

# Memories are Made of This...

Virtex-5 FPGAs offer a wider range of memories and memory interfaces.

by Peter Alfke  
Distinguished Engineer  
Xilinx, Inc.  
peter.alfke@xilinx.com

All FPGA applications use various amounts of memory for data, parameters, and instructions. To store from a few bits to multiple megabytes, Xilinx® Virtex™-5 devices offer a hierarchy of three different memory implementations:

- LUT-based distributed RAM has a granularity of 64 bits
- Block RAM has a granularity of 18 Kb
- External memory can store practically unlimited amounts of megabytes with the help of on-chip memory interfaces

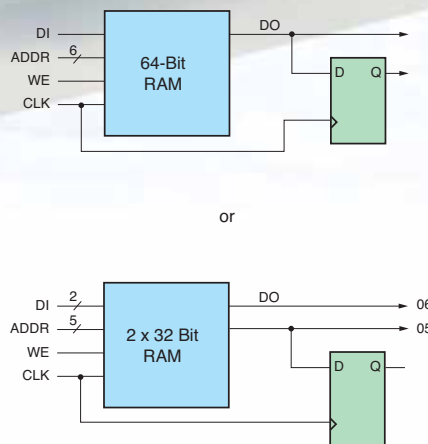


Figure 1 – LUT RAM

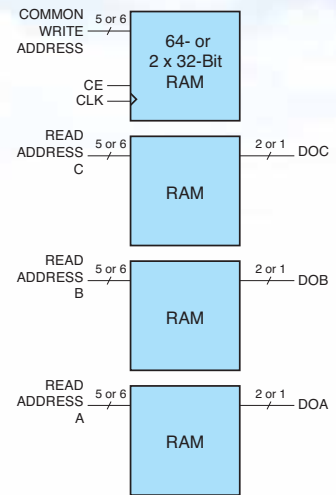


Figure 2 – Slice as quad-port RAM

## LUT RAM

Since the early days of the XC4000, Xilinx has made look-up tables (LUTs) available as user RAM. In Virtex-5 devices, the LUT has grown to 64 bits and can be used as either 64 x 1 or 32 x 2 RAM. LUT RAM (see Figure 1) offers very fast (sub-nanosecond) access time, tight integration with the logic fabric, and ultimate design flexibility (see sidebar, “Why 6-LUTs?”).

## Multiport Option

The four LUTs in a Slice M can share a common write address that does not interfere with the read addressing of the other three LUTs. Together, these four LUTs can thus implement quad-port memory, with one write port and three independent read

## Why 6-LUTs?

Xilinx invented the use of four-input LUTs in FPGAs 20 years ago. Exhaustive academic and commercial studies had shown that four inputs (16 stored bits) were the optimal size for a LUT that implements random logic.

FPGA evolution has led to ever-smaller transistors as well as ever-more routing and other dedicated structures. As a result, the highly optimized LUT became a much smaller part of the circuitry. For Virtex-5 devices, we re-evaluated the optimal LUT size and found that a four-times-larger six-input LUT (6-LUT) increases the CLB size by only 15%. Extensive benchmarking then showed that, on average, a 6-LUT packs 40% more logic functionality compared to the traditional four-input LUT. The decision was easy: spend 15% more area to gain 40% more logic (or, expressed differently, save roughly 30% in logic area).

The four-times-larger memory capacity of each LUT is an extra, very welcome bonus, as is the ability to make the LUT and the RAM 2 bits wide.

Virtex-5 devices combine four LUTs in a slice. There are two different types of slices: Slice L and Slice M, roughly equal in number on any Virtex-5 device. The LUTs in a Slice L can perform logic and contain a carry chain. The LUTs in a Slice M have the same functionality but can also be used as distributed memory or shift register logic (SRL32) functionality.

ports all accessing the same data. The newest MicroBlaze™ processor uses this feature to reduce its register file from 384 to 44 LUTs. In this kind of application, the new six-input LUT is six times more efficient than a previous-generation four-input LUT (see Figure 2).

