Washington University in St. Louis
WashU Scholarly Repository

All Computer Science and Engineering Research
Computer Science and Engineering

Report Number: WUCS-94-24

1994-01-01

Performance Comparison of Asynchronous Adders

Mark A. Franklin and Tienyo Pan

In asynchronous systems, average function delays principally govern overall throughput. This paper compares the performance of six adder designs with respect to their average delays. Our results show that asynchronous addres (32 or 64-bits) with a hybrid structure (e.g., carry-select addres) run 20-40% faster than simple ripple-carry addres. Hybrid adders also outperform high-cost, strictly synchronous conditional-sum adders.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation
Franklin, Mark A. and Pan, Tienyo, "Performance Comparison of Asynchronous Adders" Report Number: WUCS-94-24 (1994). *All Computer Science and Engineering Research.*
https://openscholarship.wustl.edu/cse_research/345

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

To appear in:
Symposium on Advanced Research in
Asynchronous Circuits and Systems
Salt Lake City, Utah
November 3-5 1994

PERFORMANCE COMPARISON of ASYNCHRONOUS ADDERS

Mark A. Franklin
Tienyo Pan

WUCS-94-24

August 1994

In asynchronous systems, average function delays principally govern overall throughput. This paper compares the performance of six adder designs with respect to their average delays. Our results show that asynchronous adders (32 or 64-bits) with a hybrid structure (e.g., carry-select adders) run 20-40% faster than simple ripple-carry adders. Hybrid adders also outperform high-cost, strictly synchronous conditional-sum adders.

Computer and Communications Research Center
Washington University
Campus Box 1115
One Brookings Drive
St. Louis, MO 63130-4899

This research has been sponsored in part by funding from the National Science Foundation under Grant CCR-9021041.

Performance Comparison of Asynchronous Adders[1]

Mark A. Franklin and Tienyo Pan
Computer and Communication Research Center
Washington University
St. Louis, MO 63130

Abstract

In asynchronous systems, average function delays principally govern overall throughput. This paper compares the performance of six adder designs with respect to their average delays. Our results show that asynchronous adders (32 or 64-bits) with a hybrid structure (e.g., carry-select adders) run 20-40% faster than simple ripple-carry adders. Hybrid adders also outperform high-cost, strictly synchronous conditional-sum adders.

1 Introduction

In clocked digital systems, speed and throughput is typically limited by the worst case delays associated with the slowest module in the system. For asynchronous systems, however, system speed may be governed by actual executing delays of modules, rather than their calculated worst case delays, and improving predicted average delays of modules (even those which are not the slowest) may often improve performance. In general, more frequently used modules have greater influences on overall performance. Statistics presented in [7] show that in a prototypical RISC machine (DLX), 72% of the instructions perform additions (or subtractions) in the data path. In addition to ADD/SUB instructions (24%), branch (17%) and memory-access (31%) instructions which require calculations of target addresses or effective addresses also use adders. Thus, performance of asynchronous RISC processors is significantly influenced by the adder speed.

Addition can be implemented using *iterative networks* [9] with breakable carry chains. That is, in any digit position i, if the two operand bits $a_i \neq b_i$, the carry output of this digit is *propagated* from its carry input. However, if $a_i = b_i$, the carry output is independent from the carry input. The worst case situation with an n-bit addition occurs when every digit requires carry propagation due to the "$a \neq b$" condition and in this case the delay of a ripple-carry addition is n. However, since each digit with the "$a = b$" condition terminates the carry chain, the worst case is unlikely to happen. Burks, Goldstine, and von Neumann [3] have showed that for a ripple-carry addition of two n-bit operands chosen at random, the mean of the longest carry sequence is bounded from above by $\log_2 n$. Briley [2] further tightened this bound to $\log_2 n - 0.5$. In this work only nonzero carry propagation is considered, however, when dealing with asynchronous self-timing addition, propagations of both zero and nonzero carries must be considered. Reitwiesner [14] and Hendrickson [6] deal with both zero and nonzero carries and develop a more accurate model for the asynchronous ripple-carry addition. Hendrickson also shows (experimentally) that the average length of the longest carry sequence can be approximated by $\log_2(5n/4)$.

The $O(n)$ maximum delay for ripple-carry addition makes this design impractical in high performance clocked systems, and consequently much of the work on adders used in this environment have focused on techniques to reduce this maximum delay. Sklansky [15] developed a strictly synchronous technique (*conditional sum*) which has $O(\log n)$ maximum delay, and Winograd [19] showed that with this technique the lower bound (on maximum delay) on addition is achievable. Other adder designs also have $O(\log n)$ maximum delays [18, 1, 10]. Ling also proposed a high-performance adder that employs wired-OR circuits [17].

