The Delta39K™/Quantum38K™ Carry Chain

Introduction

Delta39K™ and Quantum38K™ are two revolutionary Complex Programmable Logic Device (CPLD) families offered by Cypress Semiconductor. Delta39K includes abundant logic and memory resources, an embedded PLL, and configurable I/O standards. Quantum38K is a high-density CPLD specifically designed for high-volume, low-cost applications. The Delta39K and Quantum38K architectures are based on an array of logic block clusters. Each logic block cluster has 128 macrocells and an architecture based on the popular Ultra37000 device family. An important improvement made in the cluster is the addition of a carry chain structure. The carry chain allows arithmetic and other similar functions to be implemented with fewer resources and operate at a higher speed than previous CPLDs. This application note discusses the architecture of the Delta39K/Quantum38K carry chain and the benefits it provides.

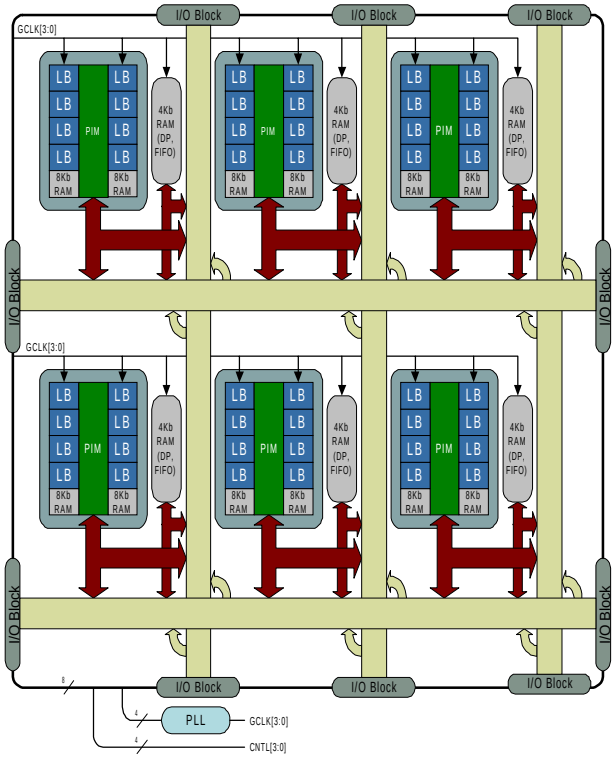

Figure 1. Delta39K50 Block Diagram

Adder Circuits in CPLDs

A full adder circuit provides the necessary logic to perform a one-bit addition. A full adder produces two outputs (sum and carry-out) from three inputs (two addends and carry-in). The logic for the outputs of a full-adder is given by the following equations:

$$Sum = A \oplus B \oplus C_{IN}$$

$$C_{OUT} = A \cdot B + A \cdot C_{IN} + B \cdot C_{IN}$$

The full adder is the basic building block of the ripple adder. In a ripple adder the carry-out of one full adder is the carry-in to the next full adder. To create a n-bit ripple adder, *n* full adder circuits need to be cascaded together. In a traditional product term based CPLD like the Ultra37000, a ripple adder is the most efficient adder implementation in terms of resource utilization. An n-bit ripple adder requires *7*n* product terms and *2*n* macrocells to implement. Unfortunately, the ripple adder is also the slowest implementation available due to the fact that propagating the carry between each full adder stage requires an additional pass through the Programmable Interconnect Matrix (PIM). For example, a sixteen-bit ripple adder uses 112 product terms and 32 macrocells, and has a maximum t_{PD} = 98.5 ns in a CY37064P100-125AC.

To increase adder performance Cypress's *Warp*™ software implements a carry-lookahead algorithm for adders in Ultra37000 devices. For more information concerning the carry-lookahead algorithm used by *Warp*, please refer to the application note "Efficient Arithmetic Designs With Cypress CPLDs." Although more resources are used by a carry-lookahead adder, the performance is much improved. The same sixteen-bit adder implemented with a carry-lookahead algorithm uses 180 product terms and 35 macrocells, and has a maximum t_{PD} = 22.0 ns in a CY37064P100-125AC.

