A CAD Framework for MALIBU: An FPGA with Time-multiplexed Coarse-Grained Elements

David Grant Chris Wang Guy G.F. Lemieux
Department of Electrical and Computer Engineering
University of British Columbia, Vancouver
{davidg,chrisw,lemieux}@ece.ubc.ca

ABSTRACT

Modern FPGAs are used to implement a wide range of circuits, many of which have coarse-grained and fine-grained components. The ever-increasing size of these circuits places great demand on CAD tools to synthesize circuits faster and without loss in quality. Synthesizing coarse-grained components onto fine-grained FPGA resources is inefficient, and past attempts to optimize FPGAs for word-oriented datapaths have met with limited success. This paper presents a CAD flow to fully compile Verilog into a configuration bitstream for a new type of FPGA with time-multiplexed coarse-grained resources. We demonstrate two approaches with gains of 61x and 42x in synthesis time on average compared to QuartusII, but due to time-multiplexing and current synthesis limitations we achieve circuit speeds of 14x and 8.5x slower on average. We show the tools can also trade density for maximum clock frequency.

Categories and Subject Descriptors

B.6.3 [**Design Aids**]: Automatic Synthesis; B.7.2 [**Design Aids**]: Placement and routing

General Terms

Algorithms, Design, Performance

1. INTRODUCTION

Modern FPGA devices contain over 1 million LUTs, over 1000 hard memory or multiplier blocks, and about 300 wires per row or column. In addition, they are continuing to grow with Moore's law. As a result, great demand is placed on synthesis tools to compile ever-larger netlists without degrading result quality or increasing run-time. Given already long FPGA CAD run-times, vendors are turning to parallel compilation. While this may help when powerful compute systems are available, light-weight approaches would be preferred.

One reason for slow CAD is that FPGAs are still bit-oriented. This partly reflects their past when they were used for glue logic, but modern usage has expanded to implementing a wide range of circuits. Increasingly popular are generators like SOPCBuilder, EDK, and C-to-gates flows that generate large hardware datapaths. These new circuits are mostly word-oriented, but they may also have many fine-grained control signals.

Past attempts to optimize FPGAs for word-oriented datapaths have met limited success. By organizing wires and logic into words, the number of configuration bits can be reduced by sharing them among a word, and multiplexer sizes can be reduced. However, only a small overall savings of roughly 10% [34] has been realized.

One feature not attempted in prior datapath-FPGA research is time-multiplexing. By time-multiplexing the coarse-grained elements and coarse-grained interconnect, the area-cost of these large components can be amortized over many clock cycles. The improved density also allows larger circuits to be mapped into smaller architectures by trading off the maximum clock frequency. Moreover, it reduces the placement and routing problem size, which reduces synthesis time. Time-multiplexing has been applied to fine-grain logic research [31, 14, 7], commercially by Tabula[12], and to CGRAs[9, 22]. However, time-multiplexed coarse-grain elements like ALUs as a logic resource for compiled HDL in an FPGA is relatively unexplored.

What is needed for these word-oriented circuits is a heterogeneous time-multiplexed architecture that combines features from FPGAs and CGRAs, and a set of CAD tools to synthesize circuits to such an architecture. This research is focused on the CAD, and specifically on the steps after logic synthesis, that is, the placement, routing, and scheduling.

This paper presents a complete CAD flow that can compile the full synthesizable subset of Verilog2005 into a configuration bitstream for this new type of FPGA. Common tools such as VPR [1], ABC [2], OdinII [13], and Verilator [29] are used within the tool flow, but new tools have been created for placing, routing, and scheduling the time-multiplexed coarse-grain logic along with the (not time-multiplexed) fine-grained logic. In Section 4 these tools are compared to standard flows with QuartusII and VPR, demonstrating that significant gains in compile-time and logic density are possible. In Section 4.4 we demonstrate that the tools can trade density for circuit speed, mapping large circuits to run slowly on a small device, or run faster on a larger device. Also in Section 4.4, we demonstrate that the tools can improve circuit speed at the cost of compile-time by altering the bit-width (W_f) of signals considered "fine-grained" and pushing more (or less) logic to the fine-grained LUTs and interconnect. While the current results do not achieve the same F_{max} performance of standard flows, there are many opportunities for significant performance gains in the future.

Our new FPGA architecture is called "Malibu" and is presented in Section 2. Our proposal is to add coarse-grained ALU-like elements (CGs) to the FPGA's CLB. The CG is time-multiplexed,

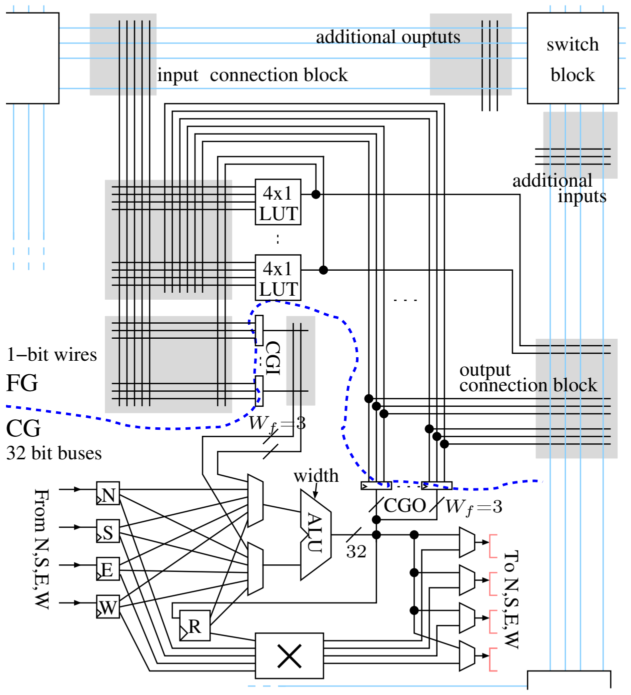

Figure 1: The Malibu Architecture CLB with the fine-grained (FG) and coarse-grained (CG) parts.

Table 1: Malibu Units and Detailed Operations

Units	Operators	Area (T)
Multiply	$*^1$	35,000
Arithmetic	$+,-,<^1,\leq^1,=,\neq,>,\geq^1$	1,995
Logic	bitwise(&, $\vert$, $\wedge$, $\sim$), ternary(?:), extend[1], reductions(&, $\vert$, $\wedge$, !)	3,208
Interconnect	4x4 32b Xbar, CGI/Os, CG muxes	5,482
BarrelShift	$\ll,\gg$, concat	1,791
FG	16 4-LUTs, LUT I/O	19,455
	Connection and Switch Blocks	17,519
	Combined, incl. Multiply	82,739
	Combined, no Multiply	47,739

[1]: Operators with both Signed and Unsigned modes

whereas the original CLB (now a portion of the CLB, called FG) and the FPGA routing fabric is not. The CG also has dedicated word-wide interconnect to the CG in the neighbouring CLBs. The ALU we propose is Verilog-specific and supports Verilog operators not found in typical ALUs, like bit concatenation (`ab[7:0] = {a,b}`) and unary logic reductions (`parity[0] = ^a`).

2. MALIBU ARCHITECTURE

For Malibu, the most important thing is to extract plenty of word-wide operations from the source Verilog. These are mapped to the time-multiplexed ALUs in the architecture. As well, the fine-grained logic "leftovers" must also be extracted and mapped to the LUTs in the architecture. However, before we can present the details of the CAD flow, we must first present a target architecture and explain the coarse-grain/fine-grain interface.

