1][count2]->next=(struct DL_LIST *)NULL;
          bucket[count][count1][count2]->prevs=(struct DL_LIST *)NULL;
          i--;
      }
        for(count2=0; count2<cnt[count]; count2++)
        {                                          /* make gain list */
          i=gain[count][count1][count2].clb_id;
          if(clb[i].locked==0)
          {
            j=gain[count][count1][count2].gain_value;
            a=MaxPin-(unsigned)j;
            gain_lst=(struct DL_LIST *)calloc(1, (unsigned)
                    (sizeof(struct DL_LIST)));
            ChkMemDL(gain_lst);
            if(count1>count)
              k=(int)count1-1;
            else
              k=(int)count1;
            clb[i].gain_bukt[k]=gain_lst;
            gain_lst->mag=i;
            Add1Gain(count, count1, a);
          }
      }
        count2=0;
        while(count2<total_gain_no &&              /* find max. gain pointer of each bucket */
            bucket[count][count1][count2]->next==(struct DL_LIST *)NULL)
          count2++;
        if(count2<total_gain_no &&
         bucket[count][count1][count2]->next!=(struct DL_LIST *)NULL)
        {
          gain_ptr[count][count1].next=bucket[count][count1][count2];
          gain_ptr[count][count1].mag=bucket[count][count1][count2]->mag;
        }
    }
  }
 total_heap_no=0;
 for(count=0; count<TtlFpgaNo; count++)            /* record all max. gain pointers in heap */
   for(count1=0; count1<TtlFpgaNo; count1++)
     if(count!=count1 && gain_ptr[count][count1].next!=(struct DL_LIST *)NULL)
     {
        if(heap[0].best_move==(struct DL_LIST *)NULL)
        {
          WriteHeap(0, count, count1);
          total_heap_no++;
        }
        else
          InsertHeap(count, count1);
     }
}

ChkMemDL(dl_lst)                                   /* check for sufficient memory for structure for DL_LIST */
struct DL_LIST *dl_lst;
{
  if(dl_lst==(struct DL_LIST *)NULL)
  {
    printf("\nOut of Memory!");
    exit(0);
  }
}

Add1Gain(x, y, z)                                  /* add 1 gain list in gain structure */
unsigned x, y, z;
{
  if(bucket[x][y][z]->next==(struct DL_LIST *)NULL)
  {
    bucket[x][y][z]->next=gain_lst;
    gain_lst->prevs=bucket[x][y][z];
    gain_lst->next=(struct DL_LIST *)NULL;
```

```
  }
  else
  {
    bucket[x][y][z]->next->prevs=gain_lst;
    gain_lst->next=bucket[x][y][z]->next;
    gain_lst->prevs=bucket[x][y][z];
    bucket[x][y][z]->next=gain_lst;
  }
}

InsertHeap(j, k)                                    /* search appropiate position & insert 1 heap record */
unsigned j, k;
{
  unsigned i;

  i=0;
  while(i<total_heap_no && gain_ptr[j][k].mag<heap[i].move_gain)
    i++;
  if(i==total_heap_no && heap[i].best_move==(struct DL_LIST *)NULL)
  {
    WriteHeap(i, j, k);
    total_heap_no++;
  }
  else
  {
    HeapShiftDown(i);
    WriteHeap(i, j, k);
    total_heap_no++;
  }
}

WriteHeap(x, y, z)                                  /* write the best gain pointer */
unsigned x, y, z;
{
  heap[x].best_move=gain_ptr[y][z].next;
  heap[x].fm_fpga=y;
  heap[x].to_fpga=z;
  heap[x].base_clb=(unsigned)heap[x].best_move->next->mag;
  heap[x].move_gain=heap[x].best_move->mag;
}

HeapShiftDown(m)                                    /* shift down by 1 records from heap[m] */
unsigned m;
{
  int l;

  for(l=(int)total_heap_no-1; l>=(int)m; l--)
    if(l>=0)
    {
      heap[l+1].best_move=heap[l].best_move;
      heap[l+1].fm_fpga=heap[l].fm_fpga;
      heap[l+1].to_fpga=heap[l].to_fpga;
      heap[l+1].base_clb=heap[l].base_clb;
      heap[l+1].move_gain=heap[l].move_gain;
    }
}

Pass()                                              /* 1 pass */
{
  unsigned i;

  move_no=0;
  no_valid_move=0;
  while(total_heap_no!=0 && !no_valid_move)
  {
/*    if(move_no==8)
    {
      exit(0);
    }*/
    MakeMove();
  }
  total_move_no=move_no;
  ReverseMove();
}

MakeMove()                                          /* move base CLB from own FPGA to another FPGA */
{
  unsigned i, j, k, l, m, n, o, p;
  int x, y;
  struct LIST *temp, *temp1;

  l=0;
  i=heap[0].base_clb;
  j=heap[0].fm_fpga;
  k=heap[0].to_fpga;
  x=heap[0].move_gain;
  if(pass==0)                                       /* perform at least 1 pass even cut size > limit initially */
  {
    if(!bal_parti)
    {
      while(l<total_heap_no && clb_cnt[k]+1>CLB_IN_FPGA)
      {                                             /* invalid if no. of CLB in FPGA > limit */
        l++;                                        /* test next heap record */
        i=heap[l].base_clb;
        j=heap[l].fm_fpga;
```

```
        k=heap[l].to_fpga;
        x=heap[l].move_gain;
    }
  }
  else
  {
    while(l<total_heap_no && (clb_cnt[j]-1<min_clb_no ||
          clb_cnt[k]+1>max_clb_no))
    {
        l++;                                        /* test next heap record */
        i=heap[l].base_clb;
        j=heap[l].fm_fpga;
        k=heap[l].to_fpga;
        x=heap[l].move_gain;
    }
  }
}
else if(pass>0)
{
  if(!bal_parti)
  {
    while(l<total_heap_no &&                        /* test valid move */
          (clb_cnt[k]+1>CLB_IN_FPGA || cut_size-(unsigned)x>CUT_LMT))
                                                    /* invalid if cut size or no. of CLB in FPGA > limit */
    {
        l++;
        i=heap[l].base_clb;
        j=heap[l].fm_fpga;
        k=heap[l].to_fpga;
        x=heap[l].move_gain;
    }
  }
  else
  {
    while(l<total_heap_no && (clb_cnt[j]-1<min_clb_no ||
          clb_cnt[k]+1>max_clb_no || cut_size-(unsigned)x>CUT_LMT))
    {
        l++;                                        /* test next heap record */
        i=heap[l].base_clb;
        j=heap[l].fm_fpga;
        k=heap[l].to_fpga;
        x=heap[l].move_gain;
    }
  }
}
if(l==total_heap_no)
{
  no_valid_move=1;                                  /* no valid move in heap */
  return(0);
}
clb[i].locked=1;                                    /* lock base CLB */
clb[i].fpga_id=(int)k;
clb_cnt[j]--;
clb_cnt[k]++;
cut_size=cut_size-(unsigned)x;                      /* update cut size */
WriteMoveRcd(i, j, k, cut_size);                    /* record move */
UpdateGainPtrBf(j, k, i, x, 0);
RemoveGainNode(i, j, k);                            /* remove gain node */
for(l=0; l<TtlFpgaNo; l++)
{
  if(l!=k && j!=l)
  {
    m=0;
    while(gain[j][l][m].clb_id!=(int)i)
        m++;
    x=gain[j][l][m].gain_value;
    if(x==gain_ptr[j][l].mag)
        UpdateGainPtrBf(j, l, i, x, 0);
    RemoveGainNode(i, j, l);
  }
}
temp=clb[i].sig_first;                              /* update CLB distribution, gain & gain structure */
while(temp!=(struct LIST *)NULL)                    /* before move */
{
  l=temp->lst_id;
  if(clb_dist[l][k]==0)
  {
    temp1=net[l].blk_first;
    while(temp1!=(struct LIST *)NULL)
    {
        m=temp1->lst_id;
        if(clb[m].locked==0)
        {
          if(clb[m].fpga_id==(unsigned)j)
          {
            n=0;
            while(gain[j][k][n].clb_id!=(int)m)
              n++;
            y=gain[j][k][n].gain_value;
            o=MaxPin-(unsigned)y;
            UpdateAll(j, k, n, m, o, y, 1);
          }
          for(o=0; o<TtlFpgaNo; o++)
            if(o!=j && o!=k)
              if(clb[m].fpga_id==(unsigned)o)
              {
                  n=0;
```

```
                while(gain[o][k][n].clb_id!=(int)m)
                  n++;
                y=gain[o][k][n].gain_value;
                p=MaxPin-(unsigned)y;
                UpdateAll(o, k, n, m, p, y, 1);
              }
        }
        temp1=temp1->next_lst;
    }
}
else if(clb_dist[l][k]==1)
{
    temp1=net[l].blk_first;
    while(temp1!=(struct LIST *)NULL)
    {
        m=temp1->lst_id;
        if(clb[m].locked==0 && clb[m].fpga_id==(int)k)
        {
          n=0;
          while(gain[k][j][n].clb_id!=(int)m)
            n++;
          y=gain[k][j][n].gain_value;
          o=MaxPin-(unsigned)y;
          UpdateAll(k, j, n, m, o, y, 0);
          for(o=0; o<TtlFpgaNo; o++)
            if(o!=j && o!=k)
            {
              y=gain[k][o][n].gain_value;
              p=MaxPin-(unsigned)y;
              UpdateAll(k, o, n, m, p, y, 0);
            }
        }
        temp1=temp1->next_lst;
    }
}
clb_dist[l][j]--;                                   /* move */
clb_dist[l][k]++;
if(clb_dist[l][j]==0)                               /* after move */
{
    temp1=net[l].blk_first;
    while(temp1!=(struct LIST *)NULL)
    {
        m=temp1->lst_id;
        if(clb[m].locked==0)
        {
          if(clb[m].fpga_id==(int)k)
          {
            n=0;
            while(gain[k][j][n].clb_id!=(int)m)
              n++;
            y=gain[k][j][n].gain_value;
            o=MaxPin-(unsigned)y;
            UpdateAll(k, j, n, m, o, y, 0);
          }
          for(o=0; o<TtlFpgaNo; o++)
            if(o!=j && o!=k)
              if(clb[m].fpga_id==(int)o)
              {
                  n=0;
                  while(gain[o][j][n].clb_id!=(int)m)
                    n++;
                  y=gain[o][j][n].gain_value;
                  p=MaxPin-(unsigned)y;
                  UpdateAll(o, j, n, m, p, y, 0);
              }
        }
        temp1=temp1->next_lst;
    }
}
else if(clb_dist[l][j]==1)
{
    temp1=net[l].blk_first;
    while(temp1!=(struct LIST *)NULL)
    {
        m=temp1->lst_id;
        if(clb[m].locked==0 && clb[m].fpga_id==(int)j)
        {
          n=0;
          while(gain[j][k][n].clb_id!=(int)m)
            n++;
          y=gain[j][k][n].gain_value;
          o=MaxPin-(unsigned)y;
          UpdateAll(j, k, n, m, o, y, 1);
          for(o=0; o<TtlFpgaNo; o++)
            if(o!=j && o!=k)
            {
              y=gain[j][o][n].gain_value;
              p=MaxPin-(unsigned)y;
              UpdateAll(j, o, n, m, p, y, 1);
            }
        }
        temp1=temp1->next_lst;
    }
}
temp=temp->next_lst;
}
```

```
}

WriteMoveRcd(i, j, k, l)                          /* record all moves within a pass */
unsigned i, j, k, l;
{
  move_rcd[move_no].base_clb=i;
  move_rcd[move_no].fm_fpga=j;
  move_rcd[move_no].to_fpga=k;
  move_rcd[move_no].move_gain=(int)l;
  move_rcd[move_no].best_move=heap[0].best_move;
  move_no++;
  if(move_no==REAL_CLB_LMT)
  {
    printf("\nError! (Move no)\n");
    exit(0);
  }
}

UpdateGainPtrBf(i, j, n, k, o)                    /* update gain pointer before removing gain node */
unsigned i, j, n, k, o;
{
  unsigned l, m;

  if(o==DEC)
    l=MaxPin-k+1;
  m=0;
  while(heap[m].fm_fpga!=i || heap[m].to_fpga!=j)
    m++;
  if(gain_ptr[i][j].next->next->next==(struct DL_LIST *)NULL)
  {                                               /* update max. gain pointer */
    if(o==DEC)
    {
      HeapShiftUp(m);
      if(k+MaxPin!=0)                             /* max. gain pointer not at bottom of bucket */
      {
          while(l<total_gain_no &&
              bucket[i][j][l]->next==(struct DL_LIST *)NULL)
            l++;
          if(l<total_gain_no &&
           bucket[i][j][l]->next!=(struct DL_LIST *)NULL)
          {
            gain_ptr[i][j].mag=bucket[i][j][l]->mag;
            gain_ptr[i][j].next=bucket[i][j][l];
            InsertHeap(i, j);                     /* update heap */
          }
          else if(l==total_gain_no)
          {
            gain_ptr[i][j].next=(struct DL_LIST *)NULL;
            gain_ptr[i][j].mag=-1-(int)MaxPin;
          }
      }
      else
      {
          gain_ptr[i][j].next=(struct DL_LIST *)NULL;
          gain_ptr[i][j].mag=-1-(int)MaxPin;
      }
    }
    else
    {
      gain_ptr[i][j].next=(struct DL_LIST *)NULL;
      gain_ptr[i][j].mag=-1-(int)MaxPin;
    }
  }
  else
  {
    if(o==DEC)
    {
      if(gain_ptr[i][j].next->next->mag==(unsigned)n)
          heap[m].base_clb=(unsigned)(gain_ptr[i][j].next->next->next->mag);
    }
    else
    {
      gain_ptr[i][j].next=(struct DL_LIST *)NULL;
      gain_ptr[i][j].mag=-1-(int)MaxPin;
    }
  }
}

HeapShiftUp(i)                                    /* delete heap[i] record & move all lower records up */
unsigned i;
{
  unsigned l;

  for(l=i+1; l<total_heap_no; l++)
  {
    heap[l-1].fm_fpga=heap[l].fm_fpga;
    heap[l-1].to_fpga=heap[l].to_fpga;
    heap[l-1].base_clb=heap[l].base_clb;
    heap[l-1].move_gain=heap[l].move_gain;
    heap[l-1].best_move=heap[l].best_move;
  }
  heap[l-1].best_move=(struct DL_LIST *)NULL;
  total_heap_no--;
}

RemoveGainNode(i, j, k)                           /* remove gain node of base CLB */
```

