# Using Multi-Bit Logic Blocks and Automated Packing to Improve Field-Programmable Gate Array Density for Implementing Datapath Circuits

Andy G. Ye and Jonathan Rose
*University of Toronto*
*{yeandy, jayar}@eecg.utoronto.ca*

## Abstract

*As the logic capacity of Field-Programmable Gate Arrays (FPGAs) increases, they are being increasingly used to implement large arithmetic-intensive applications, which often contain a large proportion of datapath circuits. Since datapath circuits usually consist of regularly structured components, called bit-slices, it is possible to utilize datapath regularity in order to achieve significant area savings through FPGA architectural innovations. This paper describes such an FPGA logic block architecture, called a multi-bit logic block, which employs configuration memory sharing to exploit datapath regularity. It is experimentally shown that, comparing to conventional FPGA logic blocks, the multi-bit logic blocks can achieve 18% to 26% logic block area reduction for implementing datapath circuits, which represents an overall FPGA area saving of 5% to 13%. A packing algorithm for the multi-bit logic block architecture is also proposed in this paper; and it is used to empirically find the best values for several important architectural parameters of the new architecture, including the most area efficient granularity values and the most area efficient amount of configuration memory sharing.*

## 1. Introduction

Field-Programmable Gate Arrays (FPGAs) that process multiple bits of data at a time represent a new architectural approach for implementing datapath circuits on reconfigurable hardware that can significantly reduce the amount of programming information required to configure an FPGA. The main benefit of this reduction in programming information is the subsequent reduction of configuration memory bits, which can lead to increases in logic density. Called multi-bit FPGAs, the detailed implementation of these devices often consists of multiple-bit wide logic blocks and routing resources that take advantage of datapath regularity by sharing a single set of configuration memory across multiple sets of programmable resources. This sharing results in a denser FPGA that is especially efficient at implementing large arithmetic-intensive datapath circuits including computer graphics, multimedia, digital signal processing, and Internet routing applications.

Several multi-bit FPGA architectures have been proposed in the past [1]–[12] with a wide range of logic block designs. In this work, we focus on the study of logic cluster-based multi-bit logic blocks. In particular, we propose a specific logic block architecture along with its packing algorithm (the step in the CAD flow that chooses which logic elements to group together in a cluster). The area efficiency of the proposed logic block architecture is then empirically evaluated. The primary reason for the choice of logic cluster-based logic blocks is due to the fact that logic clusters are the building blocks of many state-of-the-art commercial FPGAs (including the Altera Flex, Stratix, and Cyclone series [17] and Xilinx 5200, Virtex, and Spartan families [18] of FPGAs), and with their ever-increasing logic capacity, commercial FPGAs are being increasingly used to implement large datapath-intensive applications.

For multi-bit FPGAs, it is essential to have a set of automated design tools in order to make the effective use of their multi-bit architectures. As a result, a set of datapath-oriented CAD tools, including synthesis, packing, placement, and routing tools, have been developed at the University of Toronto; and in this paper, we focus on the particular problem of automated packing. Packing for multi-bit FPGAs is more difficult than classical packing [14] [15] [16], because, to effectively utilize configuration memory sharing, the packer has to preserve the regularity of datapath circuits on top of the conventional packing objectives of achieving the smallest possible implementation area and the shortest possible critical path delay.

To investigate the area efficiency of the logic block architecture, we experimentally determine the best granularity values of and the best amount of configuration memory sharing for the proposed logic blocks. Extensive research [19] [20] [15] [21] has been conducted in the past in order to determine the best sizes and structures for conventional FPGA logic blocks. These studies have shown the importance of logic block architecture on the overall area-efficiency of FPGAs. None of the studies, however, considers the problem of configuration memory sharing, which requires the preservation of datapath regularity (all these studies use conventional synthesis and packing algorithms, which destroy the regularity of datapath circuits and essentially turn datapath into finite state machine-like netlists of “randomly” connected logic gates). In this study, a

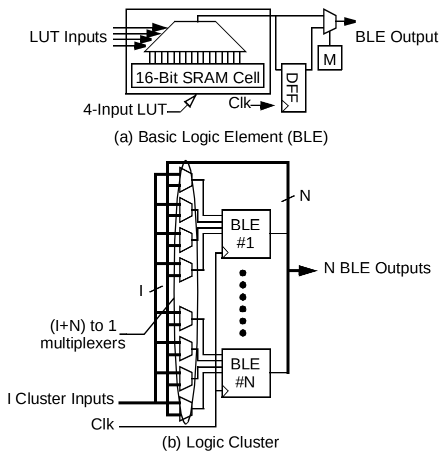


**Figure 1: Logic Cluster Structure**

datapath-oriented synthesis algorithm [22] is used, which preserves a great amount of user-specified regularity. The preserved regularity, in turn, is used by the packing algorithm to investigate the area efficiency of the proposed logic blocks.

The rest of this paper is organized as follows: Section 2 presents the multi-bit logic block architecture; Section 3 describes the packing algorithm; Section 4 presents the experimental results on the area efficiency of the proposed logic blocks; and Section 5 gives concluding remarks.