The architecture (Figure 1) starts with a standard fine-grained (FG) FPGA, and adds a time-multiplexed coarse-grained processing element (CG) to the CLB. The CG connects to the fine-grained CLB through coarse-grained inputs (CGI) and outputs (CGO). There are N_{cgi} CGIs, where each one aggregates a bundle of W_f signals from the fine-grained (FG) resources as an input to the ALU, zero-extended to 32-bits. There are N_{cgo} CGOs, where each CGO latches the W_f least significant bits produced by a specific instruction, providing them to the LUTs or FPGA routing resources. When $W_f = 0$, the fine-grained resources (*all* traditional LUTs and interconnect) are removed, leaving only the new CG.

Each CG is time-multiplexed; it always executes one instruction per cycle, and all communication is explicitly pipelined and scheduled. It operates on a system clock that is different than the user clock cycle; we anticipate a 1 GHz system clock in 65nm technology using custom layout techniques can be readily achieved. Each CG contains a schedule with exactly SL instructions (the schedule length). On the active user clock edge, the instructions start executing, one per system clock cycle. At the end of the SL instructions, the CG pauses for the next user clock edge before starting over. One complete pass of the SL is required for each user clock cycle, so this limits the F_{max} to $\frac{1}{SL} \cdot 1$ GHz.

All common Verilog operations can be easily mapped onto the ALUs in the CG. For example, adding two 4-bit values in Verilog, written as `o = a + b`, can be expressed using 32-bit ALU operations in C language as `o = ( (a&0xf) + (b&0xf) )&0xf`. However, the ALUs we propose are Verilog-specific; they automatically truncate output results to the desired width by forcing all upper bits to zero. The width is encoded in the instruction, and is a separate input to the ALU as shown in Figure 1. For some operations the width has a special meaning, for example for concat it is used to specify the number of bits to concatenate from the LSB input (the remaining $32 - width$ bits are taken from the MSB input), and the output is not truncated. Each input operand width is left unspecified and the CAD ensures the correct width is provided by inserting zero-extend or sign-extend operations where necessary.

A complete list of all operations in Malibu are shown in groups in Table 1, including the estimated area using VPR's units of minimum-width transistor area (T). The area estimate is from a manual gate-level design of each part except the multiplier [5], counting the number of basic components (gates, muxes, etc.) required for each, and then converting those into minimum-width transistors. E.g., an OR gate requires 3 NMOS and 3 PMOS transistors, and a PMOS requires 1.5x the area of an NMOS, so the OR gate requires 7.5 T.

The combined area for the blocks is slightly less than the sum of individual unit areas due to redundancy removal when combined. The acronym MALIBU, an extension of ALU, originates from the name of these groups or units. The ALU itself comprises a total of 30 operations. Multiply and comparisons have signed and unsigned variations. Each CG operation also requires a 5-bit width parameter for the output width in bits. Exceptions are sign extension, Verilog unary logic reduction, and concatenation which use the width an input because the output width is implied. To save area, only one in five columns of CLBs contain a multiplier.

The result of a CG operation can potentially be written to any address in the R memory, to any of the CGOs, and to any address in each of the N, S, E, W (NSEW) memories concurrently. Each of these memories operate synchronously using a single write port and three read ports. So far, our CAD results indicate each NSEW memory should have up to 16 entries, and the R memory should have up to 64 entries.

To simplify the tools, the FG does not contain any flip-flops. Instead, the flip-flop state is stored in either an R or NSEW memory. The value is transferred to a CGO latch at the beginning of each user clock cycle so the value is stable for the duration of the cycle.

To avoid introducing another memory block in the CG to implement user-circuit memory, we have a novel feature where user-circuit memory blocks are packed into a contiguous block of space in R. Special load/store operations are used and require one extra

Table 2: Malibu Memory Area Estimates

	Specifications	SRAM μm^2	eDRAM μm^2	Flash μm^2
NSEW	32x16, 3R1W	1,229 – 3,521	—	—
R	32x128, 3R1W	4,669 – 28,160	—	—
Instr.	90x256, 1RW	11,290 – 30,849	6,682	1,579
Instr.	90x1024, 1RW	45,158 – 96,840	26,726	6,318

system cycle to perform the required indexing. The largest user memory block in our benchmarks is 2kbit, for which we add 64 more entries to R (total 128).

Figure 1 also shows a 4x4 routing crossbar. It writes values to the NSEW memories located in the four cardinal neighbours by taking values from the local NSEW and R memories. Although there are 5 sources, a 4:1 mux is sufficient because the W crossbar output never requires the W crossbar input, for example. The crossbar routes coarse-grained signals concurrently with computation, and keeps CG communication off the FG routing resources. The ALU cannot write to the same NSEW memory as the crossbar in the same clock cycle. The CAD detects this condition and writes to R instead, then schedules a transfer from R to the target NSEW in the next available cycle. We found the ability for the ALU to write to NSEW directly is important for performance.

We have encoded each CG instruction, including all of the source and destination addresses and crossbar control, into 90 bits. In contrast, there are well over 1,000 configuration bits in a traditional VPR-style CLB (ten 6-LUTs require 640 bits, the 60 LUT inputs require at least 5 bits each, plus bits needed to configure flip-flops and all of the interconnect). However, after time-multiplexing, the CG requires $SL \times 90$ bits. If the user extensively time-multiplexes a large circuit onto very few CLBs, upwards of 1024 instructions per CLB might be required. However, our current tools show 256 is sufficient. The long-term goal of the CAD is to significantly reduce this value.

The CG makes extensive use of memories. We estimated the area of these memories, but found results can vary as shown in Table 2. The NSEW and R memories need very fast read and write access, so they should be implemented as SRAM. The instruction memory is primarily read-only and accessed sequentially, allowing it to be pipelined. It may be implementable in SRAM, eDRAM, or flash. An upper bound on SRAM area was obtained using the Artisan Memory Compiler. However, it is not optimized for small memories and includes overhead like redundancy. Using technology parameters for SRAM [24, 28], eDRAM [15] and flash [17], we computed lower bounds on area as follows. Using transistor counting, we estimate control logic overhead (decoders, sense amps, drivers) as 50% area per bit, which correlates with data in [15]. Each extra port is modelled as 100% area per bit. For example, a 3-port, 32b memory implemented in $0.25\mu m^2$/bit technology would require $0.25 \times 32 \times 3 \times 1.5 = 36\mu m^2$ area. To convert to VPR transistor-area (T) metrics, the iFAR repository assumes the area of 1T$\approx 0.5\mu m^2$ in 65nm. Using the SRAM lower bounds, each MALIBU CLB requires 17,188μm^2 for memory, compared with 15,110μm^2 for the FG and CG logic (no multiplier), and 8,760μm^2 for the FG interconnect.

One limitation of this architecture is the assumption of a single user clock domain. We believe this greatly simplifies the types of circuits created by C-to-gates flows, which we hope would naturally target this type of architecture. Nevertheless, it is important to address multiple clock domains in our future work.

For the details presented in this section, please keep in mind this architecture is intended to be a starting point for many optimizations which we have not yet performed. The emphasis in this paper is about producing a flexible CAD system that can allow us to model many variations in the architecture. We assume fully-populated C blocks and IOBs, and all FG and CG interconnect wires are directional but span only a single CLB in length. As well, we usually run the tools in an "exploratory mode", where the number of resources float, allowing us to find the natural demand rather than trying to fit to a fixed limit.

3. MALIBU TOOLFLOW

The Malibu CAD tools use a CDFG (Control and Data Flow Graph) representation of the circuit where graph nodes are circuit operations and graph edges are communication. Each node has an operation type, a set of ordered sources, a set of sinks, and an output bit-width. Each edge has a delay (number of system cycles) and a bit-width equal to that of the driving node. The value W_f is the fine-grained width threshold; all nodes and edges wider than W_f are implemented on the CG resources only, placing the nodes and edges W_f or smaller in the FG resources only.