```
unsigned i, j, k;
{
  unsigned l;

  if(k>j)
    l=k-1;
  else
    l=k;
  if(clb[i].gain_bukt[l]->next==(struct DL_LIST *)NULL)
  {
    clb[i].gain_bukt[l]->prevs->next=(struct DL_LIST *)NULL;
    free((struct DL_LIST *)clb[i].gain_bukt[l]);
    clb[i].gain_bukt[l]=(struct DL_LIST *)NULL;
  }
  else
  {
    clb[i].gain_bukt[l]->prevs->next=clb[i].gain_bukt[l]->next;
    clb[i].gain_bukt[l]->next->prevs=clb[i].gain_bukt[l]->prevs;
    free((struct DL_LIST *)clb[i].gain_bukt[l]);
    clb[i].gain_bukt[l]=(struct DL_LIST *)NULL;
  }
}

UpdateAll(i, j, k, l, m, x, o)                    /* update CLB distribution, gain, bucket & heap after a move */
unsigned i, j, k, l, m, o;
int x;
{
  unsigned n;

  if(gain[i][j][k].gain_value==gain_ptr[i][j].mag)
  { if(o==INC)
      UpdateGainPtrBf(i, j, l, x, 1);
    else
      UpdateGainPtrBf(i, j, l, x, 0);
  }
  RemoveGainNode(l, i, j);
  if(o==INC)
    gain[i][j][k].gain_value++;
  else
    gain[i][j][k].gain_value--;
  gain_lst=(struct DL_LIST *)calloc(1, (unsigned)(sizeof(struct DL_LIST)));
  ChkMemDL(gain_lst);
  if(j>i)
    n=j-1;
  else
    n=j;
  clb[l].gain_bukt[n]=gain_lst;
  gain_lst->mag=(int)l;
  if(o==INC)
    Add1Gain(i, j, m-1);
  else
    Add1Gain(i, j, m+1);
  UpdateGainPtrAt(i, j, k);
}

UpdateGainPtrAt(i, j, k)                          /* update gain pointer & heap after removing gain node */
unsigned i, j, k;
{
  unsigned l;
  int x;

  l=0;
  while(l<total_heap_no && (heap[l].fm_fpga!=i || heap[l].to_fpga!=j))
    l++;
  if(l<total_heap_no && gain[i][j][k].gain_value>gain_ptr[i][j].mag)
  {
    gain_ptr[i][j].mag=gain[i][j][k].gain_value;
    x=(int)MaxPin-gain[i][j][k].gain_value;
    gain_ptr[i][j].next=bucket[i][j][x];
    HeapShiftUp(l);
    InsertHeap(i, j);
  }
  else if(l<total_heap_no && gain[i][j][k].gain_value==gain_ptr[i][j].mag)
  {
    heap[l].base_clb=(unsigned)gain[i][j][k].clb_id;
  }
  else if(l==total_heap_no && gain_ptr[i][j].mag==-1-(int)MaxPin)
  {
    l=0;
    while(bucket[i][j][l]->next==(struct DL_LIST *)NULL)
      l++;
    gain_ptr[i][j].mag=(int)(MaxPin-l);
    gain_ptr[i][j].next=bucket[i][j][l];
    InsertHeap(i, j);
  }
}

ReverseMove()                                     /* reverse all moves until min. cut size reached */
{
  unsigned i, j, k, l, min_cut_index;
  int x;

  if(total_move_no!=0)
  {
    for(i=0; i<total_move_no; i++)                /* final min. cut size within a pass */
      if(move_rcd[i].move_gain<(int)min_cut_size)
```

```
    {
        min_cut_size=(unsigned)(move_rcd[i].move_gain);
        min_cut_index=(int)i;
    }
  if(min_cut_size<final_cut)                    /* if < final cut size, reverse move */
  {
    for(i=total_move_no-1; i>min_cut_index; i--)
    {
        j=move_rcd[i].base_clb;
        k=move_rcd[i].fm_fpga;
        l=move_rcd[i].to_fpga;
        clb[j].fpga_id=(int)k;
        clb_cnt[l]--;
        clb_cnt[k]++;
    }
    for(i=0; i<TtlFpgaNo; i++)
        temp_clb_cnt[i]=clb_cnt[i];
    for(i=0; i<total_clb_no; i++)
    {
        clb[i].final_fpga_id=clb[i].fpga_id;
        if(clb[i].locked==1)
          clb[i].locked=0;
    }
  }
  else
  {
    if(pass==0)
    {
        printf("Too many interconnections between FPGAs, can't be partitioned!");
        exit(0);
    }
    for(i=0; i<TtlFpgaNo; i++)
        clb_cnt[i]=temp_clb_cnt[i];
  }
 }
 else
 {
   if(pass==0)
   {
     printf("Too many interconnections between FPGAs, can't be partitioned!");
     exit(0);
   }
 }
}

RearrangeCLBId()
{
  unsigned i, k;
  int x;

  if(!bal_parti)
  {
    i=FPGA_LMT-1;
    while(i>=0 && !clb_cnt[i])
      i--;
    if(i+1!=TtlFpgaNo)
      TtlFpgaNo--;
    for(i=0; i<TtlFpgaNo; i++)
    {
      while(!clb_cnt[i])
      {
        for(k=0; k<total_clb_no; k++)
          {
            if(clb[k].final_fpga_id>(int)i)
              clb[k].final_fpga_id--;
          }
          for(k=i+1; k<TtlFpgaNo; k++)
            clb_cnt[k-1]=clb_cnt[k];
        clb_cnt[TtlFpgaNo-1]=0;
        TtlFpgaNo--;
      }
    }
  }
  else
  {
    for(i=0; i<4; i++)
      clb_cnt[i]=0;
    for(i=0; i<total_clb_no; i++)
    {
      k=clb[i].final_fpga_id;
      clb_cnt[k]++;
    }
  }
}

AsgnNoNeiborCLB()
{
  unsigned sort[FPGA_LMT], total, i, j;
  int x;
  struct IO_NET *temp;

  if(TtlNoClb>0)
  {
    total=0;
    for(i=0; i<TtlFpgaNo; i++)
      sort[i]=0;
```

```
    for(i=0; i<TtlFpgaNo; i++)
    {
      j=0;
      while(j<total && clb_cnt[i]>clb_cnt[sort[j]])
          j++;
      if(j<total)
      {
          for(x=(int)total-1; x>=(int)j; x--)
            sort[x+1]=sort[x];
      }
      sort[j]=i;
      total++;
    }
    i=0;
    while(clb_cnt[sort[i]]==0)
      i++;
    for(j=0; j<TtlNoClb; j++)
    {
      while(clb_cnt[sort[i]]==CLB_IN_FPGA && i<TtlFpgaNo)
          i++;
      if(i==TtlFpgaNo)
          i=0;
      no_neibor_clb[j].fpga_id=(int)sort[i];
      clb_cnt[sort[i]]++;
      temp=head_io_net;
      while(temp->clb!=-1 && strcmp(temp->no_neibor_name, no_neibor_clb[j].name))
        temp=temp->next_io_net;
      temp->fpga_id=(int)sort[i];
    }
    temp=head_io_net;
    while(temp && temp->clb==-1)
    {
      i=0;
      while(strcmp(temp->no_neibor_name, no_neibor_clb[i].name))
          i++;
      temp->fpga_id=(unsigned)no_neibor_clb[i].fpga_id;
      temp=temp->next_io_net;
    }
  }
}
```

Assignio.c

```
/*****************************************************************/
/* clb_type in con[] = 0 : neither input nor output CLB found. */
/*                     1 : only input CLB found.               */
/*                     2 : only output CLB found.              */
/*                     3 : both input & output CLBs found.     */
/*****************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include"parti.h"

#define GBL_LMT 10
#define LOC_LMT 12
#define PROG_LMT 8
#define DATA_LMT 8
#define ADD_LMT 15


extern unsigned total_net_no, net_no, TtlNoClb, TtlFpgaNo;
extern char line[CHAR_NO], entry_nm[RCD_NM_LEN], sym_type[SYM_NM_LEN];
extern char rcd_nm[RCD_NM_LEN], sym_nm[SYM_NM_LEN], pin_nm[3], pin_type[2];
extern char sig_nm[SYM_NM_LEN];
extern FILE *f_ptr, *rf_ptr;
extern struct SIGNAL net[NET_LMT];
extern struct BLOCK clb[REAL_CLB_LMT_D];
extern struct IO_NET *head_io_net;
extern struct NO_NEIBOR no_neibor_clb[NO_NEIBOR_CLB];

struct GBL_BUS
{
  unsigned pin[GBL_LMT], index;
};

struct ADD_BUS
{
  unsigned pin[ADD_LMT], shared[ADD_LMT], index;
};

struct DATA_BUS
{
  unsigned pin[DATA_LMT], shared[DATA_LMT], index;
};

struct LOC_BUS
{
  unsigned pinx[LOC_LMT], piny[LOC_LMT], index;
};

struct PROG_EVEN
{
  unsigned pinx[PROG_LMT], piny[PROG_LMT], index;
};

struct PROG_ODD
{
  unsigned pin[PROG_LMT], index;
};

unsigned offset, fm_lca, to_lca, fm_pin, to_pin, adj_lca, iob_no, sw_no;
unsigned iob_cnt[FPGA_LMT], lca_io_no;
char pin[2];
struct SYS_INTF sys_intf[FPGA_LMT];
struct GBL_BUS gbl_bus;
struct PROG_EVEN prog_even[FPGA_LMT];
struct PROG_ODD prog_odd[FPGA_LMT];
struct LOC_BUS loc_bus[FPGA_LMT];
struct ADD_BUS add_bus;
struct DATA_BUS data_bus[3];
struct CON con;
struct IOB iob[NET_LMT];
struct SW sw[PROG_PIN_LMT];
struct LCA_IO lca_io[LCA_IO_LMT];

AssignIO()
{
  unsigned i;

  for(i=0; i<GBL_LMT; i++)
    gbl_bus.pin[i]=0;
  AssignInit();                                  /* initialize structure for assigning IO */
  iob_no=0;                                      /* initialization */
  sw_no=0;                                       /* initialization */
  lca_io_no=0;                                   /* initialization */
  for(i=0; i<4; i++)
    iob_cnt[i]=0;
  for(i=0; i<LCA_IO_LMT; i++)                    /* for structure analysis only */
    lca_io[i].adj_lca=2;
  printf("\nAssigning emulation board IOs...\n");
  AssignBoardIO();
  WriteMisc();
  WriteBoardIO();
  AssignBiNet();
```

```
  printf("\nAssigning IOs for interconnection between FPGAs...\n");
  AssignInter24();                              /* assign interconnection IO between 2 or 4 LCAs */
  AssignInter3();
  i=FindLcaIO();                                /* for structure analysis only */
  WriteInterIO();
  WriteCLBRcd();
  if(i)
    WriteSwCon();                               /* write switch connection in .con */
  else
  {
    printf("\nDone.\n\n");
    return(0);
  }
}

AssignInit()
{
  unsigned i, j;

  rf_ptr=fopen("iosource.dat", "rt");
  if(!rf_ptr)
  {
    printf("\nFile iosource.dat not found!");
    printf("\n");
    exit(0);
  }
  SysIntfInit();                                /* initialize structure for system interface */
  GblBusInit();                                 /* initialize structure for global bus */
  ProgIntcInit();                               /* initialize structure for programmable interconnect */
  LocBusInit();                                 /* initialize structure for local bus */
  AddBusInit();                                 /* initialize structure for address bus */
  DataBusInit();                                /* initialize structure for data bus */
  fgets(line, CHAR_NO, rf_ptr);
  if(strncmp(line, "EOF", 3)!=NULL)
    NotDatFile();
}

SysIntfInit()                                   /* initialize structure for system interface */
{
  unsigned i, j, k;

  fgets(line, CHAR_NO, rf_ptr);
  if(strncmp(line, "BOARD IO", 8))
    NotDatFile();
  for(i=0; i<4; i++)
    for(j=0; j<SYS_LMT; j++)
      sys_intf[i].pin[j]=0;                     /* initialization */
  fgets(line, CHAR_NO, rf_ptr);
  while(strncmp(line, "END", 3))
  {
    FindLCA();
    for(i=fm_lca; i<=to_lca; i++)
    {
      k=0;
      if(fm_lca!=to_lca)
          offset=7;
      else
          offset=4;
      while(line[offset]!=';')
      {
          FindPin();
          for(j=fm_pin; j<=to_pin; j++)
          {
            sys_intf[i].pin[k]=j;
            k++;
          }
      }
    }
    fgets(line, CHAR_NO, rf_ptr);
  }
}

NotDatFile()
{
  printf("\nNot iosource.dat file format!");
  printf("\n");
  exit(0);
}

FindLCA()
{
  if(strncmp(line, "LCA", 3))
    NotDatFile();
  if(line[4]=='.')
  {
    fm_lca=(unsigned)atoi(&(line[3]));
    to_lca=(unsigned)atoi(&(line[6]));
  }
  else if(line[4]==':')
  {
    fm_lca=to_lca=(unsigned)atoi(&(line[3]));
  }
  else
    NotDatFile();
}
```

```
FindPin()                                       /* find pin no. */
{
  offset=offset+3;
  if(line[offset]==',' || line[offset]==';')    /* i, */
    fm_pin=to_pin=(unsigned)atoi(&(line[offset-1]));
  else if(line[offset]=='.')
  {
    fm_pin=(unsigned)atoi(&(line[offset-1]));
    offset=offset+3;
    if(line[offset]==',' || line[offset]==';')  /* i..i, */
      to_pin=(unsigned)atoi(&(line[offset-1]));
    else                                        /* i..ii, */
    {
      offset++;
      strncpy(pin, &(line[offset-2]), 2);
      to_pin=(unsigned)atoi(pin);
    }
  }
  else if(line[offset]==':')
  {
    fm_pin=to_pin=(unsigned)atoi(&(line[offset-1]));
    offset=offset+2;
    if(line[offset]==',' || line[offset]==';')  /* i:i, */
      to_lca=(unsigned)atoi(&(line[offset-1]));
    else                                        /* i:ii, */
    {
      offset++;
      strncpy(pin, &(line[offset-2]), 2);
      to_lca=(unsigned)atoi(pin);
    }
  }
  else
  {
    offset++;
    if(line[offset]==',' || line[offset]==';')
    {
      strncpy(pin, &(line[offset-2]), 2);       /* ii, */
      fm_pin=to_pin=(unsigned)atoi(pin);
    }
    else if(line[offset]=='.')
    {
      strncpy(pin, &(line[offset-2]), 2);
      fm_pin=(unsigned)atoi(pin);
      offset=offset+3;
      if(line[offset]==',' || line[offset]==';')  /* ii..i, */
          to_pin=(unsigned)atoi(&(line[offset-1]));
      else                                        /* ii..ii, */
      {
          offset++;
          strncpy(pin, &(line[offset-2]), 2);
          to_pin=(unsigned)atoi(pin);
      }
    }
    else if(line[offset]==':')
    {
      strncpy(pin, &(line[offset-2]), 2);
      fm_pin=to_pin=(unsigned)atoi(pin);
      offset=offset+2;
      if(line[offset]==',' || line[offset]==';')  /* ii:i, */
          to_lca=(unsigned)atoi(&(line[offset-1]));
      else                                        /* ii:ii, */
      {
          offset++;
          strncpy(pin, &(line[offset-2]), 2);
          to_lca=(unsigned)atoi(pin);
      }
    }
  }
}

