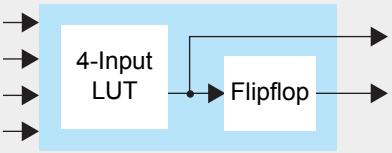
Why the XLR Cell is a Big Deal

Figure 1 Basic Logic Element



People talk about the logic element (LE), or logic cell, as being the basic building block of FPGAs. In reality, the LE is a fictional structure. If you look at the actual architecture diagrams for FPGAs, they vary significantly. Some have simple logic blocks. Some have clustered blocks with complex logic functions. Most have dedicated routing lines. Some have advanced blocks for arithmetic functions. All have some form of embedded RAM. But no two FPGA architectures are alike, making a true comparison challenging. However, by considering the logic capability of an FPGA in LEs, customers can compare FPGA product families and vendors more easily. At 易灵思, we define an LE as a 4-input look-up table (LUT) and a flipflop (Figure 1), which is a definition commonly used in the industry.

Traditional Architectures

Having dedicated logic and dedicated routing seems like a good idea on the surface. In reality it means that you are leaving something on the table, area-wise. Most complex logic blocks have only so many ways to enter and leave the block. Once you have used up all of the doorways, you cannot access any remaining logic in the block. And the dedicated routing may get signals across the chip quickly, but if there are only a few ways to get on and off the routing line, there is congestion that limits its usefulness. Think of a freeway with only a few on/off ramps. If everyone has to get on or off at the same place, you are going to be sitting in a traffic jam. But if there are a lot of entries on and off, the traffic spreads out and can run more smoothly. The XLR cell functions on that concept: because any XLR cell can be routing, software can choose the best paths to avoid congestion.

Congestion results from few on/off ramps

eXchangeable Logic and Routing Cell

The basic architecture for 易灵思 FPGAs, which we call the Quantum architecture (or fabric), comprises a regular grid structure of logic blocks plus RAM and multipliers/DSP Blocks. The logic block is called the eXchangeable Logic and Routing (XLR) cell. The XLR cell is exactly what the name implies: a block that can act as logic *and* as routing (Figure 2). So instead of dedicated routing lines and big complex blocks, 易灵思 FPGAs have a small, simple structure, the XLR cell, that does everything.

In our Trion family, the data sheet reports the logic capacity for a given FPGA in LEs. The XLR cell in the Trion family is essentially a 4-input LUT with a flipflop, but it also has built-in adder capability, which gives it the ability to perform additional logic functions. Although the XLR cell includes the extra adder functionality, 易灵思 considers that Trion FPGAs have a 1:1 mapping of LEs to XLR cells. Each XLR cell is equivalent to one LE. In the Efinity software, when you compile your design, you get a report of the logic usage in LEs so you can understand how much of the available logic capacity is used.

The 钛金系列 XLR Cell packs in 20% more logic capacity than a regular LE.

钛金系列 XLR Cells are Upgraded

The high-performance 钛金系列 family is a little different. In addition to giving the number of LEs for each family member, the data sheet also reports the number of XLR cells. For example, the Ti60 Data Sheet gives the number of LEs as 62,016 and the number of XLR cells as 60,800. You might wonder: why are those numbers different and what does that mean for the logic capacity?

As you probably guessed, the mapping of XLR cells to LEs is not 1:1. In the 钛金系列 family, we have upgraded the Quan-tum architecture to better support compute functions. One upgrade adds 8-bit shift register functionality to the XLR cell.

Adding this function gives the XLR cell the ability to do more than a regular 4-input LUT plus a flipflop. Additionally, the 4-input LUT is *fracturable*, which means that it can function as a 3-input LUT and a 2-input LUT at the same time, providing even more capabilities (Figure 3). These improvements have a strong effect on the amount of logic the XLR cell can

