Hindawi Publishing Corporation
VLSI Design
Volume 2014, Article ID 493189, 13 pages
http://dx.doi.org/10.1155/2014/493189

Research Article

Investigation of a Superscalar Operand Stack Using FO4 and ASIC Wire-Delay Metrics

Christopher Bailey[1] and Brendan Mullane[2]

[1]*Department of Computer Science, University of York, Heslington, York YO10 5DD, UK*
[2]*Circuits and Systems Research Centre, University of Limerick, Limerick, Ireland*

Correspondence should be addressed to Christopher Bailey; chrisb@cs.york.ac.uk

Received 3 June 2014; Revised 3 October 2014; Accepted 23 October 2014; Published 18 December 2014

Academic Editor: Kiyoung Choi

Complexity in processor microarchitecture and the related issues of power density, hot spots and wire delay, are seen to be a major concern for design migration into low nanometer technologies of the future. This paper evaluates the hardware cost of an alternative to register-file organization, the superscalar stack issue array (SSIA). We believe this is the first such reported study using discrete stack elements. Several possible implementations are evaluated, using a 90 nm standard cell library as a reference model, yielding delay data and FO4 metrics. The evaluation, including reference to ASIC layout, RC extraction, and timing simulation, suggests a 4-wide issue rate of at least four Giga-ops/sec at 90 nm and opportunities for twofold future improvement by using more advanced design approaches.

1. Introduction

Current trends in semiconductor technology, and in particular the International Technology Roadmap for Semiconductors [1], suggest that future concerns in microarchitecture at the VLSI level will pose significant challenges. These include increasing power density [2], progressively severe thermal hot spots in increasingly complex designs [3], the impact of growing static power [4], and the problem of wire versus gate-delay and power scaling [5, 6]. Such problems are often most acutely exposed in key mainstream processor components such as cache, register related logic such as reorder buffers, rename logic, and the register file itself. Any alternative scheme to the traditional register-based computing paradigm can therefore open up the possibility of new approaches to these problems. However, register files are so highly optimized that measuring alternatives now requires complete layout of an optimal design for comparison, followed by timing and power analysis and nothing as simple as functional comparison of abstract logic. This paper focuses upon one possible unexplored option for operand storage which is alternative in its structure to that of a register file. The questions we examine are (a) can a LIFO (last-in-first-out) stack support superscalar operand access and (b) what is its performance relative to established mainstream approaches.

This work is undertaken with a 90 nm UMC CMOS process library; however, we ultimately utilize FO4 as a delay metric [7] in order to provide a general measure of performance that can be scaled to other process nodes. The work is undertaken using standard cell digital libraries and not at the transistor level. Although this is not therefore an optimal solution, it permits rapid assessment of multiple implementation schemes and semicustom design of the most promising candidate. This also means that performance results act as a conservative (lower) limit on potential performance.

Inevitably this work has some relevance to the age-old argument of stack processors versus register machines, particularly as significant improvements have been made in stack-processor code efficiency and code optimization [8–10] and in design of architectures capable of multiple instruction issue with out-of-order completion and/or previous attempts to parallelize stack structures [11–14].

However, the fundamental focus and aim of this paper is not to compare two competing processor paradigms but simply to establish in its own right the feasibility of a stack structure capable of permitting efficient access to operands in a superscalar access mode. It is also possible, for example, to have stacks employed in situations where they are not the basis for complete programmable processors. Therefore, the application space for such a solution is recognizably, but not exclusively, in the CPU design space.

In a traditional operand stack, multiple operands are held in a stack area organized notionally as a pushdown stack or LIFO buffer. Often the physical implementation employs a pointer into a small memory area. However, this leads to serious bottlenecks and there is little difference between this approach and a register file, especially where multiple operands need to be accessed in the same clock cycle (a requirement for superscalar operand access). In CPU oriented applications, it is not unusual to simply map a virtual stack onto an architectural register file, but undoubtedly this offers a poor presentation of a stack-oriented system since it is only an emulation of a true stack. It does however potentially allow superscalar execution since register files can be superscalar, though with significant hardware cost in terms of hazard avoidance and related logic. A true stack structure therefore appears at first sight to be restricted in its ability to act outside of its serial LIFO mode of operation. However, in this paper we demonstrate a discrete stack with multiport capability and superscalar functionality.

2. Superscalar Stack Issue Array (SSIA)

We propose a novel stack structure, the superscalar stack issue array (SSIA), whereby a small number of independent hardware stack cells are capable of being accessed collectively by multiple ports concurrently. However, such operations are inseparable from their stack effects (push, pop, or no effect). Consequently, collective stack reordering is performed as a single cycle internal operation, such that operands can be dispatched and stack-state kept coherent even with multiple actions in a group. The use of a novel tag scheme permits out-of-order write-back. All of these attributes can be achieved with basic logic structures. An unexpected benefit of destructive readout, usual in a stack structure, is that it eliminates the RAW/WAR dependency problem which hinders register-based processors to the extent as to demand renaming and reorder buffers and virtual registers with significant power, area, and thermal penalties. Therefore, an SSIA module can support superscalar operand issue and out-of-order completion without encountering RAW/WAR hazards or requiring a renaming scheme for its contents. These could conceivably be important benefits as described.

The tagging scheme allows delayed write-back to the stack structure, whereby any uncommitted stack cell content is supplanted by a unique tag which simply reserves and occupies the stack cell in question until the write-back can be completed. The unique aspect of this tagging scheme is that tags *move* with stack content, and therefore operands and results are actually *nomadic* in their behavior, and we refer to these incomplete contents as *nomads*. There is no possibility of multiple writes being able to target the same reserved space and hence no RAW/WAR.

If we consider how this looks at the logic level and how it differs from a standard stack, it may become clearer.

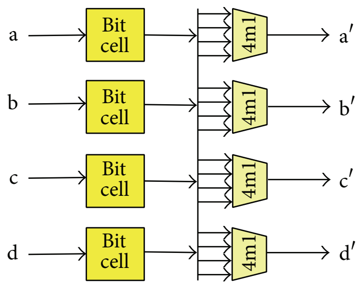


Figure 1: Fundamental stack logic scheme.

Figure 1 illustrates a fundamental stack implementation scheme. Each stack element bit is translated from its current state {a, b, c, d} into its respective next state {a′, b′, c′, d′} by use of a multiplexer arrangement, which performs local reordering, to reflect the stack effect of the applied action. In a standard scalar stack, the next state is fed back to the state retention flip-flops. However, if multiple reorder-mux-stages are cascaded, then the state feedback represents the stack effects of a *group* of actions, and the intermediate stages provide multiple operand pairs simultaneously, as illustrated in Figure 2. It is also notable that only one state transition is observed at the state retaining flip-flop, no matter how many issue slots are cascaded. A superscalar stack will have significantly fewer such transitions than a serial/scalar stack given the same sequence of actions, in effect one that actually saves dynamic flip-flop power by going from a scalar to a superscalar structure. By adding logic to permit insertion of a tag-matched value from a common data bus (CDB), it is possible for stack cells to contain tags reserving locations for delayed write-back values and for these to "retire" when a CDB tag is matched.

The extra inputs on the 5-way multiplexers facilitate placement of an immediate operand d_n via the topmost multiplexer or selection of an optional "stack fill" value m_{fn} from memory in the bottom-most multiplexer. How such memory-to-stack data movement is handled has many options, and we do not explore these here; however, even trivial buffer schemes of small capacity or a memory queue approach is a feasible solution [13, 15, 16]. Thus, this scheme achieves the ability for *in-order issue* with *out-of-order completion*. This is investigated further in the literature [11, 13, 14, 17].

