Ph.D. Progress Report — Report #3
Architecture of Datapath-Oriented Coarse-Grain Logic and Routing for FPGAs

(March 2002 – March 2003)

Andy Gean Ye

Abstract

As the logic capacity of FPGAs increases, they are being used to implement ever-larger applications. Large applications, whether they are CPUs, graphics processors, digital signal processors, or packet switching networks, typically contain a greater amount of datapath logic, which is highly regular in structure. The efficient implementation of these highly regular structures has become an increasingly important issue to the overall area and performance of many FPGA applications. One promising architectural feature that can increase the area efficiency of datapath circuits implemented on FPGAs is the coarse-grain routing resources. Previously, there has been little study on FPGA architectures containing coarse-grain routing resources, primarily because of the lack of datapath benchmarks and the lack of automated CAD tools targeting coarse-grain routing resources. In this paper, we propose a new FPGA architecture that utilizes coarse-grain routing resources to increase the area efficiency of datapath circuit implementations. We also propose a new routing algorithm that intelligently uses the coarse-grain routing resources provided by the new architecture. Using this router, several other datapath-oriented CAD tools that we designed, and a set of datapath benchmarks that we extracted from the Pico-Java processor, we investigated several variants of our proposed FPGA architecture. We found that, our architecture is the most area efficient when 40% to 50% of the total routing tracks are coarse-grain. Furthermore, our architecture consistently outperform conventional FPGA architecture for implementing datapath designs. Overall, our architecture uses 9.6% less area than the conventional FPGA architecture.

1. Introduction

In the past decade, we have seen a dramatic increase in the logic capacity of FPGAs. As the logic capacity increases, FPGAs are being used to implement ever-larger applications. Large applications, whether they are CPUs, graphics processors, digital signal processors, or packet switching networks, typically contain a greater amount of datapath logic, which is highly regular in structure. The efficient implementation of these highly regular structures has become an increasingly important issue to the overall area and performance of many FPGA applications.

Previous research in [7][8][9][10][11][12][13][14][15] has shown that regularity driven synthesis, placement and routing can be used to improve the density and speed of datapath circuits. The work in [6] also show that further area savings can be

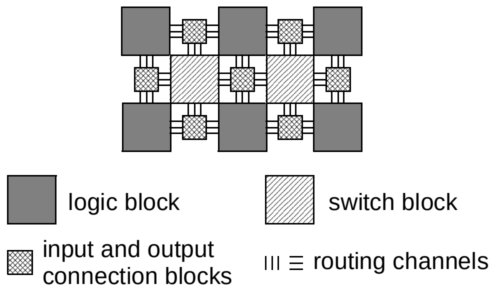

Figure 1: A Typical FPGA Architecture

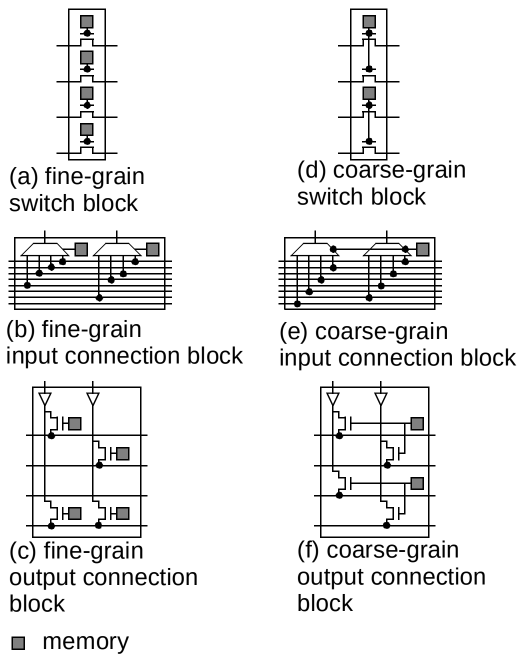
(a) fine-grain switch block
(d) coarse-grain switch block
(b) fine-grain input connection block
(e) coarse-grain input connection block
(c) fine-grain output connection block
(f) coarse-grain output connection block
memory

Figure 2: Fine-grain Routing Resource Topology vs. Coarse-grain Routing Resource Topology

achieved by incorporating datapath-specific features into regular FPGA architectures. One particularly compelling architectural feature is the coarse-grain routing resources suggested by [6]. Their basic notion was to amortize configuration bit area across multiple wires when these wires are data buses. In this paper we perform a detailed explanation of the area advantage of a number of variants of this architectural structure.

By way of a more complete introduction to the architectural concept of coarse-grain routing structures, we first review more traditional FPGA routing. Figure 1 illustrates a typical FPGA, in which logic is implemented in logic blocks [1][3][4][5] that consist of tightly connected look-up tables (LUTs). Logic blocks are then connected together through global routing resources composed of input connection blocks, output connection blocks, switch blocks and routing tracks. These routing resources are made configurable by the programmable switches controlled by SRAM cells. In a typical FPGA, each reconfigurable switch is controlled by a unique set of configuration memory. We call these resources fine-grain routing

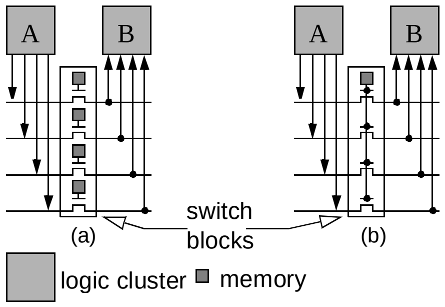

Figure 3: Fine vs. Coarse-Grain Routing

resources. Various fine-grain routing resources, including an input connection block, an output connection block and a switch block, are shown in Figure 2 (a), (b), and (c).

Coarse-grain routing tracks, on the other hand, are grouped together in some number M, and switches associated with each group are collectively controlled by a single set of configuration memory bits. The number of tracks in a single group, M, is called the granularity of the coarse-grain track. Figure 2 (d), (e), and (f) show the structures of various coarse-grain routing resources including an input connection block, an output connection block, and a switch block. Each of these coarse-grain routing resources has a granularity value of two. When routing datapath circuits, coarse-grain routing tracks often can be more efficient at connecting a group of signals from a common source to a common destination and consequently achieve significant area savings. Figure 3 shows such an example. Suppose that there are four one-bit wide signals to be connected between a source Block A and a destination Block B. Figure 3 (a) illustrates the use of four fine-grain routing tracks to connect Block A and Block B. A total of four bits of SRAM is used in the switch box. In Figure 3 (b), a group of four coarse-grain tracks is used instead. Since these four coarse-grain tracks share configuration memory, only one bit of SRAM is used.

