

Carry-Lookahead Adders

Arithmetic logic blocks, such as adders and subtractors, are increasingly becoming performance bottlenecks in high-performance logic designs. Carry-lookahead adders are generally faster than cascaded adders because they reduce the time needed to generate carry propagation. The carry-lookahead can be achieved if the input carry bit for stage i is generated directly from the inputs to the preceding stages $i - 1, i - 2, \dots, i - k$ rather than allowing the carry bit to be cascaded and rippled from stage to stage. An n - bit carry-lookahead adder can be constructed using k stages, each of which is a full adder stage modified by replacing its carry output line $c0$ by two signals called carry generate and propagate. These signals gi and pi are defined by the following logic:

$$gi = ai \cdot bi$$

$$pi = ai + bi$$

That is, a stage will generate a carry if both of its addend bits are '1', and it propagates carries if at least one of its addend bits is '1'.

Therefore, the carry signal that will be generated for the stage $i + 1$ is defined as follows from the generate and propagate signals:

$$ci + 1 = gi + pi \cdot ci$$

If we recursively expand the ci term for each stage, and multiply out to obtain a two-level sum-of-products expression we can eliminate the carry ripple that is associated with cascaded adders. If this technique is followed, we can obtain equations for the carry out bits for each stage as shown below.

$$c1 = g0 + p0 \cdot c0$$

$$c2 = g1 + p1 \cdot c1$$

$$= g1 + p1 (g0 + p0 \cdot c0)$$

$$= g1 + p1 \cdot g0 + p1 \cdot p0 \cdot c0$$

$$c3 = g2 + p2 \cdot c2$$

$$= g2 + p2 (g1 + p1 \cdot g0 + p1 \cdot p0 \cdot c0)$$

$$= g2 + p2 \cdot g1 + p2 \cdot p1 \cdot g0 + p2 \cdot p1 \cdot p0 \cdot c0$$

$$c4 = g3 + p3 \cdot c3$$

$$= g3 + p3 (g2 + p2 \cdot g1 + p2 \cdot p1 \cdot g0 + p2 \cdot p1 \cdot p0 \cdot c0)$$

$$= g3 + p3 \cdot g2 + p3 \cdot p2 \cdot g1 + p3 \cdot p2 \cdot p1 \cdot g0 + p3 \cdot p2 \cdot p1 \cdot p0 \cdot c0$$

Each of the above equations corresponds to a circuit with only three levels of delay associated with it – one for the generate and propagate signals, and two for the sum-of-products shown. A carry-lookahead adder uses three-level equations such as these in each adder stage.

Building Blocks of an n -Bit Carry Lookahead Adder

F3ADD (F3ADD_1, F3ADD_2): A three-bit full adder with propagate and generate outputs

PG1 .. PG4: Carry/Borrow bit generator utilizing propagate and generate inputs

F3ADD: The F3ADD macro shown below performs the three-bit addition of $a0 .. a2 + b0 .. b2$, it also performs the propagate and generate functions. The propagate function determines if any of the addend bits are '1' by Oring each set of addend bits. If any of the addend bits are a one a propagate will be generated.

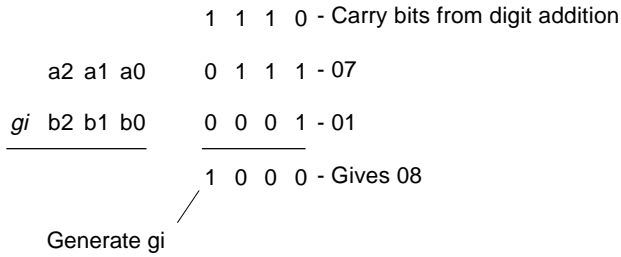
As shown below, we see that when either of a digit's addend bits are one, a propagate will be generated:

	a2	a1	a0	0	1	1
	b2	b1	b0	0	0	1
propagate pi	a2+b2	a1+b1	a0+b0	0	1	1

Note: only a single propagate will be produced although more than one may be generated as shown. The generate function determines if a carry to the next digit should be generated from the previous digit addition, as shown earlier in the binary addition basics section. Essentially, it performs the same function as the carry out of a regular adder, but does not incorporate the carry in signal in its logic. An example of how a generate bit is produced is shown below.

