Introduction to Delta39K's Carry Chain

Introduction

Cypress's fourth generation of Complex Programmable Logic Devices (CPLD), Delta39K, architecture is based on its predecessor, Ultra37000. It is built on clusters of Ultra37128 (*Figure 1*). While there are many improvements and a handful of new features, one important addition to the architecture is the carry chain structure. It changes the way the synthesizer handles arithmetic functions among other applications. It reduces resource utilization and improves operating speed.

The main reason to add this carry chain feature was to improve the handling of a common and very important application, Adder. However, there are other advantages that come with it.

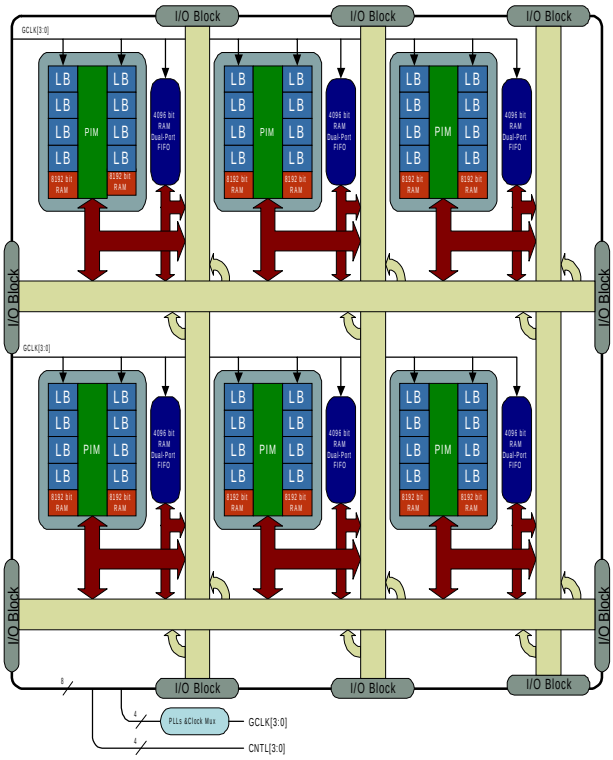


Figure 1. Delta39K50 Block Diagram

The Carry Chain Structure

As in Ultra37128, there are 4 logic blocks on both the left and the right side of the Programmable Interconnect Matrix (PIM). Each logic block consists of 16 macrocells. The carry chain structure connects the 4 logic blocks on each side together as shown in *Figure 2*. The Selin of the first logic block of each chain is connected to V_{CC}. The Selout of each logic block

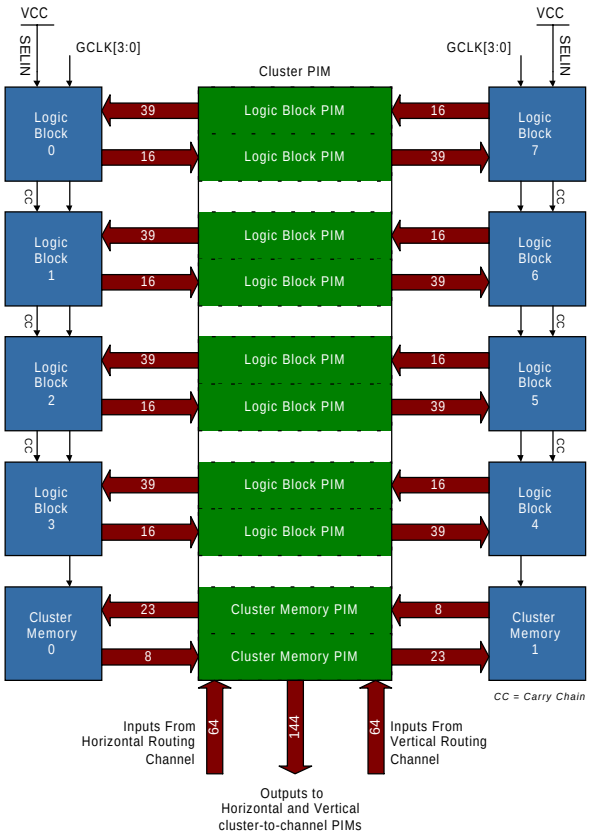


Figure 2. Logic Block Cluster Diagram

connects directly to the Carry In of the next logic block. The Carry Out of the last logic block is not connected to anything.