Analyzing this SSIA structure one can make several observations; first of all it appears that each issue slot contributes a logic delay to the overall cycle time of the system. The logic delay for cascaded SSIA structures is typically *linear with respect to issue width*, a fact that is not true of register files; indeed area growth in register files can be cubic or quadratic as a function of port count [18] and have considerable impact in design scaling [19]. A further observation is that SSIA operand access delays are unequal. Again this is not true of a register file where all operands are emitted from parallel read ports with approximately equal latency. For a superscalar stack scheme, each operand pair is accessed with a different latency, ranging from near zero latency to a larger latency at the final issue slot. This may or

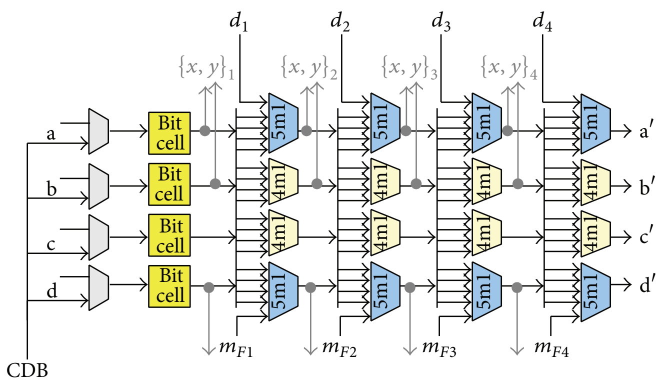


FIGURE 2: Superscalar stack issue array (issue width = 4).

may not be exploitable to some advantage; we simply observe the characteristic at this stage rather than speculating too far. However, what we would expect to observe is a similar story for power behavior where the *peak* power will not be a linear multiple of issue width, because each slot will peak at a *different* time and not at the *same* time. Total power over the whole cycle should however scale with issue width.

3. Implementation Scope and Variations

It should be clear, by analyzing these several cases of advanced stack operand stores, that the basic component building block of such a system is a multiplexer of some given characteristics, primarily the number of inputs or *input-ordinal*. It also follows that the best multiplexer design (in terms of speed, area, or power) will yield the best structure for the proposed stack-operand management scheme for the same optimization targets. An obvious choice is to use standard cell multiplexer components; however, this proves to be suboptimal where standard cell multiplexer choices are limited to a few cases and not necessarily implemented in the best way to facilitate ganging into suitable combinations. In order to evaluate this, it is necessary to consider the building block design at the logic level, its fan-out behavior, the logical effort, localized wire delay, and ultimately the full structure layout when a number of issue slots are cascaded and the logic is duplicated to represent the "*n*" bits of stack width (typically 16, 32, or 64 data bits per operand, for example).

We also note at this point that the depth of the discretely implemented stack is usually much greater than four elements. The top four elements only represent the "hot-zone" of the stack, where activity is more complex. Below this is a region we refer to as the "tidal zone" whereby all elements pop or push in unison, like ebb and flow of a tide, but are not reordered. Figure 3 highlights this difference in stack element zone behavior. It is noted that these deeper stack elements may be clock-gated or subject to dynamic power management when they are "empty." The nature of the stack makes this easily identifiable. We do not consider this in the analysis presented here however.

The tidal stack-element multiplexing arrangements are less complex. A 3-to-1 multiplexer selects either the existing

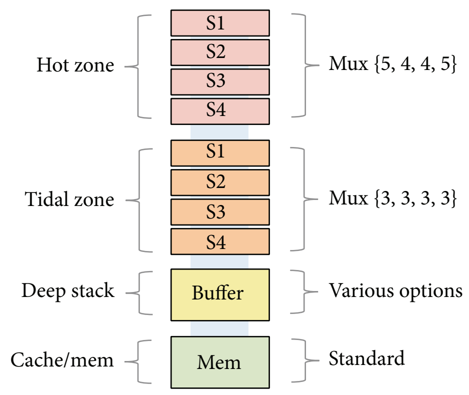


FIGURE 3: Functionally different regions of SSIA stack.

cell-state or one of two neighboring cells (above or below). How deep this tidal zone should be is a matter for further research. It should be as small as possible to reduce area/power cost, but a tidal zone of say 4 or 8 elements might not be sufficient to allow tags to achieve delayed write-back within the deeper stack before content migrates into memory or cache. This is an area for future research and we therefore assume, for the present, a discrete stack portion consisting of multiplexer ordinals as a set $\{3, 3, 3, 3, 5, 4, 4, 5\}$ reading bottom to top. The $\{5, 4, 4, 5\}$ portion being the "hot-zone" is symbolized by Figure 2 and the $\{3, 3, 3, 3\}$ portion being a "tidal zone" is as in Figure 3. This configuration is assumed in the rest of the paper. There are numerous possibilities for implementing multiplexers, each has its own advantage and disadvantage. Some options are summarized below:

(1) combinational logic using AND/OR/MUX,

(2) tristate selection of multiple inputs,

(3) pass-gate transistor based solutions.

In this paper, we focus on cases (1) and (2), and we present timings as an FO4 delay metric and as an absolute timing. In both cases, we derive FO4 results from a 90 nm 1 volt process but also quote absolute timing for comparison between design choices. This allows FO4 delays to be comparable across process nodes up to a point. In later sections, we compare the SSIA with alternatives using similar process nodes.

In this paper, we identify several standard cells of interest for our design objectives and comparative implementations:

(i) AO22: dual two-input-AND feeding into an OR-gate,

(ii) AOI22: equivalent to AO22 but omits final inverter,

(iii) INV: an inverter,

(iv) INVT: inverter with tristate output,

(v) MUX2/MLX2: two input multiplexers, implemented using pass-gate method,

(vi) MUX3/MLX3: 3 input versions of MUX2 and MLX2, implemented as cascaded MUX2 and MLX2 circuit implementation styles,

(vii) QDBAH: active low data latch.

For our experiments we always select the X1 variation of the available standard cell. This is not the fastest option but achieves low area cost. The trade-off between larger faster cells and smaller slower cells is not straightforward: leakage-power differs, and larger cells will lead to longer wires in the full structure and alter RC loading effects. Using these cells, we evaluate the ganged tristate approach of Figure 4, and also four more typical implementations, based upon the schemes illustrated in Figures 5(a), 5(b), 6(a), and 6(b). We had predicted that the tristate selection method would perform better than combinational logic, based upon initial logical effort analysis. The tristate model assumes the preceding decode stage generates one-hot selection signals. Other cases assume encoded selection signals, generated in the same way. We can thus define five initial models for evaluation, which we will refer to using the notation given as follows:

(i) CEN: combinational logic, encoded select lines, and noninverting logic,

(ii) CEI: combinational logic, encoded select lines, and inverting logic where possible,

(iii) MEN: MUX2 based design, encoded select inputs, and noninverting cell outputs,

(iv) MEI: MLX2 inverting mux used where possible, encoded select inputs,

(v) TNN: tristate mux model with nonencoded select inputs, and noninverting output per stage.

In addition to the basic switching structure, each model has a constant delay associated with the state retention flip-flop/latch and tag-match data-insertion multiplexer. Referring back to Figure 2, it can be shown that each multiplexing stage cascades into its successor stage with fan-out of 5 (*FO5C*), with the exception of the final stage (or only stage in a nonsuperscalar case). The final stage feeds in to the storage bit cell, possibly via the tag match insert logic, which can be placed either immediately before the bit-cell input or immediately after the bit-cell output. Therefore, it has a fan-out of 1 rather than the fan-out of 5 encountered when cascading to further stages. We therefore have case *FO1T* (feeding a tag-logic stage) and case *FO1L* feeding a latch stage.