Adder and Subtractor Macros Using Lattice Design Tools

Here we show a three-bit addition with a generate being produced:

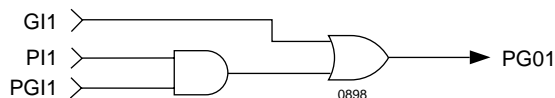


gi will be generated because $= a0 \cdot a1 \cdot a2 \cdot b0$; here we can see that if $a0 \cdot b0 = 1$ a carry will be generated for those two addend bits. Therefore, if that carry propagates to the next set of bits $a1 \cdot b1$ and either of them are a one a carry will be generated to the next set. This function is recursive and if you look at the logic for the generate function below you will see that all combinations have been accounted for.

$$\begin{aligned}
 gi &= (a0 \cdot a1 \cdot a2 \cdot b0 \\
 &\# a1 \cdot a2 \cdot b1 \\
 &\# a0 \cdot a2 \cdot b0 \cdot b1 \\
 &\# a2 \cdot b2 \\
 &\# a0 \cdot a1 \cdot b0 \cdot b2 \\
 &\# a1 \cdot b1 \cdot b2 \\
 &\# a0 \cdot b0 \cdot b1 \cdot b2)
 \end{aligned}$$

Figure 1. Logic and Truth Table for the PG1 Macro

PGI1	PI1	G11	PG01
x	x	1	1
1	1	x	1



PG1 .. PG4: The propagate-generate macros are carry bit generators utilizing propagate and generate inputs. The logic and truth table for the PG1 macro is shown in Figure 1.

The PGI1 input of the macro is the carry in input from the initial stage of your adder, the PI1, and G11 inputs are the propagate and generate inputs associated with the F3ADD macro outputs. Looking at the logic of the PG1 macro it can be seen that whenever a generate bit was produced from the F3ADD macro a carry out signal will be generated from the PG macro, assuming the PG macro is using the inputs from the F3ADD macro outputs. This also holds true if both of the propagate inputs into the PG1 are '1' since the initial carry in would be a '1' and one of the addend bits is '1', this would result in a carry out. Basically, the PG1 macro performs the function associated with the carry out of a regular adder.

As shown previously in the carry-lookahead adder section, the carry signal that will be generated for the stage $i + 1$ is defined as follows from the generate and propagate signals:

$$ci + 1 = gi + pi \cdot ci \quad (\text{where } ci + 1 = PG01, gi = G11, pi = P11, \text{ and } ci = PG11)$$

$$\text{Therefore: } PG01 = G11 + P11 \cdot PG11$$

If we recursively expand the PG01 term for each stage, i.e. PG02 .. PG0n and multiply out to obtain a two-level sum-of-products expression, we can eliminate the carry ripple that is associated with cascaded adders. If this technique is followed, we can obtain equations for any PG0n bits for each stage as shown previously, but substituting the generate and propagate signals into the ci equations as we did above.

For stage two (macro PG02):

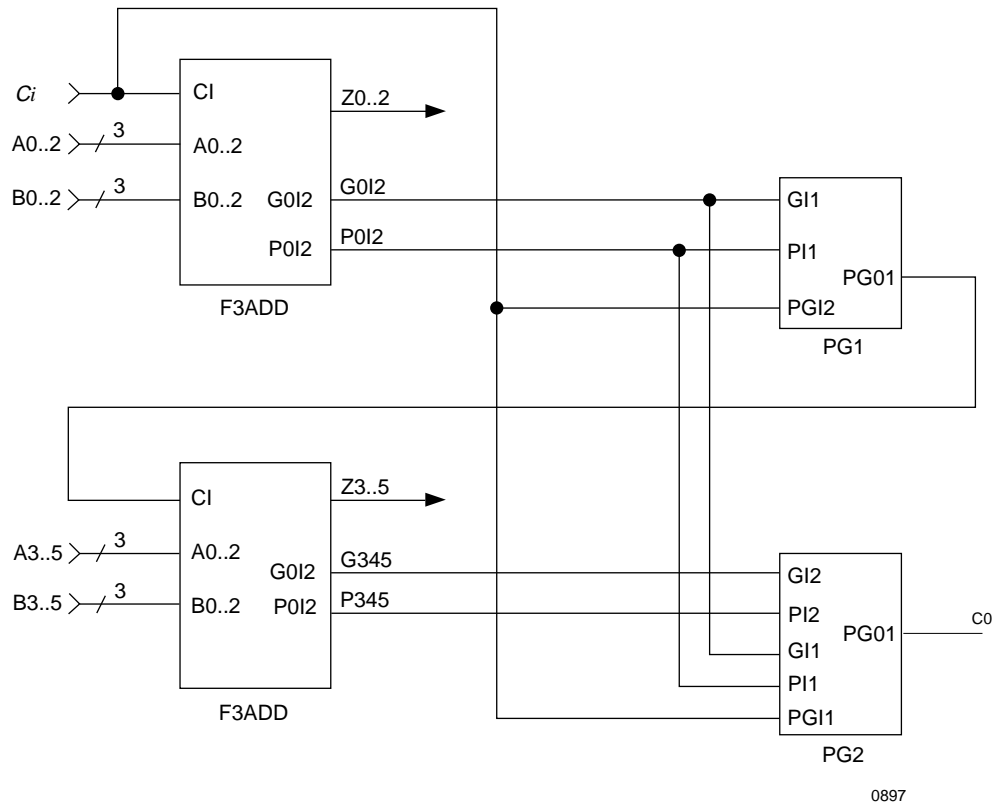
$$\begin{aligned}
 c2 &= g1 + p1 \cdot c1 \\
 &= g1 + p1 (g0 + p0 \cdot c0) \\
 &= g1 + p1 \cdot g0 + p1 \cdot p0 \cdot c0
 \end{aligned}$$

$$PG02 = G12 + P12 \cdot G11 + P12 \cdot P11 \cdot PG01$$

Here we show a six-bit adder utilizing two F3ADD macros, PG1 and PG2. Remember that the propagate and generate inputs to the PGn macro is associated with that stages adders outputs. That is, for PG02, G11 and P11 would come from the first adder, and G12 and P12 would come from the second adder.