Compared with the studies of maximum delays which have been motivated by the requirements of clocked system design, research on average delays which affect the performance of asynchronous systems has been less extensive. This paper studies several design alternatives for adders operating in an asynchronous environment. Alternative designs are

[1]This research has been sponsored in part by funding from NSF under Grant CCR-9021041.

simulated and their speeds are compared. The system implications of these speed differences are explored in the context of a simple asynchronous RISC processor model. Our results show that a variety of hybrid adders are faster than ripple-carry adders (with respect to average speed) by 20-40%, and also faster than strictly synchronous adders (which may also be larger). Although hybrid adders are roughly 1.5 to 3 times larger than ripple-carry adders, the cost/performance tradeoff may well be worthwhile in the context of RISC processor performance.

In Section 2, the pros and cons of asynchronous designs are reviewed, a model that compares the performance of clocked and asynchronous systems is formulated, and the origins of the potential speed advantage associated with asynchronous design are discussed. Six adder design alternatives are described in Section 3. These designs represent different structures (serial, tree, or hybrid) and techniques (carry lookahead or carry select). Section 4 presents material relating to simulation of the different adders. Simulation results are shown and interpreted, and some design recommendations are made. In Section 5 the effect of the different adders on RISC processor performance is analyzed. Section 6 presents conclusions and suggestions for further research in this area.

2 Clocked versus Asynchronous Delay Models

The general argument in favor of using asynchronous design methodologies rests on four ideas. First, as clocked systems increase in size, clock skew increases and inevitably limits clock rate. The equivalent delay is not present in asynchronous systems (although other delays are present) [16]. Second, hierarchical, modular design techniques are generally ill suited to handling global design constraints such as clock distribution. Asynchronous techniques permit one to focus on the functional and logical sequencing aspects of design and not on such global issues thus making the design task more manageable. Third, asynchronous systems require less power than clocked systems since unused modules in asynchronous systems do not require charging/discharging [5]. Finally, the clock period in a clocked system is generally based on the worst case time for component functional units. In asynchronous systems, however, average function delays may govern overall throughput rates thus potentially resulting in higher performance. Naturally, the potential advantages associated with asynchronous design are subject to a host of qualifications, and are the subject of research and debate.

The principal drawbacks associated with asynchronous designs are three fold. First, if *dual-rail encoding* is employed to generate completion signals [12, 13], increased chip area is needed to implement the complementary logic, completion detectors, and the routing of differential input and output lines. Second, completion detection and handshaking requirements add extra overhead to asynchronous systems computation delays. These are analogous to the clocking overheads associated with synchronous systems. Third, to most of the digital design community, clocked systems appear to be easier to design, in part due to the availability of CAD tools oriented towards the clocked methodology.

Our research focuses on issues of speed. An analytical model that compares the speeds of clocked and asynchronous systems based on a pipelined architecture has been proposed by the authors [4]. This model can be simplified and described as follows:

$$Cycle\ time = t_{comp} + t_{sync} \tag{1}$$

For both clocked and asynchronous systems, a processing cycle *Cycle time*, consists of the computation time, t_{comp}, and the synchronization time, t_{sync}. A module executes during its computation time, and the results are sent to next pipeline stage during the synchronization time. For clocked designs, the computation time is fixed to the worst case computation delay, and the synchronization time corresponds to the clocking delay (i.e. clock skew and latching delay). For this case the equation can be rewritten as:

$$Cycle\ time_{clk} = t_{worst-case-comp} + t_{clocking} \tag{2}$$

Compared with the clocked counterparts, asynchronous designs usually have larger synchronization time since, with current technologies, the time consumed by the completion detection and handshaking protocols is typically greater than the clock skew. This can be simply modeled by introducing a multiplicative factor h ($h > 1$) to modify $t_{clocking}$. Furthermore, since the *Cycle time* for asynchronous pipelines is based on average computation delays rather than worst-case delays, the *Cycle time* in this case is:

$$Cycle\ time_{asyn} = t_{average-comp} + h \cdot t_{clocking} \tag{3}$$

The average computation time, $t_{average-comp}$, can be simply modeled in terms of $t_{worst-case-comp}$ by introducing two modifying parameters, i and d ($i, d \leq 1$). i is an *instruction-dependent parameter* which reflects