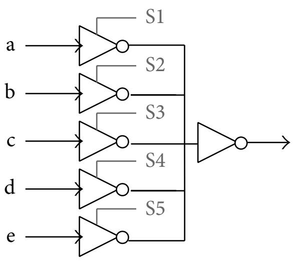


Figure 4: TNN-ganged tristate multiplexer scheme.

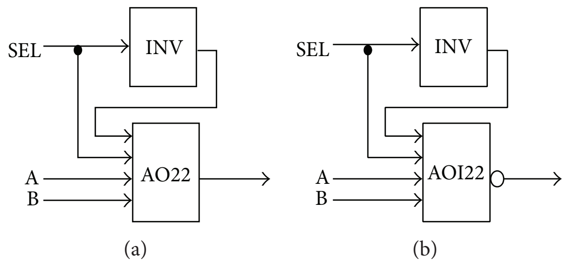


Figure 5: Combinational logic gate schemes. (a) CEN-two-input combinational Mux, (b) CEI-inverting equivalent.

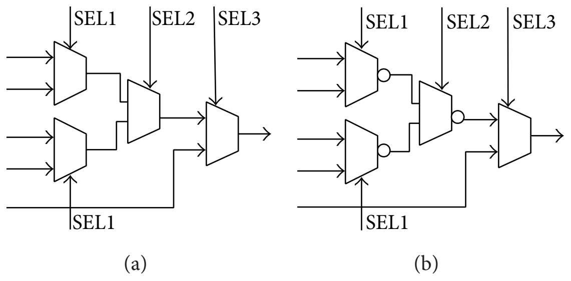


Figure 6: Cascaded multiplexer schemes. (a) MEN-two-input cascaded Mux, (b) MEI-inverting equivalent.

The loading effect of these choices then result in three fan-out loading cases that need to be calculated for each multiplexer style, relating to the position in the cascade chain:

FO5C: fan-out 5 cascaded,

FO1T: fan-out 1 to tag insert mux,

FO1L: fan-out 1 to Latch.

We evaluate this worst case here and results of our evaluations are presented in Table 1. From this initial analysis, it appears that the TNN model has the best speed advantage, and that either FO1T or FO1L variants are preferable for speed.

4. Tristate Multiplexer Models in Detail

In order to assess the tristate multiplexer strategy with as much accuracy as possible, we utilized several evaluations. First of all, evaluations based upon the 90 nm process data sheets were used to generate an initial timing estimate for

Table 1: Timing of 5m1 component models.

	FO5C	FO1T	FO1L
CEN	291 ps	229 ps	270 ps
CEI	211 ps	149 ps	164 ps
MEN	225 ps	188 ps	208 ps
MEI	168 ps	130 ps	131 ps
TNN	125 ps	97 ps	97 ps

Table 2: Data-sheet predictions.

Gang	Delay	Tristate fF	Inv fF
1	89 ps	2.67 fF	13.05 fF
2	98 ps	4.41 fF	13.05 fF
3	107 ps	6.15 fF	13.05 fF
4	116 ps	7.90 fF	13.05 fF
5	125 ps	9.64 fF	13.05 fF
6	134 ps	11.38 fF	13.05 fF
7	143 ps	13.13 fF	13.05 fF
8	152 ps	14.87 fF	13.05 fF

ganged tristate multiplexers, as shown in Table 2, using the model introduced earlier in Figure 4.

The total load capacitance (Cload) driven by the active tristate in the group was calculated as a function of fan-out (F) and ganging (G) plus the next stage inverter, such that load capacitance equated to

$$\text{CLOAD} = 2.668 \times F + 1.743 \times (G - 1). \tag{1}$$

The value 2.668 refers to input capacitance of the assumed next stage (an Inverter INVX1 cell), whilst the value of 1.743 refers to the capacitance of the outputs of the other inverters (referred to in the data sheet as input capacitance). Both values are extracted from the cell-library data sheets. These estimates appear to be sufficient for a first-order comparison of implementation methods. However, this may not be wholly accurate: signal behavior, layout, and wire effects have some importance, as will be highlighted later in this paper.

Having evaluated the timing models in each case and repeated the analysis for each of the FO5C, FO1T, and FO1L output loading cases, we have enough data to perform an evaluation of delay variation as a function of issue width. However, a complete analysis still needs to incorporate the constant delay of latch and tag insertion timing parameters themselves and not just the effect they have on output loading of preceding stages. This is outlined in the following Sections 4.1 and 4.2.

4.1. Tag Insertion Multiplexer. The MUX2 standard cell (X1 type) drives a single fan-out load for the data input of the latch QDBAH (X1 type). QDBAHX1 has an input capacitance of 1.976 fF for the D input node. This allows us to calculate a delay of 70 ps using the straight-line fit to MUX2 propagation delay data versus capacitive loading of the driven QDABHX1 latch input.

4.2. State Storage Latch. The latch itself has several important timing requirements. Setup and hold time amounts to just under 35 picoseconds. Data propagation from D to Q varies according to output load. This will be fan-out 5 in all cases but the target cell could be any one of MUX2, MLX2, AO22, AOI22, or INVT depending upon the implementation. We therefore have a further table (Table 3) for values derived for this latch propagation delay and also the final total with tag logic included.

Examination of Table 3 presents the data latch and total timing (tag stage plus latch timings). This shows that the variation in timing of the bit cell latch for different input variants is marginal.

Table 3: Bit cell and tag timing data.

FO5	MUX2	MLX2	AO22	AOI22	INVT
D-Q	74 ps	76 ps	76 ps	76 ps	76 ps
Total	179 ps	181 ps	181 ps	181 ps	181 ps

Table 4: Example aggregate timing formulae.

IW	Delay summation
1	Latch/tag + CEN.FO1T
2	Latch/tag + CEN.FO5C + CEN.FO1T
3	Latch/tag + (2 × CEN.FO5C) + CEN.FO1T
4	Latch/tag + (3 × CEN.FO5C) + CEN.FO1T

Table 5: Cycle times.

IW	CEN	CEI	MEN	MEI	TNN
1	410 (9.1)	330 (7.3)	367 (8.1)	311 (6.9)	306 (6.8)
2	701 (15.5)	541 (12.0)	592 (13.1)	479 (10.6)	431 (9.5)
3	992 (22.0)	752 (16.7)	815 (18.1)	647 (14.4)	556 (12.3)
4	1283 (28.5)	963 (21.4)	1042 (23.1)	815 (18.1)	681 (15.1)

FO4 delays in brackets.

5. First-Order Timing Projections

With timings projected for each component under all of the encountered input and output conditions, it is possible to assemble a timing analysis for a stack issue-logic slice for each of the component models introduced. At this stage, the subcomponent delays can simply be summed according to the configurations used. So, for example, the CEN model using combinational noninverting logic has configurations with respect to issue width as given in Table 4.

Replacing "CEN" with any of the other models allows the same cascading to be used to calculate the respective delays of each model. These are given in Tables 5 and 6 and plotted as graphs in Figures 7 and 8. Tables 5 and 6 show the absolute timing for the chosen 90 nm process, and the FO4 delay figure extracted on the basis that FO4 delay for 90 nm technology is 45 ps. By way of confirmation, our experimental timing measures derived an average FO4 delay of around

Table 6: Access times.

IW	CEN	CEI	MEN	MEI	TNN
1	0 ps 0	0 ps 0	0 ps 0	0 ps 0	0 ps 0
2	291 ps 6	211 ps 5	225 ps 5	168 ps 4	125 ps 3
3	582 ps 6/13	422 ps 5/9	450 ps 5/10	336 ps 4/7	250 ps 3/5
4	873 ps 6/13/19	633 ps 5/9/14	675 ps 5/10/15	504 ps 4/7/11	375 ps 3/5/8

FO4 delays shown beneath for issues slots 2/3/4.

