2006

A Field Programmable Gate Array Architecture for Two-Dimensional Partial Reconfiguration

Fei Wang
Wright State University

A Field Programmable Gate Array Architecture for Two-Dimensional Partial Reconfiguration

Dissertation in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

By

Fei Wang

Bachelor, Shandong University of Technology, 1989
Master, Beijing University of Aeronautics and Astronautics, 1994

2006

Wright State University

Wright State University

School of Graduate Studies

November 8, 2006

I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER MY SUPERVISION BY Fei Wang ENTITLED A Field Programmable Gate Array Architecture for Two-Dimensional Partial Reconfiguration BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Doctor of Philosophy**.**

_____

Jack S. N. Jean, Ph.D.,
Dissertation Director

_____

Nikolaos Bourbakis, Ph.D., Director,
Computer Science and Engineering
Ph.D. Program

_____

Joseph F. Thomas, Jr., Ph.D.,
Dean, School of Graduate Studies

Committee on
Final Examination

_____
Jack Jean, Ph.D.

_____
T.K. Prasad, Ph.D.

_____
Henry Chen, Ph.D.

_____
Travis Doom, Ph.D.

_____
John Loomis, Ph.D.

Abstract

Wang, Fei. Ph.D., Department of Computer Science and Engineering, Wright State University, 2006. A Field Programmable Gate Array Architecture for Two-Dimensional Partial Reconfiguration

Reconfigurable machines can accelerate many applications by adapting to their needs through hardware reconfiguration. Partial reconfiguration allows the reconfiguration of a portion of a chip while the rest of the chip is busy working on tasks. Operating system models have been proposed for partially reconfigurable machines to handle the scheduling and placement of tasks. They are called OS4RC in this dissertation.

The main goal of this research is to address some problems that come from the gap between OS4RC and existing chip architectures and the gap between OS4RC models and practical applications. Some existing OS4RC models are based on an impractical assumption that there is no data exchange channel between IP (Intellectual Property) circuits residing on a Field Programmable Gate Array (FPGA) chip and between an IP circuit and FPGA I/O pins. For models that do not have such an assumption, their inter-IP communication channels have severe drawbacks. Those channels do not work well with 2-D partial reconfiguration. They are not suitable for intensive data stream processing. And frequently they are very complicated to design and very expensive. To address these problems, a new chip architecture that can better support inter-IP and IP-I/O communication is proposed and a corresponding OS4RC kernel is then specified.

The proposed FPGA architecture is based on an array of clusters of configurable logic blocks, with each cluster serving as a partial reconfiguration unit, and a mesh of segmented buses that provides inter-IP and IP-I/O communication channels. The

proposed OS4RC kernel takes care of the scheduling, placement, and routing of circuits under the constraints of the proposed architecture. Features of the new architecture in turns reduce the kernel execution times and enable the runtime scheduling, placement and routing. The area cost and the configuration memory size of the new chip architecture are calculated and analyzed. And the efficiency of the OS4RC kernel is evaluated via simulation using three different task models.

Table of Contents

List of Figures

List of Tables

Acknowledgement

First I highly appreciate those valuable contributions from my advisor, Professor Jack Jean. Without his continues guidance and support, it is impossible for me to finish this dissertation. In the past five years, I benefited a lot from those discussions with my advisor. During our discussions, interesting topics were identified, and promising methods were initiated, and mistakes were removed. Via discussion, a lot of difficulties were made not so difficult. Sometimes our discussion looked like arguing because of our different perceptions. He showed enough patience to convince me. His adroit communication skills were so impressive. His generous help to me not only enhanced my academic research, but also many other perspectives. When I did not run my teaching assistant job smoothly, he contributed effective suggestions to improve.

I also thank other professors on my committee: Professor Henry Chen, Professor T.K. Prasad, Professor Travis Doom, and Professor John Loomis. Using their profound knowledge and experience, they contributed a lot on my proposal design and progress evaluations.

This thesis also benefits a lot from professors outside my committee: Professor Marteen Rizki and Professor Thomas Sudkamp. Some experimental framework and algorithms were improved after helpful electronic mail correspondences or walk-in office talks. In the time of a shrinking world, I was able to talk with many practitioners in the community of reconfigurable computing via emails. They are spread in different

universities and companies both inside and outside of United States, even though I did not list them here.

I appreciate financial support from the Department of computer Science and Engineering of Wright State University and DAGSI (Dayton Area Graduate School Institute) board. In a very competitive environment, I am a lucky one to get continuous financial support to finish my degree.

This thesis concludes my Ph.D. program, but it is also a new starting point of my research career. The later I believe is more important.

Wang, Fei

June 2006

Dayton, Ohio, USA

1 Introduction

For the past 20 years, Reconfigurable Computing has attracted huge amount of research interest due to its ability to accelerate a wide variety of demanding computations. A. J. Elbirt et al. showed that an FPGA (Field Programmable Gate Array) implementation of Serpent Block Cipher had a speedup of 18 compared with a software version running on a Pentium PC [Elbirt00]. Michael Rencher et al. demonstrated a 100 times speedup of FPGA over an HP workstation in an automatic target recognition application [Rencher97]. Other applications included: video processing [Haynes00], template matching [Jean03], Boolean satisfiability [Skliarova03], string matching [Weinhardt99] and so on.

This chapter gives a brief introduction of this dissertation. In Section 1.1, the history of reconfigurable machines is introduced. In Section 1.2, some important terminologies are explained, and problems with partial reconfiguration are presented. In Section 1.3, a new FPGA architecture and a corresponding operating system are introduced. In Section 1.4, original contributions of this research are summarized. In Section 1.5, the organization of this dissertation is listed.

1.1 Reconfigurable Machine and FPGA

Reconfigurable computing model was initially discussed by Estrin et al. in 1963 [Estrin63]. G. Estrin proposed a model called “Fixed plus Variable”. This model combined a traditional processor with a fine-grained reconfigurable engine. Today, many

reconfigurable machines still follow this “fixed plus variable” model, even though the relative sizes of the two parts may be varied on different machines.

The reconfigurable engine is generally in charge of those tasks that are computation intensive and with some regular computing pattern. One typical example is DSP applications. The configuration of the reconfigurable engine can be varied from task to task. While the fixed part, frequently a CPU, is in charge of those tasks which are not suitable for reconfigurable engine and management tasks of the reconfigurable engine.

FPGA was invented in the mid 1980s. It is generally composed of two layers. One layer is an array of CLBs (configurable logic blocks) and pre-fabricated routing segments between CLBs. The other layer is composed of configuration memory cells. A CLB can be configured as one or more multi-input combinatorial logic functions and/or flip-flops. Different routing segments can be connected when programmable transistors between them are turned on. Data stored in the configuration memory cells specify the internal configuration of every CLB and the on/off status of transistors between those routing segments.

Reconfigurable Computing received renewed attention when the first FPGA chip was created in 1984 at Xilinx. Even though FPGA was initially invented as a substitute to expensive ASICs (Application Specific Circuits), it is considered to be a good incarnation of reconfigurable engine because of its capability of fast reconfiguration.

1.2 Some Terminologies about FPGA Chip Architectures

Before further discussion of chip architectures, some important terminologies are explained first.

1.2.1 Single Context and Multi-context Reconfiguration

In early days, because of FPGA structure constraints, only full-chip single context configuration was allowed. Once reconfiguration is initiated, the configuration bit stream for a full chip is downloaded. The information stored in a configuration memory is entirely overwritten, even if only a portion of the configuration memory is to be updated. During the configuration period, FPGA cannot work. Many commercial FPGA chips work in this style, such as Xilinx XC4000 and Altera Flex10K.

As the capacity of FPGA increases, the configuration time cost also increases. The benefit from hardware acceleration may be overshadowed by this configuration time, which can take up to hundreds of milliseconds. Multi-context configuration and partial configuration are two effective technologies used to overcome this problem.

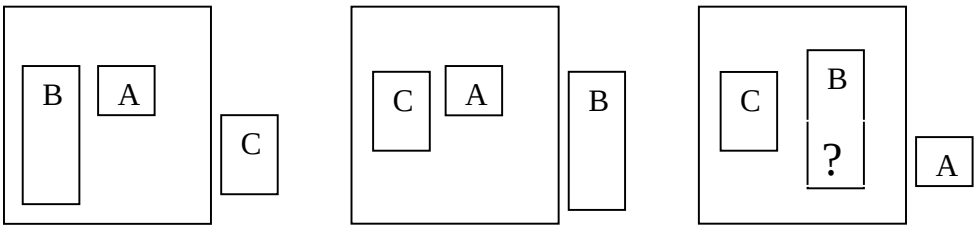
Unlike single context FPGA, a multi-context FPGA can have more than one set of configuration memory. At one moment, only one set is active, while the other sets are inactive. The behavior of the chip is determined by configuration data inside the active set. Because the configuration of inactive memory sets can be loaded in the background, also because the switch from one context to another takes only a few clock cycles, the configuration time cost can be drastically reduced.

On FPGA chips which support partial reconfiguration, the underlying configuration memory is divided into many pieces, each can be accessed independently. Configuration can be based on updating only a few pieces of configuration memory at a time. When those pieces are being updated, the rest of the chip may still be working without being disturbed. Most existing FPGAs that support partial reconfiguration are

still single context chips. The reconfiguration model discussed in this thesis is based on single context partially reconfigurable FPGAs.

1.2.2 Partial Reconfiguration and Flexible IP Placement

Let's illustrate IP (Intellectual Property) circuit or hardware task relocation using the following example. Suppose initially there are two active circuits working inside the chip: circuits A and B, as shown in Figure 1.1 (a). Operating system then decides to replace circuit B with circuit C, while circuit A is still working (see Figure 1.1 (b)). Suppose a moment later, circuit A's task is completed and circuit B is to be swapped in. If circuit B can be placed away from its initial position, as shown in Figure 1.1 (c), it is a circuit that can be relocated; otherwise, it is not.



(a) (b) (c)

Figure 1.1 IP Relocation

From the above example, IP relocation has to be supported by partial reconfiguration, while partial reconfiguration does not imply IP relocation. Obviously, IP relocation can provide users more flexibility. In the above scenario, circuit B and circuit C may run in parallel if relocation is allowed; otherwise, circuit B has to wait until circuit

C is swapped out. Unfortunately, almost no existing FPGA chip can support IP circuit relocation directly.

There are two cases of IP relocation: IP translation (if IP can only have fixed orientation) and IP rotation (if IP do not have to keep fixed orientation). Obviously, IP rotation can provide additional flexibility, as an IP may not fit into a chip where its orientation is changed [Compton00].

To support IP relocation on a chip, some requirements have to be satisfied:

(1) Sub-area of FPGA (a CLB or a chunk of CLBs) can be accessed separately.

(2) Partial configuration bit stream has to be address-independent. It should be possible to use configuration data for CLB at one specific position to configure CLB at another position. Only when address information can be separated from logic information, it is possible to effectively translate hardware circuit from one location to another location. Xilinx Virtex chips are partially reconfigurable, but they do not support circuit relocation directly. But after some modification on bit streams, they can support circuit relocation under some conditions [Horta02].

(3) FPGA chip internal resources have to be homogenously distributed. On Xilinx Virtex chips, block RAMs are located only at some limited locations. For IP circuits that have to work with block RAMs, the locations they can be relocated to are quite limited.

(4) Symmetrical architecture is required to support IP rotation only. A CLB output /input pin at one side needs a substitute pin at each of the three other sides except some specific pins. These specific pins may include clock pins, global set/reset pins, as they are hooked on dedicated routing wires. And programmable connection points

between individual CLB and nearby routing channels also have to be symmetrical. Otherwise an IP may have difficulty in connecting with routing resources after rotation.

1.2.3 Problems with IP Relocation

Even above requirements are satisfied, there are still limitations to IP relocation:

(1) Inter IP communication problem

For a long time, it has been assumed that no communication exists between IP circuits residing on an FPGA chip. So their swap in (out) of FPGA has no impact on each other. But in many cases, such communications do exist. Recently researchers start to pay attention to this problem.

(2) IO problem

Sometimes an IP circuit has to be connected to fixed IO pads on the FPGA peripheral. For example, a USB interface chip may be mounted at a fixed position nearby an FPGA chip. Whenever an IP circuit inside an FPGA chip communicates with this USB chip, it has to connect to those IO pads.

(3) IP circuit relocation with block RAMs:

In FPGA chips, block RAMs are dedicated resources. But in most cases, they are available at only a few fixed locations. If an IP circuit has to work with block RAMs, then its placement options are limited. To address this problem, block RAMs should be uniformly distributed.

1.3 Proposed Architecture and Operating System

To address above problems, a new chip architecture is proposed, which has two distinct properties:

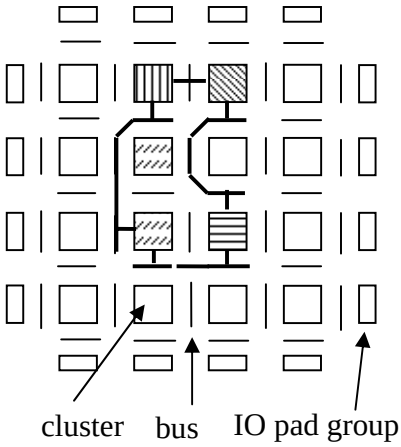


Figure 1.2 Cluster-Segmented Bus Structure of the New Chip

(1) 2D partial reconfiguration is supported, and the minimum partial reconfiguration unit is a square cluster of CLBs. Compared with 1D and non-square partial reconfiguration architecture (e.g., Xilinx Virtex-II chips [Xilinx03] [Xilinx04]), the square cluster unit improves chip utilization [Wang05][Keating02].

(2) Segmented buses are introduced around clusters so as to support inter-IP and IP-IO communications. These buses are used solely for runtime support. They are not used for intra-IP connections, which are still supported by traditional FPGA routing resources. At each intersection of bus segments, there is a switch block to relay bus connections, which is different from traditional switch blocks inside clusters. Different IP circuits can be connected together via bus segments and bus switch blocks.

(3) An operating system kernel working along with the new architecture is proposed. For each newly arrived IP circuits, the operating system schedules and allocates a spatial slot on the chip, which does not have to be available right away. The slots may be occupied by one or more running circuits at the moment. Necessary physical connections are built up among on-chip circuits that need to communicate with each other.

1.4 Original Contributions

The original contributions of this dissertation can be listed as follows:

(1) With the proposed architecture, an FPGA chip is partitioned into many square clusters, with each cluster being the minimum partial reconfiguration unit. This structure is new, although 2D partial reconfiguration is not new. (On XC6000, partial reconfiguration is practiced on the CLB level).

(2) Segmented buses are proposed to be used for runtime inter-IP or IP-IO connections. They are faster and less complicated than using packet switching, as some researchers proposed previously.

(3) Based on the new chip architecture, an operating system kernel is designed and implemented. Compared with many existing operating system prototypes, this kernel has an on-line router that is dedicated to interconnections between on-chip IP circuits. This is different from previous routers that are used for CLB level routings. The efficiency of this on-line router is demonstrated in Chapter 6.

(4) More realistic task models are proposed to evaluate the performance of the operating system. For task models used in many previous works, a computing task is carried out by only one IP circuit whose execution time is assumed to be known in advance.

Those assumptions are relaxed so that a task may be carried out by more than one circuit, with interconnections among them, and the execution time of some circuits may be unknown when they are scheduled.

1.5 Dissertation Outlines

This thesis is organized as follows.

In Chapter 2, a literature survey on reconfigurable computing is given. The survey covers chip structure, reconfigurable machine architecture, runtime operating systems and programming of reconfigurable machines.

In Chapter 3, aspect ratio issues of circuits are discussed. Simulation results with MCNC benchmark circuits indicate that partial reconfiguration based on square clusters is more efficient than current approaches based on CLBs or based on columns of CLBs.

In Chapter 4, the proposed new chip architecture is described, which can support inter-IP and IP-IO communications. The extra area cost associated with the communication infrastructure is estimated.

In Chapter 5, an operating system kernel targeting the new chip architecture is presented. Algorithms corresponding to different components of the operating system kernel are specified.

In Chapter 6, under different parameter settings and task models, the performance of the operating system kernel is evaluated via simulations.

In Chapter 7, the thesis is concluded, and future works are considered. In Chapter7, as future works, a 3D FPGA architecture is briefly introduced which may be

used to incorporate block memories in the proposed architecture. Block memories are not considered in this dissertation.

2 Literature Survey

This chapter gives a summary to reconfigurable machines. In Section 2.1, FPGA chip architectures are discussed. In Section 2.2, some works on runtime operating systems are summarized. In Section 2.3, the programmability of reconfigurable machines is discussed. Even though Section 2.3 is not closely related to this dissertation, it is included to make a more complete overview of reconfigurable machines.

2.1 FPGA Chip Architectures

Many FPGA chip architectures were created in the past twenty years. This section summarizes the architecture of XC4000, XC6200, DPGA and AT6000/AT40k. They were selected because of their unique features. Although XC6200 was not a success commercially and DPGA has never been commercialized, their partial reconfiguration supports are innovative in their days.

2.1.1 XC4000

Xilinx XC4000 chips are a classic device family. Many other products, such as Xilinx's Virtex and Spatan, Altera's Stratix and Flex, and Lucent's ORCA FPGA, have similar architectures. So XC4000 is briefly reviewed here.

Generally, an FPGA chip is composed of two layers: the upper logic layer and lower configuration memory layer. The logic layer can be represented as an array of programmable cells. Between these cells, there are routing channels. Functions of every

cell can be configured, and cells can be connected in different ways if appropriate programmable transistor switches along the connection route are turned on. Those configuration data for every cell and every switch are stored in the configuration memory layer. Different manufacturers have their own terminologies on those cells. Xilinx uses CLB (configurable logic block), while Altera uses LAB (logic array block).

Cells of different products can have different granularities in terms of number of **BLE** (Basic logic element) inside the cell. A **BLE** (Basic logic element) is defined as a lookup table (commonly with four inputs), plus a register, as shown in Figure 2.1.

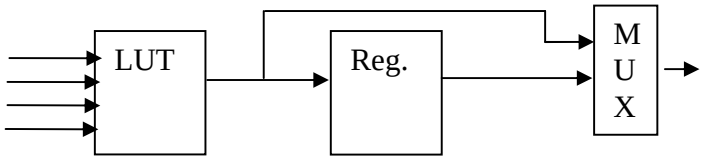


Figure 2.1 A BLE [Betz99]

In XC4000, every CLB is composed of two BLEs. While in Virtex chips, every CLB is composed of 4 BLEs, each Altera Stratix LAB consists of 10 BLEs [Stratix03]. Figure 2.2 shows a CLB with multiple BLEs locally interconnected. As shown in Figure 2.3, a XC4000 CLB is composed of two 4-input look-up tables (F and G), two flip-flops and some multiplexers. Connections between CLBs have to go through dedicated routing channels outside CLBs.

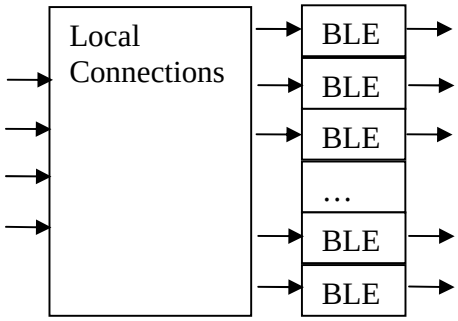


Figure 2.2 A CLB [Betz99]

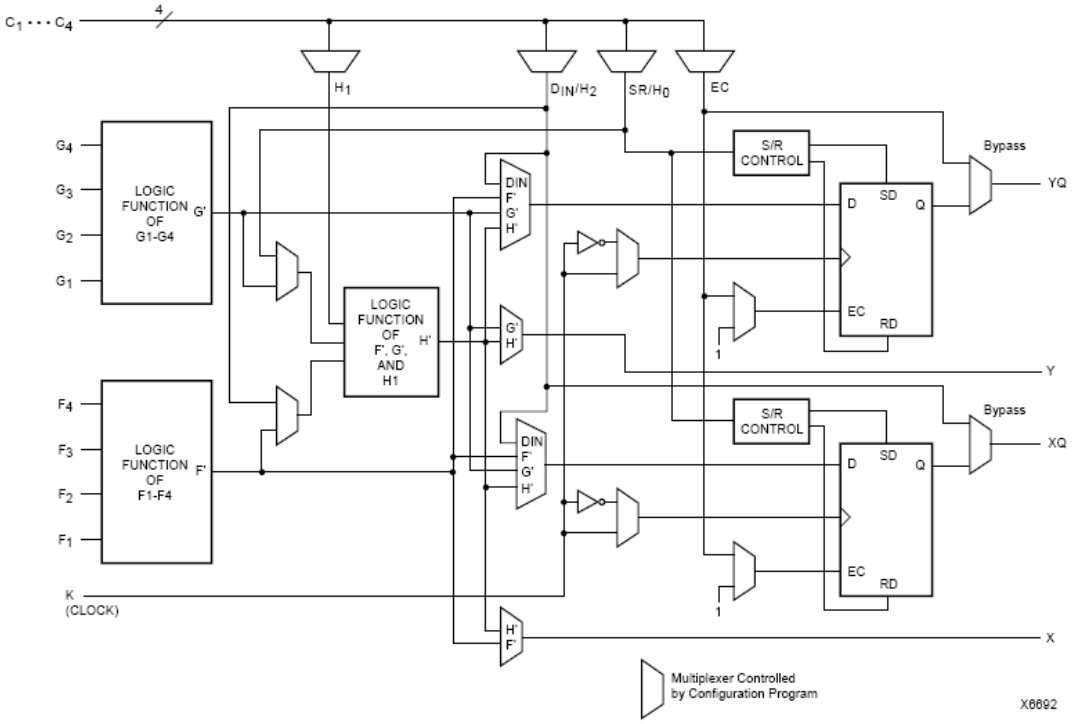


Figure 2.3 Detail of a XC4000/XL Block [Xilinx94]

2.1.2 XC6200

An XC6200 cell is shown in Figure 2.4 [Xilinx96]. Inside each cell, there is a function unit, i.e., a flip-flop and a look-up table. The look-up table has three input-signals, X_1, X_2 and X_3, coming from three input-multiplexers. Each multiplexer has eight possible inputs: N, S, E, W, N4, S4, E4, and W4. One of them is selected to be connected to the function unit input. Each cell has four outputs: $N_{out,} S_{out,} E_{out,} W_{out}$. Any of them can take the output of the function unit or input signals from neighbor cells. For example, N_{out} may be one out of the values from N, E, W, F. So an XC6200 has some routing capability to directly connect with nearby cells. This is a big difference from an XC4000 cell. When all the four output multiplexers are used for routing signals, then the internal function unit is wasted.

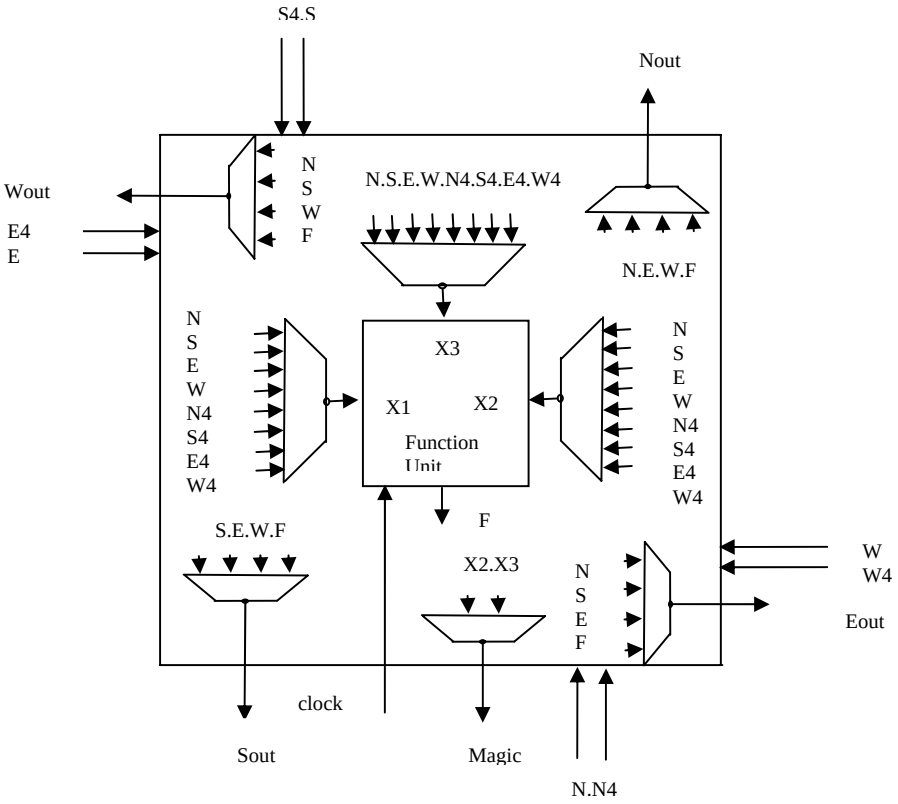


Figure 2.4 Detail of a XC6200 Cell [Xilinx96]

Routing resources of XC6000 (including XC6200) chips are organized in a hierarchical manner [Xilinx96]. Here an XC6216 chip as shown in Figure 2.5 is used as an example to illustrate this organization. At the bottom of the hierarchy, every CLB has direct interconnections with immediate neighbors.

The next level of hierarchy is composed of (4*4) CLB blocks. At the boundary of such a (4*4) CLB block, there are routing switches. The length-4 wires are driven by special routing multiplexers within those routing switches at the boundary of (4*4) blocks. Those length-4 wires, running across CLBs inside the (4*4) blocks, supply those N4, S4,

E4 and W4 inputs to every CLB input multiplexers. At the third level, (4*4) blocks can be used as elements to build a (16*16) cell block. At boundaries of each (16*16) cell block array, length-16 wires are provided to run across the (16*16) cell block. Similarly (64* 64) cell array can be built, and chip-length wires are constructed.

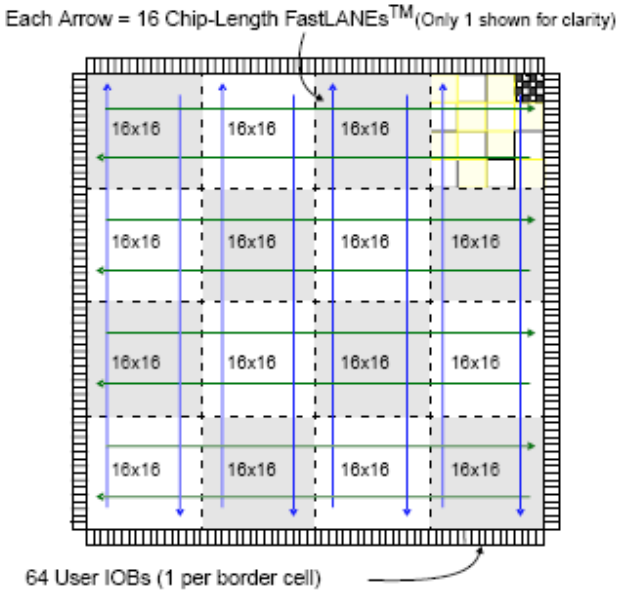


Figure 2.5 Hierarchical Organization of XC6200 [Xilinx96]

XC6200 has certain number of control registers, which can be accessed by an external processor via CPU Interface. By writing different control words to those registers, some special operations can be applied to XC6200:

(1) The output of any cell's functional unit can be read out, and the flip-flop within any cell can be written. The accesses are managed by the built-in control store interface. Through the Map Register, the XC6200 provides a mechanism for selecting some cell outputs from a column and mapping them onto the 8, 16 or 32-bit external data bus.

(2) XC6200 has a full CPU interface, referred to as FastMap™, which makes configuration SRAMs and logic cells appear as conventional memory. Via appropriate register (WildCard registers) operations, configuration memory data for all those CLBs within the same row/column can be written simultaneously.

Because of its hierarchical routing structure, when two 4*4 (16*16) blocks are not close to each other, longer wires (length 4 or length 16) are used to connect them if needed. When similar communications are competing for longer wires, these long wires may become hot spots.

Even though XC6000 chips support partial reconfiguration, the cell-by-cell reconfiguration granularity is too fine. At times of mapping IPs onto chips, users focus more on inter-IP connections rather than details of each cell. Cell-level rerouting is frequently very time-consuming if quite a lot of cells are reconfigured; reconfigurable computing is not attractive when the time to place and route an IP core is longer than its execution time. Changing inter-cell connections is not an automatic placement and routing procedure and is frequently a dangerous operation. This situation can be found later in the JBits (not targeted to XC6000) toolkit.

With routing capability combined into the logic cell, many derived structures were proposed. Similar structure can also be found in Atmel's AT40K, AT6000 and other structures [Atmel99] [Compton00][Silviu01a]. Some of them have symmetric structures, and may be able to support IP circuit rotation.

2.1.3 DPGA

DPGA was fabricated at MIT in 1995. It is the first chip to implement a multi-context switching. Unlike common single context FPGAs, DPGA can have four different context configuration memories. Every time only one out of the four contexts is active, while the other three are idle. Current FPGA chip functions are determined by the active context. Switching between different contexts is controlled by a 2-bit context identifier. While the active context is working, idle contexts can be loaded with new configuration bit streams. The reconfiguration time cost can be reduced.

DPGA uses a "Delta architecture", "which is designed to fully exploit the high level of symmetry in each of its sub-component hierarchies, by replicating to form larger homogeneous logic blocks" [Dehon96]. It has totally three levels of component hierarchies:

(1) The leaf level node is a 4-input lookup table (LUT) with an optional flip-flop, or a BLE, which is defined as an *array element*.

(2) Sixteen (4*4) array elements are combined into a *sub-array* with dense programmable local interconnection resources.

(3)At the chip level, nine (3*3) sub-arrays are connected by crossbars. Communication at the edge of the sub-arrays goes off chip via programmable I/O pins. Figure 2.6 shows the three top-level hierarchies of DPGA's architecture.

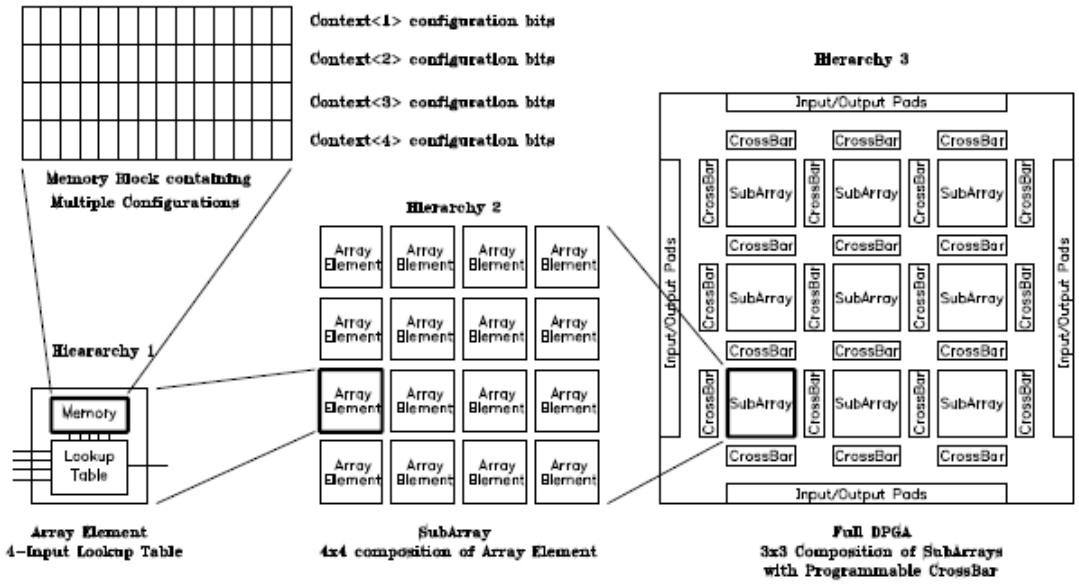


Figure 2.6 High Level Architecture of DPGA [Dehon96]

2.1.4 Atmel AT6000/AT40K

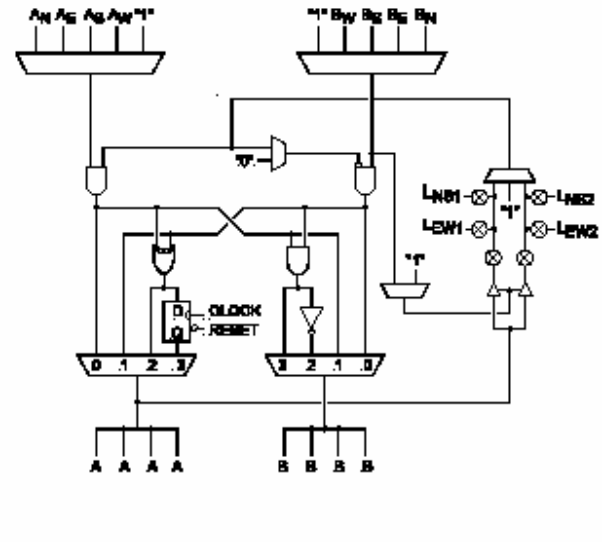


Figure 2.7 Detail of an AT6000 Cell [Atmel99]

Figure 2.7 shows a cell of AT6000. It has two sets of inputs from its immediate NEWS neighbors (A_N, A_E, A_W, A_S and B_N, B_E, B_W, B_S) and four outputs (As and Bs) to its

four immediate neighbors. As shown in Figure 2.8, every (8*8) array of cells composes a sector, which is enclosed by connection units called repeaters. Inside each routing channel, there is a local bus and an express bus. Cells are hooked on local bus.