The objective of the tool flow is to assign each coarse-grained node to a <CG, timeslot>-tuple, each fine-grained node to a LUT, and to route all edges. The tools are divided into several steps for fine-grained synthesis and FPGA mapping, placement, routing, and a step called scheduling to order the time-multiplexed operations over time. All steps are timing-driven, which means minimizing the schedule length.

The academic FPGA CAD flow is shown in Figure 2a. It uses T-VPack and VPR[20], which have become the *de facto* standards for academic FPGA clustering, placement, and routing. Synthesis from Verilog to technology-mapped LUTs is done with QuartusII since OdinII only implements a subset of Verilog. This toolfow is used as a baseline comparison for results in Section 4.

This section describes two approaches for mapping a circuit to the Malibu architecture: Malibu-CAD (M-CAD) and Malibu-HOT (M-HOT). Both approaches extend the academic FPGA CAD flow to add support for coarse-grained time-multiplexed entities. Figure 2b shows the M-CAD approach, which follows the same order of operations as the traditional CAD flow. Figure 2c shows the height-oriented tool (M-HOT) approach which performs placement, routing, and scheduling simultaneously. These two approaches are identical up to the clustering step. Generally, M-CAD runs faster but produces slower circuits than M-HOT.

Prior work [11] details RVETool, which is the coarse-grained part of the M-CAD approach. Therefore, in this paper we only include the details required for the fine-grained integration and for the new M-HOT approach. We refer the reader to [11] for the details of the M-CAD coarse-grained synthesis.

The input Verilog is parsed, elaborated, then split into fine-grained and coarse-grained parts. The fine-grained part is synthesized to LUTs and then merged with the coarse-grained operations for clustering, placement, routing, and scheduling. The parse step outputs the fine-grained circuit parts using a subset of Verilog compatible with OdinII and ABC for synthesis to LUTs.

Not shown in Figure 2 is an architecture file input used to specify the number of PEs, the width of buses, the size of each memory, and the resources in each PE (e.g., if the PE can perform I/O). These parameters act as constraints in the tool flow and are available at all steps.

3.1 Parse and Coarse-Grained Synthesis

The first step in both the M-CAD and M-HOT flow is to construct a CDFG representation of the circuit. A modified version of Verilator [29] parses the input and performs several simple optimizations like module elaboration, dead code elimination, and constant fold-

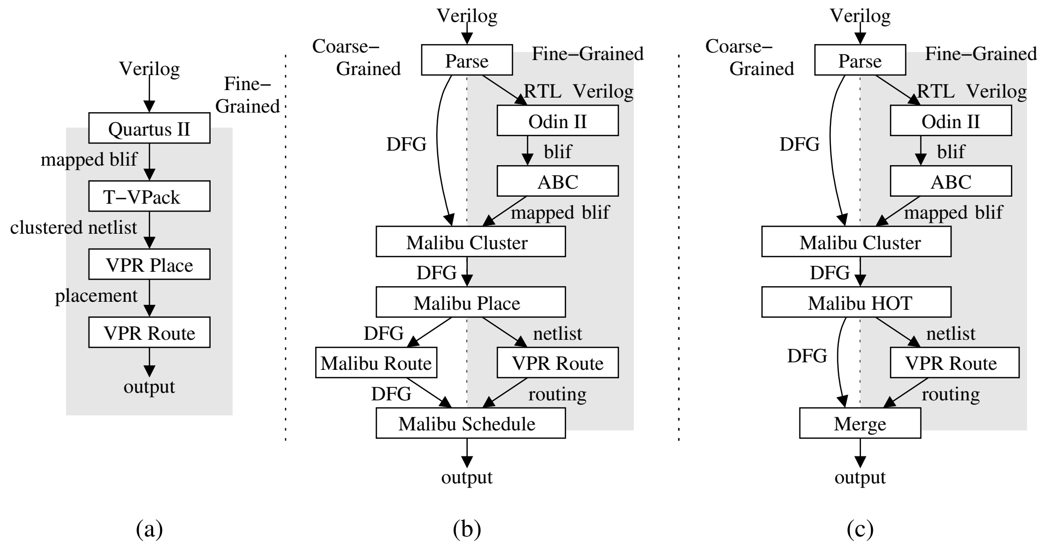

Figure 2: Three CAD flows: (a) Academic (traditional), (b) M-CAD, (c) M-HOT. M-CAD follows the traditional place-then-route flow. M-HOT performs placement, routing, and scheduling simultaneously.

ing. Verilator is not a full-scale commercial-quality synthesis tool, it is designed to output a sequential C++ program, not perform full synthesis of parallel logic. We use Verilator because it is easy to extract a CDFG before it begins to serialize the graph, and because it generates high-quality output for many circuits. However, it appears to perform poorly in some cases (see Section 5).

The Verilator output requires further processing for the Malibu architecture. Signals and operations (CDFG edges and nodes) less than or equal to a specified width ($W_f = 3$ in Figure 1), are considered fine-grained and marked to use the FG resources. Coarse-grained operations that compute FG signals (like a comparison) are mapped to CGOs, while CG operations that depend on FG signals are mapped to CGIs. The remaining operations are considered coarse-grained, and most map trivially onto the supported CG operations. However some require more complicated graph transformations [11] to legalize the CDFG for the architecture.

Before producing the final CDFG, the FG logic is separated from the CG logic and written to a distinct RTL Verilog file for OdinII synthesis.

3.2 Fine-Grained Synthesis

The fine-grained parts of the circuit are synthesized to LUTs using OdinII and ABC, then merged back into the coarse-grained CDFG for clustering. Using RTL Verilog simplifies the description when $W_f > 1$: OdinII elaborates the design to single-bit operations, then ABC cleans up any dangling logic (e.g. from an add operation where the carry bit is discarded), and tech-maps to LUTs.

LUT packing into CLBs is integrated with the placement tool and done with the CG placement. The number of CGI and CGO interfaces needed in a CLB will change while performing CG placement, so LUT packing is done as part of placement.

3.3 Malibu Cluster

The clustering tool collects CG operations into CLB clusters for each CLB. It also groups the LUTs which source or sink CG values into the same CLBs as the CG operation. The overall goal is to reduce the amount of communication by the CG logic.

The tool can cluster code to varying degrees to target any number of CLBs, allowing a tradeoff between area (number of CLBs) and performance (F_{max}). This is demonstrated in Section 4.4.

For the M-CAD approach, the clustering tool uses hMETIS [16] to partition the graph using recursive bisection. To guide hMETIS, all nodes are assigned a weight of 1, except constants which are replicated as needed and assigned a weight of 0 so they can be placed in any CLB for free. Load and store operations from user memory, and CGI/CGO interface operations, are connected with very high edge weights to ensure they will not be separated.

The M-HOT approach keeps CG instructions independent (cluster size of one) because it favours a "move-and-compute" model where computation is done while values are being routed through the current CLB en route to a final CLB instead of being computed in a CLB, stored locally, and then routed later. Other than keeping together the load/store and CGI/CGO interface operations, no clustering is done for M-HOT.

3.4 M-CAD Flow

The M-CAD tool flow performs placement, routing, and scheduling in distinct steps as in traditional CAD. Information is only passed forward so the routing result, for example, cannot be used to go back and generate a better placement. This section briefly describes each step and how the coarse-grained and fine-grained resources are handled. Prior work [11] details just the CG aspects, but the FG aspects presented here are new.

3.4.1 Malibu Place

The Place tool assigns the code clusters to CGs and LUTs into FGs. The goal is to keep the critical path small.

The tool uses VPR's [20] timing-driven annealing placement algorithm with two changes to the cost function. First, a different definition of "delay" is used in the cost function computation to handle both the fine-grained and coarse-grained operations being placed. Second, a parameter $penalty$ discourages illegal placements.