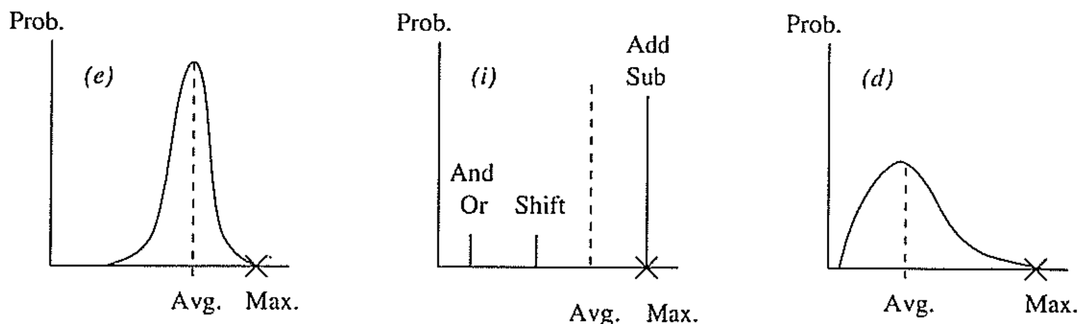


Figure 1: Distributions of e, i, and d

the fact that different types of instructions require different computation times, and d is a *data-dependent parameter* which reflects the variation in computation times due to different operand values. Thus, $t_{average-comp} = i \cdot d \cdot t_{worst-case-comp}$.

Finally, to ensure correct operation, clocked designs must be based on worst case assumptions concerning fabrication tolerances and environmental operating conditions. This introduces additional delays which must be built into the synchronous design. For example, the clock period must be set for worst case temperature conditions even though, in general, operating temperatures are not worst case. Asynchronous systems can take advantage of the increased speed associated with nominal operating environments. To reflect this in the model an environmental parameter e ($e < 1$) is introduced. The complete asynchronous cycle time is thus:

$$Cycle\ time_{asyn} = e \cdot (i \cdot d \cdot t_{worst-case-comp} + h \cdot t_{clocking}) \quad (4)$$

Examples of delay distributions of these parameters are shown in Figure 1. In a clocked system, the clock cycle has to match the maximum value of each parameter (i.e., the points denoted by 'x') while asynchronous system performance is dependent more on average delays (i.e., the dotted lines). The *environmental parameter* is decided by operating conditions, and the *instruction-dependent parameter* is determined by applications and programming styles. Both parameters give asynchronous designs an advantage over their clocked counterparts, however, only the *data-dependent parameter* relies directly on adder circuit design and designers can alter designs to attempt to obtain smaller average delays. This paper considers several design alternatives for asynchronous adders, where the focus is on the data-dependent parameter.

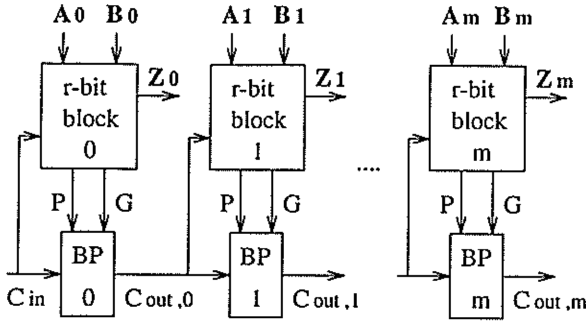
3 Design Alternatives

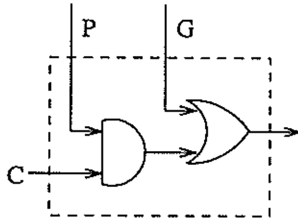
Many adder designs, mainly used in clocked systems, have been proposed. Some of these adders have the merit of small size, some have short (maximum) delays, and some have a good balance of the two. After a review of the major design options, six candidate designs will be selected and reviewed from the view point of inclusion in an asynchronous RISC processor pipeline.

3.1 Classification of Adders

Adder designs can be classified into *serial*, *tree*, and *hybrid* according to their high-level structures. The serial structure is present in ripple-carry adders whose worst case delay is $O(n)$ and whose size (in terms of number of gates) is also $O(n)$. The tree structure greatly reduces the worst case delay. For a worst case addition (i.e., $a_i \neq b_i$ for all i), computation of the most-significant bit requires information from all n bits. The fastest way to collect this information is by utilizing a tree structure which yields an $O(\log n)$ delay. Most high-speed adders, such as carry-lookahead [18], conditional-sum [15], carry-skip [10], and carry-select [1], are forms of tree structures. The main drawback associated with such structures is the $O(n \log n)$ size required.