Adder Circuits in Delta39K and Quantum38K

The carry chain allows for the implementation of fast ripple adders by quickly generating and propagating the carry-in for each stage. The performance of a Delta39K or Quantum38K ripple adder is comparable to, and often faster than, carry-look-ahead adders implemented in traditional CPLDs. In addition, the carry chain logic and the exclusive-or (XOR) gate available in each macrocell make Delta39K and Quantum38K adders more resource-efficient than ripple and carry-lookahead adders in a traditional CPLD.

The carry-in for any given adder stage (n+1) is given by the carry-out of the previous stage (n). By logical manipulation, the logic used by the carry chain can be realized:

$$INn+1 = C_{OUTn} = A_n \cdot B_n + A_n \cdot C_{INn} + B_n \cdot C_{INn}$$

$$INn+1 = (C_{INn} + \overline{C_{INn}}) \cdot A_n \cdot B_n + A_n \cdot C_{INn} + B_n \cdot C_{INn}$$

$$INn+1 = C_{INn} \cdot A_n \cdot B_n + \overline{C_{INn}} \cdot A_n \cdot B_n + A_n \cdot C_{INn} + B_n \cdot C_{INn}$$

$$C_{INn+1} = \overline{C_{INn}} \cdot (A_n \cdot B_n) + C_{INn} \cdot (A_n + B_n)$$

$$C_{INn+1} = \overline{C_{INn}} \cdot (A_n \cdot B_n) + C_{INn} \cdot \overline{(\overline{A_n} + \overline{B_n})}$$

$$C_{INn+1} = \overline{C_{INn}} \cdot (A_n \cdot B_n) + C_{INn} \cdot \overline{(\overline{A_n} \cdot \overline{B_n})}$$

The equations show that there are two cases that will produce a carry-in to the current adder stage. The carry-in to the current stage is driven by the carry-out from the previous stage and therefore depends on the inputs to the previous stage. First, if the carry-in to the previous stage is 0, then both A and B must be 1 to generate the carry-out. The logic for generating a carry-out from stage *n* is $A_n \cdot B_n$. On the other hand, if the carry-in to the previous stage is 1, then either A or B must be 1 to propagate the carry-in to the carry-out. The logic for propagating a carry-in to the carry-out is: $A_n + B_n$ or $\overline{A_n} \cdot \overline{B_n}$.

As shown in *Figure 2*, the Delta39K/Quantum38K carry chain uses a 2x1 multiplexer to determine whether the generate or propagate product term is needed to produce a carry-in to the current adder stage. The generate (CPT0) and propagate (CPT1) product terms each use a product term from the Product Term Array. If these two product terms are used by the carry chain, then they are unavailable for general-purpose logic. The carry-in from the previous adder stage is connected to SELIN to select between CPT0 and CPT1.

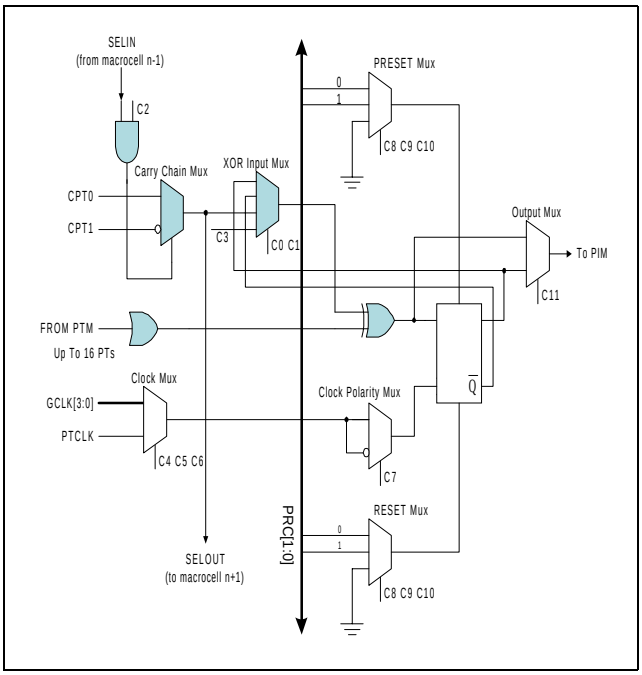

Figure 2. Macrocell Structure

The sum output for a given adder stage is given by:

$$um_n = A_n \oplus B_n \oplus C_{INn}$$

The XOR of A and B is accomplished by expanding the logic to a sum of products (SOP) expression that is implemented by the Product Term Allocator:

$$A_n \oplus B_n = A_n \cdot \overline{B_n} + \overline{A_n} \cdot B_n$$

The output of the SOP is then connected to one input of the macrocell's XOR gate. The other input to the XOR gate is C_{INn} which comes from the output of the carry chain multiplexor. C_{INn} is also propagated through SELOUT to the adder stage to determine the carry-in of the next stage, C_{INn+1}. This logic is then repeated for each stage of the entire adder.

