Adders, Subtracters and Accumulators in XC3000

XAPP 022.000

Application Note By PETER ALFKE and BERNIE NEW

Summary

This Application Note surveys the different adder techniques that are available for XC3000 designs. Examples are shown, and a speed/size comparison is made.

Xilinx Family	***Demonstrates***
XC3000A / XC3100A	Adder Techniques

Introduction

There are many ways to implement binary adders, subtracters and accumulators in LCA devices. Various approaches offer different trade-offs between size and speed.

Most compact, but slowest, is a bit-serial technique that operates on one or two bits per clock cycle, generating sum and carry. The sum is fed to an output shift register; the carry is stored and used in the subsequent bit time.

The most compact combinatorial (parallel) adder, subtracter, or accumulator consists of cascaded CLBs. Each CLB implements a full adder, accepting one bit of each operand and an incoming carry. The CLB generates the sum and an outgoing carry. A 16-bit function is completed in 16 CLB delays, and requires 16 CLBs.

With its 5-input function generator, an XC3000 CLB can implement additions two bits at a time. Three CLBs can each handle two input bits of each operand and an input carry to generate the two sum outputs and an outgoing carry. A 16-bit function requires 24 CLBs but the operation is completed in eight CLB delays.

For faster operation, a look-ahead carry technique can be used. Made popular by the 74181 ALU and its descendents, look-ahead carry uses Carry Propagate and Carry Generate signals to reduce the ripple-carry delay. Using look-ahead carry techniques in the XC3000, a 16-bit addition can be completed in five CLB delays, using 30 CLBs.

An even faster conditional-sum algorithm was originally described by J. Sklansky. Using this algorithm, a 16-bit adder requires 41 CLBs, but settles in only three CLB delays. With careful layout, the propagation delay through such an adder can be less than 20 ns in an XC3100-3.

Note that all Xilinx adder structures can be used as accumulators with no size penalty. Unlike conventional gate arrays and similar structures, LCA devices provide dedicated flip-flops in each CLB that can be used for the accumulator register. Since the flip-flop set-up time through the function generator usually matches the combinatorial propagation delay of the CLB, the set-up time for accumulator operands is similar to the propagation delay of the adder.

Bit-Serial Adders

The CLB architecture is ideally suited for bit-serial arithmetic. As shown in Figure 1, the two operands are serialized in shift registers, and presented, LSB first, to the serial arithmetic unit. The sum is created as a serial bit stream, again LSB first, that is converted to parallel data in a third shift register. Alternatively, one of the input shift registers may serve as the output register, with the sum shifted in to replace the operand.

The arithmetic unit, Figure 2, comprises a 1-bit full adder/subtracter and a carry/borrow flip-flop, and can be implemented in a single CLB. Before commencing an operation (addition or subtraction) the carry/borrow flip-flop must be cleared. Subsequently, sum or differences are passed to the output shift register, while carries or borrows are stored for inclusion in the next bit of the serial operation.