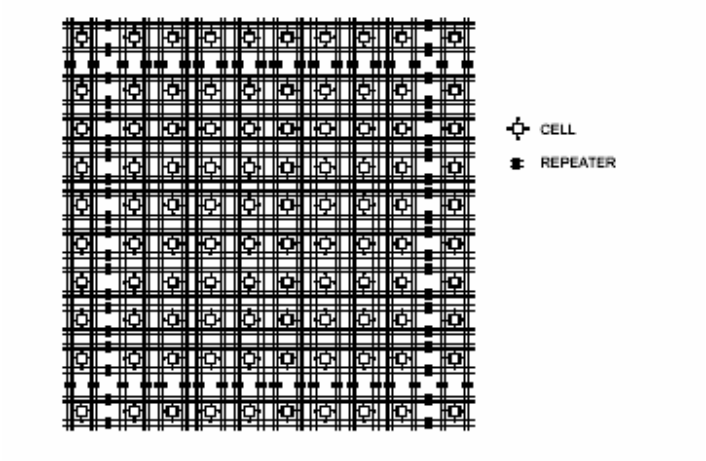


Figure 2.8 Bus Network of AT6000 [Atmel99]

The cell structure of AT40K (shown in Figure 2.9) is similar to that of AT6000. But each AT40K cell has direct outputs and inputs with its 8 immediate neighbors, and is hooked on 5 horizontal local bus wires and 5 vertical local bus wires. A sector of AT40K is composed of (4*4) cells [Atmel99].

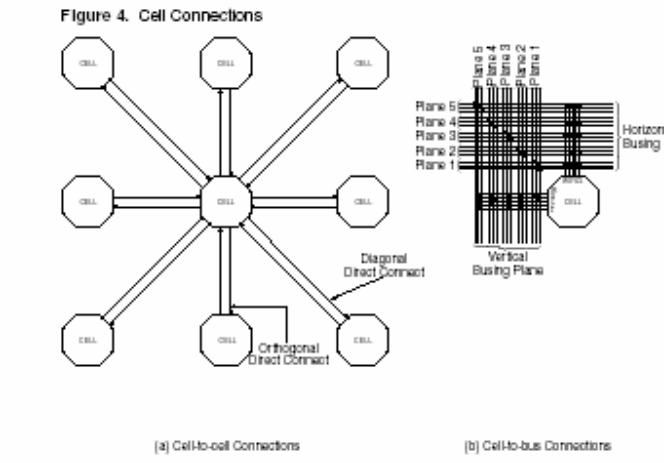


Figure 2.9 Detail of AT40K Cell [Atmel02]

Both AT40K and AT6000 are claimed to support dynamic partial reconfiguration. Like XC6200, they support cell level partial reconfiguration. The problems of cell-level partial reconfiguration have been discussed earlier in Section 2.1.2. Because of the fully symmetric structure of AT6000, IP cores can be put into an array with a rotated orientation. But this feature does not guarantee correct connections between IP cores after some of them are rotated.

2.1.5 Three-Dimensional FPGAs

Most existing FPGAs are called 2-D FPGAs, because their configuration logic blocks and routing resource are arranged on a 2-D plane. Generally they are composed of two layers: the configurable resources layer (CLBs and routing resources) and configuration memory layer. If an FPGA chip has more than one layer of configurable resources, it is called a 3-D FPGA chip.

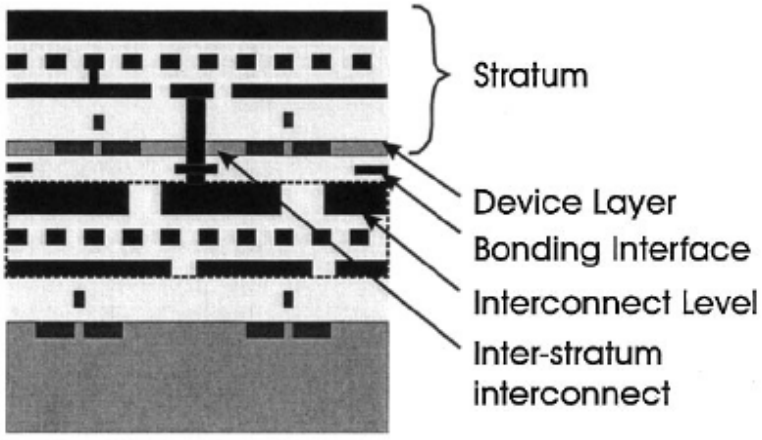


Figure 2.10 Cross Section of a 3-D FPGA Based on Wafer Bounding [Rahman03]

3-D FPGAs benefit from recent semiconductor technology achievements. Different technologies such as wafer transfer, wafer bonding, selective epitaxial growth or re-crystallization are used to build 3-D FPGA chips [Silviu01a] [Ramm97] [Sailer97] [Pae99] [Fan01]. The cross section of a 3-D IC based on wafer bonding is shown in Figure 2.10, where the interconnections between layers are formed by vias etched through the thinned silicon layer [Ramm97] [Fan01] [Silviu01a]. With epitaxial growth or recrystallization technology, the back end of the line (BEOL) processing can be used to construct inter-layer interconnects.

Advantages of 3D-FPGAs over traditional 2-D FPGAs are as follows [Ramm97]:

(1) Reduced propagation delay. With the help of routing resources on the 3rd dimension, it is much easier to connect two points. It is estimated that routing track lengths can be reduced 45%~60%, compared with 2D FPGAs.

(2) Reduced Power Dissipation. This benefit comes along with the shortened routing track lengths. Estimation based on a chip with 20K cells indicates that power dissipation can be reduced 35%~55% over 2D chips.

(3) 3D routing reduced the need of routing channels on each 2D larger chip. Routing channels are expected to be narrower, and logic cell density is increased 25%~60% over 2D chips.

No commercial 3D chip has been reported. Silviu Chiricescu et al. fabricated a 3D chip, which we call NE3D FPGA [Silviu01b], as shown in Figure 2.11**.** NE3D FPGA is composed of 3 layers: the Routing Logic layer (RLB layer), Routing Layer and Memory Layer. The RLB layer is composed of routing resources and logic cells; the

Routing Layer is just composed of routing resources; and the Memory Layer is just composed of configuration memory.

NE3D takes a XC6200 like but a fully symmetric cell structure, as shown in Figure 2.12. Input signals to a cell not only come from neighboring cells but also from the lower routing layer. Similarly, its output signal goes both to neighboring cells and the routing layer. The routing layer of NE3D FPGA is composed of an array of switch boxes. Every (4*4) logic blocks on the RLB layer form a cluster. Under each cluster at the routing layer, there is a corresponding switch box. The switch box connects not only with the cluster but also with four nearby switch boxes. Each cell has a connection of width $Ro = 1$ to the switch box below it, while accepts $Rin = 2 \sim 3$ incoming wires from the switch box. On each side of the switch box, there are $W = 4 \sim 6$ outgoing wires plus W incoming wires, connecting with neighboring switch boxes.

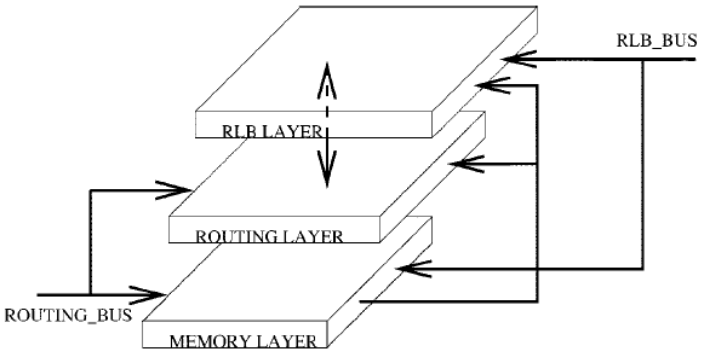


Figure 2.11 The Internal Structure of the Switch Box [Silviu01b]

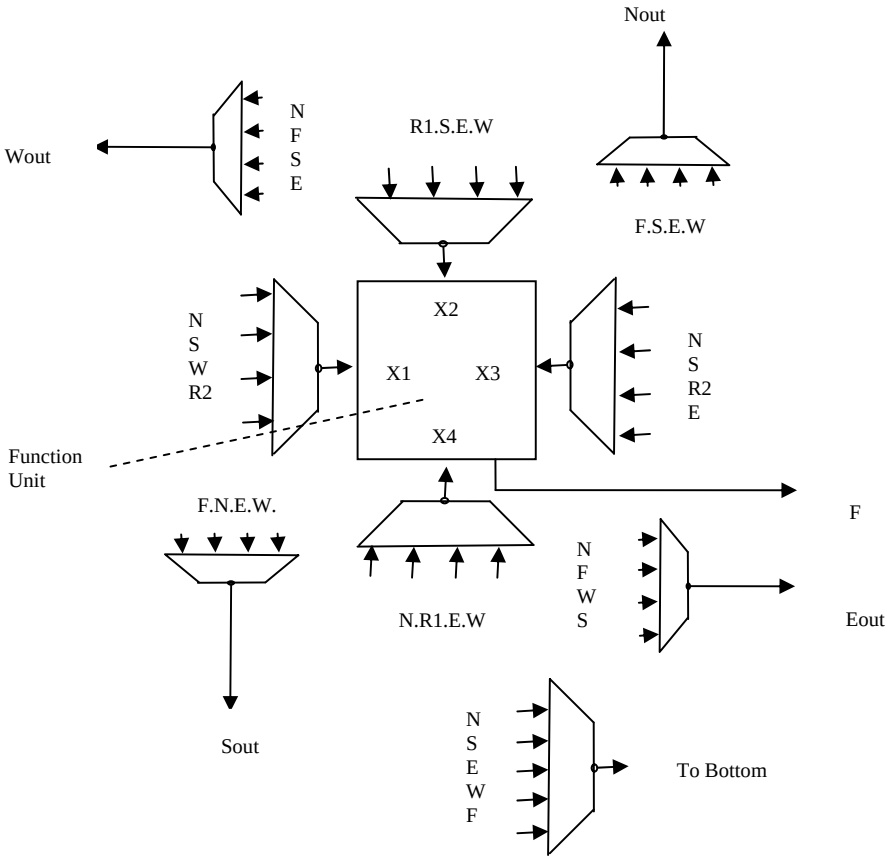


Figure 2.12 Cell Structure of the 3-D FPGA [Silviu01b]

2.2 Coupling between the Fixed and Variable Part

As discussed in Section 1.1, a reconfigurable machine is composed of a fixed part and a variable part. In terms of the connections between them, there are different methods:

(1) FPGAs are loosely coupled with a host machine. The fixed part is frequently a front-end host computer (a PC or workstation). The variable part is frequently an FPGA engine. The two parts exchange data via a general-purpose I/O interface bus. A typical example is the WildForce board by Annapolis [Annapolis98]. On the board more than one FPGA chip is connected by a crossbar and memories. The board can be inserted in a

PCI bus slot in a PC case. Data to be processed by FPGAs are downloaded to the on-board memory via the PCI interface, and the configuration bit stream can be downloaded to FPGAs via the same interface. Processed data from FPGAs are stored in on-board memories, and then read back by PC via PCI interface.

(2) FPGA works as a node on a network. This is the most loosely coupled method. This method is suitable for those situations where data exchanges between FPGAs and fixed machines are not frequent. S. Guccione et al. built such a prototype called Cam-E-leon [Guccione02]. In that system, users, web camera and IP servers are different nodes on a network. Users may operate a camera via the network, and IP cores associated the web camera can be updated (reconfigured) with new service uploaded by the IP server.

(3) On a traditional processor, its data path, control path and instruction set are totally fixed. But on a reconfigurable processor, these elements can be adjusted to satisfy specific applications' needs. Tensilica's Xtensa LX is such a reconfigurable processor. It has a basic IP core with an integer pipelined data path. Extra registers, execution units, and user-defined circuits can be added to the data path for specific applications. When a new execution unit is added to the data path, the original instruction set is changed, and the finite state machine corresponding to the control path is also re-configured. More about Xtensa LX's develop environment will be discussed later.

(4) FPGAs work as a co-processor of a CPU. The FPGA chip can exchange data with the CPU directly. An example of this type of coupling is the work by Ralph D. Wittig et al. [Wittig 96]. On the basic five-stage MIPS processor pipeline, reconfigurable logics were inserted on the EX (execution) stage, running in parallel with the basic functional unit (BFU) in the form of programmable functional units (PFU). BFU is

responsible for some elementary, arithmetic and logical operations required by many programs. The PFU can implement many application specific functions. Extra logics were required between the instruction fetch (IF) – execution (EX), execution (EX) – memory access (MEM) stages to orchestra data or control information.

2.3 Interconnection between On-chip Hardware Tasks

When partial reconfiguration is practiced, on-chip IP circuits should be connected to either other on-chip circuits or IO pads. Such connections are discussed in this section. Cases of 1D and 2D partial reconfiguration are discussed separately.

2.3.1 Connection Strategies for 1D Partial Reconfiguration

When partial reconfiguration is only allowed along one dimension, to solve the connection problem seems not difficult. On Xilinx Virtex FPGA chips, a method frequently adopted is to use tri-state buffers and long lines inside. With these resources, interface with IP circuits or “sockets” can be built. Different IP circuits may talk to each other via these interfaces.

N.P. Sedcole et al. used such a strategy and built a system for video image processing [Sedcole03]. As shown in Figure 2.13, processing elements (PEs) are attached to a global bus for communication. Every PE can convey data flow to the next nearby PE via unidirectional “chain bus”. Both “global bus” and “chain bus” are 32 bits wide. This architecture is suitable to video processing. Data flows go through cascaded processing modules along the pipeline. Due to the constraint of Xilinx Virtex FPGA, modules on the

system have to be column shaped, and this is likely to cause low CLB utilization. A similar method can also be found in the DHP project [Horta02].

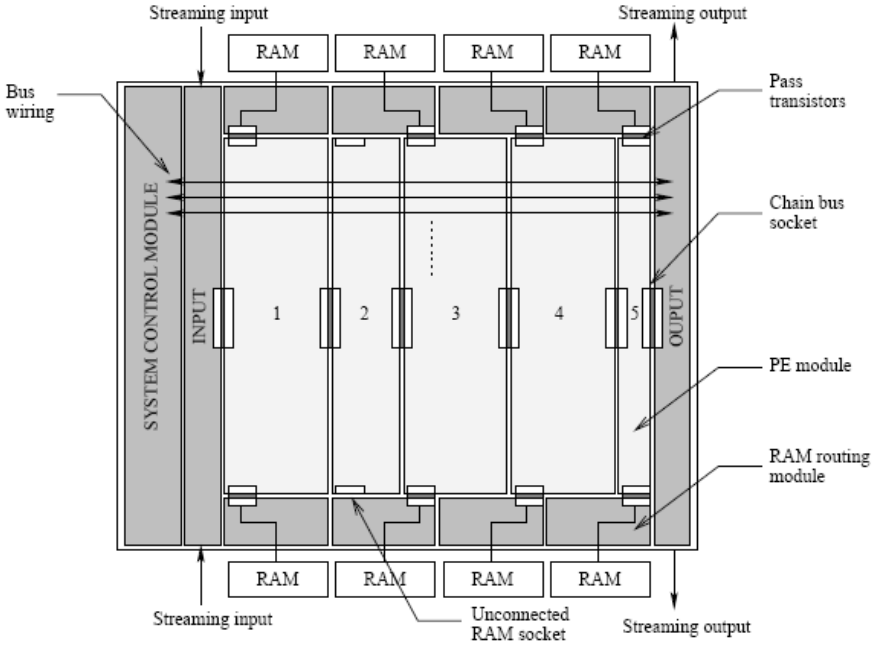


Figure 2.13 Partial Reconfiguration with Connections along One-dimension [Sedcole03]

Katherine Compton et al. suggested an architecture model to support 1D IP relocation [Compton00] [Li00]. To accommodate new IP circuits, IP circuits can be relocated vertically. Two long wires in each routing channel are assumed to connect circuits to IO pads at top/bottom side of the chip. After relocation, circuits can still be connected to original IO pads using the same two long wires. For this reason, IP circuits are not allowed to translate horizontally. No inter-IP connection is considered.

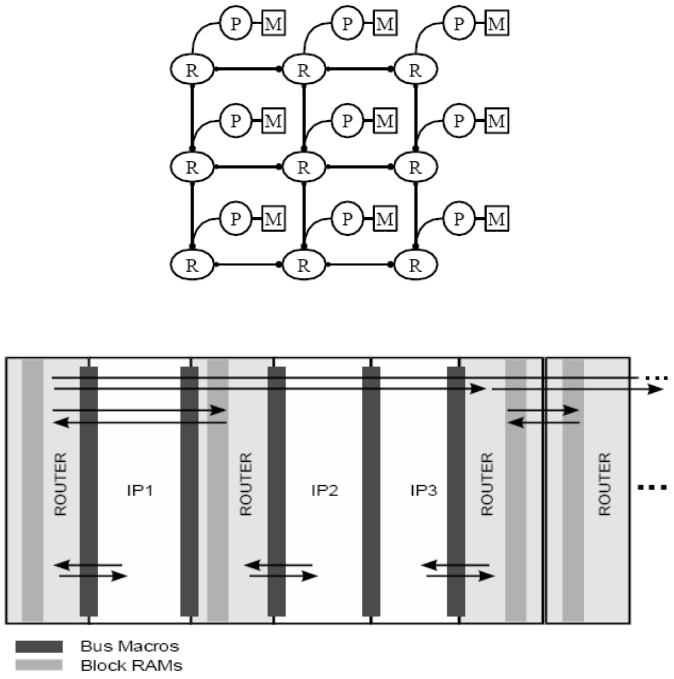


Figure 2.14 Folded Torus NoC [Marescaux02]

2.3.2 Connection Strategies for 2D Partial Reconfiguration

Compared with 1D partial reconfiguration, 2D partial reconfiguration can decrease fragmentation on the chip estate and therefore improve the chip utilization. However 2D partial reconfiguration makes the connection problem more difficult, as the circuit size and its location on the chip are quite flexible. How to support inter-IP connections on a 2D partially reconfigurable chip is still not resolved. There are two possible methods to build connections between IP circuits: packet switching and circuit switching.

Recently, NOC (network on chip) becomes a hot topic in the field of SOC (system on chip). On NOC, every module is hooked on the network via a standardized network interface, and follows the same network protocol [Sonics] [Dally01]. “Sonic” is a

centralized “Octagon” network, where eight processors in charge of packet forwarding and differentiation are hooked on [Sonics] [Karim01]. To address the problem of ad-hoc wiring strategy in chip design activities, William J. Dally and Brain Towels suggested a torus as the communication backbone network among 4*4 tile arrays. Each tile can exchange data with four nearby neighbors via input/output ports. Each port is composed of a 256-bit data field and a 38-bit control field [Dally01]. In that torus network virtual channels can be reserved for timing-critical communications. These two networks are based on ASICs, on which IP circuit relocations are not allowed.

Theodore Marescaux suggested a folded 2-D torus network, and prototyped it on Xilinx Virtex chips [Marescaux02], as shown in Figure 2.14. An operating system working along with such a network was also built [Marescaux04]. A hardware circuit has to be fit into a tile. All tiles have the same size. Physical connections between tiles are pre-defined at compile time rather than runtime, although packet routing paths can be changed. Christophe Bobda et al. suggested a dynamic NoC model [Bobda04] that assigns circuits (with varied size) at runtime with necessary connections. Routers on the net can be activated or disconnected to satisfy the communication needs. The area cost to build such a network was estimated on Xilinx Virtex chips, which allows only 1D partial reconfiguration.

A big advantage of packet switching is its scalability. Any two nodes on the net can communicate with each other. For the time being, only simple packet switching mechanisms such as wormhole routing have been built on FPGA chips. Packets (or flits) have to go through intermediate routers to reach a target node. Buffers of finite sizes at routers have impacts on the network performance. When a buffer capacity is reached, the

router stops forwarding packets. Hand-shaking on each link segment also reduces network throughputs.

Packet switching is not cheap in terms of the chip area. On a 3*3~4*4 mesh network, routers alone are reported to consume 6%~9% of total chip area. But for each circuit hooked on the net, it also needs a network interface circuit (NIC) that contains many components: packet-processing unit, synchronization unit, buffers for outgoing/incoming data, network busy/free detection unit and so on. For example, on a 3*3 mesh, routers consumed 8.3% of the chip area, and at each tile NIC consumed 1.8% of the chip area [Marescaux04]. Therefore totally 24.5% ($=8.3\%+9*1.8\%$) of the chip area was taken up by the routers and NICs. Other problems for packet switching includes: low bandwidth, extra packet processing time and complicated synchronization mechanisms between two communicating modules.

As a competing technology, on-line circuit switching can partially address those problems. It has higher bandwidth and less area cost. It is therefore more suitable for low granularity data exchange as is normally used in inter-IP communication. Some drawbacks of circuit switching are that its scalability may not be as good as that of packet switching, and extra on-line routing time is incurred.

On a reconfigurable machine, the routing overhead at runtime needs to be much lower than that in the traditional design flow. By separating the communication and computation part in the proposed architecture, this routing cost is greatly reduced. In later part, it can be seen that the routing cost is frequently around 10 ms and no more than 20 ms. For large Xilinx Virtex2 chips (with more than 8000 CLBs, comparable to our

assumed chip size used later), partially configuring only one unit costs more than 10 ms via JTAG [Xilinx04, page 309] as shown in Table 2.1.

Device	CLBs Row * Col	Partial Configuration Units	JTAG download Time (33MHz) ms	Download Time per Unit ms
XC2V6000	96*88	44	598.27	13.6
XC2V8000	112*104	52	793.17	15.25

Table 2.1 Configuration Cost of Two Xilinx FPGA Chips

Overlapping on-line placement and routing with circuit running or configuration can hide this cost. Routing time cost can be compensated by sophisticated algorithms, architectures and high-speed computing machines.

When a routing try is not successful, rip-up and rerouting are used which may interfere with existing running tasks. This is not the unique problem of circuit switching. For example, whether based on circuit switching or packet switching, tasks are interrupted by on-line compaction or de-fragmentation [Diessel98][Li00]. In the case of packet switching, even without on-line compaction, updating routing table on routers when the network topology is changed (dynamic NoC) may also interfere with running tasks.

2.4 Operating Systems for Reconfigurable Machines

In order to explore the capabilities of reconfigurable machines, runtime resource management may be handled by operating systems. Based on different models, researches around the word have spent quite some efforts in studying such kind of

operating systems. Because of the lack of chip architecture support, most such operating systems are not based on real chips. Performances of these systems are evaluated using theoretical calculation or numerical simulation rather than through on-chip testing.

In recent years, many operating systems targeting to 2-D partial reconfiguration were presented. Most of those works are concentrated on the runtime task placement and task scheduling. Connections between hardware tasks are frequently omitted. This omission partly comes from the lack of physical infrastructures to support necessary connections, and partly comes from the difficulty of the problem itself. Several such operating systems are summarized as follows.

2.4.1 ReconfigME

ReconfigME is an operating system developed by Grant Brian Wigley at University of South Australia [Wigley05].

When an application is submitted to ReconfigME, the application is broken into multiple small tasks (or processes). Partition is a very important component in this system. A big task may be broken into small subtasks to fit into available vacant slots available on the chip. The finer a task is broken into, the more time-consuming the partition is. Partition is more a compile-time work than an operating system work. This dissertation assumes that all tasks have been partitioned at compile time.

In ReconfigME, different tasks on chip are assumed to be hooked on a network controlled mainly by a memory controller. Communications between tasks are assumed by sharing data stored in the off-chip memories. The memory controller may become the bottleneck if many tasks compete for the memory controller. There was no discussion

about converting these infrastructures into real designs. The omission was probably due to the difficulty of runtime routing of such a communication network and the lack of practical toolkits [Wigley05, pp129].

As applications submitted to ReconfigME have to be characterized in a dataflow graph, dependencies between them should be known before runtime. This constraint limits its applicability.

In this work on-line routing between on-chip tasks was not discussed [Wigley05], but discussed in an earlier work, where pre-routed IPs were assumed [Wigley00]. Because intra-IP and inter-IP routing recourses were not separated, it is hard to reserve or differentiate those detail routing recourses in large quantity.

2.4.2 ETHOS

This operating system prototype was proposed at Swiss Federal Institute of Technology (ETH) [Steiger04]. Here it is named as ETHOS for short. ETHOS was targeted to a chip that supports 2-D partial reconfiguration. In ETHOS, a chip is assumed to be composed of many reconfigurable units (RCUs), each an atomic unit of partial reconfiguration. On-chip task preemption is not allowed. Each task/circuit is composed of many small RCUs enclosed in a rectangle.

The granularity and internal structure of RCU are not described [Steiger04]. But from the setup of simulation parameters, a RCU can be deduced as a CLB like structure. In this sense, ETHOS's targeting architecture is quite different from the cluster-based architecture proposed in this dissertation.

The beauty of ETHOS is its scheduling algorithm. If a new arrival task cannot be scheduled immediately, the operating system does not simply reject this task. Instead the operating system predicts the space available in the most recent future, and reserves this space for the task.

Even though the needs of inter-task or task-IO communication are noticed, it is still assumed that no such communication occurs in ETHOS. Based on this assumption, placement of tasks is bottom-left oriented. Vacant spatial slot searching is based on non-overlapped rectangles.

ETHOS assumed a one task–one circuit model, and there is no dependency among tasks. There was much discussion about the targeted chip. All those works reported were based on simulation.

2.4.3 RLCOS

It was proposed by Oliver Frank Diessel [Diessel98]. This operating system model is based on a 2-D partially reconfigurable chip.

In this work, an FPGA chip was modeled to be configured cell by cell. On-chip hardware tasks can be relocated or compacted to accommodate a waiting task. One interesting feature is that compaction is not based on re-loading of a configuration bit stream. Instead configuration bit information can be copied from cell to cell via dedicated channels. More than one of these dedicated channels can be run in parallel. Two different kinds of channels were suggested, i.e., mesh model and segmented bus. Here the mesh/segmented bus is different from that described in this dissertation. In Diessel's work, the segmented bus was used only for on-line task relocation rather than inter-IP

communications. Compared with the architecture of traditional FPGA chips, the time cost of task relocation can be reduced greatly. No commercial chip is reported to have such an infrastructure. The author proposed this conceptual infrastructure, without giving any circuit level information. Inter-IP or IP-IO connections are not considered in this work.

More than one on-chip hardware task may have to be relocated to reduce the fragmentation so that the waiting task can be swapped in. A promising position for the waiting task is such a position that the relocation cost is minimal. Two relocation strategies were suggested. i.e., local compaction and ordered compaction. The local compaction is based on two-dimensional bin packing, so the original relative positions of those hardware tasks are ignored. The ordered compaction can guarantee those relative positions based on shifting those relocated tasks along a 1-D direction to one end of the chip. If a hardware task was relocated, there was a time delay between it was removed and it was placed on the chip again. Among all those relocated tasks, the problem of minimizing the maximal delay was explored. This problem was defined as the scheduling problem of relocation. This problem was proved to be $NP-complete$, and a heuristic search algorithm was constructed.

2.4.4 IMEC OS4RC

Corresponding to the fixed torus NOC, an OS4RC was built with RTAI [Marescaux04] [Nollet03]. This operating system is targeted to a heterogeneous system, on which, traditional instruction set processors and reconfigurable processors (FPGA) co-exist. Communications among different types of processors are allowed. A two-level task scheduling strategy was proposed. The top-level scheduler resides on a central instruction

set processor. It performs the task of processor assignment. The lower-level schedulers reside on their local processors, performing temporal ordering of the tasks that have been assigned to them. But for the lower-level schedulers on the FPGA part, no-detail architecture was reported. As tasks have to fit into tiles of the same size, the OS4RC can do placement with minimal overhead at the cost of higher fragmentation. Task relocation was supported subject to the bottleneck of downloading streams into FPGAs. Some simple tasks were tested with this operating system.

2.5 Reconfigurable Machine Programmability

This section describes the programmability problem of reconfigurable machines and summarizes some solutions proposed by the community. The following comments by Wim Roelandts show that industries are striving to improve the programmability of reconfigurable machines.

> ‘The next step is really to make FPGAs disappear. Today our customers are hardware engineers. But FPGAs are programmable devices. If we can create a level of abstraction that appeals to software engineers, we can increase our customer base by at least 10x. That's really where our future is. As long as you have a set of interfaces that you can program to, you don't have to know what the hardware looks like.’
>
> Wim Roelandts, Xilinx CEO, June 2005

2.5.1 Why Reconfigurable Machines Are Not Popular

For a long time, the poor programmability of a reconfigurable machine made it hard to be widely accepted. For an application that is partitioned into a software part and a hardware part, two different kinds of languages are frequently used to program on reconfigurable machines. Codes for the software part are frequently written in high-level languages (e.g. C/C++), while codes for the hardware part are written in hardware description languages (e.g., Verilog / VHDL). The two parts are compiled separately.

To most software programmers, circuit design is beyond their interest or knowledge. Even when they design circuits in HDLs, they prefer behavior descriptions (C/C++ style), rather than structural descriptions. Compared with programmers on traditional machines, reconfigurable machine programmers are overburdened. On the other hand, if the HDL compiler is not smart enough, circuits synthesized from behavior descriptions frequently are clumsy both in terms of area and speed, compared with circuits from structural descriptions.

Recently, software begins to run on CPU IP cores in an FPGA chip. For an application, there is a need to decide which part runs on software, and which part runs in hardware. There are many different possible options, depending on available resources and performance requirements. If both the hardware and software parts can be written in a uniform language, debugged and simulated within a uniform environment, the system design process will be much more efficient.

2.5.2 Reconfigurable Machine Programming: C/C++ vs. HDL

HDLs are designed for hardware engineers who want to create sophisticated circuits whereas high level languages (HLLs) are developed for programmers who do not need to have any hardware knowledge at all. Many algorithms have been implemented in C/C++ languages. If these algorithms can be transplanted to reconfigurable machines without translating into HDLs, the code development period can be largely shortened.

HDLs have been proven successful for RTL (register-transistor-level) abstraction. During an RTL simulation, state changes of each gate/register are tracked for many clock cycles. At this time of deep sub-microns, multi-million gates are integrated on one chip and it is hard for simulators to track every states. System simulation and verification become very slow. New methods which can characterize systems in higher level abstractions are needed to shorten the simulation time.

Furthermore, at this time of system on chip (SOC), IPs from different sources are integrated together to shorten the design cycle. System design focuses more on inter-IP connections rather than IP internal structures. It is not necessary (or sometimes impossible) to access the internal information of IPs, especially when IPs come from a third party. High level abstraction is again required.

The emerging methodology is called ESL (Electronic System Level) design methodology. Such a method focuses on behavior descriptions rather than structural descriptions of IPs. Most importantly, C/C++ are languages with higher level of

abstractions over HDLs. Coding in C/C++ can enhance design productivity and speed up system verification or simulation for very complex systems.

2.5.3 C-like Hardware Synthesis Languages

Standard C/C++ cannot be used to implement an ESL design directly. C-like hardware synthesis languages are frequently subsets or supersets of standard C/C++ with extended libraries. The reasons are as follows.

(1) Concurrency is a fundamental characteristic of hardware. Traditional C/C++ code for sequential algorithms does not support this concurrency. A good hardware compiler can extract this concurrency automatically.

(2) As semantics of C/C++ have intrinsic sequential characteristic, synchronization between different concurrent hardware processes is another problem. Because standard C/C++ does not have a time model to characterize causality, this makes achieving timing constraints difficult.

(3) C/C++ does not support variable bit resolution.

In the past ten years, many C-like hardware synthesis languages have been proposed by industries and academic institutes [Edwards 05], such as Cones [Stroud88], HardwareC [Ku90], Transmogrifier C [Galloway95], SystemC [Gr¨otker02], Ocapi [Schaumont98], C2Verilog [Soderman98], Cyber [Wakabayashi99], Handel-C [Celoxica03], SpecC [Gajski00], Bach C [Kambe01] and CASH [Budiu02].

Out of so many C-like languages, System C and Handel C are distinguished over others. Both of them support hardware/software co-design, and they are two leading candidates to be accepted by the EDA industry.

2.5.3.1 Handel C

Handel C was initially created by the Hardware Compilation Group at Oxford University [Oxford 97]. It is a subset of ANSI C. Only limited data types of ANSI C are inherited: integer, unsigned, char, long and short, while other data types, such as float, double, enum and so on, are removed. Extra data types, such as channel, RAM and interface are added. Other additional features are parallel expression of communications between parallel processes and operators for detail control of hardware. As Handel C supports a large set of ANSI C constructs, porting between the two languages is possible. This is the basis for Hardware/Software Co-design.

2.5.3.2 System C

System C is based on standard C++, but extended with new class libraries. Many software features, such as concurrency, events and data types, are included. In System C, interfaces between circuit blocks are defined as communication methods and protocols rather than signals. This is a big difference from common HDL descriptions. Design abstractions and therefore design efficiency are drastically increased. This increase comes from the extended class libraries, which provide new mechanisms to model system architectures with hardware elements, concurrency and reactive behavior. System C 2.0 introduced a feature called TLM (transaction level modeling). Communications are

modeled as channels and transaction requests use interface method calls of these channel models. Unnecessary details of communications are hidden (but can be worked out later on). System C supports both ESL and RTL simulations.

2.5.3.4 Emerging Reconfigurable Machine Environment

For an application written in a C/C++ derived language, some but not necessarily all for-loops are likely to be accelerated on FPGAs. Applications have to be partitioned to decide which part is mapped to general purpose CPUs (or host computers) and which part is mapped onto FPGAs. This problem is generally *N-P hard*, and subject to different constraints, such as FPGA resources, real-time deadlines, and even energy consumptions. Compilers for these C/C++ derived languages are expected to partition an application automatically using some optimization criteria. These works are called Hardware/Software Co-design. Although manual adjustments and human directions are often demanded, easy-to-use reconfigurable machine environments are emerging. They are summarized as follows.