Figure 3 contains a block diagram that illustrates how individual adder stages are cascaded together to form larger

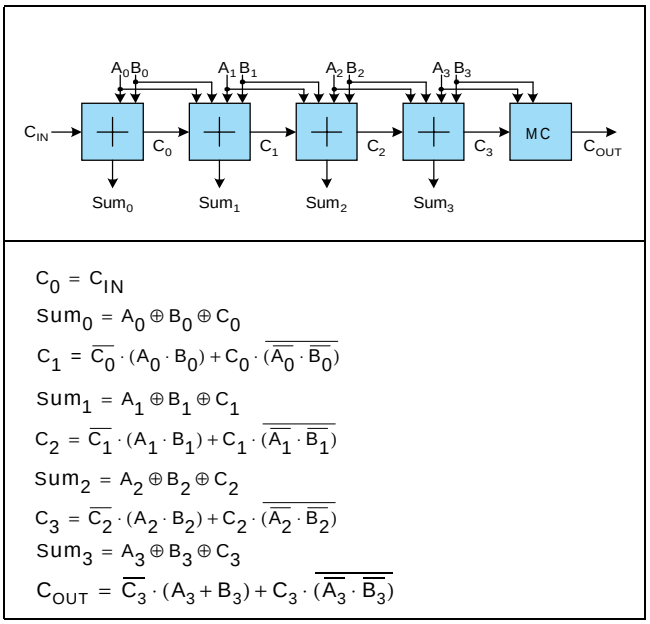

Figure 3. Block Diagram of a Four-bit Adder

adders. The equations for the carry logic and sum outputs are also specified in this figure. *Figure 4* shows the implementation of the same four bit adder (without the carry-out) in a Delta39K or Quantum38K logic block. An n-bit adder requires *(4*n) minus 1* product terms and *n* macrocells to implement in Delta39K or Quantum38K. In addition, if it is desired to extract the carry-out (or overflow) of the add operation, two product terms and an additional macrocell are required. It is worth noting that all stages of the adder require four product terms except the first stage which only requires three.

Due to the carry chain, adders implemented in Delta39K and Quantum38K use fewer device resources and can generally operate at higher speed than adders in traditional CPLDs. A sixteen-bit adder in a CY39100Z208-125NC uses 63 product terms and 16 macrocells, and has a maximum t_{PD} = 15.6 ns.

Carry Chain Architecture

Each cluster contains eight logic blocks, four on each side of the cluster PIM. There are sixteen macrocells in each logic block. The carry chain structure connects the four logic blocks on each side together, as shown in *Figure 5*. All of the macrocells in a logic block are also connected in series by the carry chain. Therefore, there are 64 macrocells cascaded together by each carry chain. The SELIN input to each macrocell is driven by the C_{IN} of the previous adder stage. The SELOUT output from each macrocell is the C_{IN} of the current adder stage. In general, SELOUT drives the SELIN of the next macrocell in the chain. The SELIN of the first macrocell in the carry chain is connected to V_{CC}. The SELOUT of the last macrocell is unconnected. The propagation delay from SELIN to SELOUT is approximately 0.2 ns, but varies with the speed grade of the device.

CPT0 and CPT1 are directly connected to the carry chain mux from the Product Term Array. However, both are inputs to the Product Term Allocator and can be used for general purpose logic when not needed for carry chain logic. Applications utilizing the carry chain can start and end anywhere in the chain. If a chain of greater than 64 macrocells is required, the

