Mapping Arbitrary Logic Functions into Synchronous Embedded Memories For Area Reduction on FPGAs

Gordon R. Chiu, Deshanand P. Singh, Valavan Manohararajah, and Stephen D. Brown
Toronto Technology Center, Altera Corporation
gchiu|dsingh|vmanohar|sbrown at altera.com

ABSTRACT

This work describes a new mapping technique, *RAM-MAP*, that identifies parts of circuits that can be efficiently mapped into the *synchronous* embedded memories found on field programmable gate arrays (FPGAs). Previous techniques developed for mapping into *asynchronous* embedded memories cannot be used because modern FPGAs do not have *asynchronous* embedded memories. After technology mapping, an area-prediction cost function is used to guide the selection of logic cones to be placed in embedded memories. Extra logic is added to compensate for missing asynchronous functionality on the synchronous memories. Experiments conducted on Altera's Stratix device family indicate that this embedded memory mapping technique can provide an average area reduction of 6.2% and up to 32.5% on a large set of industrial designs. A small architecture change that increases the size of the FPGA fabric by 0.05% can increase the average area reduction to 14.1% and up to 59.1% on the same design set.

1. INTRODUCTION

Designs often have a large amount of timing slack. In these situations, the designers' greatest concern is using the smallest possible device that will fit their circuits as these devices are generally less costly than larger devices. We present a technique for using unused synchronous memories to implement portions of logic traditionally implemented with LUTs. This, in combination, with other area based techniques provides the designer with a tool to implement their circuit on the smallest possible programmable device.

Modern FPGAs [1, 2] provide embedded memory blocks (EMBs) to be used as on-chip memories. While there are an increasing number of applications that make use of this on-chip memory, the area devoted to EMBs will be wasted if an application does not require the memory. A potential solution to this problem is to use the EMBs as a ROM that is capable of implementing a multi-input multi-output logic function. Logic that would traditionally be mapped into logic elements is mapped into unused EMBs instead, thereby increasing the amount of logic that can be potentially packed into the FPGA. In cases where the area savings are significant, it may even be possible to select a smaller device to implement the circuit.

Techniques for mapping combinational logic clusters into embedded memories have been considered in the literature [3, 4, 5, 6]. Methods similar to those used during LUT mapping were used to identify a multi-input multi-output logic cluster which could be placed in an EMB. However, a limitation with these methods is that they cannot be used to map logic into the *synchronous* embedded memories present in modern FPGAs. A method that identifies sequential logic clusters is needed, and we present such a method in this work. In addition, we describe a technique for handling architectural restrictions of synchronous memories such as the inability to implement the asynchronous reset/preset behaviour of synchronous logic clusters. In these situations, additional circuitry is added to emulate the functionality expected of an asynchronous reset signal.

2. MEMORIES AND MEMORY MAPPING

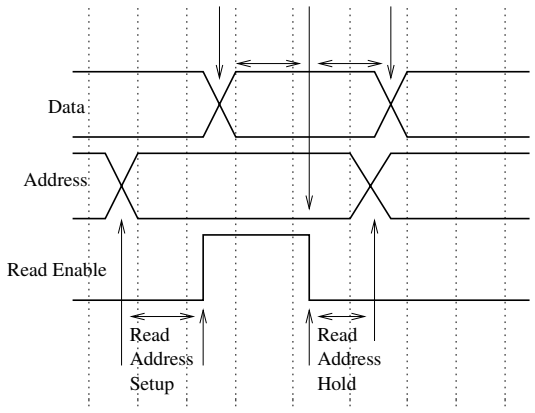

Figure 1: Asynchronous Memory Timing

Earlier use of switchable asynchronous or synchronous memories in early commercial FPGAs [7] has largely shifted

ICCAD'06, November 5–9, 2006, San Jose, CA

to fully synchronous memories [1, 2]. There are several notable differences between synchronous memories and their traditional asynchronous counterparts. Most importantly, synchronous memories enforce that all read and write operations are synchronized to a clock edge. Contrast this approach with the asynchronous read operation shown in Figure 1. The system is required to generate a *Read Enable* pulse for each data read. This signal must meet some strict timing constraints to ensure correct functionality. For example, the address lines must be stable for a certain amount of time before the leading edge of the read enable (Address Setup Time) and remain stable for a certain amount of time after the falling edge of the enable signal (Address Hold Time).

Synchronous memories avoid these complications as the designer only needs to ensure that their address, data and control signals reach the memory's registered interface before the next active clock edge. The synchronous memory block can then internally generate a read enable strobe that is guaranteed to meet the asynchronous setup and hold constraints. Note that there is still a setup and hold requirement at the registered interface; however, these constraints are guaranteed to be met by the architecture if a global clock is utilized. The synchronous design style greatly reduces the potential for errors due to misalignments in the timing of the asynchronous signals. In addition, the registered interface greatly reduces the switching activity of signals entering the internal memory circuitry and offers a potential for significant power savings. Given these advantages, synchronous memories have become a popular building block in modern FPGA fabrics [1, 2].