## 2. The Multi-Bit Logic Block Architecture

The basic building blocks of the multi-bit logic blocks are logic clusters, which were first introduced in [20] as a generalized form of the logic array blocks used in the Altera FLEX8K and FLEX10K series of FPGAs. As shown in Figure 1, each logic cluster is constructed out of Basic Logic Elements (BLEs), which consist of a 4-input Look-Up Table (LUT), a register, and a multiplexer; and a total of 17 Static Random Access Memory (SRAM) bits are required to control the configuration of a single BLE.

$N$ BLEs are grouped together to form a single logic cluster; and each BLE input is connected to the $I$ cluster level inputs and the $N$ BLE outputs through an $I + N$ to 1 multiplexer. Each multiplexer is, in turn, controlled by $\lceil \log_2(I + N) \rceil$ SRAM bits. Since there are a total of $4N$ BLE inputs in a logic cluster, $4N\lceil \log_2(I + N) \rceil$ bits of SRAM are required to control all the multiplexer configurations. Finally, assuming that two SRAM cells are used to control the set/reset logic of each logic cluster [23], the total number of SRAM bits required to control a logic cluster is given by the following equation:

$$C_{SRAM} = 17N + 4N\lceil \log_2(I + N) \rceil + 2 \tag{1}$$

Using the observation in [20] that $I$ should be equal to $2N + 2$, Equation 1 becomes:

$$C_{SRAM} = 17N + 4N\lceil \log_2(3N + 2) \rceil + 2 \tag{2}$$

which is a monotonically increasing function of $N$. The SRAM count for various cluster sizes is summarized in column 3 of Table 1.

As in previous studies [23], in this paper, the active area (the area consumed by transistors), $A$, is used to estimate the overall resource consumption of various logic cluster components. This area is measured as the number of minimum width transistors using the following formula:

$$A = \sum_{\text{All Transistors}} \left(0.5 + \frac{\text{Drive Strength of the Current Transistor}}{2 \times \text{Drive Strength of Min. Width Transistor}}\right) \tag{3}$$

Table 1 lists the total active area consumed by logic clusters of various sizes and the total area consumed by the SRAM bits in column 4 and 5, respectively. The SRAM area as a percentage of the total cluster area is shown in column 6. As shown, unlike the SRAM count, the total SRAM area as a percentage of the total cluster area nearly monotonically decreases with increasing $N$. For small cluster sizes, the SRAM area consists of near 50% of the total cluster area; for extremely large cluster sizes, on the other hand, the SRAM cells consume less than 10% of the total cluster area. Most importantly, however, for the cluster sizes of 4 to 10, which were determined to be the most efficient cluster sizes by previous studies [20], the SRAM cells consume a substantial amount (between 48% to 39%) of the total cluster area. (Note that for the active area calculations, all transistors in a logic cluster are properly sized using the methodology outlined in [23].)

The large amount of area consumed by the SRAM cells motivates the multi-bit logic block design, which shares the configuration memory across the logic clusters. Figure 2 shows the structure of a multi-bit logic block. Here, each logic block contains $M$ logic clusters, where $M$ is called the granularity of the logic block. Note that each cluster is designed to implement a single bit-slice of a datapath circuit and the clusters from a single logic block are used to implement the adjacent bit-slices.

As shown in Figure 2, the configuration memory is shared among $M$ corresponding resources from distinct logic clusters. It is assumed that when the configuration memory of a BLE is shared, the configuration memory of all of its input multiplexers must also be shared. It is also assumed that not all BLEs in a logic cluster must be controlled by shared configuration memory; and the degree of configuration memory sharing, $N_s$, is defined to be the actual number of BLEs in each logic cluster that are controlled by shared configuration.

Table 2 shows the average active area per logic cluster for cluster size ($N$) of 4 and cluster input ($I$) of 10 over a wide range of values for $M$ and $N_s$. This cluster area is compared against the area of a conventional logic cluster to calculate the percentage of area reduction due to configuration memory sharing. As shown, the sharing of SRAM cells can result in significant clus-

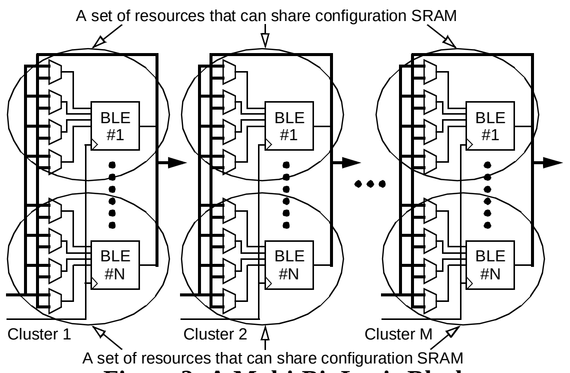

A set of resources that can share configuration SRAM
BLE #1
BLE #N
BLE #1
BLE #N
BLE #1
BLE #N
Cluster 1
Cluster 2
Cluster M
A set of resources that can share configuration SRAM

