High Performance Instruction Scheduling Circuits for Out-of-Order Soft Processors

Henry Wong, Vaughn Betz, and Jonathan Rose
Department of Electrical and Computer Engineering, University of Toronto
{henry, vaughn, jonathan.rose}@ece.utoronto.ca

Abstract—**Soft processors have a role to play in easing the difficulty of designing applications into FPGAs for two reasons: first, they can be deployed only when needed, unlike permanent on-die hard processors. Second, for the portions of an application that can function sufficiently fast on a soft processor, it is far easier to write and debug single-threaded software code than to create hardware. The breadth of this second role increases when the performance of the soft processor increases, yet there has been little progress in the performance of soft processors since their commercial inception — in particular, the sophisticated out-of-order superscalar approaches that arrived in the mid 1990s are not employed, despite the fact that their area cost is now easily tolerable. In this paper we take an important step towards out-of-order execution in soft processors by exploring instruction scheduling in an FPGA substrate. This differs from the hard-processor design problem because the logic substrate is restricted to LUTs, whereas hard processor scheduling circuits employ CAM and wired-or structures to great benefit. We discuss both circuit and microarchitectural trade-offs, and compare three circuit structures for the scheduler, including a new structure called a *fused-logic matrix scheduler*. With this circuit, large schedulers up to 40 entries can be built with the same cycle time as the commercial Nios II/f soft processor (240 MHz). This careful design has the potential to significantly increase both the IPC and raw compute performance of a soft processor, compared to current commercial soft processors.**

I. Introduction

The design effort required to build large modern FPGA systems has become a key focus of the industry. Many of the approaches to reduce design time take the form of transforming software directly into hardware. An alternative is to simply implement that software on a processor, and the modern hard processors in FPGAs can take on some of that role. However, various subsystems may require their own processor for performance, security or design isolation reasons, and the limited number of hard processors may not suffice. In that case, the ability to deploy a soft processor is important, and the performance of the soft processor is key to determining how much of the subsystem can be implemented in software. High performance soft processors may be a better vehicle to attach custom-hardware accelerators to, given their inherent flexibility.

Despite this, there are still no commercial out-of-order superscalar soft processors, yet there is clear evidence from the hard processor arena that the move to out-of-order results in a significant performance increase. This is illustrated in Table I, which provides SPECint scores between pairs of historical hard processor architectures that moved from in-order to out-of-order microarchitectures. The ratio of each pair of performance numbers in that table are normalized to the same operating frequency to isolate instructions per cycle (IPC) from clock frequency improvements. The table shows that performance increases by a factor of 1.6 to 2 times moving to out-of-order. This performance improvement largely arises from exploiting instruction-level parallelism and tolerating the multicycle latency of memory operations.

If these cycle-count performance gains can be obtained without sacrificing operating frequency (f_{max}), then soft processors can achieve significant performance gains. Prior academic work in this arena often failed to achieve reasonable f_{max} [3]–[5].

This paper focuses on a key component of an out-of-order processor, the instruction scheduler, and explores the microarchitecture and design of scheduler circuits that yield high IPC and large gains in f_{max} operating frequency. The closest prior work has shown that out-of-order instruction schedulers on an FPGA can be built at reasonable f_{max} [6]. Our new circuit designs improve on these earlier results, achieving 60% greater f_{max} for the same size of scheduler.

This paper begins with an overview of instruction scheduling trade-offs in Section II followed by a description of the classical scheduling circuits in hard processors in Section III. Section V discusses FPGA circuit designs that are evaluated in Section VII. Section VIII describes how this work can be extended to multiple-issue schedulers, as well as further optimizations.

II. Review of Instruction Scheduling in Out-of-Order Processors

The key attribute of out-of-order processors is that they execute instructions in *dataflow* order (based on data dependencies) rather than program order. In typical processor

Vendor	SPECint	In-order		Out-of-Order		Ratio
MIPS [1]	95	R5000 180 MHz	4.8	R10000 195 MHz	11.0	2.1
Alpha [1]	95	21164 500 MHz	15.0	21264 500 MHz	27.7	1.9
Intel [1]	95	Pentium 200 MHz	5.5	Pentium Pro 200 MHz	8.7	1.6
Intel [2]	2006	Atom S1260 2 GHz	7.4	Atom C2730 2.6 GHz	15.7	1.6

Table I
Comparison of SPECint scores between in-order and out-of-order processors and frequency-normalized ratio

pipelines, this dataflow ordering occurs after the instructions are fetched, decoded, and register renamed in program order. They are then inserted into the instruction scheduler, which executes instructions as they become ready. Instructions leave the scheduler when completed. Finally, completed instructions are committed in program order.

The instruction scheduler is responsible for tracking the readiness of every not-yet-completed instruction and for choosing which ready instruction should be executed each cycle. An instruction is ready to execute when all of its source operands are available, having been computed by previously executed instructions.

An instruction scheduler holds a pool of instructions that have not yet executed which are waiting to be executed. The *wakeup* portion of the scheduler is responsible for determining when a waiting instruction is ready for execution. It does this by observing which instructions are completing in each cycle and comparing their outputs with the required inputs for each waiting instruction. The *selection* logic is responsible for selecting one of the ready instructions for execution.

