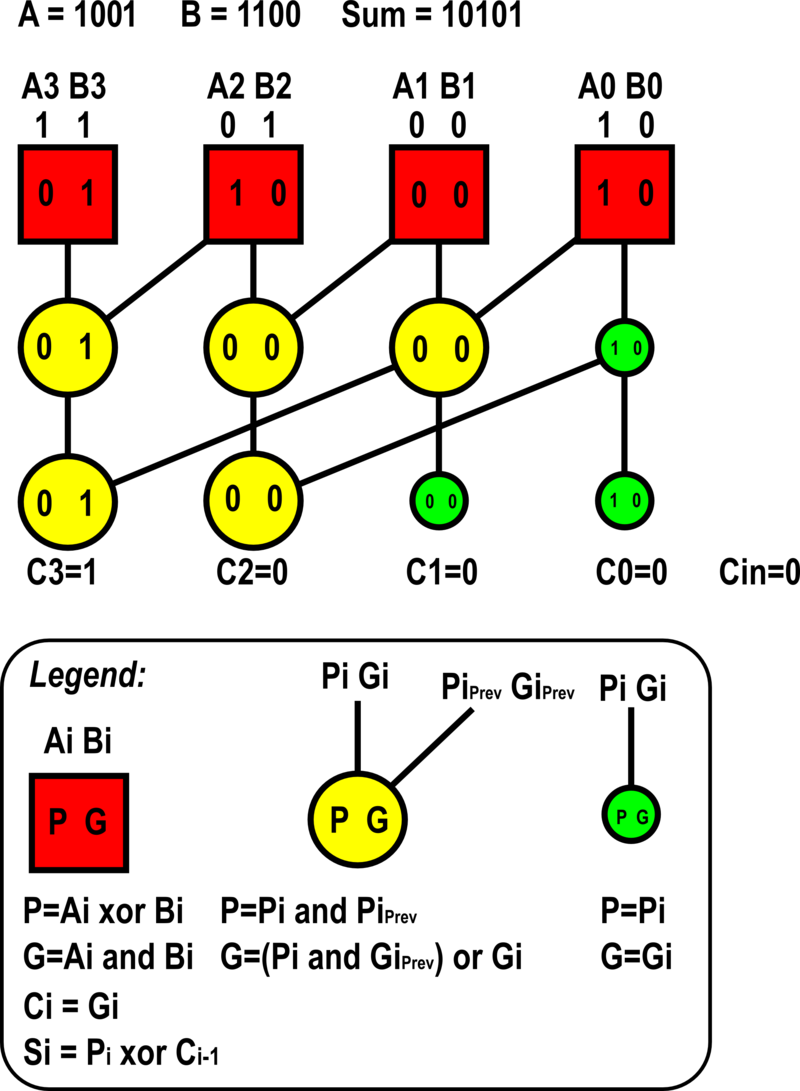
**High-Speed Adders**

**Kogge-Stone Adder:**

The Kogge-Stone adder (KSA) is a parallel prefix form carry look-ahead adder. It is widely considered to be the fastest adder and is widely used in industry for high-performance arithmetic circuits. Parallel prefix adders are used because they pre-compute and generate the necessary signals, they have less delay compared to traditional adders, and they also work with the associative law.

The functioning of a KSA can be understood by analyzing in in terms of three vertical stages: the pre-processing stage, the prefix stage, and the final computation stage. Each stage produces a “propagate” and a “generate” bit. A schematic of a 4-bit KSA with sample inputs is shown in Figure 1 below.



**Figure 1:** The schematic of a 4-bit KSA with inputs and , and a carry-in of zero ().

Pre-processing stage:

The pre-processing stage is illustrated by the red cells in Figure 1. The calculations of (the propagate bit) and (the generate bit) are based on the following two equations, where :

Prefix stage:

The yellow cells make up the prefix stage in Figure 1. The calculations of and are based on the following equations:

Note that and here refer to the propagate and generate bits directly vertical to the current cell, while and are diagonal.