Figure 2 Grid of XLR Cells

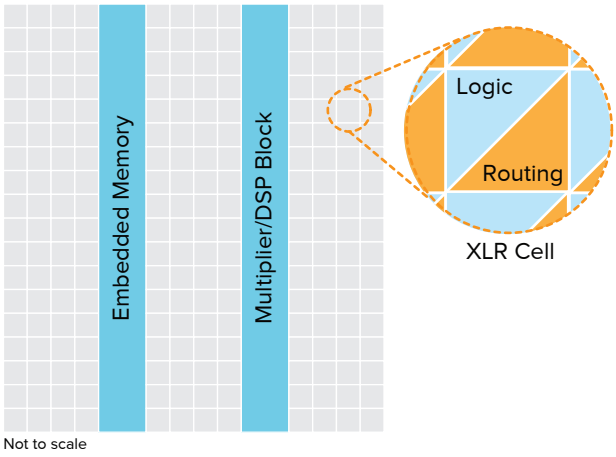
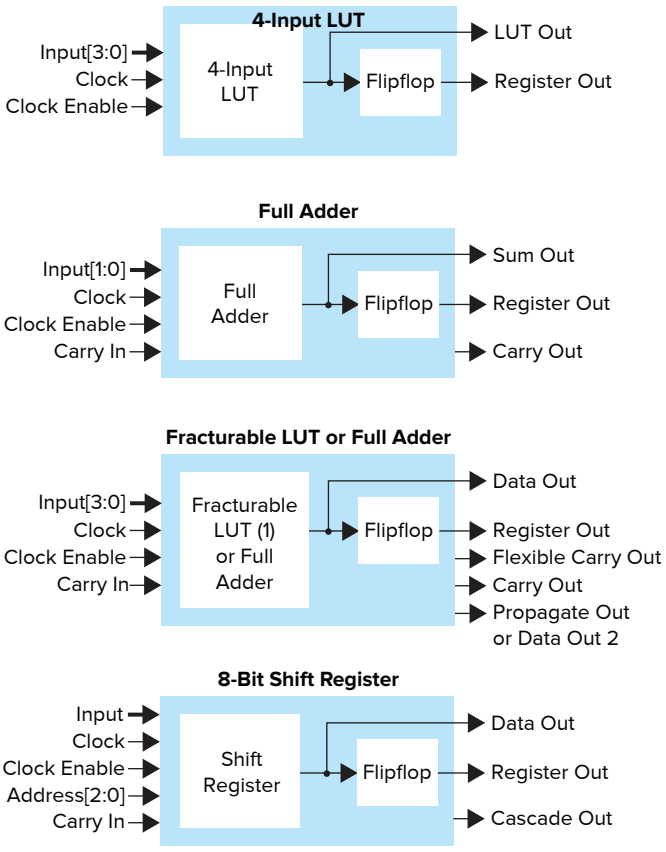


Figure 3 Logic Functions in 钛金系列 XLR Cell



1. The fracturable LUT is a combination of a 3-input LUT and a 2-input LUT. They share 2 of the same inputs.

implement. After thousands of experiments to quantify the effect of 钛金系列 XLR cell enhancements, 易灵思 has deter-mined that they provide the ability to pack 20% more logicthan a regular LE. So every XLR cell is worth 1.2 LEs. Usinga multiplier to equate a complex block to LEs is commonin the FPGA industry. As we mentioned earlier, the LE is afictional construct and most FPGA architectures are muchmore complex than that. One vendor uses ratios like 1.6:1 or2.18:1 to get their LE counts.[1] Another uses 2.94.[2]

Stating that the ratio of LEs to XLR cells is 1.2:1 does not explain why the reported number of XLR cells is *lower* than the reported number of LEs in the data sheet. Remember that the XLR cell is used for both logic and routing. 易灵思 reserves 15% of the XLR cells for routing; the percentage is based onthe results of extensive experimentation. The data sheet showsthe total number of XLR cells available for both logic androuting. Of those, 15% are reserved for routing, leaving 85%for logic. To do the math:

60,800 XLR cells x 0.85 x 1.2 = 62,016 LEs

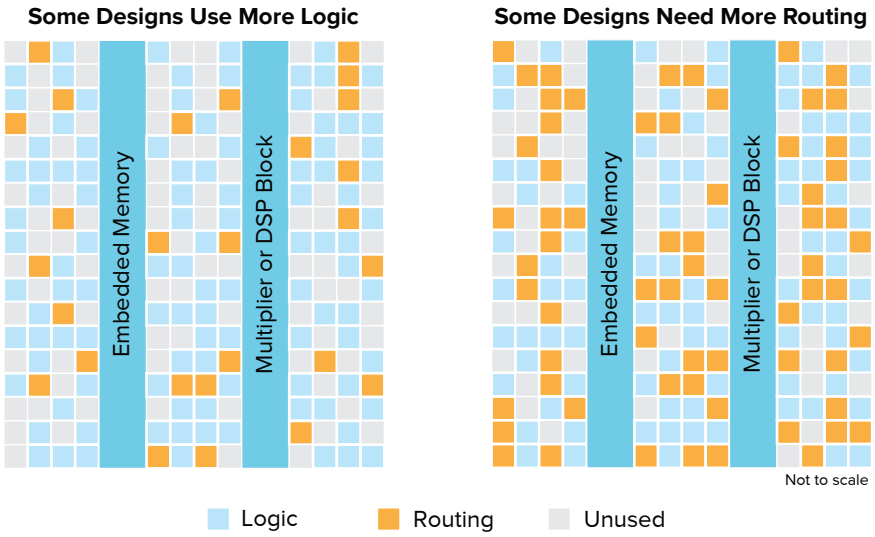
Dynamic Logic and Routing

Some designs need more routing, some need less, and that is where the XLR cell shows its quality. The Efinity software has heuristics that estimate the number of XLR cells your design may need for routing. If your design does not need all of the reserved routing XLR cells, the software can dynamically use them for more logic (Figure 4). Additionally, because the XLR cell supports both logic *and* routing, after the software places all of the logic it can use those same XLR cells for routing too. Consider a scenario where your design is progressing well, everything is routing fine, the FPGA is nearly packed full of logic. But you need to add just a bit more. With a traditional FPGA, when the logic is used up there is nowhere left to go. With the XLR cell, you can go that extra step, pack in that extra bit of logic, push it to eleven.[3] The bottom line: you can use *more* XLR cells, and therefore *more* LEs, than are actually shown in the data sheet. That point is actually so important that it bears repeating:

钛金系列 ***FPGAs can give you more logic than the data sheet says.***

If you can get more logic than the data sheet shows, then you might ask: why show a lower number, why not inflate the logic count? As we explained earlier, 易灵思 has performed extensive experimental testing that has guided the values given in the data sheet. Although many designs may be able to get away with less routing, others really need all of those reserved XLR cells. The data sheet gives a conservative estimate of the number of routing XLR cells required for average designs. Obviously, no one wants routing to "steal" XLR cells from the logic capacity.

Figure 4 Comparing Logic vs. Routing Usage



The XLR Cell in Action

When you read the Efinity placer report, the software reports the usage in XLR cells for logic and the estimate it needs for routing. Figure 5 is a report snippet that shows the resource usage for the 32-bit RISC processor (oc_m1_core) open-source design. This design is stamped 8 times and targets the Ti60 FPGA. Stamping designs is a common way for FPGA vendors to create large designs to fill up a device for benchmarking and experimentation.[4] The FPGA is packed pretty full with over 93% of XLR cells used and just over 9% of the available XLR cells are reserved for routing.

Analyzing the numbers in the report:

- The design uses 5,598 XLR cells for routing, which is about 60% of the 9,120 XLR cells that are reserved for routing (15% of 60,800).
- The design uses 51,194 XLR cells for logic, which is equivalent to 61,432.8 LEs.
- There are 4,008 XLR cells unused, which can be logic and routing. If we save 15% of the unused cells for routing, there are 3,406 (rounding down) left for logic. That number equates to 4,087 LEs left, which is about 3,500 LEs more than what the data sheet reports.

Figure 6 shows the floorplan for this design, which gives a visual representation of how the logic and routing are packed (blue cells are logic and orange are routing).