**Figure 2: A Multi-Bit Logic Block**

**Table 1: SRAM Bits and Area Per Cluster**

| $N$ | $I = 2N+2$ | $C_{SRAM}$ | $A_{Cluster}$ | $A_{SRAM}$ | $\frac{A_{SRAM}}{A_{Cluster}}$ |
|---|---|---|---|---|---|
| **1** | 4 | 19 | 374 | 186 | 50% |
| **2** | 6 | 60 | 730 | 360 | 49% |
| **3** | 8 | 101 | 1205 | 606 | 50% |
| **4** | **10** | **134** | **1681** | **804** | **48%** |
| **5** | **12** | **187** | **2324** | **1122** | **48%** |
| **6** | **14** | **224** | **2919** | **1344** | **46%** |
| **7** | **16** | **261** | **3563** | **1566** | **44%** |
| **8** | **18** | **298** | **4254** | **1788** | **42%** |
| **9** | **20** | **335** | **4994** | **2010** | **40%** |
| **10** | **22** | **372** | **5781** | **2232** | **39%** |
| **12** | 26 | 494 | 7788 | 2964 | 38% |
| **14** | 30 | 576 | 9747 | 3456 | 35% |
| **16** | 34 | 658 | 11897 | 3948 | 33% |
| **20** | 42 | 822 | 16775 | 4932 | 29% |
| **24** | 50 | 1082 | 22997 | 6492 | 28% |
| **28** | 58 | 1262 | 29506 | 7572 | 26% |
| **32** | 66 | 1442 | 36784 | 8652 | 24% |
| **40** | 82 | 1802 | 53643 | 10812 | 20% |
| **48** | 98 | 2354 | 74726 | 14124 | 19% |
| **56** | 114 | 2746 | 97921 | 16476 | 17% |
| **64** | 130 | 3138 | 124189 | 18828 | 15% |
| **80** | 162 | 3922 | 185939 | 23532 | 13% |
| **96** | 194 | 5090 | 262281 | 30540 | 12% |
| **112** | 226 | 5938 | 348992 | 35628 | 10% |
| **128** | 258 | 6786 | 447990 | 40716 | 9% |

ter area reduction. This is especially true for high values of $M$ and $N_s$. For example, the maximum area reduction per logic cluster can be as much as 45% for $M = 16$ and $N_s = 4$.

Although multi-bit logic blocks can consume much less area per cluster due to configuration memory sharing, they might also have lower rate of utilization if they are used to implement irregular circuits. The rest of this paper proposes an automated packing algorithm that preserves as much datapath regularity as possible; and the algorithm is then used to investigate the appropriate granularity values and degrees of configuration memory sharing for multi-bit logic blocks.

**Table 2: SRAM Sharing and Area Reduction**

| $M$ | $N_s$ | $A_{Cluster}$ | % Reduction |
|---|---|---|---|
| **2** | 1 | 1588 | 6% |
| | 2 | 1488 | 12% |
| | 3 | 1387 | 18% |
| | 4 | 1287 | 24% |
| **4** | 1 | 1538 | 8% |
| | 2 | 1387 | 18% |
| | 3 | 1236 | 27% |
| | 4 | 1086 | 36% |
| **8** | 1 | 1513 | 10% |
| | 2 | 1337 | 21% |
| | 3 | 1161 | 31% |
| | 4 | 985 | 42% |
| **12** | 1 | 1504 | 11% |
| | 2 | 1320 | 22% |
| | 3 | 1136 | 33% |
| | 4 | 952 | 44% |
| **16** | 1 | 1500 | 11% |
| | 2 | 1312 | 22% |
| | 3 | 1123 | 33% |
| | 4 | 935 | 45% |

## 3. Packing for Multi-Bit Logic Blocks

The overall flow of the packing algorithm consists of two major steps. In step 1, initialization, the algorithm adjusts the granularity of a graph that represents the input datapath circuit and performs timing analysis. In step two, packing, the algorithm groups nodes of the graph into multi-bit logic blocks. The graph and the two packing steps are described in turn.

### 3.1. Datapath Circuit Representation

Since the primary purpose of the packing algorithm is to preserve datapath regularity, an appropriate format for specifying datapath regularity must be defined. The format used in this work consists of a graph, $G(V, A)$, which is called the *coarse-grain node graph.* The nodes, $V$, of $G(V, A)$, represent the BLEs; and the edges, $A$, of $G(V, A)$ represent the two-terminal connections that connect the BLEs together. Each node of $G(V, A)$ can contain either one or several identical BLEs; and the number of BLEs contained in the node is called the *granularity* of the node. A node containing one BLE is called a *fine-grain node;* and it represents a BLE that does not belong to any datapath. A node containing more than one BLE, on the other hand, is called a *coarse-grain node;* and each BLE in the coarse-grain node is from a unique bit-slice of a datapath circuit.

An example of the coarse-grain node graph is shown in Figure 3, which represents the datapath circuit shown in Figure 4. The graph consists of 11 interconnected nodes representing the 25 BLEs in the circuit. Nodes A through F are 3-bit wide coarse-grain nodes; along with the 2-bit wide nodes E' and F', they represent the eight bit-slices of the datapath. Nodes G, H, and I, on the other hand, are fine-grain nodes, which represent BLEs with the corresponding labels in the irregular logic part of the circuit.