All of the macrocells in a logic block are also connected in series by the carry chain. There is around 0.2 ns delay associated with each carry chain connection between macrocells. *Figure 3* shows the architecture of the macrocell. The carry chain structure consists of two product terms (CPT0 and CPT1), Carry Chain Mux, Carry In, and Carry Out.

CPT0 and CPT1 come directly from the Product Term Array (PTA), not the Product Term Matrix (PTM). However, both are inputs to the PTM, so they can be used by the macrocell to form other logic equations. Applications utilizing carry chain can start and end anywhere in the chain. When more than 64 macrocells required in a chain, the carry out of the 64th macrocell can be routed to the Logic Block PIM as an input to the first macrocell of the next chain. However, this means that the 64th macrocell can not be used to generate other logic functions.

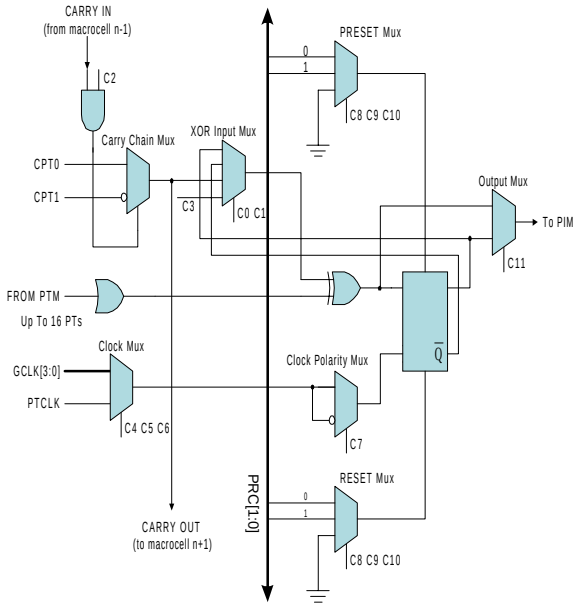


Figure 3. Macrocell Structure

Full Adder

The default implementation of the add function in Ultra37000 is the Carry-look-ahead adder, which is a fast adder. UltraGen has the capability to generate the best implementation that the architecture can do. To learn more about writing an efficient adder function in Ultra37000, please refer to the application note, "Efficient Arithmetic Designs With Cypress CPLDs." Below is an example of a very simple 'Add' function:

```
ENTITY two_bit_adder IS PORT(
      in1, in2: IN std_logic_vector(1 DOWNTO 0);
      sum: OUT std_logic_vector(1 DOWNTO 0)
   );
   END ENTITY;

   ARCHITECTURE add OF two_bit_adder IS
   BEGIN
     sum <= in1 + in2;
   END add;
```

This 2-bit adder code will generate these equations for Ultra37000:

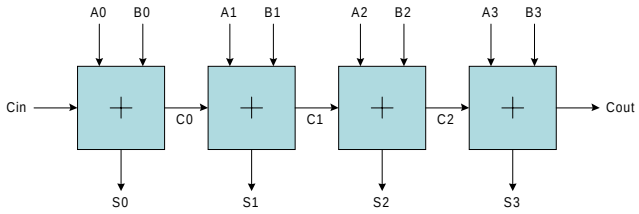
```
    sum(1) =
        /in1(1) * in1(0) * /in2(1) * in2(0)
        + in1(1) * in1(0) * in2(1) * in2(0)
        + in1(1) * /in2(1) * /in2(0)
        + /in1(1) * in2(1) * /in2(0)
        + in1(1) * /in1(0) * /in2(1)
        + /in1(1) * /in1(0) * in2(1)
    sum(0) =
        in1(0) * /in2(0)
        + /in1(0) * in2(0)
```