GblBusInit()                                    /* initialize structure for global bus */
{
  unsigned j, k;

  k=0;
  fgets(line, CHAR_NO, rf_ptr);
  if(strncmp(line, "GBL BUS", 7))
    NotDatFile();
  fgets(line, CHAR_NO, rf_ptr);
  while(strncmp(line, "END", 3))
  {
    offset=7;
    while(line[offset]!=';')
    {
      FindPin();
      for(j=fm_pin; j<=to_pin; j++)
      {
          gbl_bus.pin[k]=j;
          k++;
      }
    }
    fgets(line, CHAR_NO, rf_ptr);
  }
}

ProgIntcInit()                                  /* initialize structure for programmable interconnect */
```

```
{
  unsigned i, j, k;

  fgets(line, CHAR_NO, rf_ptr);
  if(strncmp(line, "PROG INTC", 9))
    NotDatFile();
  fgets(line, CHAR_NO, rf_ptr);
  while(strncmp(line, "END", 3))
  {
    FindLcaProg();
    if(fm_lca%2==1)                                  /* odd path */
    {
      i=(fm_lca-1)/2;
      offset=2;
    }
    else                                             /* even path */
    {
      i=(fm_lca-2)/2;
      offset=7;
    }
    k=0;
    while(line[offset]!=';')
    {
      FindPin();
      for(j=fm_pin; j<=to_pin; j++)
      {
          if(fm_lca%2==1)
            prog_odd[i].pin[k]=j;
          else
          {
            if(to_lca==0)
              prog_even[i].pinx[k]=j;
            else
              prog_even[i].piny[k]=j;
          }
          k++;
      }
    }
    fgets(line, CHAR_NO, rf_ptr);
  }
}

FindLcaProg()
{
  if(line[0]=='P' && line[2]==':')
    fm_lca=(unsigned)atoi(&(line[1]));
  else if(line[0]=='P' && line[2]==',')
  {
    fm_lca=(unsigned)atoi(&(line[1]));
    to_lca=0;
  }
  else if(!strncmp(line, "  ,LCA", 6))
    to_lca=1;
  else
    NotDatFile();
}

LocBusInit()                                         /* initialize structure for local bus */
{
  unsigned j, k;

  fgets(line, CHAR_NO, rf_ptr);
  if(strncmp(line, "LOC BUS", 7))
    NotDatFile();
  fgets(line, CHAR_NO, rf_ptr);
  while(strncmp(line, "END", 3))
  {
    FindLCAPair();
    k=0;
    to_lca=0;
    offset=6;
    while(line[offset]!=';')
    {
      FindPin();
      for(j=fm_pin; j<=to_pin; j++)
      {
          if(to_lca==0)
            loc_bus[fm_lca].pinx[k]=loc_bus[fm_lca].piny[k]=j;
          else
          {
            loc_bus[fm_lca].pinx[k]=j;
            loc_bus[fm_lca].piny[k]=to_lca;
          }
          k++;
      }
    }
    fgets(line, CHAR_NO, rf_ptr);
  }
}

FindLCAPair()
{
  if(strncmp(line, "LCA", 3))
    NotDatFile();
  fm_lca=(unsigned)atoi(&(line[3]));
  if(line[4]!=',')
```

```
    NotDatFile();
}

AddBusInit()                                    /* initialize structure for address bus */
{
  unsigned j, k;

  k=0;
  fgets(line, CHAR_NO, rf_ptr);
  if(strncmp(line, "ADD BUS", 7))
    NotDatFile();
  fgets(line, CHAR_NO, rf_ptr);
  while(strncmp(line, "END", 3))
  {
    FindLcaOther();
    offset=7;
    while(line[offset]!=';')
    {
      FindPin();
      for(j=fm_pin; j<=to_pin; j++)
      {
          add_bus.pin[k]=j;
          add_bus.shared[k]=to_lca;
          k++;
      }
    }
    fgets(line, CHAR_NO, rf_ptr);
  }
}

DataBusInit()
{
  unsigned j, k;

  k=0;
  fgets(line, CHAR_NO, rf_ptr);
  if(strncmp(line, "DATA BUS", 8))
    NotDatFile();
  fgets(line, CHAR_NO, rf_ptr);
  while(strncmp(line, "END", 3))
  {
    FindLcaOther();
    offset=7;
    while(line[offset]!=';')
    {
      FindPin();
      for(j=fm_pin; j<=to_pin; j++)
      {
          k=k%8;
          data_bus[fm_lca].pin[k]=j;
          data_bus[fm_lca].shared[k]=to_lca;
          k++;
      }
    }
    fgets(line, CHAR_NO, rf_ptr);
  }
}

FindLcaOther()
{
  if(!strncmp(line, "LCA", 3) && line[4]==',')
  {
    fm_lca=(unsigned)atoi(&(line[3]));
    to_lca=(unsigned)atoi(&(line[6]));
  }
  else if(!strncmp(line, "    ,", 4))
    to_lca=(unsigned)atoi(&(line[6]));
  else
    NotDatFile();
}

AssignBoardIO()
{
  unsigned i, j, k;
  struct IO_NET *temp;

  temp=head_io_net;
  while(temp)
  {
    if(temp->lca_con_no==1)
    {
      if(temp->clb!=-1)
      {
        k=clb[temp->clb].final_fpga_id;
        temp->fpga_id=k;
      }
      else
        k=temp->fpga_id;
      j=sys_intf[k].index;
      if(j==SYS_LMT)
      {
          printf("\nNo vacancy in System Interface on LCA[%d]. Assigning board IO fails...\n",
                  clb[temp->clb].final_fpga_id);
          exit(0);
      }
      else
```

```
  {
      temp->pin_id=sys_intf[k].pin[j];
    temp->lca_con_no=1;
      sys_intf[k].index++;
      lca_io[lca_io_no].lca_con_no=1;
      lca_io_no++;
      ChkLcaIOLmt();
  }
}
else
{
  net_no=temp->net_id;
  i=temp->net_id;
  net_no=i;
  FindLcaNo();
  temp->lca_con_no=con.lca_con_no;
  lca_io[lca_io_no].lca_con_no=con.lca_con_no;
  if(con.lca_con_no==4 || con.lca_con_no==3)
      lca_io[lca_io_no].bd_intf=1;
  lca_io_no++;
  ChkLcaIOLmt();
  switch(con.lca_con_no)
  {
      case 4:
      case 3:
        i=gbl_bus.index;
        if(i==GBL_LMT)
        {
          printf("\nNo vacancy in Global Bus. Assigning Board IO fails...\n");
          exit(0);
        }
        if(temp->i && !temp->o)
        {
          temp->pin_id=gbl_bus.pin[i];
          if(con.lca_con_no==3)
          {
            j=0;
            while(con.lca_id[j])
                j++;
            temp->clb=(int)j;
            k=clb[j].final_fpga_id;
            temp->fpga_id=k;
          }
        }
        else
        {
          printf("\nError! (Output net)\n");
          exit(0);
        }
        gbl_bus.index++;
        break;
       case 2:
        i=0;
        while(!con.lca_id[i])
          i++;
        if((i==0 && con.lca_id[3]) || con.lca_id[(i+1)%4])
                                          /* check adjacent FPGAs */
        {
          adj_lca=1;
          lca_io[lca_io_no-1].adj_lca=1;
          if(i==0 && con.lca_id[3])
          {
            LocCon(1, 3);
            temp->fpga_id=3;
          }
          else
          {
            LocCon(1, i);
            temp->fpga_id=i;
          }
          if(!net[net_no].connected)
          {
            printf("\nNo vacancy in Local Bus. Assigning Board IO fails...\n");
            exit(0);
          }
        }
        else
        {
          adj_lca=0;
          lca_io[lca_io_no-1].adj_lca=0;
          temp->fpga_id=i;
          ProgCon(1, i);
          if(!net[net_no].connected)
          {
            printf("\nNo vacancy in Programmable Bus. Assigning Board IO fails...\n");
            exit(0);
          }
        }
        break;
       case 1:
        i=0;
        while(!con.lca_id[i])
          i++;
        j=sys_intf[i].index;
        if(j==SYS_LMT)
        {
          printf("\nNo vacancy in System Interface on LCA[%d]. Assigning Board IO fails...\n",
```

```
                    i);
            exit(0);
          }
          else
          {
            temp->pin_id=sys_intf[i].pin[j];
            temp->clb=(int)i;
            temp->fpga_id=clb[temp->clb].final_fpga_id;
          }
          sys_intf[i].index++;
          break;
      }
    }
    temp=temp->next_io_net;
  }
}

ChkLcaIOLmt()                                   /* for structure analysis only */
{
  if(lca_io_no==LCA_IO_LMT)
  {
    printf("\nError! (exceeds lca_io_lmt)\n");
    exit(0);
  }
}


AssignBiNet()
{
  unsigned i, o;
  struct LIST *temp;

  for(net_no=0; net_no<total_net_no; net_no++)
  {
    if(net[net_no].bi_net)
    {
      i=0;
      o=0;
      temp=net[net_no].blk_first;
      while(temp)
      {
          if(temp->lst_type==1)
            i=1;
          if(temp->lst_type==0)
            o=1;
          temp=temp->next_lst;
      }
      if(i==1 && o==1)
      {
          FindLcaNoBiNet(0);
          ConBiNet();
          FindLcaNoBiNet(1);
          ConBiNet();
      }
    }
  }
}

FindLcaNoBiNet(i)
unsigned i;
{
  unsigned j;
  struct LIST *temp;

  for(j=0; j<4; j++)
    con.lca_id[j]=0;                            /* initialization */
  con.lca_con_no=0;                             /* initialization */
  con.op_clb=4;                                 /* initialization */
  temp=net[net_no].blk_first;
  while(temp)
  {
    j=clb[temp->lst_id].final_fpga_id;
    if(!con.lca_id[j] || con.op_clb==4)
    {
      if(con.op_clb==4)
      {
          if(!i && temp->lst_type==0)           /* output net */
            con.op_clb=j;
          if(i && temp->lst_type==2)
            con.op_clb=j;
      }
      if(!con.lca_id[j])
      {
          if(!i && temp->lst_type!=1)
          {
            con.lca_con_no++;
            con.lca_id[j]=1;
          }
          if(i && temp->lst_type!=0)
          {
            con.lca_con_no++;
            con.lca_id[j]=1;
          }
      }
    }
    temp=temp->next_lst;
```

```
  }
}

ConBiNet()
{
  lca_io[lca_io_no].lca_con_no=con.lca_con_no;
  lca_io_no++;
  ChkLcaIOLmt();
  switch(con.lca_con_no)
  {
    case 4:
      Con4Lca(0);
      break;
    case 3:
      Con3Lca(0);
      break;
    case 2:
      Con2Lca(0);
      break;
    default:
      break;
  }
}

AssignInter24()                                    /* assign interconnection IO between 2 or 4 io */
{
  for(net_no=0; net_no<total_net_no; net_no++)
  {
    if(!(net[net_no].ip_net^net[net_no].op_net))
    {
      FindLcaNo();
      if(con.lca_con_no==2 || con.lca_con_no==4)   /* for structure analysis */
      {
          lca_io[lca_io_no].lca_con_no=con.lca_con_no;
          lca_io_no++;
          ChkLcaIOLmt();
      }
      switch(con.lca_con_no)
      {
          case 4:
            Con4Lca(0);                            /* connect 4 LCAs together */
            break;
          case 2:
            Con2Lca(0);                            /* connect 2 LCAs together */
            break;
          default:
            break;
      }
    }
  }
}

FindLcaNo()                                        /* find no. of LCAs required to connect net */
{
  unsigned j;
  struct LIST *temp;

  for(j=0; j<4; j++)                               /* initialization */
    con.lca_id[j]=0;                               /* initialization */
  con.lca_con_no=0;                                /* initialization */
  con.op_clb=4;
  temp=net[net_no].blk_first;
  while(temp)
  {
    j=clb[temp->lst_id].final_fpga_id;
    if(!con.lca_id[j] || con.op_clb==4)
    {
      if(con.op_clb==4)
      {
          if(temp->lst_type==0)                    /* output net */
            con.op_clb=j;
      }
      if(!con.lca_id[j])
      {
          con.lca_con_no++;
          con.lca_id[j]=1;
      }
    }
    temp=temp->next_lst;
  }
}

Con4Lca(i)
unsigned i;
{
  GblCon();
  if(!net[net_no].connected)
    ProgTCon(i, 0);
  if(!net[net_no].connected)
    LocCon(i, con.op_clb);
  if(!net[net_no].connected)
    AssignFail();
}

AssignFail()
{
```

```
  printf("\nAll IO resources have been assigned. Assigning Interconnection IO fails...\n");
  exit(0);
}

Con2Lca(i)
unsigned i;
{
  unsigned j;

  j=0;
  while(!con.lca_id[j])
    j++;
  if((j==0 && con.lca_id[3]) || con.lca_id[(j+1)%4])
  {
    adj_lca=1;                                    /* adjacent LCAs */
    lca_io[lca_io_no-1].adj_lca=1;
    if(j==0 && con.lca_id[3])
      LocCon(i, 3);
    else
      LocCon(i, j);                               /* use local bus */
  }
  else
  {
    adj_lca=0;                                    /* not adjacent LCAs */
    lca_io[lca_io_no-1].adj_lca=0;
    ProgCon(i, j);                                /* use programmable bus */
  }
  if(!net[net_no].connected)
  {
    if(j==0 && con.lca_id[3])
      ProgTCon(i, 3);
    else
      ProgTCon(i, j);                             /* use programmable bus */
  }
  if(!net[net_no].connected)
    GblCon();                                     /* use global bus */
  if(!net[net_no].connected)
    AssignFail();
}