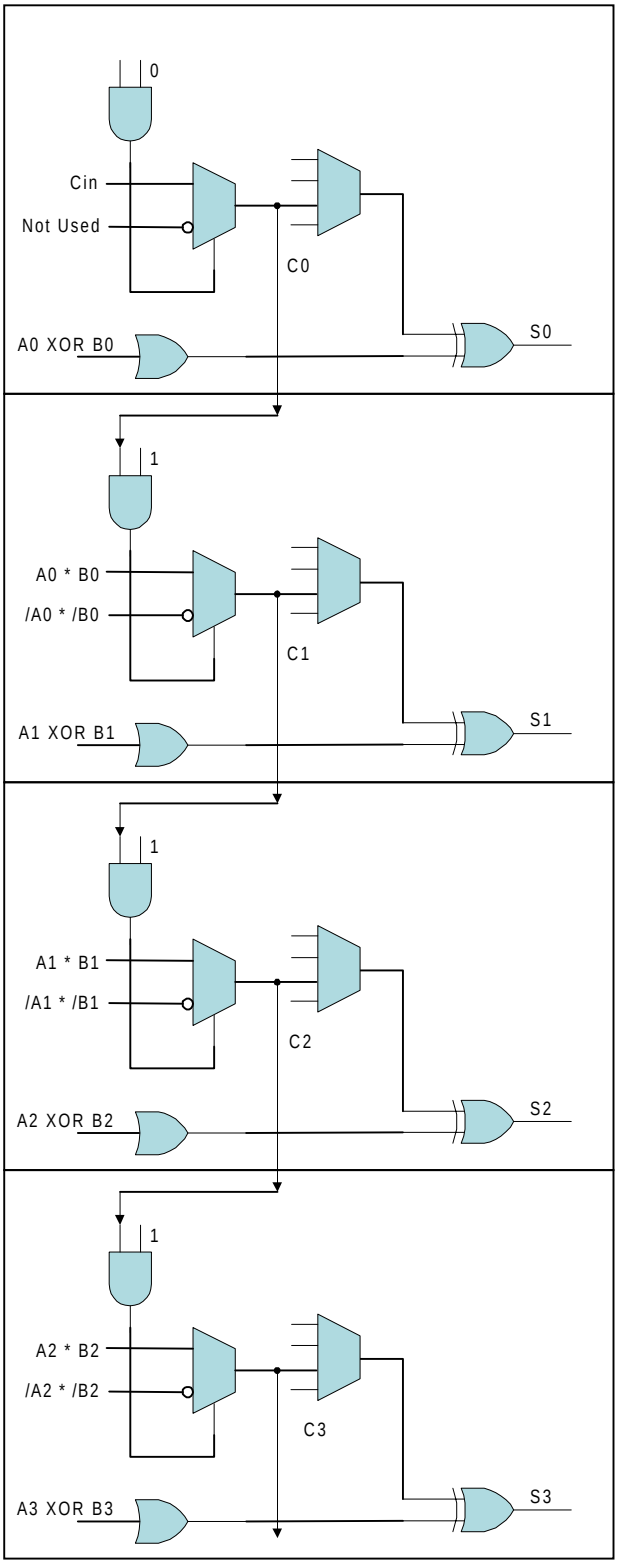

Figure 4. Four-bit Adder Utilizing Carry Chain

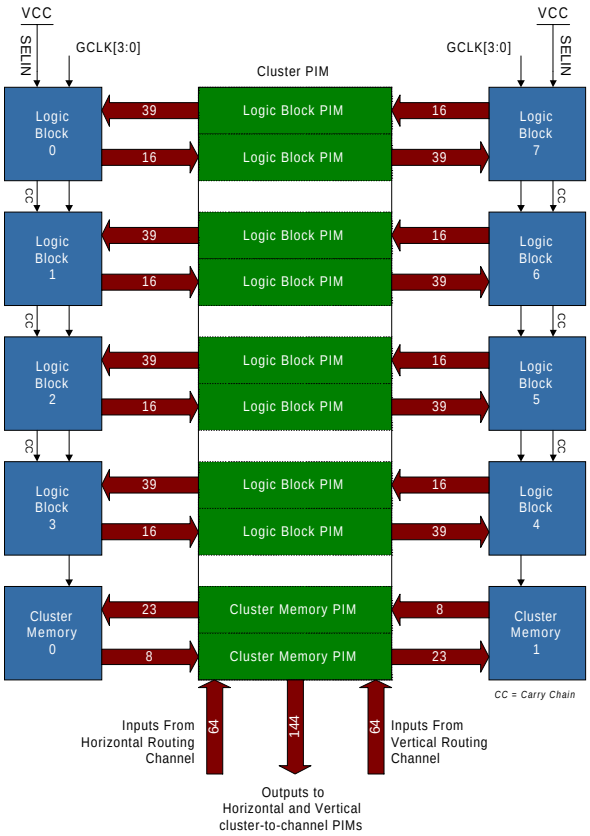

Figure 5. Logic Block Cluster Diagram

64th macrocell in the first chain implements the SELOUT logic, which is passed through the cluster PIM to the first macrocell in the second chain.