2.1 Limitation on Memory Mapping

As modern FPGAs do not have asynchronous embedded memories, previously described techniques in the literature [3, 4, 5, 6] are not applicable. The forced use of synchronous embedded memories further restricts the set of permissible logic cones that can be mapped into a memory: each path through the cone of logic must have one and only one register. It is not apparent how to modify the previously described techniques to enforce selection of cones that meet this constraint. The problem of mapping logic into synchronous embedded memories has not been previously considered in the literature.

3. PRELIMINARIES AND PROBLEM

We use the following definitions for this paper. A circuit is represented by a directed graph $G(V, E)$ where the vertices V represent combinational (4-LUT) or register nodes, and the edges E represent dependencies between the nodes. A node may be combinational or sequential (register with one input). Given a node v, a *cone* rooted at v is a sub-network containing v and some of its predecessors. We define a cone rooted at a set of nodes W to be a subnetwork containing each node in W along with nodes that are predecessors of at least one node in W. Given a cone C rooted at a node v, we can we define the *support set* of v to be the set of inputs, M, to the largest cone rooted at v which contains no register nodes (other than possibly v). It is noted that the largest such cone is unique for any given v. A *fanout-free cone* is a cone in which no node in the cone (except the root) drives a node not in the cone. The *maximum fanout-free cone* (*MFFC*) for a node is the fanout-free cone rooted at the node containing the largest number of nodes.

If the delay within circuit components and the delay of connections between circuit components are known, timing analysis can be used to establish the *slack* [8] of every connection. The slack of a connection is defined to be the amount of delay that can be added to the connection before it becomes *critical*. A connection is critical if the delay of a path it belongs to exceeds the path-length constraint set by the user. Timing analysis also establishes a *slack ratio* for each connection. The slack ratio is a value between 0 and 1 which indicates the relative importance of each connection to overall circuit timing. Connections that have a significant effect on circuit timing have slack ratios closer to 0 while connections that have negligible effect on circuit timing have slack ratios closer to 1.

In the absence of timing constraints, the relationship between slack ratio and slack is given by:

$$slack\_ratio(c) = \frac{slack(c) - minslack}{T_{max}}$$

where $minslack$ refers to the worst case connection slack in the circuit and T_{max} refers to the maximum delay of any register-to-register path. Connections that have a significant effect on circuit timing have slack ratios closer to 0 while connections that have negligible effect on circuit timing have slack ratios closer to 1. A precise definition of slack ratios, in the presence of multiple timing constraints, is beyond the scope of this paper. However, from an optimization perspective, slack ratios provide the most accurate criticality information as the formulation accounts for multi-cycle clocks, inverted clocks and clock skew.

One of the most powerful delay optimization techniques is *sequential retiming* [9, 10]. This technique moves registers across combinational circuit elements to reduce the length of timing-critical paths. We define *implicit retiming* to be the sequential retiming implicitly performed when restructuring a cone of logic to be compatible with an EMB which requires registers on all inputs.

We make the assumption that the maximum number of available EMBs on a chip is fixed. Assuming the chip has N available EMBs, we wish to find a mapping of the circuit into n synchronous EMBs and m logic elements such that $n \leq N$ and m is minimized.

The *width* of an EMB is defined as the width (number of bits) of each data word of the EMB. The *depth* of an EMB is defined as the number of address bits required. Thus an EMB has 2^d words, each word w bits in length.

3.1 Target Architecture Assumptions

We consider two different classifications of FPGA memory architectures for the purposes of this study. Specifically, the two classes differ in the amount of functionality available on the embedded memory block. The embedded memory blocks in the first architecture class, $arch_{no\_aclr}$, have no asynchronous clear functionality on the output of the memory block; most modern commercially available FPGAs [1, 2] fall in this category.

The second architecture class, $arch_{aclr}$, is identical to the first architecture but adds asynchronous clear func-

tionality to the outputs of the embedded memory blocks in the FPGA fabric. That is, when the asynchronous clear is active, the outputs of the embedded memory block immediately clear to a zero or preset value. This adds a small number of transistors per output of the embedded memory block and is estimated to increase the area of the embedded memory block by approximately 0.05%.

4. THE *RAM-MAP* TECHNIQUE

The *RAM-MAP* technique consists of several steps. First, a seed node is selected from the circuit. A cone of nodes is then grown from the seed node using a cost function to achieve greater area reduction. If the cost function indicates that mapping the cone will result in an area decrease, the cone of logic is replaced by an equivalent EMB and, if necessary, asynchronous fix-up logic. The process is then repeated until all possible seed nodes are exhausted or all available EMBs are used.

It is difficult to discuss the selection of sequential logic cones for mapping without an understanding of the method through which the cones are mapped into EMBs. Thus, the stages of the *RAM-MAP* technique are presented in reverse order. Given a set of nodes X which satisfy certain constraints, section 4.1 presents a method for deriving an EMB Y which is equivalent to the cone X. Section 4.3 describes the cost function used to evaluate a set of nodes. Section 4.4 describes a heuristic for selecting sets of nodes that satisfy the constraints and whose mapping will result in a significant area reduction.