### 3.2. Step 1: Initialization

The initialization step consists of two sub-steps. First, each coarse-grain node whose granularity value is

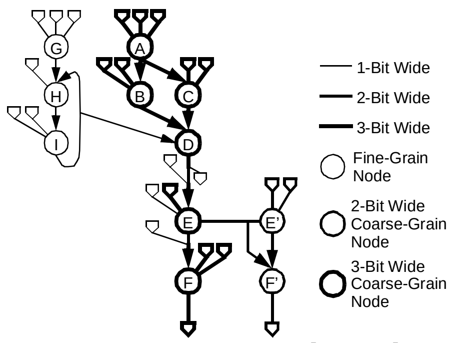


**Figure 3: Coarse-Grain Node Graph**

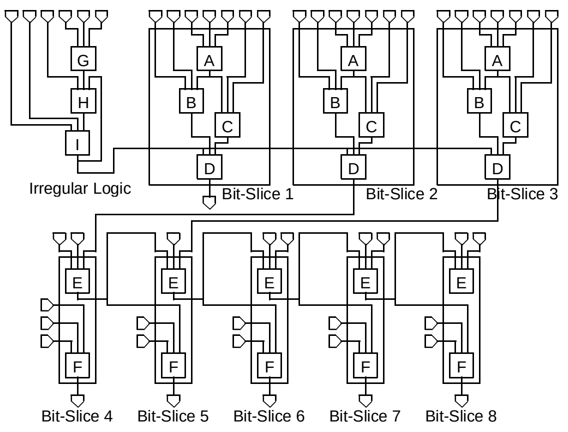


**Figure 4: Datapath Circuit Represented by the Coarse-Grain Node Graph**

greater than the granularity value of the target architecture ($M$) is transformed into a set of nodes. Each node in the set has a granularity value that is smaller than or equal to the granularity of the multi-bit logic blocks. In particular, given a coarse-grain node that is more than $M$ bits wide, starting at the most significant bit of the node, the packing algorithm continuously groups $M$ neighboring BLEs into new coarse-grain nodes. If there are less than $M$ BLEs remaining at the least significant end, these remaining BLEs are grouped by themselves into a node that is less than $M$-bit wide. These newly formed nodes are then used to substitute the original node in the coarse-grain node graph.

Timing analysis is then performed on the input circuit. During timing analysis, the propagation delay and the expected arrival time of each BLE input or output pin is calculated. The slack of each net is then derived from the delay and the expected arrival time. Finally, the criticality value [15] of each net is calculated using the formula:

$$\text{criticality} = 1 - \frac{\text{slack}}{\text{max\_slack}} \tag{4}$$

where $\text{max\_slack}$ is the maximum slack of the input circuit.

## 3.3. Step 2: Packing

During step 2, new multi-bit logic blocks are created one at a time and each logic block is filled with nodes from the coarse-grain node graph. Nodes are added to a logic block in a predetermined order. Assuming that the *i*th BLE in the *j*th cluster is denoted by the

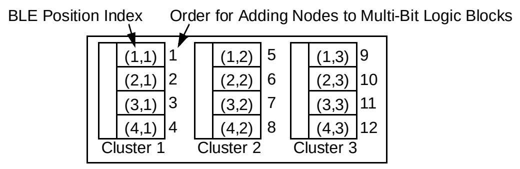


**Figure 5: Order for Filling Multi-Bit LB with N=4, M=3**

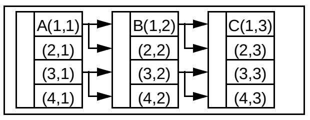


**Figure 6: Equivalence of BLEs in Clusters**

pair of integers $(i, j)$, a node is added to position $(1, 1)$ first. Then, as shown in Figure 5, nodes are sequentially added to positions $(2, 1)$, $(3, 1)$, ..., $(N, 1)$, $(1, 2)$, $(2, 2)$, ... $(N, 2)$, ..., $(1, M)$, $(2, M)$, ... $(N, M)$, if these positions are not already occupied by BLEs.

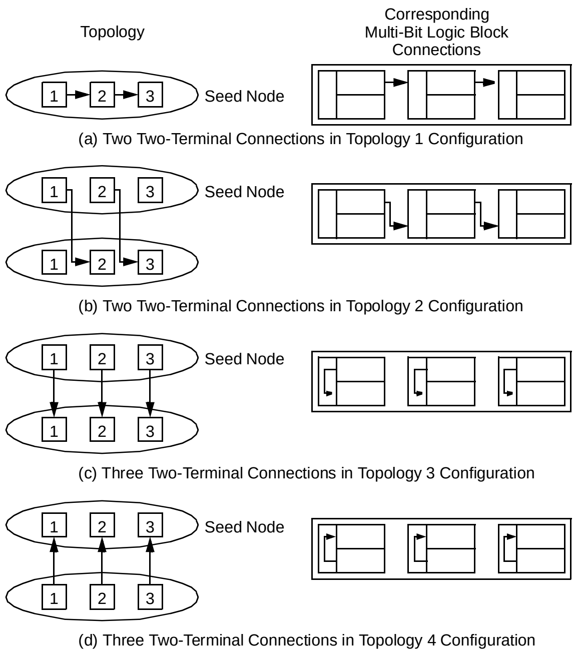
This order of adding nodes to the logic blocks guarantees that if no BLEs have been added to position $(i, j)$, BLE positions $(i, j+1)$, $(i, j+2)$, ..., $(i, M)$ will also be unoccupied. To find the most suitable node for BLE position $(i, j)$, the packing algorithm first finds all nodes whose granularity is less than $M - j + 1$. If $(i, j)$ is equal to $(1, 1)$, then the seed criticality function is used to select the most suitable node from this group of nodes. Otherwise, the attraction criticality function is used. Once the most suitable node with a granularity value of $m$ is determined, the BLEs in this node are added to consecutive BLE positions $(i, j)$, $(i, j+1)$, ..., $(i, j+m-1)$ with the least significant BLE added to position $(i, j)$ and the most significant BLE added to position $(i, j+m-1)$.