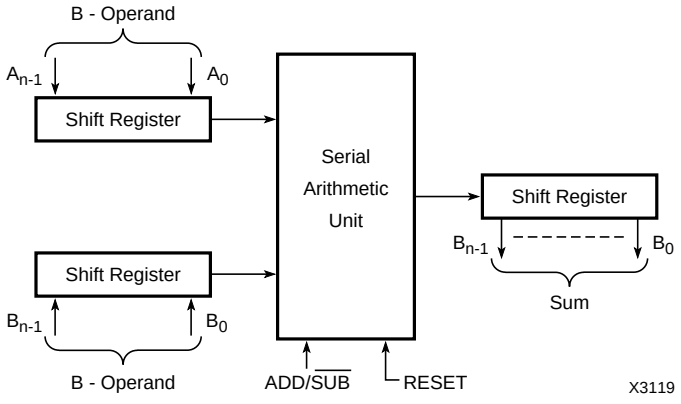

Figure 1. Serial Bit Adder/Subtracter

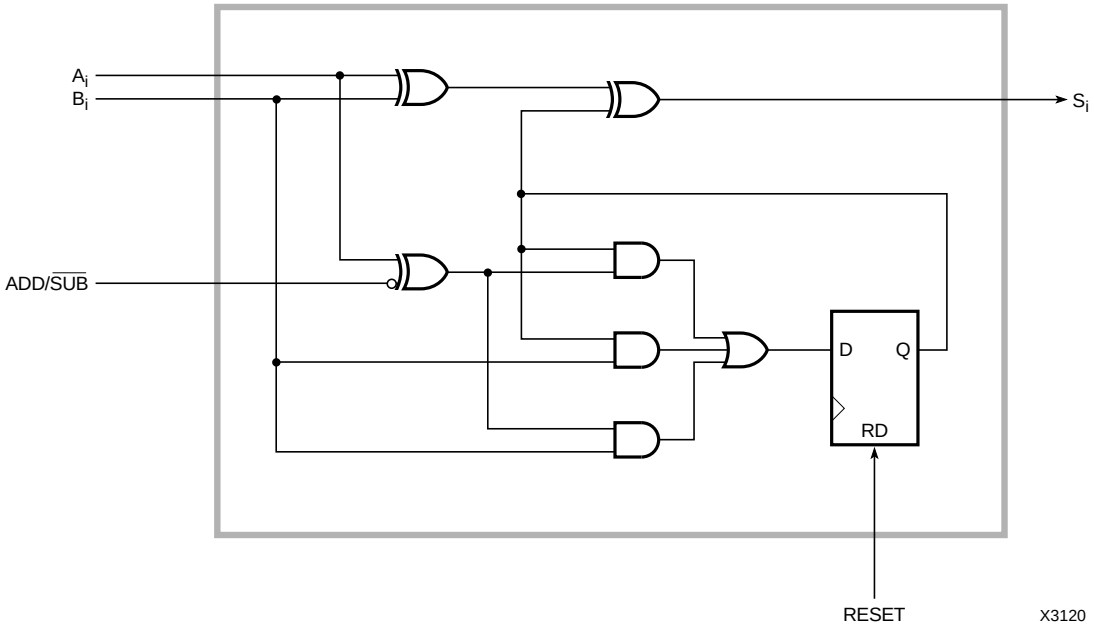

Figure 2. Serial Arithmetic Unit

While the number of clocks required to complete the operation equals the number of bits, the clock period can be very small because of the shallow logic. For maximum clock speed, the first bit of the output shift register should be implemented in the same CLB as the arithmetic unit.

Faster bit-serial operation can be obtained by simultaneously operating on two bits, Figure 3. Odd and even bits of each operand are loaded into separate shift registers. The arithmetic unit takes in two bits of each operand, and produces two sum bits per clock. These sum bits are loaded into odd and even output shift registers.

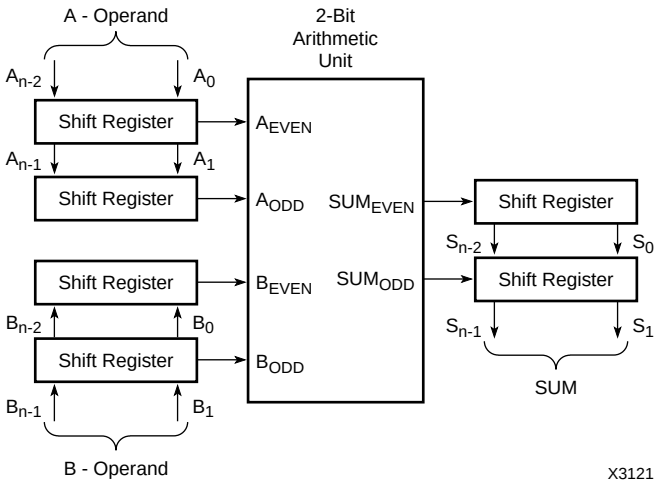

Figure 3. 2-Bit Serial Adder

Figure 4 shows the 2-bit arithmetic unit. Both sum bits are derived in parallel, and a single carry is generated and stored for the next cycle. This arithmetic unit permits adders and subtracters to be constructed, but not adders/subtracters. For adders/subtracter operation, the arithmetic unit should implement an adder; to generate A–B, the A-operand should be inverted while loading the operand shift register, and the sum bits should be inverted into the output register. The carry flip-flop is cleared before each operation, regardless of whether it is an addition or subtraction.

While the clock rate is similar to the 1-bit scheme, only half as many clocks are required to complete the operation.