4.1 Mapping Sequential Logic

Let X be a cloud of sequential logic. Let M and N be the inputs to and outputs from the cloud. The set N is the set of nodes n such that $n \in X$ and n drives a node not in X. The set M is the set of all nodes m such that $m \notin X$ and m drives a node in X. Note that the above conditions imply the constraint that every node in X must be in the fanout-free cone rooted at the set of nodes N. Thus X is a subset of the maximum fanout-free cone rooted at the set of nodes N. Let R be the set of register nodes within the cone X. Let us assume each path from each of the inputs (M) to the outputs (N) traverses at least one register, and all registers share the same control signal set (clock, clock enable, synchronous clears, etc). For now, a simplifying assumption is made: no asynchronous clear or reset signals are used. The cone X can be seen as a multi-input, multi-output, state machine whose state is encoded in the set of registers R. If we can implement an equivalent state machine Y using EMBs, and connect those nodes driven by N to Y, the cone of sequential logic X can be entirely removed.

Figure 2 is an example showing the restructuring of an arbitrary cone of sequential logic into one compatible for mapping. Let F be all registers in the cone whose inputs are, directly or indirectly, from another register in the cone. More formally, F is the set of all registers $f \in R$ such that the support set of f, $SS(f)$ contains a register in R; that is, $SS(f) \cap R \neq \emptyset$. We note that we can restructure the cone to ensure each path through the cone traverses exactly one register node. We force the input to each of the registers in F to leave the cone and re-enter the cone. In the example in Figure 2, additional cone inputs and outputs F_1 and F_2 are created.

We can transform this newly restructured cone into an EMB Y of width w and depth d where the inputs are driven by the original inputs and the new feedback inputs ($M \cup F$) and the outputs are driven by the original outputs and the new feedback outputs ($N \cup F$). Note that each path through Y traverses exactly one register node. The F feedback signals are both outputs of and inputs to Y and represent the state of state machine encoded into $|F|$ bits.

We note that the register nodes on each path through Y can be implicitly retimed to the inputs of Y. An embedded memory block can be used to implement any Y derived in this fashion provided the number of bits required does not exceed the capacity of the memory. The requirement that inputs be registered is inherently satisfied. Figure 3 shows the implicit retiming of the restructured cone into an EMB.

4.2 Compensating for Asynchronous Resets

Many synchronous circuits use asynchronous reset signals. It is expected that for the majority of sequential logic cones selected for mapping, registers in the cone will have asynchronous reset signals. We assume that each cone of logic X has a maximum of one unique asynchronous clear signal, s.

One major architectural constraint regarding target architecture class $arch_{no\_aclr}$, which includes most modern FPGAs, is due to the lack of asynchronous clear signals on EMBs. When applied to the input register, the asynchronous clear signal immediately clears the input registers. However, the output of the memory does not show the effect of the asynchronous clear until the next rising clock edge. Thus, the asynchronous clear of the EMB cannot implement the required asynchronous reset functionality of the register. As a result, for this architecture, we need to add additional logic outside the EMB, in the form of additional logic elements, to give the correct behaviour upon asynchronous reset: one register per asynchronous clear signal s and one combinational node per output of the EMB, as seen in Figure 4.

Although the register on the asynchronous clear signal can be shared with subsequent mappings, the combinational nodes for each output cannot. This additional logic reduces the expected area gains. Often, this compensation logic has a larger area than the replaced cone, rendering the operation counter-productive. The cost function for area reduction is modified to account for this, as described in 4.3. In addition, the delay increases with the addition of the extra combinational logic.