43 ps for timing combined tpHL and tpLH. The results show that TNN is the best choice with this level of analytical detail. The combinational logic model CEN is by far the worst option even for a scalar issue model, and TNN is almost twice as fast as CEN for the highest issue width examined.

6. ASIC Layout and Core Evaluations

After performing timing estimates based primarily upon reported standard cell characteristics, it appeared that the tristate model offered the best delay characteristics without resorting to full custom cell design. This implementation model was then evaluated further during a collaborative visit to the Circuits and Systems Research Centre (CSRC), University of Limerick, by one of the authors.

VHDL coded descriptions of the 5m1 multiplexer, using tristate internal selection, were synthesized to create a standard-cell based mux core. After manually tidying up, the 5m1 cell appeared as illustrated in Figure 9.

Examining the cell critical wires as in Figure 10, wire lengths approximate to 13.7 μm (ganged wire), 2.7 μm (input), and 2.0 μm (output). After DRC and LVS checks, the 5m1 cell was cropped and rechecked by removal or addition of one or more tristates to create a range of cells from 2m1 through to 8m1. At this stage, a number of effects were then considered in order to get an accurate delay estimate for the building block. These are outlined as follows.

6.1. Slew Rates. Our tests revealed that input slew rates of the input test signals have an appreciable effect on timings, adding more than 10 ps to measurements for a conservative slew rate of 100 ps. We used a buffer chain to condition the input test signals and give realistic signal properties for our tests.

6.2. Standard Cell RC Delay. With RC behavior included, the delays for our multiplexer combinations are increased noticeably. The data sheets only provide for transistor characteristics and not layout related to wire and metal effects. Adding wire-related delays for the ganging interconnect in the multiplexer further increases the delay. Simulation data for these measurements is given in Table 7.

6.3. Fan-Out Conditions. Delays were measured under a fan-out of five-tristate inverter load, to match the anticipated output gate loading conditions. At this stage it becomes obvious that the initial timing projections given in Section 5 are conservative when compared against real layout and RC extracted timings (as shown in Table 7). Examination of the internal behavior of a 5m1 ganged tristate multiplexer core shows the effect of ganging and the unequal switching times for tPHL and tPLH transients. This is illustrated in Figure 11,

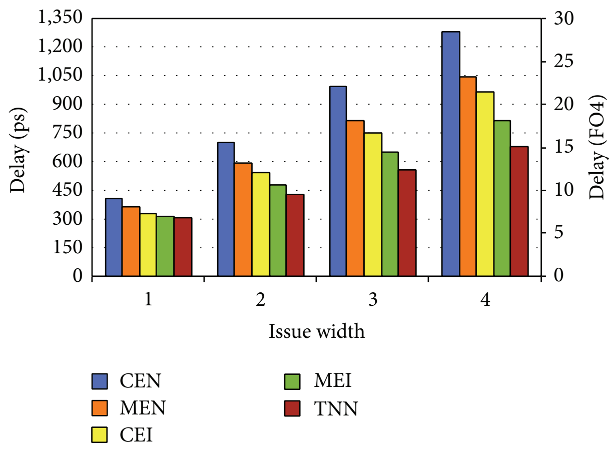


Figure 7: Issue width versus cycle time for various implementations.

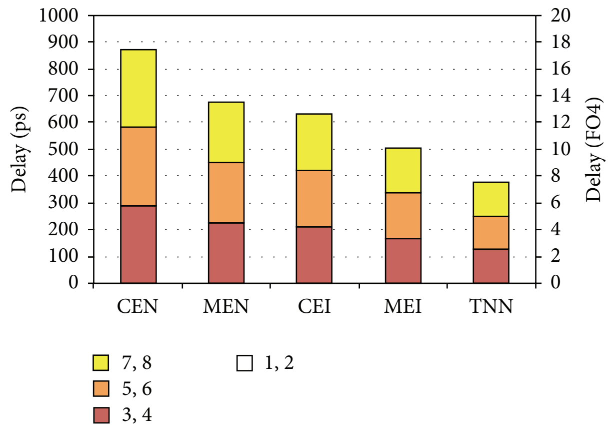


Figure 8: Issue width versus access time for stack using various implementation schemes. Showing operand access time for operand pairs 1-2, 3-4, 5-6, and 7-8, where operand pairs 1-2 have zero access time.

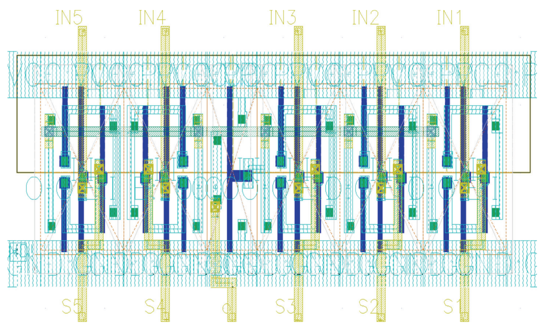


Figure 9: 5m1 tristate multiplexer layout.

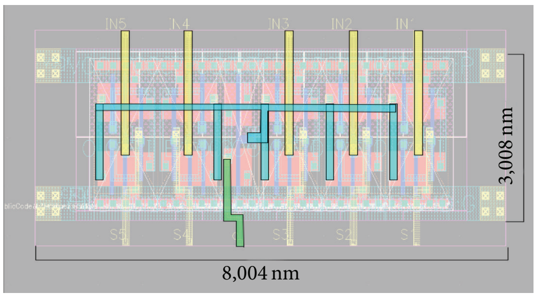


Figure 10: Significant wires in the 5m1 tristate cell, with dimensions in nm.

Table 7: Delays with and without local wires, for TNN MUX.

	Data sheets (FO5)	Cell delays (FO5)	Cell and wire (FO5)
2m1	89 ps	105 ps	117 ps
3m1	98 ps	121 ps	136 ps
4m1	107 ps	134 ps	155 ps
5m1	116 ps	150 ps	173 ps
6m1	125 ps	164 ps	190 ps
7m1	143 ps	179 ps	208 ps
8m1	152 ps	192 ps	225 ps

which shows an input signal of matched slew rate, causing internal node switching on the ganged tristate, which in turn drives the final output inverter. Both tPHL and tPLH are shown; it is clear that the transient on the internal node is the critical issue for this design. Using a dynamic precharging method on the ganged bus (when all tristates are disabled) might have a significant impact on this problem, allowing faster operation. This would certainly be an area for further investigation.

6.4. Fully Cascaded Structure and Bus Interconnect. To make comparative analysis easier, we derived a model for tristate behavior under given output loading conditions and multiplexer input counts. We began with the plot shown in Figure 12, which shows schematic timings data (blue), RC extracted data (red), and our final model (green: Figure 13) incorporating a distribution bus metal structure suitable for driving a 5-4-4-5 multiplexer row, which is the required worst case for the cascaded n-wide issue structure described in this paper. We also performed simulations of multiplexer cores at schematic and layout levels, and with cascaded cells, to derive timings approaching those for a full ASIC implementation. The data in Figure 13 is used for the final performance estimates, leading to the revised component timings for the TNN model, as given in Tables 8(a), 8(b), and 8(c), with wire effects of Table 9 and predicted access and cycle times in Table 10. This incorporates schematic derived timing data for the latch and tag stage (see Tables 8(b) and 8(c)), and wire effects (Table 9).

The full layout structure is shown in several figures given as follows: a single row of hot-zone elements is shown in Figure 14. When several of these rows are stacked one below another, interlocking the input and output bus lines, the structure appears as in Figure 15, which represents a 4-wide "hot-zone" structure for a single bit of stack word width.

Table 8: (a) Timing models, various simulation modes. (b) Simulation timings for TNN multiplexer driving various next stages. (c) Schematic timings for LATCH and TAG-MUX components.