Utilization and Performance Comparison

Adders of various widths were targeted to Ultra37000 and Delta39K devices to provide a complete comparison of resource utilization and adder performance in the two devices. In each case, devices from the 125-MHz speed bin were targeted. The VHDL code for the adder used in the test is shown in *Figure 6*. Utilization and performance characteristics for the CY37512 and CY39100 are given in *Table 1* and *Table 2*, respectively.

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;

entity adder is
    generic(width: integer:= 16);
    port(
    A, B: in std_logic_vector(width-1 downto 0);
    Sum: out std_logic_vector(width-1 downto 0));
end adder;

architecture behv of adder is
begin
   Sum <= A + B;
end behv;
```

Figure 6. VHDL code for a Simple Adder

Table 1. CY37512P352-125BGC Adder Implementations

Adder Width	Macrocells	Unique PTs	t_{PD} (ns)
2-bit	2	8	10.0
4-bit	5	27	16.0
8-bit	15	74	22.0
16-bit	35	180	22.0
32-bit	75	440	22.0
64-bit	189	1166	28.0

Table 2. CY39100Z676-125MBC Adder Implementations

Adder Width	Macrocells	Unique PTs	t_{PD}(ns)
2-bit	2	7	10.2
4-bit	4	15	10.6
8-bit	8	31	11.6
16-bit	16	63	13.2
32-bit	32	127	19.2
64-bit	64	255	30.5

Subtracters and Comparators

Subtracters and comparators are two other common applications that can take advantage of the carry chain. In a subtraction operation the difference is calculated by subtracting the subtrahend (S) from the minuend (M). In addition to the difference, each stage of a subtracter also produces a borrow-out output. The logic equations for each stage of the subtracter are given by the following equations:

$$Diff = M \oplus S \oplus B_{IN}$$

$$B_{OUT} = \overline{M} \cdot S + \overline{M} \cdot B_{IN} + S \cdot B_{IN}$$

As the equations above suggest, the operation of a subtracter is similar to the operation of an adder. In Delta39K and Quantum38K, individual full subtracter stages are linked by the carry chain. The XOR of M and S is implemented by an SOP in the PTM and connects to one input of the macrocell XOR gate. The borrow-in is produced by the carry chain logic and is connected to the other input of the macrocell XOR gate. Logical manipulation reveals the logic used by Delta39K and Quantum38K to produce the borrow-in for each stage:

$$_{INn+1} = B_{OUTn} = \overline{M_n} \cdot S_n + \overline{M_n} \cdot B_{INn} + S_n \cdot B_{INn}$$

$$_{INn+1} = (B_{INn} + \overline{B_{INn}}) \cdot \overline{M_n} \cdot S_n + M_n \cdot B_{INn} + S_n \cdot B_{INn}$$

$$_{INn+1} = B_{INn} \cdot \overline{M_n} \cdot S_n + \overline{B_{INn}} \cdot \overline{M_N} \cdot S_n + \overline{M_n} \cdot B_{INn} + S_n \cdot B_{INn}$$

$$B_{INn+1} = \overline{B_{INn}} \cdot (\overline{M_n} \cdot S_n) + B_{INn} \cdot (\overline{M_n} + S_n)$$

$$B_{INn+1} = \overline{B_{INn}} \cdot \langle \overline{M_n} \cdot S_n \rangle + B_{INn} \cdot (\overline{\overline{M_n} + S_n})$$

$$_{INn+1} = \overline{B_{INn}} \cdot (\overline{M_n} \cdot S_n) + B_{INn} \cdot \overline{(M_n \cdot \overline{S_n})}$$

The borrow-out of the final stage of a subtracter can be used to compare the magnitude of the two inputs to the subtraction. If the borrow-out is 0, then the minuend was greater than or equal to the subtrahend. If the borrow-out is 1, then the minuend was less than the subtrahend. *Table 3* shows the comparisons that can be made using the carry chain. The carry chain is not utilized for straight equality or non-equality comparisons.

Table 3. Comparisons Using the Carry Chain

Operation	Borrow-Out	Comparison
A - B	0	A >= B
A - B	1	A < B
B - A	0	A <= B
B - A	1	A > B

Using the Carry Chain in Warp

Warp Release 6.0 and later supports Delta39K and Quantum38K designs including those that use the carry chain. No special action is required to make use of the carry chain. *Warp*'s UltraGen™ synthesis automatically uses the carry chain for those operations that will benefit from it.

On occasion, the user may identify an application for the carry chain where *Warp* does not make use of it by default. In this circumstance the user can instantiate stages of the carry chain through the use of two primitive components. The necessary components are called *cy_c39kcarry* and *cy_c39kxor*, and are found in the *rtlpkg* package in the *work* library. *Cy_c39kcarry* is the primitive for the carry chain multiplexer and *cy_c39kxor* is the two-input XOR gate in the macrocell. *Figure 7* shows a VHDL example of how *cy_c39kcarry* and *cy_c39kxor* could be used to create the logic of a full adder. An equivalent Verilog example is shown in *Figure 8*.

In order to use the carry chain primitive components efficiently the user should keep several guidelines in mind. First, the

Cpt0 and *Cpt1* inputs to the *cy_c39kcarry* primitive should be connected to logic consisting of a single product term. Second, *Selin* can only be connected to '0', '1', or the *Selout* of the previous stage in the carry chain. Finally, the *Sin* input to the *cy_c39kxor* primitive should only be connected to the *Selout* of the corresponding *cy_c39kcarry* primitive. Violation of guidelines may result in either inefficient carry chain implementations or compiler error.

Conclusion

The carry chain is an exciting new feature added to the Delta39K and Quantum38K architectures. Cypress's *Warp* software uses the carry chain to implement arithmetic functions such as adders, subtracters, and comparators. The resulting logic uses far fewer resources and often operates at a faster speed than the equivalent implementations in traditional CPLDs. Due to the inclusion of the carry chain, Delta39K and Quantum38K are exceptional architectures for implementing arithmetic designs.