(1). Tensilica' Xppres is a compiler that works with its reconfigurable processor Xtensa LX. When an application written in C/C++ is submitted to Xppres, the complier evaluates it based on a basic processor configuration as the starting point of exploring different possible configurations. Then the speed gains and area costs for different configurations are presented to the user so that the person can select one. Xpress then automatically produces all solutions to data path and control path. The instruction set used reflects the new processor architecture, and execution codes of the application are

optimized based on the new instruction set. It is claimed that the speedup of the re-configured processor is comparable to that of RTL circuits

Unlike most other HW/SW co-design situations, the processor is synthesized from an available working processor rather than from scratch. Another feature of this system is that the application is submitted in standard C/C++, and no modification is needed.

(2) During the ERSA 2005 conference, Stretch Inc. demonstrated an integrated development environment called Stretch IDE. On that platform, application codes are written in C/C++ (with Stretch C extension). It is claimed that circuits synthesized by Stretch C Complier are comparable to those from HDL structural descriptions.

(3) Celoxia's DK is a hardware design and synthesis environment that supports Handel-C. Cray Inc. is collaborating with Celoxica to make DK Design Suite available to customers of its reconfigurable machine, the Cray XD1™ supercomputer [Celoxica05]. Using a high level language software design flow, programmers can accelerate their applications with FPGAs on Cray XD1.

(4) Commercial tools such as Cadence's NC-System C/Incisive and CoWares' Convergence SC System Design support mixed-language integrations. ESL design can be started in System C and end up with gate-level representations for fabrication.

In summary, the semiconductor industry has produced FPGA chips with increasingly higher capacity and faster speed, and this trend will continue. Compiling/synthesis technology is becoming more and more sophisticated, and circuits based on behavior descriptions are getting better and better. The industry has demonstrated some successful products to overcome the programming bottleneck, and

more effort is still undergoing. More powerful and easy-to-use reconfigurable machines are emerging and wining over more audience.

3 Aspect Ratio Effects

This chapter presents the effects of different aspect ratios on chip area, performance and CLB utilization. The proposed architecture in this dissertation uses square clusters because of these results. In Section 3.1, the definition of aspect ratio is given, and some background information is supplied. In Section 3.2, effects of reshaping a circuit by changing its aspect ratio are shown. Results indicate that reshaping might cause chip area inflation as well as performance degradation, which can not be ignored in practice. In Section 3.3, the square cluster structure suggested is shown to be close to optimal in terms of chip utilization. Section 3.4 concludes this chapter.

3.1 Introduction

The adoption of IP cores can speed up FPGA design and testing, and therefore shorten the time-to-market. Partial reconfiguration allows the loading of a new IP core without interfering with the working of other circuits on the FPGA chip.

Generally, there are two different kinds of IP cores, i.e., hard IP cores and soft ones. Hard IP cores have already been placed and routed in advance. Even though a hard IP core may be put at different locations inside the chip, relative positions of all its building logic blocks and connections between those blocks are fixed. Soft IP cores have not been placed and routed. To simplify discussion, the shapes of both hard and soft IP cores are approximated as rectangles.

Definition: Aspect ratio, r **,** is defined as the ratio of a circuit's height to width.

A hard IP core has a fixed aspect ratio (shape) once it is created, even though at design time its aspect ratio may be adjusted to achieve some trade-offs. While the aspect ratio of a soft IP core can be adjusted at placement or compile time. A soft IP core can be reshaped before being loaded to a partially reconfigurable FPGA chip. That is, its placement and routing can be adaptive to current space or routing constraints. Consider a runtime reconfigurable machine that allows the dynamic swap in and out of hardware tasks (or IP circuits). The shape and size of available space may be such that a hard IP cannot fit while a soft IP of the same functionality can. But in both cases, reshaping is not free, and consequential impacts have to be considered.

3.2 Impacts of Reshaping an IP Core

Previously the reshaping of a soft IP is based on the assumption that the logic area of a soft IP core remains the same [Tessier02] [Kalte04]. A similar assumption was also previously used in ASIC floor planning [Sait95]. Because of the large area used for routing, adopting such an assumption without accounting for the routing area may lead to overly optimistic expectation. In this section the effects of soft IP reshaping are studied. It will show that inappropriate reshaping leads to area inflation as well as performance degradation. Before reshaping is implemented, proper evaluation of these negative effects is necessary.

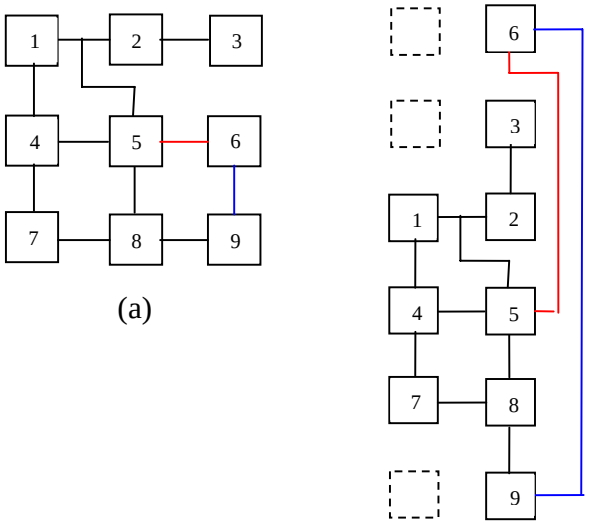


Figure 3.1 An Example of Reshaping

3.2.1 Example

Here an example is used to illustrate the negative effects of inappropriate reshaping. Suppose an IP circuit is composed of 3×3 CLBs (Configurable Logic Blocks), as shown in Figure 3.1 (a). When it is reshaped into a 6×2 rectangle as in Figure 3.1 (b), CLBs 3, 6 and 9 have to be relocated to new positions. Connections associated with them become longer than before, and the time delays along these stretched connections also increase. Chip utilization may also deteriorate, as those CLBs in dashed lines are wasted because of fragmentation.

3.2.2 Simulation Results

Simulations were performed to determine the effects of reshaping on area cost and critical path delay. Our experiment is based on running of VPR on 20 MCNC benchmark

circuits with various aspect ratios [Betz99] [VPR00]. After those benchmark circuits are completely placed and routed, the area cost (in a unit of 10^6 minimum transistors) and critical path delays (in a unit of 10 nanoseconds) are measured. Their averages are summarized in Figure 3.2. The following virtual chip architecture is assumed: every CLB is composed of four look-up tables and flip-flops, and all routing tracks inside channels have unit length.

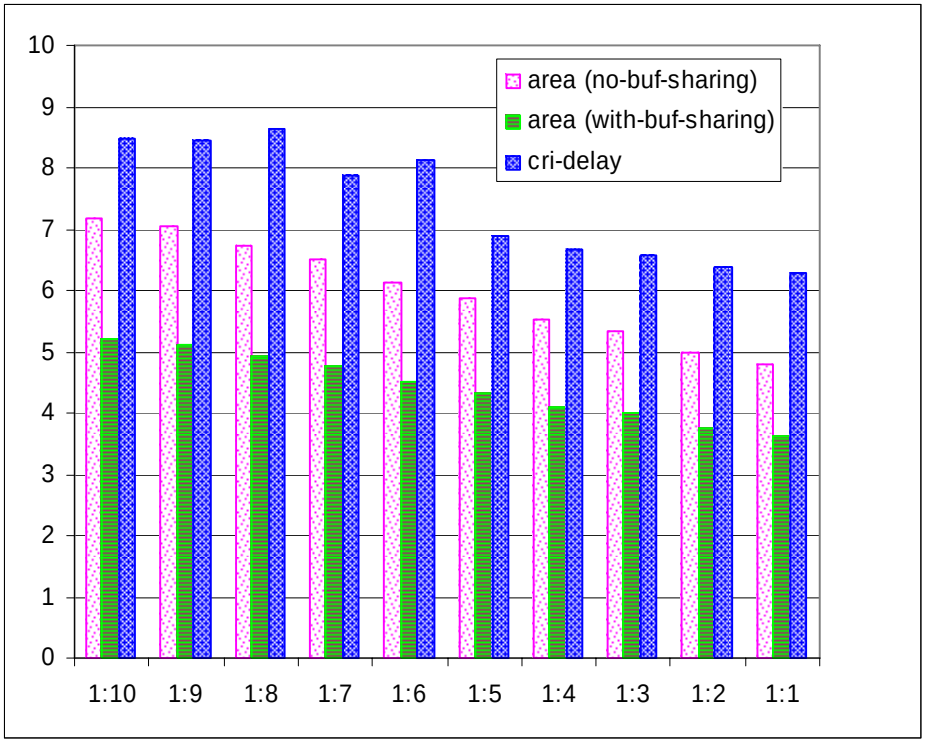


Figure 3.2 Area Cost and Critical Path Delay

From Figure 3.2, it can be observed that the case of $r = 1:1$ has the smallest area and the lowest delay. The average critical path delay at $r = 1:1$ is only 73% of the maximum value. Two different kinds of routing segment structure are considered, i.e.,

routing segments share buffers or not [Betz99]. For the cases of with (without) buffer sharing, the area costs at $r = 1:1$ is just 69% (67%) of the maximum value.

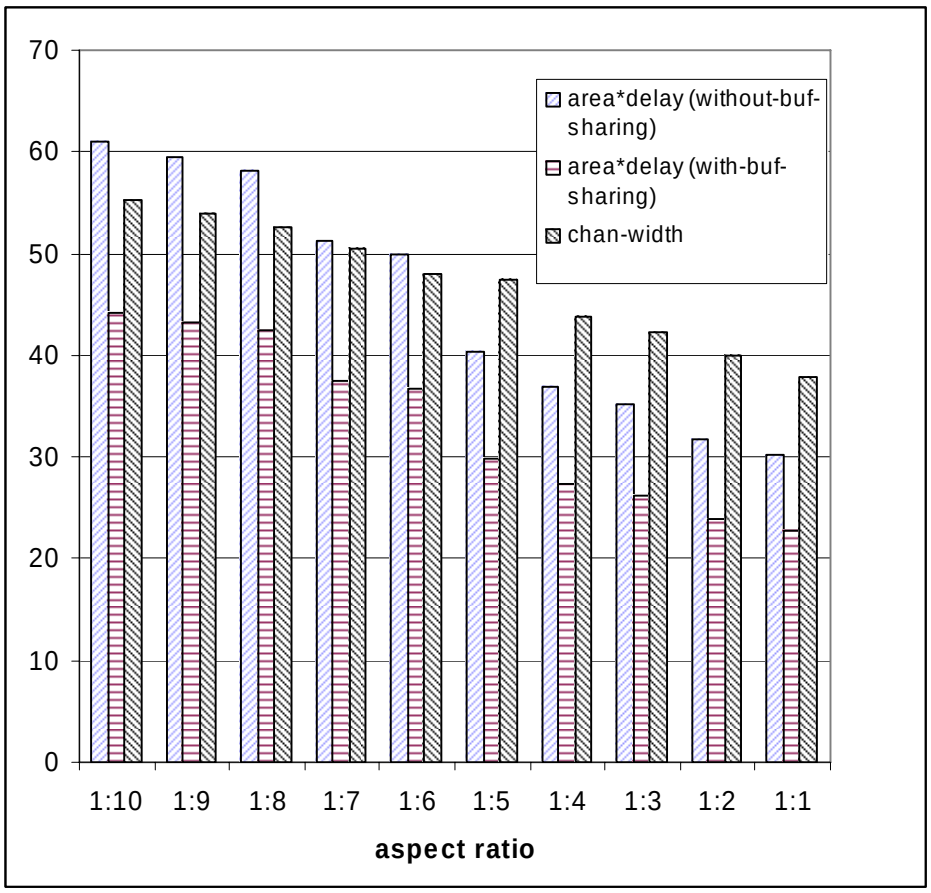


Figure 3.3 Area-delay Product and Channel Width

Area and delay are trade-offs in VLSI design; so area-critical path delay product is frequently used to evaluate this trade-off. Figure 3.3 summarizes the simulation results in terms of area-delay product (measured in 10^7 minimum transistors * nanoseconds) and channel width. Channel width is the minimum number of tracks needed per channel to route those benchmark circuits successfully.

In Figure 3.3, for the case with (without) buffer sharing on routing segments, the area- delay product at $r = 1:1$ is 23 (30), while at $r = 1:10$, the corresponding value is 44 (62). Area-delay products at $r = 1:1$ are almost 50% of corresponding values at $r = 1:10$. As the aspect ratio gradually decreases, the channel width monotonically increases. At $r = 1:1$, an average channel width of 38 is enough, while at $r = 1:10$, an average channel width of 56 is required. Since the channel width of a commercial FPGA chip is fixed, the case that requires the least number of channel widths has the best chance of being routable. Note that simulation results are based on the averages on 20 circuits. There is no reason to require every IP be of a square shape. In fact, using a column of CLBs may be perfect for some circuits [Kalte04].

Curves in Figure 3.2 and 3.3 can be similarly obtained for individual IP cores. They can be used as follows. Before a possible reshaping is really taken, its effects can be estimated by looking up corresponding curves and applying interpolation or extrapolation. With those estimates, the on-line compile time for the soft IP circuit can be shortened by pruning off those infeasible shapes, and avoiding unnecessary placement and routing time. Note that it is impractical to try to pre-place-and-route an IP core on all possible shapes.

3.3 Aspect Ratio Consideration on the Design of Partial Reconfiguration Structure

3.3.1 A Cluster Based Partial Reconfiguration Unit

Partial reconfiguration capability is supported by a chip's physical infrastructure, at two different levels: CLB level and Cluster level.

On Xilinx's XC6200 FPGA chip, every cell of an array can be individually reconfigured [Xililinx96]. Partial reconfiguration targeted to this kind of chip infrastructure is called CLB level partial reconfiguration. This level of partial reconfiguration may be too fine grained than necessary. At runtime, users concern more about whether a hardware task with certain functions has been swapped into an FPGA chip. Those functions normally require many CLBs. Users seldom worry about changing the internal structure of a hardware task. To users, the CLB level manipulation is complicated and time consuming. When using IP circuits from 3rd party vendors, users may even have no chance to manipulate those single CLBs or interconnections between them.

Partially reconfiguring a batch of CLBs at a time is called cluster level partial reconfiguration, which seems more practical. Xilinx Virtex chips support cluster level partial reconfiguration where the minimum partial reconfiguration unit is a frame, containing four columns of CLBs. A chip is partitioned into a number of columns, each of them spanning from chip top to bottom. In this case the height of every hardware task is fixed, while the width is variable. Partial reconfiguration based on this architecture is in fact one dimensional. Compared with a CLB level partial reconfiguration structure, Virtex is an improvement. However, there is no reason to require hardware tasks to be shaped into columns. For example, a carry-ripple adder can fit into a column easily, but it is hard to fit a carry-prediction adder. Squeezing a hardware task into columns by brute force may incur not only low utilization but also performance degradation as indicated in the previous section.

An ideal structure will support 2-D partial reconfiguration at cluster level. On such a 2-D architecture, the fixed height constraint for hardware tasks is removed. Theoretical works have shown 2-D partially reconfigurable architectures have advantages over 1-D architectures. On a 2-D architecture, the percentage of rejected tasks by operating system is far lower than that based on a 1-D architecture [Steiger04], although no 2-D cluster level partial reconfiguration structure has been adopted in commercial FPGAs.

3.3.2 Utilization and Cluster Aspect Ratio

This section examines the relationship between utilization and cluster aspect ratio based on a statistical approach.

Definition: CLB **utilization** is the ratio of the number of CLBs actually used for a circuit to the total number of CLBs in those clusters allocated for the circuit. CLB utilization is less than or equal to one.

Let w and h denote the width and the height of a cluster respectively. Suppose $w \cdot h = s$ is a constant. Let X and Y be respectively the width and the height of a circuit to be loaded into the FPGA chip. X and Y are random variables of positive integer values. For a specific circuit CKT_i, $X = x_i$ and $Y = y_i$. The area of a circuit is generally bigger than that of a cluster, i.e., $X \cdot Y \succ\succ w \cdot h$. If $E\{\cdot\}$ represents expectation, and $\lceil \cdot \rceil$ represents the ceiling operation, then the utilization, U, can be expressed as:

$$U = E\left\{\frac{X \cdot Y}{\left\lceil \frac{X}{w} \right\rceil \cdot \left\lceil \frac{Y}{h} \right\rceil \cdot w \cdot h}\right\} = E\left\{\frac{X \cdot Y}{\left\lceil \frac{X}{w} \right\rceil \cdot \left\lceil \frac{Y}{h} \right\rceil \cdot s}\right\} \quad \text{(Eq. 3.1)}$$

Here $\left\lceil \frac{X}{w} \right\rceil \cdot \left\lceil \frac{Y}{h} \right\rceil$ is the number of clusters needed for the circuit and therefore the denominator is the cluster area for the circuit. Our interest is in finding a good cluster size that leads to high utilization for various circuit sizes. Depending on the distribution of circuit sizes, the optimal cluster size may be different. That is why there is an expectation term in Eq. 3.1.

If the cluster aspect ratio is defined as $h/w = r$, then an interesting problem is to find an r such that:

$$U(r) = \max E\left\{\frac{X \cdot Y}{\left\lceil \frac{X}{w} \right\rceil \cdot \left\lceil \frac{Y}{h} \right\rceil \cdot w \cdot h}\right\}$$

In order to identify the impact of the cluster aspect ratio on the CLB utilization, Eq. 3.1 was computed based on three different distributions for the widths and heights. They are uniform, Poisson, and normal distributions.

3.3.3 Simulation Results

Some Monte-Carlo simulations were performed to provide a different perspective on the relationship between CLB utilization and cluster ratio. The idea is to compute Eq. 3.1 based on various (rectangular) circuit sizes and cluster sizes. For each cluster size,

10,000 circuits were randomly generated. The widths and heights follow three different distributions, i.e., normal, Poisson and uniform. Results for five different "approximated" cluster sizes are summarized in Figure 3.4 and 3.5. In each diagram, there are five curves, one for each approximated cluster size. They are arranged from top to bottom, corresponding to approximated cluster sizes of 1, 4, 12, 16, and 25, respectively. Here a cluster size is approximated because points on the same curve do not have exactly the same cluster size. For example, the curve whose approximated cluster size is 12, corresponding to clusters of dimension (h x w) 1x12, 2x6, 3x4, 4x3, 5x2, 6x2, 7x2, 8x1, 9x1, 10x1, 11x1 and 12x1. Six out of them have an exact area of 12 as marked on the curve. To show those results clearly, horizontal axes (i.e., aspect ratio, $h/w = r$) in Figures 3.5 and 3.6 are given in logarithm scale.

Observations from Figure 3.4 and 3.5 are as follows:

(1) The smaller the cluster is, the higher the utilization is. This is intuitive because, as the cluster size decreases, the fragmentation effect is alleviated. When the cluster size is equal to one, curves regress into one point at 100% utilization.

(2) For every curve, the maximum point occurs on/near the point where the cluster height is equal to its width (i.e., r = 1).

(3) For Poisson and normal distributions, their results are quite close to each other. This is because the distributions are very similar to each other when their average values are bigger than 10.

(4) Even though the results from uniform distribution are not close to results from the other distributions, the trend of changes are the same irrespective of distributions. In

other words, the utilization depends more on cluster shape ($h/w = r$) than on the exact statistical distribution.

(5) In Figure 3.4, those curves are almost symmetric. For example, for $w \cdot h = s$ = 16**,** the utilizations for 2x8 and 8x2 are very close. In Figure 3.5, there is no such symmetry, because $E\{X\}$ is not equal to $E\{Y\}$.

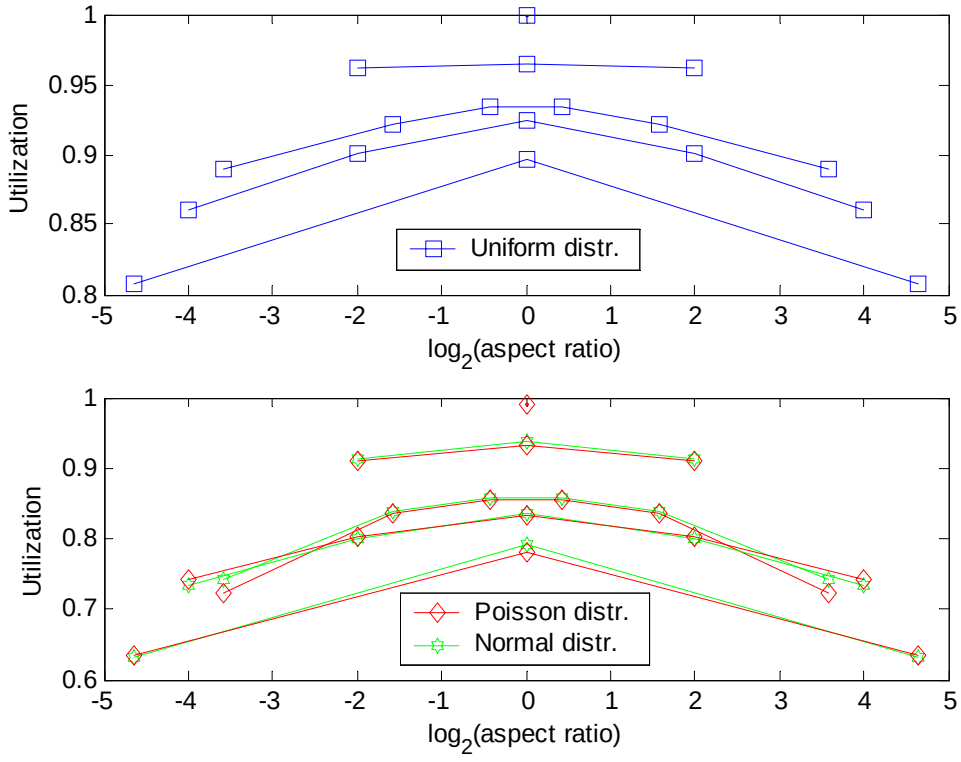


Figure 3.4 Utilization When E{x}=16 and E{y}=16

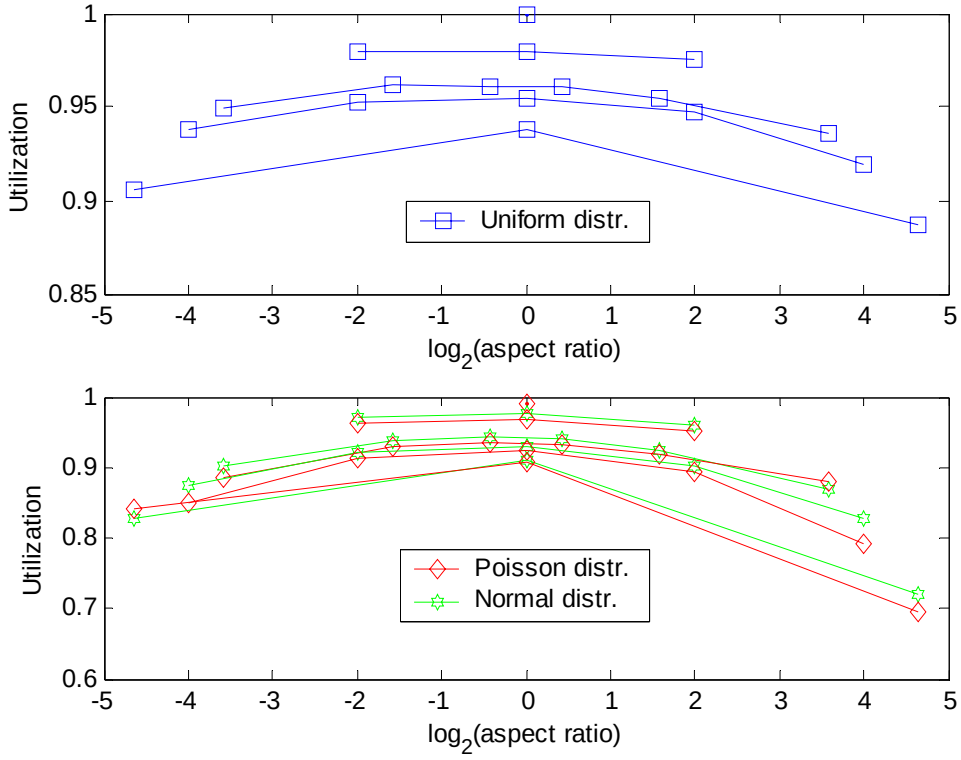


Figure 3.5 Utilization When E{x}=50 and E{y}=35

The analysis and simulation in this section are based on a statistical method to determine the aspect ratio of a partial reconfiguration unit. Even though the effects of a circuit aspect ratio on area and path delay are not included here, Section 3.2 can be used as a guide to shape the partial reconfiguration unit when hardware tasks need to be accommodated into partial reconfiguration units.

3.3.4 Verification with Benchmark Circuits

In another experiment, the 20 MCNC benchmark circuits mentioned in Section 3.2 were used. They were first placed and routed using VPR under various shape

constraints (with aspect ratios ranging from 1:1 to 10:1). For each circuit, the one shape with the lowest area-delay product was included in a set of (X, Y) pairs. The result was a set of 20 pairs. Figure 3.6 shows the CLB utilizations computed using this data set for various aspect ratios. It shows that for this set of circuits, a square cluster leads to the highest CLB utilization.

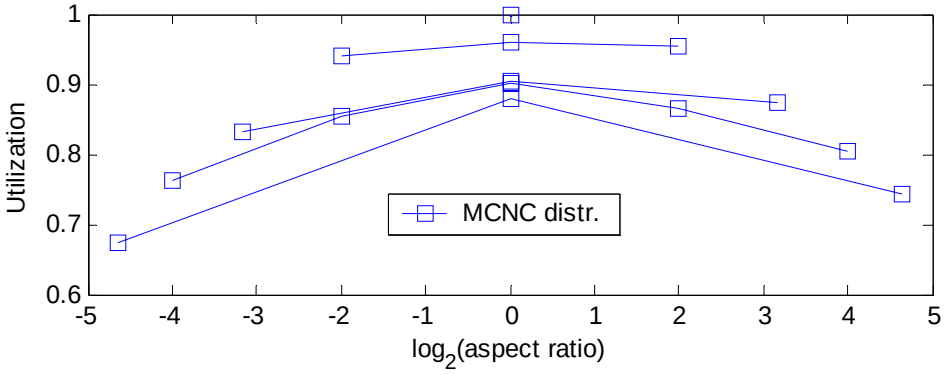


Figure 3.6 Utilization for MCNC Circuits (20-pair data set)

For each MCNC circuit, if two shapes corresponding to the two lowest area-delay products are considered, then the data set contains 40 pairs. In this case the utilization is as shown in Figure 3.7. Because aspect ratios in the VPR simulations range from 1:1 to 10:1, this case artificially adopts more (X, Y) pairs with greater than one aspect ratio. Therefore it is biased toward non-square shapes.

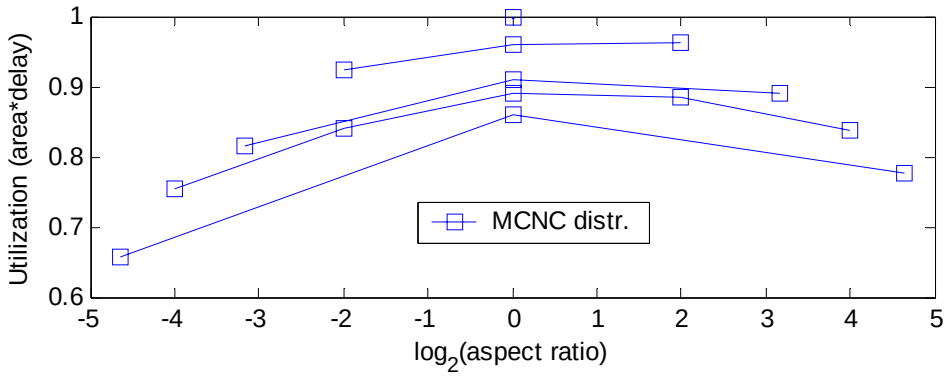


Figure 3.7 Utilization for MSNC Circuits (40-pair data set)

3.4 Conclusions

Effects of aspect ratio on soft IP reshaping as well as partial reconfiguration structure design have been discussed. Results indicate that the previous assumption of having a fixed area of soft IP should be removed, and a square-cluster based partial reconfiguration seems to be a good choice over the column based partial reconfiguration.

4 A New FPGA Architecture

In this chapter, a new FPGA chip architecture is proposed. The architecture supports cluster-based partial reconfiguration and uses a segmented bus structure to provide interconnections between on-chip hardware tasks or between hardware tasks and chip IOs during partial reconfiguration at runtime. Section 1 provides an overview of the architecture. The chip area consumed by such a segmented bus structure is evaluated in Section 2. The number of configuration bits needed to configure the segmented bus is also calculated. The area cost and the configuration cost are illustrated in Section 3 using some chip examples.

4.1 Overview of Chip Architecture

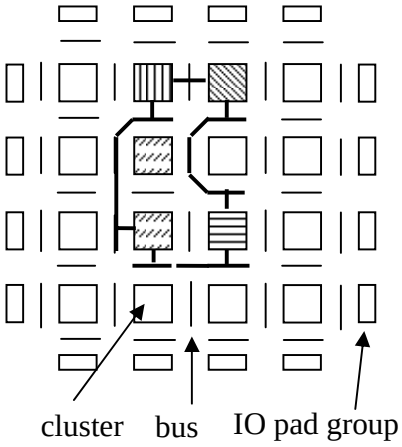


Figure 4.1 Cluster-Segmented Bus Structure of the New Chip

The proposed chip contains many square clusters. Between two nearby clusters, there are segmented buses. At each intersection of bus segments, there is a bus switch

block to relay bus connections. Different hardware tasks can be connected together via bus segments and bus switch blocks.

The chip has two distinct features:

(1) Partial reconfiguration is supported. The minimum partial reconfiguration unit is a square cluster. As described in Chapter 3, such an arrangement improves chip utilization [Wang05].

(2) Segmented buses are used solely for runtime support. They are not used for intra-IP routing, which is still supported by traditional FPGA routing resources. Bus segments and clusters can be configured separately. When bus segments are reconfigured, clusters and their internal connections are not perturbed.

Figure 4.1 shows an example with four hardware tasks mapped onto clusters. In that figure, each small square represents a cluster of NxN CLBs. A big hardware task or IP circuit can be mapped on more than one cluster. Clusters used for the same hardware task are identified with the same patterns. Just like CLBs are packed into clusters, nearby IO pads on the chip boundary are packed into groups. Each cluster or IO pad group is a network node when runtime routing is applied. More details are in Chapters 5 and 6.

The proposed architecture uses separate routing resources for inter-IP and intra-IP communications. At runtime only the segmented buses are to be routed for inter-IP communications while the intra-IP communications using the traditional FPGA routing resources are not disturbed. This arrangement is to shorten the runtime routing time. In [Marescaux02], packet switching based on a static torus network was proposed for dynamic multi-tasking. The segmented bus is a dynamic network that uses circuit switching.

Details of the architecture in terms of the CLB model and cluster blocks are described as follows.

4.1.1 CLB Model

The CLB model as suggested by Betz is used in this architecture [Betz99]. Each CLB is composed of four BLEs. Each BLE is composed of one flip-flop and a four-input look-up table. Each CLB has four output signals, 10 external input signals, plus clock, set/reset signals. The four output signals are fed back into the CLB itself. Each BLE is associated with four 14:1 full mutiplexers, which convey the 10 external input signals plus the four output signals. Output signals from multiplexers are used as input signals of each look-up table. Totally 16 such multiplexers are included. In terms of area, such a CLB is equivalent to 1678 minimum width transistors. Such a CLB model is close to that of Xilinx Virtex chips, where each CLB is composed of two slices, and each slice has two BLEs.

A CLB block is defined as a CLB plus routing resources associated with it [Betz99]. Generally these routing resources include that CLB's input connection block, output connection block and a switch block. The area of a CLB block depends on the CLB itself and routing architectures of the chip.

Betz did not suggest any determined routing architecture, but gave out some optimum parameters, such as lengths of routing segments. So the area of a CLB block was not defined. But according to the simulation results, it is reasonable to assume that the area of a CLB block is at least five times as big as that of a CLB itself.

For traditional FPGA chips, structures of different components, such as CLBs, routing segments and so on, have been intensively investigated previously. In the

following analysis, those traditional architectures are treated as an abstract models. The proposed segmented bus and its accessories are extra sources superimposed on traditional FPGA chips. Routing resources may compose 90% of chip area [Dehon96].

4.1.2 Cluster Block

As an extension to the concept of CLB block, the concept of cluster block is suggested. As shown in Figure 4.2, a cluster block is composed of the cluster itself, the two crossbars on its bottom and right side, and the bus switch block at its lower and right corner. Crossbars at its top and left, and the bus switch block at other three corners are considered to be associated with other cluster blocks.

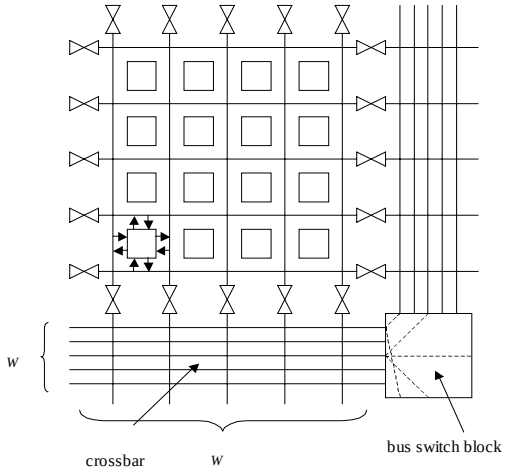


Figure 4.2 A Cluster Block

On boundaries of each cluster, there are tri-state buffers. These tri-state buffers can be used to insulate the cluster from the bus during the time of partial reconfiguration or when circuits at other clusters get the control privilege of the buses. The bus switch block is dedicated to relaying bus segments and changing bus routing directions. Crossbars and bus switch blocks are called the backbone network.

4.2 Segmented Bus Area Cost and Configuration Bits Formulation

This section presents the calculation of various components of the segmented bus structure in terms of area cost and configuration cost. The area model used is introduced first.