Hybrid structures are midway between serial and tree structures both in terms of worst case delay and size. Hybrid structures partition the n-bits into blocks, which compute block-carry conditions (i.e., propagate, generate, or clear) in parallel. The carry sequence then can be passed between blocks (as opposed to between each bit) in a serial fashion. The basic model of a hybrid adder is shown in Figure 2 where each of the input symbols, A_i and B_i , represent a group of input bits. 'P' and 'G' indicate block-carry propagation and generation respectively. If (P,G) =





A Carry-Bypass (BP) Cell

Figure 2: Block diagram of hybrid adders

(0,0), a zero carry is sent to next block, and if (P,G) = (1,1), a one carry is sent. Block-carry propagation occurs when (P,G) = (1,0). In this case the block carry output is equal to the block carry input. Finally, the (P,G) = (0,1) state is not allowed. When the number of bits for each block are properly adjusted, the maximum delays of hybrid adders are $O(\sqrt{n})$ [7] while their sizes remain $O(n)$. Although many adder designs have been originally proposed with tree structures, they are often implemented as hybrid structures since for practical values of n, the $O(\sqrt{n})$ delay is comparable to $O(\log n)$. The $O(n)$ size, however, can be several times smaller than $O(n \log n)$.

As indicated, the average delay for a serial structure is $O(\log n)$. The average delay for a tree structure is equal to the worst case delay, since computation of the most-significant bit requires a constant time. The average delay for a hybrid structure, however, is more difficult to determine. Delays associated with this structure have three origins:

- t_1: time to find P and G for each block.
- t_2: time for block carries to propagate.
- t_3: time to compute the sums, Z's.

The delays, t_1 and t_3, depend on the techniques used in generating P, G, and Z. These techniques (i.e., carry lookahead or conditional sum) are discussed later. The t_2 delay is determined by the probability of a block carry being propagated. Carry propagation occurs, however, only when every bit in the block has the "$a \neq b$" condition (i.e., P,G=1,0 case). The probability of this case is $1/2^r$ where r is the number of digits in the block or the *block length.* Therefore, the longest block-carry sequence is usually short, and thus the mean of t_2 is small. Our experiments indicate that if t_1 or t_3 is also kept small, hybrid adders can yield high asynchronous performance.

Two common techniques used in tree and hybrid structures are *carry lookahead* and *conditional sum.* With the carry lookahead techniques, carries are predicted from a Boolean function of the inputs. Thus the delay due to full carry rippling is avoided. With the conditional sum technique both possible carry inputs (0 and 1) for each digit or block are used, and two sets of sum and carry outputs are produced. When the actual carry input is known, the correct result is simply selected.

3.2 Selected Adder Designs

In this section Six candidate adder designs representing the main structures and high speed addition techniques are reviewed. The asynchronous performance of these adders is presented in later sections.

1. **Ripple carry adder (RCA):** This adder has the traditional serial structure with small size, and large worst case delay.

2. **Conditional sum adder (CSA):** With this adder, both possible carry inputs are assumed for each bit. Sum and carry bits are calculated under the assumption that carry input is 0, and (in parallel) another set of sum and carry bits are calculated under the assumption of carry bits equal to 1. Pairs of conditional sums and carries are then combined according to actual value of the carry which enters each pair of bits and aggregated values are then presented to the next level. The overall sum is obtained by continuing this process through a full $\log_2 n$ levels. CSA has a tree structure and takes a constant amount of time to complete ($O(\log_2 n)$). Since conditional-sum adders are considered one of the fastest adders (for clocked design) [8, 17], it is chosen in this paper to roughly represent the performance of clocked adders.

3. **Completion detection conditional sum adder (CDA)** is a modified form of CSA [11].

The modification gives the adder more of an asynchronous flavor by providing detection logic indicating the availability of true sum at each level. When the true sum is available, it is latched and the computation is completed. Thus, the full tree delay is often avoided. A CDA has a variable computation time whose mean is as small as $O(\log_2 \log_2 n)$. However, this design requires extra time and circuits to detect the true sum in each level and, if present, to route the completed sum from any arbitrary level to the output latch.

4. **Carry-lookahead adder (CLA)** can be designed in a tree structure with roughly constant delay. For asynchronous designs (in the discussion presented here) a CLA is considered in a hybrid structure. In such a CLA, both P and G (Figure 2) of each block are calculated from basic Boolean functions of the inputs. When the carry input arrives at a block, the sum bits of this block are calculated by an r-bit ripple-carry adder.

5. **Carry-skip adder (SKP)** is also considered in a hybrid structure here. It has been noticed that the function of P is simpler than the function of G. Thus, SKP generates P by the Boolean function and G by carry-rippling. For generating G, the carry input of the block is assumed 0.

6. **Carry-select adder (SEL)** is also considered in a hybrid structure. In this design, both P and G are obtained by carry-rippling. P is obtained by assuming carry input equal to 1, and G by assuming 0. Conditional sums are also calculated during the the generation of P and G. Once the true carry input is known, the correct sum is simply selected.