LocCon(i, k)                                      /* local bus connection */
unsigned i, k;
{
  unsigned l, m, n, o;

  switch(con.lca_con_no)
  {
    case 1:
      l=loc_bus[k].index;
      if(!k)
          m=loc_bus[3].index;
      else
          m=loc_bus[k-1].index;
      break;
    case 2:
      l=loc_bus[k].index;
      if(l==LOC_LMT)
          return(0);
      if(k==con.op_clb)
      {
          IORcd(k, 0, loc_bus[k].pinx[l]);
          if(k==3)
            IORcd(0, 1, loc_bus[k].piny[l]);
          else
            IORcd(k+1, 1, loc_bus[k].piny[l]);
      }
      else
      {
          IORcd(k, 1, loc_bus[k].pinx[l]);
          if(k==3)
            IORcd(0, 0, loc_bus[k].piny[l]);
          else
            IORcd(k+1, 0, loc_bus[k].piny[l]);
      }
      loc_bus[k].index++;
      net[net_no].connected=1;
      break;
    case 3:
      m=(k+1)%4;
      for(l=0; l<(con.lca_con_no-1); l++)
      {
          if(loc_bus[m].index==LOC_LMT)
            return(0);
          m=(m++)%4;
      }
      if((k+2)%4==con.op_clb)                     /* middle CLB is output CLB */
      {
          SingleLoc((k+1), 1);
          SingleLoc((k+2), 0);
      }
      else
      {
          if((k+1)%4==con.op_clb)                 /* starting CLB is output CLB */
          {
            SingleLoc((k+1), 0);
            SingleLoc((k+2), 0);
```

```
        }
        else                                        /* last CLB is output CLB */
        {
          SingleLoc((k+1), 1);
          SingleLoc((k+2), 1);
        }
      }
      loc_bus[(k+1)%4].index++;
      loc_bus[(k+2)%4].index++;
      net[net_no].connected=1;
      break;
    case 4:
      if(loc_bus[k].index==LOC_LMT || loc_bus[(k+3)%4].index==LOC_LMT)
          return(0);;
      m=loc_bus[(k+1)%4].index;
      n=loc_bus[(k+2)%4].index;
      if(m==LOC_LMT && n==LOC_LMT)
          return(0);
      else if(m<LOC_LMT && n<LOC_LMT)
      {
          if(m<=n)
            o=0;
          else
            o=1;
      }
      else if(m<LOC_LMT)
          o=0;
      else
          o=1;
      SingleLoc(k, 0);
      SingleLoc(k+3, 1);
      if(!o)
      {
          IORcd(((k+1)%4), 0, loc_bus[(k+1)%4].pinx[m]);
          IORcd(((k+2)%4), 1, loc_bus[(k+1)%4].piny[m]);
          loc_bus[(k+1)%4].index++;
      }
      else
      {
          IORcd(((k+2)%4), 1, loc_bus[(k+2)%4].pinx[n]);
          IORcd(((k+3)%4), 0, loc_bus[(k+2)%4].piny[n]);
          loc_bus[(k+2)%4].index++;
      }
      loc_bus[k].index++;
      loc_bus[(k+3)%4].index++;
      net[net_no].connected=1;
      break;
  }
  if(i)                                             /* system interface required */
  {
  }
}

IORcd(i, j, k)
unsigned i, j, k;
{
  iob[iob_no].fpga_id=i;
  iob[iob_no].iob_type=j;
  iob[iob_no].pin_id=k;
  strncpy(iob[iob_no].net_name, net[net_no].sig_name,
            strlen(net[net_no].sig_name)-1);
  iob_no++;
  iob_cnt[i]++;
}

SingleLoc(i, j)
unsigned i, j;
{
  unsigned k;

  k=loc_bus[i%4].index;
  if(j)
  {
    IORcd((i%4), 1, loc_bus[i%4].pinx[k]);
    IORcd(((i+1)%4), 0, loc_bus[i%4].piny[k]);
  }
  else
  {
    IORcd((i%4), 0, loc_bus[i%4].pinx[k]);
    IORcd(((i+1)%4), 1, loc_bus[i%4].piny[k]);
  }
}
                                                  /* programmable bus */
ProgCon(i, k)
unsigned i, k;
{
  unsigned l, m, n, o, p, q, r, s, t, u;

  if(con.lca_con_no==3)                           /* connect 3 LCAs */
  {
    l=0;
    while(con.lca_id[l])
      l++;
    FindPathId(2, k);
    if(!sw[sw_no-1].lca_con_no)
    {
```

```
    sw_no--;
    return(0);
  }
  r=0;
  while(r==k || r==k+2 || r==l)
    r++;
  s=loc_bus[r].index;
  t=loc_bus[(r+3)%4].index;
  if(s==LOC_LMT && t==LOC_LMT)
    return(0);
  else if(s<LOC_LMT && t<LOC_LMT)
  {
    if(s<=t)
        u=0;
    else
        u=1;
  }
  else if(s<LOC_LMT)
    u=0;
  else
    u=1;
}
else                                          /* connect 2 LCAs */
{
  FindPathId(2, k);
  if(!sw[sw_no-1].lca_con_no)
  {
    sw_no--;
    return(0);
  }
}
l=sw[sw_no-1].path_id[0];
m=sw[sw_no-1].path_id[1];
n=prog_odd[(l-1)/2].index;
o=prog_odd[(m-1)/2].index;
p=prog_odd[(l-1)/2].pin[n];
q=prog_odd[(m-1)/2].pin[o];
sw[sw_no-1].index[0]=n;
sw[sw_no-1].index[1]=o;
switch(con.lca_con_no)
{
  case 2:                                     /* connect 2 LCAs via programmable bus */
    if(!sw[sw_no-1].lca_con_no)
    {
        sw_no--;
        return(0);
    }
    if(k==con.op_clb)
    {
        IORcd(k, 0, p);
        IORcd(k+2, 1, q);
    }
    else
    {
        IORcd(k, 1, p);
        IORcd(k+2, 0, q);
    }
    break;                                    /* connect 3 LCAs via programmable bus */
  default:
    if(k==con.op_clb)
    {
        IORcd(k, 0, p);
        IORcd(k+2, 1, q);
    }
    else if(k+2==con.op_clb)
    {
        IORcd(k, 1, p);
        IORcd(k+2, 0, q);
    }
    else
    {
        if(!u)
        {
          IORcd(k, 1, p);
          IORcd(k+2, 0, q);
        }
        else
        {
          IORcd(k, 0, p);
          IORcd(k+2, 1, q);
        }
     }
     if(r==con.op_clb)
     {
        if(!u)
        {
          IORcd(r, 0, loc_bus[r].pinx[s]);
          IORcd(((r+1)%4), 1, loc_bus[r].piny[s]);
          loc_bus[r].index++;
        }
        else
        {
          IORcd(((r+3)%4), 1, loc_bus[(r+3)%4].pinx[t]);
          IORcd(r, 0, loc_bus[(r+3)%4].piny[t]);
          loc_bus[(r+3)%4].index++;
        }
```

```
    }
    else
    {
        if(!u)
        {
          IORcd(r, 1, loc_bus[r].pinx[s]);
          IORcd(((r+1)%4), 0, loc_bus[r].piny[s]);
          loc_bus[r].index++;
        }
        else
        {
          IORcd(((r+3)%4), 0, loc_bus[(r+3)%4].pinx[t]);
          IORcd(r, 1, loc_bus[(r+3)%4].piny[t]);
          loc_bus[(r+3)%4].index++;
        }
    }
    break;
  }
  prog_odd[(l-1)/2].index++;
  prog_odd[(m-1)/2].index++;
  net[net_no].connected=1;
  if(i)
  {
  }
}

FindPathId(i, j)                                    /* find path id. for programmable bus */
unsigned i, j;
{
  unsigned k;

  sw[sw_no].lca_con_no=i;
  if(i==2)                                          /* connect 2 non-adjacent LCAs via programmable bus */
  {
    if(j==0)                                        /* LCA[0] & LCA[2] */
    {
      if(prog_odd[0].index==PROG_LMT || prog_odd[2].index==PROG_LMT)
          sw[sw_no].lca_con_no=0;
      else
      {
          sw[sw_no].path_id[0]=1;                   /* use path[1] & path[5] */
          sw[sw_no].path_id[1]=5;
          sw[sw_no].sw_id=1;                        /* switch 1 */
      }
    }
    else                                            /* LCA[1] & LCA[3] */
    {
      if(prog_odd[1].index==PROG_LMT || prog_odd[3].index==PROG_LMT)
          sw[sw_no].lca_con_no=0;
      else
      {
          sw[sw_no].path_id[0]=3;                   /* use path[3] & path[7] */
          sw[sw_no].path_id[1]=7;
          sw[sw_no].sw_id=2;                        /* switch 2 */
      }
    }
  }
  else                                              /* connect 3 consecutive LCAs via programmable bus */
  {
    switch(j)
    {
      case 0:                                       /* LCA[1], LCA[2] & LCA[3] */
          k=(unsigned)FindOptPath(3, 6, 7, 4);      /* either path[3] & [6] or path[7] & [4] */
          if(k==0)
          {
            sw[sw_no].path_id[0]=3;                 /* use path[3] & path[6] */
            sw[sw_no].path_id[1]=6;
          }
          else if(k==1)
          {
            sw[sw_no].path_id[0]=7;                 /* use path[7] & path[4] */
            sw[sw_no].path_id[1]=4;
          }
          else
            sw[sw_no].lca_con_no=0;                 /* no path available */
          break;
      case 1:                                       /* LCA[0], LCA[2] & LCA[3] */
          k=(unsigned)FindOptPath(1, 6, 5, 8);      /* either path[1] & [6] or path[5] & [8] */
          if(k==0)
          {
            sw[sw_no].path_id[0]=1;                 /* use path[1] & path[6] */
            sw[sw_no].path_id[1]=6;
          }
          else if(k==1)
          {
            sw[sw_no].path_id[0]=5;                 /* use path[5] & path[8] */
            sw[sw_no].path_id[1]=8;
          }
          else
            sw[sw_no].lca_con_no=0;                 /* no path available */
          break;
      case 2:                                       /* LCA[0], LCA[1] & LCA[3] */
          k=(unsigned)FindOptPath(7, 2, 3, 8);      /* either path[7] & [2] or path[3] & [8] */
          if(k==0)
          {
            sw[sw_no].path_id[0]=7;                 /* use path[7] & path[2] */
```

```
            sw[sw_no].path_id[1]=2;
          }
          else if(k==1)
          {
            sw[sw_no].path_id[0]=3;              /* use path[3] * path[8] */
            sw[sw_no].path_id[1]=8;
          }
          else
            sw[sw_no].lca_con_no=0;              /* no path available */
          break;
      default:                                   /* LCA[0], LCA[1] & LCA[2] */
          k=(unsigned)FindOptPath(1, 4, 5, 2);   /* either path[1] & [4] or path[5] & [2] */
          if(k==0)
          {
            sw[sw_no].path_id[0]=1;              /* use path[1] & [4] */
            sw[sw_no].path_id[1]=4;
          }
          else if(k==1)
          {
            sw[sw_no].path_id[0]=5;              /* use path[5] & [2] */
            sw[sw_no].path_id[1]=2;
          }
          else
            sw[sw_no].lca_con_no=0;              /* no path availble */
          break;
    }
  }
  sw_no++;
}

FindOptPath(i, j, k, l)                          /* find optimal path: path[i] & [j] or path[k] & [l]? */
unsigned i, j, k, l;
{
  unsigned m, n;

  if((prog_odd[(i-1)/2].index==PROG_LMT || prog_even[(j/2)-1].index==PROG_LMT) &&
    (prog_odd[(k-1)/2].index==PROG_LMT || prog_even[(l/2)-1].index==PROG_LMT))
    return(2);
  else if(prog_odd[(i-1)/2].index<PROG_LMT && prog_even[(j/2)-1].index<PROG_LMT &&
          prog_odd[(k-1)/2].index<PROG_LMT && prog_even[(l/2)-1].index<PROG_LMT)
  {
    m=prog_odd[(i-1)/2].index+prog_even[(j/2)-1].index;
    n=prog_odd[(k-1)/2].index+prog_even[(l/2)-1].index;
    if(m<=n)
      return(0);
    else
      return(1);
  }
  else if(prog_odd[(i-1)/2].index<PROG_LMT && prog_even[(j/2)-1].index<PROG_LMT)
    return(0);
  else
    return(1);
}

GblCon()                                         /* use global bus to make connection */
{
  unsigned i, j, k;

  if(gbl_bus.index==GBL_LMT)
    return(0);
  i=gbl_bus.index;
  k=gbl_bus.pin[i];
  for(j=0; j<TtlFpgaNo; j++)
  {
    if(con.lca_id[j])
    {
      if(j==con.op_clb)
          IORcd(j, 0, k);
      else
          IORcd(j, 1, k);
    }
  }
  net[net_no].connected=1;
  gbl_bus.index++;
}

ProgTCon(i, j)
unsigned i, j;
{
  unsigned k, l, m, n, o, p, q;

  switch(con.lca_con_no)
  {
    case 4:
      k=con.op_clb;
      if(prog_odd[(k+2)%4].index==PROG_LMT || ((prog_even[(k+3)%4].index==PROG_LMT
           && loc_bus[k].index==PROG_LMT) && (prog_even[k].index==PROG_LMT
           || loc_bus[(k+3)%4].index==PROG_LMT)))
          return(0);
      else if(prog_odd[(k+2)%4].index<PROG_LMT && prog_even[(k+3)%4].index<PROG_LMT
                && loc_bus[k].index<PROG_LMT && prog_even[k].index<PROG_LMT
                && loc_bus[(k+3)%4].index<PROG_LMT)
      {
          l=prog_even[(k+3)%4].index+loc_bus[k].index;
          m=prog_even[k].index+loc_bus[(k+3)%4].index;
          if(l<=m)
```