(a)

Mux	Data sheet	Schematic	Cell + wire	With bus
3m1	107 ps	121 ps	136 ps	156 ps
4m1	116 ps	134 ps	155 ps	176 ps
5m1	125 ps	150 ps	172 ps	195 ps

(b)

	3m1	4m1	5m1
FO5C	156 ps (3.4)	176 ps (3.9)	195 ps (4.3)
FO1T	127 ps (2.8)	144 ps (3.2)	159 ps (3.5)
FO1L	127 ps (2.8)	144 ps (23.2)	159 ps (3.5)

Includes std cell RC effects, intercell wires, and interrow/slot distribution bus. Bracketed data gives equivalent FO4 delays.

(c)

LATCH TO MUX2	106 ps	Total
MUX to {5, 4, 4, 5} row FO4	132 ps	**237 ps**
MUX2 to latch	102 ps	Total
LATCH to {5, 4, 4, 5} row	135 ps	**238 ps**

Table 9: Delay data for simple wire of length 0 um–60 um.

Wire	0 um	20 um	40 um	60 um
Delay	~	3.0 ps	6.2 ps	8.7 ps

Table 10: Cycle/access times versus issue width.

	IW 1	IW 2	IW 3	IW 4
Access	0 ps	195 ps	390 ps	585 ps
Time	(0)	(4.3)	(8.7)	(13.0)
Cycle	397 ps	592 ps	787 ps	982 ps
Time	(8.8)	(13.1)	(17.5)	(21.8)

Hot-zone control wire supply lines are required for each issue slot, 20 per issue slot. These are capable of being routed over the cells via a higher metal layer (seen running vertically top to bottom in Figure 15). These control lines do not impact upon the structure's overall area and standard cell packing density and have no influence on the dimensions of buses used to connect critical data paths between stacked issue slots. Finally, Figure 16 shows the additional cells, added to the left hand side of the layout, representing the {3, 3, 3, 3} "tidal" stack zone attached to each hot-zone issue slot module, with shared control lines coming from the left-hand side in this case (though over-cell routing is possible). The practical consideration to be made here is what impact the interconnect buses (between hot-zone modules) have upon

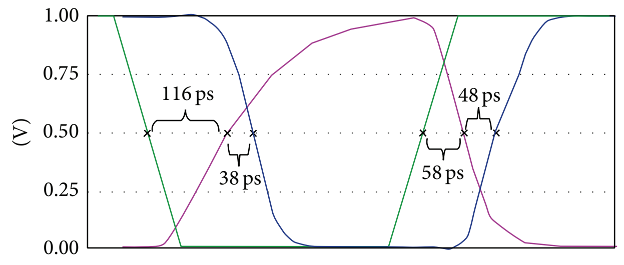


Figure 11: External and internal signals relationships for 5m1 RC extracted simulation. CADENCE timing data plot (redrawn for clarity).

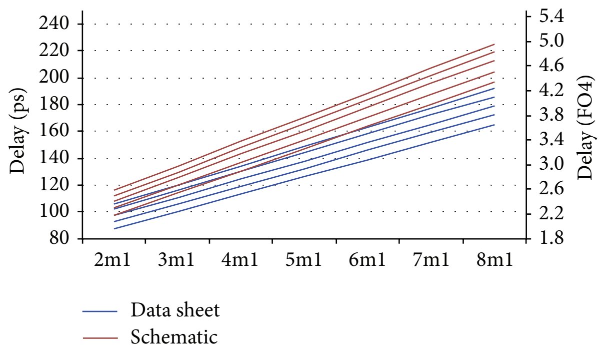


Figure 12: Data sheet versus schematic driven timings. Data sheet (Blue) measured schematic (red).

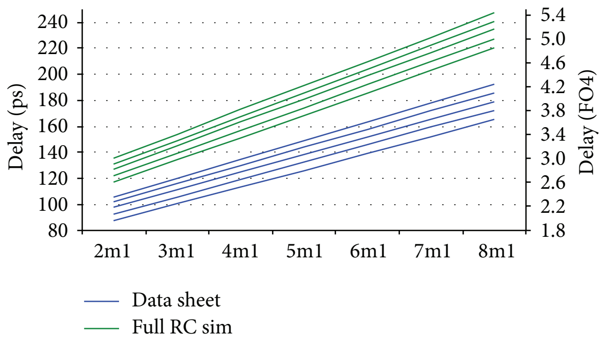


Figure 13: Data sheet versus RC delay with bus metal. Data sheet (blue) distribution bus (green).

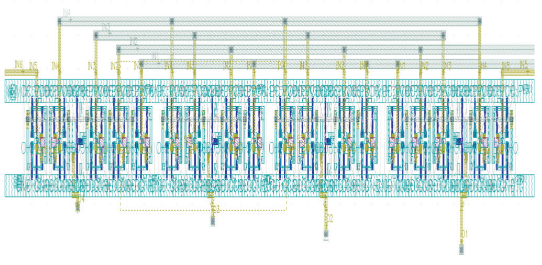


Figure 14: Hot-zone row {5, 4, 4, 5} configuration, and interconnect bus detail. Four outputs at the bottom are used to drive the four bus lanes of the next cell (equivalent to those at the top of this cell).

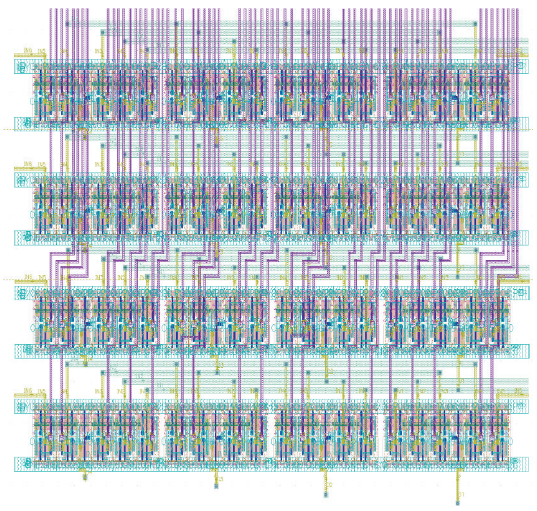


Figure 15: 4-wide {5, 4, 4, 5} issue structure.

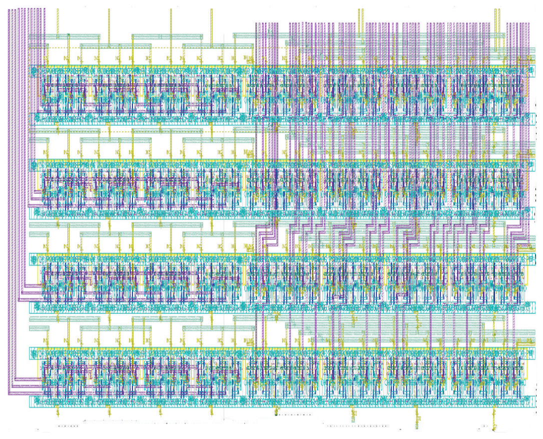


Figure 16: Complete {3, 3, 3, 3, 5, 4, 4, 5} × 4-slot structure, showing metal layers only.

signal delay due to added capacitance. These bus lines are singularly driven but fan out to 5 destinations. We performed simulation of cascaded rows using the bus structure shown in Figure 16 in order to account for this and found that bus related delays were typically of the order of 20 ps.