4.2.1 Area Model

Area estimation is more accurate after physical layout is completed, which depends on the specific semiconductor process. Before physical layout, an FPGA area can be estimated via counting the number of transistors. This method is independent of a specific semiconductor process, and is widely used [Betz99] [Silviu01b]. According to Betz et al., the area of an FPGA chip is dominated by the number of transistors rather than interconnections. This point was claimed to be supported by Xilinx and Altera architects [Betz99, page 132]. Transistors on the chip may have different sizes. This size variance generally comes from driving capability considerations, i.e., big transistors can drive heavy load with higher speed. Areas of big transistors can be scaled with transistors with the minimum channel width. Let the NMOS transistor with the minimum channel width have a unit driving capability. A transistor with twice as much driving capacity does not have to have twice as much area, (e.g., when parallel diffusion technique is used). The space distance between different transistors also does not increase with the transistor size. Betz gave out a formulation to calculate the area of a transistor after sizing, and used the summation of sized transistors to evaluate the area of an FPGA chip. The area of a transistor after sizing can be calculated as [Betz99, page 133]:

$$0.5 + \frac{0.5 * \text{drive strength of the sized transistor}}{\text{drive strength of the minimum transistor}}$$

Betz's model is used in this dissertation. In later parts of this section, detailed structures of each bus component are described, and the corresponding areas are calculated. Some notations (and their values) used often are listed here for later reference [Betz99]. Appendix A derives the areas of some primitive components, including some listed in Table 3.1.

Notation	Primitive Component	Size
A_{mcel}	one-bit memory cell	6
$A_{mux(F)}$	F:1 mutiplexer	
$A_{mux(2)}$	2:1 mutiplexer	11
$A_{mux(3)}$	3:1 mutiplexer	21
$A_{mux(4)}$	4:1 mutiplexer	23
A_{3st}	tri-state buffer	20
A_{3st_bnd}	tri-state buffer pair at cluster bound (two-bit memory cells included)	35

- Size is scaled in the number of minimum NMOS transistors.
- Areas for multiplexers and tri-state buffers include memory cells associated with selection lines and control lines.

Table 3.1 Area Cost of Some Primitive Components

4.2.2 Connections between CLBs and Buses

As described in the previous section, a cluster is connected to the backbone network via W wires going through tri-state buffers. At the crossbar side, these W wires are bi-directional. At the cluster side, each wire bifurcates into separated incoming/outgoing bus wires (see Figure 4.3) and they are all unidirectional. Incoming/outgoing bus wires as an extended part of the segmented bus are called the local network. CLBs are connected with the local network directly.

With k denoting the bus wire density, there are k incoming bus wires and k outgoing bus wires in each routing channel on average. If a cluster has N*N CLBs, then W = k*N. Here it is assumed that there are only 0.5*k incoming (outgoing) bus wires at boundaries of each cluster, while there are k incoming (outgoing) wires at internal channels. As a contrast, Xilinx Virtex chips have only two long wires along each horizontal channel.

Outgoing bus wires at north, east, north and south side of a CLB are marked as N[0..k-1], E[0..k-1], W[0..k-1] and S[0..k-1], respectively. Similarly, incoming bus wires are marked as *N[0..k-1], E[0..k-1], W[0..k-1]* and *S[0..k-1],* respectively.

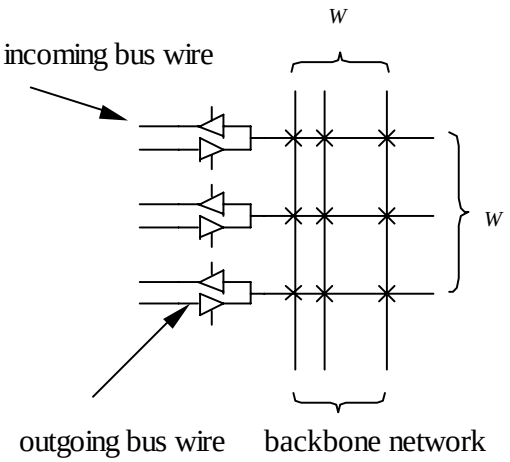


Figure 4.3 Segmented Bus Wires Bifurcate at Tri-state Buffers

For the crossbar, the popularity, P_{CRS_BAR}, is defined as the ratio of the number of intersection points that have NMOS switch transistors deployed over the total number of intersection points. It is used to characterize the crossbar connection flexibility. If on each intersection point of the crossbar, there is a NMOS switch transistor deployed, i.e. P_{CRS_BAR} =1, then this crossbar is called a crossbar with full connectivity. Otherwise,

P_{CRS_BAR} <1, and this crossbar is just a partial crossbar, as only P_{CRS_BAR} of intersection points can conduct.

4.2.2.1 Output Connection Block

Figure 4.4 indicates how CLB output signals are connected to the local network. One specific path is used to illustrate such connections. Each CLB has four BLE output signals, i.e., f, g, h and i. One of them first goes through a sequence of primitive components, the top multiplexer, a turned-on tri-state buffer (marked with a square), and a turned-on switch transistor (marked in square). It then reaches one of the outgoing bus wires at the south side, i.e., S[0] (in bold dash line). At each end of S[0], there is a bifurcating tri-state buffer pair, which can carry signals to the backbone network.

Similarly, the same signal can reach outgoing bus wires at the north, east, west and south (NEWS) directions. They are labeled as N[0..k-1], E[0..k-1], W[0..k-1] and S[0..k-1], respectively. Which direction to take depends on the routing solutions decided by the operating system at run time.

Each tri-state buffer drives k NMOS switch transistors. Only one of them is allowed to conduct at one time. The tri-state buffer at the other end of the wire in bold line is associated with another CLB. Only one tri-state buffer is allowed to drive those k NMOS transistors at a time.

Suppose that there are F signals out of (f, g, h, i) outgoing to bus (F = 1~4), and suppose that each cluster has N*N CLBs, then on each outgoing bus wire, e.g., S[0], there are at most N*F switch transistors hooked on. At one time, only one of them conducts, and the others are turned off.

Outgoing bus wires and wires driven by tri-state buffers compose crossbars. At intersection points of crossbars, NMOS switch transistors with the minimum channel width are deployed.

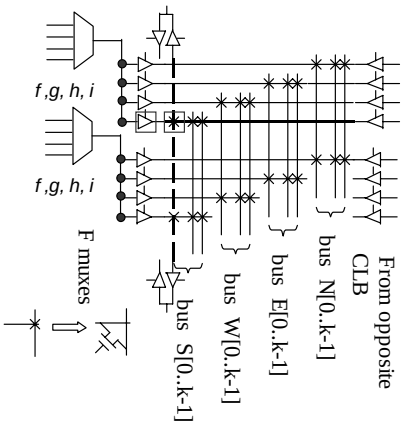


Figure 4.4 Connections between CLB Output Signals and Local Network

Suppose that the popularity of the crossbar is P_{OUT}. The area of the crossbar part can be expressed as:

$$(k*4*F + k*4*F*A_{mcel})*P_{OUT}$$

The constant of four represents four directions, NEWS. The first item corresponds to those switch transistors, while the second item represents the associated configuration memories. This crossbar part is the common estate with nearby CLBs. For each CLB, it just owns one half, i.e.,

$$0.5*(k*4*F + k*4*F*A_{mcel})*P_{OUT}$$

If F = 1, there should be one 4:1 multiplexer to decide which one out of the four BLEs has connections with buses. If F = 4, each BLE drives signal to buses. There is no

need to differentiate them and no multiplexer is needed. If F = 2 or 3, smart decisions have to be made to save area.

Any two out of (f,g,h,i) combinations can be created with two 3:1 mutiplexers. Inputs corresponding to the two multiplexers can be (f,h,i) and (g,h,i), respectively. Similarly, any three out of (f,g,h,i) combinations can be created with three 2:1 mutiplexers. Inputs corresponding to the three multiplexers can be (f,i), (g,i) and (h,i), respectively. Therefore the area of multiplexers can be expressed as $F*A_{mux(5-F)}$.

$$A_{mux(5-F)} = A_{mux(4)}, \text{ if } F = 1.$$
$$A_{mux(5-F)} = A_{mux(3)}, \text{ if } F = 2.$$
$$A_{mux(5-F)} = A_{mux(2)}, \text{ if } F = 3.$$
$$A_{mux(5-F)} = A_{mux(1)}, \text{ if } F = 4 \text{ and } A_{mux(1)} = 0$$

The area of tri-state buffers and associated control memory cells can be expressed as:

$$F*4*A_{3st}$$

In summary, the additional area to each CLB is therefore equal to:

$$0.5*(k*4*F + k*4*F*A_{mcel})*P_{OUT} + F*4*A_{3st} + F*A_{mux(5-F)} \quad \text{(Eq.4.1)}$$

If the function $b_{mux(5-F)}$ is used to represent the number of configuration bits associated with multiplexer $A_{mux(5-F),}$ the function values are listed as follows.

$$b_{mux(5-F)} = b_{mux(4)}=2, \text{ if } F = 1.$$
$$b_{mux(5-F)} = b_{mux(3)}=2, \text{ if } F = 2.$$
$$b_{mux(5-F)} = b_{mux(2)}=1, \text{ if } F = 3.$$
$$b_{mux(5-F)} = b_{mux(1)}=0, \text{ if } F = 4.$$

As each switch transistor on the crossbar or tri-state buffer needs only one configuration bit, the number of configuration bits associated with output connection block is therefore equal to

$$0.5* k*4* F* P_{OUT} + F*4*1+ F*b_{mux(5-F)}$$

4.2.2.2 Input Connection Block

Connections between CLBs and incoming bus wires are shown in Figure 4.5. According to Betz's CLB model, each CLB has four BLEs, and each BLE is associated with four 14:1 full multiplexers. It is not difficult to expand a 14:1 full multiplexer into a 16:1 full multiplexer. Only another two input pins are needed, shown in bold lines in Figure 4.5. Corresponding to the two inputs, another two NOMS transistors with the minimum channel width are added to the original 14:1 multiplexer. Signals from incoming bus wires are fed into CLBs with these two additional pins. For each CLB, extra 32 transistors are needed to extend mutiplexers inside the CLB.

Each CLB is associated with four groups of incoming bus wires at different directions. Each group has k wires. They are *N[0..k-1], E[0..k-1], W[0..k-1]* and *S[0..k-1]*. At one time only one out of the k signals is selected via a 4:1 multiplexer. Hence at most two out of k signals can be selected to feed into the CLB. Incoming bus wires and wires associated with those 4:1 multiplexers compose crossbars again. At intersection points of this crossbar, NMOS switch transistors with minimum channel width are deployed again.

Figure 4.5 illustrates how an incoming signal reaches a CLB. A signal from the backbone network first goes through the tri-state buffer in square, then an incoming bus wire N[0] (in bold dash line), followed by the 4:1 mutiplexer at upper right, and finally reaches those extended 16:1 multiplexers in the CLB.

For a cluster with N*N CLBs, along each incoming bus wire, e.g., *N[0]*, there are at most 2*N switch transistors hooked on. But only very few of them will conduct,

because for a well designed IP circuit, signal delivery relies more on its internal routing rather than the local bus network.

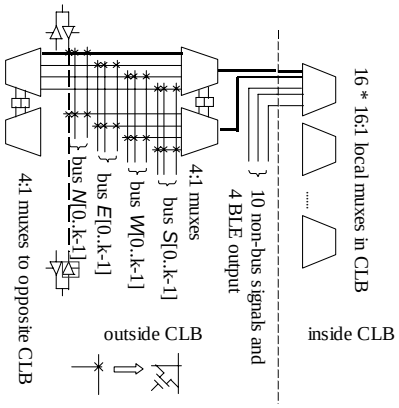


Figure 4.5 Connections between CLB Inputs and Local Network

If the popularity of the crossbar is $P_{IN,}$ then the area cost of the crossbar is equal to

$$k*4*P_{IN} + k*4*A_{mcel}*P_{IN}$$

The constant four represents bus wires at four sides of a CLB. Each NMOS switch transistor is controlled by a one-bit memory cell. That is why the second item.

The area of the two 4:1 multiplexers associated with the CLB can be expressed as

$$2*A_{mux(4)} - 2*A_{mcel.}$$

As the area of one multiplexer A_{m4} includes memory cells associated with selection lines already, the negative part comes from the sharing of memory cells between the two 4:1 multiplexers.

In summary, the additional area associated with one CLB is therefore equal to

$$0.5*(k*4 + k*4*A_{mcel})*P_{IN} + 2*A_{mux(4)} - 2*A_{mcel.} + 2*16 \qquad \text{(Eq. 4.2)}$$

The coefficient 0.5 again comes from the sharing of crossbars between nearby CLBs. The last item is because of the extended multiplexers inside the CLB.

As each switch transistor on the crossbar needs only one configuration bit, and the two 4:1 multiplexers need only two configuration bits, the number of configuration bits associated with the input connection block is therefore

$$B=0.5* k*4 * P_{IN} + 2$$

4.2.3 Bus Switch Block

If a chip has m*m clusters, then there are totally (m+1)*(m+1) bus switch blocks. It is assumed that the disjoint switch block architecture is used. On a normal bus switch block, as shown in Figure 4.6, six tri-state buffer pairs are needed to switch an incoming signal on one side to any of the other three sides.

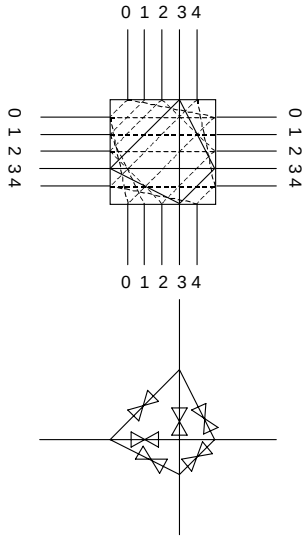


Figure 4.6 Bus Switch Block

The number of wires on each side of the switch block is W, the same as the number of wires going through a cluster. Totally W*6 tri-state buffer pairs are needed. That is, totally

$$W *6*2$$

bi-directional tri-state buffers are needed. Each switch point is associated with two one-bit SRAM cells. Therefore the area of a bus switch block is:

$$A_{SW\_BLK} = W *6*A_{3st}*2 \qquad \text{(Eq. 4.3)}$$

At boundaries, the structure of bus switch blocks becomes simpler because of reduced routing flexibility. For example a bus switch block at the left boundary can be shown like Figure 4.7 (a). Hence its area is just half of that of an internal bus switch block. Figure 4.7 (b) represents a bus switch block element of a bus switch block on the right boundary.

Two such bus switch blocks can be combined into one normal switch block when their areas are calculated. Switch blocks at corners have even simpler structures. One such switch block has only 1/6 of tri-state buffers compared with a normal bus switch block.

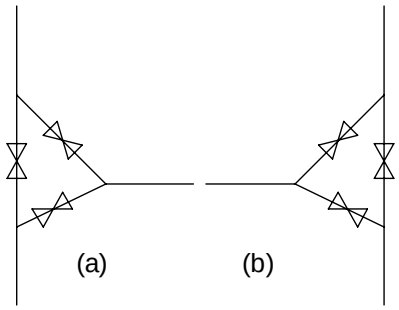


Figure 4.7 Bus Switch Blocks at Chip Boundaries

Because of the above reasons, the area of (m+1)*(m+1) bus switch blocks is equivalent to that of (m*m) –1/3 normal bus switch blocks, or roughly (m*m) bus switch blocks.

Because each tri-state buffer pair needs two bits, the configuration cost can easily be decided as:

$$B_{SW\_BLK} = W*6*2$$

4.2.4 Crossbar between Clusters

Clusters are connected to the segmented bus via crossbars. Totally there are

$$W*W*P_{CRS\_BAR}$$

switch transistors on each crossbar, and the same amount of associated configuration memory cells. Here it is assumed that the minimum width NMOS transistors are used as switches. The area of such a crossbar is:

$$A_{CRS\_BAR} = W*W*P_{CRS\_BAR} + W*W* A_{mcel} *P_{CRS\_BAR} = 7*W*W*P_{CRS\_BAR} \quad \text{(Eq.4.4)}$$

Its configuration cost is

$$B_{CRS\_BAR} = W*W*P_{CRS\_BAR}$$

4.2.5 Connection between IOBs and Segmented Bus

On boundaries of the chip, IOBs are supposed to be hooked on backbone crossbars directly. Hence it is assumed that IOBs do not incur additional area or configuration cost to the crossbar. This assumption is made because IOBs exist independent of whether segmented buses are used or not. On the boundaries of real FPGA chips, frequently there are commonly two (usually no more than four) IOBs that fit into one row (column).

4.2.6 Tri-state Buffers at Cluster Boundaries

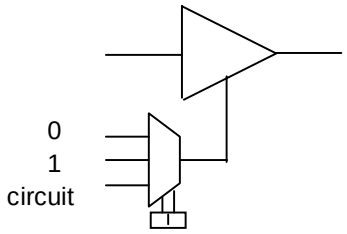


Figure 4.8 Tri-state Buffer at Cluster Boundary

Figure 4.8 shows a tri-state buffer at a cluster boundary. The enable pin of the buffer can be driven by a circuit inside the cluster, e.g., a finite state machine or fixed voltage levels. This can be achieved by controlling the multiplexer through varying the contents in the two bit memory cells. During the time of partial reconfiguration, such tri-state buffers that are next to the bus being configured are disabled by the operating system. After configuration, they may be controlled by a circuit inside a cluster.

On each side of a cluster, there are W bi-directional tri-state buffer pairs associated with the cluster. Each tri-state buffer in the pair is controlled by two-bit memory cells. The area of these tri-state buffers and associated memory cells is:

$$A_{BND\_BUF} = 4* W * A_{3st\_bnd}*2 \qquad \text{(Eq.4.5)}$$

Its configuration cost is:

$$B_{BND\_BUF} = 4* W * 2*2$$

4.2.7 Total Segmented Bus Area and Configuration Bits

The extra area per CLB comes from the local network, or Eq. 4.1 and Eq. 4.2. Therefore

$$A_{CLB} = 0.5*(k*4*F+k*4*F*A_{mcel})*P_{OUT} + F*4*A_{3st}+F*A_{mux(5-F)} +$$

$$0.5*(k*4+k*4*A_{mcel}) *P_{IN} + 2*A_{mux(4)} - 2*A_{mcel} +32 \qquad \text{(Eq.4.6)}$$

The extra area per cluster block is due to the extra area for N*N CLBs, one bus switch block and two crossbars. That is,

$$A_{CLUST} = N*N*A_{CLB}+2*A_{CRS\_BAR}+A_{SW\_BLK} +A_{BND\_BUF.} \qquad \text{(Eq. 4.7)}$$

Suppose the chip has totally m*m clusters. There are 2*m backbone network crossbars at the top and left boundary that are not associated with any cluster block. The total segmented bus area is as follows.

$$A_{CHIP} = m*m* A_{CLUST} + 2*m*A_{CRS\_BAR} \qquad \text{(Eq. 4.8)}$$

This area is expressed in the number of transistors. It can be converted into the equivalent number of CLB blocks since a CLB block contains 8,390 (=1,678*5) transistors.

Following a similar procedure, it can be decided that the configuration cost associated with a CLB is:

$$B_{CLB} =0.5* k*4* F* P_{OUT} + F*4*1+ F*B_{mux(5-F)} + 0.5*k*4 *P_{IN} + 2$$

The configuration cost associated with a cluster is:

$$B_{CLUST} = N*N*B_{CLB}+2*B_{CRS\_BAR}+B_{SW\_BLK} +B_{BND\_BUF.}$$

The total configuration cost for the chip is therefore:

$$B_{CHIP} = m*m* B_{CLUST} + 2*m*B_{CRS\_BAR} \qquad \text{(Eq. 4.9)}$$

4.3 Area Cost and Configuration Cost Computation Results

In this section some chip examples are used to illustrate the equations derived in the previous section.

4.3.1 Area Cost

The area cost of segmented bus structure is evaluated based on Equation 4.8 in Section 4.2.7.

Here an imaginary FPGA chip with a traditional architecture is chosen as a reference. This chip has 120*120 CLBs. In terms CLB numbers, it is close to a high-end Xilinx Virtex chip. For a specific bus wire density k and a fixed cluster size N, the number of clusters a chip has (m^2) may be varied. Let m fall in the range of $\left\lfloor \frac{120}{N+1} \right\rfloor$ ~$\left\lceil \frac{120}{N} \right\rceil$. Among all these possible value, m is chosen so that the chip has an area closest to 120*120 CLB blocks. It means that the imaginary chip with a traditional architecture and the chip with square clusters have similar numbers of CLBs.

From equations Eq. 4.1 ~ Eq. 4.9, it is clear that many factors may influence the chip area. To have a more practical architecture design space, the following assumptions are made.

1. The maximum cluster size is set as 14*14, and the maximum k is set as eight. Bus widths may vary a lot, but have to be practical. A UART has only a few wires. A PCI bus is composed of around 100 wires after discounting those redundant VCC pins, GND pins and reserved pins. Buses too wide are not

considered in the following computations. For embedded systems, a bus width of a few dozens of wires should be enough in most cases.

2. The number of signals from CLB outputs going to buses, F, is set as two. In Xilinx Virtex chips, next to each CLB there are only two tri-state buffers, which can be used to hook the CLB on long wires. It is not difficult to calculate for other F values.

3. Initially P_{CRS_BAR}, P_{OUT} and P_{IN} are set as one, i.e., all crossbars are fully populated.

Computation results are plotted in Figures 4.9, 4.10 and 4.11. The horizontal coordinates represent the cluster size along one dimension. From these figures, it can be found that

(1) The smaller the cluster size is, the bigger the segmented bus overhead is. When the chip is partitioned into finer clusters, more bus segments, especially those on the backbone network, are needed.

(2) The higher the number of bus wires inside each rouging channel is, the higher the segmented bus overhead is. The addition of two wires in each channel incurs around 5% area increase.

(3) The proposed segmented bus is not very expensive. For cases of k =2 or k=4, the bus overhead is mostly less than 10%. With k =6, the bus overhead in most situations occupies 11%~ 15% of the total chip area. With k=8, bus overhead falls in the range of 16%~21%.

(4) The minimum number of CLB occurs at (k=2, N=14). There are still 12500 CLB (blocks) for circuit placement, and the segmented bus

structures consume almost 500 CLB blocks. Compared with the original 120*120 CLB (blocks), around 1400 CLB blocks are removed. The biggest chip area happens at (k=8, and N= 13). At this point, there are 13689 CLB blocks for circuit placement, and the bus overhead is 2625 CLB blocks. Compared with the original chip with 120*120 CLB blocks, another 1914 CLB blocks are added.

(5) On Figure 4.11, it can be found that chip areas vary more when the cluster size increases. At the point (k=2, N=11), the chip area decreases dramatically, compared with (k=2, N=10) and (k=2, N=12). In the range of $\left\lfloor \frac{120}{11+1} \right\rfloor \sim \left\lceil \frac{120}{11} \right\rceil$, when m takes the value of 10, the chip area is 13067 CLB blocks; when m takes the value of 11, the chip area is 15808. The former is closer to 120*120. That is why the "V" shape on the curve.

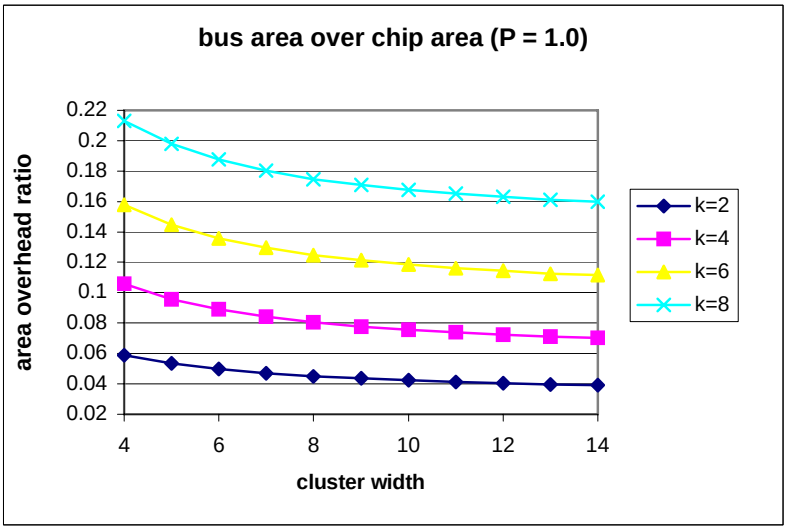


Figure 4.9 Bus Area Overheads Over Total Chip Areas (P =1.0)

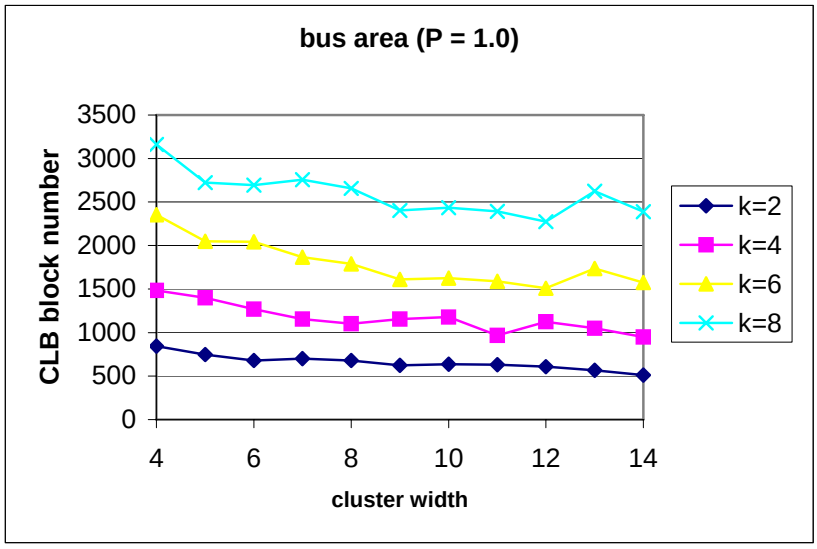


Figure 4.10 Bus Areas In Terms of CLB Blocks (P =1.0)

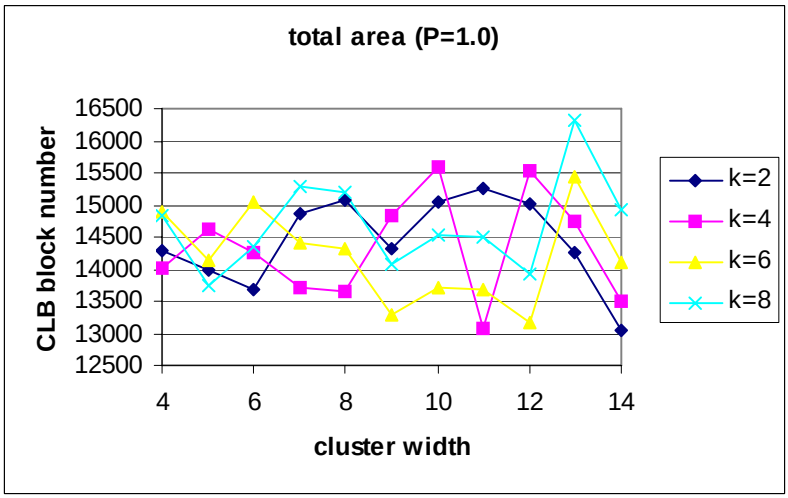


Figure 4.11 Chip Areas in Terms of CLB Blocks (P=1.0)

Note that some of the above parameter settings may not be realistic. For example, in the case of (N=4, k=8 and F=2), a 4*4 cluster may be connected to the segmented bus through 32 wires. That means each CLB can send two signals to the segmented bus at the same time. This setting may have too high a connection flexibility.

In the suggested architecture, three different kinds of crossbars are used. They are associated with local incoming buses, local outgoing buses and the backbone network, respectively. They are all fully populated crossbars, i.e., on each intersection point there is a switch transistor.

A fully populated crossbar is expensive in terms of area cost due to the large amount of switches and associated tri-state buffers that may need higher driving capability. Another drawback is the degrading of speed performance. When too many switch transistors are hooked on a wire, they cause heavy capacitance load even if only a portion of them conducts. On the other hand, bus protocols are explicitly defined, and a well-designed IP circuit has clear interface to bus. Bus signal distribution should rely more on routing resources inside IP circuits rather than high connection flexibility to bus wires. Limited connection flexibility may still be needed, which can give IP circuit designers certain space to trade off.

In summary, the area cost of the segmented bus can be further reduced if crossbars of smaller popularities are considered. In Figures 4.12, 4.13 and 4.14, it is assumed that all those crossbars are only 50% populated. From these figures, it is found that the area costs are further reduced, and chips can therefore have more space to accommodate IP circuits.

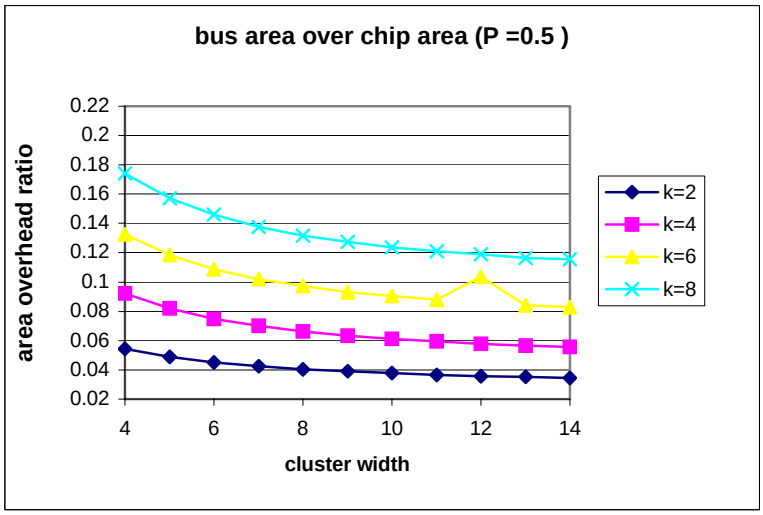


Figure 4.12 Bus Area Overheads Over Total Chip Areas (P =0.5)

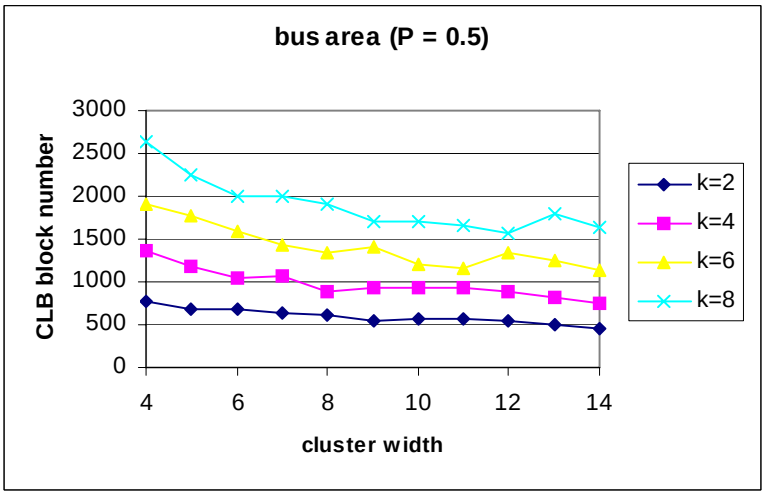


Figure 4.13 Bus Areas in Terms of CLB Blocks (P =0.5)

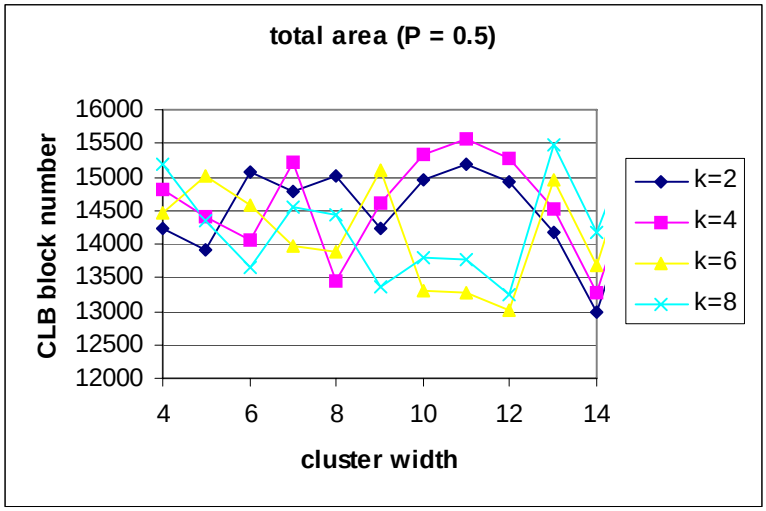


Figure 4.14 Chip Areas in Terms of CLB Blocks (P=0.5)

When Figure 4.12 is compared with Figure 4.9, it can be found that for k=2, there is only around 0.4%~0.5% percent area reduction; for the k=4 case, the area reduction is around 1.4%~1.5%; for the k=6 case, the reduction is 2.6%~2.9%; and for k=8, 4%~5%.

4.3.1.1 Comparison with Wormhole Routing

Here the segmented bus is compared with the wormhole packet-switching network in terms of area overhead.

T. Marescaux et al. built such a wormhole network on XC2V6000 [Marescaux04]. Routers on a 3*3 mesh network consumed 2800 Virtex2 slices, or 8.3% out of all resources. In each reconfiguration tile, the NIC (network interface component) part consumed 361 slices, or 1.81% of all resources. Therefore totally routers and NICs consumed

$$8.3\% + 3*3*1.81\% = 24.5\%$$

of all resources, i.e., around 2070 CLBs (each CLB has four slices). In Section 4.3.1, chips which have 9*9 clusters are used. Each cluster has 13*13 CLBs, or 13*13*4 slices. The calculation area results are listed in Table 4.1. From the table, it can be found that except for the ($K = 8$ and $P = 1$) situation, the segmented bus structure generally uses fewer CLB blocks. As the segmented bus structure has wider data width, it can have higher bandwidth. It should be noted, however, that the segmented bus data are not obtained from implementation on real FPGA chips.