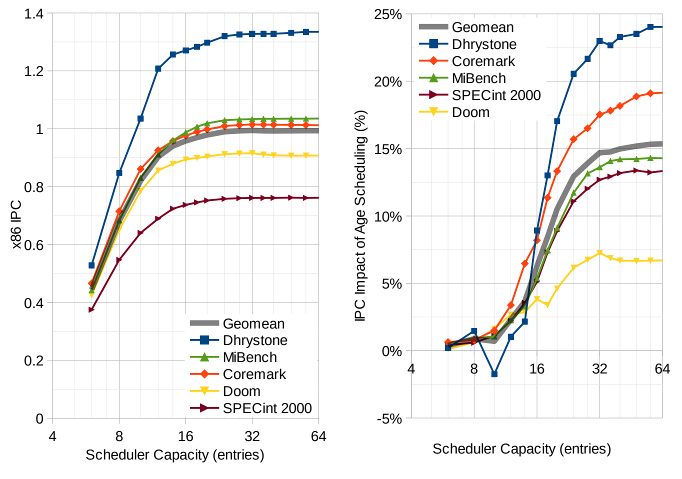
A. *Scheduler Trade-offs*

Processor design is all about trading off IPC, f_{max}, and design complexity. Here we discuss three major design decisions that affect this trade-off.

First, the number of scheduler entries affects how far ahead in the instruction stream instructions can be considered for execution. A small number of entries limits the ability to extract instruction-level parallelism (ILP) whereas larger schedulers (with more entries) increase ILP and IPC, but require more area and tend to have lower f_{max}. For example, Figure 1(a) shows how IPC improves with scheduler size on the system we explore in this paper. It shows that more scheduler entries eventually give diminishing returns — this is because other parts of our processor limit the number of in-flight instructions to 64 (reorder buffer size). The figure also shows that there is severe IPC loss with small schedulers of less than 16 entries.

Second, the selection policy — how to decide which of several ready instructions should execute — has an impact on IPC: choosing the oldest instruction first is a known good heuristic as it is more likely that an older instruction blocks execution of later dependent operations. However, an oldest-first heuristic requires tracking the age of entries in the scheduler, which has a hardware cost. Figure 1(b) shows the impact of an oldest-first selection policy compared to random selection. The IPC impact is small for small schedulers because the chance of having more than one ready instruction is lower, but the impact grows to over 15% for large schedulers. Prior out-of-order processors have mostly employed age-based selection selection [7]–[10].

The third key decision is whether wakeup and selection operations complete in a single cycle, which allows execution of dependent operations in consecutive cycles. Without back-to-back execution of dependent instructions, a processor will suffer a roughly 10% IPC penalty for adding just one



(a) IPC vs. Scheduler Capacity (b) Benefit of Age Priority vs. Random

Figure 1. IPC sensitivity to scheduler capacity and age-based selection policy. The simulated processor has 1 each of branch, ALU, AGU, and store-data execution units, and a peak IPC of 2.

extra cycle [11], [12]. Back-to-back execution of dependent instructions does make circuit timing challenging, however.

In this work, which focuses on fast circuits for high-performance soft processors, we make the following two up-front design decisions: 1) a requirement of single cycle wakeup and 2) an oldest-first selection policy (although we will measure the impact of omitting this for one case). For all scheduler designs we explore, we will measure the impact of a wide range of the number of entries.

III. Background on Scheduler Circuits

As described above, schedulers have two key components: wakeup logic to determine which instructions are ready and selection logic to choose among the ones that are ready to execute in the next cycle. In this section we describe how classical hard processor *CAM*-based and *matrix* [13] schedulers do these two functions.

A. *Wakeup Logic*

CAM-based schedulers track operand dependencies using physical register numbers (after register renaming). Each entry in the scheduler's wakeup array holds an instruction's two source operand register numbers and two comparators that compare them to the destination register number of instructions completing each cycle. A source operand is available after its register number has been broadcast on a result bus, and an instruction is ready when all source operands are ready.

Matrix-based schedulers track dependencies by the position of producer instructions in the scheduler. Each entry (row) of the wakeup array contains a bit vector indicating which *instructions* in the scheduler will produce the source operands. The result bus bit vector indicates which *instructions* are granted execution each cycle, and an instruction is ready when all producer *instructions* have completed. This arrangement uses wired-OR gates instead of comparators to compute when each entry is ready. Grant signals broadcasting vertically to the horizontal wired-OR gates with an SRAM cell at each intersection results in a circuit that resembles a matrix.

B. Selection Logic

The selection logic is responsible for choosing one instruction for execution from a set of ready instructions. The simplest and fastest selection logic uses fixed priority, prioritizing instructions based only on an instruction's position in the scheduler. However, age-based selection heuristics are better than random selection for IPC (Section II-A). Age-based selection can be achieved by maintaining age ordering of scheduler entries and compacting holes so that scheduler position corresponds to age, or allowing random ordering of instructions in the scheduler and augmenting the selection logic with age information.

Compacting schedulers insert new instructions at the top, and shift scheduler entries down to fill holes left behind by instructions that have completed execution. Compaction allows a fast fixed-priority selection circuit to be used. The main drawback is in the power consumption of shifting the scheduler entries and the delay of the multiplexer required for shifting.

The alternative of explicitly tracking instruction age makes selection logic more complicated due to dynamic priority. There are many methods to track age, including precise and approximate methods (e.g., [7], [9]). Our matrix scheduler uses age matrices, a precise method that uses a matrix where the bits in each row indicate which instructions are older than the instruction occupying the row [14].

Hybrids approaches have also been used, such as the Alpha 21264 that uses a compacting scheduler that tracks register numbers, but uses wired-OR dynamic logic instead of comparators [8].

IV. Scheduler Circuits on FPGAs

FPGA logic is composed of LUTs and wires, while custom CMOS has much more flexibility in implementation. Unfortunately, matrix schedulers rely heavily on dynamic logic and wired-OR circuits with dense, regular layouts, so matrix-style schedulers become less appealing on FPGAs. However, its LUT-based matrix circuits can still be optimized.