```
      n=0;
    else
      n=1;
}
else if(prog_odd[(k+2)%4].index<PROG_LMT && prog_even[(k+3)%4].index<PROG_LMT
          && loc_bus[k].index<PROG_LMT)
    n=0;
else
    n=1;
l=prog_odd[(k+2)%4].index;
IORcd(((k+2)%4), 1, prog_odd[(k+2)%4].pin[1]);
sw[sw_no].lca_con_no=3;
sw[sw_no].path_id[0]=((k+2)%4)*2+1;
sw[sw_no].index[0]=l;
if(!n)
{
    m=prog_even[(k+3)%4].index;
    o=loc_bus[k].index;
    IORcd(k, 0, loc_bus[k].pinx[o]);
    IORcd(((k+1)%4), 1, loc_bus[k].piny[o]);
    IORcd(((k+3)%4), 1, prog_even[(k+3)%4].pinx[m]);
    IORcd(k, 0, prog_even[(k+3)%4].piny[m]);
    prog_even[(k+3)%4].index++;
    loc_bus[k].index++;
    sw[sw_no].path_id[1]=(((k+3)%4)+1)*2;
    sw[sw_no].index[1]=m;
    FindSwId(sw_no);
}
else
{
    m=prog_even[k].index;
    o=loc_bus[(k+3)%4].index;
    IORcd(((k+3)%4), 1, loc_bus[(k+3)%4].pinx[o]);
    IORcd(k, 0, loc_bus[(k+3)%4].piny[o]);
    IORcd(k, 0, prog_even[k].pinx[m]);
    IORcd(((k+1)%4), 1, prog_even[k].piny[m]);
    prog_even[k].index++;
    loc_bus[(k+3)%4].index++;
    sw[sw_no].path_id[1]=(k+1)*2;
    sw[sw_no].index[1]=m;
    FindSwId(sw_no);
}
prog_odd[(k+2)%4].index++;
net[net_no].connected=1;
sw_no++;
break;
case 3:
 FindPathId(3, j);
 if(!sw[sw_no-1].lca_con_no)
 {
    sw_no--;
    return(0);
 }
 k=sw[sw_no-1].path_id[0];
 l=sw[sw_no-1].path_id[1];
 m=prog_odd[(k-1)/2].index;
 n=prog_even[(l/2)-1].index;
 o=prog_odd[(k-1)/2].pin[m];
 p=prog_even[(l/2)-1].pinx[n];
 q=prog_even[(l/2)-1].piny[n];
 sw[sw_no-1].index[0]=m;
 sw[sw_no-1].index[1]=n;
 FindSwId(sw_no-1);
 if((k-1)/2==con.op_clb)
    IORcd(((k-1)/2), 0, o);
 else
    IORcd(((k-1)/2), 1, o);
 if((l/2)-1==con.op_clb)
    IORcd(((l/2)-1), 0, p);
 else
    IORcd(((l/2)-1), 1, p);
 if((l/2)%4==con.op_clb)
    IORcd(((l/2)%4), 0, q);
 else
    IORcd(((l/2)%4), 1, q);
 prog_odd[(k-1)/2].index++;
 prog_even[(l/2)-1].index++;
 net[net_no].connected=1;
 break;
case 2:
  if(adj_lca)
  {
    k=prog_even[j].index;
    if(k==PROG_LMT)
      return(0);
    if(j==con.op_clb)
    {
      IORcd(j, 0, prog_even[j].pinx[k]);
      IORcd(((j+1)%4), 1, prog_even[j].piny[k]);
    }
    else
    {
      IORcd(j, 1, prog_even[j].pinx[k]);
      IORcd(((j+1)%4), 0, prog_even[j].piny[k]);
    }
    prog_even[j].index++;
```

```
    }
    else
    {
        k=prog_odd[j+2].index;
        l=prog_even[j].index;
        m=prog_even[(j+3)%4].index;
        if(k==PROG_LMT || (l==PROG_LMT && m==PROG_LMT))
          return(0);
        else if(k<PROG_LMT && l<PROG_LMT && m<PROG_LMT)
        {
          if(l<=m)
            o=0;
          else
            o=1;
        }
        else if(k<PROG_LMT && l<PROG_LMT)
          o=0;
        else
          o=1;
        if(j+2==con.op_clb)
          IORcd(j+2, 0, prog_odd[j+2].pin[k]);
        else
          IORcd(j+2, 1, prog_odd[j+2].pin[k]);
        sw[sw_no].lca_con_no=3;
        sw[sw_no].path_id[0]=((j+2)*2)+1;
        sw[sw_no].index[0]=k;
        if(!o)
        {
          if(j==con.op_clb)
            IORcd(j, 0, prog_even[j].pinx[l]);
          else
            IORcd(j, 1, prog_even[j].pinx[l]);
          sw[sw_no].path_id[1]=(j+1)*2;
          sw[sw_no].index[1]=l;
          FindSwId(sw_no);
          prog_even[j].index++;
        }
        else
        {
          if(j==con.op_clb)
            IORcd(j, 0, prog_even[(j+3)%4].piny[m]);
          else
            IORcd(j, 1, prog_even[(j+3)%4].piny[m]);
          sw[sw_no].path_id[1]=(((j+3)%4)+1)*2;
          sw[sw_no].index[1]=m;
          FindSwId(sw_no);
          prog_even[(j+3)%4].index++;
        }
        prog_odd[j+2].index++;
        sw_no++;
    }
    net[net_no].connected=1;
    break;
  default:
    break;
  }
  if(i)
  {
  }
}

FindSwId(i)
unsigned i;
{
  unsigned temp;

  switch(sw[i].path_id[0])
  {
    case 1:
      if(sw[i].path_id[1]==4)
          sw[i].sw_id=3;
      else if(sw[i].path_id[1]==6)
          sw[i].sw_id=4;
      else
          SwIdError();
      break;
    case 3:
      if(sw[i].path_id[1]==6)
          sw[i].sw_id=7;
      else if(sw[i].path_id[1]==8)
          sw[i].sw_id=8;
      else
          SwIdError();
      break;
    case 5:
      if(sw[i].path_id[1]==2)
      {
          sw[i].sw_id=5;
          temp=sw[i].index[0];
          sw[i].index[0]=sw[i].index[1];
          sw[i].index[1]=temp;
      }
      else if(sw[i].path_id[1]==8)
          sw[i].sw_id=10;
      else
          SwIdError();
```

```
      break;
    case 7:
      temp=sw[i].index[0];
      sw[i].index[0]=sw[i].index[1];
      sw[i].index[1]=temp;
      if(sw[i].path_id[1]==2)
          sw[i].sw_id=6;
      else if(sw[i].path_id[1]==4)
          sw[i].sw_id=9;
      else
          SwIdError();
      break;
  }
}

SwIdError()
{
  printf("Error! (Switch id.)");
  exit(0);
}

AssignInter3()
{
  for(net_no=0; net_no<total_net_no; net_no++)
  {
    if(!(net[net_no].ip_net^net[net_no].op_net))
    {
      FindLcaNo();
      if(con.lca_con_no==3)
      {
          lca_io[lca_io_no].lca_con_no=con.lca_con_no;
          lca_io_no++;
          ChkLcaIOLmt();
          Con3Lca(0);
      }
    }
  }
}

Con3Lca(i)
unsigned i;
{
  unsigned j;

  j=0;
  while(con.lca_id[j])
    j++;
  ProgTCon(i, j);
  if(!net[net_no].connected)
    GblCon();
  if(!net[net_no].connected)
    LocCon(i, j);
  if(!net[net_no].connected)
  {
    if(j%2==0)
      ProgCon(i, 1);
    else
      ProgCon(i, 0);
  }
  if(!net[net_no].connected)
    AssignFail();
}

FindLcaIO()
{
  unsigned i, j[FPGA_LMT], m, k, l, n, o;

  for(i=0; i<FPGA_LMT; i++)                    /* initialization */
    j[i]=0;                                    /* initialization */
  k=0;                                         /* initialization */
  l=0;
  n=0;
  o=0;
/*  printf("\nLCA IO RECORD:");*/
  for(i=0; i<lca_io_no; i++)
  {
    m=lca_io[i].lca_con_no;
    j[m-1]++;
/*    printf("\n%d.\tconnect to %d LCAs. ", i, lca_io[i].lca_con_no);*/
    if(lca_io[i].lca_con_no==2)
    {
      if(lca_io[i].adj_lca==1)
      {
/*        printf("(adj_lca)");*/
        n++;
      }
      else if(!lca_io[i].adj_lca)
      {
/*        printf("(non_adj_lca)");*/
        o++;
      }
      else
      {
        printf("Error! (2)");
        exit(0);
      }
```

```
    }
    if(lca_io[i].lca_con_no==4 && lca_io[i].bd_intf)
    {
/*      printf("(board IO)");*/
      k++;
    }
    if(lca_io[i].lca_con_no==3 && lca_io[i].bd_intf)
    {
/*      printf("(board IO)");*/
      l++;
    }
  }
  printf("\nBus ratio --");
  printf("\nBoard IO : Global bus : Local bus : Programmable bus(2) : ");
  printf("Programmable bus (3)");
  printf("\n(%d : %d : %d : %d)\n", j[0], j[3]+l, n, o, j[2]-l);
  if(!o && !(j[2]-l))
    return(0);
  else
    return(1);
}
```

Write.c

```
#include<string.h>
#include<stdio.h>
#include"parti.h"

extern unsigned TtlFpgaNo, iob_no, bal_parti, net_no, total_net_no, lca_io_no;
extern unsigned clb_no, clb_cnt[FPGA_LMT], blk_nm, sw_no;
extern long cur_f_ptr;
extern char infile[FILENM_LEN], line[CHAR_NO], sym_type[SYM_NM_LEN];
extern char entry_nm[RCD_NM_LEN], sym_nm[SYM_NM_LEN], rcd_nm[RCD_NM_LEN];
extern char pin_nm[3], pin_type[2], sig_nm[SYM_NM_LEN];
extern FILE *f_ptr;
extern struct IOB iob[NET_LMT];
extern struct BLOCK clb[REAL_CLB_LMT_D];
extern struct SIGNAL net[NET_LMT];
extern struct NO_NEIBOR no_neibor_clb[NO_NEIBOR_CLB];
extern struct IO_NET *head_io_net;
extern struct SYS_INTF sys_intf[FPGA_LMT];
extern struct CON con;
extern struct LCA_IO lca_io[LCA_IO_LMT];
extern struct SW sw[PROG_PIN_LMT];

unsigned no_neibor_no, buf_in_lca;
char outfile[FPGA_LMT][FILENM_LEN], temp_line[CHAR_NO], confile[FILENM_LEN];
FILE *ptr[FPGA_LMT];
struct LIST *clb_ptr;

WriteMisc()                                     /* write netlist & executive programs version no. & power record
*/
{
  unsigned i, j, k;

  buf_in_lca=4;                                 /* initialization */
  j=(unsigned)(strlen(infile));
  if(j==FILENM_LEN-1)
    j=j-6;
  else if(j==FILENM_LEN-2)
    j=j-5;
  else
    j=j-4;
  MakeFileName(j);
  for(i=0; i<TtlFpgaNo; i++)                    /* open destination files */
    if(outfile[i][0])
    {
      ptr[i]=fopen(outfile[i], "wt");
      if(!ptr[i])
      {
          printf("\nCannot open destination file after partition!\n");
          exit(0);
      }
    }
  fseek(f_ptr, 0, 0);
  fgets(line, CHAR_NO, f_ptr);
  while(strncmp(line, "SYM,", 4))               /* copy first several lines up to SYM records */
  {
    for(i=0; i<TtlFpgaNo; i++)
      if(outfile[i][0])
          fputs(line, ptr[i]);
    cur_f_ptr=ftell(f_ptr);
    fgets(line, CHAR_NO, f_ptr);
  }
  sscanf(line, "%s %s %s", rcd_nm, sym_nm, sym_type);
  clb_no=0;                                     /* initialization */
  no_neibor_no=0;                               /* initialization */
  while(strncmp(sym_type, "CLB", 3) && strncmp(sym_type, "IOB", 3))
  {
    if(!strcmp(sym_nm, clb[clb_no].blk_name))
    {
      k=(unsigned)(clb[clb_no].final_fpga_id);
      WriteMiscRcd(k);
      clb_no++;
    }
    else if(!strcmp(sym_nm, no_neibor_clb[no_neibor_no].name))
    {
      k=(unsigned)(no_neibor_clb[no_neibor_no].fpga_id);
      WriteMiscRcd(k);
      no_neibor_no++;
    }
    else
    {
      printf("\nERROR! (Write Misc records)");
      exit(0);
    }
  }
}

MakeFileName(j)                                 /* make output filenames end with 00, 01, 02 or 03 */
unsigned j;
{
  unsigned i;

  for(i=0; i<TtlFpgaNo; i++)
  {
    if(clb_cnt[i])
```

```
    {
      strncpy(outfile[i], infile, j);
      switch(i)
      {
          case 0:
            strcat(outfile[0], "00.map");
            break;
          case 1:
            strcat(outfile[1], "01.map");
            break;
          case 2:
            strcat(outfile[2], "02.map");
            break;
          case 3:
            strcat(outfile[3], "03.map");
            break;
      }
    }
  }
}

WriteMiscRcd(i)                                   /* write whole CLB records in file corresponds to different FPGAs
*/
unsigned i;
{
  do
  {
    fputs(line, ptr[i]);
    fgets(line, CHAR_NO, f_ptr);
  }
  while(strncmp(line, "END", 3));
  fputs(line, ptr[i]);
  cur_f_ptr=ftell(f_ptr);
  ReadSYM();
}

WriteBoardIO()                                    /* write emulation board IO */
{
  unsigned i, j;
  struct IO_NET *temp;

  printf("\nWrite IOB records of board IOs...\n");
  sscanf(line, "%s %s %s", rcd_nm, sym_nm, sym_type);
  while(!strncmp(sym_type, "IOB", 3))              /* IOB encountered */
  {
    strcpy(entry_nm, "PIN,");
    SkipLines(entry_nm, 4, f_ptr);
    sscanf(line, "%s %s %s %s", rcd_nm, pin_nm, pin_type, sig_nm);
    temp=head_io_net;
    while(temp && strcmp(temp->net_name, sig_nm))
      temp=temp->next_io_net;
    if(temp)
    {
      if(temp->lca_con_no==1)
      {
          if(temp->clb!=-1)
          {
            i=clb[temp->clb].final_fpga_id;
            j=temp->pin_id;
            WriteIOB(i, j);
          }
          else
          {
            i=temp->fpga_id;
            j=temp->pin_id;
            WriteIOB(i, j);
          }
      }
      else
      {
          j=temp->pin_id;
          switch(temp->lca_con_no)
          {
            case 4:
              for(i=0; i<TtlFpgaNo; i++)
              {
                if(outfile[i][0])
                    WriteIOB(i, j);
              }
              break;
            case 3:
              for(i=0; i<TtlFpgaNo; i++)
              {
                if(i!=temp->fpga_id && outfile[i][0])
                    WriteIOB(i, j);
              }
              break;
            case 2:
              WriteIOB(temp->fpga_id, j);
              break;
          }
      }
    }
    else if(!temp)
    {
      net_no=0;
```

```
    while(net_no<total_net_no && strcmp(net[net_no].sig_name, sig_nm))
        net_no++;
    if(net_no<total_net_no)
    {
        FindLcaNo();
        lca_io[lca_io_no].lca_con_no=1;
        lca_io_no++;
        ChkLcaIOLmt();
        i=sys_intf[con.op_clb].index;
        if(i==SYS_LMT)
        {
          printf("\nNo vacancy in System Interface on LCA[%d]. Assigning board IO fail...\n",
                  con.op_clb);
          exit(0);
        }
        else
          WriteIOB(con.op_clb, sys_intf[con.op_clb].pin[i]);
    }
    else
    {
        TestBufInLca();
        lca_io[lca_io_no].lca_con_no=1;
        lca_io_no++;
        ChkLcaIOLmt();
        i=sys_intf[buf_in_lca].index;
        if(i==SYS_LMT)
        {
          printf("\nNo vacancy in System Interface on LCA[%d]. Assigning board IO fail...\n",
                   buf_in_lca);
          exit(0);
        }
        else
          WriteIOB(buf_in_lca, sys_intf[buf_in_lca].pin[i]);
    }
   }
   else
   {
     printf("\ntesting");
     strncpy(entry_nm, "EXT,", 4);
     SkipLines(entry_nm, 4, f_ptr);
   }
   cur_f_ptr=ftell(f_ptr);
   ReadSYM();
  }
}