The last three candidates have hybrid structures. The delay of a hybrid addition, as mentioned in Section 3.1, includes the time for generating P and G (t_1), time for bypassing carries (t_2), and time for generating the sum (t_3). All three hybrid candidates have the same t_2. CLA has a short t_1 because of the carry-lookahead technique, but its t_3 is decided by an r-bit ripple-carry addition which has variable delays. For certain operand values t_3 could be long. SKP has both t_1 and t_3 decided by r-bit carry-rippling. This does not hurt the maximum delay of SKP (i.e., the use of SKP in clocked systems) since in the worst case t_2 usually dominates the overall delay. However, our results show that SKP has the worst average delay among the hybrid candidates. SEL has t_1 decided by

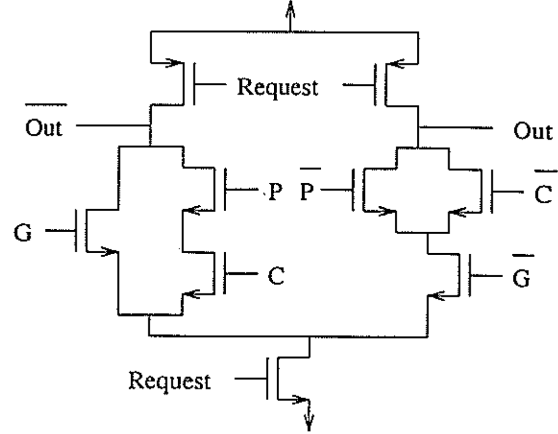


Figure 3: Carry bypass cell implemented in DCVSL

carry-rippling, but t_3 is only the delay of a multiplexor which selects the correct sum.

To minimize the worst case delay, clocked system designers use hybrid adders with variable block sizes, such that t_1 and/or t_3 can be hidden by t_2. However, our results indicate that to minimize the average delay, the block size should be fixed, with the optimal block size varying with designs and the value of n. For the work presented here, several block sizes have been simulated for each hybrid design, and the one with the best performance selected.

4 Adder Simulations

For the work presented adder implementations are assumed to employ differential cascode voltage switch logic (DCVSL). A DCVSL implementation of the bypass logic of Figure 2 (i.e., "$out = G + PC$") is shown in Figure 3. Initially, *request* is low (reset) and the NMOS circuit is precharged with both *out* and $\overline{out}$ going high. If the other inputs (i.e., P,C, and G) are present, then when *request* goes high, the circuit is evaluated. However, if after *request* goes high, all the inputs are not present (e.g., G is available and equal to 1 but P and C are unavailable), then the correct output may still be generated. In other words, completion of a function does not require the presence of all input signals. This illustrates early completion feature of the circuit and the breaking of carry chains that can be achieved by DCVSL cells. Note that delays associated with concurrent precharging which are not data dependent can be included in the synchronization time (see Equation (1)).