Instruction schedulers are usually implemented with separate wakeup and select circuits, performed sequentially. For matrix schedulers on FPGAs, the wakeup logic's wide OR gate reductions and the selection logic's (conceptually) linear pick-first-ready scan logic are both implemented as trees of LUTs. In some circuits, it is possible to reformulate the logic function to combine two reduction or scan operations into one, improving delay. The sum-addressed decoder is one well-known example of this kind of transformation [15].

Inspired by this strategy, we present a new scheduler circuit, the fused-logic matrix scheduler, that combines both the wakeup wide-OR and select linear scan operations into a single tree of LUTs. This circuit is faster than both the CAM and age-based matrix schedulers for most scheduler sizes.

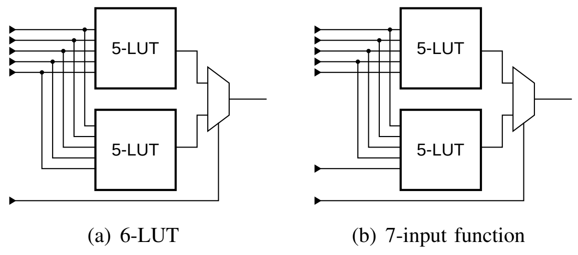


Figure 2. Altera ALMs can implement some 7-input functions

Before we discuss detailed circuit implementations in the next section, we first explain the 7-input mode of Altera Adaptive Logic Modules (ALMs), which we use in several of our circuits.

The Stratix IV ALM contains an 8-input fracturable 6-LUT. Although it is mainly intended to allow fracturing into two smaller LUTs (e.g., two independent 4-LUTs), the ALM can also implement 7-input functions that can be expressed as a 2-to-1 multiplexer selecting between two 5-input LUTs sharing 4 inputs (Figure 2). Having LUTs that implement logic functions with more inputs can reduce logic depth. Priority multiplexers and our new fused select-and-wakeup logic are mapped to 7-input functions that fit into an ALM.

V. Detailed Circuit Designs

This section discusses the circuit designs of the three scheduler circuits we implemented on the Stratix IV FPGA: a compacting CAM scheduler, a non-compacting matrix scheduler, and our new fused-logic matrix.

A. CAM

Our CAM scheduler implementation uses compaction to maintain age ordering and allows back-to-back scheduling of dependent operations. In each cycle, ready bits are used to select an instruction for execution. The selected instruction's destination tag is then broadcast on the result bus, and consumers of the newly-produced register are woken up.

1) Wakeup: Each entry in the CAM wakeup logic has two source operand tags and an associated pair of comparators. The comparators monitor the result bus for a physical register number that indicates when an operand becomes available. An instruction is ready when all operands are available and has not already been selected for execution. The register number is assumed to be large enough to hold at least twice the scheduler capacity, so comparators compare two $log_2 N + 1$ bit numbers. Each 6-LUT can do three bits of a comparison, which is followed by an AND tree, so the total logic depth for two comparators is roughly $log_6(4(log_2 N + 1))$. The ready bit of every entry, forming a *ready vector*, is sent to the selection logic.

2) Compaction: Our CAM wakeup logic can shift down to eliminate up to one hole per clock cycle. This is enough because only one instruction can be selected for execution each cycle, so new holes are created no faster than one

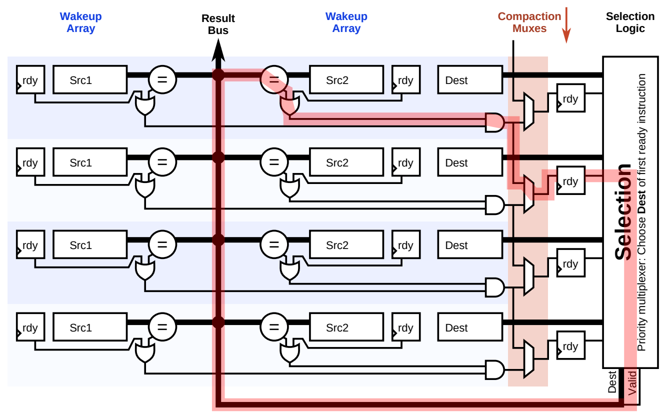


Figure 3. CAM Wakeup Circuit. Entries compact downwards. An example critical path is highlighted in red.

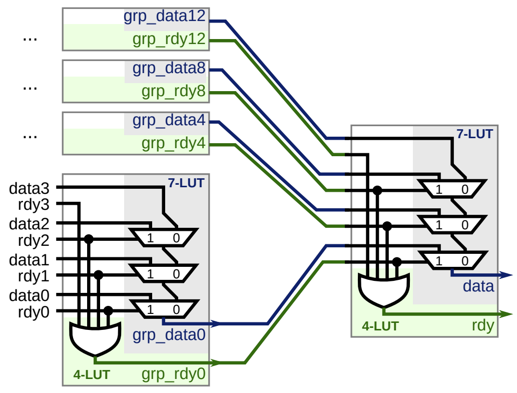


Figure 4. Priority multiplexer built from radix-4 blocks, each of which is a size 4 priority multiplexer. This figure shows how a depth-2 tree can implement a 16-entry priority multiplexer. **grp_data** and **grp_rdy** fit in a single 7-input and 4-input LUT, respectively. Used for CAM selection and fused-logic matrix selection.

per cycle. Compacting by one position occurs through 2-to-1 multiplexers immediately before the set of pipeline registers.

The control logic to decide whether each entry should shift down is a prefix OR operation, computing for each entry whether there is a vacant entry at or below the current position. This prefix OR function is implemented using a tree of LUTs with logic depth $log_6(N)$ using a radix-6 Han-Carlson prefix tree with sparsity 6 [16]. The radix and sparsity were chosen to suit a 6-LUT FPGA architecture, rather than the more typical radix-2 used in custom CMOS designs. This is much faster than a naïve implementation that uses a linear chain of 6-input OR gates with depth $(N-1)/5$.