WriteIOB(i, j)
unsigned i, j;
{
  char block_name[SYM_NM_LEN], temp_blk_nm[SYM_NM_LEN];

  strncpy(temp_blk_nm, "", SYM_NM_LEN);
  strncpy(block_name, "", SYM_NM_LEN);
  fseek(f_ptr, cur_f_ptr, 0);
  fgets(line, CHAR_NO, f_ptr);
  sscanf(line, "%s %s %s", rcd_nm, block_name, sym_type);
  strncpy(temp_blk_nm, block_name, strlen(block_name)-1);
  if(blk_nm)
    fprintf(ptr[i], "%s %s %s LOC=P%d, BLKNM=%s\n", rcd_nm, block_name,
              sym_type, j, temp_blk_nm);
  else
    fprintf(ptr[i], "%s %s %s, LOC=P%d\n", rcd_nm, block_name, sym_type, j);
  fgets(line, CHAR_NO, f_ptr);
  while(strncmp(line, "SYM,", 4))
  {
    fputs(line, ptr[i]);
    fgets(line, CHAR_NO, f_ptr);
  }
  sscanf(line, "%s %s %s", rcd_nm, sym_nm, sym_type);
  if(blk_nm)
    fprintf(ptr[i], "%s %s %s LOC=P%d, BLKNM=%s\n", rcd_nm, sym_nm, sym_type,
             j, temp_blk_nm);
  else
    fprintf(ptr[i], "%s %s %s, LOC=P%d\n", rcd_nm, sym_nm, sym_type, j);
  fgets(line, CHAR_NO, f_ptr);
  while(strncmp(line, "EXT,", 4))
  {
    fputs(line, ptr[i]);
    fgets(line, CHAR_NO, f_ptr);
  }
  sscanf(line, "%s %s %s", rcd_nm, sym_nm, sym_type);
  if(blk_nm)
    fprintf(ptr[i], "%s %s %s , LOC=P%d, BLKNM=%s\n", rcd_nm, sym_nm, sym_type,
             j, temp_blk_nm);
  else
    fprintf(ptr[i], "%s %s %s , LOC=P%d\n", rcd_nm, sym_nm, sym_type, j);
  strncpy(temp_blk_nm, "", SYM_NM_LEN);
}

TestBufInLca()
{
  unsigned j, k;


  if(buf_in_lca==4)
  {
```

```
    printf("\nAssigning IOs for buffers...\n");
    buf_in_lca=0;
    j=0;
    k=sys_intf[j].index;
    while(j<TtlFpgaNo)
    {
      if(k<SYS_LMT && k<sys_intf[buf_in_lca].index)
          buf_in_lca=j;
      j++;
      k=sys_intf[j].index;
    }
  }
}

WriteInterIO()
{
  unsigned i, j;

  printf("\nWrite IOB records of interconnection IOs...\n");
  for(i=0; i<iob_no; i++)
  {
    j=iob[i].fpga_id;
    if(iob[i].iob_type)                              /* input IO pad */
    {
      fprintf(ptr[j], "SYM, %s_CUT, IOB, LOC=P%d", iob[i].net_name,
              iob[i].pin_id);
      if(blk_nm)
          fprintf(ptr[j], ", BLKNM=%s_CUT\n", iob[i].net_name);
      else
          fprintf(ptr[j], "\n");
      fprintf(ptr[j], "CFG, Base IO\n");
      fprintf(ptr[j], "CFG, Config IN:I OUT TRI\n");
      fprintf(ptr[j], "PIN, I, O, %s,\n", iob[i].net_name);
      fprintf(ptr[j], "MODEL\n");
      fprintf(ptr[j], "SYM, %s_IB, IBUF, LOC=P%d", iob[i].net_name,
              iob[i].pin_id);
      if(blk_nm)
          fprintf(ptr[j], ", BLKNM=%s_CUT\n", iob[i].net_name);
      else
          fprintf(ptr[j], "\n");
      fprintf(ptr[j], "PIN, O, O, %s,\n", iob[i].net_name);
      fprintf(ptr[j], "PIN, I, I, %s_CUT,\n", iob[i].net_name);
      fprintf(ptr[j], "END\nENDMOD\nEND\n");
      fprintf(ptr[j], "EXT, %s_CUT, I, , LOC=P%d", iob[i].net_name,
              iob[i].pin_id);
      if(blk_nm)
          fprintf(ptr[j], ", BLKNM=%s_CUT\n", iob[i].net_name);
      else
          fprintf(ptr[j], "\n");
    }
    else                                             /* output IO pad */
    {
      fprintf(ptr[j], "SYM, %s_CUT, IOB, LOC=P%d", iob[i].net_name,
              iob[i].pin_id);
      if(blk_nm)
          fprintf(ptr[j], ", BLKNM=%s_CUT\n", iob[i].net_name);
      else
          fprintf(ptr[j], "\n");
      fprintf(ptr[j], "CFG, Base IO\n");
      fprintf(ptr[j], "CFG, Config IN OUT:O TRI\n");
      fprintf(ptr[j], "PIN, O, I, %s,\n", iob[i].net_name);
      fprintf(ptr[j], "MODEL\n");
      fprintf(ptr[j], "SYM, %s_OB, OBUF, LOC=P%d", iob[i].net_name,
              iob[i].pin_id);
      if(blk_nm)
          fprintf(ptr[j], ", BLKNM=%s_CUT\n", iob[i].net_name);
      else
          fprintf(ptr[j], "\n");
      fprintf(ptr[j], "PIN, O, O, %s_CUT,\n", iob[i].net_name);
      fprintf(ptr[j], "PIN, I, I, %s,\n", iob[i].net_name);
      fprintf(ptr[j], "END\nENDMOD\nEND\n");
      fprintf(ptr[j], "EXT, %s_CUT, O, , LOC=P%d", iob[i].net_name,
              iob[i].pin_id);
      if(blk_nm)
          fprintf(ptr[j], ", BLKNM=%s_CUT\n", iob[i].net_name);
      else
          fprintf(ptr[j], "\n");
    }
  }
}
                                                     /* write output .map files with only CLB records */
WriteCLBRcd()
{
  unsigned i, k;
  struct IO_NET *temp;

  printf("\nWrite CLB records...\n");
  while(!strncmp(line, "EXT,", 4))
  {
    sscanf(line, "%s %s %s %s", rcd_nm, pin_nm, pin_type, sig_nm);
    temp=head_io_net;
    while(temp && strcmp(temp->net_name, pin_nm))
      temp=temp->next_io_net;
    i=temp->fpga_id;
    fputs(line, ptr[i]);
    cur_f_ptr=ftell(f_ptr);
```

```
  ReadSYM();
 }
 sscanf(line, "%s %s %s", rcd_nm, sym_nm, sym_type);
 do
 {
  if(!strcmp(sym_nm, clb[clb_no].blk_name))
  {
    k=(unsigned)(clb[clb_no].final_fpga_id);
    WriteCLB(k);
    clb_no++;
  }
  else if(!strcmp(sym_nm, no_neibor_clb[no_neibor_no].name))
  {
    k=(unsigned)(no_neibor_clb[no_neibor_no].fpga_id);
    WriteCLB(k);
    no_neibor_no++;
  }
  else
  {
    printf("\nERROR! (Write CLB records)");
    exit(0);
  }
 }
 while(strncmp(line, "EOF", 3)!=(int)NULL);
 for(i=0; i<TtlFpgaNo; i++)
 {
  if(outfile[i][0])
  {
    fputs(line, ptr[i]);
    fflush(ptr[i]);
  }
 }
}

WriteCLB(i)                                    /* write whole CLB records in file corresponds to different FPGAs
*/
unsigned i;
{
  do
  {
    fputs(line, ptr[i]);
    fgets(line, CHAR_NO, f_ptr);
  }
  while(strncmp(line, "ENDMOD", 6)!=(int)NULL);
  fputs(line, ptr[i]);
  fgets(line, CHAR_NO, f_ptr);
  fputs(line, ptr[i]);
  ReadSYM();
}

WriteSwCon()
{
  unsigned i, j, k, l;
  short int con_data, shift;

  printf("\nWrite switch connection in file.\n");
  i=(unsigned)(strlen(infile));
  i=i-4;
  strncpy(confile, infile, i);
  strcat(confile, ".con");
  f_ptr=fopen(confile, "wt");
  if(!f_ptr)
  {
    printf("\nCannot open output file %s!\n", confile);
    exit(0);
  }
  shift=0x0001;                                /* initialization */
  for(i=0; i<8; i++)
  {
    for(j=0; j<8; j++)
    {
      con_data=0x0000;                         /* initialization */
      for(k=0; k<sw_no; k++)
      {
          if(sw[k].index[1]==i && sw[k].index[0]==j)
          {
            l=sw[k].sw_id-1;
            shift=shift<<l;
            con_data=con_data|shift;
            shift=0x0001;
          }
      }
      if(con_data<0x10)
          fprintf(f_ptr, "000");
      else if(con_data<0x100)
          fprintf(f_ptr, "00");
      else if(con_data<0x400)
          fprintf(f_ptr, "0");
      else
      {
          printf("\nError! (Con_data)");
          exit(0);
      }
      fprintf(f_ptr, "%0x", con_data);
    }
```

```
    fprintf(f_ptr, "\n");
  }
  fclose(f_ptr);
  printf("\nDone.\n\n");
}
```

Upbpc.c

```
#include <stdio.h>
#include <io.h>
#include <conio.h>
#include <dos.h>

#define CR 0x0d                                 /* carriage return */
#define ESC 0x1b                                /* escape */

#define menu_top_x 17                           /* menu x co-ordinate */
#define menu_top_y 7                            /* menu y co-ordinate */
                                                /* COM 1 used */
#define RBR 0x03f8                              /* Receiver Buffer Register */
#define THR 0x03f8                              /* Transmitter Hold Register */
#define DLL 0x03f8                              /* Divisor Latch */
#define IER 0x03f9                              /* Interrupt Enable Register */
#define DLM 0x03f9                              /* Division Latch */
#define LCR 0x03fb                              /* Line Control Register */
#define MCR 0x03fc                              /* Modem Status Register */
#define LSR 0x03fd                              /* Line Status Register */

int function_key;

config_data()                                   /* download configuration data to FPGAs */
{
  FILE *fcptr;
  int cdata;
  char cfilename[30];
  unsigned char key;

  printf("\n");
  do
  {
    printf("\nEnter the filename of FPGA configuration data: ");
    scanf("%s", cfilename);
    fcptr = fopen(cfilename, "r");
  }
  while(!fcptr);                                /* file not found */
  clrscr();
  printf("\nDownloading FPGA configuration data...Please wait.");
  while(!feof(fcptr))
  {
    cdata=fgetc(fcptr);
    send(cdata);
  }
  fclose(fcptr);
  clrscr();
  printf("\nDownloading FPGA configuration data completed.\n");
  clrscr();
  do
  {
    printf("\nDownload switch connection data (Y/N)? ");
    key=getch();
  }
  while(key!='Y' && key!='y' && key!='N' && key!='n');
  clrscr();
  if(key=='Y' || key=='y')
    sw_data();
  else
    send(0x00);
}

sw_data()
{
  int cdata;
  char cfilename[30];
  FILE *fcptr;

  do
  {
    printf("\nEnter the filename of switch connection: ");
    scanf("%s", cfilename);
    fcptr = fopen(cfilename, "rt");
  }
  while(!fcptr);
  printf("\nDownloading switch connection data...Please wait.");
  send(0xAA);
  while(!feof(fcptr))
  {
    cdata=fgetc(fcptr);
    send(cdata);
  }
  fclose(fcptr);
  clrscr();
  printf("\nDownloading FPGA switch connection data completed.\n");
  clrscr();
}

emulator()
{
  int keypress, command[20], counter;

  clrscr();
  keypress = 0x00;                              /* initialization */
```

```
counter = 0;
do
{
  if(kbhit())
  {
    keypress = getch();
    command[counter] = keypress;
    if(keypress == 0x08)                        /* if back space */
        counter--;
    else
        counter++;
    if(keypress != 0x1b)                        /* if not ESC */
        send(keypress);
    if(keypress == 0x0d)                        /* if carriage return, determine command */
    {
        counter = 0;
        do
        {
          if(command[counter] == 0x20)          /* if space, skip it */
            counter++;
          else if((command[counter] == 0x44) || (command[counter] == 0x64))
          {                                     /* if D/d, check F/f or S/s */
            counter++;
            if((command[counter] == 0x46) || (command[counter] == 0x66))
            {                                   /* if F/f, check any spaces */
              counter++;
              while(command[counter] == 0x20)   /* skip any spaces */
                  counter++;
              if(command[counter] == 0x0d)      /* until carriage return is detected */
                  config_data();                /* download FPGA configuration data */
            }
            if((command[counter] == 0x53) || (command[counter] == 0x73))
            {                                   /* if S/s, check any spaces */
              counter++;
              while(command[counter] == 0x20)   /* skip any spaces */
                  counter++;
              if(command[counter] == 0x0d)      /* until carriage return is detected */
                  sw_data();                    /* download switch data */
            }
          }
          else
          {
            command[counter] = 0x0d;
          }
        }
        while(command[counter] != 0x0d);
        counter = 0;
    }
  }
  if((inportb(LSR) & 0x01) == 1)
  {
    printf("%c", inportb(RBR));
    outportb(THR, 0x55);
  }
}
while(keypress != 0x1b);
}

send(i)
int i;
{
  outportb(MCR, 0x02);
  while((inportb(LSR) & 0x20) == 0)             /* transmitter hold register */
  {                                             /* not empty */
  }
  outportb(THR, i);
  outportb(MCR, 0x00);
}

main()
{
  outportb(LCR, 0x83);
  outportb(DLL, 0x03);
  outportb(DLM, 0x00);
  outportb(LCR, 0x03);
  do
  {
    textcolor(LIGHTGRAY);
    clrscr();
    textcolor(YELLOW);
    gotoxy(menu_top_x, menu_top_y-1);
    cprintf("==========================");
    gotoxy(menu_top_x, menu_top_y);
    cprintf("Universal Prototyping Board");
    gotoxy(menu_top_x, menu_top_y+1);
    cprintf("==========================");
    textcolor(CYAN);
    gotoxy(menu_top_x, menu_top_y+3);
    cprintf("Press <CR> to begin...");
    gotoxy(menu_top_x, menu_top_y+5);
    cprintf("At any time, press <ESC> to quit...");
    textcolor(LIGHTGRAY);
    function_key = getch();
    if(function_key == CR)
      emulator();
  }
```