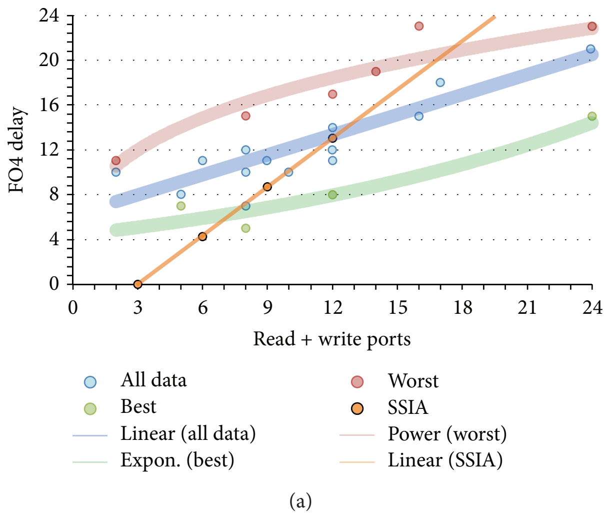
7. Comparative Performance

In previous sections, we developed a structural model for a superscalar operand stack and presented timing estimates for five different (relatively straightforward) implementation strategies. The next question to pose is to ask how fast a superscalar stack is with respect to register file alternatives. However, given the radically different nature of the superscalar operand store, as compared to the register file paradigm, blind like-for-like comparisons are difficult to assert. Consider a likely configuration of our superscalar stack unit, with perhaps four top-of-stack "hot-zone" elements (representing the {5, 4, 4, 5} multiplexer arrangement) and between four and eight deeper "tidal" stack elements implemented with 3-to-1 multiplexers. If we take the latter case then we have a total of 12 stack elements. The question is raised: *is this SSIA equivalent to a register file of 12 registers?* Our rationale for comparison is as follows.

7.1. Word Count. Such a stack as described above can push operands deeper than the 12 discrete elements implemented directly in logic, so it may not be accurate to describe it as having equivalent capacity to a register file of 12 words. Conversely, the number of "useful" registers in a register file under certain workloads is a small fraction of full capacity. Many registers are either awaiting write-back or contain dead data that will never be read again. Useful lifespans are typically measured in low 10's of clock cycles [39], so much so that some researchers have even proposed discharging unused registers to save power [40]. Stack content is almost exclusively live and useful, and it is much rarer to have dead "nomads"; thus, direct comparison is misleading. Empty stack elements are always identifiable easily and this correlates with our early comment about empty stack elements being able to be power or clock gated in a more sophisticated SSIA solution. However, we note that the fact that redundant register content is seen as a significant concern for static power wastage suggests that the stack approach has potential here as it rarely contains redundant content, which could be exploited for static/dynamic power reductions.

7.2. Port Count. Typical superscalar register files are arranged into "n" write ports and "$2n$" read ports. A 12-port register file is often organized as 8 read ports and 4 write ports. Such a configuration matches the expectations of a 4-issue machine, with 4 pairs of operands (8 reads) and 4 possible results (4 writes). An SSIA will have an identical read-port count (a 4-issue stack provides 8 operands). However, each stack element is capable of retiring an uncommitted value to the SSIA, so it might be thought that there are "n" write ports for an n-deep discrete logic stack portion. However, there are limitations on practical numbers of retirement buses, and in practice the ability to retire 2, 3, or 4 results to the stack is more realistic and has an impact on the design of the tag match logic (which we have not considered in this analysis) and therefore makes low orders of concurrent writes more desirable.

A further complication is that each issue slot can write an operand to the stack, albeit only to the top element. Do each of these write channels count as write ports in the traditional sense? With the restricted nature of the destination (top of stack only) it seems that it makes sense not to count these as true write ports in their own right. The postulated 12-deep stack with an issue width of four could retire four ALU results and also accept up to four new data values, whilst reading out 8 operands. The extreme interpretation is that this is therefore an 8+8 port operand store. Given that the 8 write channels are actual two groups of four channels with functionally different behaviors, we do not think this is the best way to compare like-for-like (as far as this is possible). We therefore come to a fairly loose conclusion: a suitable SSIA model for comparison would assume the equivalent of 1 write port and two read



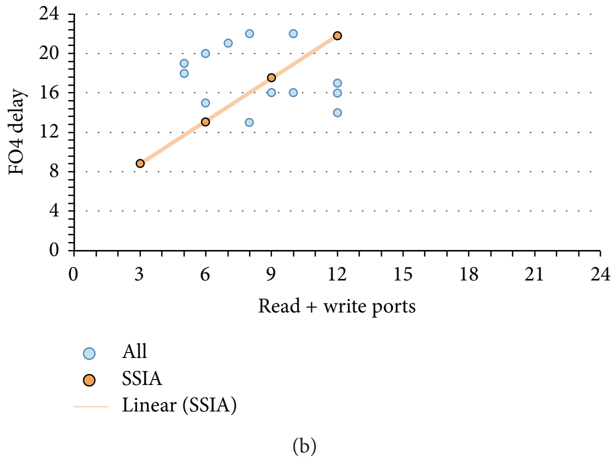


FIGURE 17: Comparison between SSIA and register file timing. (a) Access times and (b) cycle times, plotted on same scales.

ports per issue slot. Note that the stack size has no direct impact on the number of write ports assumed as this is largely a function of issue width; however, this is a parameter that could be investigated further.

To perform a comparison of SSIA versus register file performance, we collated a range of over forty FO4 timing data citations and timings for register files as reported in the literature [20–37]. These are detailed in Table 11. Access-time data appears to be more widely reported than cycle times (28 citations versus 13 in our sample). We consider both the access time and the cycle time of our SSIA in comparison to the register file. The population of access-time data for register files (from Table 11) was large enough to derive both best-case and worst-case envelopes and also a general trend for port count versus access delay, based on the whole data set. The data for cycle time was not sufficiently populated to support meaningful trend extrapolation. However, the position of the SSIA cycle time predictions within the group highlights its relative timing behavior. Plotting this analysis gives the graph shown in Figures 17(a) and 17(b), which show the population of register file FO4 characteristics for access time (Figure 17(a)) and cycle time (Figure 17(b)) alongside

Table 11: Reported register file delay times.

Reference	Ports[a] Tot (R + W)	T_{READ}[b] (FO4)	T_{CYCLE}[b] (FO4)	Comments
[20, 21]	10 (5 + 5)		22	250 nm CMOS, AMD K7 88 × 90 bit
[20, 22]	8 (4 + 4)	7		130 nmCMOS, 256 × 32 bit
[20, 23]	5 (3 + 2)	7	18	250 nm CMOS, 16 × 64 bit
[20, 24]	5 (3 + 2)	8		250 nm CMOS, 32 × 64 bit, IBM PowerPC
[20, 25]	8	0	13	500 nm CMOS, 32 × 64 bit
[20]	6 (4 + 2)	11	15	32 × 64 bit logical effort model
[20]	9 (6 + 3)	11	16	32 × 64 bit logical effort model
[20]	12	12	17	32 × 128 bit logical effort model
[26]	12 (8 + 4)	17		100 nm CMOS, 16 × 32 bit, low power
[26]	12	8		100 nm CMOS, 16 × 32 bit, high speed
[27]	8 (6 + 2)	5		250 nm SOI, 32 × 64 bits
[28]	12 (4 + 4)	11		100 nm, 160 × 64, two-bank, 440 reg
[28]	12 (4 + 4)	14		100 nm, 160 × 64, two-bank, 80 reg
[28]	24 (16 + 8)	15		100 nm, 160 × 64, two-bank, 40 reg
[28]	24 (16 + 8)	21		100 nm, 160 × 64, two-bank, 80 reg
[29]	17	18		1 um CHMOS, 128 × 64 bit,
[30]	14 (10 + 4)	19		400 nm, 116 × 64 bit
[31]	16 (10 + 6)	23		110 nm CMOS, 34 × 64 bit
[32]	24 (16 + 8)	23		130 nm, 512 reg
[33]	10	10	16	80 × 64 bit, various 250–35 nm
[34]	16 (12 + 4)	15		75 nm CMOS, 128 register,
[35]	2 (1 + 1)	11		180 nm, 160 reg 8 bank
[35]	2 (1 + 1)	10		180 nm, 160 reg 4 bank
[35]	8 (4 + 4)	15		180 nm, 160 reg, 1 bank
[35]	8 (4 + 4)	12		180 nm, 100 reg, 1 bank
[35]	8 (4 + 4)	10		180 nm, 60 reg, 1 bank
[35]	8 (4 + 4)	10		180 nm, 60 reg, 4 bank
[36]	5 (3 + 2)		19	500 nm CMOS, 128 reg
[36]	6 (3 + 3)		20	500 nm CMOS, 128 reg
[36]	7 (4 + 3)		21	500 nm CMOS, 128 reg
[36]	8 (4 + 4)		22	500 nm CMOS, 128 reg
[37]	12 (8 + 4)		14	500 nm CMOS, 48 reg,
[37]	12 (8 + 4)		16	500 nm CMOS, 96 reg