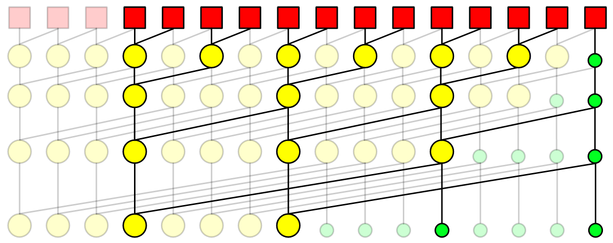
The green cells in Figure 1 can be thought of as intermediate steps of the prefix stage. The propagate and generate bits are simply the corresponding bits directly vertical to the cell:

Final computation stage:

The sum and carry bits are computed in the final stage. Each carry bit corresponds directly to a generate bit while each sum bit is a function of the propagate bit and the previous carry bit:

Expansion:

The KSA can be expanded to support higher-bit additions. The schematic of a 16-bit adder is shown in Figure 2 below.



**Figure 2:** The schematic of a 16-bit KSA.

Note how with each passing level in the prefix stage the distance between a cell and the connected diagonal cell in the previous level increases by , where is the current level number with the first level being one. Similarly, each level in the prefix stage has green cells, or cells where the propagate and generate bits directly above are simply carried through. Finally, the number of required levels in the prefix stage can be computed by taking the base-2 logarithm of the adder size ():

Enhancements:

Enhancements to the original implementation include increasing the radix and sparsity of the adder. The *radix* of the adder refers to how many results from the previous level of computation are used to generate the next one. The original implementation uses radix-2, although it is possible to create radix-4 and higher. Doing so increases the power and delay of each level, but reduces the number of required levels. In the sparse KSA, the *sparsity* of the adder refers to how many carry bits are generated by the carry tree. Generating every carry bit is called sparsity-1, whereas generating every other is sparsity-2 and every fourth is sparsity-4. The resulting carries are then used as the carry-in inputs for much shorter ripple carry adders or some other adder design, which generates the final sum bits. Increasing sparsity reduces the total needed computation and can reduce the amount of routing congestion. Figure 2 is an example of a KSA with sparsity-4. Cells eliminated by sparsity are shown as transparent. The power and area of the carry generation is improved significantly, and routing congestion is substantially reduced. Each generated carry feeds a multiplexer for a carry select adder or the carry-in of a ripple carry adder.

4-Bit KSA Implementation:

Sample Verilog code for the implementation of the 4-bit KSA is provided below.

S0 = (A0^B0)^Cin;

S1 = (A1^B1)^((A0&B0)|(A0^B0)&Cin);

S2 = (A2^B2)^(((A1^B1)&((A0&B0)|(A0^B0)&Cin))|(A1&B1));

S3 = (A3^B3)^((((A2^B2)&(A1^B1))&((A0&B0)|(A0^B0)&Cin))|(((A2^B2)

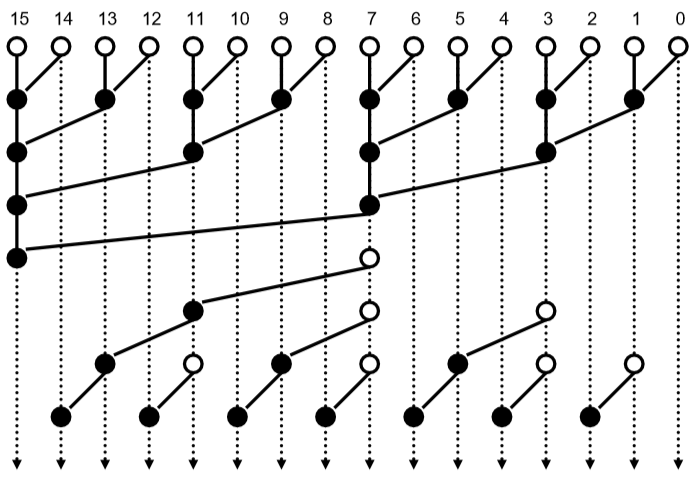
&(A1&B1))|(A2&B2)));

Cout = (A3&B3)|(A3^B3)&((((A2^B2)&(A1^B1))&((A0&B0)|(A0^B0)&Cin))

|(((A2^B2)&(A1&B1))|(A2&B2)));

**Brent-Kung Adder:**

The Brent-Kung adder (BKA) is similar to the KSA in that the pre-processing, prefix stage, and final processing stage cells are calculated in the exact same way. The difference between the two adders lies in the fan-out and the number of cells required for implementation. For instance, take the schematic of a 16-bit BKA shown in Figure 3 below.