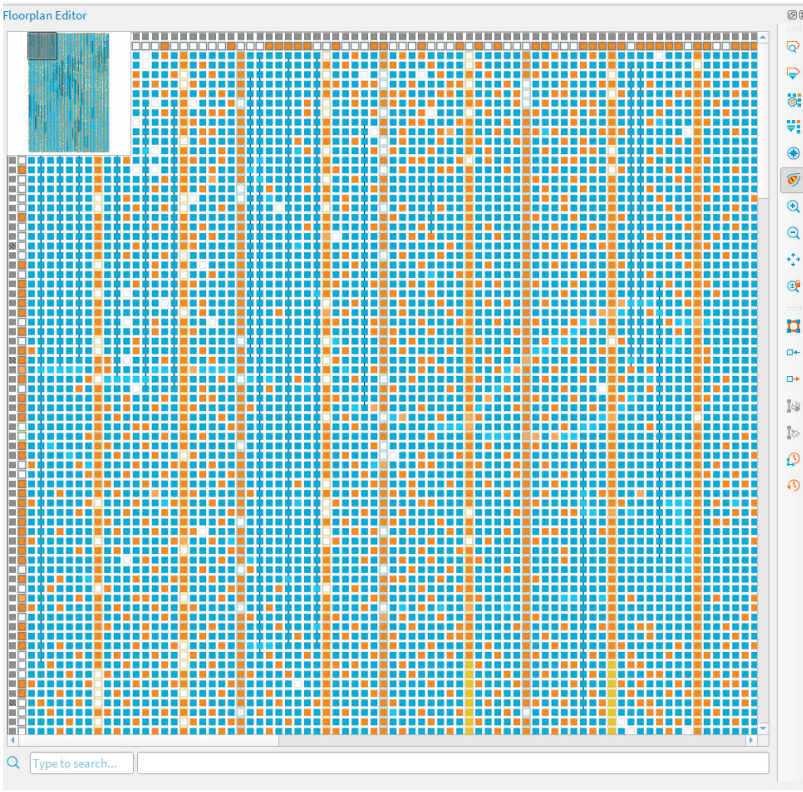
The XLR Cell is a Big Deal

The bottom line is, because the XLR cell can function as logic *and* routing, you can potentially use *all* of the available XLR cells for logic. With the XLR cell you get more. More logic, more efficiency, more room to innovate. And that is why the XLR cell is a big deal.

Figure 5 Efinity Placer Report Excerpt

```
---------- Resource Summary (begin) ----------
Inputs: 4 / 1411 (0.28%)
Outputs: 1 / 2252 (0.04%)
Global and Regional Buffers: 0 / 64 (0.00%)
XLRs: 56792 / 60800 (93.41%)
        XLRs needed for Logic: 28645 / 60800 (47.11%)
        XLRs needed for Logic + FF: 15580 / 60800 (25.62%)
        XLRs needed for Adder: 4568 / 60800 (7.51%)
        XLRs needed for Adder + FF: 312 / 60800 (0.51%)
        XLRs needed for FF: 2089 / 60800 (3.44%)
        XLRs needed for SRL8: 0 / 14720 (0.00%)
        XLRs needed for SRL8+FF: 0 / 14720 (0.00%)
        XLRs needed for Routing: 5598 / 60800 (9.21%)
Memory Blocks: 16 / 256 (6.25%)
DSP Blocks: 0 / 160 (0.00%)
---------- Resource Summary (end) ----------
```

Figure 6 32-Bit RISCV Processor Design Floorplan



References

[1] "System logic cells," Xilinx forum, https://forums.xilinx.com/t5/Versal-and-UltraScale/Sysem-logic-cells/m-p/959067#M9565, 2019.

[2] Intel Stratix 10 GX/SX Product Table, Intel. Gen-1023-2.0. Divide the number of LEs by the number of ALMs to get the ratio. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/pt/stratix-10-product-table.pdf

[3] Paraphrasing Nigel Tufnel's elucidative discourse on the importance of going to eleven, "This Is Spinal Tap," Embassy Pictures, 1984.

[4] Won, Martin and Monga, Madhu. Intel Arria 10 FPGA Performance Benchmarking Methodology and Results, Intel. WP-01271-1.0. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01271-intel-arria10-performance-benchmarking-methodology-and-results.pdf