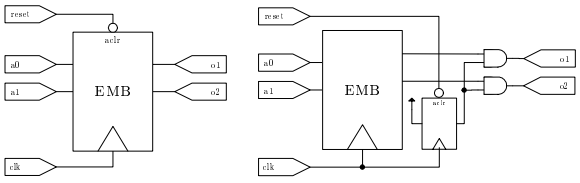

Figure 4: Memory with Asynchronous Reset, and Equivalent Implementation

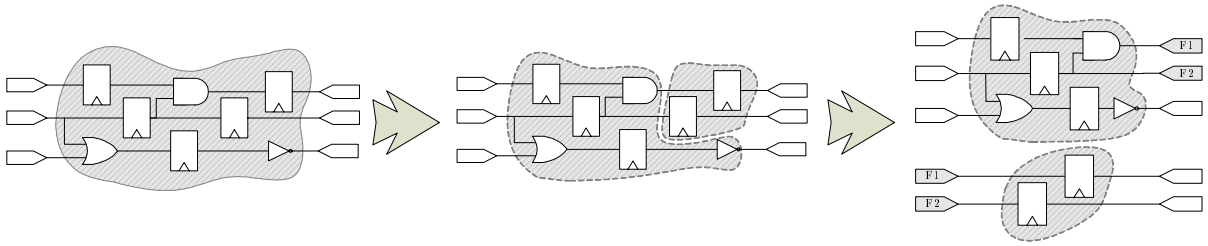

Figure 2: Restructuring a Sequential Logic Cloud

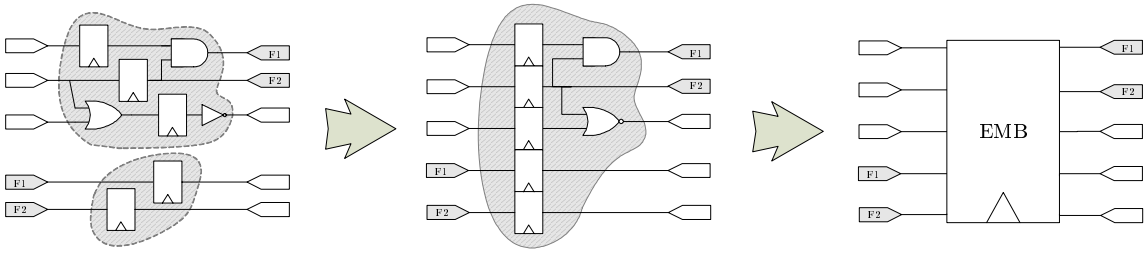

Figure 3: Implicit Retiming into an Embedded Memory Block

For FPGAs of the architecture class $arch_{aclr}$, no additional asynchronous fix-up circuitry is required.

4.3 Area Reduction Cost Function

Given a cone of sequential logic X, a cost function is used to guide the growing of the cone. The cost function consists of a weighted sum of two components: the *Area Reduction Cost*, proportional to the predicted change in the number of logic elements, and the *Memory Use Cost*, proportional to the number of bits of memory required to implement the cone. The area reduction cost is given as: $c(X) = -k - j + a$ where k is the predicted number of logic elements removed, j is the predicted reduction in logic elements due to collapsing, and a is the number of logic elements added to correct the asynchronous reset behaviour. If an asynchronous reset is used, the number of logic elements required $a = d$, the number of outputs of the new EMB. If no asynchronous reset is used, $a = 0$. A logic element can be removed if both its combinational and register nodes can be removed. If each node is either in X or unused, the logic element is predicted to be removed. A combinational-only logic element can be collapsed if it has fan-out of one and can be merged into the output. It is expected that some added compensation nodes can be collapsed in this manner. The memory use cost is the size of the EMB in bits, $w2^d$, where d and w are defined in section 4.1. A large penalty is assigned if the required size is larger than the size of all available EMBs.

4.4 Cone Growth Heuristic

Our heuristic for growing the cones to be mapped proceeds in two phases. In the first, a set of nodes on the output of register nodes is selected for mapping. Second, nodes from the input of the register nodes are added to the set and implicitly retimed. We refer to the first stage as *selection* and to the second as *expansion*. Figure 5 gives an overview of the cone grow heuristic, and Figure 6 shows an example cone selection and expansion.

4.4.1 Node Set Selection

At the beginning of each iteration of the heuristic, a seed node c is selected from all nodes in the circuit who have not participated in a mapping. A simple greedy heuristic is then used to grow the cone from the seed node. The set of all registers compatible with each register in the support set of c (those that share all control signals) is determined. The live set, LiveSet(c), is the union of the compatible register set with all nodes whose support set is a subset of the compatible register set. Thus LiveSet(c) is the set of possible nodes to add which still satisfy the register compatibility constraint. If the live set is empty the iteration of mapping fails and is repeated with a new seed node.

From the live set, nodes are greedily selected and added to the set of nodes to be mapped. The process of adding a node to the set may cause multiple nodes to be added. The input nodes are recursively added up to and including the support set. This ensures that the cone to be mapped remains connected. Each candidate node in turn is test-added to the set, *expanded* through the node set expansion heuristic, and then evaluated by the cost function. The candidate node resulting in the lowest cost is added to the set. The iteration is terminated when adding each node results in a higher cost. Figures 6(a) and 6(b) show an example of the selection process. A candidate node, and all nodes up to its support set, are added to the set.

4.4.2 Node Set Expansion

Given a set X to be mapped (with register nodes R), during the expansion phase we add nodes from the inputs of R into our mapping set. When mapping is performed, the registers are implicitly retimed across these added nodes. Only nodes that are in the fanout-free cone rooted at the register nodes R and not in the LiveSet are eligible for inclusion. At the time of node-set expansion, w, the number of outputs of the set X is known. We can calculate the maximum number of inputs d such