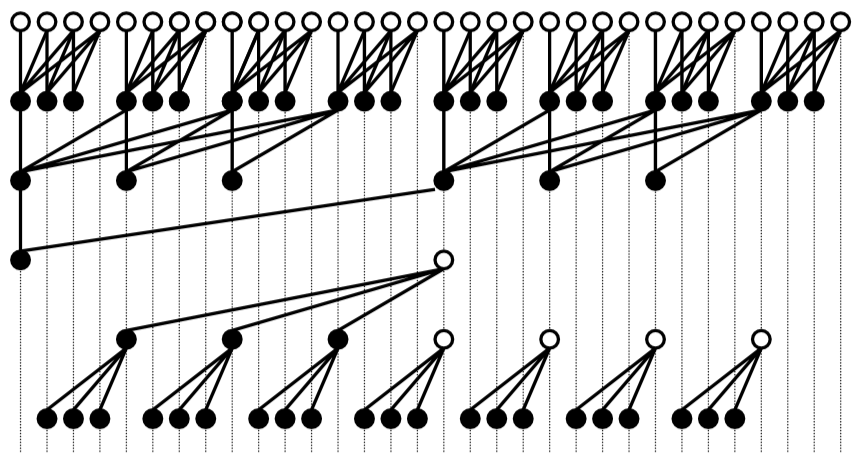
**Figure 3:** The schematic of a 16-bit BKA.

The white cells on the top row of Figure 3 constitute the pre-processing stage while the black cells make up the prefix stage. The white cells in the middle are the intermediate steps of the prefix stage. Comparing this schematic to that of the 16-bit KSA in Figure 2, it can be seen that the BKA requires fewer cells to implement than the KSA, which makes the BKA a simpler design. This also means that the BKA consumes less power than the KSA. However, the BKA has a larger fan-out than the KSA (while the fan-out of any KSA is only limited to 2, the fan-out of BKA can go as high as , where is the adder size). A large fan-out may not a good thing as it may split and weaken the current propagating through the adder. Furthermore, the BKA requires two additional logic level, increasing delay.

A final observation of the BKA concerns the bit-location of the maximum fan-out. It can be seen in Figure 3 that bit 7 has the greatest fan-out at 5. Bit 7 also happens to be just before the mid-way point of 16 bits. This holds true for any BKA; the bit with the highest fan-out is located at , with being the fan-out. Again, refers to the adder size. This bit may also be thought of as the “trunk” of the inverse carry tree that constitutes the bottom half of the BKA.

Enhancements:

Just like the KSA, the BKA can be enhanced by increasing the radix. Doing so can reduce the complexity of the design by reducing the number of cells or logic levels require; however, increasing radix in a BKA may also increase fan-out. An example of a mixed-radix 32-bit BKA is shown in Figure 4 below.

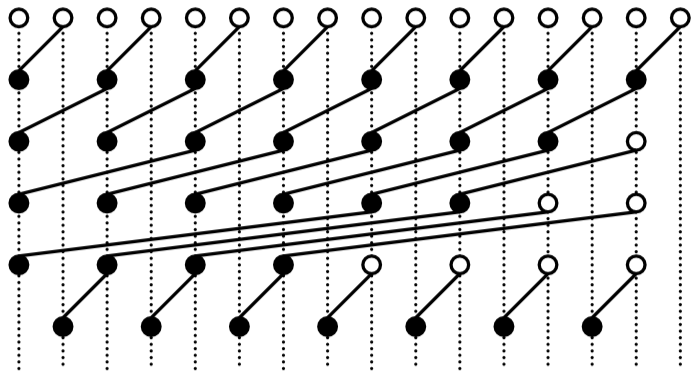


**Figure 4:** The schematic of a mixed-radix 32-bit BKA.

While increasing the radix of the first level of the prefix stage to 4 has reduced the number of required cells from 57 down to 55 for the BKA in Figure 4, it has also increased the maximum fan-out from 6, as a 32-bit BKA would normally have, to 8.

**Han-Carlson Adder:**

The Han-Carlson adder (HCA) can be thought of as a sparsity-2 KSA. An example of a 16-bit HCA is shown in Figure 5 below.