Ripple-carry Adders

The 1-bit serial adder, described above, can easily be converted into a ripple-carry parallel adder. It is simply a matter of replicating the arithmetic unit once for each bit, removing the carry/borrow flip-flops and connecting the carry/borrow outputs from one bit to the next, Figure 5. The carry/borrow input of the LSB is set to zero for no carry in an addition, and for no borrow in a subtraction.

At one CLB per bit, this design uses fewer CLBs than any other parallel adder. However, this compactness is achieved at the expense of speed; the settling time is one CLB delay per bit. By placing the CLBs of the adder adjacent to each other, interconnect delay in the ripple path can be minimized, or even eliminated.

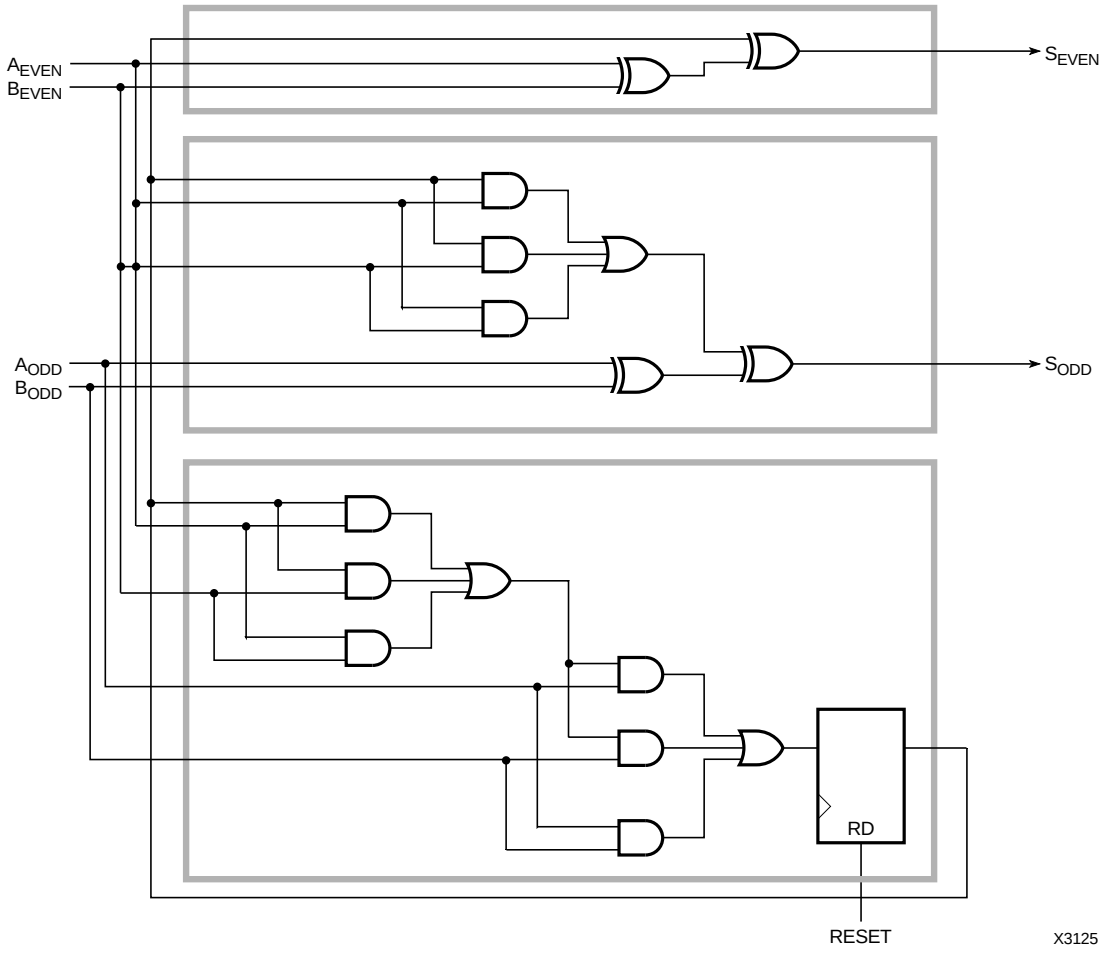

Figure 4. 2-Bit Serial Arithmetic Unit

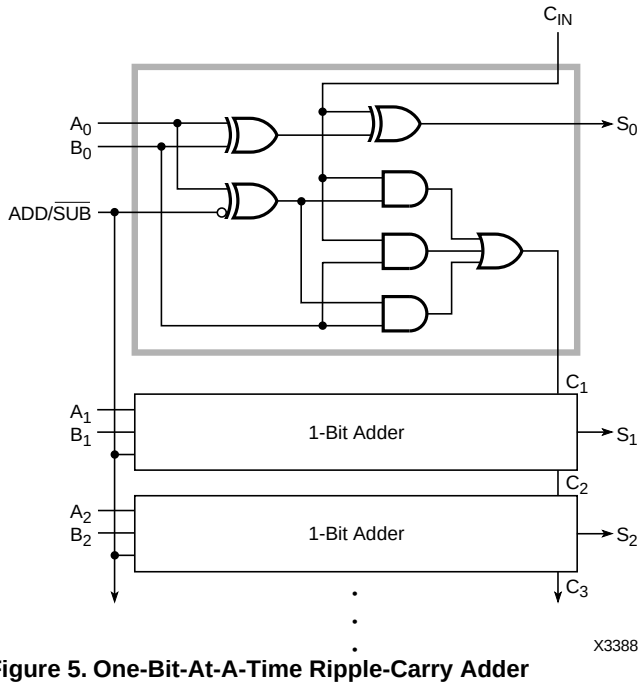

Figure 5. One-Bit-At-A-Time Ripple-Carry Adder

A faster settling time can be achieved by changing the replicated cell from a 1-bit adder to a 2-bit adder, Figure 6. The carry output and the more significant sum of each bit-pair are functions of five inputs. Consequently, each requires an entire CLB, increasing the CLB requirement to 1 1/2 per bit. However, the settling time is reduced to one CLB delay per two bits, half that of the previous design.

The 5-input function generators permit this design to be used for adders and subtracters, but not for adder/subtracters. To implement an adder/subtracter, one of the operands to an adder must be modified before being input into the adder.

For the operation A–B, there are two choices, both of which require additional XOR gates to invert one of the operands while subtracting. The technique used in the bit-serial adder and the one-bit-at-a-time adder is to invert the A-operand into the carry logic only; the A-operand is input to the sum logic unmodified. In this case, the carry/borrow input is active-high for both add and subtract, and may be tied Low if no input carry or borrow is required.