To simplify placer delay estimates, all delays are expressed as integers. Each hop of a coarse-grained communication path has a delay of one, with the total delay being the number of hops. We precomputed Elmore delays with VPR to estimate how far (in CLBs) a fine-grained signal travels in one system clock cycle. The placer can thus estimate FG delays quite easily. The Schedule tool ultimately determines the order of execution of instructions using the actual delays from VPR routing, so an estimate at this point is sufficient.

The time-multiplexed network introduces an additional level of complexity not found in regular FPGAs: two nodes within the same CG may be scheduled in timeslots far apart, causing additional delay not modeled by the number of hops. Unfortunately this delay is not known until scheduling is complete, so at this stage we assume it is zero.

The delay computation is used with a slack and criticality computation to calculate the $timing\_cost$, which is part of the placement cost function. The slack, criticality, and $timing\_cost$ computations are the same as in VPR.

The placer allows illegal placements to be considered. The $penalty$ parameter adds a fixed cost each time memory size is exceeded, unavailable CG or FG resources are used, too many CGI/CGO registers are used, or too few/many CLBs, are used.

At the end of placement, if required, the Place tool also packs multiple small user memories into each CG, ensuring they do not overlap. However, the problem of splitting a large user memory across multiple CG is left for future work.

3.4.2 *Malibu Route*

After placement, VPR's PathFinder router is used to route the fine-grained logic. The schedule tool reads the delay information from VPR and records the delay of each fine-grained link in the DFG.

The coarse-grained routing problem is different from conventional CAD flows because the routing network is time-multiplexed, so temporal as well as spatial decisions must be made. The spatial routing is done using a simple horizontal-then-vertical routing strategy. The router follows existing routes from the same source as far as possible before branching the route towards the new destination CG.

The temporal routing decisions are made during scheduling. When the endpoint of a route is to be scheduled, the scheduler follows each hop of the route, checking that the necessary CG resources are available. If a conflict arises, the route is held in place for as many timeslots as necessary until the resources are available at the next hop. For the F_{max} results in Section 4.1 it was never necessary to hold a value to avoid a routing conflict for any circuit. In practise over all our experiments we have never seen a value held more than two cycles, meaning the architecture has an abundance of CG routing resources.

3.4.3 *Malibu Schedule*

The Schedule tool orchestrates the overall execution of code and movement of data to reproduce the behaviour of the original circuit. It assigns each instruction to a timeslot in a CG, it assigns each coarse-grained route-hop to a timeslot resolving all routing collisions along the way, and it ensures all values are produced/consumed at the appropriate times on the fine-grained resources.

The scheduling algorithm is variation of list scheduling. It begins at $timeslot = 0$ and assigns as many operations as it can across all CGs in that timeslot. It iterates over the sinks of the scheduled operations and uses the routing delay information to compute the minimum timeslot in which those sinks may be scheduled. It then moves on to the second timeslot, and so on. This timeslot-oriented approach ensures the scheduler is fast and is always making forward progress. NOP instructions are inserted in all timeslots that do not contain a circuit node after scheduling.

An operation may be scheduled in $timeslot$ if:

- The $timeslot$ is empty in the CG's ALU.
- All source signals have arrived in time.
- All internal CG resources required by the operation are available.
- All routing resources required by the output of the operation (fine-grained and coarse-grained) are available for the first-hop of the route.

At each timeslot, nodes are considered in order of criticality as computed during placement. This simple ordering reduces the final SL and thus increases the F_{max} by $\approx$10%.

At the end of scheduling, accesses to the NSEW and R memories are assigned specific offsets using a greedy approach. At this point, the CG operations and FG LUTs for each CLB are packed into a single output bitstream.

3.5 M-HOT Flow

The Malibu height-oriented tool flow (M-HOT) is shown in Figure 2c. M-HOT is based a modulo graph embedding scheduler [25], which was tested on a CGRA up to 4x4 PEs. There are several distinctions from the work presented here which are elaborated upon in the following sections.

- Support was added to place, route, and schedule fine-grained operations in parallel with the coarse-grained operations. VPR is called to obtain the FG routing delays.
- Support was added for registers. M-HOT ensures that expected flip-flop behaviour is reproduced.
- M-HOT uses a variable-length schedule that is increased as needed. This is sub-optimal, but it avoids searching for the lowest schedule length through multiple invocations of the tool. M-HOT also supports a fixed schedule length.
- The placement cost function was modified to encourage nodes at each height to spread out to many CLBs, and also modified so that multiple CDFG paths that end at the same user register (logic reconvergence) will tend towards the CLB which holds the register state.

The M-HOT approach makes better decisions because integrated placement, routing, and scheduling has greater information about resource usage than approximate cost functions in segregated flows like M-CAD. However, it does increase runtime over M-CAD.

The top-level code of the algorithm is shown in Figure 3. The algorithm accepts a CDFG and computes an as-late-as-possible (ALAP) height for each node (operation). At the bottom at height 0 are the CDFG outputs. It processes each height, starting at the largest (which are always CDFG inputs), because those nodes have the longest path to the outputs at the bottom, so they are the most critical.

At each height, it computes an affinity matrix and performs a low-temperature anneal to assign each node to a CLB and the earliest timeslot that gives it the lowest cost. When annealing is complete, all nodes at the current height are locked so they cannot be moved, and the next height is annealed.

3.5.1 *Annealing*

At each height, the annealer assigns coarse-grained operations to CGs and fine-grained operations to LUTs in the FGs. The coarse-grained and fine-grained nodes are annealed together, so either can be a move candidate. After choosing a move, the annealer invokes

```
1: Compute ALAP height of each node
2: for $height = maxheight$ to 0 do
3:     $nodes \leftarrow$ all nodes at height=$height$
4:     $aff \leftarrow$ compute_affinity($nodes$)
5:     anneal($aff$, $nodes$)
6: end for
7: Finalize modulo routes
```

Figure 3: M-HOT Top-Level Code

the router to determine the earliest timeslot for the current operation in the chosen CLB. Coarse- and fine-grained routing delays are computed the same as the M-CAD approach.

If the operation is registered, it requires special handling. Not only are the routes computed from the source operations to this register, but unlike traditional operations the destination sinks will already have known locations and timeslots. Hence, the paths to them can be computed as well. They are known because a registered operation is always the terminus of a path, so it is always at height 0 in the ALAP tree. However, the sinks of a register are placed as some of the earliest operations. These routes are "modulo routes", wrapping around the schedule back to timeslot 0. The M-HOT approach does not target a fixed SL. Instead, it lengthens SL as needed. As a result, the links on these modulo routes may become broken as additional timeslots are added. The last step of the main loop in Figure 3 completes and reconnects these dangling routes.

At the heart of the annealer is a cost function. The annealing schedule is from VPR but with a lower initial temperature. The annealing cost function is from [25], but modified for the Malibu architecture to achieve better results for mapping circuits. The cost function for a node is:

$$\begin{aligned} cost =& producer\_cost + affinity\_cost + parallel\_cost \\ &+ register\_cost + penalty \end{aligned}$$

3.5.2 *Producer Cost*

The $producer\_cost$ is the cost of placing a CG operation in a certain CLB at a certain timeslot, or a LUT in a certain CLB. It uses actual routing information to compute the real cost (something the M-CAD approach can only approximate by using the Manhattan distance). The cost is the sum of the timeslot differences from each source to the current node.

This is the same cost as in [25], except as follows. When the CG operation has a register as a source, that source will not be placed until height 0 is processed. Yet, a placement is needed to compute this cost. In this case, to avoid spreading out the siblings of that source register and incurring lengthy fanout delays when the register is finally placed, the producer cost is the total Manhattan distance from the candidate CLB location to each of the already-placed siblings of the register.