[a]Ports are stated as T (R + W) where T is port total, and bracketed figures (R + W) represent read and write ports where known.

[b]Delays are stated as FO4 delay, assuming 1 FO4 delay equates to an approximation scale of 2 nm per ps [38].

the same timing characteristics for the tristate-multiplexer SSIA model.

The comparative analysis presented in Figure 17 shows that the predicted models for the proposed SSIA configuration have a delay characteristic that is very competitive with register file for both access time and cycle time. This is certainly true for operand pair issue widths of 1, 2, 3, and 4 (equating to port counts of 3, 6, 9, and 12). One can observe that SSIA model appears to have increased delay penalty relative to register file for extreme port counts. For cycle times, where data was only available for register files up to 12 ports, the SSIA model appears competitive across the issue width/port count range plotted. Overall, it can be stated, based upon the data available in this comparison, that SSIA has highly competitive performance across both access and cycle times for up to 8 read ports and issue widths up to 4.

Taking the worst case delays from Table 10 (cycle times), it is possible to make tentative frequency estimates for a pipelined architecture limited by the superscalar store, with frequencies of around 2.5 GHz, 1.7 GHz, 1.3 GHz, and 1.0 GHz for issue widths of 1, 2, 3, and 4, respectively. A complete layout for one bit of an n-wide SSIA is given in Figure 18.

8. Conclusions

In this paper, we have considered a novel approach to operand management, using a stack based approach with a scheme which we believe is a new and novel approach for multiple operand issue, permitting superscalar in-order issue and out-of-order completion. Our methodology has been detailed and a model for building suitable stack structures is demonstrated.

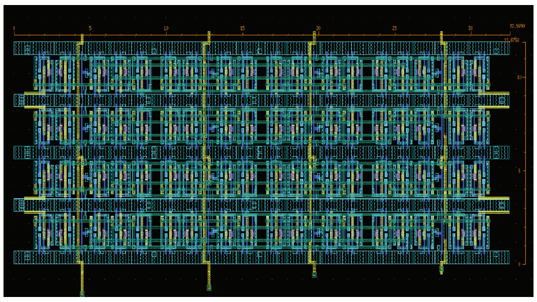


Figure 18: Suggested abutted multiplexer row/issue slot structure for 4-wide issue. Rows represent issue slots. Core area is ~32 × 11.9 μm = 378 μm^2 per bit (top four cells).

The cascading nature of the logic structure has two aspects: first of all, it allows a linear growth in logic cost and area, as well as delay characteristics and power consumption, which are potentially advantageous. However, the cascading nature also means that at some point (an issue width of 8, equivalent to a port count of 20, for example) the SSIA as presented here becomes less desirable as those costs accumulate. However, with realistic issue widths the cascading effect does not reach an uncompetitive behavior. Delay evaluations have been made, utilizing industry standard tools for core building block characterization, and making reasonable assumptions for configurations of systems equating to various issue widths. We have compared our projected performance data against a range of existing register file models and found comparable performance is viable. There are a number of possible enhancements that could be made to this base design in order to improve performance. These include

(i) advanced bit-cell design,

(ii) ganged pass gates rather than tristates,

(iii) look-ahead schemes to reduce cascading,

(iv) dynamic precharge of common node,

(v) selective clock-gating of empty cells.

Combining these possibilities, we envisage that cycle delays might be halved and perhaps more. This suggests that a 2 GHz 4-wide issue stack is conceivable with 90 nm CMOS with careful optimization and more innovative design, implying operand issue bandwidth somewhere in the region of 8 to 12 Gigawords per second at 90 nm. Naturally, more advanced processes will offer further performance gains.

One of the most interesting aspects of the superscalar stack is its ability to deliver issue slot operands at different times within the cycle time window, a unique behavior which we believe will allow more freedom in layout floor planning at a higher level of abstraction. This is particularly interesting if wire pipelining is introduced [41]. We also expect to observe (and perhaps to enhance further) a power-spreading effect that reduces peak power over machine cycle time scales due to the ripple-through effects of logic in our structures. This has some potential to reduce power density hot spots in the operand store as well as reducing peak power spikes. When combined with the knowledge that "useful" register lifespans are often a small fraction of the power-hungry register file [39, 40, 42], SSIA starts to look more interesting as a possible candidate upon which to base superscalar systems. Work on more advanced multiplexer design continues to be a current topic [43], and there is substantial scope to learn from this and improve upon the implementations reported here.

In the wider context, it has been fashionable to consider the stack machine as outdated. However, the potential for such architectures to deliver complex operand and instruction issue models highlights fresh opportunities and offers a new twist in the development of stack machines and related queue machines. Combining this with significantly better stack code optimization frameworks and models highlighted earlier [8–10, 14, 15] suggests that stack machines might be overdue a fresh examination in view of the trend for many simpler cores per chip rather than fewer but more complex ones. We believe that new avenues have been opened up by our initial study, in answering one question we have uncovered many others. A more comprehensive VLSI oriented study is a highly desirable next step in this work. Collectively these objectives will allow a complete design characterization for a prototype superscalar stack processor to be achieved. We therefore expect to continue to evaluate these new and novel SSIA architectures in the future and hope to report further findings in due course.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This paper was supported by work undertaken via visiting academic spells hosted by the Circuits and Systems Research Centre, University of Limerick.

References

[1] http://www.itrs.net/links/2011itrs/home2011.htm.

[2] G. Venkatesh, J. Sampson, N. Goulding-Hotta, S. K. Venkata, M. B. Taylor, and S. Swanson, "QsCores: trading dark silicon for scalable energy efficiency with quasi-specific cores," in *Proceedings of the 44th Annual IEEE/ACM Symposium on Microarchitecture (MICRO '44)*, pp. 163–174, ACM, December 2011.

[3] R. J. Ribando and K. Skadron, "Many-core design from a thermal perspective," in *Proceedings of the 45th Design Automation Conference (DAC '08)*, pp. 746–749, Anaheim, Calif, USA, June 2008.

[4] D. Sylvester and H. Kaul, "Future performance challenges in nanometer design," in *Proceedings of the 38th Design Automation Conference*, pp. 3–8, ACM, June 2001.

[5] H. O. Ron, K. W. Mai, and A. Fellow, "The future of wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, 2001.

[6] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*, pp. 365–376, IEEE, 2011.

[7] I. E. Sutherland, R. F. Sproull, and D. F. Harris, *Logical Effort: Designing Fast CMOS Circuits*, Morgan Kaufmann, 1999.

[8] P. Koopman, "A preliminary exploration of optimized stack code generation," in *Proceedings of the Rochester Forth Conference*, Rochester, NY, USA, 1992.