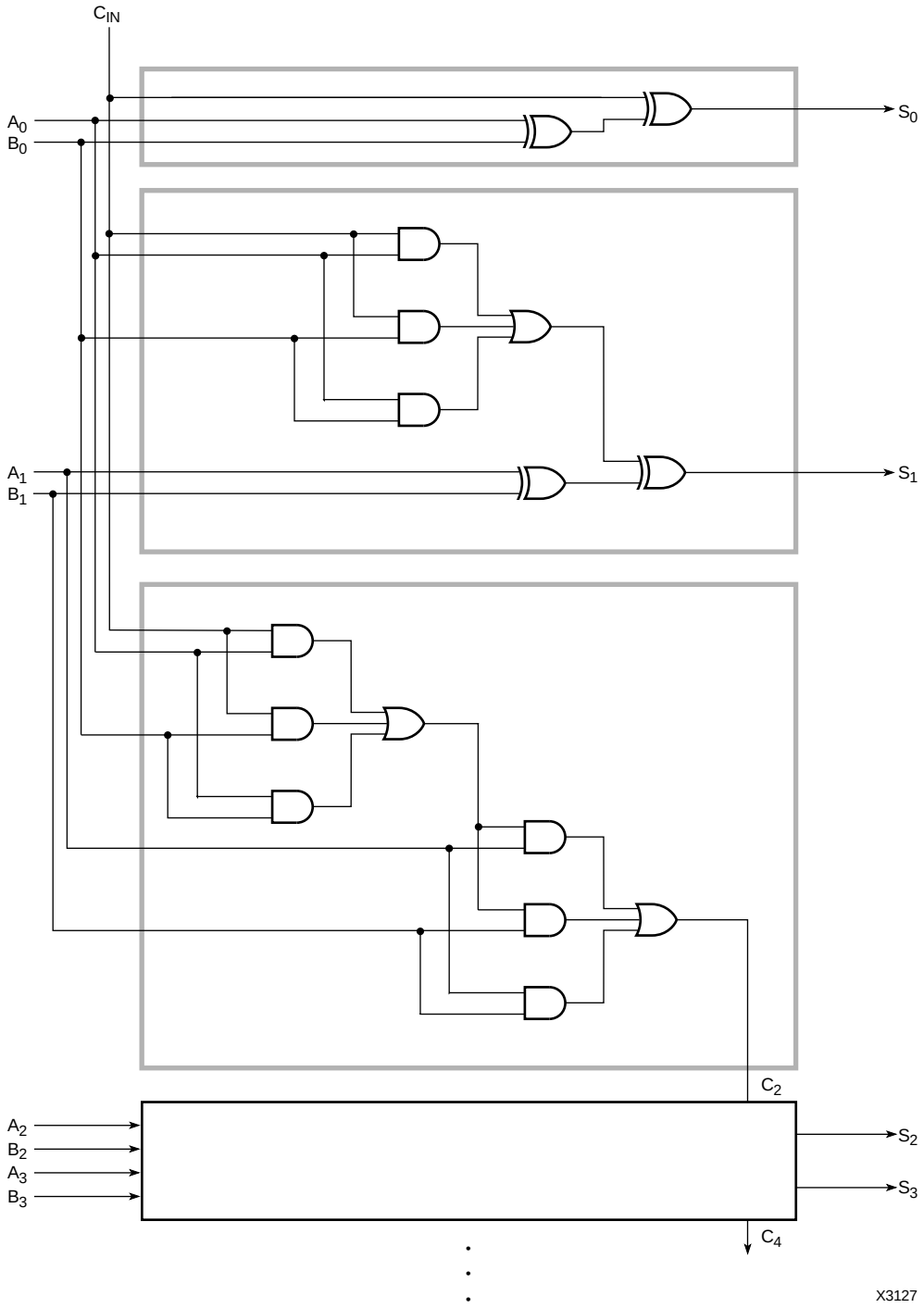

Figure 6. Two-Bits-At-A-Time Ripple-Carry Adder

A more conventional approach is to invert the B-operand into both the sum and carry logic. However, if no input borrow or carry is required, the input must be Low during an addition, and High during a subtraction.

Look-ahead-carry Adders

For faster operation in large adders, look-ahead carry look-ahead-carry technique uses two signals, Carry Generate and Carry Propagate (P and G), that are typically outputs of an arithmetic block, often of four bits. Since both of these signals do not depend on the incoming carry signal, they can be generated immediately from input data.

As the name implies, Carry Generate is asserted if the block creates an overflow (carry), regardless of incoming carry. For example, in a 4-bit adder, Carry Generate is asserted if the sum of the operand bits, excluding the incoming carry, exceeds 15.

If the block does not generate a carry by itself, but would generate a carry as a result of an incoming carry, Carry Propagate must be asserted; its assertion is optional if the block generates a carry without requiring an incoming carry. In our 4-bit example, Carry Propagate must be asserted when the sum, excluding the incoming carry, is exactly 15, and may optionally be asserted when the sum is greater.