A completion signal may be produced from the two

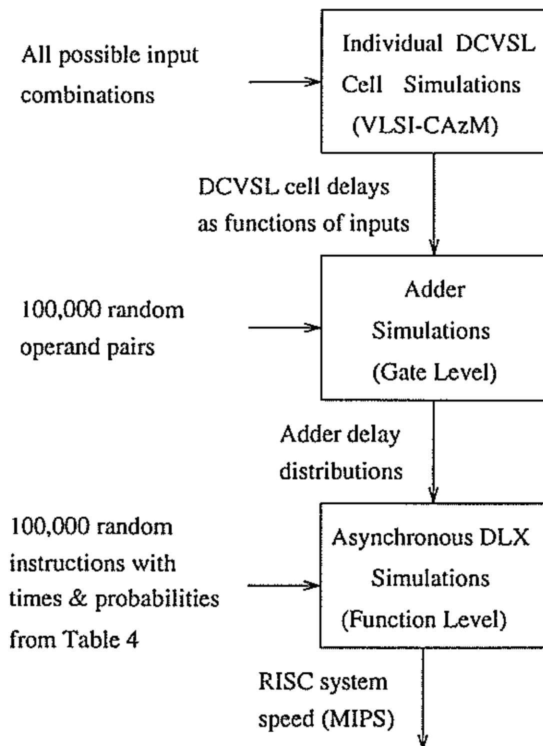


Figure 4: Simulation Hierarchy

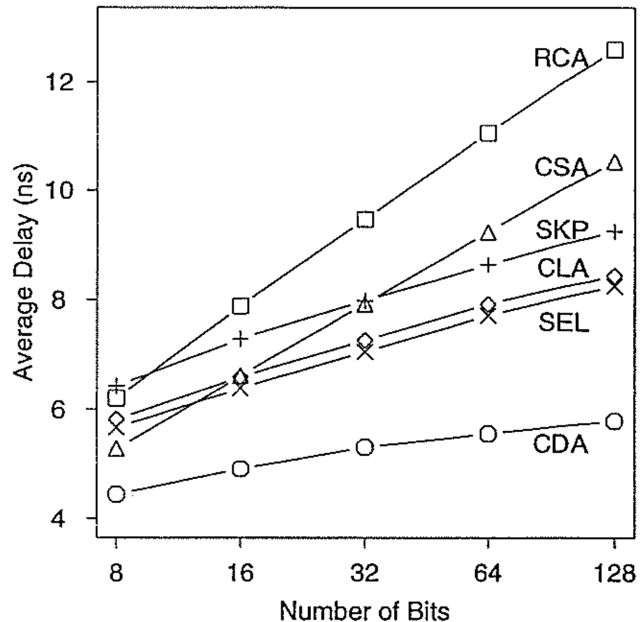


Figure 5: Average delays as a function of number of bits

complementary outputs. Other logic functions (cells) required by the adders can be formed in a similar fashion and all of the cells may be precharged and *requested* concurrently. DCVSL details may be found in [13].

Finding the average delay of an adder requires a large number of test operands. The most straightforward way to do this is to perform a VLSI circuit layout, and then simulate the circuit a large number of times with different operand values. However, each n-bit addition requires $2n + 1$ inputs and, with large n, the number of simulations required to find an accurate mean value combined with the time per simulation makes this impractical. The approach taken here has been to first perform the circuit layout for a number of basic DCVSL cells. The delay of each basic cell is obtained by a timing simulation tool, CAzM, (using a 1.2μ model). As illustrated in Figure 4, this represents the first level of a simulation hierarchy which is employed to finally obtain the effect of adder design on overall asynchronous RISC processor performance.

These delays are used as inputs to a set of gate level simulation programs (the second level of the simulation hierarchy) which have been written for different adders where each gate consists of a basic DCVSL cell. Since the individual cell delays are operand dependent, the gate level simulation accesses a table of delays for each cell type, the actual delay used being a function of the operands being tested. For example, the bypass cell yields 0.98ns delay for $(g, p, c) = (1,1,1)$ but 1.33ns for (0,1,1). This seems to be a good combination of realism and practicality. Table 1 shows basic cells and their functions and delays.

'Propagate' and 'generate' are the lookahead functions that generate P and G. The number in square brackets denote the number of digits included in one lookahead cell. Although only average and maximum delays are shown in Table 1 the delays are based on the input combinations and a complete delay distribution is obtained.

The adder simulation programs are driven by 100,000 pairs of randomly selected operands[2]. The value of each operand bit has equal probability of 0 or 1. Figures 5 shows the simulated average delays of the six adder designs. Ripple-carry adder (RCA) and conditional-sum adder (CSA) both have $O(\log_2 n)$ delays. Average RCA delay is larger than $\log_2 n$ according to Hendrickson's result [6], but CSA delay is

[2]The assumption of random operand selections is for the simplicity and generality of our models. At least one study [5] has indicated that carry chain distributions produced by real code has a somewhat higher mean than that obtained from a random model.

Table 1: Average and Maximum delays of DCVSL cells (1.2μ technology)

Function name	Boolean function	Delay (ns)	
		Average	maximum
Carry	$ab + bc + ca$	1.44	1.62
Sum	$a \oplus b \oplus c$	1.49	1.56
Bypass	$g + pc$	1.15	1.33
Multiplexor	$sa + \bar{s}b$	1.23	1.31
Propagate[2]	$(a_1 + b_1) \cdot (a_0 + b_0)$	1.23	1.32
Propagate[3]	$(a_2 + b_2) \cdot propagate[2]$	1.36	1.69
Propagate[4]	$(a_3 + b_3) \cdot propagate[3]$	1.43	2.08
Generate[2]	$a_1 b_1 + (a_1 + b_1) \cdot a_0 b_0$	1.52	1.77
Generate[3]	$a_2 b_2 + (a_2 + b_2) \cdot generate[2]$	1.59	2.22
Generate[4]	$a_3 b_3 + (a_3 + b_3) \cdot generate[3]$	1.16	2.27

exactly $\log_2 n$ levels. This is the reason that RCA and CSA have parallel curves, and the RCA curve is higher.

Completion detection conditional sum adder (CDA) has the best performance among the six due to its $O(\log_2 \log_2 n)$ average delay. However, as mentioned in earlier, CDA requires extra time to detect the availability of true sum in each level and direct the completed sum, which may happen in any level, to the latch. If this overhead is greater than 2 ns, CDA will lose its edge. Since CDA is the only design that has this overhead, problems of fairness may be involved in the comparison. Therefore, CDA is not included in studies of overall system performance which are considered in the next section.