3.5.3 *Affinity Cost*

The affinity cost keeps nodes with common descendants close together to reduce future routing costs. It is computed among all nodes at each height, and is calculated as described in [25]. Briefly, it weights the Manhattan distance between a pair of nodes at this height by the affinity weight between them. The affinity weight counts the common sinks between the nodes in future graph levels which are not yet placed. It looks up to 3 levels deep, with the common sink count being counted 4 times at level 1, twice at level 2, and once at level 3. Thus, two nodes with many common sinks in the very next level of the graph will be penalized by a higher affinity cost if they are placed too far apart.

3.5.4 *Parallel Cost*

This cost attempts to spread out nodes from the same level so they are placed in *different* CLBs. This cost simply counts the number of node-pairs placed into the same CLB. Without this cost, they tend to bunch up in the same CLB, requiring more timeslots to schedule. The implementation in [25] prefers to place nodes "on the left" of the array, since this is where I/O is located. For Malibu, it is better to force the nodes to spread out.

3.5.5 *Register Cost*

In a circuit with registers, two sinks of a register may be placed far apart since there is no cost to tie them together, in turn creating unnecessarily long routes and artificially inflating the SL. [25] avoids this problem by pre-placing all inputs, outputs, registers, and memories. If the circuit has a specific pin mapping this would be a reasonable approach for M-HOT too, however it is not always desirable to impose this restriction so two costs are added to the cost function: a cost to keep sinks of registers together, this was shown previously as the $producer\_cost$, and a cost for register path reconvergence.

To encourage reconvergence, a $register\_cost$ is computed to be the sum of the Manhattan distances between each node in the current height and the sinks of any registers in the node's respective fanout cone. This encourages the operation to be placed along the straight-line path between the source and eventual register location. While the registers themselves have not yet been placed, the sinks of those registers have been placed, so the Manhattan distance can be computed. The cost for sinks is exponentially weighted by powers of 2 based on depth in the same way as the affinity calculation.

3.5.6 *Penalty Cost*

The $penalty$ cost discourages invalid/illegal placements. It is computed the same way as the M-CAD approach.

3.5.7 *Routing*

Because placement, routing, and scheduling are done together with the M-HOT tool, there is no opportunity to incorporate the actual FG routing delays into the flow without of invoking VPR in the inner loop of the annealer.

Instead, the M-HOT tool estimates the FG routing delay based on Manhattan distance and Elmore delay, and calls VPR at the end of scheduling to compute the actual delay for each route. In all our benchmark trials we have found that even when the routing delay is underestimated (e.g. a route has to go around some CLBs to avoid congestion), it is close enough to the actual delay that the value will still arrive before it is needed. M-HOT flags any routes with timing violations and reports an error.

4. EXPERIMENTAL RESULTS

In the following sections, the M-CAD and M-HOT tools are evaluated on the Malibu architecture. For baseline comparison, the benchmarks were synthesized with QuartusII 10.0 for a StratixIII (EP3SL340F1760I4L) FPGA, and with VPR 5.0 using 65nm iFAR [33] architecture parameters (n10k04l04.fc15.area1delay1.cmos65nm). For VPR, the ten 4-LUT architecture was selected for area efficiency. The length-four wires in the architecture were changed to length-one wires without changing the delay characteristics to over-compensate for the CG area being added to each of the FG CLBs. The architecture was also modified to place the CLB pins only on the top or right of the CLB to mimic overhead routing.

In Section 4.1 the maximum user clock speed (F_{max}) is compared. In Section 4.2 synthesis time is compared to QuartusII. Sec-

Table 3: Benchmark Circuit Information

	QuartusII		VPR	Malibu			
Circuit	ALM	18×	4LUT	Nodes	Nets	%r	%1b
ethernet	6868	0	19626	9693	13686	15	61
fft16	6412	84	17006	2120	2236	29	0
wb_conmax	5349	0	16098	17917	23558	2	40
fft8	2075	28	5248	800	836	29	0
dma	1714	0	5071	18514	23650	6	41
ac97_ctrl	1254	0	3538	4911	6097	8	47
aes_core	1154	0	5021	3380	3970	1	8
tv80	850	0	2330	12186	16027	2	44
jpeg_enc	791	64	2836	4486	5882	11	13
systemcaes	716	0	2181	3043	3799	0	23
spi	488	0	987	664	856	6	37
des	298	0	865	4114	5497	0	34
systemcdes	237	0	650	1688	2131	0	24
pci_master	137	0	325	957	1342	8	71
me	5148	0	14388	5954	7020	14	0
chem	3526	175	36143	568	714	0	0
honda	1216	52	3795	249	293	0	0
dir	1150	8	6620	884	1190	6	22
mcm	1057	56	3067	232	288	0	0
wang	797	24	2275	134	152	0	0
pr	646	18	1893	176	194	0	0

tion Section 4.3 looks at the minimum-transistor-area required for each benchmark. And finally in Section 4.4 we show the tools can make tradoffs among F_{max}, compile time, and density.

To evaluate these metrics, a variety of Verilog benchmarks are used, summarized in Table 3. The **chem**, **dir**, **honda**, **mcm**, **pr**, and **wang** benchmarks [30] are dataflow– and DSP-style non-pipelined computational circuits described in behavioural Verilog; **me** is our own motion estimation design; **fft8** and **fft16** are our own deeply pipelined 8- and 16-point complex FFTs implemented using a radix-2, decimation-in-time decomposition; **jpeg_encoder** is from [32]. The other benchmarks are the 11 largest (in ALM count) from the IWLS 2005 benchmark set[6], excluding the ones with names of the form sXXXXX which appear to be the output of another synthesis tool since they are composed entirely of 1-bit logic gates with mangled names.

Table 3 shows the synthesized size of each benchmark using QuartusII for a StratixIII, VPR for the modified iFAR architecture, and Malibu. The 18× column is the number of 18×18 multipliers used. We were unable to use QuartusII to produce BLIF with hard multipliers, and OdinII's limited Verilog support would require a massive rewrite of our benchmarks, so the VPR results use LUTs exclusively. The %r column is the percent of nodes which are registered, and is important in the HOT approach where these nodes are all at height=0. The %1b column is the percent of nodes which output a single-bit value. These nodes are implemented on the fine-grained resources when $W_f \geq 1$. For some circuits, like **dma** and **tv80**, much fewer QuartusII ALMs than Malibu nodes are needed. This highlights the strong need for us to examine and improve the front-end logic synthesis in our flow. Since that is a major undertaking, we leave that for future work.

4.1 Maximum Frequency (F_{max})

Figure 4 graphs the F_{max} across all circuits for the baseline Quartus and VPR test, and for the M-CAD and M-HOT tools with $W_f = 1$. The data is fastest result of 10 trials, and for the Malibu results is the fastest result for all architecture sizes. The data, sorted by the Quartus F_{max}, allows us to highlight three trends. First, the QuartusII F_{max} is 10x higher on average than M-CAD with $W_f = 1$. Changing to $W_f = 4$ (not shown), M-CAD improves to an 8.4x gap. Alternatively, if we eliminate the FG LUTs completely ($W_f = 0$), the M-CAD F_{max} is 14x lower. We consider this a fairly good result for time-multiplexing.