3) Selection: The CAM scheduler's selection logic performs two functions. It must grant execution to the oldest ready instruction, and it must also select that instruction's destination register number and broadcast it on the result bus to wake up dependent operations.

One grant signal per entry indicates whether that entry has been selected for execution. Oldest-ready grant logic is implemented using the same radix-6 Han-Carlson prefix tree used for computing the wakeup compaction multiplexer control signals.

Generating the destination register is done with a priority multiplexer that selects the destination register number field of the oldest ready instruction. The priority multiplexer has a logic depth of $log_4 N$ LUTs, implemented as a radix-4 tree using 7-input ALMs. Figure 4 shows this circuit.

B. Matrix

The matrix scheduler implementation tracks dependencies of instructions using a wakeup matrix of dependency bits. We evaluated the matrix scheduler both with and without age-based selection. Age-based selection tracks the age of each entry using an age matrix, without compaction. In each cycle, the ready bits (and age matrix) are used to select an instruction for execution, and grant signals are broadcast into the wakeup matrix to wake up dependent instructions.

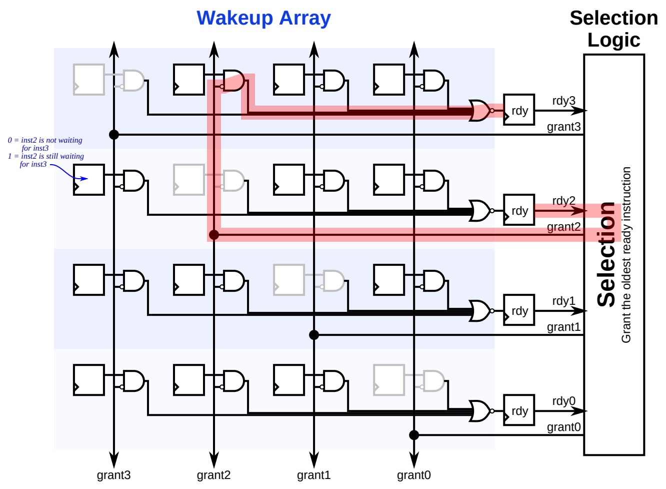

Wakeup Array
Selection Logic
0 = inst2 is not waiting for inst3
1 = inst2 is still waiting for inst3
rdy
rdy3
grant3
rdy2
grant2
rdy1
grant1
rdy0
grant0
Selection
Grant the oldest ready instruction
grant3
grant2
grant1
grant0

Figure 5. Matrix wakeup circuit. Diagonal is omitted as an instruction does not depend on itself. An example critical path is shown.

1) Wakeup: The wakeup array consists of a matrix of dependency bits. Each row corresponds to an instruction in the scheduler, and the bits in each row indicate which other instructions must execute before this one may do so. These bits are eventually cleared by the grant signals of the parent instructions when they execute. When all of the bit positions were ready or just granted, the ready bit for the row is set, resulting in a N-wide NOR of required-and-not-granted functions. An N-wide NOR of two-input functions ($2N$ inputs) can be computed with a tree of 6-input LUTs with depth $log_6(2N)$.

2) Position-Based Selection: The position-based select logic grants a ready instruction if there are no other ready instructions before it. As scheduler position does not correlate with instruction age, position priority is essentially random priority. It is implemented as a prefix OR function using the same radix-6 Han-Carlson prefix tree as found in the CAM

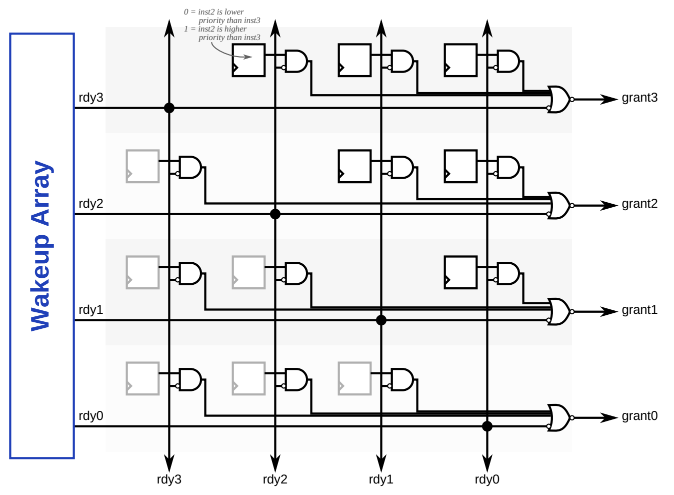

0 = inst2 is lower priority than inst3
1 = inst2 is higher priority than inst3
Wakeup Array
rdy3
rdy2
rdy1
rdy0
grant3
grant2
grant1
grant0
rdy3
rdy2
rdy1
rdy0

Figure 6. Age matrix selection circuit. An entry is granted if it is ready and the grant isn't "killed" by a higher-priority grant. Lower triangle registers are omitted as it is the complement of the upper triangle.

compaction control (Section V-A2) and CAM selection grant logic (Section V-A3). It is simpler and has higher frequency than the age-based selection logic, but with lower IPC.

3) Age-Based Selection: The age-based selection logic uses an age matrix to dynamically specify age priority, as the scheduler entries are not ordered by age. An age matrix specifies for each row which other instructions are older than itself. When a new (youngest) instruction is inserted into the scheduler, its corresponding row in the age matrix is set to 1 to indicate that every other instruction is older, and its corresponding column is cleared to 0 to indicate to every other instruction that the newly-inserted instruction is younger than it. A ready instruction is granted execution if there are no older ready instructions, which is a N-wide NOR of ready-and-older functions. This is computed with a radix-6 tree with logic depth $log_6 2N$, shown in Figure 6. We note that the age matrix has symmetry (if instruction A is older than B, then B must be younger than A), so we omit half of the matrix to reduce area.