It is also assumed that each multi-bit logic block contains a carry network as the one shown in Figure 6. Because of the carry network, not all BLE positions in a cluster are logically equivalent. This lack of equivalency is the reason why the packing algorithm must select nodes for each specific positions in a logic block. An example is shown in Figure 6. Here there are three BLEs, A, B, and C, in a logic block. These BLEs are connected by a carry chain through the carry network. In the figure, the BLE position $(1, 1)$ is equivalent to the BLE position $(3, 1)$; therefore, BLE A can be moved to position $(3, 1)$ provided that BLEs B and C are also moved to position $(3, 2)$ and $(3, 3)$ respectively. However, BLE A cannot be moved to position $(2, 1)$ or $(4, 1)$ since these two positions are not equivalent to BLE position $(1, 1)$ due to the difference in their carry connections.

The remainder of this section describes the two criticality functions, including the seed criticality function and the attraction criticality function, which are used in the packing process.

## 3.4. Seed Criticality

The first node added to a logic block is called a seed. It is selected using a metric called the seed criti-



(a) Two Two-Terminal Connections in Topology 1 Configuration

(b) Two Two-Terminal Connections in Topology 2 Configuration

(c) Three Two-Terminal Connections in Topology 3 Configuration

(d) Three Two-Terminal Connections in Topology 4 Configuration

**Figure 7: Identifying Potential Local Connection**

cality. Implementing a seed in a logic block by itself does not necessarily improve the performance of a circuit; however, when a subsequent node, A, is added to the same logic block, many two-terminal connections that connect the seed node and node A can then be implemented in the local routing networks or the carry network of the logic block, which are inherently much faster than global routing. Consequently, the performance of the circuit is improved. The seed criticality measures the maximum possible performance improvement; and each two-terminal connection that can be implemented inside the logic block is called a *potential local connection*.

Potential local connections can be identified using a pattern matching process against one of the four topologies shown in Figure 7. Here topology A and B contain connections that can be implemented in the carry network of the logic block; and topology C and D contain connections that can be implemented in the local routing networks of the logic clusters.

The formula for calculating seed criticality is shown in Equation . In the equation, the function $\max(X)$ returns the maximum value in a set, $X$, of real numbers. Function $\mathrm{cnt}(X)$, returns the number of elements that are equal to $\max(X)$ in the set $X$. $S(n)$ is the complete collection of all the net criticality values from all potential local connections of node $n$.

$$\begin{gathered}\mathrm{seed\_criticality}(n) \; = \\ \max(S(n)) + \varepsilon \times \mathrm{cnt}(S(n)) + \varepsilon^2 \times d(n) \\ (\varepsilon \ll 1)\end{gathered} \qquad (5)$$

The function, $\max(S(n))$, corresponds to the maximum speed improvement achievable by implementing $n$ as a seed node. $\mathrm{cnt}(S(n))$ is a tie breaker; and it counts the number of potential local connections that can achieve the maximum speed improvement. Note that $\max(S(n))$ and $\mathrm{cnt}(S(n))$ are analogous to the base seed criticality and the number of path affected metrics used in [15], respectively. These functions, however, are more general in nature and are applicable to a wider range of FPGA clustering architectures than the fully connected topology assumed by [15].

The metric distance to source, $d(n)$, on the other hand, is an unmodified version of the same metric defined in [15]. Nodes with the same $\max(S(n))$ values usually are connected together by a single critical path. $d(n)$ measures the order of these nodes along the critical path. Everything else being equal, the node that is the furthest from the source of the critical path is given the highest priority for implementation as a seed node.

## 3.5. Attraction Criticality

Once a seed is added to a logic block, the logic block is then filled based on the attraction criticality metric. Here, each node in the coarse-grain node graph is assigned an attraction criticality value according to Equation 6. The metric consists of four parts: the base seed criticality, $B(n)$, accounts for the performance improvement of implementing the node in the logic block; shared I/O count, $C(n)$, accounts for the number of additional cluster I/Os that is needed to implement the node; and finally secondary attraction criticality, $B_p(n)$, and common I/O count, $C_p(n)$, account for the closeness of the placement resulting from adding the node to the logic block. These four parts are weighted and summed into the attraction criticality. Each part is described in turn.

$$\begin{gathered}\mathrm{attraction\_criticality}(n) = \\ \alpha \times \left(\beta \times B(n) + (1-\beta) \times \frac{C(n)}{P_{max}}\right) + \\ (1-\alpha) \times \left(\tau \times B_p(n) + (1-\tau) \times \frac{C_p(n)}{M \times P_{max}}\right)\end{gathered} \qquad (6)$$

**3.5.1. Base Seed Criticality** As shown in Figure 8, for logic blocks containing at least one node, the connections between the node and the logic block can be classified into two types. The first type consists of connections that can be implemented in the local routing networks of the clusters or the carry network that connects the clusters together. The second type consists of connections that have to be routed through global routing. The implementation of the first type of connections often results in increased performance; and this increase is measured by the base attraction criticality. It is equal to the maximum criticality among all type one connections in addition to all the internal connections of the node that can be implemented in the carry network.