			K=2	K=4	K=6	K=8
Segmented Bus 9*9 mesh	Data width (bits)		26	52	78	104
	Chip Area	P=0.5	14188	14511	14945	15492
	Bus Area		499	822	1256	1803
	Bus Weight		3.5%	5.7%	8.4%	11.6%
	Chip Area	P=1.0	14256	14739	15425	16314
	Bus Area		567	1050	1736	2625
	Bus Weight		4.0 %	7.1%	11.3%	16.1%
Wormhole Net 3*3 mesh	Data width (bits)	16				
	Chip Area	96*88				
	Bus Area	2070				
	Bus Weight	24.5%				

* Area is scaled in CLBs (blocks)

Table 4.1 Area Cost Comparison between Circuit Switching and Packet Switching

4.3.1.2 Comments on Area Cost

From the previous analysis, the proposed segmented bus structure is quite practical in terms of area overhead. It provides an in-expensive mechanism to support on-chip inter-IP or IP-IO connections under a dynamically partial reconfiguration environment.

4.3.2 Configuration Cost

Using Eq. 4.9, configuration costs corresponding to each case in Section 4.2.7 can be calculated. Results are depicted in Figure 4.15 and Figure 4.16. It can be seen that about 0.5 M ~3 M bits are needed to configure the segmented bus structure. For a Xilinx XC2V8000 chip, which has 112*104 CLBs, it needs totally 26M (26,174,720) bits (header bits not included) to configure. For all cases in Figure 4.15 and 4.16, the minimum number of CLBs is 11664 (N=12, k=8, m =9), and the maximum number of CLBs is 13689 (N=13, k=8, m=9). Therefore FPGA chips used in these calculations are a bit larger than XC2V8000. Assume a little more than 26M bits are needed to configure the non-segmented-bus part. An additional 2% ~ 12% ($0.5/26 \sim 3/26$) increase in configuration bits is needed to support the segmented bus. Note that k=6 and k =8 cases are not very likely to happen, because bus wires are too densely populated (considering each CLB has only four BLEs). Nevertheless, even a 12% additional configuration bits are not too heavy an overhead.

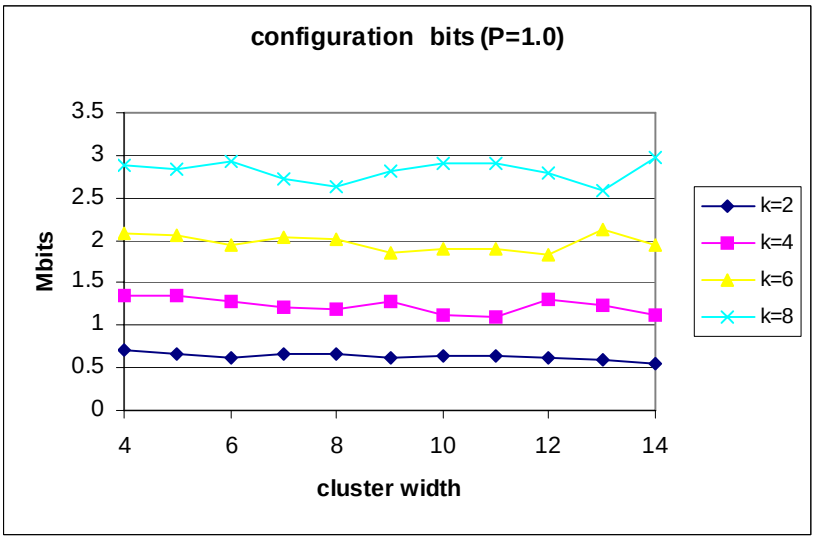


Figure 4.15 Configuration Cost (P =1.0)

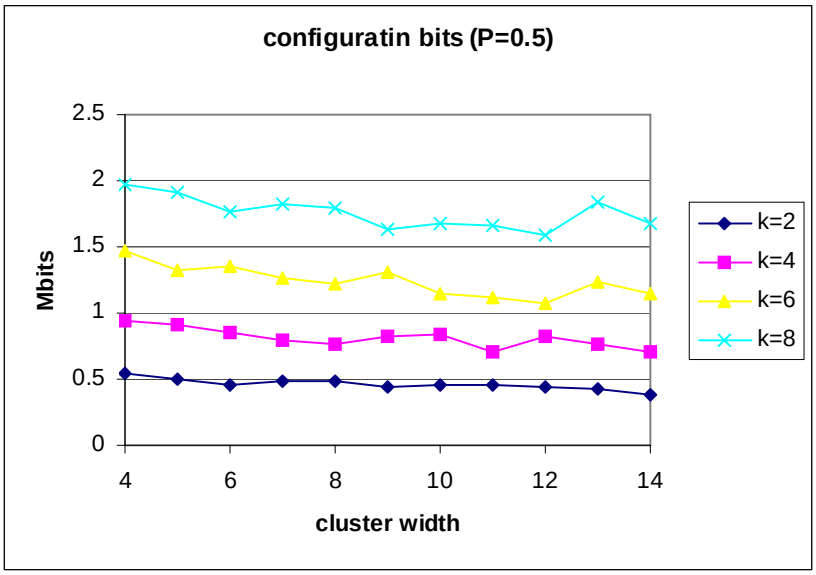


Figure 4.16 Configuration Cost (P =0.5)

5 The Operating System Kernel

Just like a traditional computer, a reconfigurable machine can use an operating system to manage its resources, to schedule tasks and to dispatch tasks. Operating system for reconfigurable computer is called **OS4RC** in short. In this research, a computational task can be carried out by more than one IP circuits. In this chapter, the three basic components of an **OS4RC** are described:

(1) Placer: it searches for a suitable spatial slot to accommodate each circuit (an IP, Intellectual Property).

(2) Router: it builds inter-IP or IP-IO communication channels, which are necessary for computing.

(3) Scheduler: it decides the order in which the submitted circuits are executed so as to optimize some optimal criteria, such as the maximization of the utilization of the reconfigurable machine, or to guarantee the task fairness. In this research, the shortest total execution time is emphasized.

Routing on the segmented bus is a special feature of the OS4RC kernel in this chapter. As described in Chapter 2, placement and scheduling problems have been researched quite a lot in many existing **OS4RC** works [Bazargan00] [Steiger04] [Diessel98], while inter-IP or IP-IO interconnection problems were rarely considered. In some works, connections were based on packet switching. To the best knowledge of the author, no similar circuit switching work has been proposed.

This chapter is organized as follows. Section 5.1 provides an overview of the kernel structure. Section 5.2 introduces an important data structure *staircase*. All those tasks closely related to the *placer*, such as vacant slot searching, and vacant or occupied cluster maintenance, are based on such a data structure. Section 5.3 outlines algorithms that are built in the placer. Section 5.4 describes the algorithm built into the router. Section 5.5 shows how the task scheduler works.

5.1 OS4RC Overview

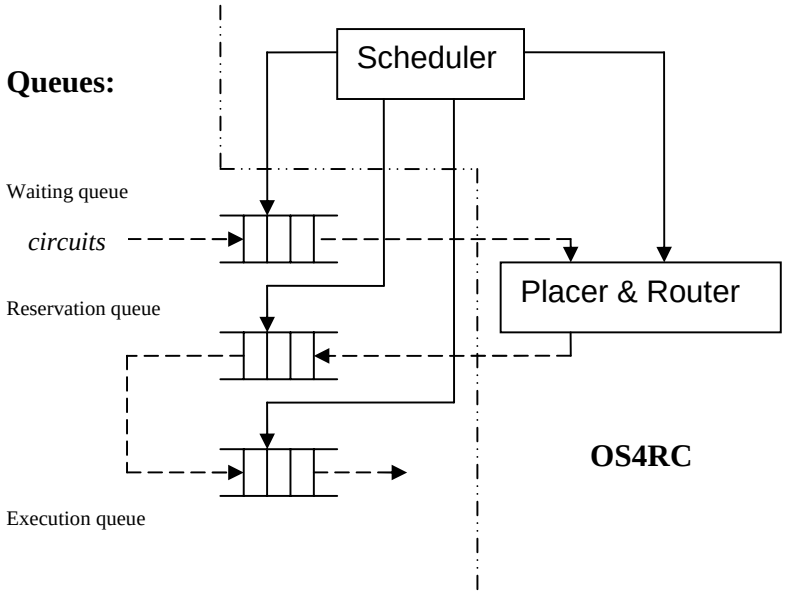


Figure 5.1 Overview of the Proposed OS4RC Kernel Structure

The structure of the OS4RC kernel can be illustrated in Figure 5.1. Circuits submitted to the reconfigurable machine go through different stages before their exiting from the system. The task flow is indicated by those dashed arrow lines. Each stage is characterized by a different queue. There are three queues: waiting queue, reservation queue and execution queue. The exact meaning of each queue is described later. The

scheduler monitors those queues, and moves those circuits from one queue to another, depending on the availability of resources. Circuits cannot move from the waiting queue to the reservation queue unless the corresponding placement and routing have completed.

Placement and routing in **OS4RC** at run time are quite different from corresponding problems at compile time. At compile time, a circuit well documented in a hierarchical and modular HDL description is first flattened. A placement algorithm, normally based on simulated annealing is then applied to a set of flattened CLBs, and then a routing algorithm builds connections between those CLBs. At run time an **OS4RC** cannot afford such an expensive placement and routing. For an IP circuit at run time, connections and the relative positions of its internal building blocks (CLBs) should be preserved. Routing between circuits can only be based on very simple and regular architectures.

As described in Chapter 4, with a mesh like segmented bus, the new chip architecture can support more flexible IP-IO or inter-IP connections. Communication channels are configured on the fly, depending on circuit locations and available routing resources at that moment. At run time, every circuit has communication channels by attaching to nearby bus segments. Just like the circuit switch mode in telecommunication, after a circuit is swapped out of the chip, resources occupied by its communication channels are released for use by later circuits.

5.2 Representation and Management of the 2-D FPGA Real Estate

As the prerequisite work of placement, all resources on an FPGA chip have to be expressed in a well designed data structure. This is so that when circuits are added to or removed from the FPGA chip, the corresponding data structure maintenance and therefore resource management can be manipulated efficiently.

Due to the dynamic swapping in and out of circuits, after some time the distribution of empty clusters and occupied clusters will be very irregular. To simplify the discussion, circuits are assumed to be a series of rectangles. For example, in Figure 5.2, an FPGA chip has totally $8 \times 8 = 64$ clusters. Every occupied cluster is represented as a "1" (identified with different patterns for different circuits), while every empty cluster is represented as a "0".

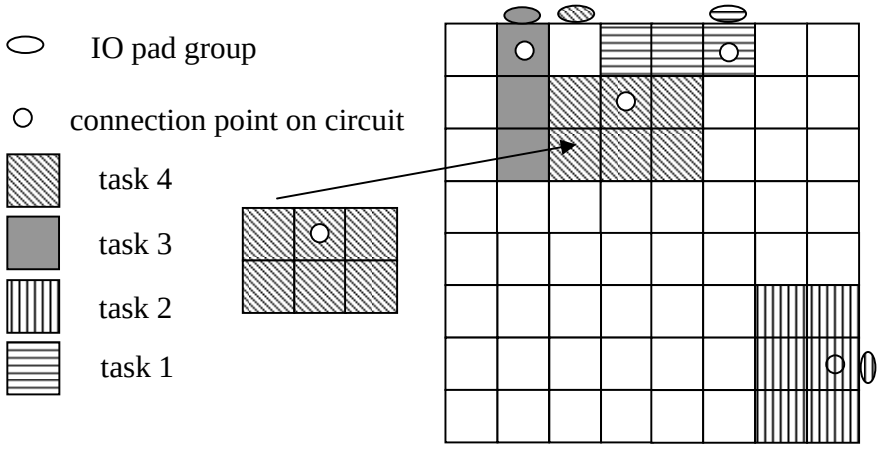


Figure 5.2 An Example of Dynamic Hardware Circuit Swapping

Because of the 2-D organization, the space management task in an **OS4RC** is far more complex than the page management work in traditional operating systems. How to manage the 2-D space real estate efficiently has long been a topic in **OS4RC.** Different

authors gave out different real estate representations, such as quad tree, approximated maximal rectangle, real maximal rectangle and so on [Bazargan00] [Steiger04] [Diessel98]. The quad tree method and approximated maximal rectangle method are shown in Figure 5.4 and Figure 5.4, respectively. In the case of the quad tree expression, when a vacant rectangle straddles over boundaries of different quadrants, the job of searching and data structure maintenance become very complicated. Those occupied quadrants may not be at the same level of the tree, as indicted in Figure 5.3. In Figure 5.4, the vacant area can be approximated with rectangle *ABDC* plus *FGHD* , or *ABFE* plus *EGHC* . Different criteria were given by Kiarash Bazargan [Bazargan00]. Due to the approximation of rectangles, whether a new circuit with the size of *JGHK* can fit into a vacant slot depends on how the vacant area is represented. In Figure 5.4, the new circuit can fit if the vacant area is split as *ABFE* plus *EGHC* . However it cannot fit if the vacant area is split as *ABDC* plus *FGHD* . In terms of search quality, the approximated rectangle method is not very good, especially when there is not much real estate.

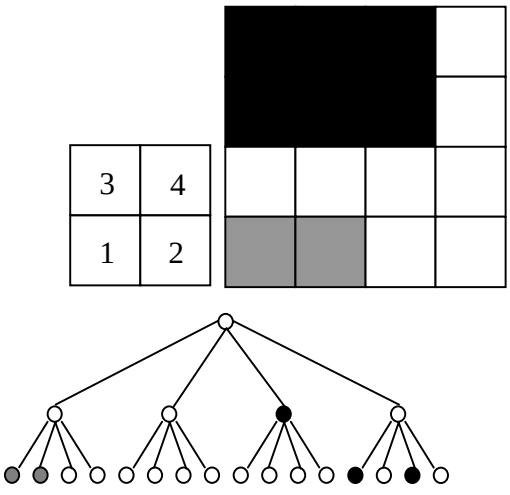


Figure 5.3 An Example of Quad Tree Expression

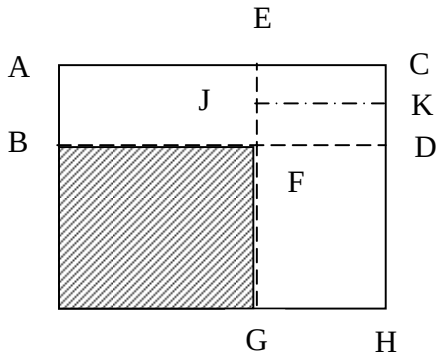


Figure 5.4 An Example of Approximated Rectangles

To address the constraint of approximated rectangle expression, the data structure suggested by Jeff Edmonds is adopted [Edmonds03] to characterize those vacant chip areas, even though this method was originally designed for the purpose of data mining rather than for **OS4RC.** Manish Handa was the first researcher to adopt this data structure for OS4RC [Handa04a]. Based on Jeff Edmonds' data structure, it is possible to develop an efficient vacant area search engine, which can greatly reduce the search time. And this search engine can be one of the hardware tasks on the chip.

Jeff Edmonds' Method

Jeff Edmonds suggested a data structure called *staircase* to characterize those "0" elements in an array. A *staircase*(x, y) is in fact a stack of partially overlapping rectangles with (x_i, y_i) as their upper-left corners and with the same point (x, y) as their lower-right corners.

For example, in Figure 5.5, *staircase*$(x_0, y_0) =< (x_1, y_1), (x_2, y_2), (x_3, y_3) >$ is defined, where the lower-right point (x_0, y_0) is the common lower-right corner of

rectangles $<(x_1, y_1),(x_0, y_0)>$, $<(x_2, y_2),(x_0, y_0)>$ and $<(x_3, y_3),(x_0, y_0)>$. That staircase is stored as a list of integer pairs, i.e., (x_0, y_0), (x_1, y_1), (x_2, y_2) and (x_3, y_3).

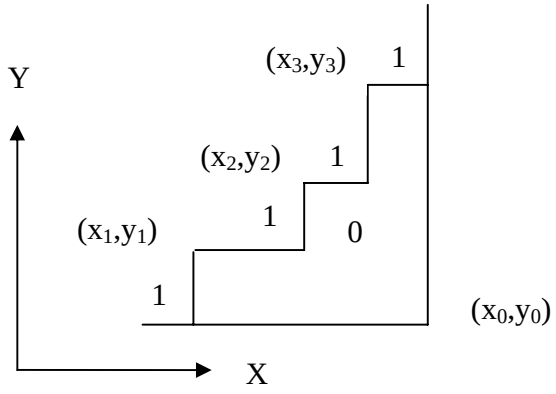


Figure 5.5 The Definition of Staircase

Among all these rectangles composing of elements of "0", some of them may be maximal as defined below.

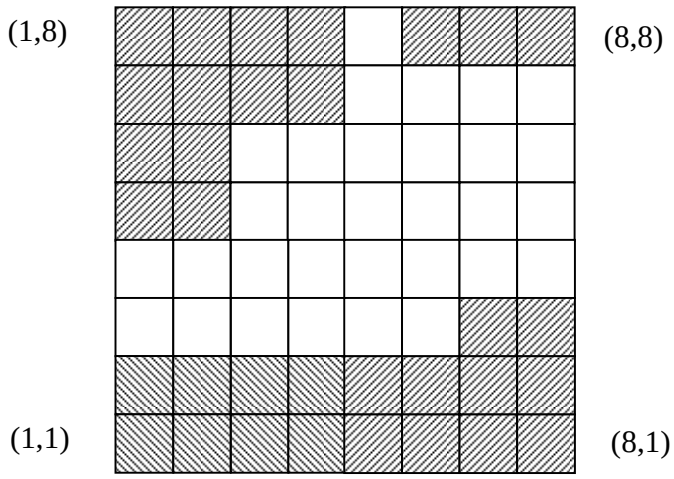


Figure 5.6 An Example of Maximal Rectangles and Corresponding Staircases

Definition: an empty rectangle is maximal if it cannot be extended along either the X- or Y- axis because there is at least one "1" entry on each of the borders of the rectangle to block its extending further [Edmonds03].

In Figure 5.6, maximal rectangles are $<(1,4),(6,3)>$, $<(3,6),(6,3)>$, $<(5,7),(6,3)>$, $<(5,8),(5,3)>$, $<(1,4),(8,4)>$, $<(3,6),(8,4)>$ and $<(5,7),(8,4)>$. They are included in *staircase*(5,3) , *staircase*(6,3) and *staircase*(8,4) , which together are used to request the current vacant spaces. Intermediate *staircase*(4,3) will be replaced with *staircase*(5,3) as the staircases are being constructed. This will be seen later.

All those empty maximal rectangles not only cover all empty resources inside the array, but also sharply reduce the number of rectangles to maintain. It has been proved that any maximal rectangle will be contained in one and only one staircase [Handa04a]. But not every rectangle contained in *staircase*(x, y) is maximal.

Obviously, if a waiting circuit cannot fit in any empty maximal rectangle, then there is no vacant space available for the circuit at this time. Otherwise, it is better to pick the smallest empty maximal rectangle it can fit so as to reduce the fragmentation. After a circuit is swapped in (out), corresponding array elements are marked as "1"s (or "0"s). These operations are not complex, but many existing maximal rectangles will be disrupted. Manish Handa has shown that not all maximal rectangles have to be updated. Suppose the top left corner of the newly added (removed) circuit is (m, n), only those covered by the area below and to the right of $(m-1, n+1)$ have to be updated as Figure 5.7 indicated [Handa04b].

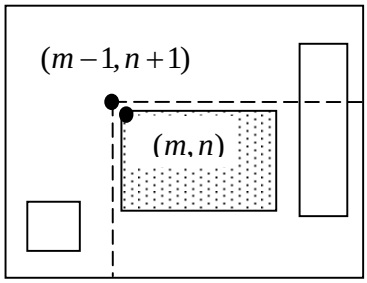


Figure 5.7 Those Disturbed Staircases

Jeff Edmonds suggested an efficient method to construct all staircases contained in the array: scanning the matrix from left to right and from top to bottom row by row, all staircases can be constructed in one pass. It is summarized as follows:

Let y_r be the Y coordinate of the top most (with the biggest Y coordinate) "0" element in $staircase(x, y)$; while let y_r' be the similar value extending from $staircase(x-1, y)$. The contents of $staircase(x, y)$ can be constructed from $staircase(x-1, y)$ easily. If the entry at (x, y) is a "1" element, then $staircase(x, y)$ does not exist, or $staircase(x, y) = null$. Otherwise, three different cases may happen as in Figure 5.8.

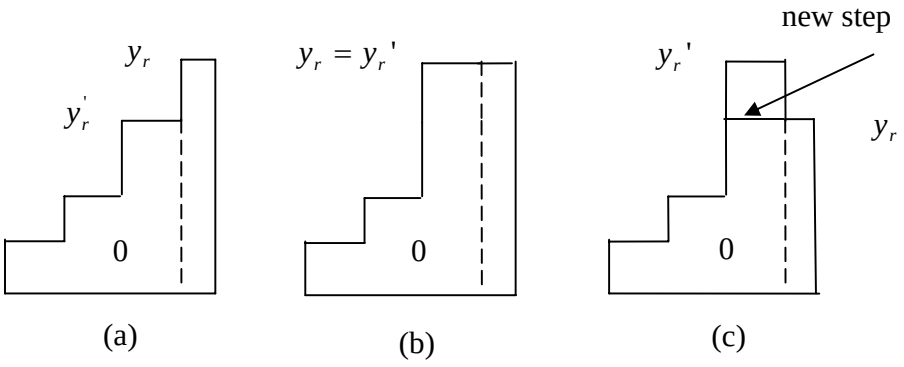


Figure 5.8 Three Possibilities When $staircase(x, y)$ **is Derived from** $staircase(x-1, y)$

(a) $y_r > y_r'$. This means $staircase(x, y)$ can be constructed from $staircase(x-1, y)$ by adding another stair step (x, y_r) to $staircase(x-1, y)$.

(b) $y_r = y_r'$. This means $staircase(x, y)$ is exactly equal to $staircase(x-1, y)$, except the lower right corner is shifted one step to the right.

(c) $y_r < y_r'$ .This means $staircase(x, y)$ can be constructed from $staircase(x-1, y)$ by cutting off all those steps in $staircase(x-1, y)$ which are higher than y_r. A new step with a height of y_r may be added as the last entry of $staircase(x, y)$ if there is no step with this height in the original $staircase(x-1, y)$.

Fortunately, it is not necessary to calculate $y_r(x, y)$ by counting the number of "0" elements extending up from the lower right corner (x, y) . The relationship $y_r(x, y) = y_r(x, y+1)$ holds if both the entries at (x, y) and $(x, y+1)$ are "0"s. Otherwise

$y_r(x, y) = y$ if the entry at (x, y) is "0" and the entry at $(x, y+1)$ is not. If the entry at (x, y) is not "0", then $y_r(x, y) = null$, corresponding to $staircase(x, y) = null$.

For a block of "0" entries that starts from $(x, y-1)$ and extends to the left, let $x_*(x, y)$ denote the X coordinate of the left most "0"entry. Similarly, for a block of "0" entries that starts from $(x+1, y)$ and extends up, let $y_*(x, y)$ denote the Y coordinate of the up most "0"entry. Jeff Edmonds gave the necessary and sufficient condition to decide whether a rectangle $< (x_i, y_i), (x, y) >$ is maximal:

> **Consider a step in** $staircase(x, y)$ **with a top-left corner** (x_i, y_i)**. The rectangle** $< (x_i, y_i), (x, y) >$ **is maximal if and only if** $x_i < x_*(x, y)$ **and** $y_i > y_*(x, y)$**.**

According to this necessary and sufficient condition, maximal rectangles can be extracted from $staircase(x, y)$ by throwing away those non-maximal rectangles. In this way redundancy of data is removed, and the search can be accelerated. Finally all those maximal rectangles can be put into a list, and this list is organized in an ascending order of maximal rectangle size.

5.3 Placement

Placement is to pick up an appropriate empty spot for an arriving circuit according to certain criteria. Frequently higher utilization of FPGA chip area is emphasized.

Whenever the candidate maximal rectangle is found, decisions have to be made on how to fit the circuit into this maximal rectangle. In previous works where the inter-IP or IP-IO connection problem is not considered, a bottom left corner biased placement is often used, just like Figure 5.9(a).

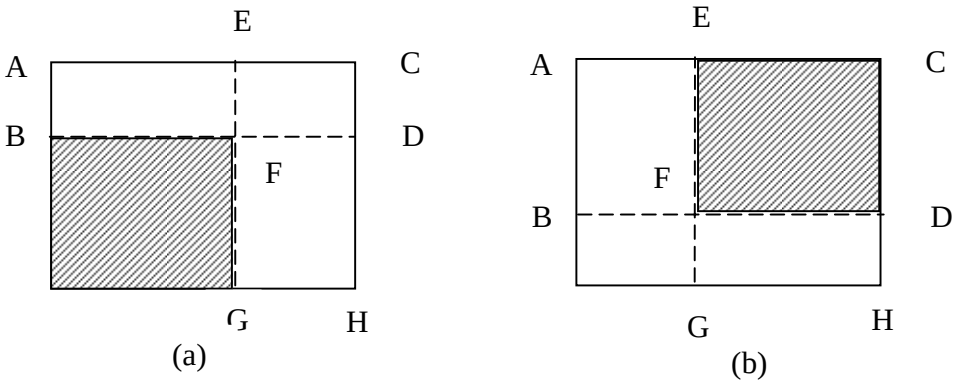


Figure 5.9 Two Possible Circuit Fit-in Strategies

Steiger et al. suggested a placement method based on the prediction of the most recently available rectangle [Steiger04]. If a new circuit arrives, a vacant spot is searched based on the currently available resources. If such a spot is not available right away, the search is then based on the expected resources after a circuit exits in the most recent future. This procedure continues with more circuits exiting until a suitable vacant spot is available. The worst case happens when all current on-chip circuits need to exit from the chip.

A similar strategy is used here, but the vacant spot search is based on true maximal rectangles rather than Steiger's "non-overlapping free rectangles". Empty maximal rectangles may overlap with each other. But for a given allocation, the set of empty maximal rectangles is unique, while the set of "non-overlapping free rectangles" is not. The search and management of empty maximal rectangles is very time-consuming, compared with those of non-overlapping free rectangles. In our situation, the minimum rectangle is a cluster rather than a CLB. Using approximated maximal rectangles will cause too much waste. When a chip has only a few hundreds of clusters, an exhaustive search over such a chip is not time consuming.

For the reason of providing contrast, situations with and without segmented bus are both discussed in later sections. For the situation without bus, communications between two circuits are still possible when two circuits are close to each other. Similarly, a circuit can talk with external world when it is on the chip boundary. But abutment is not enough; alignment is still necessary to make sure correct relative positions between two components communicating with each other. Therefore the abutment-and-alignment (AAA) strategy is applied.

One connection point can be a cluster or an IO pad group. If a connection point on one circuit is aligned with another connection point on another circuit (or chip boundaries), then the two circuits are said aligned and they satisfy the AAA requirements. In Figure 5.10, circuits c_{i0}- c_{i2}, c_{i0}-c_{i3}, c_{i3}-P satisfy the AAA requirements at different directions. P is an IO pad group on the chip boundary.

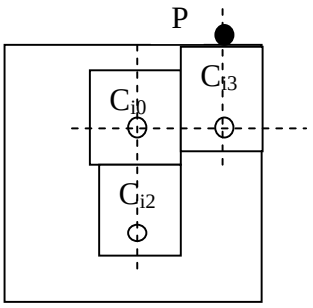


Figure 5.10 Abutment and Aligned Placement

Because of constraints of AAA requirements, if dedicated communication channel is not available, vacant rectangle size is not the only consideration when a circuit is placed. On the contrary, with segmented bus, two circuits can exchange data even when they are not close to each other. Vacant rectangle size is almost the only consideration.

For the later situation, circuits are said to be placed with the most recently available minimum maximal rectangle (M3R) method. Both methods will be used in the following sections. When a circuit is placed with the M3R method, it will be put into the up-left corner of maximal rectangle.

5.4 Routing

Once a waiting circuit is assigned a vacant spot, connections associated with this circuit need to be built. This is the **OS4RC** routing problem. In this section, the Pathfinder algorithm and its core elementary algorithm, the Maze algorithm, are described. A complete procedure combining both algorithms is introduced later on as in Figure 5.10.

5.4.1 Maze Routing Algorithm

The maze router, developed by Lee [Lee61], is the basis for many existing FPGA routing algorithms, such as Tracer, Pathfinder, Srouter and VPR [Lee95] [Ebeling95] [Wilton97] [Betz99].

Using a breadth-first search strategy, the maze routing algorithm was designed to find the shortest path between two points on rectangular grids. Starting from the source node of a net, the maze algorithm tries to expand each neighboring node. All those neighboring nodes of each expanded node are then expanded further. After a series of wave propagation expanding operations, finally the sink node of the net is reached. In this wave propagation phase, more than one possible connection can be built. Among all these candidate connections, one is selected as the final connection during the back trace

phase. If the sink node cannot be reached during the wave propagation phase, then no possible route is claimed. To reduce the computation time, improved maze algorithms, such as Hadlock's algorithm and Soukup's algorithm, were suggested [Soukup78][Hadlock77].

A circuit generally has many multi-terminal nets, but the core idea of maze algorithms is to build a connection between two points. In practice, a multi-terminal net is decomposed into many two-point connections and every time only one two-terminal subnet is routed. Gradually such a complete net is routed. Another problem is the congestion, i.e., more than one net may compete for the same routing resource (such as a metal segment). Two strategies are frequently used to resolve this competition: rip-up and multi-iteration. In the first strategy, those competing networks are ripped-up and then re-routed. While in the second strategy, only one net is ripped up and re-routed, and this net does not have to be a congested net. In both cases, the final success of the routing may depend on the selection of the net to rip up or the order.

5.4.2 Pathfinder

The Pathfinder algorithm, developed by Ebeling et al. [Ebeling95], is a routing algorithm that tries to optimize the number of tracks and circuit delay simultaneously. It is used as a framework of the router in this chapter for three reasons:

(1) Effectiveness. It is adopted by many existing routers, such as VPR and other toolkits. Its effectiveness has been proved.

(2) Availability of the open source code. Many data structures and code organizations can be extracted from VPR [VPR00], which is available with source code on line.

(3) Scalability. As Pathfinder is a detail router, where more complex routing architectures are considered. Compared with those architectures, the mesh-cluster architecture is quite simple. This research begins with the case of one task one IO connection and expands to more general situations, such as one task multiple connections and multiple connections between tasks. See Chapter 6 for more task model details.

A pseudo code of the Pathfinder algorithm is explained later and is shown in Figure 5.11. During each iteration, each net is ripped-up and re-routed in the same order. Different nets are allowed to share common routing resources with each other. As iterations proceed, the sharing of routing resources is penalized with a cost function which increases gradually with each iteration. After a large number of iterations, congested routing resources are allocated to those nets which need those resources most, while nets that do not absolutely require those congested routing resources can be relocated to other places. This is called the process of negotiation, a distinguished feature of Pathfinder. When all congestions are resolved, the routing process is terminated.

It should be noted that ripping-up a net requires an assumption about IP cores. The OS needs to be able to stop them for reconfiguration of the segmented bus and then to resume their computations afterwards. In order for this assumption to work, IP cores need to be designed with this in mind. This assumption can be removed by using only

the maze algorithm for routing and using no ripping-up. In that case, the routing becomes much more difficult.

Before the algorithm of Pathfinder is examined, some terminologies are explained first:

$sink(i, j)$: The jth terminal on the multi-terminal network i.

$slack(i, j)$: For $sink(i, j)$, the amount of delay that can be added before it influences the circuit's critical path.

$D_{\max}$: Delay of the critical path.

$critical(i, j)$: It is defined as $1 - \frac{slack(i, j)}{D_{\max}}$

node n: As a portion of the connection of the jth sink of net i, it may correspond to a wire segment in a routing channel.

$b(n)$: base cost of node n.

$h(n)$: historical congestion cost of node n.

$p(n)$: present congestion cost of node n.

$cost(n)$: it can be expressed as :

$$cost(n) = critical(i, j) \cdot delay(n) + [1 - critical(i, j)] \cdot [b(n) + h(n)] \cdot p(n)$$

According to this expression, the Pathfinder algorithm is designed to make trade-off between the routability and timing delay.

$RT(i)$: The routing tree of net i. Net i is just a set of logical connections, while all physical routing resources that are used to build this net compose the corresponding routing tree.

PQ: priority queue

```
[1]   slack(i, j) = 1 for all net sources i and sinks j                    // initialize
[2]   For  try = 1; try <=max_try;  try++
        If ( try ==max_try){ // after max_try times effort, assume no feasible routing solution
                Routing_success = False;
                Break;}
[3]     For all net i
[4]       Rip up the routing tree RT(i)
[5]       RT(i)= source of net i                                           // start with source
[6]       For all sink(i, j)s in the decreasing order of critical(i, j)
[7]       PQ = RT(i) at the cost critical(i, j) · delay(n) for each node n in RT(i)
            //resource in RT(i) are used as starting points of the wave propagation
[8]           While sink(i, j) is not reached          // wave propagation
[9]               Get the lowest cost node m from PQ
[10]              Add all neighboring nodes n of node m to PQ with
[11]              cost = cost(n) + path cost from the source to m
[12]          End
[13]          For all nodes n in path t(i, j) to source      //back trace
[14]              Update cost(n)
[15]              Add n to RT(i)
[16]          End
[17]      End     // end of sink(i, j) for loop
[18]    End       // end of net for loop
        If ( Congestion_ flag = congestion _ check() ){
          Calculate the path delay and critical(i, j);
          }
        Else {
            Record this routing solution;
            Routing_success = True;
            Break;
          }
[19]  End     // end of try loop
```

Figure 5.10 Pseudo code of the Pathfinder

5.4.3 Routing Resource Graph

The routing architecture of a FPGA chip can be described as a routing graph, and an appropriate routing algorithm is then applied on this routing graph. Routing a net corresponds to finding a path on the routing graph. When every net find its own path with no resource conflict between any two nets, the routing process is done. In the routing graph, every vertex represents a routing resource. It may be an input or output pin on a CLB, or a track in the routing channel. An edge of the graph represents a possible connection between the two vertices. As on a FPGA chip, signal transmission on some resources (such as a tri-state buffer) is unidirectional, a directed routing graph is quite popular. All output pins (or input pins) of a CLB are equivalent in function, and they are exchangeable with each other.