If we increase the size of this adder to 4 bit, here are the other equations:

```
    sum(3) =
        /in2(2) * /MODULE_1.CMB * in1(3) * /in2(3)
        + /in2(2) * /MODULE_1.CMB * /in1(3) * in2(3)
        + /in1(2) * /MODULE_1.CMB * in1(3) * /in2(3)
        + /in1(2) * /MODULE_1.CMB* /in1(3) * in2(3)
        + in2(2) * MODULE_1.CMB * /in1(3) * /in2(3)
        + in2(2) * MODULE_1.CMB * in1(3) * in2(3)
        + in1(2) * MODULE_1.CMB * /in1(3) * /in2(3)
        + in1(2) * MODULE_1.CMB * in1(3) * in2(3)
        + /in1(2) * /in2(2) * in1(3) * /in2(3)
        + /in1(2) * /in2(2) * /in1(3) * in2(3)
        + in1(2) * in2(2) * /in1(3) * /in2(3)
        + in1(2) * in2(2) * in1(3) * in2(3)
    sum(2) =
        in1(2) * /in2(2) * /MODULE_1.CMB
        + /in1(2) * in2(2) * /MODULE_1.CMB
        + /in1(2) * /in2(2) * MODULE_1.CMB
        + in1(2) * in2(2) * MODULE_1.CMB
    MODULE_1 =
        in1(0) * in2(1) * in2(0)
        + in1(1) * in1(0) * in2(0)
        + in1(1) * in2(1)
```

As the example shows, a 4-bit adder requires more than double the resources of the 2-bit adder. MODULE_1 signal was internally generated by the synthesis to minimize the number of product terms used since the maximum product terms available for each macrocell is 16. MODULE_1 takes an extra pass through the PIM to generate, which adds a timing delay to the outputs.

With the carry chain feature of Delta39K, UltraGen will generate a ripple-carry adder. Each bit of the adder requires the same number of resources (4 product terms in 1 macrocell), except the first bit, which requires one less product term. *Figure 4* is an example of a 4-bit full-adder; and *Figure 6* shows the implementation in Delta39K utilizing the carry chain structure. Cout signal may or may not be used for other logics.



S0 = Cin XOR A0 XOR B0
C0 = Cin

S1 = C0 XOR A1 XOR B1
C1 = (/Cin * A0 * B0) + (Cin * /A0 * /B0)

S2 = C1 XOR A2 XOR B2
C2 = (/C1 * A1 * B1) + (Cin * /A1 * /B1)

S3 = C2 XOR A3 XOR B3
Cout = (/Cin * A2 * B2) + (Cin * /A2 * /B2)

Figure 4. Four-bit Full-Adder

Tests and calculations were done to compare the resource utilization on Ultra37000 and Delta39K in implementing n-bit adders. Devices used in the comparison were Ultra37256-154 and Delta39K100-250 (preliminary data). The results are shown in *Table 1* and *Table 2*.

Table 1. Ultra37256-154 N-bit Adder Implementation

N-bit Adder	Macrocell	Unique PTs	t_{PD} (ns)
2-bit	2	8	7.5
4-bit	5	27	12.0
8-bit	15	74	16.5
16-bit	35	180	16.5
32-bit	75	440	16.5
64-bit	156	928	25.5

Table 2. Delta39K100-125 N-bit Adder Implementation

N-bit Adder	Macrocell	Unique PTs	t_{PD}(ns)
2-bit	2	7	7.7
4-bit	4	15	8.1
8-bit	8	31	8.9
16-bit	16	63	10.5
32-bit	32	127	13.7
64-bit	64	255	20.1

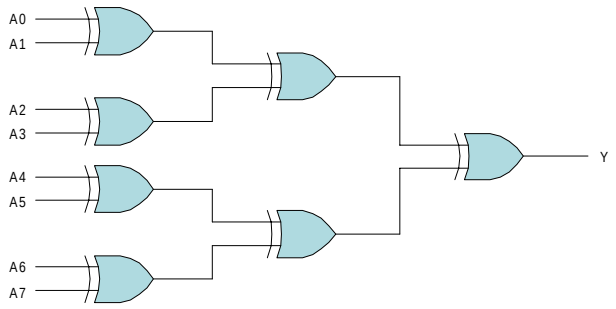
Other Applications

Other logic functions can be implemented easily using carry chain, such as a subtractor. To implement a subtractor, the synthesizer will invert the inputs of the value to be subtracted and set the initial carry in to 1 (2's complement), then implement a true subtractor cell that creates the difference and the borrow in.

This subtractor can also be used to implement a comparator. For example, if A–B is implemented then the borrow out can be used to determine if A>=B (borrow out = 0) or if A<B (borrow out = 1).