The three hybrid adders, carry-lookahead (CLA), carry-skip (SKP), and carry-select (SEL), have similar curve slopes. This is because t_2 (i.e., the time for block-carry bypassing) is the only delay that grows with n, and all three hybrid designs have the same t_2. SKP has the worst performance among the three, because it requires carry-rippling in both t_1 (i.e., time to generate P and G) and t_3 (i.e., time to generate sum). SEL and CLA only use carry-rippling in t_1 or t_3, so their average delays are very close. This result shows that hybrid structure is more feasible than serial and tree structures for the design of adders in asynchronous environments.

Though not shown here, we have found that the optimal block size of hybrid adders is two when n is equal to 64 or less and three when it is greater than 64. The reason for small block sizes is because the probability of block-carry propagation is so small that t_2 is usually short. Larger block sizes will not help much in reducing t_2 but will increase t_1 and/or t_3. When n is as large as 128, however, two bits per block makes t_2 too long and, for this case, three bits per block results in better performance.

Average, standard deviation, and the worst case delays for 32 and 64 bit adders are shown in Tables 2 and 3 respectively. In a pipelined system, a higher standard deviation will result in an increased probability of a given stage blocking earlier stages. Thus, RCA has another disadvantage when used in an asynchronous RISC processor instruction pipeline.

5 Adders in an Asynchronous System

In this section the candidate adder designs are considered for use in a simple single pipeline asynchronous DLX machine. The machine is assumed to have five pipelined stages: instruction fetch (IF), instruction decode (ID), execution (EX), memory access (MA), and write back (WB). For a branch instruction, addition is performed in calculating the target address in the ID stage. For memory-access and ADD/SUB instructions, addition is performed in calculating the effective address, and in evaluating the ALU result in the EX stage.

To develop a simple functional level simulation of the DLX machine, a number of assumptions are made. It is assumed that the DLX pipeline has no stalls. That is, there is 100% success on branch predictions, no floating-point operations are present, there are no cache misses, and there are no data-dependent stalls. In addition, it is assumed that an instruction-prefetching buffer is available and that 75% of the instructions are fetched from this buffer. This relieves the bottleneck found in the IF stage. The delays asso-

Table 2: Performance Comparison of 32-bit adders (in ns)

Adder type	Average delay	Deviation	Worst Delay
Ripple carry (RCA)	9.47	2.55	51.84
Conditional sum (CSA)	7.92	0	7.92
Completion detect (CDA)	5.30	0.57	7.92
Carry lookahead (CLA)	7.25	1.10	24.96
Carry skip (SKP)	7.98	1.08	26.37
Carry select (SEL)	7.04	1.09	24.75

Table 3: Performance Comparison of 64-bit adders (in ns)

Adder type	Average delay	Deviation	Worst delay
Ripple carry (RCA)	11.06	2.70	103.68
Conditional sum (CSA)	9.24	0	9.24
Completion detect (CDA)	5.55	0.55	9.24
Carry lookahead (CLA)	7.92	1.15	46.24
Carry skip (SKP)	8.64	1.14	47.65
Carry select (SEL)	7.71	1.14	46.03

Table 4: Computation times at stages of simplified DLX machine

Instr. type	Pr. (%)	Computation times (ns)				
		IF	ID	EX	MA	WB
branch	20	3/10	Add	0	0	0
Add/Sub	25	3/10	3	Add	3	3
logical	25	3/10	3	3	3	3
memory	30	3/10	3	Add	10	3

Table 5: Performance of an Asynchronous DLX Using Different Adders

Adder type	System speed (MIPS)		Adder size
	32-bit adder	64-bit adder	
RCA	117.1	107.3	small
CSA	130.9	122.9	large
CLA	135.0	129.7	medium
SKP	129.4	125.2	medium
SEL	137.0	131.9	medium

ciated with each instruction type are found in Table 4 along with their execution probabilities [7]. Delays associated with all stages are 3 ns, except when a cache access is involved. In this case the time is 10 ns. Delay distributions of the adders (ADD) are obtained from second level adder simulations (Tables 2 and 3). Although the above assumptions over-simplify the RISC system, the results indicate the impact of adder speed on overall system performance.

The processor simulation (Figure 4) is driven by a sequence of 100,000 randomly generated instructions whose occurrence probabilities are given in Table 4. Each time an instruction requires an addition, the delay distribution associated with the given adder design is sampled to determine the ADD stage delay. The results of the simulation in terms of processor throughput are shown in Table 5. Since the DLX model is simplified and synchronization delays are ignored, the throughputs presented here are optimistic. However, the relative performance reflects the impact of the different adders. The results show that a 32-bit asynchronous RISC machine which employs SEL outperforms an identical machine which employs RCA by 17%. For a 64-bit machine, this advantage increases to 23%. Although SEL is about 2.5 times bigger than RCA, the cost of adder is small compared with the overall processor costs. Therefore, from the cost/performance view point, hybrid adders have higher performance than serial and tree adders in the selected asynchronous system.