Although coarse-grain tracks are more efficient at routing groups of signals that share a common source and a common destination, they are inefficient at routing individual signals. When a group of coarse-grain tracks are used to route a single signal, only one track in the group is utilized. All the other tracks are unused, and the silicon area associated with these tracks is wasted. An efficient FPGA architecture for datapath circuits should therefore contain a mixture of fine and coarse-grain routing resources, as all application circuits are likely to contain both types of signals — signals that can be routed in groups and signals must be routed individually. For FPGAs containing both fine and coarse-grain resources, we need a router that can differentiate these resources and use the most appropriate one for a given type of nets. In this paper, we propose a routing algorithm for FPGAs containing both coarse-grain and fine-grain routing tracks. To our knowledge this is the first published coarse-grain routing algorithm for FPGAs. Using this algorithm, we investigated the question of how many coarse-grain routing tracks should be included in an FPGA targeting highly regular datapath circuits in order to achieve maximum area savings. We also address the question of what is an area efficient segment length for coarse-grain routing tracks.

The rest of this paper is divided into five sections. In Section 2, we give a complete description of a parameterized set of coarse-grain FPGA architectures that we explore. Section 3 describes the routing algorithm. In Section 4, we describe the experimental methodology we use to explore the architecture variant and to compare it against more typical architectures.

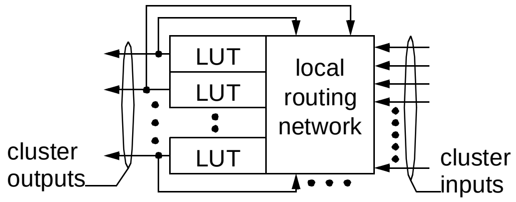

Figure 4: A Logic Cluster [4]

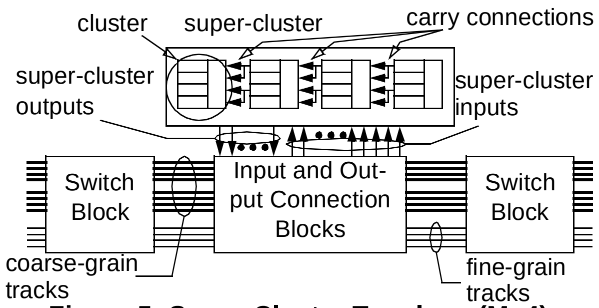

Figure 5: Super-Cluster Topology (M=4)

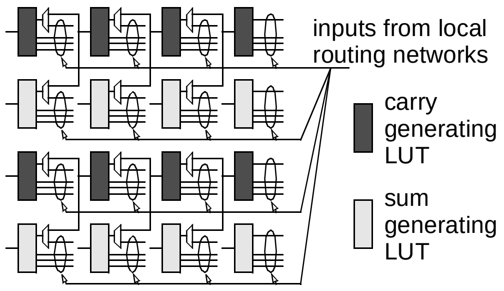

Figure 6: Carry Structures (M=4)

The synthesis, packing, and placement tools are also described in this section. Section 5 presents experimental results for the best percentage of coarse-grain tracks in overall routing resources and the best segment length for coarse-grain routing tracks. We also compare the area efficiency of our proposed architecture with a standard FPGA architecture. Section 6 presents concluding remarks.

2. A Coarse-Grain Datapath FPGA Architecture

In order to utilize the regularity of datapath circuits on coarse-grain routing resources, we need an FPGA architecture that can easily capture the regularity of datapath circuits. Once captured, one should be able to easily map this regularity information onto coarse-grain routing resources. Very few FPGA architectures, either academic or commercial, are designed with datapath regularity and coarse-grain routing resources in mind. As the result, we designed our own architecture (which we will refer to as the datapath architecture in the remainder of this paper) based on a conventional FPGA architecture (which we will refer to as the standard VPR architecture in the remainder of this paper) described in [4]. In our architecture, a logic block is called a super-cluster which consists of M conventional VPR clusters. M is called the granularity of our datapath

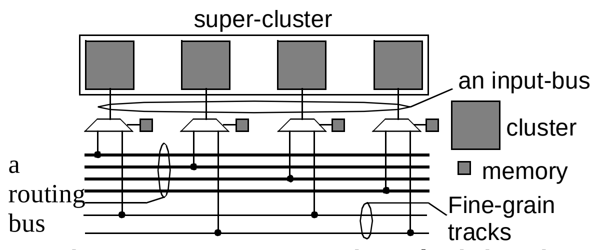

Figure 7: Input Connection Block (M=4)

FPGA architecture. For the ease of illustration, most of our figures in this paper are drawn by setting M to be four. Each VPR cluster contains N four-input Look-Up Tables (LUTs). It also contains a fully connected local routing network as shown in Figure 4. M clusters are grouped into a super-cluster using the topology shown in Figure 5.

This logic-clustering architecture is motivated by the fact that datapath circuits often consist of many identical bit-slices and these bit-slices are the source of signal buses — regularly structured connections that maps well onto the coarse-grain routing resources. Using our architecture, we implement portions of bit-slices in a datapath circuit in clusters. Then we group the clusters that implement identical portions of bit-slices together into super-clusters. By doing so, we can maximize the chance of capturing datapath buses onto inter-super-cluster connections without sacrificing the utilization of local routing network inside each cluster. Once captured, these buses can then be efficiently routed through the coarse-grain routing resources in the global routing network described later in this section.

Within each super-cluster, special connections supporting arithmetic carries are provided. The details of the carry structure is shown in Figure 6. When in the carry mode, two neighboring LUTs in a cluster are grouped together. One LUT implements the carry generation logic and the other implements the sum generation logic. The number of super-cluster inputs is equal to the total number of cluster inputs in a given super-cluster; and the number of super-cluster outputs is equal to the total number of cluster outputs in a given super-cluster. Each cluster input is directly connected to a super-cluster input; and each cluster output is directly connected to a super-cluster output.

The global routing resources of the datapath FPGA consist of both coarse-grain routing resources with a granularity value of M and conventional fine-grain routing resources. Each routing channel contains a fixed number of coarse-grain routing tracks and a fixed number of fine-grain routing tracks.

An input connection block is shown in Figure 7. The input connection block connects super-cluster inputs to both fine-grain routing tracks and coarse-grain routing tracks. Super-cluster inputs to fine-grain routing track connections are similar to conventional cluster inputs to routing track connections. Each input pin can be connected to a fixed number, Fc_if, of fine-grain routing tracks.

For each super-cluster, we group corresponding inputs of the M clusters together to form M-bit wide buses. We call these buses input-buses. Since the granularity of the coarse-grain routing is M, the number of coarse-grain tracks is always an even multiple of M; and coarse-grain routing tracks are grouped into M-bit wide buses. We call these buses routing-buses. Each input-bus is connected to a fixed number, Fc_ic, of routing-buses. As shown in Figure 7, when connecting an input-bus to a routing-bus, we connect the corresponding bits of each bus together.