Compared to the compacting CAM scheduler, the 2-to-1 compaction multiplexer and radix-4 priority multiplexer are removed from the critical loop. Dynamic-priority grant logic is slower than fixed-priority grant logic, with depth $log_6(2N)$ rather than $log_6(N)$. The CAM and matrix wakeup delays scale differently with scheduler size, favouring matrix wakeup for small sizes, but CAM wakeup for large sizes.

C. Fused-Logic Matrix

As noted in Section IV, matrix schedulers were originally formulated for dynamic wired-OR logic, which, in the FPGA context, have to be replaced with trees of LUTs. With separate wakeup and selection circuits, both the CAM and matrix schemes contained two such reduction trees of LUTs in their critical path. In the CAM scheduler, the operand register number comparators reduce the many bits of both operand comparisons down to a single ready bit (wakeup), and the selection logic reduces a vector of ready bits down to a single destination register number for the granted instruction (selection). In the matrix scheduler, a LUT tree reduces one row of the wakeup matrix down to one ready bit (wakeup), and the selection logic reduces a vector of ready bits and one row of age matrix down to a single grant signal (selection). To further improve speed, we endeavoured to create a scheduler with a critical loop containing only *one* reduction tree that would perform *both* wakeup and select functions.

The resulting design is a compacting matrix scheduler with fused wakeup and select logic. Dependency information is expressed as a matrix of dependency bits like the matrix scheduler, but select *and* wakeup are computed using a single radix-4 tree of LUTs. Conceptually, instead of having one instance of selection logic broadcasting its result to per-entry wakeup logic, the selection logic is also replicated per entry and merged with the wakeup logic. Figure 7 shows this arrangement.

1) Wakeup and Select: Scheduler entries are ordered by age using compaction, so the selection uses fast fixed-priority selection. Each row has a combined select-and-wakeup circuit. The two inputs to each instance of the select-wakeup logic are a ready vector indicating which instructions are ready for execution, and a dependence vector indicating whether the instruction in the current row is dependent each instruction. The select-wakeup logic computes whether the current instruction depends on the oldest ready (i.e., selected) instruction. If so, this means one dependency has been satisfied, and a two-bit counter storing the number of outstanding dependencies is decremented. An instruction is ready when the counter reaches zero. Grant logic is still used to generate grant signals to clear dependency bits in the matrix, but is now moved off the critical path.

The select-wakeup logic is equivalent to a priority multiplexer, implemented using the circuit in Figure 4, which is a radix-4 tree of 7-input ALMs with a logic depth of $log_4 N$. The priority multiplexer finds the first ready instruction and selects the one bit of data indicating whether the instruction depends on the selected (oldest ready) instruction.

There is more preprocessing that needs to be done than for the matrix scheduler. In addition to encoding dependencies as positions in the scheduler, we also need to count *how many* dependencies are outstanding, which is a population count of the dependence vector. In this implementation, the single-cycle preprocessing is a critical timing path. However, because it is outside the wakeup-select loop, it should be possible for future implementations to further pipeline preprocessing without giving up the ability to schedule dependent instructions in back-to-back cycles.

2) Compaction: Compaction of a matrix is more complex than for a CAM. In a matrix scheduler, dependencies are represented as a bit vector indexed by the scheduler position of the parent instruction, whose position can change due to compaction. As scheduler entries are compacted downwards in a matrix scheduler, the dependency bit vectors are also

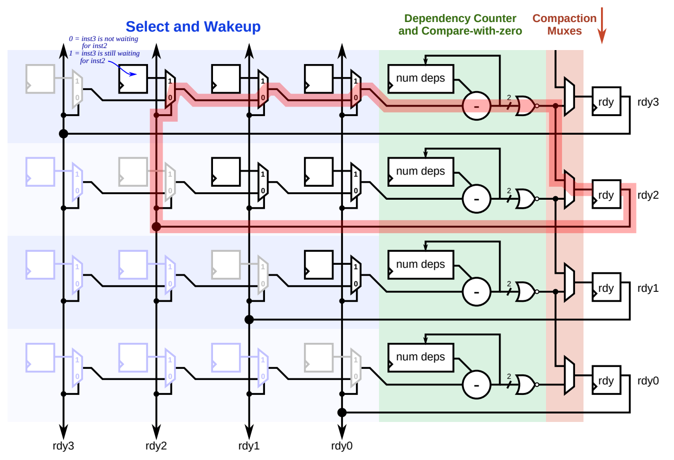

Select and Wakeup
0 = inst3 is not waiting for inst2
1 = inst3 is still waiting for inst2
Dependency Counter and Compare-with-zero
Compaction Muxes
num deps
rdy
rdy3
rdy2
rdy1
rdy0

Figure 7. Fused-logic matrix circuit. Selection and wakeup are merged and implemented as a one-bit wide priority multiplexer and a two-bit counter. Lower triangle is omitted as instructions do not depend on itself or future instructions.

compacted horizontally to track the changing instruction positions as they shift down the scheduler. Fortunately, the extra compaction logic is off the critical wakeup and select loop.

VI. Evaluation Methodology

The main objective of this work is to evaluate area and f_{max} of different circuit-level implementations of broadcast-based instruction schedulers. We build optimized circuits for the circuits described in the previous section (CAM, non-compacting matrix, and fused-logic matrix) targeting the a Stratix IV FPGA (smallest, fastest speed grade, EP4SGX70-C2) using Quartus 15.0.