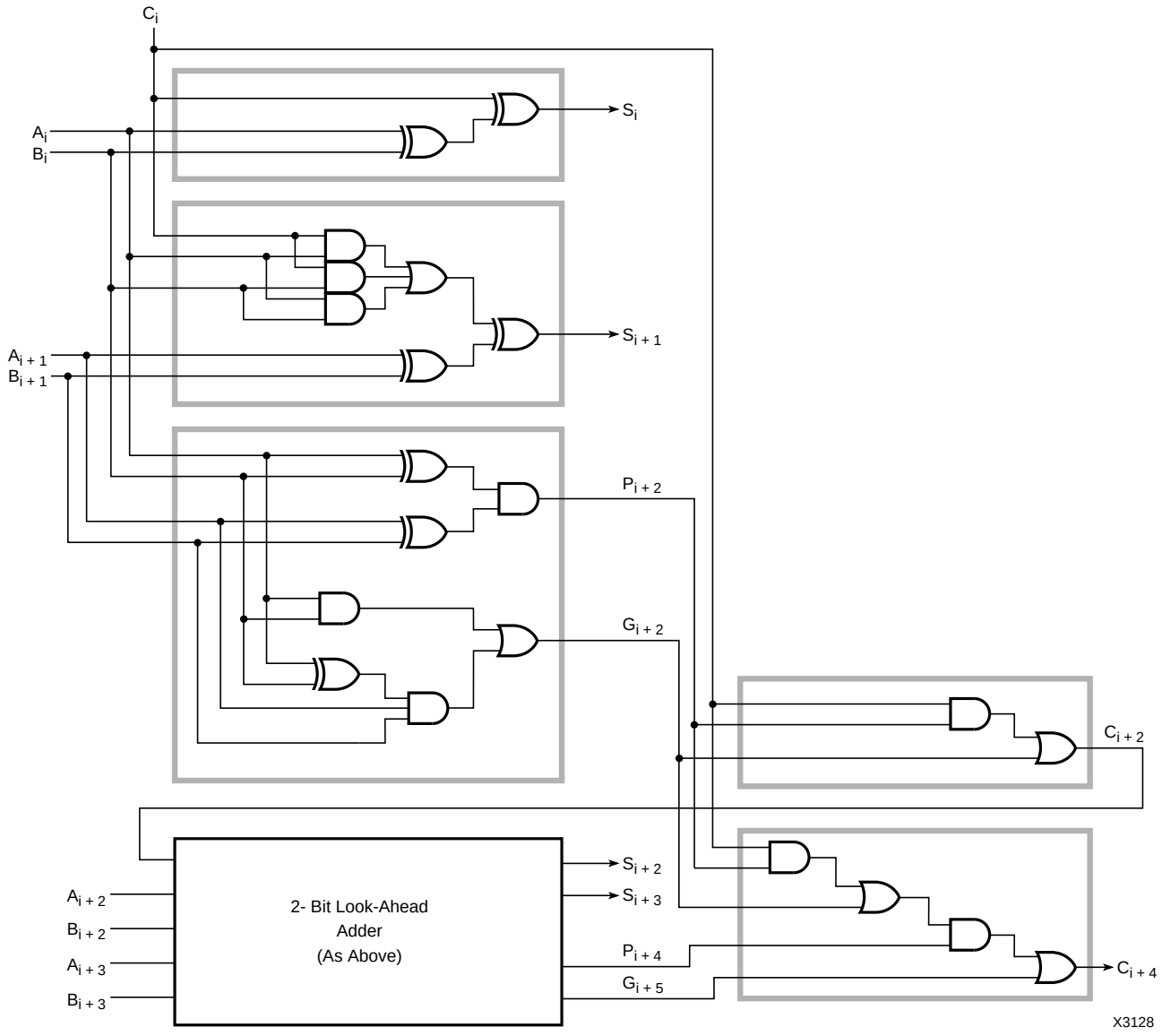

Figure 7. Four-Bits-at-a-time Adder Block with Internal Look-Ahead Carry

In XC3000 LCA devices, look-ahead carry is most effective when used to combine two 2-bit blocks into a 4-bit block that cascades using ripple carry, Figure 7. The 4-bit block has a one-CLB delay from carry in to carry out, but a two-CLB delay from carry in to the sum output of the more significant bit-pair. The delay from the operand inputs to the carry output is also two CLBs.

A 16-bit adder may be implemented in two ways. The most straightforward way is to cascade four 4-bit blocks, as shown in Figure 8(a). With this design, the carry-in-to-carry-out delay is only four CLBs, while the operand-to-sum delay is six CLBs; the operand-to-carry-out and carry-in-to-sum delays are both five CLBs The carry output is available one CLB delay before the sum, and the carry input need not be present until one CLB delay after the operands. The design requires 32 CLBs.

While a shorter carry delay may sometimes be desirable, the design in Figure 8(b) is faster overall, balancing all four delays at five CLBs. The 2-bit ripple-carry block, described in the ripple-carry section, is used to implement the most and least significant bit-pairs, and only 30 CLBs are required.

Either design can be adapted to any multiple of four bits by simply adding or subtracting 4-bit blocks in the center of the adder. The advantage over the 2-bit ripple-carry technique increases with the number of bits in the adder.