**3.5.2. Secondary Attraction Criticality** Adding a node to a logic block also makes all BLEs in the node physically closer to the BLEs in the logic block. This physical closeness potentially can improve the performance of type two connections. The secondary attraction criticality is used to measure this speed up. It is equal to the maximum criticality among all type two

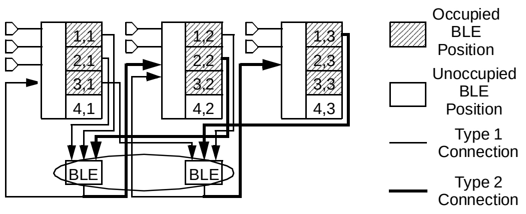


**Figure 8: Adding a Node to a Multi-Bit Logic Block at Position (4,1)**

connections in addition to all internal connections of the node that must be routed through the global routing network.

**3.5.3. Shared I/O Count** Since cluster inputs are limited routing resources, it is important to minimize their usage when adding nodes to logic blocks. As in [15], it is preferable to choose BLEs with the following three types of I/Os for a cluster:

1. a BLE input that is connected to the same net as one of the cluster inputs
2. a BLE input that is connected to one of the cluster outputs
3. a BLE output that is connected to a cluster input

The shared I/O count metric measures the I/O commonalities between a node and a logic block. It is equal to the total number of the three types of BLE I/Os in a node when each BLE is matched with its corresponding cluster. Note that, in Equation 6, $P_{max}$ is defined to be the maximum possible value of the shared I/O count metric. It is used in the equation to normalize the shared I/O count to a value that is between 0 and 1.

**3.5.4. Common I/O Count** Adding a node to a logic block might increase the number of common I/O signals shared by various clusters in the logic block. Routing an input signal that is shared by the two clusters usually requires fewer resources than routing two distinct inputs. Similarly, routing the output of one cluster to another requires fewer routing resources if both clusters are in the same logic block. The common I/O count is used to account for this increase in routing efficiency. It is analogous to the shared I/O count. However, instead of measuring the number of I/O signals that are in common between each BLE and its corresponding cluster, the common I/O count is equal to the total number of BLE I/Os in a node that is in common with all the I/Os of a logic block excluding the signals that have already been counted by the shared I/O count.

Note that for all experiments performed in this thesis, $\alpha$, $\beta$, and $\tau$ are set to be 0.85, 0.75, and 0.75 respectively. These values are experimentally shown to generate good packing results.

## 4. Experimental Results

The packing algorithm has been used to pack several benchmark circuits into multi-bit logic blocks with various granularity values and degrees of configuration memory sharing. The packing results shown in this section are based on the fifteen datapath circuits from the Pico-Java Processor from Sun Microsystems [24]. Each circuit is first synthesized into several granularity values using a datapath-oriented synthesis algorithm [22]; and Table 3 gives the name, size (number of BLEs) of each circuit for a given synthesis granularity value (here the synthesis granularity is defined as the maximum datapath width that is preserved by the synthesis process).

**Table 3: Experimental Circuits**

| **Circuits** | **#BLEs Obtained at Each Synthesis Granularity** | | | | | |
|---|---|---|---|---|---|---|
| | **1** | **2** | **4** | **8** | **12** | **16** |
| **code_seq_dp** | 362 | 364 | 364 | 364 | 364 | 364 |
| **dcu_dpath** | 958 | 962 | 966 | 974 | 982 | 974 |
| **ex_dpath** | 2823 | 2747 | 2649 | 2719 | 2947 | 2955 |
| **exponent** | 467 | 517 | 517 | 539 | 567 | 565 |
| **icu_dpath** | 3254 | 3237 | 3245 | 3245 | 3273 | 3277 |
| **imdr_dpath** | 1286 | 1268 | 1255 | 1286 | 1288 | 1283 |
| **incmod** | 870 | 862 | 867 | 940 | 948 | 1005 |
| **mantissa_dp** | 912 | 919 | 942 | 966 | 971 | 982 |
| **multmod_dp** | 1602 | 1636 | 1634 | 1636 | 1636 | 1636 |
| **pipe_dpath** | 452 | 499 | 452 | 503 | 503 | 501 |
| **prils_dp** | 363 | 396 | 393 | 385 | 385 | 393 |
| **rsadd_dp** | 350 | 314 | 313 | 305 | 305 | 305 |
| **smu_dpath** | 561 | 557 | 557 | 560 | 563 | 561 |
| **ucode_dat** | 1264 | 1273 | 1304 | 1278 | 1282 | 1286 |
| **ucode_reg** | 78 | 80 | 82 | 86 | 86 | 94 |

The synthesized circuits are then packed into a set of multi-bit logic blocks containing a variable number, $M$, of clusters. Several values of $M$ are investigated. These values are the same as the ones shown in Table 3, namely 1, 2, 4, 8, 12, and 16; and for each value of $M$, the degree of configuration memory sharing, $N_s$, is also varied from 0 to 4. Note that each cluster is assumed to contain 10 ($I$) input pins and 4 ($N$) BLEs. The experimental results on regularity, cluster count, and area are presented in turn.

### 4.1. Regularity Results

Two yardsticks are used to measure the amount of regularity contained in the benchmark circuits based on the concept of a datapath component. Here, a datapath component is defined to be a group of identically configured BLEs that is a part of a datapath circuit. The number of BLEs in a datapath component is called the width of the component.