The cluster-mesh architecture can be represented with a routing resource graph as shown in Figure 5.12, on which appropriate routing algorithms are applied. To make it clear, only a portion of routing resource graph is drawn.

When Pathfinder is used in VPR, all routing resources are represented with a routing resource graph. It is assumed that each CLB has a source/sink node on the routing resource graph, and each routing segment corresponds to one node on the graph. A routing resource graph for the proposed architecture is used by the router in the suggested OS kernel. Each cluster has one source/sink node, and all bus segments between two clusters correspond to only one node. That is another reason why routing is not time consuming with the proposed architecture and the routing algorithm.

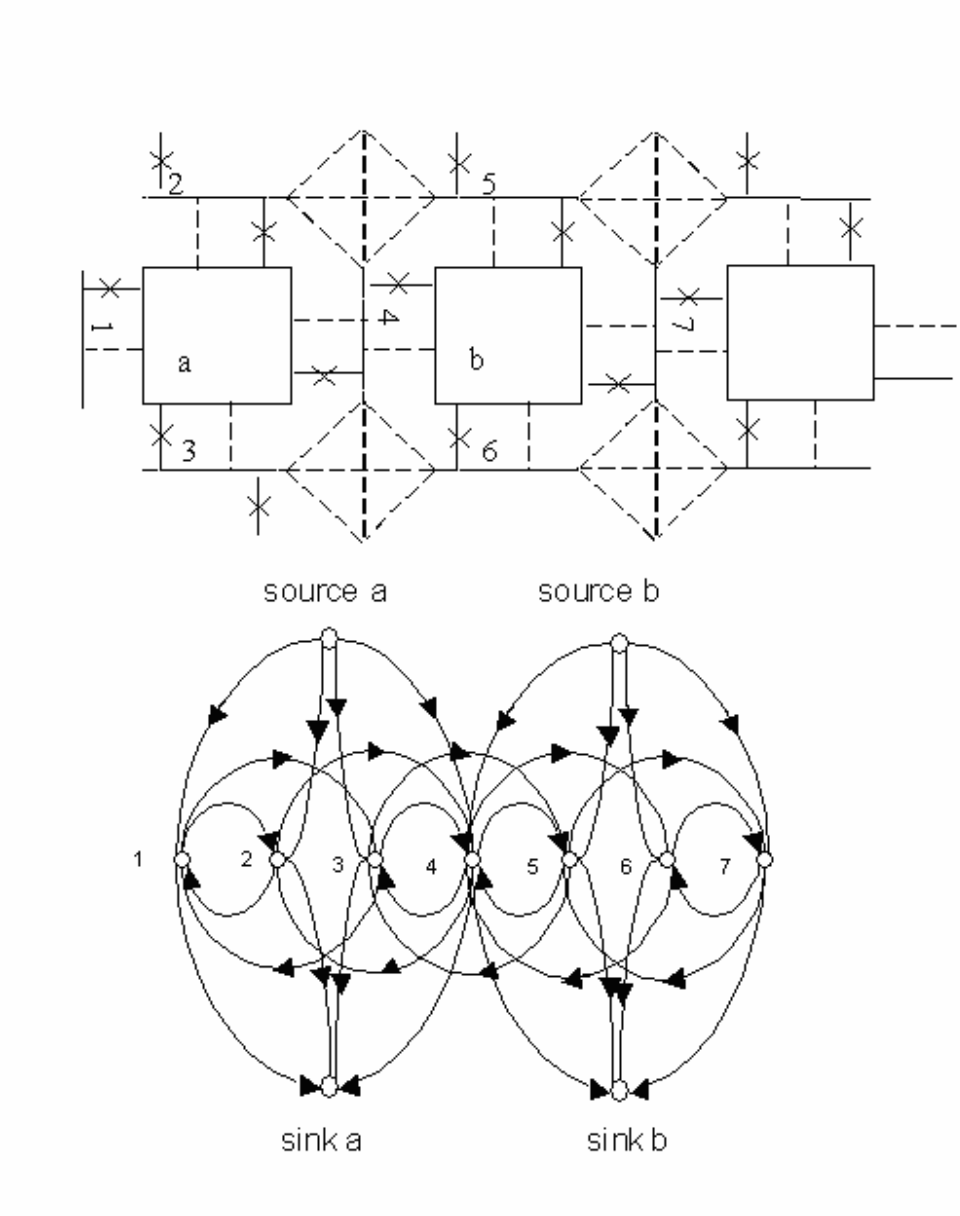


Figure 5.11 A Portion of Routing Resource Graph

5.5 Scheduler

If a circuit can be placed and connected appropriately, Scheduler will decide the order in which all submitted circuits run, so as to achieve optimal performance. In terms

of the performance metric, different criteria may be applied, such as the shortest running time of all submitted circuits, lowest ratio of rejected circuits and so on.

5.5.1 Assumptions

- Each time only one circuit arrives.
- Only limited connections exist between circuits, generally no more than five.
- Routing is only limited to inter-IP or IP-IO connections. Internal connections of IP circuits are pre-determined. Routing re-configuration or re-routing is assumed to be far faster than cluster configuration.

The above assumptions are considered practical at this point. They were made to reduce placement and routing time. No on-line circuit compaction is expected, as the time cost of compactions based on bit stream re-loading and context information storing is very expensive.

5.5.2 Framework of Scheduling Algorithm

There are three queues as follows:

Q^e: executing queue, keeps trace of all currently executing circuits.

Q^r: reservation queue, keeps trace of those currently not executing but scheduled circuits.

Q^w: waiting queue, keeps trace of those newly arrived but not yet scheduled circuits.

The execution time of a circuit is the time that this circuit runs on the chip [Steiger04]. In previous works, circuit execution times are assumed to be known in

advance. This type circuits are called K-type circuits [Bazargan00] [Steiger04] [Diessel98]. But in practice, there is another type of circuits, whose execution time is unknown in advance, but determined by external events. They are called U-type circuits. For example, a reconfigurable machine is sending a compressed file over an Ethernet. A network interface IP circuit is needed before any packet is sent out. Once the file is sent out completely, the encoder circuit and the network interface circuit can be swapped out of the FPGA chip. Because the traffic on the networks varies a lot, it is hard to know how much time it takes to send the complete file. The network interface module is such a U-type circuit. Similar situations can be found when a reconfigurable machine is receiving a file. In the later example, the network interface circuit may be swapped out of the chip while some K-type circuits are still processing the last packet received. K-type and U-type circuits are dealt with separately.

Every K-type circuit C_i has a width w_i and height h_i, and has an arrival time α_i, a starting time s_i once scheduled and an executing time e_i. It will finish at $f_i = s_i + e_i$. No deadline is used. For K-type circuits, as their resource release times are deterministic, the information can be used by the scheduler to reserve those resources for circuits arrived later.

The scheduler cannot consider resources released by U-type circuits in advance. But the operating system can be made aware of the termination of a U-type circuit (via a mechanism such as interrupt), and recycle its resources released. The operating system now has two options: (1) schedule a circuit in the waiting queue and (2) re-schedule those circuits in the reservation queue.

The trade-off of these two options is as follows. The scheduling effort of option (1) is minimal. However, it may miss some opportunities because some earlier circuits currently in the reservation queue may be rescheduled when a U-type circuit terminates.

So all possible events include:

(1) A circuit C_i arrives at α_i. Its arrival time α_i is used for scheduling by using policies, such as FCFS (first come first serve).

(2) A reserved circuit C_i in Q^r is dispatched when the system clock reaches s_i.

(3) An executing K-type circuit will terminate at $f_i = s_i + e_i$, when the system clock reaches f_i. The record of the executing circuit is erased, and corresponding resources used, mainly logic clusters and bus wire segments, are released for later use.

The starting time s_i can be equal to α_i if the circuit can be dispatched right away; or it can be equal to $f_k (k \neq i)$ if dispatching of C_i depends on the resources released by C_k at time f_k. Without any lose generality, we say that a circuit's dispatching depends on the most recently available resources in either case.

All s_i' s and f_i s (if known for U-type circuits at any moment) are sorted in an increasing order, and stored as check points in a queue Q^T in time domain. The operating system checks this queue frequently. When the system clock reaches a checkpoint, the operating system processes corresponding tasks appropriately. When such a scheduler is evaluated by simulation, this checkpoint queue can also be used to control the corresponding event-driven simulation [Law00].

Figures 5.13 and 5.14 show the pseudo codes of the scheduler and the spatial slot searching procedure used in the scheduler. Note that the last line of the code in Figure 5.13 involves the search for a routing solution which is detailed in Figure 5.15.

```
(The Scheduler is always trying to schedule the first circuit C_1^w in the waiting queue. It
can be triggered by the system clock or an interrupt mechanism)
While (any queue not empty)
{
    if ( any circuit in Q^e terminating at the current system time t ){
            release its resources, remove this circuit from Q^e,
            and update related data structures.}

    if ( any reserved circuit in Q^r to dispatch at the current system time t ){
        download the corresponding configuration bit stream, configure the
        corresponding inter-IP/IP-IO connections, and activate this circuit.
        It is removed from Q^r , and added to Q^e .
        Related data structures are updated.}

     //Try to locate the most recently available spatial slot for the first circuit in Q^w under
     //the condition of resource at current system time t . The worst case is that this
     // circuit will wait until all executing circuits exit.
    if ( any circuit in Q^W to schedule at current system time t ){
      f_k = current system time t ; // temporarily assume a virtual circuit C_k ∈ Q^e
                                    // terminating at the current system time t ;
        Search for the most recently available spatial slot ( f_k , C_k );
        Search for the most recently feasible routing solution;}
}
```

Figure 5.13 Pseudo Code of Scheduler

placement_success = False; //initialization

while (placement_success = =False){

try to locate an empty spatial slot at f_k;

if (such an empty spatial slot is found){

select the minimum empty spatial slot;

break;}

}

else{

$k = k + 1$, $f_k = f_{k+1}$

// with f_k as the next most recent finish time of a circuit in Q^e;

//defer all circuits in the waiting list until the next executing circuit exits

}

Figure 5.14 Procedure of Searching for a Most Recently Available Spatial Slot

```
route_success  = False; //initialization
while (route_success  = = False){
        route_success  = Routing( C_1^w , f_k );
            //routing for the waiting circuit under consideration
            // under the assumption of f_k .
        if  (route_success  = = true){
            put this circuit into Q^r
            update related data structures
            adjust Q^w head pointer to the next circuit if any.
            route_success  = True;
            break;
        }
        else {
            k = k + 1, and  f_k = f_{k+1}
            // with  f_k  as the next most recent finish time of a circuit
            // in Q^e ;
            //defer all circuits in the waiting list until the next executing
            //circuit exits
        }
    }
```

Figure 5.15 Search for the Most Recently Feasible Routing Solution

After the simulation is terminated, the total time to finish all circuits can be calculated, the average waiting time can be extracted, and the efficiency of the placement and routing strategy can then be evaluated.

6 Simulation Results

In this chapter, simulations are performed to evaluate the performance of the OS4RC implemented and the impacts of varying architecture parameters on runtime circuits. Simulation results based on three different task models are discussed. Each model has different parameter settings. These parameters cover different chip sizes, different circuit arrival interval times, and cases with/without bus support options. Operating system performance for each parameter setting is evaluated via simulations. Performance metrics include the average waiting time (or response time) of a new arrival circuit, the average execution time for each circuit, the average length of the reservation queue and so on.

6.1 Simulation Framework

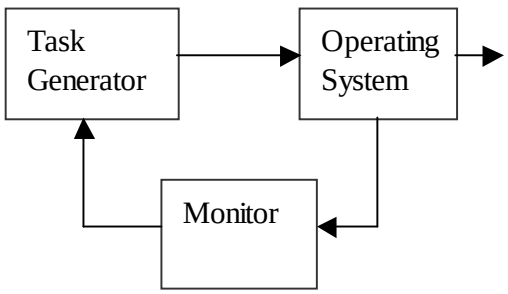


Figure 6.1 Simulator Internal

The simulator includes three parts: the task generator, the operating system kernel and the monitor, as shown in Figure 6.1. The monitor controls the start/stop of the simulation, keeps track of the running of the operating system, evaluates the performance of the operating system, and supplies information to the task generator. The task generator produces randomly generated circuits to feed into the operating system. It has built-in random data generators for different parameters of each circuit. It also accepts information from the monitor to make necessary adjustments.

The circuit arrival interval time is assumed to follow an exponential distribution, i.e., the number of circuits arrived during certain time follows a Poisson distribution. The execution time of each circuit is also assumed to follow an exponential distribution. Circuit sizes are assumed to follow a uniform distribution.

As the operating system kernel is based on queues, the average arrival interval time has impacts on the operating system performance, especially the waiting time. Different situations may occur:

(1) When the arrival interval is long enough, for example, many times longer than the average execution time, the average waiting time is close to zero. That is, new arrivals do not have to wait. This situation is called *lightly-loaded*. Obviously, working in this situation, the chip utilization is low, and there may be very few tasks running in parallel.

(2) When the arrival interval time is very short, the average waiting time increases dramatically. When a circuit arrives, too many circuits are backlogged ahead of it. As time goes, more and more circuits have accumulated at the waiting queue or the reservation queue. Such a system is said *unstable*, and the operating system is *heavily-loaded*. Users will not be patient to work with a heavily loaded machine, even though the

machine is highly utilized. The processing capability of the machine under this situation can be thought as the maximum capacity.

(3) When the arrival interval time is chosen appropriately, each new arrival circuit does not wait for too long, and the chip utilization is not too low. This is called *normally-loaded*. Our research is focused on this situation.

The critical point of the arrival interval is the interval time that separates Case (2) and Case (3). When the average arrival interval is smaller than this value, the average waiting time increases dramatically, and the system becomes unstable. A bigger chip has a smaller critical arrival interval compared with that of a smaller chip.

Each measurement summarized in this chapter is an averaged result from 25 Monte-Carlo simulations and each Mote-Carlo simulation uses 10000 randomly generated circuits. In other words, each measurement result is based on the results of running 250,000 circuits through the OS.

6.2 Different Task Models

In previous works, each task is assumed to be carried out by one circuit, and the execution time of each circuit is known [Bazargan00][Diessel98][Steiger04] [Tatineni02]. In this section three different task models in increasing complexity are used in the simulations.

6.2.1 Mode 1: Single Net – Single Circuit Model

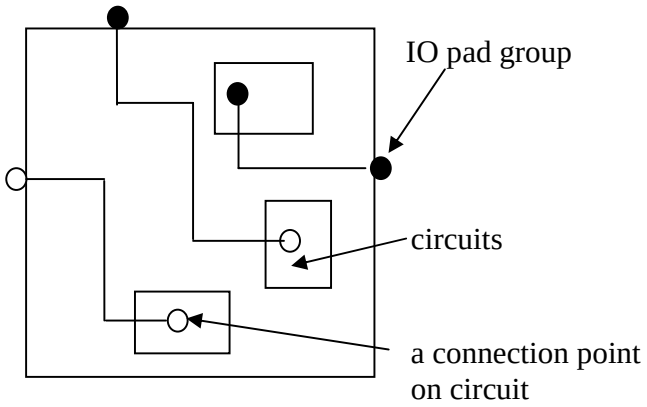


Figure 6.2 Single Net – Single Circuit

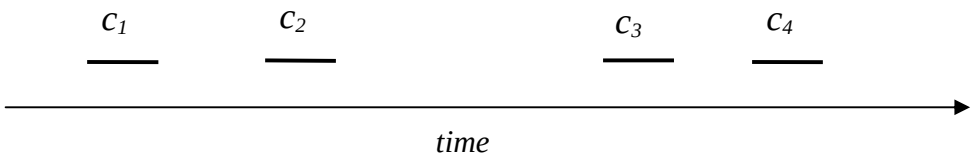


Figure 6.3 Arrival of Circuits for Model 1

In Model 1, each task T_i needs only one K-type circuit c_i . Recall that the execution time of a K-type circuit is known at its arrival time. Each circuit needs a connection with an IO pad group on the boundary. In Figure 6.2, three such tasks are depicted. Only IP-IO communications are considered, without any inter-IP communications. Each connection itself composes an independent net, and only one circuit is hooked on each net. Figure 6.3 shows the arrival times of four circuits.

When segmented bus is available, finding a vacant spot is almost the only determining factor for the scheduling of a circuit. Given this point, this model is quite close to models used in many previous works, even though communication was not considered in those works. Performance claimed in those previous works is overly optimistic as the impact of communication requirements on scheduling, or at least the time cost to build communications, is not considered.

A circuit may contain more than one cluster. One of its composing clusters is selected as a connection point on the circuit. Correspondingly one IO pad group on the boundary is randomly selected (uniform distribution assumed) as another connection point. When this circuit is loaded, a physical path is built between the two connection points, depending on available resources.

In later parts of this chapter, the OS4RC performance corresponding to both with/without bus cases is evaluated. Different placement algorithms are applied to the two situations, and advantages of segmented-bus will be quantified.

6.2.2 Model 2: Single Net–Multi Circuit Model

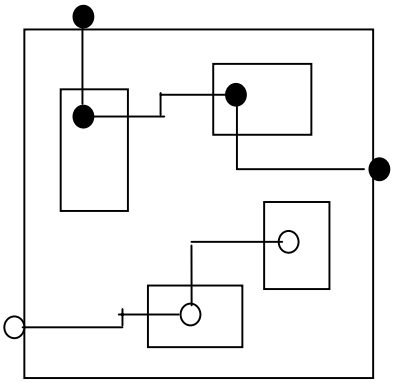


Figure 6.4 Single Net-Multi Circuit Model

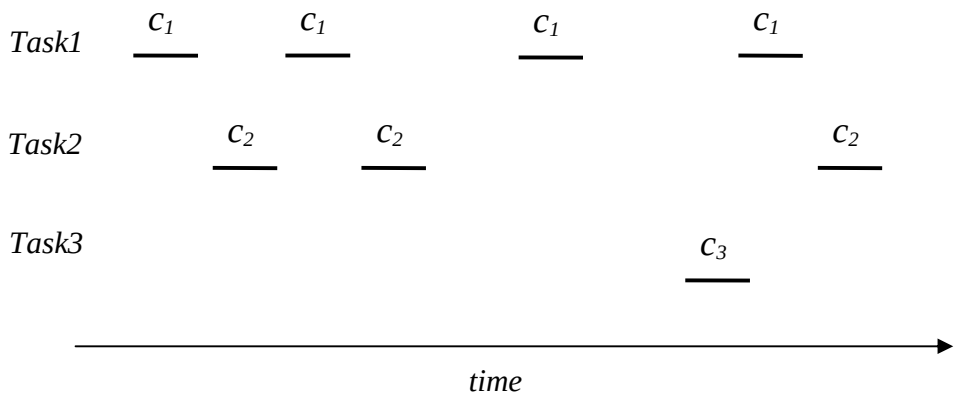


Figure 6.5 Arrival of Model 2 Tasks

Generally a task is composed of several components, and each component is implemented by an independent IP circuit. To maximize chip utilization, multiple tasks are processed at the same time.

In Model 2, each task is carried out by a U-type circuit plus several K-type circuits, but each circuit just serves one task. For a specific task, there is a net that connects all its circuits together, and each circuit is only associated with one net. At a specific time, the number of nodes on the net and the physical path this net takes are quite random, depending on available resources. In Figure 6.4, two separated nets are represented. They have black and white nodes respectively. There may be more than one task $T_i, T_j, \ldots$ running in parallel.

Both inter-IP and IP-IO communications are allowed. Each circuit has one and only one random cluster to hook on its net. Whether a circuit has an IO connection follows a uniform distribution. If any circuit has an IO connection, it can only have one.

If more than one circuit has IO connections, then all associated IO pad groups will be hooked on the same net.

Arrivals of circuits interleave with each other randomly, as shown in Figure 6.5. Suppose that Task T_i needs circuits c_{i0}, c_{i1} … $c_{ik(i)}$, and they are ordered in their arrival time. The total number of circuits required by T_i is random. Among them, the first circuit c_{i0} is U-type, and all other circuits are K-type. Circuit c_{i0} has to be the first one to be scheduled. Among all those K-type circuits associated with T_i, $c_{ik(i)}$ is the one that arrives the latest, and $c_{i\_last\_exit}$ is the one that exits the last. The circuit that exits the last does not have to be the same as the circuit that arrives the latest. The circuit $c_{i\_last\_exit}$ cannot even be decided before $c_{ik(i)}$ is scheduled.

Corresponding to c_{i1}, c_{i2} … $c_{ik(i)}$, their starting times are s_{i1}, s_{i2} … $s_{ik(i)}$, respectively, and their execution times are e_{i1}, e_{i2} … $e_{ik(i)}$, respectively. The exiting time of $c_{i\_last\_exit}$ can be decided as:

$$f_{i\_last\_exit} = \max\{s_{ij} + e_{ij}\}, \quad j = 1,2...,k(i). \qquad \text{(Eq. 6.1)}$$

Before $f_{i\_last\_exit}$ is decided, the execution time of c_{i0} is assumed to be infinity. Circuit c_{i0} will exit from the chip immediately after $c_{i\_last\_exit}$. This assumption is used to simplify the simulation coding. The scheduling of a K-type circuit depends on a U-type circuit, and finding a vacant spot is no longer the only consideration in this model. This is another difference from Model 1.

6.2.3 Model 3: Multi Net-Multi Circuit Model

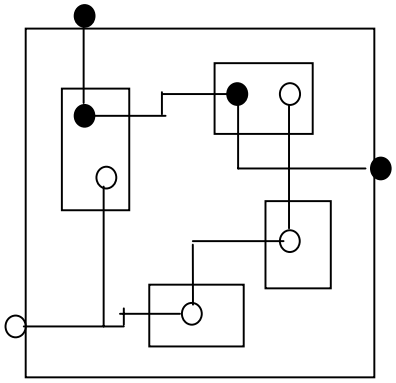


Figure 6.6 Multi Net -Multi Circuit Model

For complex computation, there exist connections between tasks, and an IP circuit may be utilized by more than one task. For example, an UART interface may be shared by different tasks at different time. Therefore this UART IP core can be attached to more than one task. As another example, two tasks may exchange information via a dual port memory.

Model 3 is a further extension of Model 2. Each task still needs one U-type circuit plus some K-type circuits. In Model 3, a U-type circuit can only be associated with one task (net), but a K-type circuit can be associated with more than one task (net) at a time. How many tasks (nets) a K-type circuit is associated with is random. Figure 6.6 shows a typical example. Two separated nets are represented, and they have black and white nodes respectively. For a specific circuit, different nets connect to points at different clusters, and each net connects at only one cluster. For a small circuit (say, less

than five clusters), no more than half of its composing clusters are allowed to be connected to nets.

For each task more than one circuit may have IO connections. Whether a circuit has an IO connection has a fixed probability, following a uniform distribution. A circuit cannot have more than one IO connection. If the circuit that has an IO connection is K-type, its IO connection is randomly assigned to one of associated nets; if it is a U-type, this IO connection is assigned to the same net as this U-type circuit is associated with.

Because of the difference between Model 3 and Model 2, some modifications are made to the task generator when Model 3 is simulated. When a K-type circuit is created, all those U-type circuits, either already on the chip or in the reservation queue, are checked. Tasks that still need K-type circuits are identified, and this newly created K-type circuit is randomly associated with some of them.

6.3 Deadlock Issues

Previous works did not consider the deadlock problem [Bazargan00] [Steiger04] [Wigley05]. In this section, the deadlock phenomena are analyzed, and the corresponding detection and resolution methods are proposed.

6.3.1 Deadlock

In Model 2 and Model 3, because of those U-type circuits, deadlock may happen. Consider the following Model 2 case depicted in Figure 6.7:

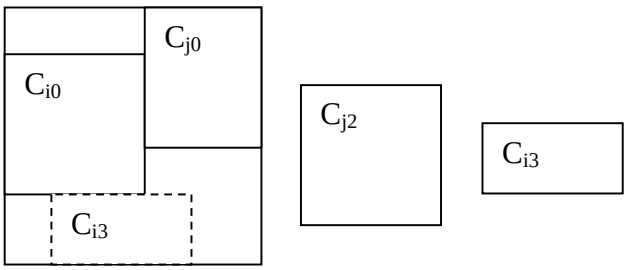


Figure 6.7 An Example of Deadlock

Task T_i and T_j are running in parallel. Task T_i needs four IP circuits: c_{i0}, c_{i1}, c_{i2} c_{i3}; and T_j need three IP circuits: c_{j0}, c_{j1}, c_{j2}. Suppose that c_{i0} and c_{j0} have co-existed on the chip already. Then c_{j2} arrives, followed by c_{i3}. Assume c_{j2} is too big to fit into any vacant slot. Although it is possible for c_{i3} to fit into the chip (as the space framed with dash lines), because of the first-come-first-serve scheduling policy, c_{i3} cannot be scheduled until c_{j2} is scheduled. Before c_{i3} and c_{j2} are scheduled, c_{i0} and c_{j0} cannot exit from the chip as their exit times are at infinity. So the processes corresponding to T_i and T_j cannot finish, and all those circuits in the reservation queue are blocked. Deadlock happens.

6.3.2 Deadlock Detection and Resolution

If a circuit c_{ij} cannot be loaded into the FPGA chip right away, it has to wait until some circuits exit from the chip. The circuits that need to exit may be on the chip or still in the reservation queue. If a circuit c_{ij} cannot be scheduled even when all K-type circuits

are assumed to have exited the system, the load-in time of c_{ij} depends on a U-type circuit whose exit time is infinity. Then a deadlock happens.

In previous operating system prototypes, if circuits submitted to the machine cannot be processed before a deadline, they are simply rejected [Bazargan00] [Steiger04]. Even though simple rejection is not a perfect approach, such idea is followed in the proposed OS kernel to resolve deadlocks in a partial reconfiguration based environment.

If a deadlock is detected when c_{ij} is scheduled, and c_{ij}'s start time depends on c_{k0}, two possible situations may occur. Corresponding to the two situations, different strategies are conducted:

(1) Circuit c_{ij} is not associated with task T_k (or c_{k0}), i.e., $i \neq k$. If this happens, let c_{k0} exit after all its scheduled K-type circuits have exited. Task T_k is therefore only partially finished without completing some unscheduled K-type circuits, while T_i still has a chance to finish.

(2) Circuit c_{ij} is associated with T_k (or c_{k0}), i.e., $i = k$. If this happens, c_{ij} is rejected as an impractical circuit for the simulation and a circuit of different resource requirement is generated at a later time to replace c_{ij}.

In Case (1), T_k is preempted. When c_{k0} was created, T_k was originally expected to have a certain number, say *m*, circuits. Because of preemption, not all of them were simulated, and *m*-*n* circuits are cut off (or rejected). In Case (2), the execution time of c_{k0} is therefore delayed further.

6.4 Guidelines to Task Generator

For Model 1, the design of the random circuit generator is simple. According to certain probability distribution function on various parameters, the task generator automatically produces only K-type circuits, one for each task. The monitor starts the task generator at the beginning of each simulation, and stops it when 10000 circuits have been created.

For Model 2 and Model 3, some specific strategies are applied to the task generator to control simulations:

(a) In each Mote-Carlo simulation, before the total number of circuits simulated exceeds 10000, whether a circuit to be generated is of U-type or K-type is based on a probability distribution. If a U-type circuit is created, then a new task is created, and a random number is assigned to indicating the number of K-type circuits to be associated with this U-type circuit. The execution time of this U-type circuit is set as infinity.

(b) When a K-type circuit is created, all those U-type circuits that have been scheduled are checked to see if they have enough K-type circuits already. If they all have enough already, this newly created K-type is converted into a U-type circuit, indicating a new task is initiated. If ψ out of those U-type circuits still do not have enough K-type circuits, the newly created K-type circuit is randomly associated with ε out of those ψ tasks. For Model 2, ε is always equal

to one. For Model 3, ε is a random integer in the range of 1~ψ. A K-type circuit is assigned with a random execution time.

(c) If a task is preempted because of deadlock resolution, no new circuit is created for that task any more.

(d) Corresponding to the second case of deadlock resolution, if a circuit c_{ij} is rejected, the number of circuits required by its task T_i is not affected. The task generator will produce the remaining random circuits at some future time. The c_{ij} produced the next time most probably will not have exactly the same parameters as those when c_{ij} is rejected.

(e) When the total number of circuits reaches 10000, no new U-type circuit is created afterwards, and only K-type circuits are created for those tasks that are still running. Generally, a bit more than 10000 circuits are simulated.

6.5 Different Placement Methods Applied to Different Models

For Model 1 and Model 2, different placement algorithms are applied, depending on whether the segmented bus is used.

For Model 1, only IP-IO connections are allowed. When there is no segmented bus, there is no dedicated communication infrastructure, and circuits are placed with the AAA method only (see Section 5.3). When the segmented bus is available, circuits are placed with the M3R method.

For Model 2, things are a little bit more complicated. Connection point locations, either on circuits or chip boundaries, are quite random. When the segmented bus is not available, it is hard to guarantee a K-type circuit can abut and align with both the chip boundary and its associated U-type circuit at the same time. Therefore it is hard for Model 2 to allow more than one circuit to have IO connections. In simulations, only U-type circuits are allowed to have IO communications, as they stay in the chip for the longest time compared with their associated K -type circuits.

For similar reasons, when the segmented bus is not available, inter-IP connections are only limited to K -type circuits and their associated U-type circuit. There is not direct communication between K-type circuits. If there is any communication between K-type circuits, the dynamic coming and going of circuits may cause some circuits to be isolated from other circuits associated with the same task.

For Model 2 while without segmented-bus support, placement strategies can be summarized as the following: For a U-type circuit with an IO connection, the AAA placement strategy is used; otherwise it is placed with the M3R method, which can allow it to be placed as soon as possible. All K-type circuits are placed with the AAA method only.

For Model 2 while with segmented bus support, all circuits are placed with M3R method. More than one task can have IO connections at the same time. It is not necessary to worry about whether a circuit is U-type or K-type.

If a chip has no dedicated infrastructure for inter-IP or IP-IO communications, it has no way to support the flexibility of Model 3. That is why Model 3 is simulated only with the segmented bus.

6.6 Simulation Results

Model	cases	Array size (clusters)	U-type rate	Bus
1	Case 1	20*20=400	0%	Yes
	Case 2	16*16=256	0%	Yes
	Case 3	20*20=400	0%	No
	Case 4	22*22=484	0%	No
2	Case 1	20*20=400	30%	Yes
	Case 2	16*16=256	30%	Yes
	Case 3	22*22=484	30%	No
3	Case 1	20*20=400	30%	Yes
	Case 2	16*16=256	30%	Yes

Table 6.1 Different Simulation Cases

Chips with different numbers of clusters under different situations are listed in Table 6.1. In the following simulations, the average execution time of K-type circuits is assumed to be 200 time units. The area of each circuit falls in the range of 0.04*16*16 ~ 0.08*16*16 clusters. For Model 2 and Model 3, each task needs one U-type circuit plus 1~5 K-type circuits, and 20% of circuits have IO connections. For Model 3, a K-type circuit is associated with 1~5 tasks/nets.

6.6.1 Average Waiting Time

The simulation results of average waiting times for different models are summarized in Figures 6.8, 6.9 and 6.10. Irrespective of whether there are segmented buses, the average waiting times decrease when the chip area increases. This is because a bigger chip can accommodate more circuits at a time, which results in a quicker circuit dispatch. In Figures 6.8 and 6.9, compared with chips with segmented buses, chips without segmented bus have bigger area but get longer waiting times. That demonstrates

one benefit of the segmented bus. With the segmented bus, increased chip area can greatly reduce the average waiting time when the arrival interval time is small. Therefore the chip processing capacity is obviously enhanced.

In Model 2 and Model 3, the placement of a K-type circuit depends on a U-type circuit. As a result, the average waiting time is longer than that of Model 1. In Model 3, because a K-type circuit may be associated with more than one task at the same time, those U-type circuits stay shorter time compared with the Model 2 situation. Therefore Model 3 has shorter average waiting times.