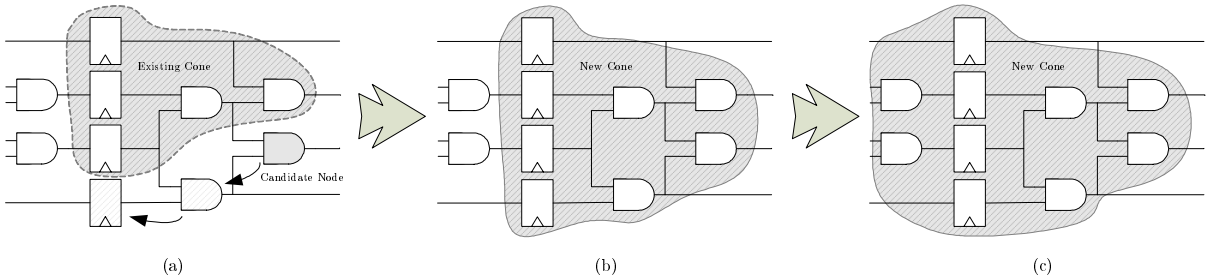

Figure 6: Example Cone Selection and Expansion

```
1  C ← Circuit
2  for  c ∈ C
3       if   LiveSet(c) ≠ {∅}
4            eSet ← InputAndSupport(c)
5            set ← Select(eSet, LiveSet(c))
6            (mapSet, mapCost) ← Expand(set)
7            if   mapCost < 0
8                 PerformMapping(mapSet)
9            end if
10      end if
11 end for
12
13 function Select(X, L)
14      bestSet ← X
15      (set, bestCost) ← Expand(X)
16      for  x ∈ L
17           X' ← X ∪ {x} ∪ InputAndSupport(x)
18           (set, cost) ← Expand(X')
19           if   cost < bestCost
20                bestSet ← X'
21                bestCost ← cost
22           end if
23      end for
24      bestSet ← Select(bestSet, L − bestSet)
25      return bestSet
26 end function
27
28 function Expand(X)
29      bestSet ← X
30      bestCost ← Cost(X)
31      for  x ∈ MFFC(X) ∩ Inputs(X)
32           X' ← X ∪ {x}
33           if   Cost(X') < Cost(X)
34                bestSet ← X'
35                bestCost ← Cost(X')
36           end if
37      end for
38      (bestSet, bestcost) ← Expand(bestSet)
39      return (bestSet, bestcost)
40 end function
```

Figure 5: An overview of the Cone Grow Heuristic.

that the resulting d-input, w-output function will fit into the largest available EMB. The problem is similar to finding the maximum-volume d-feasible cut of the maximum fanout free cone rooted at the registers R. An algorithm for finding this cut was presented in [11], but would not to be appropriate due to our specialized cost function.

For our implementation, we employ a simple greedy heuristic to perform the node set expansion. We test-add each node to the set, and evaluate it by the cost function. The node resulting in the lowest cost is added to the set. The process is repeated until a local minima is reached. Figures 6(b) and 6(c) show an example of the expansion process. Nodes on the inputs of the registers are added to the cone to be mapped.

4.4.3 Performance Considerations

EMBs are considerably slower than combinational lookup tables, so it is expected that, without modification, the *RAM-MAP* technique will significantly reduce the maximum frequency of operation of the circuit. The technique can be modified to prevent the *selection* of critical combinational nodes. We first perform a timing analysis step using a statistical delay model described in [12]. The *expected slack* (ES) of a cone of logic after mapping to memory can then be estimated using the minimum expected slack of all outputs:

$$ES = \min_{o \in outputs}(slack_o + LUTDelay_o - memoryDelay)$$

where $slack_o$ is the slack at an output of the cone, $LUTDelay_o$ is the delay of the shortest path from the output to a register in the cone, and *memoryDelay* is the expected combinational delay of the EMB and asynchronous fix-up logic. The *expected slack ratio* (ESR) is then calculated from the *expected slack* and the concept of *slack ratio threshold* (SRT) is employed. The SRT defines a threshold below which the expected slack ratio should not fall. If the expected slack ratio is below the threshold, the operation is deemed to significantly and adversely affect timing and is not performed. If the expected slack ratio remains above the threshold, the operation is performed as normal. Thus, when selecting nodes, a candidate node which causes the cone to have an $ESR < SRT$ is rejected.

Due to the implicit retiming, the combinational logic delay on the input of the EMB cannot increase. Thus any node is acceptable for inclusion during the *expansion* process.

4.5 Implementation Efficiency

The implementation of our algorithm includes several optimizations that do not reduce the worst-case runtime, but still significantly speed up the technique.

Dynamic programming is utilized in the *expansion* phase of the technique. A solution cache is indexed on two characteristics of the cone: the set of registers of the cone X to be expanded as well as the allowable increase in number of

inputs (calculated from the maximum memory size, and the current number of outputs). Two cones sharing these characteristics can use the same solution from the cache.

Branch pruning is frequently employed when performing the greedy cone selection and expansion heuristics. One example is the removal of nodes from the LIVESET. When the node resulting in the largest area reduction is added to the cone, the nodes consistently resulting in large area increases are removed from future consideration for inclusion with this cone. This pruning does not significantly affect the quality of the final solution.

5. EXPERIMENTAL RESULTS

Altera's Stratix [13] chips were used as the target for the logic mapping experiment. The chip is comprised of I/O elements (IOEs), logic array blocks (LABs), digital signal processing blocks (DSPs) and embedded memory blocks (M512, M4K and M-RAM). A LAB in a Stratix device contains 10 logic elements (LEs). The Stratix LE contains a four-input lookup table (4-LUT), a register and some logic that facilitates the creation of arithmetic circuits.