We sweep scheduler capacity (entries) and observe area and f_{max} scaling as the scheduler size varies. All results are the mean of 100 random seeds. We focus on area and delay here because all of the scheduler circuits have nearly the same cycle-by-cycle behaviour: they wake up all ready instructions every cycle and select either a random instruction or the oldest ready instruction for execution.

VII. Results

This section presents area and f_{max} results of implementing CAM, matrix, and fused-logic matrix scheduler circuits on a Stratix IV FPGA.

A. Area

Figure 8 compares the area of the three scheduler circuit types as scheduler size changes. The matrix schedulers scale similarly, as the size of the matrix grows quadratically with the number of scheduler entries, but the matrix with position-based selection is smaller as it does not have an age matrix. CAM schedulers have better area at large sizes, as the size of comparators increases logarithmically (register number width) but the size of each matrix row's OR gate increases linearly. This can be seen more clearly when plotting area per entry, in Figure 8(b). Because we scale register number width with the scheduler size, there are small discontinuities at powers-of-two sizes when the register number width is incremented.

For out-of-order FPGA soft processors, we are mostly interested in small schedulers, generally below 20 entries. All circuit types have similar area below 20 entries, so delay targets will usually determine which scheduler circuit style to choose. The poor area scaling of matrix wakeup logic was also true in custom CMOS, where the original matrix scheduler proposal saw more than four times greater wakeup array area when replacing the CAM wakeup logic with matrices for a 48-entry scheduler, in exchange for halving the wakeup delay [13].

B. Delay

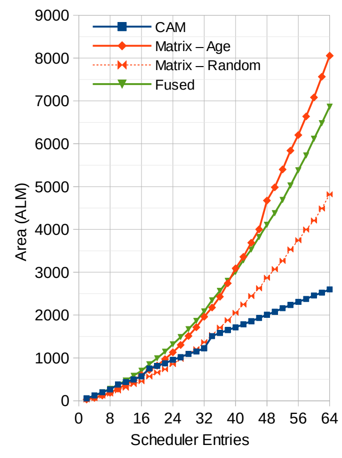
Figure 9 shows the achieved f_{max} for the three scheduler circuit types as scheduler capacity is varied. The general trend, unsurprisingly, is that larger schedulers are slower. The delay for the matrix schedulers increase faster than CAM schedulers at large sizes. On an FPGA where there are no fast wired-OR circuits, we see smaller improvements than those reported for custom CMOS implementations [13].

Among the three age-based schedulers, our new fused-logic matrix scheduler is the fastest option beyond 6–10 entries, though at very large sizes, excessive area causes poor routing delays. CAM schedulers are slow at small sizes, only being faster than the matrix scheduler beyond 24 entries. At small scheduler sizes where giving up age-based selection is acceptable, the position-based matrix scheduler is the fastest.

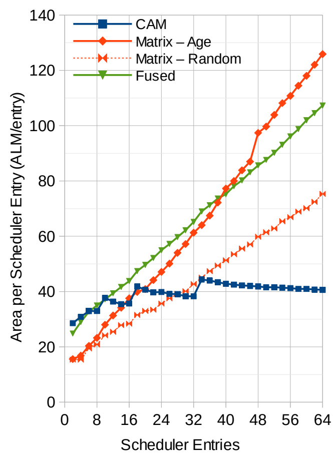
For out-of-order FPGA soft processors, we are interested in small schedulers. Below 20 entries, both types of matrix scheduler are faster than CAM schedulers, with little difference in area. To match the clock speed of a Nios II/f on the same FPGA (240 MHz or 4.2 ns), the largest age-based scheduler that will fit a 4.2 ns cycle time is around 20 entries for CAM, 22 entries for age-based matrix, and 42 entries for the compacting fused-logic matrix. A 44-entry position-based matrix scheduler also fits in a 4.2 ns period, but has limited usefulness at this size given the large IPC degradation of the selection policy. For comparison, current high-end x86 processors have 40–60 scheduler entries [7], while earlier out-of-order processors have far less (20 for Alpha 21264 [8], 16 for Pentium 4 [9]). This suggests that moderately aggressive out-of-order designs are feasible on FPGAs even when targeting the same frequency as simple single-issue in-order soft processors.

C. Instructions per Second

Figure 10 combines the IPC (Figure 1) and delay (Figure 9) results into a single plot, showing the performance trade-offs of the different scheduler circuits. The curved grid lines mark instruction throughput in MIPS, which is the product of IPC and frequency in MHz. Each point on the plot shows the IPC and frequency for a scheduler of a particular type and capacity. For example, a 10-entry matrix scheduler with random selection policy has 0.81 IPC, 434 MHz, and 353 MIPS, while a 12-entry scheduler of the same type achieves



(a) Area vs. Capacity



(b) Area per Scheduler Entry

Figure 8. Area of four scheduler types. Area per entry gives insight into scaling trends with scheduler size.