For even numbers of bits that are not multiples of four, any of the designs in Figure 9 may be used. For a 14-bit adder, the Figure 9(a) design balances all four delays at five CLBs, and requires 25 CLBs. The Figure 9(b) and 9(c) designs each use two additional CLBs, but are one CLB delay faster in the carry path. In the Figure 9(b) design the carry out appears one CLB delay before the sum, and in the Figure 9(c) design, the carry in need not be present until one CLB delay after the operand. Again, for different length adders, simply add or subtract 4-bit blocks at the center of the adder.

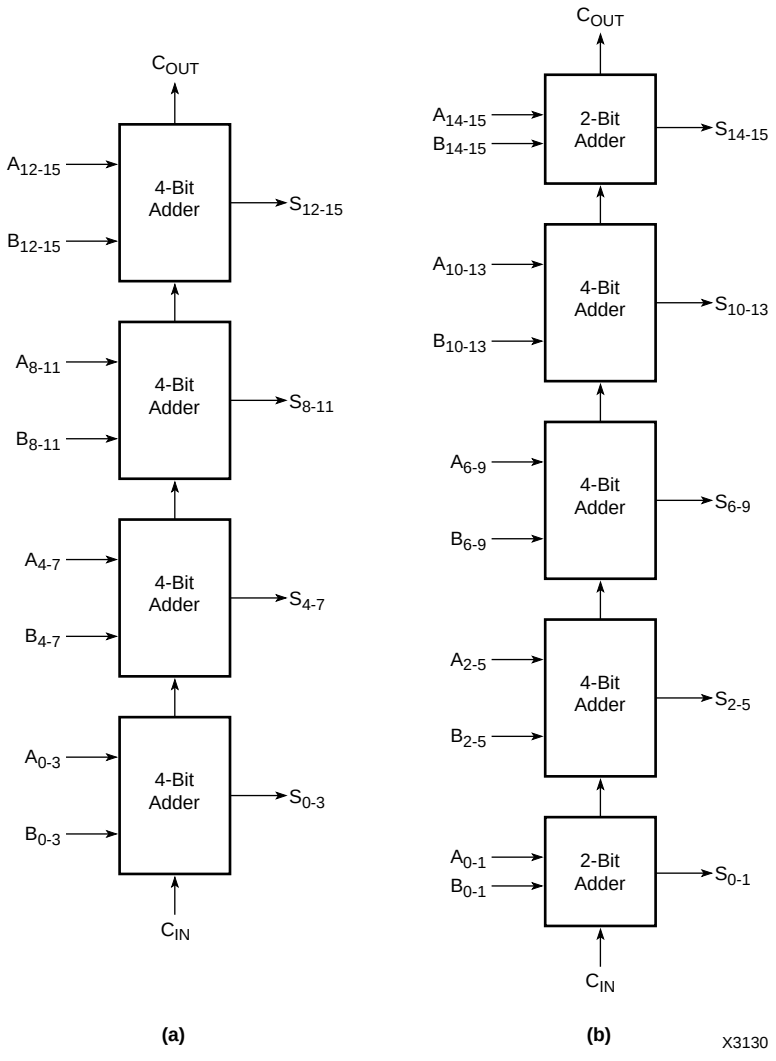

Figure 8. 16-Bit Adder Configurations

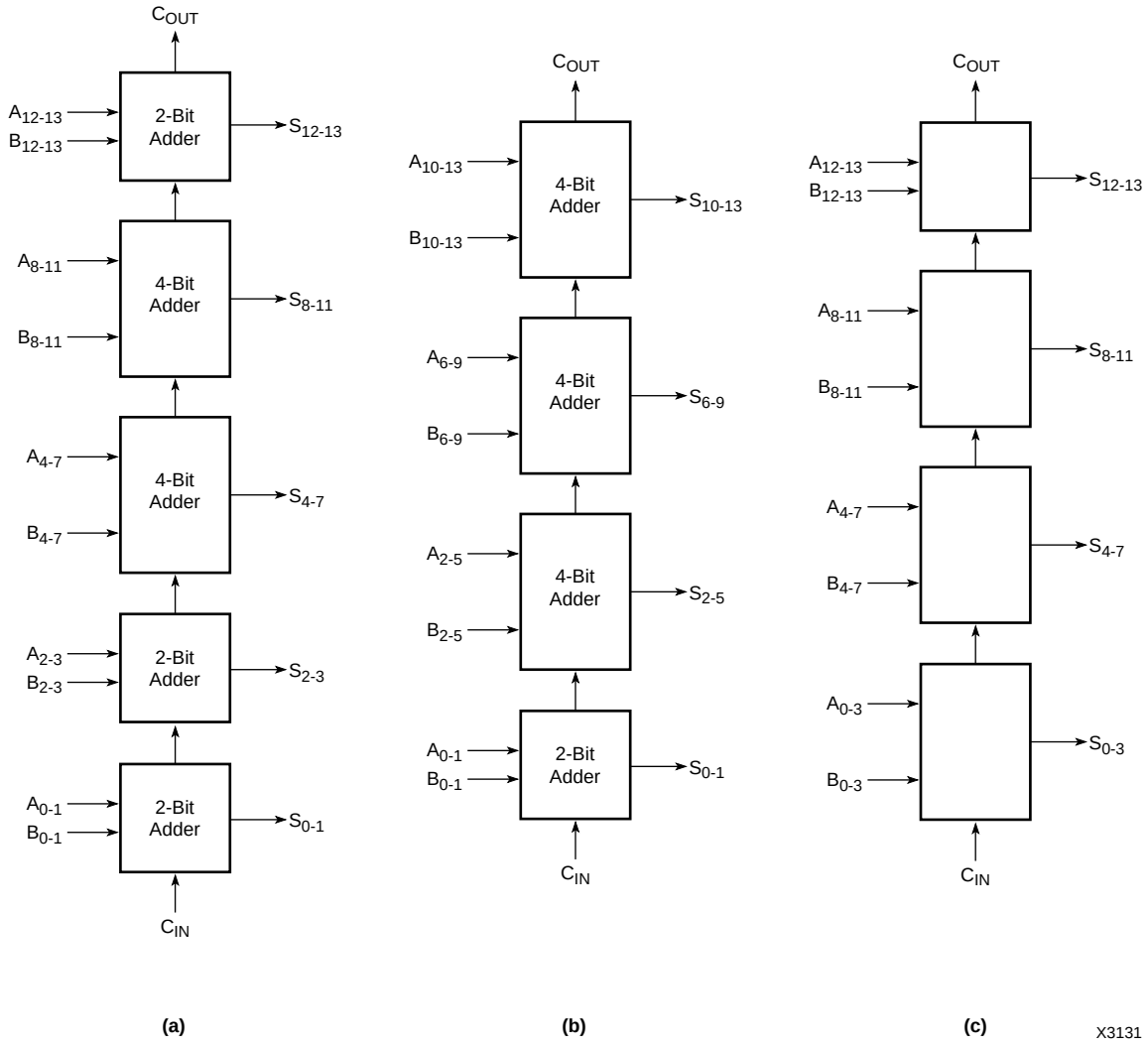

Figure 9. 14-Bit Adder Configuration

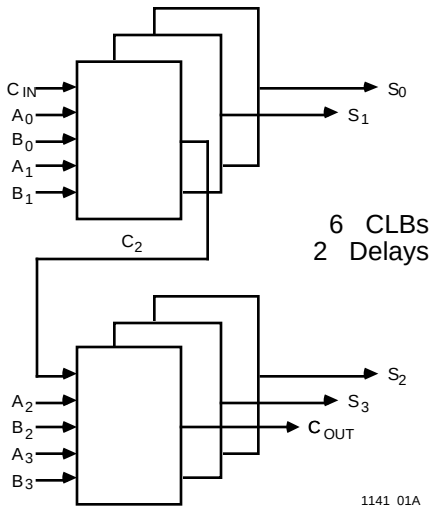

Figure 10. 4-Bit Adder

Conditional-sum Adder

Conditional-sum adders, originally described by J. Sklansky in the June 1960 issue of the IRE Transaction on Electronic Computers, reduce settling time at the expense of much higher logic complexity. The version described below was created by Matt Klein of Hewlett Packard, who modified the algorithm to fit the XC3000 architecture. With careful placement and routing, the total delay can be kept below 20 ns in an XC3100-3.

Forty-one CLBs are required, 27 of which generate one function of up to five variables, while the remaining 14 CLBs each generate two functions of four variables. Figure 10 shows how these CLBs are connected. For more information, please refer to the original paper and the Xilinx Technical Bulletin Board.