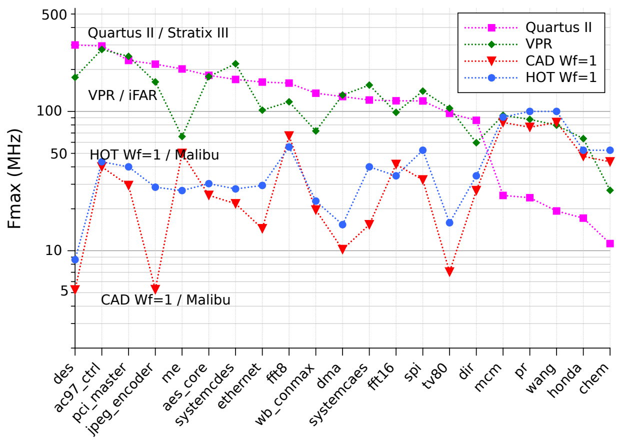

Figure 4: F_{max} results (ordered by QuartusII F_{max}.

Table 4: Synthesis Time Speedup vs. Quartus

	All Circuits	Circuits with fine-grained signals			
Tool	W_f=0	W_f=0	W_f=1	W_f=2	W_f=4
M-CAD	61.1x	20.1x	8.5x	8.6x	8.7x
M-HOT	42.0x	4.9x	12.1x	11.8x	12.1x

Second, the F_{max} results for the M-HOT tool are, on average, 45% higher than M-CAD across all values of W_f. Compared to Quartus the M-HOT tool F_{max} is 8.5x, 6.3x, and 4.8x lower for W_f=0, 1, 4 respectively. For time-multiplexed architectures, this indicates that this may be a better way to map circuits than following traditional CAD.

Third, the benchmarks towards the left of Figure 4 have a large percentage of fine-grained wires. FPGAs implement these very well, but the time-multiplexed Malibu architecture does not, averaging 15x slower (M-CAD, $W_f = 1$). However, on the right of the graph where the FPGA performs poorly are the coarse-grained benchmarks. These are the types of circuits the Malibu architecture is targeted for, and they show a 3x higher F_{max} over the Stratix III result.

4.2 Compile Time

Table 4 shows speedup, compared to Quartus, of the complete Verilog-to-bitstream compile time for the Malibu M-CAD and M-HOT toolflows.

Both Malibu CAD approaches show a significant speedup compared to Quartus ranging from 249x faster (chem) down to 1.3x faster (dma) for M-CAD. Malibu compile-time is designed to be fastest with coarse-grained-only circuits (fft16, me, chem, fft8, honda, mcm, wang, and pr). When removed, the overall speedup decreases as expected. Since these circuits have no fine-grained signals, setting $W_f > 0$ has no effect on results as the fine-grained handling code is never invoked.

For the remaining circuits, on average, there is still a speedup when fine-grained signals are implemented on the FG resources. However, the speedup range changes from 42x faster (spi) to almost 50x slower (wb_conmax). We consider wb_conmax to be anomalous because a traditional VPR route of wb_conmax takes 14 minutes, whereas VPR called by the Malibu to route the FG resources takes 5 hours. This further highlights the need for improvements to our front-end synthesis: the traditional VPR route uses Quartus-generated BLIF, whereas Malibu is extracting nearly the same number of FG nodes as the entire Quartus-optimized circuit.

For both the M-CAD and M-HOT flows, the compile-time

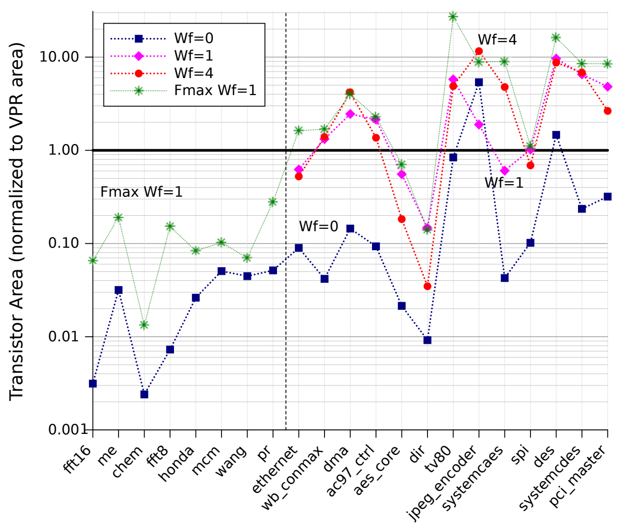

Figure 5: Area savings for the minimum-required-area compared to the F_{max} area. Normalized to the VPR area results.

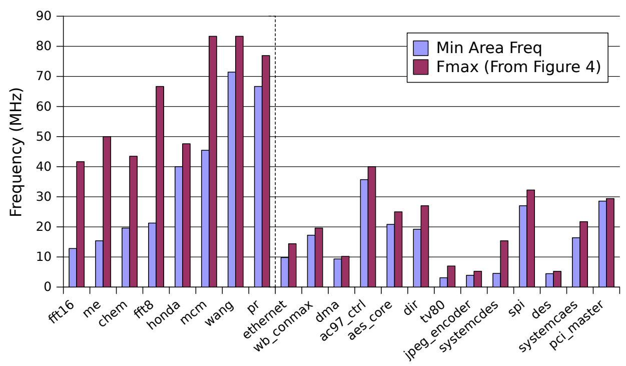

Figure 6: Frequency lost to achieve the minimum-area compared to the F_{max} for $W_f = 1$.

changes little as W_f increases. Once the fine-grained resources are invoked they require approximately the same time to process even though the quantity of FG logic increases. Also, M-CAD gets slower but M-HOT gets faster with $W_f > 0$ because M-HOT anneals fewer objects at each height as W_f increases.

4.3 Area

Figure 5 shows the transistor area, normalized to the VPR area result, of the smallest Malibu architecture size required to successfully synthesize each benchmark. It also shows the area required for the F_{max} results presented in Figure 4 for each benchmark. The Malibu area is the sum of the tile area (Tables 1 and 2) and the FG routing area as reported by VPR. In many cases, particularly for $W_f = 0$, the minimum area is achieved by time-multiplexing the circuit on just a few CLBs. However the tools are enforcing resource constraints so this is not possible for all circuits. The mapping is constrained by the instruction memory (256 instructions), R (64 plus 64 user data-memory entries), NSEW (16 entries), CGIs (4 per CLB), CGOs (16 per CLB), LUTs (16 per CLB), and circuit I/Os (one CG input and one CG output per CLB).

The benchmarks to the left of the vertical divider have no fine-grained elements so the result for $W_f = 1, 2, 4$ is the same as $W_f = 0$. For $W_f = 0$, on average, the required architecture is 15.5x smaller, with an average user clock speed of 20.1 MHz. This is expected because the smallest area is most often a 3x3 or 4x4

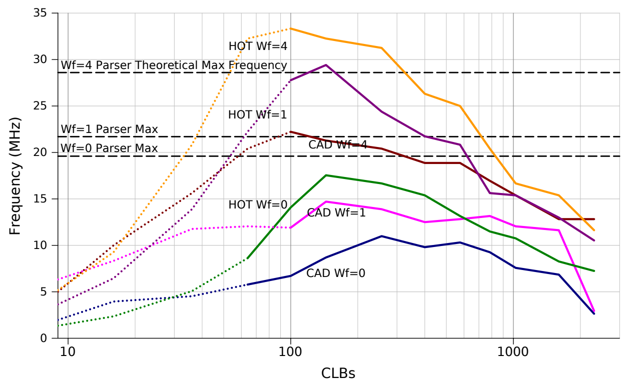

Figure 7: Frequency versus area tradeoff for the ethernet benchmark. M-HOT produces faster results than M-CAD. Increasing W_f also improves the frequency.

array of CLBs, and the Malibu CAD tools fold the operations in time to fill the instruction memory which is the limiting constraint.