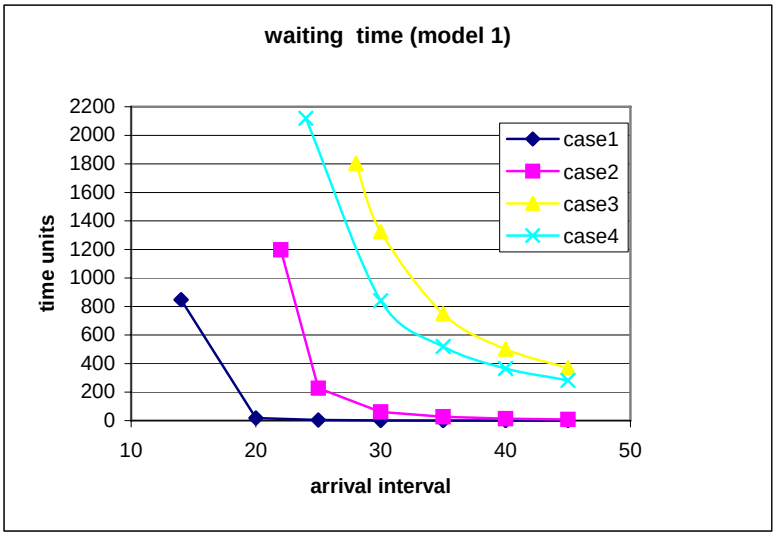


Figure 6.8 Average Waiting Time for Model 1

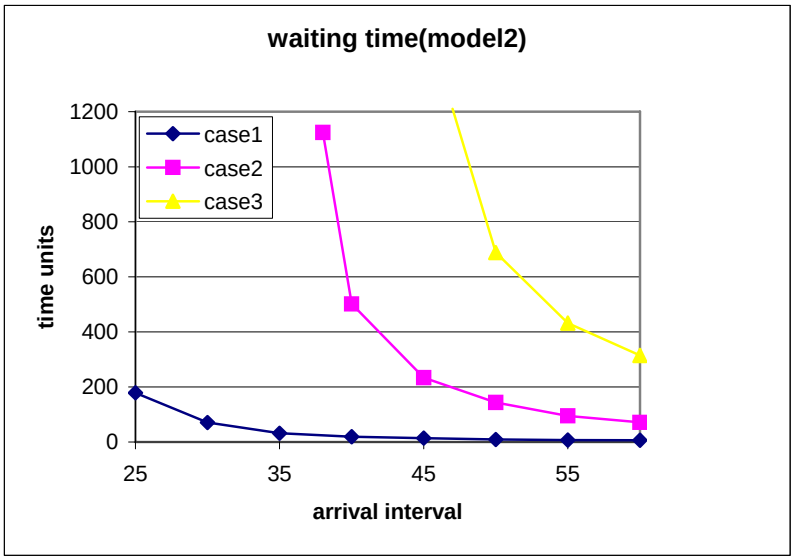


Figure 6.9 Average Waiting Time for Model 2

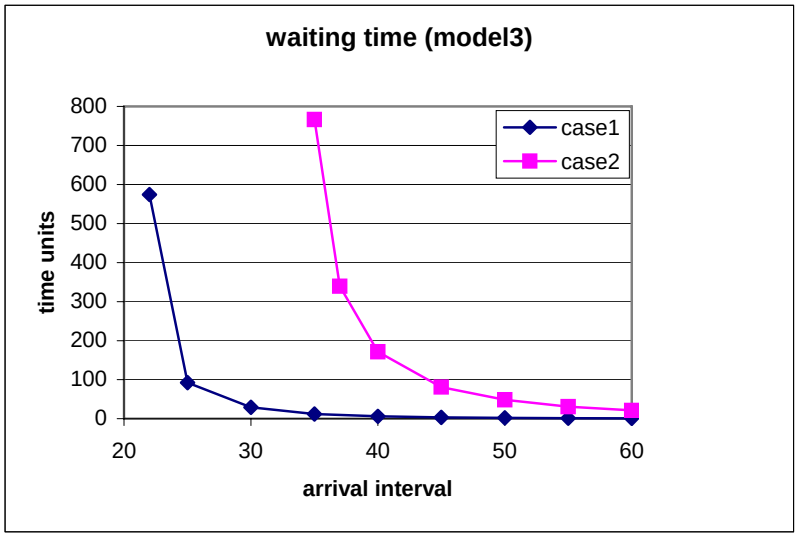


Figure 6.10 Average Wait Time for Model 3

6.6.2 Average Placement and Routing Time

The average placement and routing times for these three models during simulations are summarized in Figure 6.11, Figure 6.12 and Figure 6.12. Routing time cost is notoriously expensive in traditional FPGA design flow, frequently taking tens of minutes. With the separation of inter-IP connections and intra-IP connections using segmented buses, the routing is expected to be much faster, as required for on-line routing. From Model 1 to Model 3, as routing complexity increases, the average placement and routing times also increase. In most situations, for each circuit, the average placement and routing time is less than 10 ms. For every situation in those three figures, the average placement and routing time is less than 20 ms.

The placement and routing time usually increases with the chip area increase, because routing on a bigger chip means higher routing complexity. On each curve of those figures, when the arrival interval time decreases and passes a certain point, the placement and routing time increases dramatically. This point is called the critical point. When the arrival interval time is bigger than the critical point value, it is found that more than 90% of the P&R time is due to the routing (the Monitor records the placement and routing time costs separately). When the arrival interval is smaller than the critical point value, it is found that the P&R time increase almost totally comes from the placement. The placement component begins to dominate the P&R time. This is due to the characteristic of the placer, which exhausts all efforts trying to place a circuit.

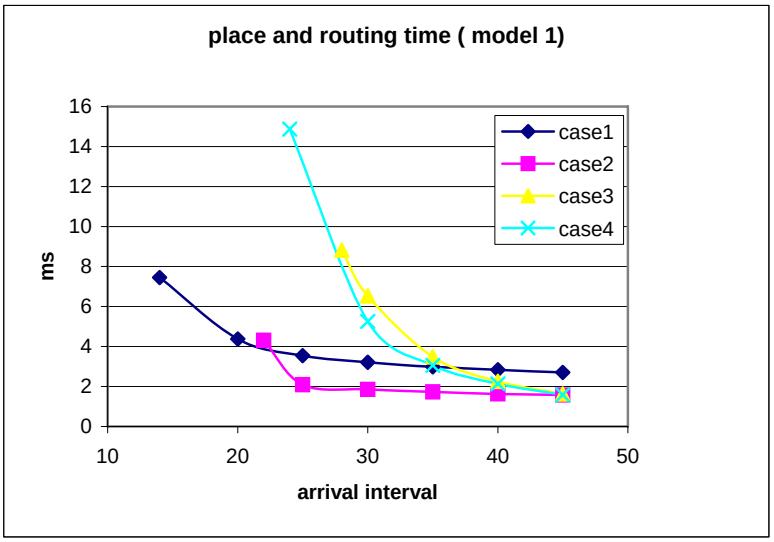


Figure 6.11 Average Placement and Routing Time for Model 1

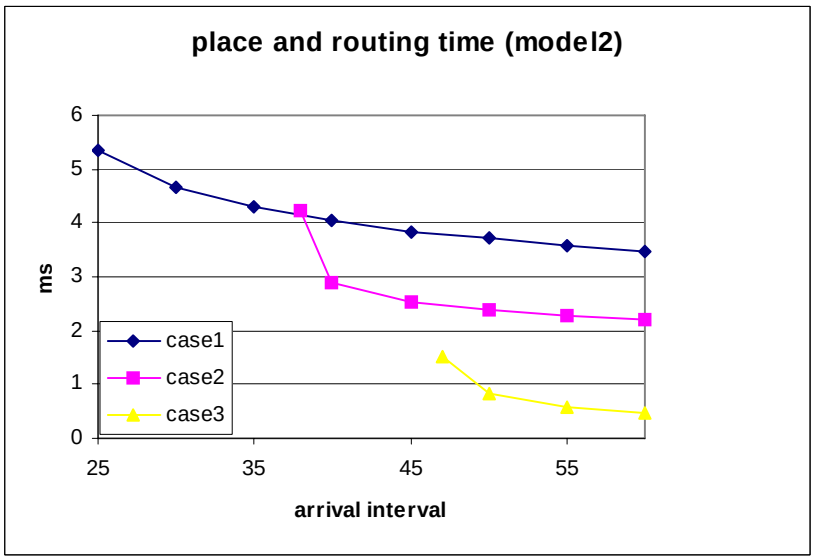


Figure 6.12 Average Placement and Routing Time for Model 2

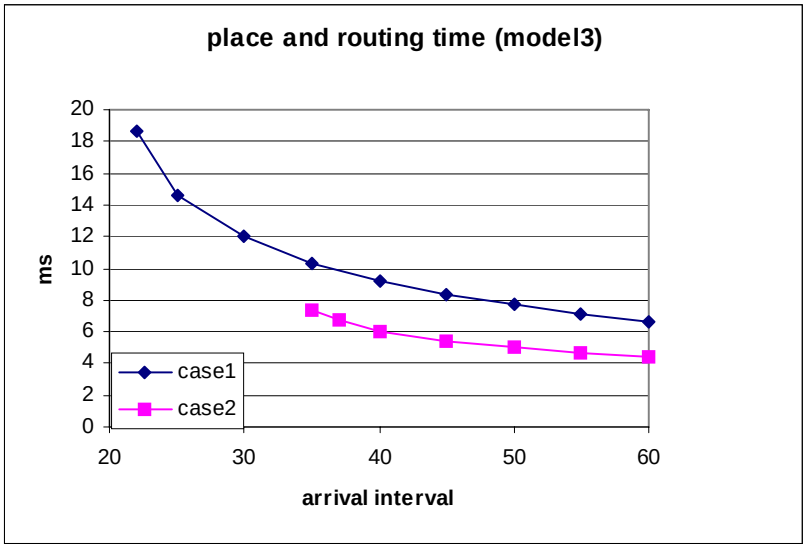


Figure 6.13 Average Placement and Routing Time for Model 3

6.6.3 Reservation Queue

Average lengths of reservation queues corresponding to different models are depicted in Figure 6.14, 6.15 and 6.16. When the arrival interval is small, many pre-placed and pre-routed circuits are in the reservation queue. The curves of these figures are similar to those in Section 6.6.1. This makes sense, as the more circuits stay in the reservation queue, the longer a new circuit has to wait. One problem for too many scheduled circuits staying in the reservation queue is its consumption of memory storage. When the memory capacity is limited, this may cause problems.

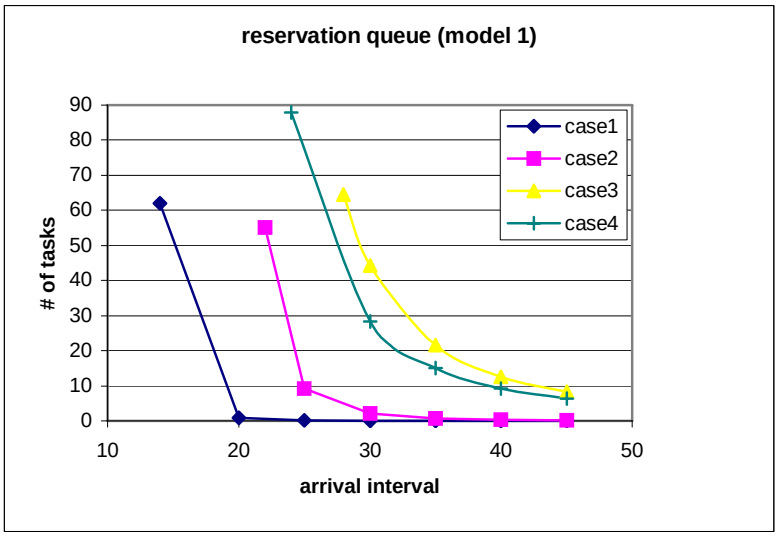


Figure 6.14 Reservation Queue Lengths for Model 1

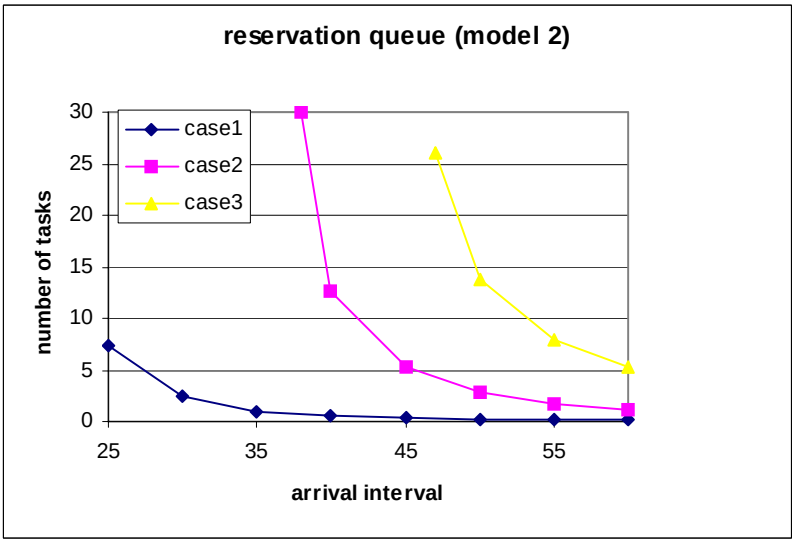


Figure 6.15 Reservation Queue Lengths for Model 2

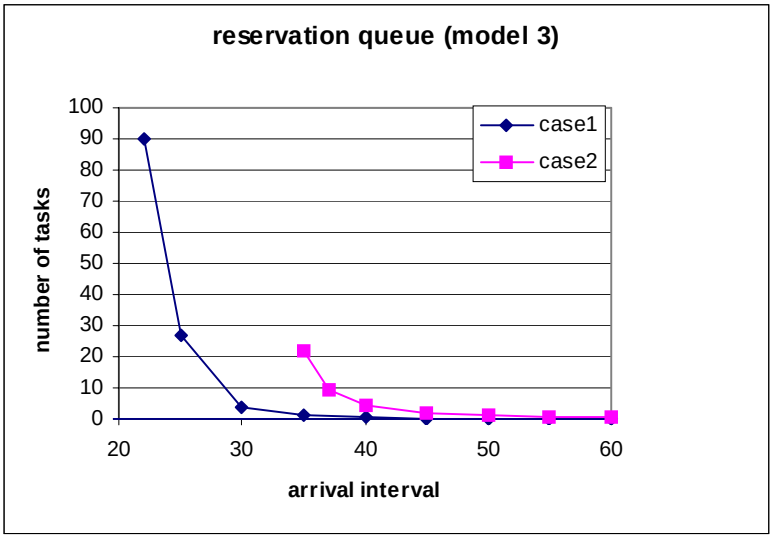


Figure 6.16 Reservation Queue Lengths for Model 3

6.6.4 Average Execution Time

For Model 1, execution times for different cases are fixed, i.e. 200 time units. For Model 2 and Model 3, their execution times are depicted in Figure 6.17 and Figure 6.18, respectively. Because of those U-type circuits, average execution times are all longer than 200 time units. In most situations, the larger the average arrival interval time, the longer the average execution time. In Model 3, each K-type circuit contributes to different tasks at the same time and U-type circuits stay shorter amount of time on the chip. So execution times are shorter than cases in Model 2.

In Figure 6.17, the average execution time for Case 3, i.e. the case without bus, is shorter than with-bus cases. This comes from our simulation methodology, and has nothing to do with the efficiency of the without-bus case. When a K-type circuit is produced by the task generator, it is to be assigned to an available net, which is associated with a U-type circuit. When segmented buses are not available, less number of tasks/nets

is running in parallel because of those strict placement constraints as depicted in Section 6.5. Hence possibilities to assign a newly arrived K-type circuit are very limited. In other words, all circuits associated with the same net arrive in a short time period. Therefore the associated U-type circuit does not stay on the chip for long. On the other hand, when bus is available, possibilities to assign a newly arrived K-type circuit are much higher because of higher parallelism due to relaxed placement constraints. As a result, the average execution time for Case 3 in Figure 6.17 is shorter than those for Cases 1 and 2.

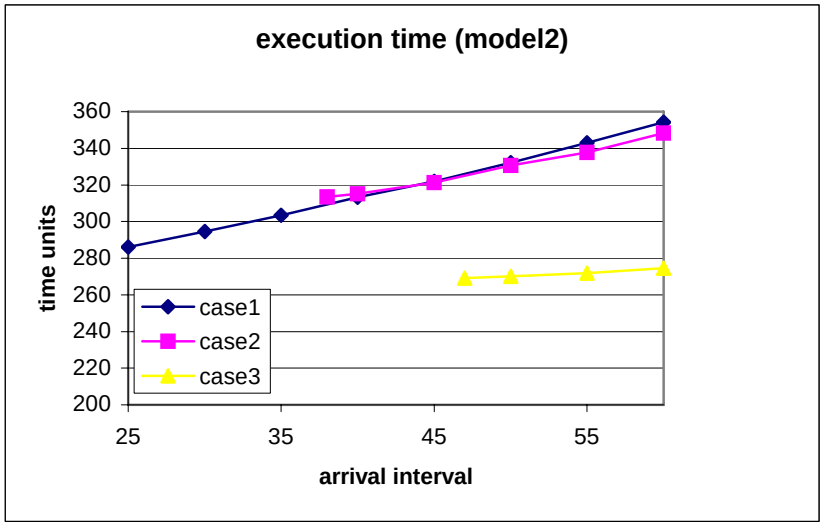


Figure 6.17 Average Execution Time for Model 2

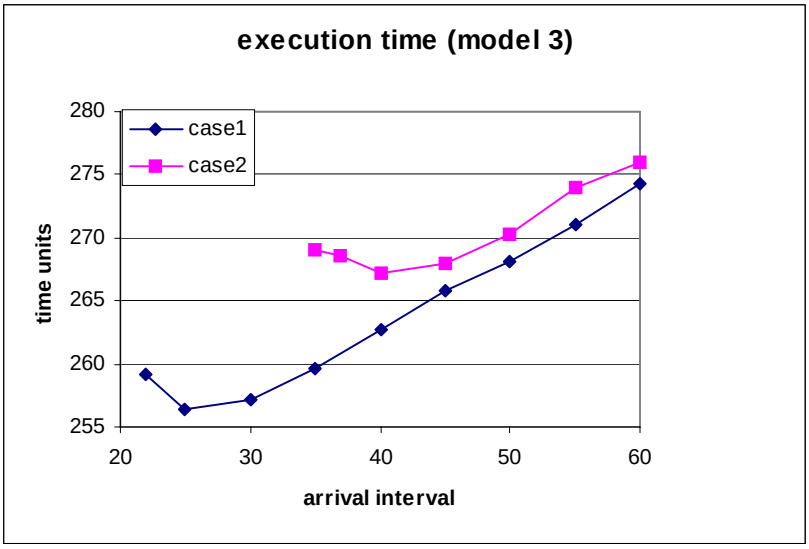


Figure 6.18 Average Execution Time for Model 3

6.6.5 Deadlock Resolution

Deadlocks do happen. Figures 6.19 and 6.20 record the ratio of circuits that are rejected or cut off due to deadlock. Even with segmented-bus support, deadlocks happen more often on small chips. The reason is quite straightforward. Deadlock happens when the scheduling of a circuit depends on a U-type circuit whose exit time is still undefined. Higher throughput can reduce the chance. Deadlocks happen more often with Mode 2 when compared with Model 3, because of its lower throughput. On each curve of Figures 6.19 and 6.20, no clear relationship between arrival time interval and rejection/cut-off ratio was observed. Perhaps deadlocks are small probability events, and longer simulation series are needed.

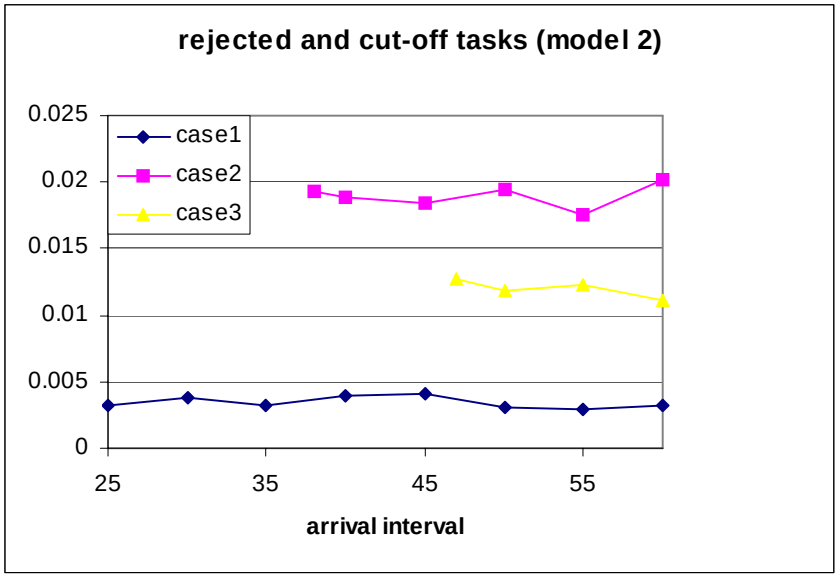


Figure 6.19 Rejected and Cut-off Circuits Ratio, Model 2

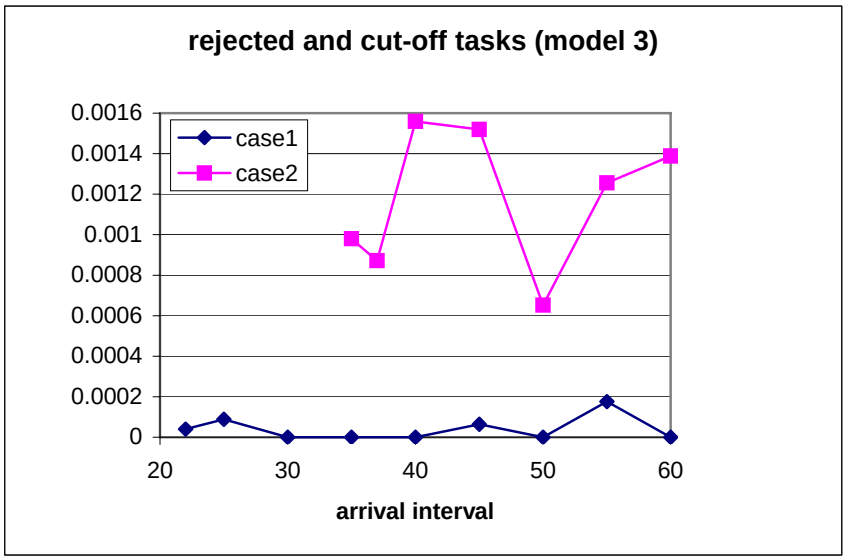


Figure 6.20 Rejected and Cut-off Circuits Ratio, Model 3

6.6.6 Best Cluster Size

One important parameter of the proposed architecture is the cluster size. If the cluster is too small, the segmented bus part is going to consume too much chip area. That leads to smaller area to accommodate circuits and longer average waiting time. If the cluster is too big, there are fewer bus segments which lead to limited routing flexibility. Other factors that may influence the selection of a cluster size include the total chip area, the statistical distribution of circuit size and so on.

Simulations can be used to determine an optimal cluster size, which apparently depends on the specific cost function used. Here one possible cost function is given based on waiting time, area penalty and timing delay. It is as follows.

$$\cos t = (S_{chip} + S_{dif}) * T_{avg\_wait} * m \qquad \text{(Eq. 6.2)}$$

where S_{chip} : Chip area scaled in CLBs. It includes both the cluster part and the segmented bus part.

S_{dif} : Difference chip area between S_{chip} and the original 120*120 CLBs.

T_{avg_wait} : Average waiting time, in time units.

m : The number of clusters on the chip along one dimension.

In the above cost function, the $(S_{chip} + S_{dif})$ part represents a penalty if S_{dif} is positive, or a reward if negative. The parameter m is introduced to characterize the propagation delay form one corner of the square chip along the diagonal direction to another corner. That path is $2 \cdot m$ long scaled in Manhattan distance. Along this path, $4 \cdot m$ tri-state buffers are inserted.

Different T_{avg_wait} corresponding to different partitions of a chip can be obtained from simulations. Using Eq. 6.2, costs corresponding to different cluster sizes are illustrated in Figure 6.21 and 6.22. Figure 6.21 corresponds to situations where all crossbars are fully populated, and Figure 6.22 corresponds to situations where all crossbars are half populated. At the minimum point of cost, a cluster with "optimum" size is thought located.

From Figure 6.21 and 6.22, almost all minimum occur when the cluster size, N, is four or eight. The only exception is the point ($N = 7, k = 4$) on Figure 6.22 where cost is very close to the cost at ($N = 8, k = 4$).

In Section 3.4, to evaluate the bus overhead, a chip with 120*120 CLBs is partitioned into clusters with different sizes. The operating system is evaluated for those different partitions. A medium arrival interval, 35 time units, is chosen. In those simulations, circuits are in the range of 0.04*16*16~0.08*16*16 clusters. No internal fragmentation is considered. For simulation summarized in this section, circuits are in the range of 0.02*120*120~0.05*120*120 CLBs. If a circuit has a width w and a height h (both scaled in CLBs), it needs $\left\lceil \frac{w}{N} \right\rceil \times \left\lceil \frac{h}{N} \right\rceil$ clusters because of internal fragmentation. Since internal fragmentation is considered, smaller circuit size ratios (0.02~0.05) are used for simulation in this section.

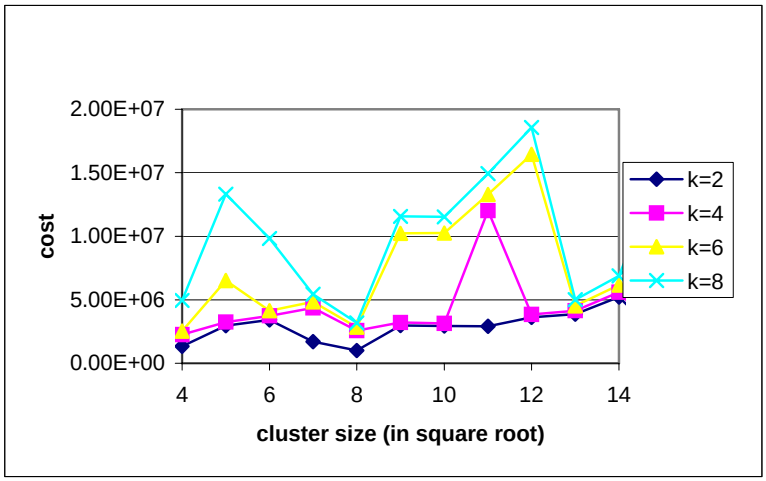


Figure 6.21 Cost Function (P =1.0)

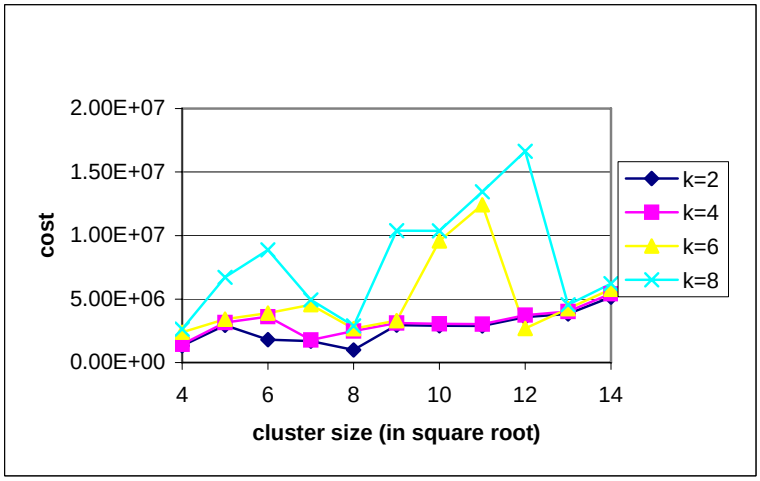


Figure 6.22 Cost Function (P =0.5)

6.7 Conclusions

(1) Model 3 is more efficient compared with Model 2. Model 3 has more flexibility, and is closer to practical situations. Model 1 is too constrained. It is used to show inflated performance when effects of IO connections are not considered.

(2) Increase chip area without segmented buses is not very helpful. The proposed segmented bus is very effective in reducing the circuit waiting time. The increase of chip area can show more gain when the arrival interval is smaller.

(3) The placement and routing cost has been evaluated. This is different from those operating system prototypes where only placement is considered [Bazargan00] [Steiger04] [Diessel98]. With the proposed FPGA chip architecture, the placement and routing cost is mostly around 10 ms. This is mainly due to the usage of dedicated routing structures for IP-IO and inter-IP connections. The proposed architecture and placement/routing algorithms are therefore suitable for on-line systems.

It is noticed that when the arrival interval is small, the average waiting time and placement time increase dramatically. More sophisticated scheduling algorithms may be necessary in those cases.

7 Conclusions and Future Works

7.1 Conclusions

Reconfigurable machines can accelerate many applications by adapting to their needs through hardware reconfiguration. OS4RC stands for Operating System for Reconfigurable Computing. The OS4RC in this dissertation focuses on runtime partial reconfiguration which allows the swapping in and out of IP circuits at run time.

The main goal of this research is to address some problems that come from the gap between OS4RC and existing chip architectures and the gap between OS4RC models and practical applications. Those problems include (1) for some OS4RC models, there is no data exchange channel between tasks residing on a Field Programmable Gate Array (FPGA) chip and between a circuit and FPGA I/O pins or (2) for other models, their inter-IP communication channels do not work well with 2-D partial reconfiguration. To address these problems, a new FPGA chip architecture has been proposed and a corresponding OS4RC kernel is then specified.

Based on an array of clusters of configurable logic blocks, with each cluster serving as a partial reconfiguration unit, and a mesh of segmented buses, the new architecture provides inter-IP and IP-I/O communication channels. The proposed OS4RC kernel has been developed with the new architecture in mind.

The area cost and the configuration memory size of the new chip architecture have been calculated and analyzed. They are judged to be relatively low compared to

commercial FPGA chips. The efficiency of the OS4RC kernel has been evaluated via simulation using three different task models. It is determined that the OS4RC is quite efficient in terms of runtime placement and routing time. Some other performance metrics have also been used to evaluate the performance of the implemented OS4RC.

For further improvements of this work, there are two directions: (1) the routing algorithm can be relaxed so that no ripping-off is used which would require some more routing resources/performance tradeoff study, and (2) the incorporation of block RAMs, which are quite common on modern FPGA chips, can be incorporated in the new architecture. The latter one is described in more details in the rest of the chapter by considering its impacts on architectures and operating systems separately.

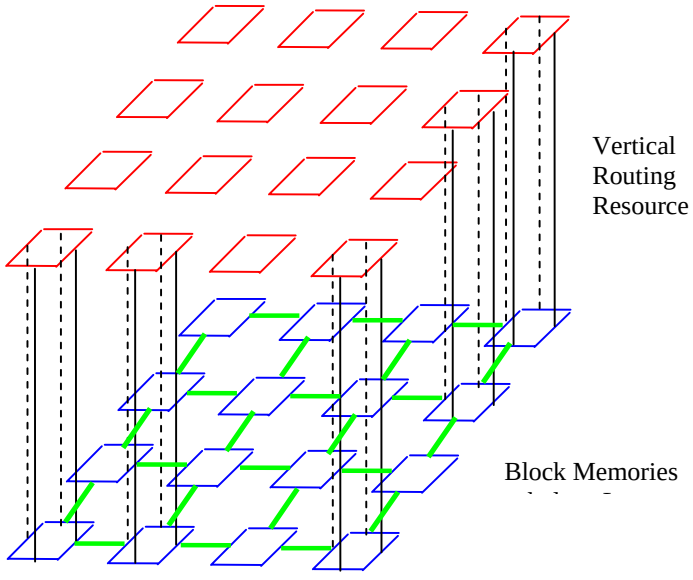


Figure 7.1 Block Memories on a 3D FPGA

7.2 Future Works on Architecture

One possible solution is to take advantage of modern 3D FPGA technology and to add an extra memory layer to our proposed architecture. Therefore under each cluster there is a memory block, as shown in Figure 7.1. Block RAMs at the bottom layer have their own dedicated data and address buses. Buses associated with block RAMs are independent from those general-purpose routing wires and segmented buses. They are used to connect to memory blocks.

7.2.1 Why 3D Architecture?

There are several reasons.

(1) A dedicated memory layer can support high memory capacity. Most current commercial FPGAs have only limited capacity of on-chip memories, and off-chip memories still play an important role. As there are only limited data buffered on chip, data exchange between FPGA and off-chip memory becomes intensive and an I/O bottleneck problem occurs. For applications which have continuous data flow, such as DSP applications, the possible acceleration capability of FPGA chips is largely compromised.

For example, a black and white image with a size of 768*512 *8 bits needs a frame buffer of 384K bytes. Most existing FPGA chips cannot support such a requirement except some very large ones. Virtex II Pro $XC2VP100$ is such a chip. It has 7992k bits (or 999k bytes) block RAMs. But for $RGB(\alpha)$ color image processing, the memory capacity needs to be as high as 3×384 K or 4×384 K bytes. For *MPEG* applications, M , P and I frames may reside on chip at the same time. This means extra

memory requirement again! In a practical hardware image processing system, more than one stage of image buffers (frequently dual port memories) is normally required in a processing pipeline.

(2) On most commercial chips like Xilinx Virtex, block RAMs are only available at some specific locations. If an IP circuit has to work along with block memories, then its possible location inside the chip is largely constrained by connections with block RAMs. With a dedicated layer, memory blocks can be arranged uniformly, as shown in Figure 7.1.

(3) If a large amount of memory blocks are uniformly put on the same layer as clusters stay on, the performance of the FPGA chip will degrade. As the capacity of on-chip memory increases, the associated cost on more input/output pins, buses and memory configuration mapping blocks will also increase. Memory configuration mapping block is some kind of crossbar mechanism, which can map a block memory into different width /depth combinations to satisfy different application requirements [Wilton97]. FPGA routing channels will be further widened to accommodate these extra facilities. Such overcrowded facilities may cause big capacitance between wire segments, and chip speed performance will be impaired. A 3-D FPGA architecture has been proved to be a good solution to build fast chips with a small area [Rahman03].

7.2.2 Scalability of Block RAMs

Inspired by Atmel AT40K/AT40KLV architectures, dual port block memories can be scaled into different depth / width to satisfy different requirements. Suppose each block RAM has a width W and depth D. In Figure 7.2, an example is shown how to build

a RAM with $2W$ width and $2D$ depth. The two block RAMs in dashed lines are not used. Each of the two rows contributes to half of the $2W$ width, while each of the two columns contributes to half of the $2D$ depth. A decoder can be built with logic at the top layer.