The chip is composed of three types of embedded memory blocks: the 512-bit M512, the 4096-bit M4K, and the 512-Kbit M-RAM block. Each of these blocks is synchronous, requiring registered address, data, and control signal inputs, and optionally registered outputs. Additionally, the M-RAM does not support memory initialization and cannot be used for logic mapping. The chip does not have an asynchronous clear available on the embedded memory blocks, and classifies into the $arch_{no\_aclr}$ architecture class.

Each type of embedded memory block can be used in multiple width and depth configurations. For example, the 512-bit M512 can be used in a 512-address by 1-bit word configuration (9 address inputs and 1 data output), a 256×2 configuration (8 inputs and 2 outputs), and others up to and including 32×16 (5 inputs and 16 outputs).

5.1 Results for the $arch_{no\_aclr}$ Architecture

In our experiments, all steps of the FPGA CAD flow were performed by a modified version of Quartus II v5.0. After technology mapping but prior to placement, the flow was modified to perform the *RAM-MAP* EMB mapping technique.

We study the benefits of applying the EMB mapping technique on 87 industrial circuits. Quartus was run twice, first with *RAM-MAP* turned off, and then with it turned on. For each circuit, the device chosen was the smallest Stratix-family device that could fit the circuit (with *RAM-MAP* off). The number of logic elements observed at the end of each run is used to compute the *area reduction* observed as a result of applying the technique. Note that as these circuits are industrial, a number of them already utilize the embedded memories. For some of the circuits, very few memories are available for use by *RAM-MAP*.

Figure 7 presents the area reduction observed for each circuit as a result of applying the *RAM-MAP* technique with an *slack ratio threshold* of $-\infty$ (all operations are accepted). A mean area reduction of 6.2% was observed. No circuits were observed to increase in area as a result of applying the technique because the cost function is able to predict the resulting area with perfect accuracy (and rejects area-increasing mappings). On average, the area reduction per used M4K is 4.59 logic elements and 2.28 logic elements per used M512.

5.2 Results for the $arch_{aclr}$ Architecture

It is clear that the need to compensate for asynchronous clears diminishes the area reduction available from *RAM-MAP* when using the $arch_{no\_aclr}$ architecture class, which includes Altera's Stratix device. If an asynchronous clear were available on the EMB, it is expected that the area reduction should increase. We can quantify this prediction by repeating the experiment, but assuming the Stratix architecture is modified (as per section 3.1) to be of class $arch_{aclr}$. The flow is identical, except we do not create asynchronous clear compensation logic elements and adjust our cost-function accordingly.

Figure 7 presents the area reduction observed for each circuit as a result of applying the *RAM-MAP* technique, assuming an architecture of class $arch_{aclr}$, with an SRT of $-\infty$ (all operations are accepted). A mean area reduction of 14.1% was observed. The area reduction and performance degradation are higher due to logic cones whose mapping was previously undesirable (due to the asynchronous clear cost) now being mapped.

5.3 Impact on Performance

The primary goal of the EMB mapping technique is to decrease area. EMBs are much slower than conventional combinational lookup tables, and the constrained physical location of EMBs blocks on the chip introduces additional placement and routing constraints. It is therefore expected that mapping logic into EMBs will reduce the performance of the circuit. In obtaining the area reduction indicated above for the $arch_{no\_aclr}$ architecture, with a *slack ratio threshold* of $-\infty$ (all operations are accepted), the technique decreases the maximum frequency of operation of the circuit by 18.2% (mean).

This performance degradation is large, and thus this technique is not applicable to all designs. However, for those designs with large amounts of timing slack, the designer's greatest concern is using the smallest possible device that will fit their circuit as this device is generally less costly than larger devices. This technique may help the designer utilize a smaller device than otherwise possible.

With the addition of the $arch_{aclr}$ architecture modification, the reduction in the maximum frequency of operation is 34.5% (mean). This increase in reduction is due to the increased number of mappings that take place. Note that these performance degradations are measured after placement, routing and final signoff timing analysis.

The *slack ratio threshold* can be appropriately chosen to reduce the performance penalty at the cost of less area reduction. Figure 8 shows the performance versus area trade-off for 7 values of the *slack ratio threshold*. Each point represents the average area reduction versus the average performance reduction of the entire benchmark of 87 circuits, run with a different *slack ratio threshold* parameter. Because the *RAM-MAP* technique is performed prior to placement and it is difficult to predict the post-placement delay [14], it is very difficult to both predict and control the performance reduction.