[9] B. Chris, "Inter-boundary scheduling of stack operands: a preliminary study," in *Proceedings of the EuroForth*, pp. 3–11, 2000.

[10] M. Shannon and C. Bailey, "Global stack allocation: register allocation for stack machines," in *Proceedings of the Euroforth Conference*, 2006.

[11] C. Bailey and M. Weeks, "An experimental investigation of single and multiple issue ILP speedup for stack-based code," in *Proceedings of the EuroForth Conference*, pp. 19–24, 2000.

[12] US Patent 6148391: System for Simultaneously Accessing one or More Stack Elements by multiple functional units, and related US patent 6026485: Instruction Folding for A Stack-Machine.

[13] C. Bailey, "A proposed mechanism for super-pipelined instruction-issue for ILP stack machines," in *Proceedings of the EUROMICRO Systems on Digital System Design (DSD '04)*, pp. 121–129, IEEE, September 2004.

[14] C. Bailey and H. Shi, "Instruction level parallelism of stack-code under varied issue widths, and one-level branch prediction," in *Proceedings of the IADIS International Conference on Applied Computing (AC '05)*, pp. 23–30, Algarve, Portugal, February 2005.

[15] C. Bailey, R. Sotudeh, and M. Ould-Khaoua, "The effects of local variable optimisation in A C-based stack processor environment," in *Proceedings of the 1994 Euroforth Conference*, 1994.

[16] T. J. Stanley and R. G. Wedig, "A performance analysis of automatically managed top of stack buffers," in *Proceedings of the 14th Annual International Symposium on Computer Architecture (ISCA '87)*, pp. 272–281, ACM, 1987.

[17] C. Bailey, "A proposed mechanism for super-pipelined instruction-issue for ILP stack machines," in *Proceedings of the Euromicro Symposium on Digital System Design (DSD '04)*, pp. 121–129, IEEE, 2004.

[18] C. Jesshope, "Microthreading a model for distributed instruction-level concurrency," *Parallel Processing Letters*, vol. 16, no. 2, pp. 209–228, 2006.

[19] S. Galal and M. Horowitz, "Energy-efficient floating-point unit design," *IEEE Transactions on Computers*, vol. 60, no. 7, pp. 913–922, 2011.

[20] N. Burgess, "Logical Effort analysis of multi-port register file architectures," in *Proceedings of the Conference Record of the 37th Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 887–891, IEEE, November 2003.

[21] M. Golden and H. Partovi, "500 MHz, write-bypassed, 88-entry, 90-bit register file," in *Proceedings of the Symposium on VLSI Circuits*, pp. 105–108, IEEE, June 1999.

[22] R. K. Krishnamurthy, A. Alvandpour, G. Balamurugan, N. R. Shanbhag, K. Soumyanath, and S. Y. Borkar, "A 130-nm 6-GHz 256×32 bit leakage-tolerant register file," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 5, pp. 624–632, 2002.

[23] R. L. Franch, J. Ji, and C. L. Chen, "A 640-ps, 0.25-μm CMOS, 16×64-b three-port register file," *IEEE Journal Solid State Circuits*, vol. 32, no. 8, pp. 1288–1292, 1997.

[24] O. Takahashi, J. Silberman, S. Dhong, P. Hofstee, and N. Aoki, "690ps read-access latency register file for a GHz integer microprocessor," in *Proceedings of the 1998 IEEE International Conference on Computer Design*, pp. 6–10, Austin, Tex, USA, October 1998.

[25] W. Hwang, R. V. Joshi, and W. H. Henkels, "A 500-MHz, 32-word x 64-bit, eight-port self-resetting CMOS register file," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 1, pp. 56–67, 1999.

[26] C. H. Hua and W. Hwang, "Low power multiple access port register file design in 100 nm CMOS technology," in *Proceedings of the 14th VLSI/CAD Symposium*, Hualien, Taiwan, August 2003.

[27] R. V. Joshi, W. Hwang, S. C. Wilson, and C. T. Chuang, ""Cool low power" 1 GHz multi-port register file and dynamic latch in 1.8 V, 0.25 μm SOI and bulk technology," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '00)*, pp. 203–206, July 2000.

[28] M. Kondo and H. Nakamura, "A small, fast and low-power register file by bit-partitioning," in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA-11 '05)*, pp. 40–49, IEEE, February 2005.

[29] R. D. Jolly, "A 9-ns, 1.4-gigabyte/s, 17-ported CMOS register file," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 10, pp. 1407–1412, 1991.

[30] C. Asato, "A 14-port 3.8-ns 116-word 64-b read-renaming register file," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 11, pp. 1254–1258, 1995.

[31] N. Tzartzanis, W. W. Walker, H. Nguyen, and A. Inoue, "A 34word $\times$ 64b 10R/6W write-through self-timed dual-supply-voltage register file," in *Proceedings of the IEEE International Solid-State Circuits Conference, Digest of Technical Papers (ISSCC '02)*, vol. 2, pp. 338–537, San Francisco, Calif, USA, February 2002.

[32] N. S. Kim and T. Mudge, "The microarchitecture of a low power register file," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '03)*, pp. 384–389, August 2003.

[33] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock rate versus IPC: the end of the road for conventional microarchitectures," in *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA '00)*, vol. 28, pp. 248–259, ACM, New York, NY, USA, 2000.

[34] K. Puttaswamy and G. H. Loh, "Implementing register files for high-performance microprocessors in a die-stacked (3D) technology," in *Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, IEEE, Karlsruhe, Germany, March 2006.

[35] R. Balasubraamonian, S. Dwarkadas, and D. H. Albonesi, "Reducing the complexity of the register file in dynamic superscalar processors," in *Proceedings of the 34th ACM/IEEE Annual International Symposium on Microarchitecture (MICRO '01)*, pp. 237–248, December 2001.

[36] J. Curz, A. Gonzalez, M. Valero, and N. P. Tophan, "Multi-banked register file architectures," in *Proceedings of the 27th International Symposium on Computer Architecture (ISCA '00)*, Vancouver, Canada, June 2000.

[37] K. I. Farkas, N. P. Jouppi, and P. Chow, "Register file design considerations in dynamically scheduled processors," in *Proceedings of the 2nd International Symposium on High-Performance Computer Architecture (HPCA '96)*, pp. 40–51, February 1996.

[38] D. Chinnery and K. Keutzer, *Closing the Gap between ASIC & Custom: Tools and Techniques for High-Performance ASIC Design*, Springer, 2002.

[39] P. Montesinos, W. Liu, and J. Torrellas, "Using register lifetime predictions to protect register files against soft errors," in *Proceedings of the 37th Annual IEEE/IFIP International Conference*

on Dependable Systems and Networks (DSN '07), pp. 286–295, IEEE, June 2007.

[40] L. Jin, W. Wu, J. Yang, C. Zhang, and Y. Zhang, "Reduce register files leakage through discharging cells," in *Proceedings of the 24th International Conference on Computer Design (ICCD '06)*, pp. 114–119, IEEE, October 2006.

[41] V. Nookala and S. S. Sapatnekar, "Designing optimized pipelined global interconnects: Algorithms and methodology impact," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '05)*, pp. 608–611, IEEE, May 2005.

[42] Z. Hu and M. Martonosi, "Reducing register file power consumption by exploiting value lifetime characteristics," *Proceedings of the Workshop on Complexity-Effective Design (WCED '00)*, vol. 1, pp. 1829–1841, 2000.

[43] R. Singh, G.-M. Hong, M. Kim, J. Park, W.-Y. Shin, and S. Kim, "Static-switching pulse domino: a switching-aware design technique for wide fan-in dynamic multiplexers," *Integration, the VLSI Journal*, vol. 45, no. 3, pp. 253–262, 2012.