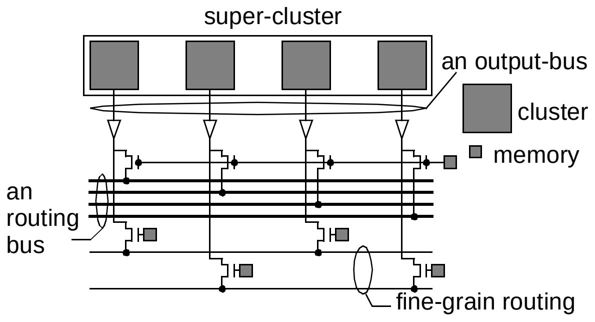

Figure 8: Output Connection Block (M=4)

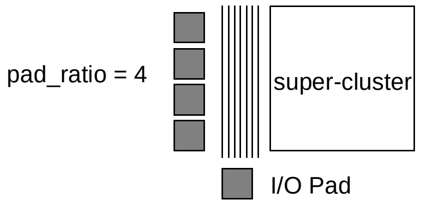

Figure 9: Pad Ratio

An output connection block is shown in Figure 8. It connects super-cluster outputs to both fine-grain routing tracks and coarse-grain routing tracks. Super-cluster outputs to fine-grain routing track connections are similar to VPR cluster outputs to routing track connections. Each output pin can be connected to a fixed number, Fc_of, of fine-grain routing tracks.

In each super-cluster, we group corresponding outputs of the M clusters together to form M-bit wide buses. We call these buses output-buses. Each output-bus is connected to a fixed number, Fc_oc, of routing-buses. As shown in Figure 8, when connecting an output-bus to a coarse-grain routing bus, we connect the corresponding bits of each bus together. The programmable switches in each bus-to-bus connection share a single set of configuration memory.

As in conventional architectures, we assume all I/O pads reside on the boundary of our datapath FPGA. We also assume uniformly distributed I/O pads as shown in Figure 9. Here the pad_ratio is the number of I/O pads that can be fitted in the pitch of a square super-cluster.

As in the conventional FPGA architecture, we assume each I/O pad to be bi-directional — each pad containing one input pin and one output pin. Both input pin and output pin have the same connection patterns to the fine-grain and coarse-grain routing tracks. Each pad input or output pin can be connected to a fixed number, Fc_pf, of fine-grain routing tracks.

We group M I/O pad input pins or M I/O pad output pins together to form M-bit wide buses. We call the buses formed by input pins pad-input buses and the buses formed by the output pins pad-output buses. Each pad-input or pad-output bus is connected to a fixed number, Fc_pc, of coarse-grain routing buses. Similar to the input-bus to routing-bus connections and output-bus to routing-bus connections, when connecting a pad-input or a pad-output bus to a routing-bus, we connect the corresponding bits of each bus together. For pad-output buses, the programmable switches in each bus-to-bus connection share a single set of configuration memory.

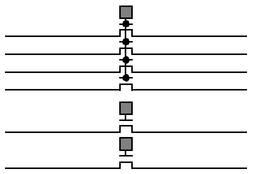

Figure 10: Switch Block (M=4)

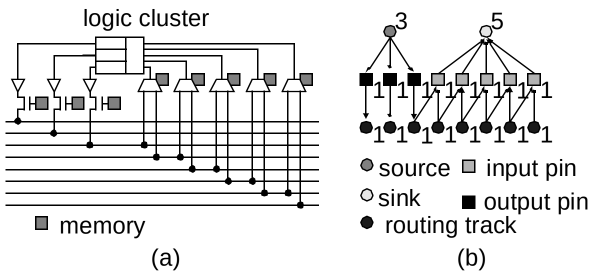

Figure 11: Routing Resource Graph

A switch block which resides at the intersection of all horizontal and vertical channels is shown in Figure 10. It contains both fine-grain routing track to fine-grain routing track connections and coarse-grain routing track to coarse-grain routing track connections. We assume that there are no connections between fine-grain routing tracks and coarse-grain routing tracks. We use the disjoint topology for fine-grain routing track connections since this is shown to be one of the most efficient topology for the conventional architecture. Each fine-grain routing track can be connected to Fs_f of other fine-grain routing tracks. We also assume the disjoint topology for connections between the coarse-grain routing-buses. Each routing-bus can be connected to Fs_c of other routing-buses. As shown in Figure 10, when connecting two routing-buses together, we connect the corresponding bits of each bus together. The programmable switches in each bus-to-bus connection share a single set of configuration memory.

3. Coarse-Grain Routing Algorithm

We designed our routing algorithm based on the VPR timing driven router [4]. The original VPR routing algorithm only models fine-grain routing resources. Unmodified, the algorithm cannot be efficiently used to route FPGAs containing coarse-grain routing resources. We enhanced the VPR algorithm in three major areas to give it the capability of handling coarse-grain routing resources. These three enhancements are the modification of the routing resource graph to support coarse-grain routing resources, the identification of buses in the input netlist, and the new cost functions designed for coarse-grain routing. We will briefly describe the original VPR algorithm in Section 3.1 The three enhancements are described in Section 3.2, Section 3.3, and Section 3.4, respectively.

```
function route_net(i)
    for each sink of net i, j
        call expand_neighbours(i,j)
    end for
end function
```

Figure 13: route_net Function

3.1 Basic VPR Timing Driven Routing Algorithm

In the VPR router, routing resources are represented by a routing resource graph, which consists of nodes and edges. Each node in the graph represents a source, a sink, a logic-block input pin, a logic-block output pin, an input pin of an I/O pad, an output pin of an I/O pad, or a routing track. (Note: In the standard VPR architecture, a logic block represents a cluster. In our datapath FPGA architecture, a logic block represents a super-cluster.) Each edge in the graph represents a programmable routing switch.

An example of the routing resource graph and its corresponding routing resources is shown in Figure 11. A standard VPR cluster and one of its neighboring routing channels is shown in Figure 11(a) and a routing resource graph representing the cluster and the routing channel is shown in Figure 11(b). This cluster contains three 4-LUTs. These LUTs are connected by a fully connected local routing network. The cluster also contains three output pins — one for each LUT — and five input pins. There are eight routing tracks in the routing channel. Each output pin is connected to one routing track, while each input pin is connected to two routing tracks through programmable switches.

Each node of the routing resource graph is associated with a capacity value, which is labeled on each node in Figure 11(b). This capacity value represents the maximum number of times that a routing resource, represented by a node, can be used in the final routing solution. For each pin or each routing track, this capacity value is one since each of these routing resources can only be legally used once. For each source or sink node, this capacity value is a positive number whose value depends on the equivalency of logic block output or input pins [4]. Since all three LUTs, in the figure, are connected by a fully connected local routing network, they are logically equivalent — only one source is needed to represent all three LUTs and the capacity value of the source is three. Also because of the fully connected local routing network, the five cluster input pins are logically equivalent — only one sink is needed for all five input pins and the capacity value of the sink is five.