Another example of application that can take advantage of the carry chain is the parity generator shown in *Figure 5* (Block diagram and equations) and *Figure 7* (Implementation in Delta39K). Implementation of an N-bit parity generator with straight product term based equation will require 2^(N–1) product terms to implement. When N is 5, it requires 16 product terms. So, a single macrocell in Ultra37000 can generate one 5-bit parity generator. For N greater than 5, it will require more than 1 macrocell to implement, which will have PIM timing delay, in addition to requiring a large number of product terms.

In Delta39K, each carry chain can contribute one input to the parity generator (uses 2 product terms as inputs to MUX1) and the last macrocell can implement a 4-input XOR using 8 product terms and then uses the XOR gate for the last stage of the parity generator. In general, an N-bit parity generator can be implemented in N–4 macrocells. With this scheme, the macrocells, except the last one, can implement other logics that don't need the carry chain logic.



Y = A0 XOR A1 XOR A2 XOR A3 XOR A4 XOR A5 XOR A6 XOR A7

Figure 5. Eight-bit Parity Generator

UltraGen

What is needed to utilize this new feature? Not much. UltraGen, the synthesis machine of *Warp*, will be able to figure out what the design needs for best performance. UltraGen will detect characteristics of common functions, such as addition and subtraction, and implement them using UltraGen modules written to optimize the utilization of resources in Cypress's CPLDs.

Manual Primitive Modules Utilization

It is possible for the programmer to manually design applications utilizing the primitive modules that build the carry chain structure. It can be useful, but will require careful planning on the programmer's side. If you let UltraGen optimize the equation that you want to achieve, the result may be similar. And the advantage that you get may not worth the effort.

Conclusion

With an architecture similar to that of Ultra37000, Delta39K inherited, among other things, the excellent routability and simple timing Ultra37000 has. Additional features, such as the carry chain, make it even better. As the examples show, utilization of the carry chain logic reduces the amount of resources required in implementing common arithmetic applications. It is easy to use, flexible, and helps improve the performance of your design.