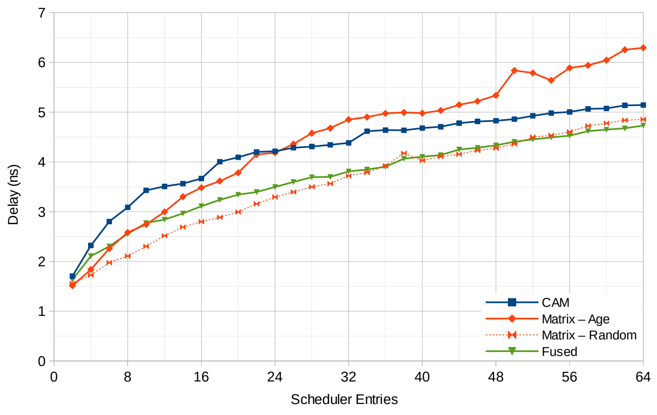


Figure 9. Delay of four scheduler types at varying capacities

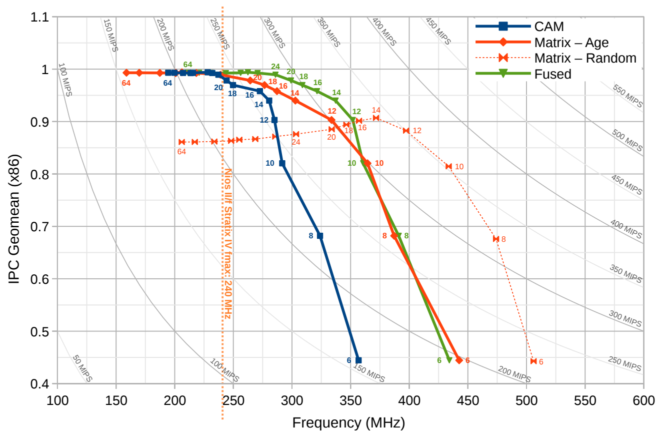


Figure 10. IPC vs. frequency of scheduler circuits from 6 to 64 entries. The curved gridlines mark x86 instruction throughput (MIPS). A higher-IPC age-based scheduler achieves higher throughput unless the rest of the processor exceeds 350 MHz.

almost the same throughput (351 MIPS) but does so with a higher IPC (0.88) and lower frequency (397 MHz). While these two design points have similar overall performance, the latter is easier to build as it imposes a less stringent timing constraint on the rest of the processor.

While the simpler position-based priority matrix can run at a higher frequency, its lower IPC at larger sizes means it performs best with small schedulers running at high frequencies above 350 MHz. For lower f_{max} targets, age-based schedulers provide higher overall performance, with the fused-logic matrix being the fastest. We expect soft processor designs would be unlikely to run above 350 MHz on a Stratix IV FPGA as the Nios II/f only runs at 240 MHz [17].

Some caution is needed in interpreting the absolute values in this chart: It combines data using f_{max} from a single-issue scheduler, but IPC from a multi-issue unified scheduler of the same size.

D. *Comparisons to Previous Work on FPGAs*

Direct comparisons with prior work are difficult to make due to differences in scheduler microarchitecture, but the approximate comparisons can still demonstrate our improvements. In most cases, to match the chip used in prior work, we re-synthesized our scheduler circuits on a different Altera FPGA than the one our circuits were designed for. Our instruction scheduler circuits achieve faster cycle times than schedulers in the literature, in some cases by substantial amounts.

1) Single-issue CAM: Aasaraai and Moshovos [6] presented a design space exploration of traditional single-issue CAM schedulers on Stratix III FPGAs. The microarchitecture of their scheduler circuits match well with our CAM (single issue, compacting age-priority, two operands) allowing for a reasonably fair comparison. On the same Stratix III FPGA, we achieve higher frequencies with our CAM scheduler circuit (+40% at 16 entries). Matrix and fused-logic matrix schedulers get additional gains (+47% and +60% at 16 entries, respectively).

2) Dual-issue CAM and Matrix: Johri compared two-issue CAM and matrix schedulers on FPGAs [18]. Our single-issue schedulers achieved twice the frequency at 16 entries for both CAM and matrix schedulers, but they use 3 source operands per instruction on a Virtex-6, while we use 2 source operands per instruction on a Stratix IV.

3) OpenRISC OPA: The OpenRISC OPA out-of-order processor merges the reorder buffer (ROB) and scheduler into a single unit, allowing some circuit simplifications and good f_{max} [19]. On the same Arria V FPGA, our fused-logic matrix scheduler in isolation achieves about 30% higher frequency than their complete processor at both 18 and 27 entries. Its main drawback is that a merged ROB and scheduler is wasteful of scheduler capacity. Schedulers only need to be 30–50% of the ROB size with almost no loss in IPC, which is also seen in our processor design with 64 ROB entries (Figure 1(a)).

4) Combined ROB and Scheduler: Rosière et al. presented a combined ROB and scheduler [5]. The microarchitecture appears highly unbalanced, with a large (128–512 entry) ROB, but only the oldest few instructions (4–16) are considered for scheduling. On a Virtex 5, they reported slow f_{max} (4.7× slower than our fused-logic matrix at 16 entries), and did not report absolute IPC numbers.

5) Non-Broadcast Scheduler: SEED is a scheduler designed to avoid broadcast behaviour [3]. Their circuits are surprisingly slow. On the same Stratix II FPGA, our fused-logic matrix scheduler achieves 1.9–2.4× higher f_{max} over their broadcast-free scheduler, and 6.5–5.5× higher f_{max} over their baseline, an Alpha 21264-like compacting CAM scheduler, for 16 to 64 entries.

VIII. Future Work

We presented circuits and results for single-issue schedulers. These serve as fundamental building blocks on our path towards a superscalar out-of-order x86 soft processor.

As part of an optimized superscalar processor, the schedulers will need to be extended to support multiple issue. This is generally straightforward, but the design space is much larger. The selection logic is similar to single-issue, but some unified schedulers may choose to use pick-N circuits to choose N ready instructions instead of multiple pick-1 circuits. The wakeup logic for multiple-issue schedulers is extended by monitoring multiple result buses for producers.

The schedulers also need to implement details of the target instruction set, such as supporting different instruction types (e.g., arithmetic vs. load/store vs. floating point), register types (e.g., general-purpose vs. condition codes), and variable latency instructions. In addition, further microarchitecture-level optimizations are available that trade IPC for faster and smaller circuits.