The VPR router takes a routing resource graph representing a target FPGA and a netlist of multi-terminal nets (nets with a single source and multiple sinks) as its inputs. This netlist represents a circuit of interconnected logic blocks. The router then finds a feasible routing solution in the routing resource graph for each multi-terminal net. Here a routing solution is defined to be a collection of interconnected nodes and edges; furthermore, the collection must contain the source node and all the sink nodes of a multi-terminal net. To be feasible, the occupancy value — the total number of times that a node appears in all routing solutions — of each node in the routing solution must be less than or equal to its capacity value. Otherwise, the routing solution is called infeasible. A net is said to be routed, once a routing solution (either feasible or infeasible) containing the source node and all the sink nodes of the net is found.

$$\text{congestion\_cost}(n) = b(n) \times p(n) \times h(n) \tag{1}$$

```
loop until maximum iteration is reached
    for each node n
        calculate congestion_cost(n)
    end for
    rip up all previously routed net
    for each net i
        for each sink of net i, j
            calculate criticality(i,j)
        end for
        call  route_net (i)
    end for
end loop
```

Figure 12: VPR Routing Algorithm

```
function expand_neighbours (i,j)
    push i onto a priority stack
    loop
        pop node, n, off the stack
        for all nodes, m, connected downstream of node, n
            if (m == j) then
                exit function
            else
                calculate expansion_cost(m)
                push m onto the priority stack using expansion_cost
            end if
        end for
    end loop
end function
```

Figure 14: expand_neighbours Function

Table 1: b(n) Values for Each Type of Routing Resource

Routing Resource	b(n)
Routing track	1
Cluster Output pin	1
Cluster input pin	0.95
Source	1
Sink	0

$$p(n) = 1 + \max(0, [\text{occupancy}(n) + 1 - \text{capacity}(n)] \times \text{pfac}) \tag{2}$$

$$h(n)^i = \begin{cases} 1 & i = 1 \\ h(n)^{i-1} + \max(0, \\ [\text{occupancy}(n) - \text{capacity}(n)] \times \text{hfac}) & i > 1 \end{cases} \tag{3}$$

The VPR router solves the routing problem using an algorithm similar to the Pathfinder routing algorithm [4], which is shown in Figure 12, Figure 13, and Figure 14. This routing process consists of a series of iterations. At the start of each iteration, VPR first assigns a congestion cost to each node, n, in the routing resource graph. As shown in Equation 1, this congestion cost is a product of the base cost, b(n), the current congestion cost, p(n), and the historic congestion cost, h(n). The base cost, b(n), is a function of the routing resource type. Different routing resources are assigned different base cost values as shown in Table 1.

The current congestion cost, p(n), is defined to be a function of the difference between the current occupancy of the node and the capacity of the node as shown in Equation 2. The scaling factor, pfac, in Equation 2 is called the routing schedule of the router. The initial value of pfac is a small (< 0.5), so during early iterations, the current congestion of the node is a small part of the total cost of the node. This allows each node to be used more than its capacity allows. The value of pfac increases by a factor of 1.5 to 2 during each routing iteration. So during the latter iterations, the current congestion cost becomes a significant factor in determining the total cost of a node. Consequently, at latter routing iterations, once a node reaches its full capacity, it cannot be used in the routing solution of other nets.

The historic congestion cost, h(n), is an accumulation of the past congestion values; and it is defined by Equation 3. For the first iteration, the historic congestion is set to be one. For each subsequent iteration, the difference between occupancy and capacity is scaled by a constant value, hfac, and is added to the total historic congestion. The usual value of hfac is between 0.2 and 1.

$$\text{criticality}(i,j) = \max\left(\left[1 - \frac{\text{slack}(i,j)}{\text{Dmax}}\right], 0\right) \tag{4}$$

After assigning the congestion cost, VPR rips up all previously routed nets. This resets the occupancy value of all nodes in the routing resource graph to be zero. Then VPR routes each net using the maze expansion algorithm. For each two-terminal connection on the net, the criticality of the connection is first calculated using Equation 4. Here i represents the source of a two-terminal connection and j represents the sink of the same two-terminal connection. As shown by Equation 4, the criticality is a value between zero and one inclusively. It is a function of the slack of the connection and Dmax, the critical path delay of the circuit. When the total delay of a net is close to the maximum delay of the circuit, the net has a very small slack. Consequently, its criticality is high. When the total delay of a net is much smaller than the maximum delay of the circuit, the net has a very large slack. Consequently its criticality is low. Both the slack and Dmax values are calculated based on the Elmore delay model using the delay values obtained from the previous routing iteration.

$$\begin{gathered}\text{expansion\_cost} = [1 - \text{criticality}(i,j)] \times \\ \text{congestion\_cost}(n) + \text{criticality}(i,j) \times \text{delay}(n) + \\ \text{future\_expansion\_cost}(n)\end{gathered} \tag{5}$$

The criticality value is used in the calculation of the expansion cost to control the amount of effort that a router should use to minimize delay over minimizing congestion. When routing a highly critical net, the router should concentrate on minimizing delay. When routing a less critical net, the router should concentrate on minimizing congestion instead. Equation 5 is used to calculate the expansion cost. The equation has three terms. The first term, congestion_cost(n), represents the total congestion cost of all the node on the current expansion. The second term, delay(n), represents the delay of the net up to the expansion node. These two terms are scaled by the criticality of the current net. If the net is very critical then the delay cost dominates. If the net is not critical then the congestion cost dominates. The third term represents the estimated future expansion cost. Its calculation is described in detail in [4]. Here we estimate the remaining number of expansions that we need to complete the routing of the entire net. The total cost of these estimated expansions are then calculated and used as the future expansion cost.

3.2 The Routing Resource Graph for Datapath FPGA

Since each super-cluster in our datapath FPGA contains M clusters, each with a fully connected local routing network, we model a super-cluster using M source nodes and M sink nodes — one source and one sink for each cluster. Each source is connected to all the output pins of its cluster and the capacity of a source is equal to the total number of LUTs in a cluster.

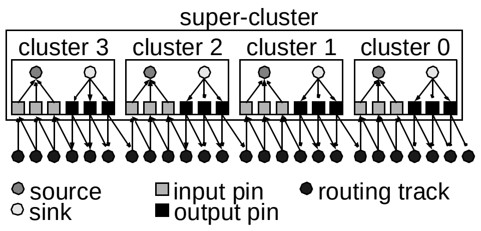

Figure 15: Super-cluster Representation

Each sink is connected to all the input pins of its cluster and the capacity of a sink is equal to the total number of cluster input pins. Figure 15 shows an example of a routing resource graph representing a super-cluster containing four clusters.

Nodes in a routing resource graph are grouped into node buses. A node bus can be M sources of a super-cluster, M sinks of a super-cluster, M cluster inputs in an input-bus, M cluster outputs in an output-bus, M routing tracks in a routing bus, M pad input pins in a pad-input bus or M pad output pins in a pad-output bus.

Edges in a routing resource graph are similarly grouped into edge-buses. Each edge-bus is associated with a flag indicating if the M switches in the bus share a single set of configuration memory. When configuration memory are shared, all M switches must be turned on and off at the same time. To ensure correct routing, we must increment or decrement the occupancy values of all the M routing tracks or M input pins connected downstream to these switches simultaneously.