```
library ieee;
use ieee.std_logic_1164.all;
use work.rtlpkg.all;

entity full_adder is port(
    Cin, A, B: std_logic;
    Sum, Cout: out std_logic);
end full_adder;

architecture behv of full_adder is
    signal C0, C1, CCPT1, CCPT2, XORPT: std_logic;
begin
    CCPT1 <= A and B;
    CCPT2 <= (not A) and (not B);
    XORPT <= A xor B;

    cc1: cy_c39kcarry port map(
        SelIn => '0',
        Cpt0 =>Cin,
        Cpt1 =>'0',
        SelOut => C0);

    xx1: cy_c39kxor port map(
        SIn => C0,
        PtmPts => XORPT,
        XorOut => Sum);

    cc2: cy_c39kcarry port map(
        SelIn => C0,
        Cpt0 =>CCPT1,
        Cpt1 =>CCPT2,
        SelOut => C1);

    xx2: cy_c39kxor port map(
        SIn => C1,
        PtmPts => '0',
        XorOut => Cout);
end behv;
```

Figure 7. Using Carry Chain Primitives in VHDL

```
module full_adder (A, B, Cin, Sum, Cout);

    input A, B, Cin;
    output Sum, Cout;
    wire C0, C1, CCPT1, CCPT2, XORPT, ground;

    assign CCPT1 = A & B;
    assign CCPT2 = (~A) & (~B);
    assign XORPT = A ^ B;
    assign ground = 0;

    cy_c39kcarry cc1(
        .selin(ground),
        .cpt0(Cin),
        .cpt1(ground),
        .selout(C0));

    cy_c39kxor xx1(
        .sin(C0),
        .ptmpts(XORPT),
        .xorout(Sum));

    cy_c39kcarry cc2(
        .selin(C0),
        .cpt0(CCPT1),
        .cpt1(CCPT2),
        .selout(C1));

    cy_c39kxor xx2(
        .sin(C1),
        .ptmpts(ground),
        .xorout(Cout));
endmodule
```

Figure 8. Using Carry Chain Primitives in Verilog