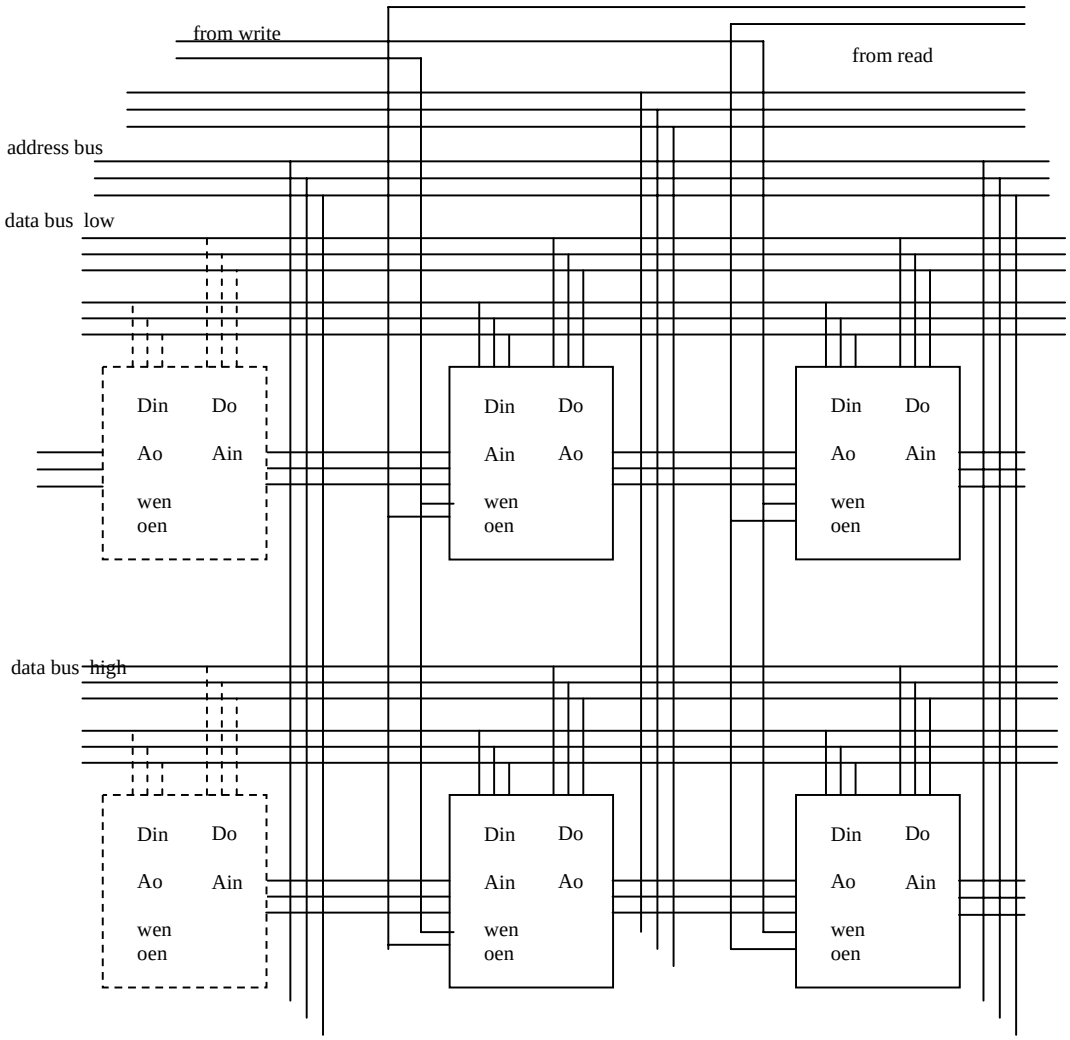


Figure 7.2 Scalability of Block RAMs

7.2.3 Connections between Clusters and Memory Blocks

Using a 3D architecture, Silviu Chiricescu et al. designed connections between CLBs and a switch box below them [Silviu01b]. Dedicated connections between a cluster and an underlying memory block can be built up using similar approaches. Outside clusters, connections can be built with 3D switch boxes. In Figure 7.3, a possible routing path is depicted with segments that have arrows.

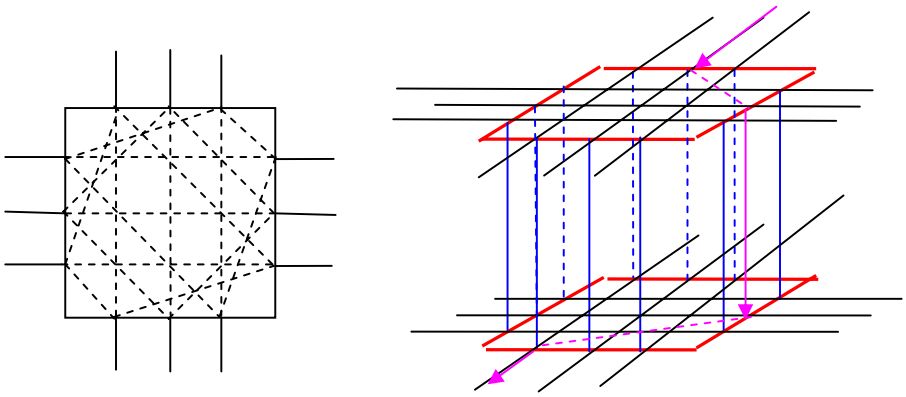


Figure7.3 Switch Box with 3D Structure

7.3 Future Works on Operating System

The capacity of a memory block underlying a cluster may be decided by letting its area (with the area of associated routing resources included) match that of the cluster above, although a cluster does not have to be associated with only one memory block, and vice versa. The area of the logic part (at the top layer) and the memory part (at the lower layer) may be quite different. As a result fragmentations on the top layer and bottom layer are also different. When a new IP circuit arrives, how to place it, route it and

schedule it with the consideration of a new dimension? This is a question that remains for future research.

Appendix: Area Cost of Some Primitive Components

In this appendix, some primitive components are listed, and their areas are calculated. These components are used in the proposed segmented bus network.

A.1 One-bit SRAM Memory Cell

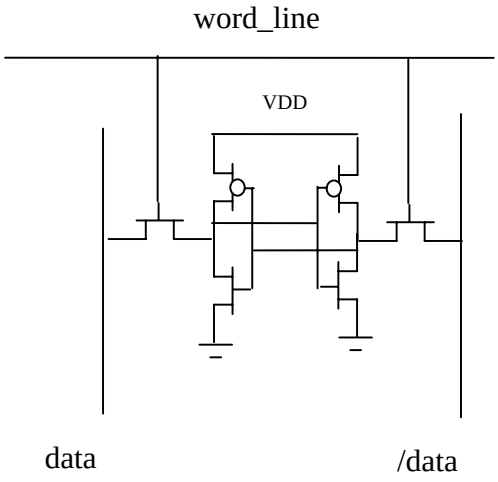


Figure A-1 One-Bit Memory Cell

Such a memory cell is composed of six transistors, even though five transistors are possible. In an FPGA chip, each programmable point is controlled by a memory cell on which both the stored data and its complementary are available. Polarity of the final output signal may be different from that of the very initial input signal if odd number of

inverters is cascaded. If this happens, the complementary version of the stored data should be used initially.

A.2 Inverter with the Minimum Driving Strength

The minimum inverter is composed with a NMOS transistor and a PMOS transistor. The NMOS transistor has the minimum channel width, while the PMOS transistor has twice of the minimum channel width. The area of a NMOS minimum transistor is used as the unit to evaluate a circuit design. The area of a transistor after sizing can be calculated as [Betz99, page 133]:

$$0.5 + \frac{0.5 * \text{drive strength of the sized transistor}}{\text{drive strength of the minimum transistor}}$$

The area of the PMOS transistor with a channel width of two is therefore

$$0.5 + 0.5*2/1 = 1.5$$

Hence the area of such an inverter has an equivalent area of 2.5 minimum transistors.

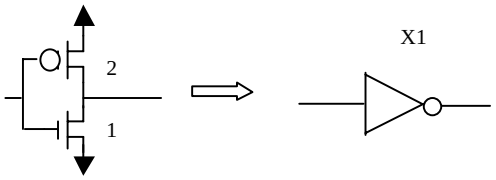


Figure A-2 Inverter with Unit Driving Capability

Inverters with higher driving capability can be built with cascaded inverters with gradually increasing sizes. In the following figure, it is a buffer with four times driving

capability of a minimum inverter. Numbers in the figure represent channel widths of corresponding transistors.

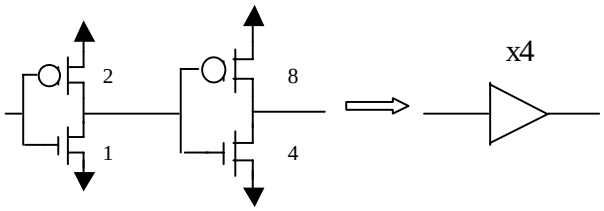


Figure A-3 Buffer with Four Times Driving Capability

A.3 Tri-state Buffers

As discussed before, a segmented bus wire going through a cluster has one bi-directional tri-state buffer at each end. The tri-state buffer has an architecture as in Figure A-4. According to Betz, a tri-state buffer with five times driving capability has an area of 13.7(≈14) minimum width transistors.

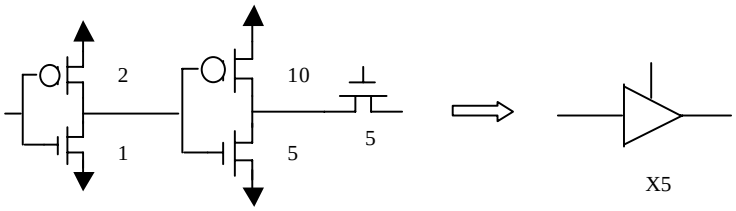


Figure A-4 Tri-State Buffer with Five Times Driving Capability

If such a tri-state buffer is associated with a one-bit memory cell, then its area is equivalent to 14+6 =20 minimum NMOS transistors.

In switch blocks, bi-directional tri-state buffers are used with an architecture as in Figure A-5. Such a buffer has an area of 40 minimum NMOS transistors. It is assumed that a one-bit memory cell is associated with each of its two control signals.

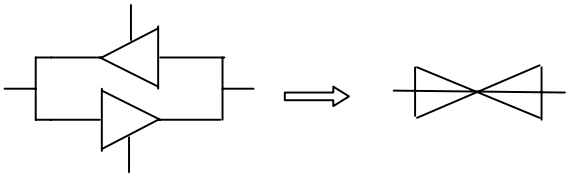


Figure A-5 Tri-state Buffer inside Switch Box

If there is any need for buffers or switch transistors with higher driving capability, sizing technique can be applied to build bigger components. In this dissertation the area of such a bi-directional tri-state buffer is denoted as A_{3st} after sizing.

A.4 Tri-state Buffer at Cluster Boundary

At the boundary of a cluster, output bus wires and input bus wires are separate, as shown in Figure A-6.

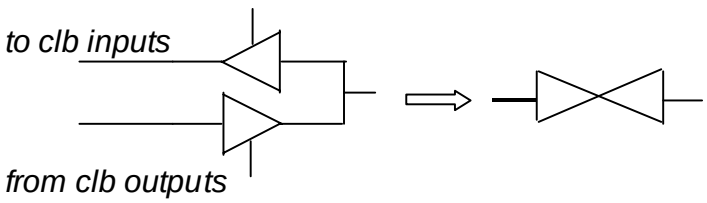


Figure A-6 Tri-State Buffer at Cluster Boundary

Two one-bit memory cells control a bi-directional tri-state buffer along each direction. The two bits decide the behavior of a buffer.

two control bits	tri-state buffer behavior
00	isolated
01	unidirectional
10	opposite unidirectional
11	bi-directional

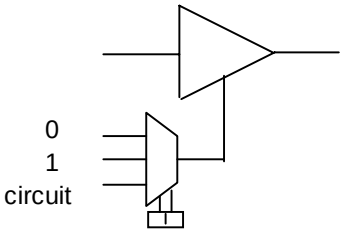


Figure A-7 Tri-state Buffer at Cluster Boundary

In Figure A-7, the area of the tri-state buffer itself is 14, and that of the 3:1 multiplexer part is 21. Therefore the total area is 35.

A.5 Multiplexers or De-multiplexers

A 4:1 multiplexer can be built as in Figure A-8. If each NMOS transistor is a minimum one, then totally six NMOS transistors are needed. Each inverter after sizing is equivalent to three minimum transistors, as the width of its PMOS transistor is twice as wide as the width of its NMOS transistor. Each one-bit memory cell is composed of six minimum transistors. Therefore such a multiplexer is equivalent to 23 minimum transistors in terms of area.

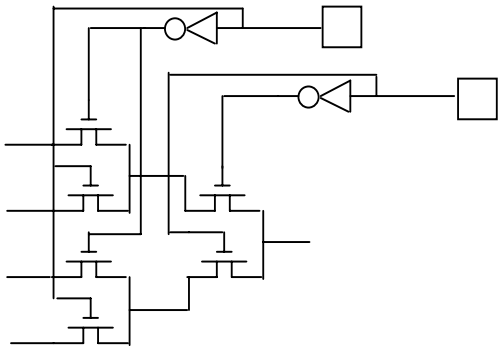


Figure A-8 4:1 Multiplexer

As NMOS transistors are bi-directional, a 4:1 multiplexer can be converted into a 1:4 de-multiplexer, if signals are transmitted in the opposite direction. Hence a 1:4 de-multiplexer has the same area as that of a 4:1 multiplexer.

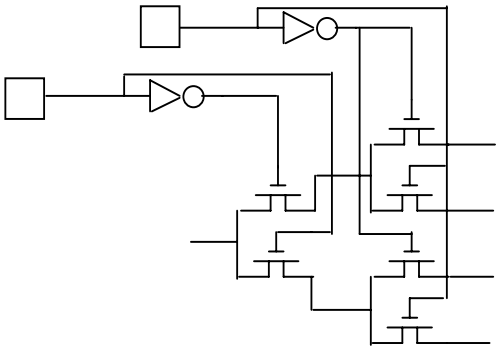


Figure A-9 1:4 De-multiplexer

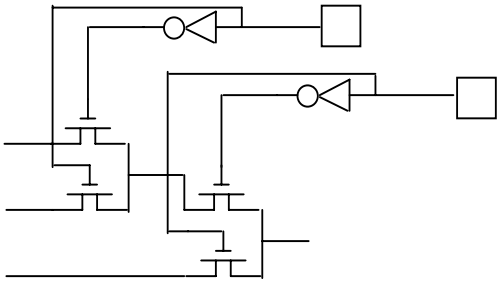


Figure A-10 3:1 Multiplexer or De-multiplexer

From Figure A-10, the area of a 3:1 multiplexer (or 1:3 de-multiplexer) is 21.

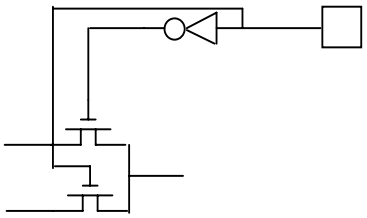


Figure A-11 2:1 Multiplexer or De-multiplexer

As shown in Figure A-11, the area of a 1:2 multiplexer (or de-multiplexer) is therefore equivalent to 10.5 (≈11) minimum transistors.

References

[Altera03] Altera, Flex8000 Programmable Logic Device Family, 2003, page 6.

[Annapolis98] Wildfire Reference Manual, Annapolis, Maryland: Annapolis microsystems, Inc., 1998.

[Atmel02] Atmel AT40K FPGA, data sheet, 2002, Atmel Co.

[Atmel99] Coprocessor Field Programmable Gate Arrays AT6000(LV)Series, data sheet, 1999, Atmel Co.

[Atmel02] FPSLIC on-chip Partial Reconfiguration, 2002, Atmel Co., page 5.

[Baker98] R. Jacob Baker, Harry W. Li and David E. Boyce, "CMOS Circuit Design, Layout and Simulation", IEEE Press, 1998.

[Bazargan00] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast Template Placement for Reconfigurable Computing Systems," IEEE Design and Test of Computers, vol. 17, no. 1, pp. 68-83, 2000.

[Betz99] V. Betz, Jonathan Rose and Alexander Marquardt, "Architecture and CAD for Deep- Submicron FPGAs", Kluwer Academic Publishers, 1999.

[Betz98] V. Betz, "Architecture and CAD for the speed and Area Optimization of FPGAs", Ph.D. dissertation , University of Toronto, 1998.

[Betz96] V. Betz and J. Rose, "Directional Bias and Non-Uniformity in FPGA Global Routing Architectures", IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, 1996, pp. 652 – 659.

[Betz98] V. Betz and J. Rose, “Effect of the Prefabricated Routing Track Distribution on FPGA Area-Efficiency”, IEEE Transactions on VLSI Systems, September 1998, pp. 445 - 456.

[Bobda04] Christophe Bobda, Mateusz Majer, Dirk Koch, Ali Ahmadinia, and J¨urgen Teich, “A Dynamic NoC Approach for Communication in Reconfigurable Devices”, International Proceedings of the Field Programmable Logic and Applications (FPL04), Antwerp, Belgium, August 2004, pp.1032–1036.

[Brown92] Stephen Dean Brown, “Routing Algorithms and Architectures for Field Programmable Gate Arrays”, Ph.D. dissertation, University of Toronto, 1992.

[Budiu02] M. Budiu and S. C. Goldstein, “Compiling application-specific hardware”, In Proc. FPL, LNCS 2438, pp. 853–863, Montpellier, France, 2002.

[Cameron98] W. Najjar, B. Draper, W. Bohm and R. Beveridge , “The Cameron Project: High-level Programming of Image Processing Applications on Reconfigurable Computing Machines”, PACT'98 Workshop on Reconfigurable Computing, Paris, France, Oct. 1998.

[Celoxica03] Celoxica, http://www.celoxica.com. Handel-C Language Reference Manual, 2003. RM-1003-4.0.

[Celoxica05] News release from Cray and Celoxica on September 6, 2005.

[Chang96a] Y.-W. Chang, D. Wong, and C. Wong, “Universal switch module design for d FPGA design”, ACM/Transactions on Design Automation of Electronic Systems, vol.1, pp. 80-101, Jan. 1996.

[Chang96b] Y.-W. Chang, D. Wong, and C. Wong, “Universal switch module design for symmetric–array-based FPGAs”, Proceedings of the of the ACM/SIGDA

International Symposium on Field Programmable Gate Arrays, pp. 80-86, Feb. 1996.

[Compton00] Katherine Compton, James Cooley, Stephen Knol, and Scott Hauck, "Configuration Relocation and Defragmentation for FPGAs", IEEE Symposium on Field- Programmable Custom Computing Machines, 2000.

[Corso86] D.Del Corso, H.Kirrman, and J.D.Nicoud, "Microcomputer Buses and Links", Academic Press, 1986.

[Dally01] William J. Dally and Brain Towels, "Route Packets, Not Wires: On-Chip Interconnection Networks", DAC2001, June, 2001,Las Vegas, Nevada, U.S.A, pp. 684~689

[Dehon96] Andre Dehon, "Reconfigurable Architectures for General-Purpose Computing", Ph.D. thesis, MIT, 1996.

[Dehon00] Andre Dehon, "Balancing interconnect and Computation in a Reconfigurable Computing Arrays", 7th International Workshop on Field-Programmable Gate Arrays, Monterey, CA, Feb. 1999.

[Diessel98] Oliver Frank Diessel, "On Scheduling Dynamic FPGA Reconfigurations-A partial Rearrangement Approach", Ph.D. dissertation, University of Newcastle, Australia, 1998.

[Donath79] Wilm E. Donath, "Placement and Average Interconnection Lengths of Computer Logic", IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol. CAS-26, No.4, April 1979.

[Ebeling95] C.Ebeling, L. McMurchie, S.A.Hauck and Burns, “Placement and Routing Tools for the Triptych FPGA”, IEEE Transactions on VLSI, Dec. 1995, pp. 473-482.

[Edmonds03] Jeff Edmonds, Jarek Grayz, Dongming Liang, Renee J. Miller, “Mining for empty spaces in large data sets”, Theoretical Computer Science, Volume 296, 2003, pp. 435~452.

[Edwards05] Stephen A. Edwards, “The Challenges of Hardware Synthesis from C-like language”, Design, Automation and Test in Europe (DATE'05), Volume 1, pp. 66-67.

[Elbirt00] A. J Elbirt and C. Paar, “An FPGA implementation and Performance evaluation of the serpent block cipher”, ACM/SIGDA international symposium on FPGAs, 2000, pp. 45-56.

[Estrin63] G. Estrin, B. Bussell, R. Turn and J.Bibb, “Parallel Processing in a Restructurable Computer System”, IEEE Transactions on Electronic Computers, Dec. 1963, pp. 747-755.

[Galloway95] D. Galloway, “The Transmogrifier C hardware description language and compiler for FPGAs”, Proceedings of FCCM, pp. 136–144, Napa, CA, 1995.

[Gajski00] D. D. Gajski, J. Zhu, R. D¨omer, A. Gerstlauer, and S. Zhao. SpecC: Specification Language and Methodology. Kluwer, 2000.

[Gr¨otker02] T. Gr¨otker, S. Liao, G. Martin, and S. Swan. System Design with SystemC. Kluwer, 2002.

[Guccione02] S. Guccione, D. Verkest, I. Bolsens, “Design Technology for Networked Reconfigurable FPGA Platforms”, Proceedings of Design, Automation and Test in Europe (DATE) Conference pp. 994-997, March 2002.

[Hadlock77] F.O. Hadlock, “A shortest path algorithm for grid graph”, Networks, 7:323-334, 1977.

[Handa04a] Manish Handa and Ranga Vemuri, “An Efficient Algorithm for Finding Empty Space for Online FPGA Placement”, Proceedings of the 41st Design Automation Conference (DAC’04), June 7–11, 2004, San Diego, California, USA.

[Handa04b] Manish Handa and Ranga Vemuri, “An Integrated Online Scheduling and Placement Methodology”, International Proceedings of the Field Programmable Logic and its Applications (FPL04), Antwerp, Belgium. August 2004.

[Haynes00] Simon D. Haynes, John Stone, Peter Y.K. and Wayne Luk, “Video Image Processing with the Sonic Architecture”, IEEE Transactions on Computer, April, 2000, pp. 50-57.

[Horta02] Edson L. Horta, John W.Lockwood, David Parlour “Dynamic Hardware Plugins in an FPGA with Partial Runtime Reconfiguration”, DAC2002, June, New Orleans, USA.

[IBM99] “CoreConnect Bus Architecture.” http://www3.ibm.com/chips/products/coreconnect/.

[Jean03] Jack Jean, Xinzhong Guo, Fei Wang, Lei Song, Ying Zhang, “A Study of Mapping Generalized Sliding Window Operations on Reconfigurable

Computers", Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, June 23 - 26, 2003, Las Vegas, Nevada, USA, pp. 51-57.

[Kalte04] H. Kalte, M. Koester, B. Kettehoit, M. Porrmann and U.Rukert, "A Comparative Study on System Approaches for Partial Reconfigurable Architectures", Proceedings of the Intl' conference on Engineering of Reconfigurable System and Algorithms, Las Vegas, Nevada, June, 2004.

[Kambe01] T. Kambe et al., "A C-based synthesis system, Bach, and its application", International Proceedings ASP-DAC, pp. 151–155, Yokohama, Japan, 2001.

[Karim01] Faraydon Karim, Anh Nguyen, Sujit Dey, and Ramesh Rao, "On-Chip Communication Architecture for OC-768 Network Processors", DAC2001, June, 2001, Las Vegas, Nevada, U.S.A, pp. 678~683.

[Keating02] Michael Keating and Pierre Bricaud, "Reuse Methodology Manual", Kluwer Academic Publishers, 2002.

[Kress00] R.Kress, R.W.Hartenstein and U.Nageldinger, "An operating system for custom computing machines based on Xputer paradigm", Proceedings 7th International Workshop on Field-Programmble Logic and Applications, London, UK, Sept. 1-3 1997, pp. 304-313.

[Ku90] D. C. Ku and G. De Micheli, "HardwareC: A language for hardware design", T.R. CSTL-TR-90-419, Stanford University, CA, Aug. 1990.

[Kuacharoen00] Pramote Kuacharoen, Mohmed A. Shalan and Vincent J. Mooney, "A Configurable Hardware Scheduler for Real-Time Systems"

[Kusse97] E.Kusse, “Analysis and circuit design for low power programmable logic modular”, M.S. thesis, Departmeant of Electrical Engineering and Computer Science, University of California at Berkeley, 1997.

[Law00] Averill M. Law and W. David Kelton, Simulation Modeling and Anylysis, third edition, Mc-Graw Hill, 2000.

[Lee61] C. Lee, "An Algorithm for Path Connections and its Applications," IRE Trans. Electron. Comput., vol. EC-10, 1961, pp. 346-365.

[Lee95] Y.-S. Lee and A. Wu, “A Performance and Routability Driven Router for FPGAs Considering Path Delays”, DAC, 1995, pp. 557 - 561.

[Li00] Z. Li, K. Compton and Scott Hauck, “Configuration Caching for FPGAs”, IEEE Symposium on Field-Programmable Custom Computing Machines, 2000.

[Marescaux02] Theodore Marescaux, Andrei Bartic etc., “Interconnection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs”, FPL02, Proc. 12th Int. Workshop on Field-Programmble Logic and Applications, Montpellier, France, Sep. 2002, pp. 795~805.

[Marescaux04] Theodore Marescaux, V. Nollet et al., “Rum-time support for heterogeneous multitasking on reconfigurable SOCs”, INTEGRATION, the VLSI journal at www.elsevier.com, March, 2004.

[Marquardt99]Alexander R. Marquardt, “Cluster-Based Architecture, Timing-Driven Placement for FPGAs ”, Master’s thesis, University of Toronto, 1999.

[Miller88] Russ Miller, V.K. Prasanna-Kumar, D.Resis and Q.F.Stout, “Meshes with reconfigurable buses”, Proceedings of the fifth MIT Conference on Advanced Research in VLSI, Boston, 1988.

[Moor65]Gordon E. Moore, “Cramming More Components Onto Integrated Circuits”, Electronics, April 19, 1965.

[Moor03] Gordon Moore, “No Exponential is Forever … but We Can Delay ‘Forever’”, presentation at the International Solid State Circuits Conference (ISSCC), February 10, 2003, http://www.intel.com/research/silicon/mooreslaw.htm.

[Najjar99] Walid A. Najjar, Edward A. Lee and Guang R. Gao, “Advances in the Dataflow Computational Model”, CAPSL Technical Memo 29, Computer Architecture and Parallel System Laboratory, Department of Electrical and Computer Engineering, University of Delaware, 1999.

[Nollet03] V. Nollet, P. Coene, D. Verkest, S. Venalde, R. Lauwereins, “Design an Operating System for a Heterogeneous Reconfigurable SoC”, Proceedings of the International Parallel and Distributed Processing Symposium, 2003

[Oxford97] Oxford Hardware Compilation Group, “The Handel language”, Technical Report, Oxford University 1997.

[Patterson03] Cameron D. Patterson, “A Dynamic Modular Server for Embeded Platform FPGAs”, Proceedings of the International Conference on Engineering of Reconfigurable System and Algorithms, Las Vegas, Nevada, June 2003.

[Rahman03] Arifur Rahman, Shamik Das, Anantha P. Chandraksan and Rafaei Reif, “Wiring Requirement and Three-Dimensional Integration Technology for Field Programmable Gate Arrays”, IEEE Transaction on very large scale Integration (VLSI) Systems, Vol. 11, No.1, February 2003, pp. 44-54.

[Rencher97] Michael Rencher and Brad Hutchings, “Automated Target Recongnition on Splash 2”, IEEE symposium on FPGAs for Custom Computing Machines, April, 1997, Napa Valley, California, pp. 192-200

[Rissa00] Tero Rissa and Jarkko Niittylahti, “A Hybrid Prototype Platform for Dynamic Reconfigurable Designs”, Proc. 10th Int. Workshop on Field-Programmble Logic and Applications, Villach, Austria, Aug. 2000, pp. 371-378

[Rose91] J.S. Rose and S. Brown, “Flexibility of Interconnection Structures for Field-Programmable Gate Arrays”, IEEE JSSC, Vol. 26, No. 3, March 1991, pp. 277-282.

[Sait95] Sadiq M. Sait and Habib Youssef, “VLSI Physical Design Automation –Theory and Practice”, IEEE Press, 1995.

[Samet90] Samet, “Application of Spatial Data Structures–Computer Graphics, Image Processing, and GIS”, Addison-Wesley Publishing Company, Inc., 1990.

[Sastry86] Sarma Sastry and Alice C. Parker, “Stochastic Models for Wireability Analysis of Gate Arrays”, IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol. CAD-5, No.1, January 1986.

[Schaumont98] P. Schaumont et al., “A programming environment for the design of complex high speed ASICs”, Proceedings of DAC, pp. 315–320, 1998.

[Sedcole03] N.P. Sedcole, P.Y.K. Cheung, G.A. Constantinides, and W. Luk, "A Reconfigurable Platform for Real Time Embedded Image processing", FPL03, Proc. 13th Int. Workshop on Field Programmable Logic and Applications, Villach, Lisben, Portugal, Sep. 2003, pp. 606-615.

[Sedra98] Andel S. Sedra and Kenneth C.Smith, “Microelectronic Circuits”, Oxford University Press, 1998.

[Sherwani95] Naveed Sherwani, Siddharth Bhingarde and Anand Panyam, “Routing in the Third Dimension—From VLSI Chips to MCMs”, IEEE press, 1995.

[Sidhu00] Reetinder Sidhu, Sameer Wadhwa, Alessandro Mei, and Viktor K. Prasanna, “A Self – Reconfigurable Gate Array Architecture”, Proceedings 10th International Workshop on Field Programmable Logic and Applications, Villach, Austria, Aug. 2000, pp. 106-120.

[Silviu01a] Silviu Chiricescu, Miriam Leeser and M. Michael Vai, “Design and Analysis of a Dynamically Reconfigurable Three-Dimensional FPGA”, IEEE Transaction on very large scale Integration (VLSI) Systems, Vol. 9, No.1, February 2001.

[Silviu01b] Silviu Chiricescu, “Parametric Analysis of a Dynamically Reconfigurable Three-Dimensional FPGA”, Ph.D. dissertation, Northeastern University,

[Skliarova03] Iouliia Skliarova, António de Brito Ferrari, “Reconfigurable Hardware SAT Solvers: A Survey of Systems”, Proceedings of Field Programmable Logic and Application, 13th International Conference, FPL 2003, Lisbon, Portugal, September 1-3, 2003, pp. 468-477.

[Soderman98] D. Soderman and Y. Panchul, “Implementing C algorithms in reconfigurable hardware using C2Verilog” Proceedings of FCCM, pp. 339–342, 1998.

[Sonics] “Sonics _Networks Technical Overview.” http://www.sonicsinc.com/sonics/support/documentation/whitepapers/data/Overview.pdf.

[Soukup78] J. Soukup, “Fast Maze Router”, Proceedings of 15th Design Automation Conference, 1978, pp. 100 - 102.

[Steiger04] Christoph Steiger, Herber Walder and Macro Platzner, “Operating System for Reconfigurable Embeded Platforms: Online Scheduling of Real-Time Tasks”, IEEE Transactions on Computers, Vol. 53, No.11, Nov. 2004. pp 1393-1407.

[Stratix03] Altera, Stratix Device Handbook, 2003, volume 1, 2003, pp. 2-3.

[Stroud88] C. E. Stroud, R. R. Munoz, and D. A. Pierce, “Behavioral Model Synthesis with Cones”, Design & Test of Computers, 5(3):22–30, July 1988.

[Swartz98] Jordan S.Swartz, “A High- Speed Timing Aware Router for FPGAs”, Master’s thesis, University of Toronto, 1998.

[Tatineni02] Shobharani Tatineni, “Dynamic Scheduling, Allocation and Compaction Scheme for Real-Time Tasks on FPGAs”, Master’s thesis, Louisiana State University, May 2002.

[Tessier02] Russell Tessier, “Fast Placement Approaches for FPGAs”, ACM Transactions on Design Automation of Electronic Systems, Vol. 7, No. 2, April 2002.

[Trimberger94] S.M.Trimberger, Ed., Field Programmable Gate Array Technology. Norwell, MA: Kluwer, 1994.

[VPR00] Vaughn Betz, “VPR and T-VPack User’s Manual 4.30”, http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html.

[VSIA03] “PBD Definitions and Taxonomy Document, Version 1.00 (PBD 1 1.0)” VSIA Platform Based Design DWG, Sept. 2003.

[Wakabayashi99] K.Wakabayashi, “C-based synthesis experiences with a behavior synthesizer”, Cyber, International Proceedings DATE, pp. 390–393, 1999.

[Wang05] Fei Wang, Jack Jean and Shuxia Sun, “Aspect Ratio Effects on Reconfigurable Computing”, Proc. of the International Conference on Engineering of Reconfigurable Systems and Algorithms, Las Vegas, Nevada, USA, June 2005

[Weinhardt99] Markus Weinhardt and Wayne Luk, “Pipeline Vectorization for Reconfigurable Systems”, FCCM 1999: pp. 52-62.

[Wilton97] Steven J.E.Wilton, “Architectures and Algorithms For Field-Programmable Gate Arrays with Embedded Memory”, Ph.D. dissertation, University of Toronto, 1997.

[Wigley00] Grant Wigley and David Keamey, “The First Real Operating System for Reconfigurable Computers”, 6th Australasian Computer Systems Architecture Conference, 2001, p 130.

[Wigley05] Grant Brian Wigley, “An Operating System for Reconfigurable Computing”, Ph.D. dissertation, The University of South Australia, 2005

[Xilinx94] Xilinx, The programmable Logic Data Book, 1994.

[Xilinx96] Xilinx, XC6200 Field Programmable Gate Arrays Data Book, 1996.

[Xilinx98] Xilinx, XC5200 Series Field Programmable Gate Arrays, Data Sheet, 1998.

[Xilinx03] Xilinx, Virtex-II Platform FPGAs: Complete Data Sheet, 2003, p34.

[Xilinx04] Virtex-II™ Platform FPGAs: Detailed Description, http://www.xilinx.com/bvdocs/publications/ds031-2.pdf

[XAPP151] Xilinx Application Note, Virtex Series Configuration Architecture User Guide, March 2003.

[XAPP290] Xilinx Application Note, Two Flows for Partial Reconfiguration: Module Based or Difference Based. November 25, 2003.