3.3 Identifying Net-buses in the Input Netlist

Each input netlist is a set of multi-terminal nets connecting a source to multiple sinks. We search through this input netlist and identify net-buses. A net-bus is defined to be a collection of M nets that have the following three properties:

1. All M nets must have sources from the same super-cluster and these M sources must be unique.
2. All M nets must contain the same number of sinks.
3. Sinks from these M nets can be grouped into groups of M with each group containing a sink from a unique net. Furthermore, all M sinks in any group must be in the same super-cluster and all these M sinks must be unique. (Each group of sinks is called a sink-bus.)

Nets that do not belong to any net-buses are called fine-grain nets.

3.4 Cost Function Design for Coarse-Grain Routing Resources

Our router not only have to balance routing delay with routing congestion, it also has to balance the use of coarse-grain routing resources with fine-grain routing resources. Besides the traditional delay and congestion considerations, a balanced use of coarse and fine grain routing resources is influenced by following four major factors:

1. the number of net-buses in an input netlist
2. the available number of coarse-grain routing tracks
3. the number of fine-grain nets in an input netlist
4. the available number of fine-grain routing tracks

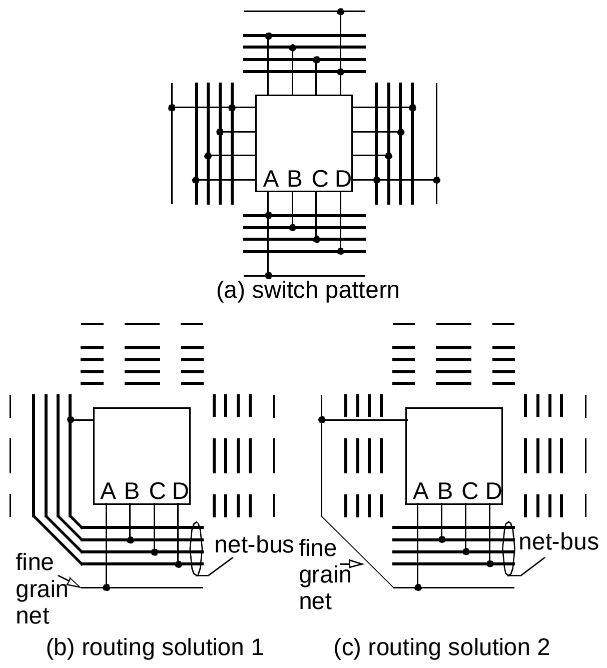

Figure 16: Competition for Resources between Fine and Coarse-grain Nets

For example, when there is a large number of coarse-grain tracks available. Routing fine-grain nets on coarse-grain tracks might be the only way to achieve a legal routing solution. On the other hand, when there are a large number of fine-grain routing tracks, it might be beneficial to route a large number of coarse-grain nets through fine-grain routing tracks.

Further complicating the issue is the fact that several routing resources can have dual personalities — they can be considered either as a part of the fine-grain routing resources or an extension of the coarse-grain routing resources. For example, a super-cluster input pin can be considered either as an individual input pin or a part of a super-cluster input-bus. When accepting an input signal from a fine-grain routing track, the pin acts as an individual input pin — it behaves like fine-grain routing resources. When accepting an input signal from a coarse-grain routing track carrying a net-bus, the pin acts in concert with three other input pins in its input-bus. This time, it behaves like coarse-grain routing resources. Other routing resources that behave similarly include super-cluster output pins, I/O pad input pins, I/O pad output pins, sources, and sinks.

Because of this dual personality, fine-grain routing often competes with coarse-grain routing for resources. Figure 16(a) shows one tile of a datapath FPGA and the switch pattern of its input and output connection blocks. We assume disjoint switch blocks for this example. As shown in Figure 16(b) and Figure 16(c), both a fine-grain net and a net in a net-bus can use pin A to get into a super-cluster. In Figure 16(b), the fine-grain net uses pin A. Avoiding to create a congestion, only pin B, C, and D can be used by the net-bus. To get the entire net-bus into the super-cluster, one more routing-bus segment has to be used. In Figure 16(c), pin A is used by the net in the net-bus instead, the entire net-bus can get into the super-cluster

```
loop until maximum iteration is reached
    for each node n
        calculate congestion_cost(n)
    end for
    rip up all previously routed net
    for each net i
        for each sink of net i, j
            calculate criticality(i,j)
        end for
    end for
    for each bus b
        call route_net (b, null)
    end for
    for each net, i, with unrouted sinks
        call route_net(null, i)
    end for
end loop
```

Figure 17: Routing Algorithm for Datapath FPGA

```
function route_net(i, b)
    if (i == NULL)
        is_bus = false
        for each sink j of i
            calculate criticality(i,j)
            call expand_neighbours(i,j,is_bus)
    else if (b == NULL)
        is_bus = true
        for each sink bus c of b
            for (k=0; k<with of the bus; k++)
                i = kth bit of b
                j = kth bit of c
                calculate criticality(i,j)
            end for
            i = first bit of b
            j = first bit of c
            call expand_neighbours(i,j,is_bus)
    end if
end function
```

Figure 18: route_net Function

```
function expand_neighbours (i,j,is_bus)
    push i onto the priority stack
    loop
        pop node, n, off the stack
        for all nodes, m, connected downstream of node, n
            if (m == j) then
                exit function
            else
                calculate expansion cost based on expansion topology
            end if
        end for
        push m onto the priority stack using expansion_cost
    end loop
end function
```

Figure 19: expand_neighbours Function

through pin A, B, C, and D. However, one extra fine-grain routing segment has to be used to get the fine-grain net into the super-cluster without causing any congestion.

The overall routing algorithm for datapath FPGA is shown in Figure 17, Figure 18, and Figure 19. To balance the use of fine and coarse-grain routing resources, our router routes net-buses simultaneously through both fine and coarse-grain routing tracks. We then choose the routing with the least cost. To be more specific, during each routing iteration, our router first routes net-buses. Each net-bus is routed as a group. Our goal is to route as much net-buses through the coarse-grain routing tracks as possible. While routing the net-bus, we also route the first bit of the bus through the fine-grain routing tracks. If the

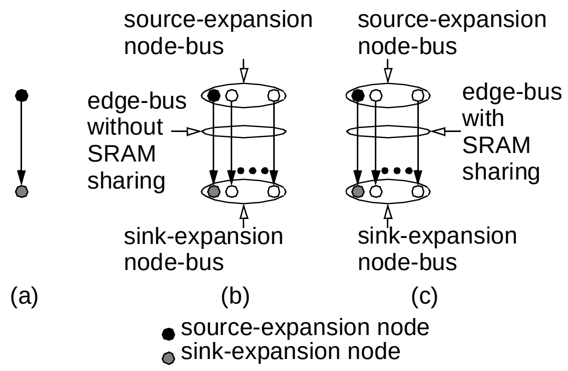
source-expansion node-bus
source-expansion node-bus
edge-bus without SRAM sharing
edge-bus with SRAM sharing
sink-expansion node-bus
sink-expansion node-bus
(a)
(b)
(c)
source-expansion node
sink-expansion node

Figure 20: Expansion Topology

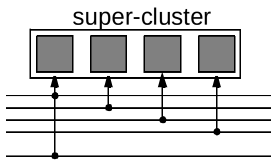

Figure 21: Double Connection in one bus bit

cost of routing the first bit through fine-grain routing tracks is less than the cost of routing the net-bus through coarse-grain routing tracks, we route all bits in the bus through fine-grain routing tracks. If using the coarse-grain routing tracks incurs lower cost, coarse-grain routing tracks are then used. To encourage net-buses to use coarse-grain routing tracks, we penalize the cost of routing the first bit of the net-bus through fine-grain routing tracks by a constant multiplication factor. We found that a factor of 20 works well for our benchmarks.