For $W_f = 1, 2, 4$ (circuits to the right of the vertical divider), the minimum required area is, on average, 1.8x, 1.5x (not shown), and 1.8x larger than the VPR area results. The limiting constraint in these cases was the number of LUTs per CLB, which was set to 16. More FG resources per CLB would help in this case, so would a heterogeneous architecture where some CLBs do not have the CG component, and so would a better CG/FG partitioning strategy and better front-end logic synthesis. The Malibu CAD tools will enable such architecture and tool exploration in future work.

For comparison, Figure 5 also shows the area for the F_{max} results presented in Figure 4 for $W_f = 1$. This architecture size is always larger than the minimum size. On average, the maximum frequency is 1.4x higher, but uses 3.1x the area of the minimum architecture size. This demonstrates the tool's ability to trade area for performance, this is further explored in the next section.

Figure 6 shows the F_{max} from Figure 4 compared to the minimum-area frequency for $W_f = 1$. The F_{max} is, on average, 1.4x larger. Again, to the left of the divide are benchmarks with no fine-grained elements, so W_f is actually zero. There is a larger frequency difference for these circuits because the CGI, CGO, and LUT constraints are removed, and the tools can time-multiplex more aggressively at the expense of speed (only really constrained by instruction memory). When the FG resources are used (to the right of the divider) the additional constraints (CGIs, CGOs, and LUTs) increase minimum area required to synthesize. This pushes the area closer to the architecture size where the F_{max} is achieved, resulting in the higher frequency. For future work we plan to investigate the ratio of CG and FG resources in each CLB, as this result indicates more FG resources may be beneficial in some cases.

4.4 Feature Evaluation

This section examines some unique features of the Malibu tool flow. Figure 7 shows the user frequency of the **ethernet** benchmark for the M-CAD and M-HOT approach for W_f =0,1,4 over a range of architecture sizes from 3x3 CLBs to 45x45 CLBs. For each size, the tools are forced to use all available CLBs, hence the decrease in performance on larger architectures once communication delay dominates the schedule. The tools would not normally be run in this mode unless fitting a large circuit on a small architecture. Usually the tools would be allowed to find the best architecture size by allowing some CLBs to remain empty. The dotted line represents the range of architectures where constraints were violated in the final result, so the synthesized circuit is not viable given our architectural settings.

This graph shows the Malibu tools can trade density (number of CLBs) for speed by targeting any sized architecture and time-multiplexing more code (or less) on the CGs. This is useful for fitting a large design in a small architecture.

The graph also shows the tradeoff involving W_f. Increasing W_f from 0 to 1 for both the M-CAD and M-HOT flows, causes the frequency to also increase. It also causes the peak F_{max} to require slightly fewer CLBs, meaning density is increased. This result is expected, because the fine-grained control logic is moved to the FG resources where it can be computed and distributed more quickly.

It is possible to estimate a theoretical maximum frequency of a circuit using the graph-depth of the CDFG after parsing by assuming each node in the CDFG takes one system clock cycle and that communication is free. These maximums are shown on Figure 7 for the ethernet benchmark for each value of W_f. For $W_f = 0$ M-CAD is achieves 56% of the maximum and M-HOT 89%. M-HOT exceeds the maximums at $W_f = 1$ and 4 because of chains of fine-grained operations are being optimized and synthesized by OdinII and ABC into LUTs, reducing the number of operations along the critical path after parsing. Because of this, the important metric is the $W_f = 0$ value where no optimizations are applied. Averaging across all circuits, M-CAD achieves 35% of the post-parsing maximum frequency, and M-HOT 52%. Therefore, the performance of M-HOT could be, at most, doubled without using optimizations or a better parsing/CDFG construction tool.

5. LIMITATIONS AND FUTURE WORK

Moving forward, we plan to remove limitations of the Malibu tool flow like the restriction of a single clock domain. There are also promising scheduling approaches like the edge-centric modulo scheduler proposed by Park et al. [27] that could further improve circuit speed or density and keep compile time low.

VPR takes several hours to route the fine-grained parts of the wb_conmax, fft16, and tv80 benchmarks from within the Malibu framework. All other benchmarks route within a few seconds from within the framework. By itself, VPR can place and route those benchmarks in under 14 minute with input generated by Quartus (see Figure 2a). Despite this long runtime, VPR still produces a high-quality result. This is something that requires investigation and could result in improvements to VPR or the fine-grained netlist generator in the Malibu tools.

There are two avenues for future work to improve the quality of the results. Figure 7 shows that M-HOT achieves 89% of the theoretical maximum frequency based on the parser output graph depth. Overall, the M-HOT approach is 52% of this maximum, so better placement, routing, and scheduling algorithms can only close this gap. Adding optimizations is one way to reduce the graph depth and increase the frequency. The other option is to replace Verilator with a commercial-quality front-end synthesis tool. Verilator was chosen because it is easy to extract a coarse-grain CDFG from parsed and elaborated Verilog. These hooks need to be added to other synthesis software.

Having a CAD flow is essential to enable architecture exploration. We also plan to test various architectural parameters like the size of the memories, the number of LUTs per CLB, and heterogeneous architectures involving multipliers, CLBs with no CGs, and CLBs with large memories, to try and reduce the required transistor area while still maintaining a high-quality synthesis result and a fast compile time. Also, the limitations of our front-end logic synthesis needs to be investigated, as we are producing graphs with significantly more nodes (and larger depth) than is likely needed.

6. RELATED WORK

Synthesizing a circuit for an FPGA is a well researched and understood problem. In this work we make use of academic tools T-VPack+VPR [1], ABC [2], and ODIN II [13], as well as QuartusII, a commercial tool.

Mapping to a CGRA also has many academic solutions, e.g. [18, 35]. These tools are designed to map software loop "kernels" from sequential *programs* into the CGRA, not HDLs. CGRA architectures are usually controlled by a host processor, and have access to global memory. When mapping *circuits* to an FPGA with coarse-grained resources, there is no host processor or global memory. Other novel CGRA scheduling solutions include reconvergent scheduling [19], DRESC [21], SPR [9], and modulo graph embedding [26]. These solutions all use iterative algorithms to achieve a high-quality mapping solution. Malibu seeks very fast compiles with some loss of quality, so iterative approaches are incompatible with our runtime objective.

Many CGRA architectures exist which time-multiplex coarse-grained resources (e.g. ADRES [22], PipeRench [10], Tartan [23], RaPiD [8], SCORE [3]). In contrast, this work offers an approach for implementing *circuits*, not programs. Unlike other work, we integrate LUTs into the architecture to implement fine-grained signals. No resources are used to implement a C or C-like programming model (e.g., no branch instructions or global memory). There is ALU support for HDL operations like bit concatenation, unary logic reduction, and automatic truncation of results.

The tradeoff between density and circuit speed in time-multiplexed architectures was first demonstrated for fine-grained FPGAs with VEGA [14] and later with TSFPGA [7]. TSFPGA also added a modulo scheduling refinement, which we do not yet implement. However, in this work we time-multiplex the coarse-grained resources, not the fine-grained resources.

Datapath-oriented FPGA research has investigated logic configuration bit-sharing [4], as well as interconnect configuration-bit sharing and bus-based multiplexers [34]. However, this research has not looked at time-multiplexing.

7. CONCLUSIONS

Modern FPGAs implement a wide range of circuits which have both coarse-grained and fine-grained elements. Great demand is on CAD tools to synthesize these ever-larger circuits faster, and without loss in quality. To address these issues, we have proposed adding coarse-grained time-multiplexed resources to the FPGA CLB to create Malibu, a new FPGA architecture.

To study the tradeoffs of new architecture, we have developed a full CAD flow which fully compiles Verilog2005 into a configuration bitstream. Two physical mapping approaches were demonstrated: M-CAD is based on the traditional FPGA CAD tool flow with scheduling at the end, and M-HOT is based on a CGRA scheduler for simultaneous placement, routing, and scheduling.