The first yardstick measures the percentage of BLEs in all datapath components of width $M$ after packing. The second yardstick measures the percentage of BLEs in all datapath components, which are at least 2-bit wide. Note that for both regularity measurements, $N_s$ is assumed to be zero.

Figure 9 plots these two metrics against the logic block granularity. As shown, over 85% of BLEs are in at least 2-bit wide datapath components regardless of the granularity values. The percentage of BLEs in $M$-bit wide datapath components drops from over 90% when $M$ is equal to 2 to slightly over 50% when $M$ is equal to 12. The high percentage of BLEs contained in $M$-bit wide datapath components for the granularity value of 2 and 4 suggest that at these granularity values,

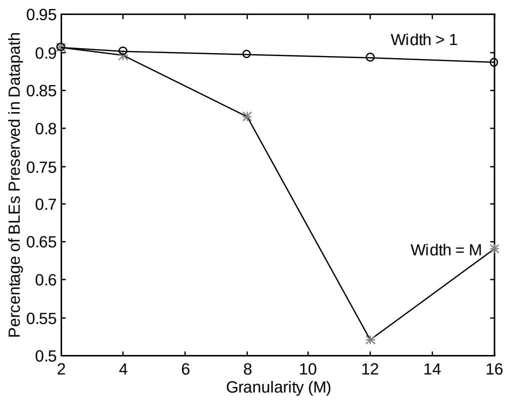


**Figure 9: Regularity vs. Granularity**

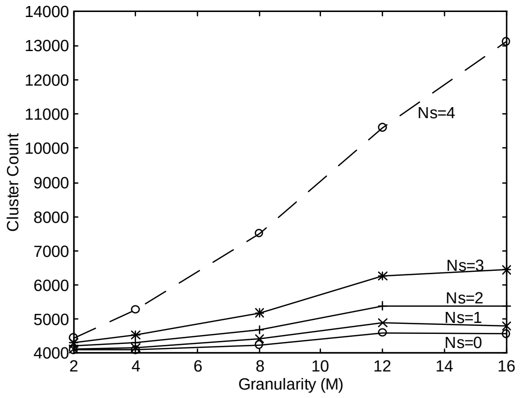


**Figure 10: Cluster Count vs. Granularity**

the multi-bit logic block architecture can benefit substantially from a high degree of configuration memory sharing.

## 4.2. Logic Cluster Count

As discussed in Section 2, the area savings of configuration memory sharing depends on two parameters — the cluster size and the cluster utilization. The cluster utilization can be easily measured by counting the total number of clusters required to implement the fifteen benchmark circuits; and this cluster count is shown in Figure 10. In the figure, the granularity value is shown on the x-axis and the total number of clusters required to implement the fifteen benchmark circuits is shown on the y-axis. There are five lines in the figure, each representing one of the five possible degrees of configuration memory sharing (0, 1, 2, 3, and 4). As expected, when there is no configuration memory sharing, the cluster count is the lowest for a given granularity value; and as the degree of configuration memory sharing increases, so does the cluster count. More interestingly, concurring with the regularity results, for the granularity values of 2 and 4, the increase in the degree of configuration memory sharing from 0 to 3, only results in small increases in cluster count (less than 5% for $M = 2$ and 11% for $M = 4$); and for the granularity value of 2, when $N_s$ is increased from 0 to 4, the cluster count is increased by only 8%. For all other granularity values substantial increases in cluster count is observed.

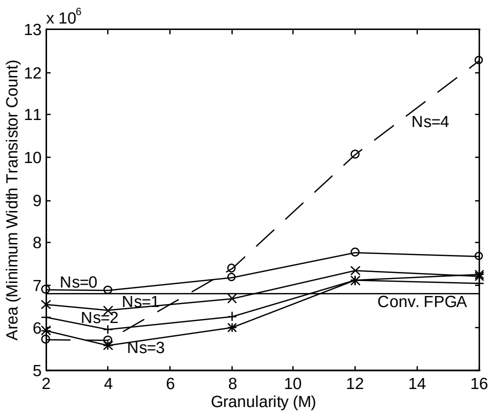


**Figure 11: Area vs. Granularity**

## 4.3. Area Results

The area consumed by the multi-bit logic blocks is plotted in Figure 11. In the figure the x-axis represents the granularity of the architecture, the y-axis represents the total logic block area required to implement the fifteen benchmark circuits. There are six lines in the figure representing the packing area for a conventional FPGA whose clusters contain 4 BLEs and 10 inputs and FPGAs containing multi-bit logic blocks with the degree of configuration memory sharing of 0, 1, 2, 3, and 4 respectively. As shown, for $N_s = 0$, all multi-bit logic block architectures perform slightly worse than the conventional FPGA mainly due to their extra carry logic. For the granularity value of 2, 4, and 8 several configuration memory sharing configurations perform better than the conventional FPGA. In particular, for $M = 2$, logic blocks with $N_s = 4$ perform the best; and this configuration is 16% smaller than the conventional FPGA. For $M = 4$ and $M = 8$, logic blocks with $N_s = 3$ perform the best. Overall, logic blocks with a granularity value of 4 and a degree of configuration memory sharing of 3 give the best area, which is 18% smaller than the conventional FPGA logic blocks. Assuming that the total logic block area consists of 30% to 50% of the total FPGA area, this logic block area saving represents an overall area saving of 5% to 9%.