Nets that do not belong to any buses are routed individually. Here again, we route each fine-grain net through both fine-grain and coarse-grain routing tracks. We compare the cost of using fine-grain and coarse-grain routing resources and choose the option which incurs the lower cost. In the next a few paragraphs, we discuss how we calculate the expansion cost for routing net-buses.

The expansion cost is calculated based on the topology of the expansion. There are three different expansion topologies as shown Figure 20. Here the node that is the source of the expansion is labeled source-expansion node. The node that we expand into is labeled sink-expansion node. The first expansion topology is illustrated Figure 20(a). In the figure, the source-expansion node and the sink-expansion node is connected by an edge that does not belongs to an edge-bus. In this topology, Equation 5 is used to calculate the expansion cost. When routing a net-bus through this topology, only the first bit of the net-bus is routed.

The second expansion topology is shown in Figure 20(b). Here the source-expansion node and the sink-expansion node are connected by an edge that belongs to an edge-bus and the switches in the edge-bus does not share a single set of controlling SRAM. We call the node-bus containing the source-expansion node the source-expansion node-bus and the node-bus containing the sink-expansion node the sink-expansion node-bus. When only routing a fine-grain net through this expansion topology, we use Equation 5 to calculate the expansion cost. When routing a net-bus through this expansion topology, we use Equation 5 to calculate expansion cost for each node. The maximum cost is used as the expansion cost. It is important to point out that, due to the dual personality of routing resources, nodes in a node-bus do not necessarily have the same congestion cost at all time. An example is shown in Figure 21. Here the nodes in the node-buses represent four input pins. All four pins are connected to a net-bus through a coarse-grain routing bus. Pin 0, however, has one more connection. It is also connected to a fine-grain net through a fine-grain routing track. The occupancy value of pin 0 is 2 while the occupancy values of pin 1, 2, and 3 are 1. Since the occupancy values are different, the expansion costs are also different.

The third expansion topology is shown in Figure 20(c). Here the source-expansion node and the sink-expansion node are also connected by an edge that belongs to an edge-bus. The switches in the edge-bus, however, shares a single set of controlling SRAM. When only routing a fine-grain net through this expansion topology, we find the maximum congestion cost over all nodes in the sink-expansion node-bus. A modified Equation 5 is then used to calculate expansion cost where the congestion_cost(n) term is substituted by this maximum congestion cost. If we have to route a net-bus through the third expansion topology, we use Equation 5 to calculate the expansion cost for each node in the sink-expansion node-bus. We then use the maximum expansion cost as the expansion cost.

4. Experimental Methodology

We performed a set of experiments to compare the area efficiency of the datapath architecture against the standard VPR architecture. Our experiments also attempt to address the following two questions:

1. What is the effect of varying track length on the area of the datapath architecture?
2. What is the effect of varying the number of coarse-grain tracks on the area of the datapath architecture?

We define the track length or the logical track length to be the number of logic clusters that a routing track expands. The physical track length is defined to be the physical length of a routing track.

Our experiments assume that each routing channel contains two types of routing tracks — conventional fine-grain routing tracks of logic length L and coarse-grain routing tracks also of logic length L. We also assume that each type of routing tracks is uniformly distributed across the FPGA. For all of our experiments, we set the granularity of the datapath architecture, M, to be four — each super-cluster contains four VPR clusters and coarse-grain routing tracks are grouped into four-bit wide buses. This granularity was shown to be one of the most efficient by the study of [6]. It is also used by the architecture described in [13].

As discussed in Section 2, the datapath architecture uses a disjoint switch block. We set the Fs_f and Fs_c values to be three for all of our experiments. An Fs_f or Fs_c value of three means that at each switch point a track is connected to three neighboring tracks. As shown in Figure 22, at the end of each track six switches are needed to connect a given track to its neighboring tracks. At the middle of each track, only one switch is needed to connect the track to its neighboring track. We also assume that all switches in our switch blocks are buffered switches.

Figure 23 shows the CAD flow of our experiments. The 15 benchmark circuits are from the Pico-Java processor from SUN Microsystems. Each benchmark circuit is from a datapath component of the Pico-Java and the benchmark set covers all major datapath components of the processor. These circuits are synthesized into LUTs using a datapath oriented synthesis

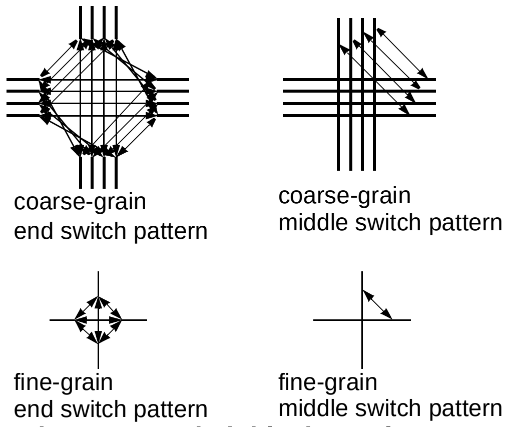

Figure 22: Switch block topology

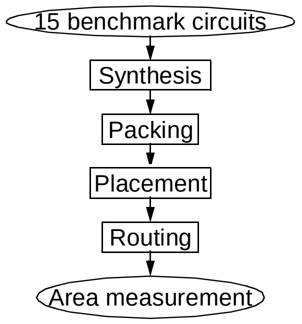

Figure 23: CAD Flow

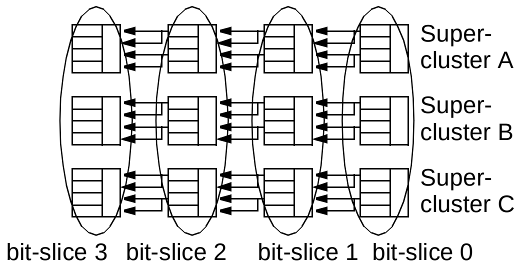

Figure 24: Packing

process described in [16]. This synthesis process preserves the regularity of datapath circuits while attempting to minimize area. The output of the synthesis tool consists of a netlist of bit-slices and LUTs. The bit-slices describe the regular datapath logic while the LUTs describe the random logic. The bit-slices themselves are netlists of LUTs. Each bit-slice is instantiated multiple times to form regular datapath components.