```
  while(function_key != ESC);
  textcolor(LIGHTGRAY);
  clrscr();
}
```

Upb.asm

```
CPUVEC  EQU     $00

CODES   EQU     CPUVEC+$000500          * program codes *
ROMDATA EQU     CPUVEC+$0079FE          * data in ROM *
INT6    EQU     CPUVEC+$007DFE          * interrupt request IRQ6 *
RAMDATA EQU     CPUVEC+$080000          * volatile data in local RAM *
STACK   EQU     CPUVEC+$0807FC          * program stack *

RAM0    EQU     CPUVEC+$0C0000
BUFFER  EQU     CPUVEC+$180021          * disable RAMRW *
RAMRW   EQU     CPUVEC+$180023          * enable RAMRW *
MUXTM   EQU     CPUVEC+$180029          * trace mode *
MUXNM   EQU     CPUVEC+$18002B          * normal mode *
LATCH   EQU     CPUVEC+$18002D          * latch to reprogram/restart FPGAs *
CLEAR   EQU     CPUVEC+$18002F          * 1 short circuit *
LCA0    EQU     CPUVEC+$1C0000          * FPGA 0 *
SWFIRST EQU     CPUVEC+$140000          * first switch address *

MR1A    EQU     $180001                 * mode register A *
MR2A    EQU     MR1A
CSRA    EQU     $180003                 * clock select register A *
SRA     EQU     CSRA                    * status register A *
CRA     EQU     $180005                 * command register A *
TBA     EQU     $180007                 * transmitter buffer A *
RBA     EQU     TBA                     * receiver buffer A *
ACR     EQU     $180009                 * auxiliary control register *
IMR     EQU     $18000B                 * interrupt mask register *
IPR     EQU     $18001B                 * input port (unlatched register) *
OPS     EQU     $18001D                 * output port set *
OPR     EQU     $18001F                 * output port reset *

EOS     EQU     $00                     * end of string *
BELL    EQU     $07                     * bell *
BS      EQU     $08                     * back space *
LF      EQU     $0A                     * line feed *
CR      EQU     $0D                     * carriage return *
SPC     EQU     $20                     * space *
COLN    EQU     $3A                     * colon *
PRMT    EQU     $3E                     * prompt *

****************
* Data Storage *
****************
        ORG     ROMDATA
MONMSG  FCB     LF,CR                   * monitor program message *
        FCC     /UNIVERSAL PROTOTYPING BOARD/
        FCB     LF,CR
        FCC     /Type <H> for HELP...../
        FCB     EOS

PROMPT  FCB     LF,LF,CR
        FCB     PRMT,EOS                * prompt *

HELPMSG FCB     LF,LF,CR
        FCC     /Universal Prototyping Board HELP Menu/
        FCB     LF,CR
        FCC     /=====================================================/
        FCB     LF,CR
        FCC     /MM <ADDR> <DATA> -- Modify Memory/
        FCB     LF,CR
        FCC     /BF <SOURCE ADDR> <DEST. ADDR> <DATA> -- Block Fill Memory/
        FCB     LF,CR
        FCC     /DM <ADDR> -- Dump Memory/
        FCB     LF,CR
        FCC     /PS <ADDR> <DATA> -- Program switch/
        FCB     LF,CR
        FCC     /DF -- Download FPGA Configuration data/
        FCB     LF,CR
        FCC     /DS -- Download switch data/
        FCB     LF,CR
        FCC     /RS -- Reset FPGA/
        FCB     LF,CR
        FCC     /RP -- Reprogram FPGA/
        FCB     LF,CR
        FCC     /TM -- Trace Mode/
        FCB     LF,CR
        FCC     /NM -- Normal Mode/
        FCB     LF,CR
        FCC     /=====================================================/
        FCB     EOS

HEADING FCB     LF,LF,CR                * memory table content heading *
        FCC     /       00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F/
        FCB     LF,CR
        FCC     /=============================================================================/
        FCB     LF,CR,EOS

ENDING  FCC     /=============================================================================/
        FCB     EOS

INVCMD  FCB     LF,CR,BELL
        FCC     /Invalid Command!!/
        FCB     EOS
```

```
INVPAR  FCB     LF,CR,BELL
        FCC     /Invalid Parameter!!/
        FCB     EOS

INVDATA FCB     LF,CR,BELL
        FCC     /Invalid Data!!/
        FCB     EOS

ACCADDR FCB     LF,CR,BELL
        FCC     /Accessible Address:/
        FCB     LF,LF,CR
        FCC     /  Local RAM  : $080800 - $080FFF &/
        FCB     LF,CR
        FCC     /  Sharing RAM: $0C0000 - $13FFFF./
        FCB     EOS

INVADDR FCB     LF,CR,BELL
        FCC     /Source > Destination Address!!/
        FCB     EOS

INVSW   FCB     LF,CR,BELL
        FCC     /Address for Programmable Switches:/
        FCB     LF,LF,CR
        FCC     /  $140000 - $14007F./
        FCB     EOS

        ORG     RAMDATA
NEWDATA BLKB    1                       * any new data received? *
RXDATA  BLKB    1                       * stored received data *
COUNTC  BLKB    1                       * character counter *
CMDBUF  BLKB    20                      * command buffer *

        ORG     CPUVEC
SSP     LONG    STACK
PC      LONG    CODES

        ORG     CPUVEC+$78
AV6     DC.L    INT6                    * level 6 interrupt autovector at $78 *

        ORG     CODES
*****************************
* Serial Port Initization *
*****************************
SERINIT MOVE.B  #$20,CRA                * reset receiver *
        MOVE.B  #$30,CRA                * reset transmitter *
        MOVE.B  #$10,CRA                * reset MR to pointer to MR1 *
        MOVE.B  #$13,MR1A               * 8 data bits, no parity *
        MOVE.B  #$07,MR2A               * 1 stop bit *
        MOVE.B  #0,ACR
        MOVE.B  #$CC,CSRA               * Tx & Rx clk.=38.4kx16 *
        MOVE.B  #$02,IMR
        MOVE.B  #$05,CRA                * Tx & Rx enabled *

************************
* Use Supervisor Mode *
************************
        MOVE.W  #$2000,SR               * supervisor mode used *

********************************
* FPGAs & Switches Initization *
********************************
        MOVE.B  #$FF,LATCH              * FPGA reprogram & reset pull high *
        MOVE.B  #$FB,LATCH              * initialize switches *
        MOVE.B  #$FF,LATCH
        MOVE.B  #$0,CLEAR               * initialize 1 short circuit *
        MOVE.B  #$0,BUFFER              * initialize buffer between 68k & sharing RAMs *
        MOVE.B  #$0,MUXTM               * normal mode selected *
        MOVE.B  #$0,MUXNM

*****************
* Main Program *
*****************
        MOVEA.L #MONMSG,A0
        JSR     TXSTR
BEGIN   MOVEA.L #PROMPT,A0
        JSR     TXSTR
        MOVEA.L #CMDBUF,A0
        CLR.L   D0
        CLR.B   NEWDATA
RSTBUF  CLR.B   (A0)+                   * reset command buffer *
        ADDI.B  #1,D0
        CMPI.B  #20,D0
        BNE     RSTBUF
        MOVEA.L #CMDBUF,A0
        CLR.B   COUNTC
NEW     BCLR.B  #0,NEWDATA              * wait until data received *
        BEQ     NEW                     * recieve data from Receiver Buffer *
        MOVE.B  D0,RXDATA               * if CR *
        CMPI.B  #CR,RXDATA              * determine which function chosen *
        BEQ     CMDANA
        CMPI.B  #BS,RXDATA
        BEQ     DELBUF
        CMPI.B  #20,COUNTC              * character entered > 20 *
        BEQ     BUFFULL                 * commandd buffer full *
        MOVE.B  RXDATA,(A0)+
```

```
        MOVE.B  RXDATA,D0
        JSR     TXCHAR                  * loop back to monitor *
        ADDI.B  #1,COUNTC
        JMP     NEW

************************************
* Delete 1 Char in Command Buffer *
************************************
DELBUF  CMPI.B  #0,COUNTC
        BEQ     DELEND
        MOVE.B  #BS,D0                  * delete character *
        JSR     TXCHAR
        MOVE.B  #SPC,D0
        JSR     TXCHAR
        MOVE.B  #BS,D0
        JSR     TXCHAR
        SUBI.B  #1,COUNTC
        SUBA.L  #1,A0
DELEND  JMP     NEW

************************
* Command Buffer Full *
************************
BUFFULL MOVE.B  #BELL,D0
        JSR     TXCHAR
        JMP     NEW

************************
* Tx A String          *
* A0: Address Pointer *
************************
TXSTR   MOVE.B  (A0)+,D0
        CMPI.B  #EOS,D0
        BEQ     TXEND
        JSR     TXCHAR
        JMP     TXSTR
TXEND   RTS

*************
* Tx A Char *
*************
TXCHAR  MOVEM.L D0,-(A7)
        BTST.B  #2,SRA                  * tx ready? *
        BEQ     TXCHAR
        MOVE.B  D0,TBA
ECHO    BCLR.B  #0,NEWDATA
        BEQ     ECHO
        MOVEM.L (A7)+,D0
        RTS

*******************
* Command Analysis *
*******************
CMDANA  MOVE.B  #CR,(A0)+
        MOVE.B  #EOS,(A0)
        MOVEA.L #CMDBUF,A0
        CLR.L   D1
        CMPI.B  #CR,(A0,D1.L)           * no command entered *
        BEQ     NOCMD
        JSR     TESTSPC                 * test & skip spaces *
        CMPI.B  #'H',(A0,D1.L)          * H/h for HELP menu *
        BEQ     HELP
        CMPI.B  #'h',(A0,D1.L)
        BEQ     HELP
        CMPI.B  #'M',(A0,D1.L)          * M/m for modify memory *
        BEQ     MODMEM
        CMPI.B  #'m',(A0,D1.L)
        BEQ     MODMEM
        CMPI.B  #'B',(A0,D1.L)          * B/b for block fill memory *
        BEQ     BLKFIL
        CMPI.B  #'b',(A0,D1.L)
        BEQ     BLKFIL
        CMPI.B  #'D',(A0,D1.L)          * D/d for dump memory or *
        BEQ     DANA                    * download FPGA configuration data *
        CMPI.B  #'d',(A0,D1.L)
        BEQ     DANA
        CMPI.B  #'R',(A0,D1.L)          * R/r for reset or reprogram FPGA *
        BEQ     RANA
        CMPI.B  #'r',(A0,D1.L)
        BEQ     RANA
        CMPI.B  #'T',(A0,D1.L)          * T/t for trace mode *
        BEQ     TRMD
        CMPI.B  #'t',(A0,D1.L)
        BEQ     TRMD
        CMPI.B  #'N',(A0,D1.L)          * N/n for normal mode *
        BEQ     NMD
        CMPI.B  #'n',(A0,D1.L)
        BEQ     NMD
        CMPI.B  #'P',(A0,D1.L)          * P/p for program switch *
        BEQ     PROGSW
        CMPI.B  #'p',(A0,D1.L)
        BEQ     PROGSW
        JMP     CMDERR                  * other is invalid *

  ***********************
  * No Command Entered *
```

```
* Start over again   *
***********************
NOCMD   JMP     BEGIN

*****************************************************
* Test & Skip Spaces Between Parameters In Command *
*****************************************************
TESTSPC CMPI.B  #SPC,(A0,D1.L)          * still space? *
        BEQ     SKIP
        RTS
SKIP    ADDI.L  #1,D1
        JMP     TESTSPC

*********************
* Display HELP Menu *
*********************
HELP    ADDI.L  #1,D1
        JSR     TESTSPC
        CMPI.B  #CR,(A0,D1.L)
        BNE     CMDERR
        MOVEA.L #HELPMSG,A0
        JSR     TXSTR
        JMP     BEGIN

*************************
* Display Error Message *
*************************
CMDERR  MOVEA.L #INVCMD,A0
        JSR     TXSTR
        JMP     BEGIN

PARERR  MOVEA.L #INVPAR,A0
        JSR     TXSTR
        JMP     BEGIN

ADDRERR MOVEA.L #INVADDR,A0
        JSR     TXSTR
        JMP     BEGIN

DATAERR MOVEA.L #INVDATA,A0
        JSR     TXSTR
        JMP     BEGIN

NOTACC  MOVEA.L #ACCADDR,A0
        JSR     TXSTR
        JMP     BEGIN

SWERR   MOVEA.L #INVSW,A0
        JSR     TXSTR
        JMP     BEGIN

*****************************
* Block Fill Memory         *
* A2: Source Address        *
* A3: Destination Address   *
*****************************
BLKFIL  ADDI.L  #1,D1
        CMPI.B  #'F',(A0,D1.L)
        BEQ     BFMEM
        CMPI.B  #'f',(A0,D1.L)
        BNE     CMDERR                  * invalid command *
BFMEM   JSR     BWTARG1
        JSR     ASCTHEX                 * ASCII to hex conversion *
        CMPI.B  #$FF,D3                 * invalid hex address *
        BEQ     PARERR                  * invalid parameter *
        JSR     TESTADD                 * test accessible address *
        MOVEA.L D0,A2                   * source address *
        JSR     BWTARG2
        JSR     ASCTHEX
        CMPI.B  #$FF,D3
        BEQ     PARERR
        JSR     TESTADD
        MOVEA.L D0,A3                   * destination address *
        JSR     BWTARG2
        JSR     ASCTHEX
        CMPI.B  #$FF,D3
        BEQ     PARERR
        CMPA.L  A2,A3                   * destination < source? *
        BLT     ADDRERR                 * invalid parameter *
        CMPI.L  #$FF,D0                 * data > $FF *
        BHI     DATAERR                 * invalid data *
        JSR     BWTARG3
        JSR     INITBUF                 * enable buffer */
FILL    MOVE.B  D0,(A2)+
        CMPA.L  A3,A2
        BLE     FILL
        JSR     CLRBUF                  * disable buffer *
        JMP     BEGIN

*****************************************
* Check Valid Command between Arguments 1 *
*****************************************
BWTARG1 ADDI.L  #1,D1
        CMPI.B  #CR,(A0,D1.L)
        BEQ     PARERR
        CMPI.B  #SPC,(A0,D1.L)
```