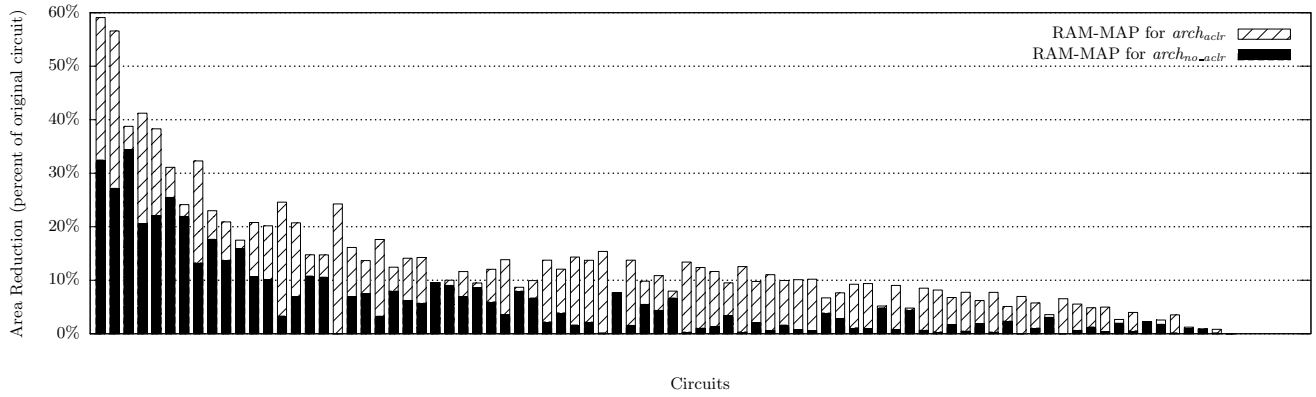

Figure 7: Area Reduction on 87 Industrial Circuits

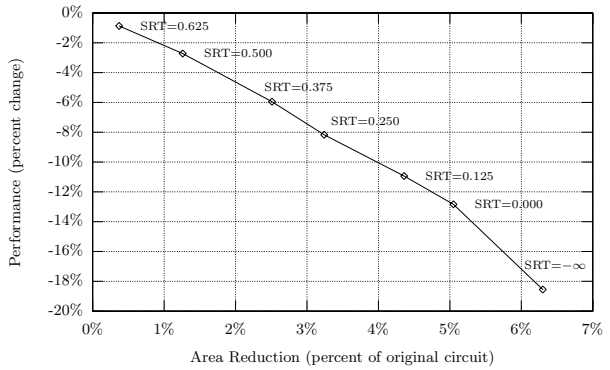

Figure 8: Performance vs. Area Trade-off

6. ADVANCED TECHNIQUES

This section presents two techniques for further increasing area reduction from the *RAM-MAP* technique.

6.1 Negative-Edge Clocked Memory

One of the limitations of the *RAM-MAP* technique described above is that all cones to be mapped must contain registers. Every path from input to output in the cone must traverse at least one register, because every EMB requires registered inputs. If purely asynchronous EMBs were available, it would be possible to map a cone of combinational logic between registers into memory.

It is occasionally possible to implement an asynchronous EMB using a synchronous EMB [15]. This is accomplished by clocking the synchronous EMB with an inverted clock. Stringent conditions must hold in order to be able to perform this mapping. First, all registers reachable by traversing the fan-in or fan-out network of the mapped cone must have compatible control signals. Secondly, two EMBs used in this manner cannot fan-in or fan-out to each other. It is important to note that fix-up logic still needs to be added, if the registers on the fan-in of the mapped cone have asynchronous clears. This is because the results of an asynchronous clear activated after the falling edge of the clock will not propagate through the memory to the inputs of the next level of registers.

Figure 9 shows an example of mapping a cone of logic into a negative-edge clocked memory element. A brief description of the algorithm follows. First, all sets of registers with compatible control sets are identified, and each set is given a unique identifier. In the example figure, the set identifier is noted on the register. Second, the fan-in and fan-out networks of each register are recursively traversed and annotated with the identifier for the register, as seen in Figure 9(a). All combinational nodes with only one annotation are bounded by compatible registers and are *mappable*. Next, regions of connected mappable nodes are identified, as seen in Figure 9(b). A subset of each region is selected for mapping by a cost function. In the example, the entire region is selected for mapping; this may be neither desirable (due to asynchronous fix-up logic) nor feasible (due to the maximum size of EMBs). Finally, the selected cone of logic is replaced by an equivalent synchronous EMB, with an inverted clock signal, as seen in Figure 9(c).

As a proof of concept, an implementation of this technique with a simple greedy heuristic to select cones can achieve a reduction in area of 1.1% (up to 7.2%) on top of any reductions realized through the *RAM-MAP* technique reported in Section 5. A better algorithm for increased area reduction is an area of future research.

6.2 State Machine Re-Encoding

The mapping method we described identifies a cone of sequential logic which is turned into a finite state machine and then placed in an EMB. Standard textbook methods of state machine reduction [16] can be used to reduce the number of states thereby reducing both the EMB size as well as any logic needed to emulate an asynchronous reset. Area reduction can be realized if the number of bits required to encode the state machine is reduced. Since the signals carrying the state machine encoding appear at both the inputs and outputs of an EMB, both the number of RAM inputs and outputs can be reduced, reducing the need for asynchronous fix-up logic.

