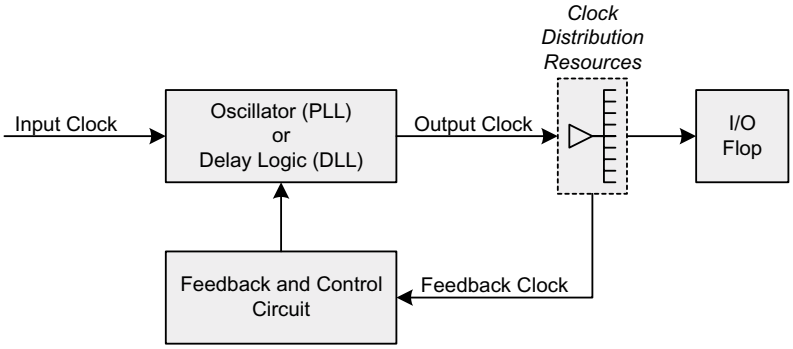


**FIGURE 11.8** PLL/DLL clock deskew function within FPGA.

clock so that the causal edge observed by the I/O flops occurs at nearly the same time as when it enters the FPGA's clock input pin. PLLs and DLLs are discussed in more detail in a later chapter.

Additional circuitry enables some PLLs and DLLs to emit a clock that is related to the input frequency by a programmable ratio. The ability to multiply and divide clocks is a benefit to some system designs. An external board-level interface may run at a slower frequency to make circuit implementation easier, but it may be desired to run the internal FPGA logic as a faster multiple of that clock for processing performance reasons. Depending on the exact implementation, multiplication or division can assist with this scheme.

RAM blocks embedded within the logic cell array are a critical feature for many applications. FIFOs and small buffers figure prominently in a variety of data processing architectures. Without on-chip RAM, valuable I/O resources and speed penalties would be given up to use off-chip memory devices. To suit a wide range of applications, RAMs need to be highly configurable and flexible. A typical FPGA's RAM block is based on a certain bit density and can be used in arbitrary width/depth configurations as shown in Fig. 11.9 using the example of a 4-kb RAM block. Single- and dual-port modes are also very important. Many applications, including FIFOs, benefit from a dual-ported

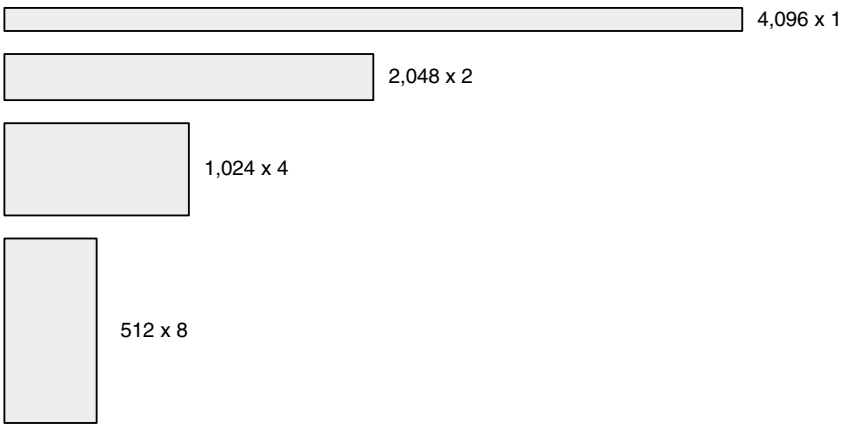


**FIGURE 11.9** Configurable FPGA 4 kb RAM block.

RAM block to enable simultaneous reading and writing of the memory by different logic blocks. One state machine may be writing data into a RAM, and another may be reading other data out at the same time. RAM blocks can have synchronous or asynchronous interfaces and may support one or two clocks in synchronous modes. Supporting two clocks in synchronous modes facilitates dual-clock FIFO designs for moving data between different clock domains.

Some FPGAs also allow logic cell LUTs to be used as general RAM in certain configurations. A $16 \times 1$ four-input LUT can serve as a $16 \times 1$ RAM if supported by the FPGA architecture. It is more efficient to use RAM blocks for large memory structures, because the hardware is optimized to provide a substantial quantity of memory in a small area of silicon. However, LUT-based RAM is beneficial when a design requires many shallow memory structures (e.g., a small FIFO) and all the large RAM blocks are already used. Along with control logic, 32 four-input LUTs can be used to construct a $16 \times 32$ FIFO. If a design is memory intensive, it could be wasteful to commit one or more large RAM blocks for such a small FIFO.

Embedding third-party logic cores is a feature that can be useful for some designs, and not useful at all for others. A disadvantage of FPGAs is their higher cost per gate than custom ASIC technology. The main reason that engineers are willing to pay this cost premium is for the ability to implement custom logic in a low-risk development process. Some applications involve a mix of custom and predesigned logic that can be purchased from a third party. Examples of this include buying a microprocessor design or a standard bus controller (e.g., PCI) and integrating it with custom logic on the same chip. Ordinarily, the cost per gate of the third-party logic would be the same as that of your custom logic. On top of that cost is the licensing fee charged by the third party. Some FPGA vendors have decided that there is sufficient demand for a few standard logic cores to offer specific FPGAs that embed these cores into the silicon in a fixed configuration. The benefit of doing so is to drop the per-gate cost of the core to nearly that of a custom ASIC, because the core is hard wired and requires none of the FPGA's configuration overhead.

FPGAs with embedded logic cores may cost more to offset the licensing costs of the cores, but the idea is that the overall cost to the customer will be reduced through the efficiency of the hard-wired core implementation. Microprocessors, PCI bus controllers, and high-speed serdes components are common examples of FPGA embedded cores. Some specific applications may be well suited to this concept.

I/O cell architecture can have a significant impact on the types of board-level interfaces that the FPGA can support. The issues revolve around two variables: synchronous functionality and voltage/current levels. FPGAs support generic I/O cells that can be configured for input-only, output-only, or bidirectional operation with associated tri-state buffer output enable capability. To achieve the best I/O timing, flops for all three functions—input, output, and output-enable—should be included within the I/O cell as shown in Fig. 11.10. The timing improvement obtained by locating these three flops in the I/O cells is substantial. The alternative would be to use logic cell flops and route paths from the logic cell array directly to the I/O pin circuitry, increasing the I/O delay times. Each of the three I/O functions is provided in both registered and unregistered options using multiplexers to provide complete flexibility in logic implementation.

More advanced bus interfaces run at double data rate speeds, requiring more advanced I/O cell structures to achieve the necessary timing specifications. Newer FPGAs are available with I/O cells that specifically support DDR interfaces by incorporating two sets of flops, one for each clock edge as shown in Fig. 11.11. When configured for DDR mode, each of the three I/O functions is driven by a pair of flops, and a multiplexer selects the appropriate flop output depending on the phase of the clock. A DDR interface runs externally to the FPGA on both edges of the clock with a certain width. Internally, the interface runs at double the external width on only one edge of the same clock frequency. Therefore, the I/O cell serves as a 2:1 multiplexer for outputs and a 1:2 demultiplexer for inputs when operating in DDR mode.