```
        BNE     CMDERR
        ADDI.L  #1,D1
        JSR     TESTSPC
        RTS

**************************
* ASCII To Hex Conversion *
* D3: Invalid Hex if FF   *
*     Valid Hex if 00     *
**************************
ASCTHEX MOVEM.L D2/A0,-(A7)
        CLR.L   D0
        CLR.L   D3
GETASC  MOVE.B  (A0,D1.L),D2
        CMPI.B  #SPC,D2                 * if space or *
        BEQ     CONEND
        CMPI.B  #CR,D2                  * carriage return or *
        BEQ     CONEND
        CMPI.B  #EOS,D2                 * end of string, end of conversion*
        BEQ     CONEND
        CMPI.L  #6,D3                   * address > $FFFFFF *
        BEQ     NOTHEX                  * not valid *
        CMPI.B  #'0',D2
        BLT     NOTHEX
        CMPI.B  #'9',D2
        BLS     CONVERT
        CMPI.B  #'A',D2
        BLT     NOTHEX
        CMPI.B  #'F',D2
        BLS     CONVERT
        CMPI.B  #'a',D2
        BLT     NOTHEX
        CMPI.B  #'f',D2
        BHI     NOTHEX
        SUBI.B  #$20,D2                 * a - f *
CONVERT SUBI.B  #'0',D2
        CMPI.B  #9,D2                   * 0 - 9 *
        BLS     GETHEX
        SUBI.B  #7,D2                   * A - F *
GETHEX  LSL.L   #4,D0
        ANDI.L  #$0F,D2
        ADD.L   D2,D0
        ADDI.L  #1,D1
        ADDI.L  #1,D3
        JMP     GETASC
CONEND  CMPI.B  #0,D3
        BEQ     NOTHEX
        CLR.L   D3
        MOVEM.L (A7)+,D2/A0
        RTS
NOTHEX  MOVE.L  #$FF,D3
        MOVEM.L (A7)+,D2/A0
        RTS

*****************************************
* Check Valid Command between Arguments 2 *
*****************************************
BWTARG2 JSR     TESTSPC
        CMPI.B  #CR,(A0,D1.L)
        BEQ     PARERR
        RTS

*****************************************
* Check Valid Command between Arguments 3 *
*****************************************
BWTARG3 JSR     TESTSPC
        CMPI.B  #CR,(A0,D1.L)
        BNE     PARERR
        RTS

***********************
* Modify Memory       *
* A1: Memory Location *
***********************
MODMEM  ADDI.L  #1,D1
        CMPI.B  #'M',(A0,D1.L)
        BEQ     MODIFY
        CMPI.B  #'m',(A0,D1.L)
        BNE     CMDERR
MODIFY  JSR     BWTARG1
        JSR     ASCTHEX
        CMPI.L  #$FF,D3
        BEQ     PARERR
        JSR     TESTADD                 * test accessible address *
        MOVEA.L D0,A1                   * memory location *
        JSR     BWTARG2
        JSR     ASCTHEX
        CMPI.L  #$FF,D3
        BEQ     PARERR
        CMPI.L  #$FF,D0                 * data > $FF *
        BHI     DATAERR                 * invalid data *
        JSR     BWTARG3
        JSR     INITBUF                 * enable buffer *
        MOVE.B  D0,(A1)                 * write data *
        JSR     CLRBUF                  * disable buffer *
        JMP     BEGIN
```

```
*****************************
* Test Accessible Address *
*****************************
TESTADD CMPI.L  #$807FF,D0              * $0-807ff unaccessible *
        BLS     NOTACC
        CMPI.L  #$80FFF,D0              * $80800-80fff accessible *
        BLS     VALADDR
        CMPI.L  #$BFFFF,D0              * $81000-bffff unaccessible *
        BLS     NOTACC
        CMPI.L  #$13FFFF,D0             * $C0000-13ffff accessible *
        BLS     VALADDR                 * initial buffer needed *
        JMP     NOTACC                  * others unaccessible *
VALADDR RTS

************************************************
* Initialize Buffer between 68k & sharing RAMs *
************************************************
INITBUF MOVE.B  #$0,BUFFER
        MOVE.B  #$0,RAMRW
        RTS

*******************************************
* Clear Buffer between 68k & sharing RAMs *
*******************************************
CLRBUF  MOVE.B  #$0,BUFFER
        RTS

*************************************
* D: Dump Memory or                 *
*    Download FPGA Configuration Data *
*************************************
DANA    ADDI.L  #1,D1
        CMPI.B  #'M',(A0,D1.L)          * M/m for dump memory *
        BEQ     DUMPMEM
        CMPI.B  #'m',(A0,D1.L)
        BEQ     DUMPMEM
        CMPI.B  #'F',(A0,D1.L)          * F/f for download FPGA *
        BEQ     DOWNLD                  * configuration data *
        CMPI.B  #'f',(A0,D1.L)
        BEQ     DOWNLD
        CMPI.B  #'S',(A0,D1.L)          * S/w for download switch data *
        BEQ     DOWNSW
        CMPI.B  #'s',(A0,D1.L)
        BEQ     DOWNSW
        JMP     CMDERR                  * others is invalid *

*********************
* Dump Memory       *
* D2: Column count  *
* D3: Row count     *
*********************
DUMPMEM JSR     BWTARG1
        JSR     ASCTHEX
        CMPI.B  #$FF,D3
        BEQ     PARERR
        JSR     TESTADD
        ANDI.L  #$00FFFF00,D0
        MOVEA.L D0,A1                   * memory location *
        JSR     BWTARG3
        JSR     INITBUF                 * enable buffer *
        MOVEA.L #HEADING,A0
        JSR     TXSTR
DSPMEM  MOVE.L  #20,D1
        JSR     PRTHEX
DUMP    MOVE.L  #$0,D2
        MOVEA.L D0,A0
        MOVE.B  #SPC,D0
        JSR     TXCHAR
        JSR     TXCHAR
        MOVE.B  #COLN,D0
        JSR     TXCHAR
        MOVE.B  #SPC,D0
        JSR     TXCHAR
        JSR     TXCHAR
        MOVE.L  #4,D1
COLUMN  MOVE.B  (A0,D2),D0
        JSR     PRTHEX
        MOVE.B  #SPC,D0                 * next column *
        JSR     TXCHAR
        JSR     TXCHAR
        CMPI.L  #$0F,D2                 * column completed? *
        BEQ     ROW
        ADDI.L  #1,D2
        JMP     COLUMN
ROW     MOVE.B  #CR,D0                  * next row *
        JSR     TXCHAR
        MOVE.B  #LF,D0
        JSR     TXCHAR
        CMPI.L  #$0F,D3                 * dump all? *
        BEQ     TABLEND
        ADDI.L  #1,D3
        MOVE.L  A0,D0
        ADDI.L  #$10,D0
        JMP     DSPMEM
TABLEND MOVEA.L #ENDING,A0
```

```
        JSR     TXSTR
        JSR     CLRBUF                  * disable buffer *
        JMP     BEGIN

************************************
* Hex to ASCII Conversion & Display *
* D1: # of Shift                   *
* D0: HEX                          *
************************************
PRTHEX  MOVEM.L D0/D1,-(A7)
HEXTASC MOVE.L  D0,-(A7)
        LSR.L   D1,D0
        ANDI.B  #$0F,D0
        ADDI.B  #'0',D0
        CMPI.B  #'9',D0
        BLS     PRT                     * 0 - 9 *
        ADDI.B  #7,D0                   * A - F *
PRT     JSR     TXCHAR
        MOVE.L  (A7)+,D0
        CMPI.B  #0,D1
        BEQ     PRTALL
        SUBI.L  #4,D1
        JMP     HEXTASC
PRTALL  MOVEM.L (A7)+,D0/D1
        RTS

****************************************
* Downloading FPGA Configuration Data & *
* switch connection                    *
* D0: ASCII                            *
* D2: HEX                              *
* D4: FPGA Data if $0                  *
*     Switch Data if $1                *
****************************************
DOWNLD  ADDI.L  #1,D1
        JSR     BWTARG3
        CLR.L   D2                      * final HEX value *
        CLR.L   D3                      * ASCII count *
        CLR.L   D4                      * initialization *
        MOVEA.L #LCA0,A1
        MOVE.B  #$FE,LATCH              * reprogram FPGA *
        MOVE.B  #$FF,LATCH
NEWCON  BCLR.B  #0,NEWDATA
        BEQ     NEWCON                  * wait until config. data received *
        ANDI.L  #$FF,D0
        CMPI.L  #'0',D0
        BLT     NEWCON
        CMPI.L  #'9',D0                 * 0 - 9 *
        BLS     NUMBER
        CMPI.L  #'A',D0
        BLT     NEWCON
        CMPI.L  #'F',D0                 * A - F *
        BLS     CAPATF
        CMPI.L  #'a',D0
        BLT     NEWCON
        CMPI.L  #'f',D0                 * a - f *
        BLS     SMLATF
        CMPI.L  #$FF,D0
        BLT     NEWCON
        CMPI.L  #$0,D4
        BEQ     FPGAEND                 * download FPGA data completed *
        MOVE.B  #$FD,LATCH              * restart FPGAs *
        MOVE.B  #$FF,LATCH
        JMP     BEGIN                   * download switch data completed *
NUMBER  SUBI.B  #$30,D0
        JMP     HEX
CAPATF  SUBI.B  #$37,D0
        JMP     HEX
SMLATF  SUBI.B  #$57,D0
HEX     ADDI.L  #1,D3
        LSL.L   #4,D2
        ANDI.L  #$F,D0
        ADD.L   D0,D2
        CMPI.L  #2,D3
        BEQ     DL
        JMP     NEWCON
DL      CMPI.L  #$0,D4
        BEQ     DLDATA                  * download FPGA data *
        MOVE.B  D2,(A1)+                * download switch data *
        JMP     DLCON
DLDATA  MOVE.W  D2,(A1)
DLCON   CLR.L   D2
        CLR.L   D3
        JMP     NEWCON
FPGAEND MOVE.B  #$0,BUFFER
TESTSW  BCLR.B  #0,NEWDATA
        BEQ     TESTSW
        ANDI.L  #$FF,D0
        CMPI.L  #$AA,D0                 * download switch data signal *
        BEQ     SW
        JMP     BEGIN
SW      MOVE.L  #$1,D4
        CLR.L   D2
        CLR.L   D3
        MOVEA.L #SWFIRST,A1             * first switch address *
        MOVE.B  #$FB,LATCH              * initialize switch *
```

```
        MOVE.B  #$FF,LATCH
        JMP     NEWCON

************************
* Download Switch data *
************************
DOWNSW  ADDI.L  #1,D1
        JSR     BWTARG3
        JMP     SW

*********************
* R: Reset or       *
*    Reprogram FPGA *
*********************
RANA    ADDI.L  #1,D1
        CMPI.B  #'S',(A0,D1.L)          * S/s for reset FPGA *
        BEQ     RESET
        CMPI.B  #'s',(A0,D1.L)
        BEQ     RESET
        CMPI.B  #'P',(A0,D1.L)          * P/p for reprogram FPGA *
        BEQ     REPMG
        CMPI.B  #'p',(A0,D1.L)
        BEQ     REPMG
        JMP     CMDERR                  * other is invalid *

**************
* Reset FPGA *
**************
RESET   ADDI.L  #1,D1
        JSR     BWTARG3
        MOVE.B  #$FD,LATCH
        MOVE.B  #$FF,LATCH
        JMP     BEGIN

******************
* Reprogram FPGA *
******************
REPMG   ADDI.L  #1,D1
        JSR     BWTARG3
        MOVE.B  #$FE,LATCH
        MOVE.B  #$FF,LATCH
        JMP     BEGIN


**************
* Trace Mode *
**************
TRMD    ADDI.L  #1,D1
        CMPI.B  #'M',(A0,D1.L)
        BEQ     TRACE
        CMPI.B  #'m',(A0,D1.L)
        BNE     CMDERR
        ADDI.L  #1,D1
        JSR     BWTARG3
TRACE   MOVE.B  #$0,MUXTM
        JMP     BEGIN

***************
* Normal Mode *
***************
NMD     ADDI.L  #1,D1
        CMPI.B  #'M',(A0,D1.L)
        BEQ     NORMAL
        CMPI.B  #'m',(A0,D1.L)
        BNE     CMDERR
        ADDI.L  #1,D1
        JSR     BWTARG3
NORMAL  MOVE.B  #$0,MUXNM
        JMP     BEGIN

*************************
* Program Switch        *
* D4: Lower Switch if $0 *
*     Upper Switch if $1 *
*************************
PROGSW  ADDI.L  #1,D1
        CMPI.B  #'S',(A0,D1.L)
        BEQ     PROG
        CMPI.B  #'s',(A0,D1.L)
        BNE     CMDERR
PROG    JSR     BWTARG1
        JSR     ASCTHEX
        CMPI.L  #$FF,D3
        BEQ     PARERR
        JSR     TESTVSW                 * test valid switch position *
        MOVEA.L D0,A1                   * switch position *
        JSR     BWTARG2
        JSR     ASCTHEX
        CMPI.L  #$FF,D3
        BEQ     PARERR
        CMPI.L  #$1,D4
        BEQ     UPPERSW                 * upper switch *
        CMPI.L  #$FF,D0                 * lower switch, data > $FF *
        BHI     DATAERR                 * invalid data *
        JMP     PROGCON
UPPERSW CMPI.L  #$3,D0                  * data > $3 *
```

```
        BHI     DATAERR                 * invalid data *
PROGCON JSR     BWTARG3
        MOVE.B  D0,(A1)                 * program switch *
        JMP     BEGIN

*******************************
* Test valid switch position *
*******************************
TESTVSW CLR.L   D4                      * initialization *
        CMPI.L  #$140000,D0             * $0-140000 invalid switch positions *
        BLT     SWERR
        CMPI.L  #$14007F,D0             * $140000-14007F valid switch positions *
        BLS     TESTUPP                 * test upper/lower switch *
        JMP     SWERR                   * others invalid *
TESTUPP BTST.L  #0,D0
        BNE     TESTEND                 * lower switch *
        MOVE.L  #$1,D4                  * upper switch *
TESTEND RTS

*********************************
* IRQ 6: RS-232C Communication *
*********************************
        ORG     INT6
        MOVE.B  RBA,D0
        MOVE.B  #$1,NEWDATA
        RTE
```

APPENDIX I

UPB Command Line Arguments

Command	Arguments Required	Description
MM	<ADDR.> <DATA>	Modify Memory: modify a single memory location ADDR. with DATA
BF	<SOUR. ADDR.> <DEST. ADDR.> <DATA>	Block Fill: modify a range of memory locations from SOUR. ADDR to DEST. ADDR. with DATA
DM	<ADDR>	Dump Memory: dump 16 x 16 data including memory location ADDR.
DF	<filename> (enter after prompted)	Download FPGA configuration data
DS	<filename>.con (enter after prompted)	Download Switch connection data
RP	--	Reprogram FPGAs
RS	--	Reset FPGAs
TM	--	Trace Mode: single step the design for debugging
NM	--	Normal Mode: normal operation after dwonloading all data
H	--	Help menu displayed