If CSA is used in a clocked version of the DLX

machine which follows the descriptions of Table 4, the clock cycle would be set to 10 ns and the resulting throughput would be 100 MIPS. This speed is worse than its asynchronous counterpart. However, this comparison only includes instruction-dependent and data-dependent parameters. Environmental parameters and synchronization times need to be considered to gain a more accurate comparison of the two design methodologies [4]. In addition, power issues may well be an important factor in determining the suitability of asynchronous versus clocked processor designs.

6 Conclusions

In this paper, six adder designs are studied, and their influence on asynchronous system performance are compared. The results indicate that most hybrid adders (in addition to ripple-carry adders) have variable (data-dependent) delays and this variability can be exploited through use of an asynchronous design. Simulation results show that a 64-bit carry-select adder runs faster than its ripple-carry counterpart by 43%. When a complete system is considered, the systems that employ carry-select adders are 23% faster than their ripple-carry counterparts. It is also noted that an adder design which is well suited to the clocked environment (CSA) may not be a good option in the asynchronous environment. This is due to the fact that its worst-case and average-case delays are about the same with no gain in size.

Research is currently being pursued to refine the above models and analysis. Asynchronous adders are compared by running real code through the simulators. Other functions are being examined where the worst-case/average ratios are significant and operand-dependent execution variations exist. In these situations asynchronous designs may be advantageous. Furthermore, since most existing functional designs are based on minimizing the worst-case delays, function design techniques which attempt to minimize average delays need to be investigated.

References

[1] O.J. Bedrij. Carry-Select Adder. *IRE Trans. Electronic Computers*, 11:340–346, June 1962.

[2] B.E. Briley. Some New Results on Average Worst Case Carry. *IEEE Trans. Computers*, 22:459–463, May 1973.

[3] A.W. Burks, H.H. Goldstine, and J. von Neumann. Preliminary Discussion of the Logical Design of an Electronic Computing Instrument. In *Papers of John von Neumann on Computing and Computer Theory*, pages 97–142. The MIT Press, 1987.

[4] M.A. Franklin and T. Pan. Clocked and asynchronous instruction pipelines. In *Proc. 26th ACM/IEEE Symp. on Microarchitecture*, Austin, TX, December 1993.

[5] J.D. Garside. A CMOS VLSI Implementation of an Asynchronous ALU. In *Proc. IFIP Conf. on Asynchronous Design Methodologies*, Manchester, England, March 1993.

[6] H.C. Hendrickson. Fast High-Accuracy Binary Parallel Addition. *IRE Trans. Electronic Computers*, 9:465–469, December 1960.

[7] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Palo Alto, CA, 1990.

[8] K. Hwang. *Computer Arithmetic: Principles, Architecture, and Design.* Wiley, 1979.

[9] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, New York, 1978.

[10] M. Lehman and N. Burla. Skip Techniques for High-Speed Carry-Propagation in Binary Arithmetic Units. *IRE Trans. Electronic Computers*, 10:691–698, December 1961.

[11] N.M. Martin and S.P. Hufnagel. Conditional-Sum Early Completion Adder Logic. *IEEE Trans. on Comput.*, C-29(8):753–756, August 1980.

[12] C. Mead and L. Conway. *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, Reading, MA, 1980.

[13] T.H. Meng. *Synchronization Design for Digital Systems.* Kluwer Academic Publishers, Norwell. MA, 1991.

[14] G.W. Reitwiesner. The Determination of Carry-Propagation Length for Binary Addition. *IRE Trans. Electronic Computers*, 9:35–38, March 1960.

[15] J. Sklansky. Conditional-Sum Addition Logic. *IRE Trans. Electronic Computers*, 9:226–231, June 1960.

[16] D.F. Wann and M.A. Franklin. Asynchronous and Clocked Control Structures for VLSI Based Interconnection Networks. *IEEE Trans. Comput.*, pages 284–293, March 1983.

[17] S. Waser and M. Flynn. *Introduction to Arithmetic for Digital System Designers.* CBS College Pub., New York, 1982.

[18] A. Weinberger and J.L. Smith. A Logic for High-Speed Addition. In E.E. Swartzlander, editor, *Computer Arithmetic*, pages 47–56. Dowden, Hutchinson and Ross, 1980.

[19] S. Winograd. On the Time Required to Perform Addition. *JACM*, 12(2):277–285, April 1965.