The synthesized circuits are then packed into super-clusters using our datapath-oriented packing tool, which is based on the T-VPACK packing algorithm. Our packing tool is timing-driven. It tries to pack every four adjacent bit-slices into a series of super-clusters. As shown in Figure 24, portions of a bit-slice are mapped into a unique cluster for each super-cluster. The packer also utilizes the super-cluster level carry connections to minimize the delay of carry chains. The packed circuits are then placed using a modified VPR placement algorithm. Our placer uses simulated annealing to find the best placement for each super-cluster. To achieve the best possible placement solutions, our placer also moves individual clusters if these clusters do not contain any datapath components (clusters containing pure random-logic). The placed circuits are then routed using the router described in Section 3

To find the effect of varying track length on the area of the datapath architecture, we varied the logic track length to be 1, 2, 4, 8, and 16. We also considered the cluster size of 2, 3, 4, 6, 8, and 10 for each track length. For all experiments, we set the percentage of coarse-grain tracks at close to 45%, since the study in [16] showed that 40%-50% of two terminal nets in highly regular datapath circuits can be grouped into four-bit wide two-terminal buses. The physical length of a wire segment increases as the cluster size is increased even when the logic length of the routing track is kept constant. We account for this increase in physical length by linearly scaling routing track buffers with the square root of the super-cluster area. This is shown to generate good area and delay results in [4]. For the architectures with the same cluster size but different logic segment length, we keep the track buffer size to be constant. This is also shown to generate good area and delay results in [4].

To find the effect of varying the number of coarse-grain tracks on the area of the datapath architecture, we performed routing using several variants of the datapath architecture, each with a different number of coarse-grain tracks. For all of our experiments, we fixed the logical track length to be two for both coarse-grain and fine-grain tracks. We also fix the cluster size, N, to be four. A track length of two and a cluster size of four are shown to generate the best area result by the experiment addressing question 1. For each architecture, we fix the total number of coarse-grain tracks that can be used and let the router search for the minimum number of fine-grain tracks that is needed to complete the routing. The number of fixed coarse-grain routing buses that we considered for each benchmark circuit is from 0 to 20 inclusively.

Finally to compare the area efficiency of a standard VPR architecture with our datapath-oriented FPGA architecture, we set the cluster size, N, to be four for both the standard VPR architecture and our datapath-oriented FPGA. Again we use a fully buffered global routing architecture for both the standard VPR architecture and our datapath-oriented FPGA. We varied several design parameters including L (the logic track length), Fc_input (the number of tracks that a cluster input can be connected to), Fc_pad (the number of tracks that a pad I/O pin can be connected to), and Fc_output (the number of tracks that a cluster output can be connected to) to find a set of design parameters that generate the best area result for the standard VPR architecture.

For our datapath-oriented architecture, we set the logic track length, L, to be two and the number of coarse-grain tracks to be zero. We varied the design parameters Fc_if, Fc_pf, and Fc_of to find a combination of these three parameters that is the most area efficient. We then assume the same set of Fc_if, Fc_pf, and Fc_of will generate the most area efficient results for any percentage of coarse-grain tracks, when Fc_ic, Fc_pc, and Fc_oc are set to be equal to Fc_if, Fc_pf, and Fc_of, respectively. Fixing Fc_if, Fc_pf, Fc_of, Fc_ic, Fc_pc, and Fc_oc, we varied the number of coarse-grain tracks to find the most area efficient number of coarse-grain routing tracks for the datapath architecture.

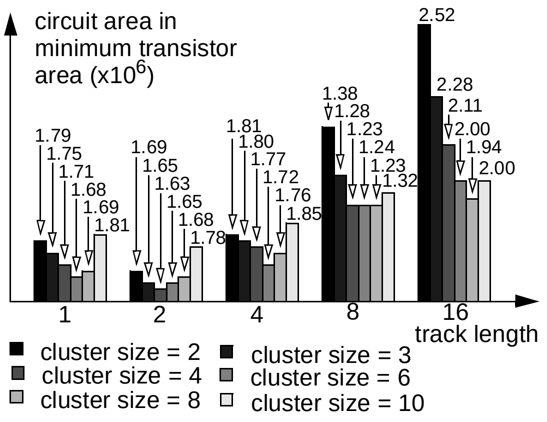

Figure 25: area vs. logical track length

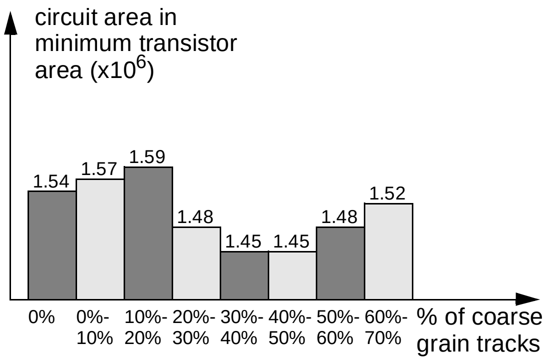

Figure 26: % of Coarse-Grain Tracks vs. Area

5. Experimental Results

Figure 25 shows circuit implementation area vs. logical track length for the datapath architecture. The x-axis represents the logical track length and the y-axis represents the circuit implementation area. We measured the circuit implementation area in terms of minimum transistor count as described in [4]. As shown by the figure, for each track length, we consider various cluster sizes of 2, 3, 4, 6, 8, and 10. The average area across fifteen benchmark circuits is then plotted for each track length and cluster size combination. For track length of one and four, cluster size of six has the lowest implementation area. For track length of two, cluster size of four has the lowest implementation area. For track length of eight, cluster size of four and eight has the lowest implementation area. Finally for track length of sixteen, cluster size of eight has the lowest implementation area. For all cluster sizes the best area result is achieved when the logical track length is two. Furthermore, the best datapath architecture has a track length of two and a cluster size of four.

Figure 26 shows the total area vs. the percentage of total tracks that are coarse-grain in the datapath FPGA routing. We also measured the area in terms of minimum transistor count as described in [4]. For each benchmark circuit, we collected the area results from a series of datapath architectures as described in Section 4. We then classify these architectures into

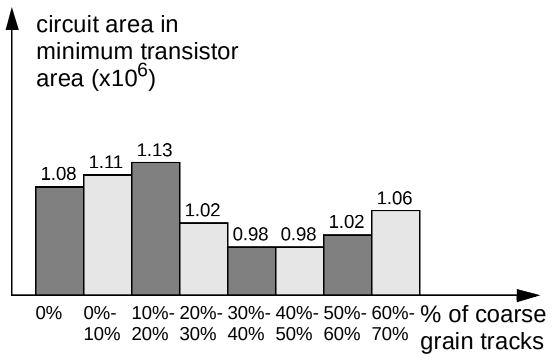

Figure 27: % of Coarse-Grain Tracks vs. Routing Area

eight groups based on the percentage of total tracks that are coarse-grain. The percentile ranges are (0%, 0%], (0%, 10%], (10%, 20%], (20%, 30%], (30%, 40%], (40%, 50%], (50%, 60%], or (60%, 70%].