Finally, Figure 12 shows that the area savings also depends on the size of the SRAM cells. In the figure, it is assumes that each SRAM cell is 1.5 times of the standard size (Larger SRAM cell sizes can be used to improve fault tolerance). This increase in SRAM size results in larger area savings. The best area is achieved when $M = 4$ and $N_s = 4$; and the area saving is 26%, which represents a total FPGA area savings of 8% (assuming 30% of FPGA area is logic block area) to 13% (assuming 50% of FPGA area is logic block area).

## 5. Conclusions

This paper has described a new multi-bit logic block architecture for FPGAs and its associated packing algorithm. Using the packing algorithm, it is empirically shown that, for logic clusters containing 4 BLEs

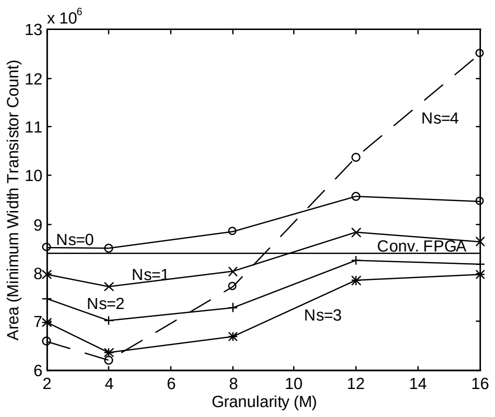


**Figure 12: Area vs. Granularity (Large SRAM)**

and 10 cluster inputs, the most area efficient variant of the multi-bit logic block architecture contains four clusters per logic block and has three BLEs per logic cluster that are controlled by shared configuration memory. In this configuration, the multi-bit logic block area is 18% smaller than the conventional FPGA logic block area. This represents a 5% to 9% reduction in the total FPGA area.

## 6. References

[1] D. Chen and J. Rabaey, "A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High-Speed DSP Data Paths," *JSSC,* Dec. 1992, pp.1895–1904.

[2] A. Yeung and J. Rabaey, "A Reconfigurable Data Driven Multi-Processor Architecture for Rapid Prototyping of High Throughput DSP Algorithms," *HICCS,* Jan. 1993, pp.169–178.

[3] R. Bittner, P. Athanas, and M. Musgrove, "Colt: An Experiment in Wormhole Run-Time Reconfiguration," *Conf. on High-Speed Computing, DSP, and Filtering Using reconf. Logic,* 1996.

[4] D. Cherepacha and D. Lewis, "DP-FPGA: An FPGA Architecture Optimized for Datapaths," *VLSI Design,* 1996, pp.329–343.

[5] C. Ebeling, et. al, "RaPiD A Configurable Computing Architecture for Compute-Intensive Applications," *FPL,* 1996.

[6] J. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," *FCCM,* Apr. 1997, pp.24–33.

[7] E. Waingold, et. al, "Baring It All to Software: Raw Machines," *IEEE Computer,* Sep. 1997, pp.86–93.

[8] T. Miyamori and K. Olukotun, "REMARC: Reconfigurable Multimedia Array Coprocessor," *FCCM,* Apr. 1998.

[9] A. Marshall, et. al, "A reconfigurable arithmetic array for multimedia applications," *FPGA,* 1999, pp.135–143.

[10]A. Alsolaim, et. al, "Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems," *FCCM,* Apr. 2000, pp.205–214.

[11]S. Goldstein, et. al, "PipeRench: a reconfigurable architecture and compiler," *IEEE Computer,* Apr. 2000, pp.70–77.

[12]K. Leijten-Nowak and J. van Meerbergen, "An FPGA architecture with enhanced datapath functionality," *FPGA,* 2003, pp.195–204.

[13]Andy G. Ye, Jonathan Rose, David Lewis, "Architecture of Datapath-Oriented Coarse-Grain Logic and Routing for FPGAs," *CICC,* Sep. 2003, pp.61-64.

[14]V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *FPL,* 1997, pp.213–222.

[15]A. Marquardt, V. Betz, and J. Rose, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density," *FPGA,* Feb. 1999, pp.37–46.

[16]E. Bozorgzadeh, S. Memik, and M. Sarrafzadeh, "RPack: Routability-Driven Packing for Cluster-Based FPGAs," *ASP-DAC,* Jan. 2001, pp.629–634.

[17]*Altera Data Sheet,* Altera, 2003.

[18]*Xilinx Data sheet,* Xilinx, 2003.

[19]J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proc. of the IEEE,* Jul. 1993, pp.1013-1029.

[20]V. Betz and J. Rose, "How Much Logic Should Go in an FPGA Logic Block?," *IEEE Design and Test Magazine,* Spring 1998, pp.10–15.

[21]G. Lemieux and D. Lewis, "Using Sparse Crossbars within LUT Clusters," *FPGA,* 2001, pp.59–68.

[22]Andy G. Ye, Jonathan Rose, and David Lewis, "Synthesizing Datapath Circuits for FPGAs with Emphasis on Area Minimization," *FPT,* Dec. 2002, pp.219-227.

[23]V. Betz, J. Rose, and A. Marquardt, Architecture and CAD for Deep-Submicron FPGAs, Feb. 1999, Kluwer Academic Publishers.

[24]*Pico-Java Processor Design Documentation,* Sun Microsystems, 1999.