An implementation of this technique can achieve a reduction in area of up to 1.2%, primarily on circuits where feedback structures are common. On average across the circuit set, a reduction of 0.1% is realized. These results

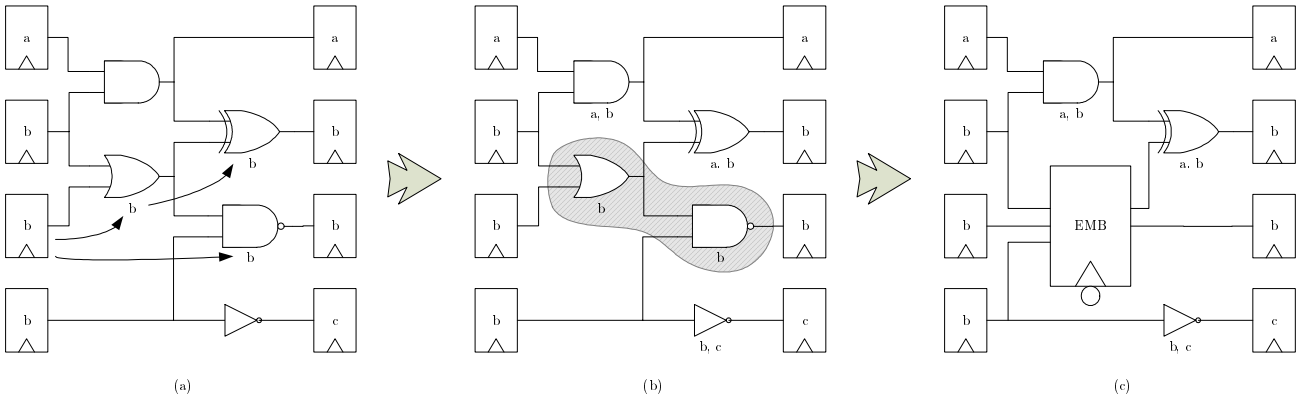

Figure 9: Mapping into Negative-Edge Clocked Memory Elements

are on top of those reported in Section 5.

7. CONCLUSION

In this paper, we describe a new mapping technique. The *RAM-MAP* technique maps combinational and sequential logic into unused embedded memory blocks to reduce the number of logic elements required to implement the circuit. The technique is also able to satisfy two constraints of the target architecture's embedded memory blocks. Extra logic is added to compensate for the lack of asynchronous clear on EMBs. Special considerations are made when selecting and manipulating cones to ensure cones can be mapped into the input-registered, synchronous EMBs. On a set of 87 industrial circuits, the *RAM-MAP* technique is able to reduce on average by 6.2% and up to 34.4% the number of logic elements required to implement the circuit on our target architecture, Stratix. With a small change to the architecture that increases overall FPGA size by 0.05%, the potential area reduction is increased on average to 14.1% and up to 59.1%.

8. REFERENCES

[1] Altera Corporation, *Altera Product Catalog*, May 2005.

[2] Xilinx Corporation, *Virtex Series FPGAs Product Matrix*, May 2005.

[3] S. J. E. Wilton, "Smap: Heterogeneous technology mapping for area reduction in FPGAs with embedded memory arrays," *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, February 1998.

[4] S. J. E. Wilton, "Heterogeneous technology mapping for area reduction in FPGAs with embedded memory arrays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 56–68, January 2000.

[5] J. Cong and S. Xu, "Technology mapping for FPGAs with embedded memory blocks," *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 179–188, February 1998.

[6] M. Kumar, J. Bobba, V. Kamakoti, "MemMap: Technology Mapping Algorithm for Area Reduction in FPGAs with Embedded Memory Arrays Using Reconvergence Analysis," *Design, Automation and Test in Europe Conference and Exhibition Volume II (DATE'04)*, pp. 922–929, 2004.

[7] F. Heile and A. Leaver, "Hybrid product term and LUT based architectures using embedded memory blocks," *International Symposium on Field Programmable Gate Arrays (FPGA)*, 1999.

[8] R. Hitchcock, G. Smith, and D. Cheng, "Timing analysis of computer-hardware," *IBM Journal of Research and Development*, pp. 100–105, January 1983.

[9] C. Leiserson, F. Rose, and J. Saxe, *Optimizing Synchronous Circuitry*, 1983.

[10] C. Leiserson and J. Saxe, *Retiming Synchronous Circuitry*, 1991.

[11] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, pp. 1–12, January 1994.

[12] D. Singh, V. Manohararajah, and S. Brown, "Two-stage physical synthesis for FPGAs," *Custom Integrated Circuits Conference (CICC)*, September 2005. To appear.

[13] Altera Corporation, *Stratix Device Handbook (Complete Two-Volume Set)*, July 2005.

[14] V. Manohararajah, G. Chiu, D. Singh, and S. Brown, "Difficulty of Predicting Interconnect Delay in a Timing Driven FPGA CAD Flow," *Proceedings of the 2006 International Workshop on System Level Interconnect Prediction*, pp. 3–8, March 2006.

[15] Altera Corporation, *Application Note 210: Converting Memory from Asynchronous to Synchronous for Stratix and Stratix GX Designs*, November 2002.

[16] Z. Kohavi, *Switching and Finite Automata Theory*. McGraw-Hill Publishing Company, 2nd ed., 1978.