Within each region, we first obtain the minimum area obtainable by each circuit. We then average these minimum area values across 15 benchmark circuits. The average area is then plotted against each percentile range as shown in Figure 26. From the graph, we can see that, initially, as the number of coarse-grain tracks increases, the total area required to implement the benchmark set also increases. This area increase continues until the number of coarse-grain tracks reaches 20% of the total number of tracks. Then the area decreases and reaches its minimum when the number of coarse-grain tracks is between 30% to 50% of the total track count. Then total area increases again as the percentage of coarse-grain tracks continues to increase.

As we start to add coarse-grain tracks into our routing fabric, we are differentiating our routing resources into two types. This differentiation reduces the routing flexibility and accounts for the initial increase in total area. As the number of coarse-grain tracks is increased to the 20% range, the benefit of coarse-grain tracks starts to outweigh the inflexibility in routing. As the result, the total area required decreases until it reaches its minimum when coarse-grain tracks account for between 30% to 50% of the total tracks. When we further increase the number of coarse-grain tracks, the number of coarse-grain tracks provided by the architecture starts to exceed the number of coarse-grain tracks required by the circuits. The router then starts to excessively use coarse-grain tracks for fine-grain routing. This reduces the efficiency of the datapath architecture and is responsible for the increase in area when more than 50% of the tracks are coarse-grain.

Overall, the best area is achieved when coarse-grain tracks account for 30% to 50% of the total tracks. At the 40% to 50% region, our benchmark circuits uses 6% less area as compared with architectures with no coarse-grain tracks. This result confirms with the observation in [16] that around 48% of two terminal nets in our datapath benchmarks can be grouped into 4-bit wide buses.

Since all of our area savings come from routing, it is instructive to just look at the routing area alone. This is shown in Figure 27. The graph shows that when the coarse-grain tracks account for 40% to 50% of the total number of tracks, the routing area is 9% smaller than architectures with no coarse-grain routing tracks.

Finally, Figure 28 shows the same results presented in Figure 26 with area normalized against the best standard VPR architecture. All coarse-grain architectures performed better than the best standard VPR architecture. When coarse-grain

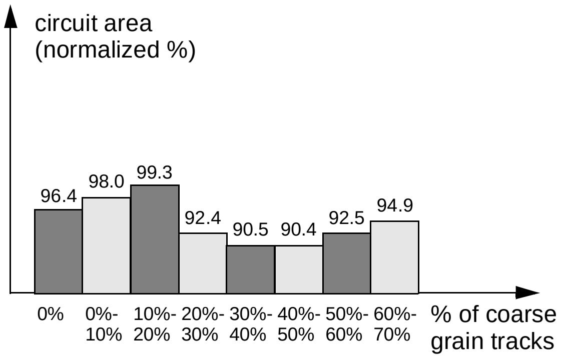

Figure 28: % of Coarse-Grain Tracks vs. Normalized Area

routing tracks consist of 40% to 50% of the total routing tracks, the total area of the datapath architecture is 9.6% smaller than the best standard VPR architecture.

6. Conclusion and Future Work

In this paper we proposed a datapath-oriented FPGA architecture with coarse-grain routing tracks. We also proposed a routing algorithm for this FPGA. We use the new router along with the datapath-oriented synthesis, packing, and placement tools to investigate the effects of coarse-grain routing tracks on FPGA area for highly regular datapath circuits.

We found that, for granularity of four, segment length of two gives the best area results for cluster size of 2, 3, 4, 6, 8, and 10. In order to achieve the best area results, 40% to 50% of the total routing tracks should be coarse-grain. Furthermore, for cluster size of four, the best datapath architecture is 9.6% smaller than the best standard VPR architecture.

Datapath-oriented FPGA architectures is an interesting and largely un-exploited field in FPGA design. Much work needs to be done in the area of benchmarks, FPGA architectures, and CAD tools. First of all, more and larger benchmarks are needed. These benchmarks should conformed to a set of standards in describing regularity, so the CAD tools can easily preserve and exploit the regularity information. Secondly, more work needs to be done in architectural design. This work only exploited the benefits of sharing SRAM amount global routing tracks. Newer architectures can exploit the idea of sharing configuration SRAM amount LUTs and several cluster level local routing networks. Current work also isolates the fine-grain routing tracks from the coarse-grain routing tracks. More advanced switch block architectures that allow signals to be routed from fine-grain routing tracks to coarse-grain routing tracks and vice versa would be interesting to exploit. An immediate follow up to this work would be to fully exploit the effect of varying M using a large set of benchmarks. Finally, many work needs to be done in terms of CAD tools. More intelligent synthesis, packing, placement, and routing tools would further enhance the area efficiency of datapath FPGAs.

References

[1] Altera Datasheet, Altera, 2002.
[2] Pico-Java Processor Design Documentation, Sun Microsystems Inc., 1999.

[3] Xilinx Datasheet, Xilinx, 2002.

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

[5] S. Brown, R. Francis, J. Rose, Z. Vranesic, **Field-Programmable Gate Arrays, Kluwer Academic Publishers**, 1992.

[6] D. Cherepacha, D. Lewis, “DP-FPGA: an FPGA architecture optimized for datapaths”, Proceedings of Ninth International Conference on VLSI Design, Pages 329-343, 1996.

[7] T. J. Callahan, P. Chong, A. DeHon, J. Wawrzynek, “Fast module mapping and placement for datapaths in FPGAs”, Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays, Pages 123–132, 1998.

[8] M. R. Corazao, M. A. Khalaf, M. Potkonjak, J. M. Rabaey, “Performance optimization using template mapping for datapath-intensive high-level synthesis”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Pages 877–888, August 1996.

[9] A. Koch, “Structured design implementation — a strategy for implementing regular datapaths on FPGAs”, Proceedings of the 1996 ACM Fourth International Symposium on Field Programmable Gate Arrays, Pages 151–157, 1996.

[10] A. Koch, “Module compaction in FPGA-based regular datapaths”, Proceedings of the 33rd Design Automation Conference, Pages 471–476, 1996.

[11] T. Kutzschebauch, L. Stok, “Regularity driven logic synthesis”, Proceedings of IEEE/ACM International Conference on Computer Aided Design, Pages 439–446, 2000.

[12] T. Kutzschebauch, “Efficient logic optimization using regularity extraction”, Proceedings of 2000 International Conference on Computer Design, Pages 487–493, 2000.

[13] A. Marshall, J. Vuillemin, B. Hutchings, “A reconfigurable arithmetic array for multimedia applications”, Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays, Pages 135–143, February 1999.

[14] A. R. Naseer, M. Balakrishnan, A. Kumar, “FAST: FPGA targeted RTL structure synthesis technique”, Proceedings of the Seventh International Conference on VLSI Design, Pages 21–24, 1994.

[15] A. R. Naseer, M. Balakrishnan, A. Kumar, “Direct mapping of RTL structures onto LUT-based FPGAs”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Pages 624–631, July 1998.

[16] A. Ye, J. Rose, D. Lewis, “Synthesizing datapath circuits for FPGAs with emphasis on area minimization”, IEEE International Conference on Field-Programmable Technology, Pages 219–227, December 2002.