Adder and Subtractor Macros Using Lattice Design Tools

Figure 2. Six-Bit Adder

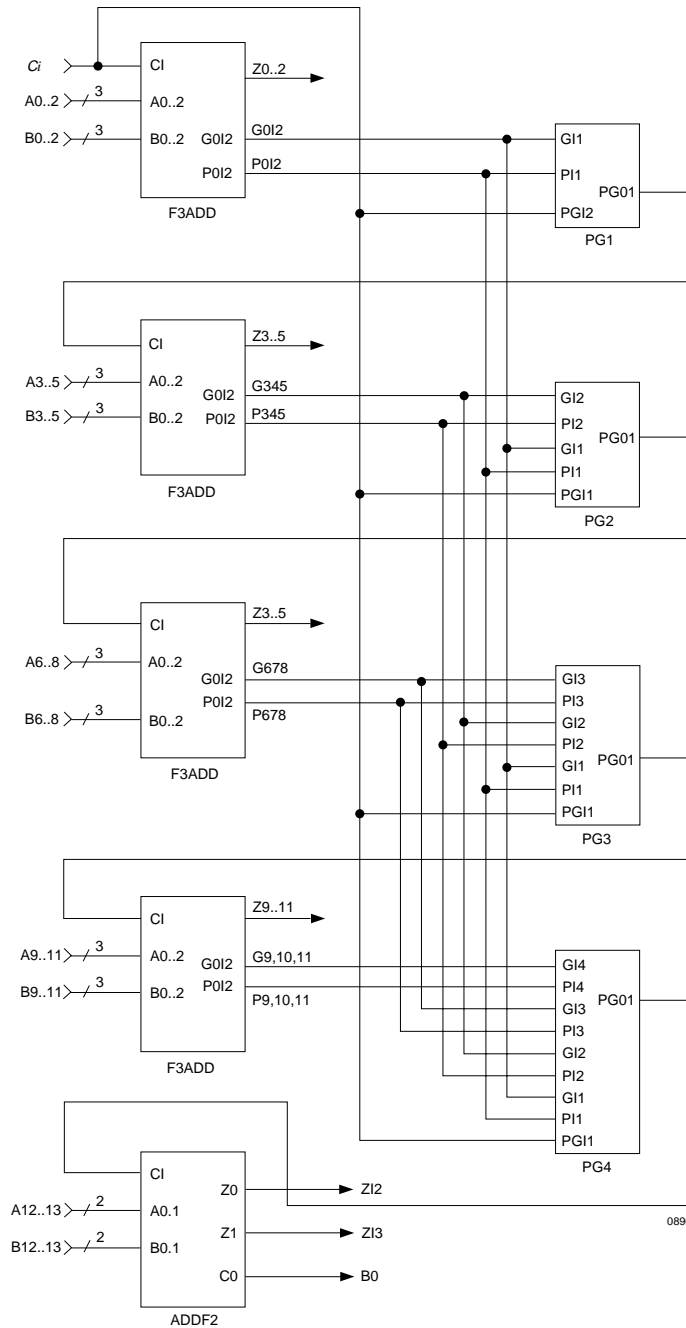


Adder and Subtractor Macros Using Lattice Design Tools

In the case of the six-bit adder, the co will be the PG02 of the second propagate and generate macro PG2. But for adders that do not have a multiple of three, one of the other regular adders should be used to account for the

extra bits and the ci of the adder would be driven by the last PG0n in that network. This is shown in the figure below.

Figure 3. 14-Bit Adder



Adder and Subtractor Macros Using Lattice Design Tools

Subtractors

F3SUB (F3SUB_1, F3SUB_2): Three-bit full subtractor with propagate and generate outputs

PG1 .. PG4: Carry/Borrow bit generator utilizing propagate and generate inputs

The same convention that was followed with the adders is followed with the subtractors. The only difference is that a borrow in is used instead of a carry in. The subtraction technique is shown in the subtractor basics section, and the propagate-generate macros are identi-

cal to those used in the adder section. As shown below in the 14-bit subtractor, the borrow bit is generated by each of the PG0n macros whereas a carry bit was generated with the adders. This bit then propagates through the subtractors.

Technical Support Assistance

Hotline: 1-800-LATTICE (Domestic)
1-408-826-6002 (International)
e-mail: techsupport@latticesemi.com

Figure 4. 14-Bit Subtractor