A. Further Microarchitectural Improvements

There has been much processor microarchitecture research that improves on the fundamental scheduler circuits. Most of these proposals still use the same circuit structures at their core, but trade some amount of IPC to improve area, speed, or power [11], [12], [14], [20]–[25]. The majority of these techniques can still be used on FPGA designs.

However, one technique for reducing matrix scheduler size, compacted matrices [14], [26], maps poorly to an FPGA substrate as it requires large multiplexers in the critical wakeup-select loop. For large schedulers on FPGAs, this is another reason to prefer CAM schedulers over matrix schedulers.

IX. Conclusions

We compared optimized circuit structures used in broadcast-based instruction schedulers. We also presented an improved age-based fused-logic matrix circuit that is substantially faster at age-based scheduling than traditional CAM- or matrix-based schedulers (∼20% faster at 22–36 entries, or twice the capacity at 240 MHz), yet is functionally equivalent. These are fundamental circuits found at the core of many optimized schedulers.

Our results show that moderately-aggressive out-of-order soft processors with schedulers of up to 40 entries are feasible on FPGAs at no frequency loss compared to the small, simple, highly-optimized Nios II/f. A matrix scheduler using position-based selection priority has high f_{max}, but is suitable only for low-IPC, high-frequency designs above 350 MHz. A higher-capacity age-based fused-logic matrix scheduler performs better for designs with lower frequency targets due to the higher IPC of age-based selection.

The IPC and performance benefit of out-of-order processors is expected to be large, on the order of 2× for a first implementation, and opens the door to even more aggressive designs in the future.

References

[1] SPEC, “SPEC CPU results.” [Online]. Available: https://www.spec.org/cpu95/results/

[2] B. Kuttanna, “Technology insight: Intel Silvermont microarchitecture,” IDF 2013, https://software.intel.com/sites/default/files/managed/bb/2c/02_Intel_Silvermont_Microarchitecture.pdf, 2013.

[3] F. J. Mesa-Martínez, M. C. Huang, and J. Renau, “Seed: Scalable, efficient enforcement of dependences,” in *Proc. PACT*, 2006.

[4] G. Schelle *et al.*, “Intel Nehalem processor core made FPGA synthesizable,” in *Proc. FPGA*, 2010, pp. 3–12.

[5] M. Rosière *et al.*, “An out-of-order superscalar processor on FPGA: The reorder buffer design,” in *Proc. DATE*, 2012.

[6] K. Aasaraai and A. Moshovos, “Design space exploration of instruction schedulers for out-of-order soft processors,” in *Proc. FPT*, Dec 2010.

[7] M. Golden, S. Arekapudi, and J. Vinh, “40-entry unified out-of-order scheduler and integer execution unit for the AMD Bulldozer x86-64 core,” in *Proc. ISSCC.*, Feb 2011.

[8] J. Farrell and T. C. Fischer, “Issue logic for a 600-MHz out-of-order execution microprocessor,” *IEEE JSSC*, vol. 33, no. 5, pp. 707–712, May 1998.

[9] S. Vangal *et al.*, “5-GHz 32-bit integer execution core in 130-nm dual-VT CMOS,” *IEEE JSSC*, vol. 37, no. 11, pp. 1421–1432, Nov 2002.

[10] L. Gwennap, “MIPS R12000 to hit 300 MHz,” *Microprocessor Report*, vol. 11, no. 13, Oct 1997.

[11] M. D. Brown, J. Stark, and Y. N. Patt, “Select-free instruction scheduling logic,” in *Proc. MICRO*, 2001.

[12] J. Stark, M. D. Brown, and Y. N. Patt, “On pipelining dynamic instruction scheduling logic,” in *Proc. MICRO*, 2000.

[13] M. Goshima *et al.*, “A high-speed dynamic instruction scheduling scheme for superscalar processors,” in *Proc. MICRO*, 2001.

[14] P. G. Sassone *et al.*, “Matrix scheduler reloaded,” in *Proc. ISCA*, 2007, pp. 335–346.

[15] W. Lynch, G. Lautterbach, and J. Chamdani, “Low load latency through sum-addressed memory (SAM),” in *Proc. ISCA*, 1998.

[16] D. Harris, “A taxonomy of parallel prefix networks,” in *Proc. Signals, Systems and Computers.*, vol. 2, Nov 2003.

[17] Altera, *Nios II Performance Benchmarks, DS-N28162004*, 2015.

[18] A. Johri, “Implementation of instruction scheduler on FPGA,” Master’s thesis, University of Tokyo, 2011.

[19] W. Terpstra, “OPA: Out-of-order superscalar soft CPU,” in *ORCONF*, 2015.

[20] D. Ernst and T. Austin, “Efficient dynamic scheduling through tag elimination,” in *Proc. ISCA*, 2002.

[21] I. Kim and M. Lipasti, “Half-price architecture,” in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, June 2003.

[22] C.-H. Chen and K.-S. Hsiao, “Scalable dynamic instruction scheduler through wake-up spatial locality,” *IEEE Trans. Computers*, vol. 56, no. 11, pp. 1534–1548, Nov 2007.

[23] S. Palacharla, N. P. Jouppi, and J. E. Smith, “Complexity-effective superscalar processors,” in *Proc. ISCA*, 1997.

[24] R. Canal and A. González, “A low-complexity issue logic,” in *Proc. Supercomputing*, 2000.

[25] P. Michaud and A. Seznec, “Data-flow prescheduling for large instruction windows in out-of-order processors,” in *Proc. HPCA*, 2001.

[26] E. Safi, A. Moshovos, and A. Veneris, “A physical-level study of the compacted matrix instruction scheduler for dynamically-scheduled superscalar processors,” in *Proc. SAMOS*, July 2009.