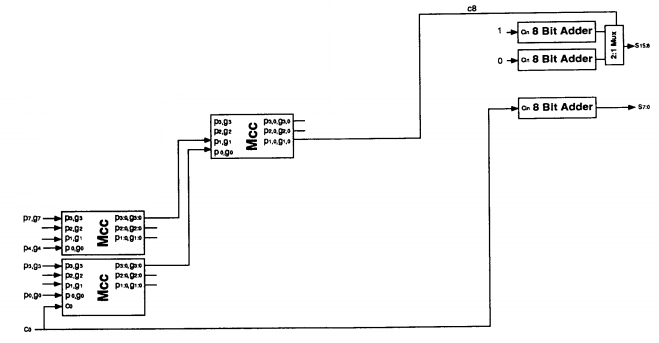


**Figure 5:** The schematic of a 16-bit HCA.

The white cells on the top row of Figure 5 constitute the pre-processing stage while the black cells make up the prefix stage. The white cells in the middle are the intermediate steps of the prefix stage. While the HCA halves the number of wires required at every level of the prefix stage (compare with Figure 2), reducing complexity, it also adds an additional logic level. This additional logic layer increases delay. On the other hand, the HCA requires fewer prefix stage cells to implement than the KSA (32 vs. 49 for a 16-bit adder, for example), reducing power consumption.

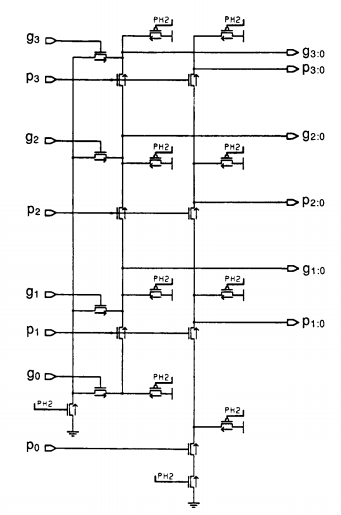
**Spanning Tree Adder:**

The schematic of a 16-bit Spanning Tree adder (STA) as described by Lynch and Swartzlander is shown in Figure 6 below.



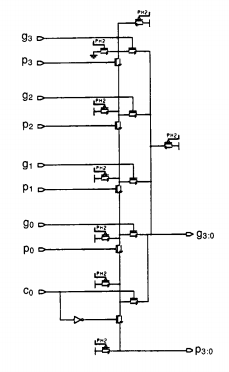
**Figure 6:** The schematic of a 16-bit STA.

Figure 6 above shows that the STA relies on multiple “MCC” modules to generate the carry bit (c8). The MCC or Manchester carry chain is a variation of the carry-lookahead adder that uses shared logic to lower the transistor count. An MCC generates the intermediate carries by tapping off nodes in the gate that calculates the most significant carry value. Typically, the input to an MCC does not exceed 4 bits for sizing and delay purposes. The schematic of an MCC is shown in Figure 7 below.



**Figure 7:** The MCC module.

An MCC module may also have a carry-in as an input (bottom-left MCC module in Figure 6), as shown in Figure 8 below.



**Figure 8:** The MCC module with a carry-in.

The propagate and generate bits shown in Figures 7 and 8 are calculated the same way as before. In fact, these schematics demonstrate how the bits are calculated at the transistor level! In this example, the carry bit c8 in Figure 6 is the generate bit g1:0 from the MCC module in the second column. This is no different from before where the carry bits were given by the last calculated generate bits (i.e. ).

Compared to the KSA, the STA is smaller, has a lower fan-out, and needs less wiring. On the other hand, the STA uses an MCC, and one of its major downsides is that the capacitive load of all of these outputs, together with the resistance of the transistors causes the propagation delay to increase much more quickly than a regular carry lookahead.

**Comparisons Between Adders:**

Table 1 below summarizes the advantages each adder introduced in this document over the other. Advantages are organized according to row over column.

**Table 1:** Summary of Advantages

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Kogge-Stone Adder | Brent-Kung Adder | Han-Carlson Adder | Spanning Tree Adder |
| Kogge-Stone Adder | --- | * Lower fan-out * Less delay | * Less delay | * Less delay (as adder size increases) |
| Brent-Kung Adder | * Less wiring * Fewer cells to implement | --- | * Less wiring * Fewer cells to implement | * Less delay (as adder size increases) |
| Han-Carlson Adder | * Less wiring * Fewer cells to implement | * Lower fan-out * Less delay | --- | * Less delay (as adder size increases) |
| Spanning Tree Adder | * Lower fan-out * Less wiring * Fewer cells to implement | * Lower fan-out | * Lower fan-out | --- |

Some conclusions can be made from the results in Table 1:

* The KSA is the fastest adder of the bunch, but requires the most wiring and cells to implement.
* The BKA has the highest fan-out.
* The HCA is simpler to implement than the KSA, but it is slower. It is a trade-off of reduced speed for reduced complexity/power consumption.
* The STA has the lowest fan-out and it might possibly require the fewest wires and cells for implementation. However, its propagation delay increases rapidly as the adder size increases.