For M-CAD we demonstrate compile-time speedups of up to 249x (61x average) compared to QuartusII synthesizing for a StratixIII FPGA. However, due to time-multiplexing and current synthesis limitations we achieve a circuit speed 14x slower on average. The M-HOT synthesis takes longer, only 42x faster than Quartus on average, but the final circuit speed is improved to only 8.5x slower. Enabling the fine-grained resources improves the final circuit speed, and this slows compile-time for M-CAD but speeds it up for M-HOT.

Finally, we demonstrate up to 15.5x savings in transistor area utilizing the tool's ability to trade density for the final circuit speed, and time-folding the circuit as much as possible onto a small target architecture.

It is difficult to compare tools such as this with more mature work such as VPR and Quartus, but the architecture and tools show promising results. Most importantly, they show it is possible to achieve fast synthesis and good performance on the Malibu architecture. This is really the starting point for significant additional future work, where various algorithms, heuristics, and architectural features can be optimized and improved. In particular, we believe significant gains in F_{max}, runtime, and density can all be obtained.

8. ACKNOWLEDGMENTS

This research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC). Equipment donations by CMC Microsystems are gratefully acknowledged. The authors would also like to thank Deming Chen, Russell Tessier, and Graeme Smecher for providing several benchmark circuits, as well as the authors and the many additional contributors to the various open-source tools used in this research. Verilator: Wilson Snyder, Duane Galbi, and Paul Wasson. OdinII: Peter Jamieson, Kenneth B. Kent, Farnaz Gharibian, and Lesley Shannon. VPR: Vaughn Betz, Jonathan Rose, and Alexander Marquardt. ABC: Alan Mishchenko.

9. REFERENCES

[1] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In *Proc. FPL*, pages 213–222, 1997.

[2] R. Brayton and A. Mishchenko. ABC: An Academic Industrial-Strength Verification Tool. In *Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 24–40. 2010.

[3] E. Caspi, M. Chu, R. Huang, J. Yeh, J. Wawrzynek, and A. DeHon. Stream Computations Organized for Reconfigurable Execution (SCORE). In *FPL*, pages 605–614, 2000.

[4] D. Cherepacha and D. Lewis. DP-FPGA: An FPGA architecture optimized for datapaths. *VLSI Design*, 4(4):329–343, 1996.

[5] K. Choi and M. Song. Design of a high performance 32x32-bit multiplier with a novel sign select booth encoder. In *Proc. International Symposium on Circuits and Systems*, pages 701–704, May 2001.

[6] Christoph Albrecht. IWLS 2005 Benchmarks. [Online]. Available: http://www.iwls.org/iwls2005/benchmarks.html, 2005.

[7] A. DeHon. *Reconfigurable architectures for general-purpose computing*. PhD thesis, Massachusetts Institute of Technology, 1996.

[8] C. Ebeling, D. C. Cronquist, and P. Franklin. RaPiD - reconfigurable pipelined datapath. In *Proc. FPL*, pages 126–135, 1996.

[9] S. Friedman, A. Carroll, B. Van Essen, B. Ylvisaker, C. Ebeling, and S. Hauck. SPR: an architecture-adaptive CGRA mapping tool. In *Proc. FPGA*, pages 191–200, 2009.

[10] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer. PipeRench: A coprocessor for streaming multimedia acceleration. In *ISCA*, pages 28–39, 1999.

[11] D. Grant, G. Smecher, G. G. Lemieux, and R. Francis. Rapid synthesis and simulation of computational circuits in an MPPA. In *Proc. FPT*, pages 151–158, Dec. 2009.

[12] T. R. Halfhill. Tabula's time machine. *Microprocessor Report*, Mar. 2010.

[13] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon. Odin II - An Open-source Verilog HDL Synthesis tool for CAD Research. In *Proc. FCCM*, pages 149–156, 2010.

[14] D. Jones and D. Lewis. A time-multiplexed FPGA architecture for logic emulation. In *Proc. Custom Integrated Circuits*, pages 495–498, 1995.

[15] M.-E. Jones. 1T-SRAM-Q quad-density technology reins in spiraling memory requirements. http://csserver.evansville.edu/˜mr56/cs838/Paper16.pdf, retrieved Sept 2010.

[16] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: applications in VLSI domain. *IEEE Trans. VLSI*, 7(1):69–79, Mar 1999.

[17] S. K. Lai. Flash memories: Successes and challenges. In *IBM Journal of Research and Development*, volume 52, pages 529 –535, Jul. 2008.

[18] J.-e. Lee, K. Choi, and N. D. Dutt. Compilation approach for coarse-grained reconfigurable architectures. *IEEE Des. Test*, 20(1):26–33, 2003.

[19] W. Lee, D. Puppin, S. Swenson, and S. Amarasinghe. Convergent scheduling. In *Proc. MICRO*, pages 111–122, 2002.

[20] A. Marquardt, V. Betz, and J. Rose. Timing-driven placement for FPGAs. In *Proc. Field Programmable Gate Arrays*, pages 203–213, 2000.

[21] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins. DRESC: a retargetable compiler for coarse-grained reconfigurable architectures. *Proc. FPT*, pages 166–173, 2002.

[22] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins. ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In *Proc. FPL*, pages 61–70, 2003.

[23] S. C. Mishra, Mahim; Goldstein. Virtualization on the Tartan reconfigurable architecture. In *FPL*, pages 323–330, 2007.

[24] K. Nii *et al.* A 65 nm ultra-high-density dual-port SRAM with 0.71um2 8T-cell for SoC. In *VLSI Circuits*, pages 130 –131, 2006.

[25] H. Park. *Polymorphic Pipeline Array: A Flexible Multicore Accelerator for Mobile Multimedia Applications*. PhD thesis, The University of Michigan, 2009.

[26] H. Park, K. Fan, M. Kudlur, and S. Mahlke. Modulo graph embedding: mapping applications onto coarse-grained reconfigurable architectures. In *Proc. CASES*, pages 136–146, 2006.

[27] H. Park, K. Fan, S. A. Mahlke, T. Oh, H. Kim, and H.-s. Kim. Edge-centric modulo scheduling for coarse-grained reconfigurable architectures. In *Proc. PACT*, pages 166–176, 2008.

[28] J. Singh, D. Aswar, S. Mohanty, and D. Pradhan. A 2-port 6T SRAM bitcell design with multi-port capabilities at reduced area overhead. In *ISQED*, pages 131 –138, Mar. 2010.

[29] W. Snyder. Verilator-3.652, June 2007.

[30] M. B. Srivastava and M. Potkonjak. Optimum and heuristic transformation techniques for simultaneous optimization of latency and throughput. *IEEE Trans. VLSI*, 3(1):2–19, 1995.

[31] S. Trimberger, D. Carberry, A. Johnson, and J. Wong. A time-multiplexed fpga. In *Proc. FCCM*, pages 22–28, 1997.

[32] University of Massachusetts. UMass RCG HDL Benchmark Collection. [Online]. Available: http://www.ecs.umass.edu/ece/tessier/rcg/benchmarks, 2006.

[33] University of Toronto. iFAR - intelligent FPGA Architecture Repository. [Online]. Available: http://www.eecg.utoronto.ca/vpr/architectures, 2008.

[34] A. Ye and J. Rose. Using bus-based connections to improve field-programmable gate array density for implementing datapath circuits. In *IEEE Trans. VLSI*, pages 3–13, 2005.

[35] J. W. Yoon, A. Shrivastava, S. Park, M. Ahn, R. Jeyapaul, and Y. Paek. SPKM: a novel graph drawing based algorithm for application mapping onto coarse-grained reconfigurable architectures. In *Proc. ASP-DAC*, pages 776–782, 2008.