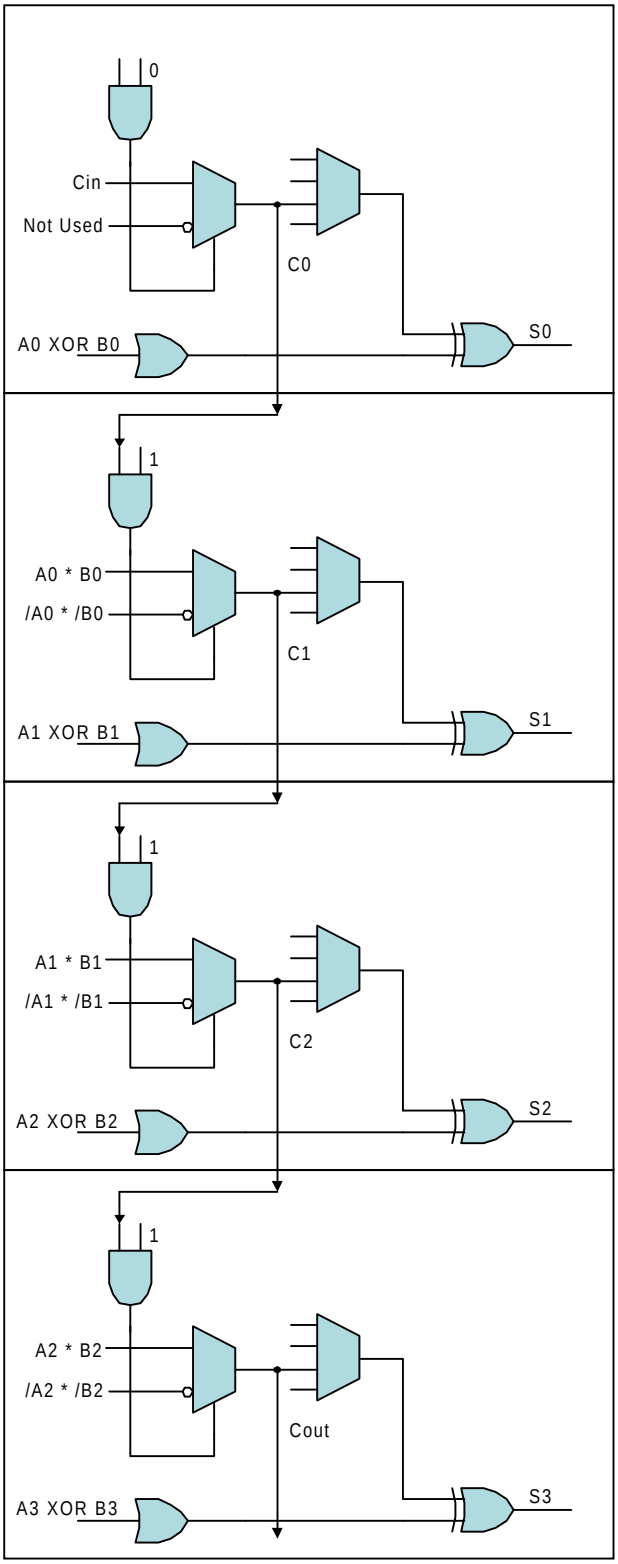


Figure 6. Four-bit Adder Utilizing Carry Chain

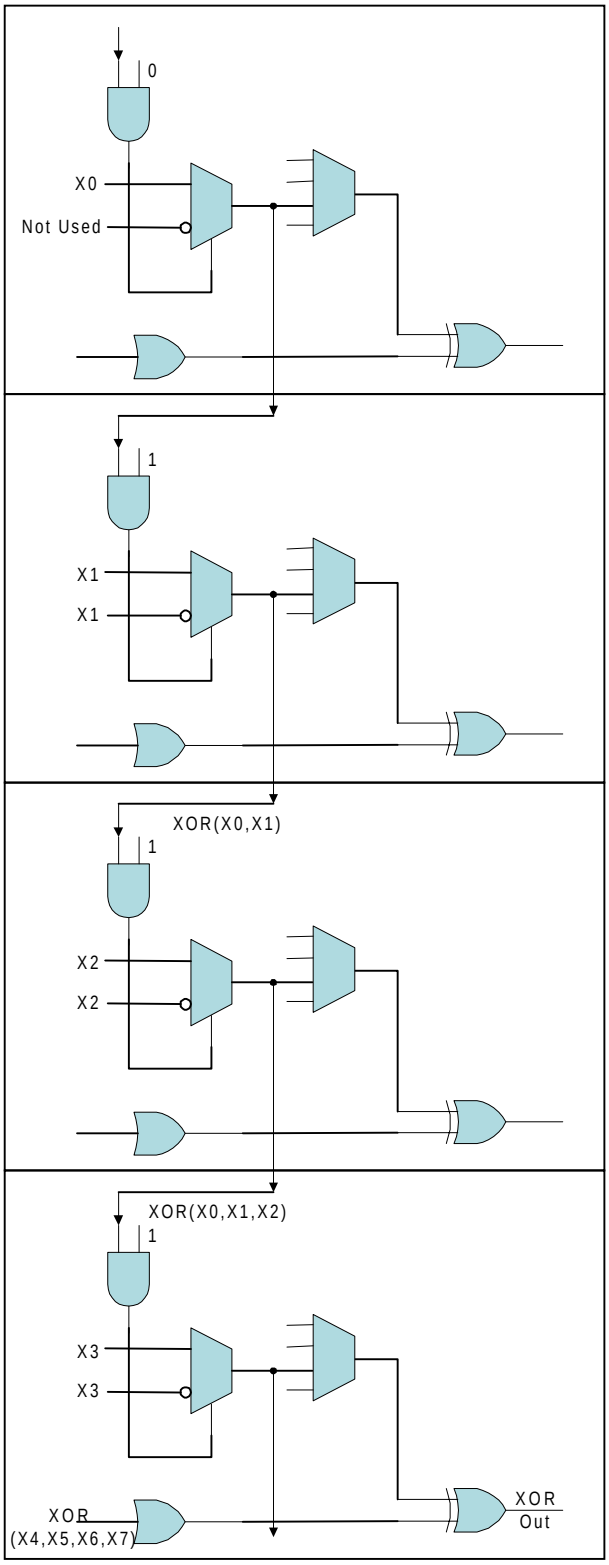


Figure 7. Eight-bit Parity Generator Utilizing Carry Chain