**Shift Register**

You can use any LUT in a Slice M as a serial shift register with addressable length. The LUT is configurable as either a single-bit shift register (a maximum of 32 bits long) or as a 2-bit-wide shift register (a maximum of 16 bits long). Different from earlier SRL16 structures, the Virtex-5 shift register uses a more traditional and scalable design with two latches per shift register bit – hence the maximum 32 bits (not 64 bits) per LUT (Figure 3).

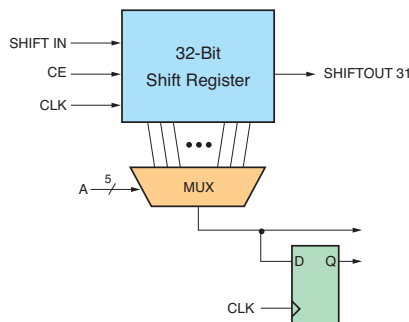


Figure 3 – LUT as shift register

**Block RAM**

For larger RAM structures, Virtex-5 devices have tens or hundreds of block RAMs, each with a capacity of as much as 36 Kb.

You can structure each block RAM through configuration as:

- 72 bits wide, 512 deep
- 36 bits wide, 1K deep
- 18 bits wide, 2K deep
- 9 bits wide, 4K deep
- 4 bits wide, 8K deep
- 2 bits wide, 16K deep
- 1 bit wide, 32K deep

You can also use the two halves of the 36-Kb block RAM separately as two 18-Kb

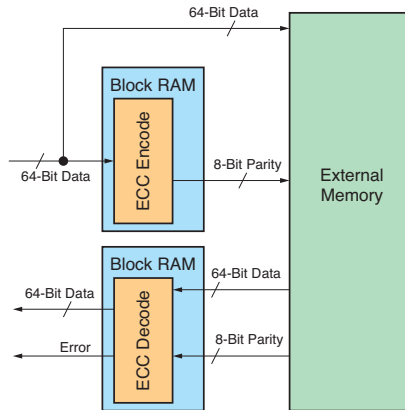


Figure 4 – ECC for external memory

block RAM, configured as:

- 36 bits wide, 512 deep
- 18 bits wide, 1K deep
- 9 bits wide, 2K deep
- 4 bits wide, 4K deep
- 2 bits wide, 8K deep
- 1 bit wide, 16K deep

Each block RAM always has two independent access ports and each port can be individually configured. This greatly simplifies data-width conversion.

**Read During Write**

Each port supports a data-in (DI) bus and a data-out (DO) bus. When writing the data on the DI bus into the memory, the DO bus presents either the previous data at the write address or the new data just being written. A third option keeps DO unchanged from its previous state. These three configuration options offer a design flexibility that is often overlooked.

All block RAM operations require a clock, even for reading data. This requirement is not always desirable, but it is absolute. Nothing happens without an enabled clock. Whenever the clock is enabled, data and address must meet the required setup and hold-time specification. Violating this requirement can contaminate the data content.

**ECC**

A 72-bit-wide block RAM can provide 64-bit-wide data with error detection and

correction (ECC) using Hamming code. The controller is built into each block RAM. It detects single and double errors and corrects all single errors.

The ECC controller can also be used to operate with external memory. In this case, one complete block RAM is necessary for writing and another for reading. The built-in ECC circuit is a great simplification for memory designers who care about the ultimate data integrity (Figure 4).

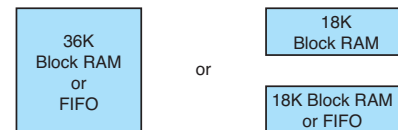


Figure 5 – Dual-ported RAM or FIFO

**FIFO**

FIFOs are usually implemented using dual-ported SRAMs, with one port used for writing and the other for reading. Many Virtex family block RAMs are traditionally used as FIFOs. That is why Xilinx chose to equip all Virtex-5 block RAMs with a built-in dedicated FIFO controller (Figure 5).

Virtex-5 devices have between 32 and 288 block RAMs, and each can be configured as a 36- or 18-Kb FIFO.

The controller can use the whole block RAM as FIFO with the following configuration options:

- 72 bits wide, 512 deep
- 36 bits wide, 1K deep
- 18 bits wide, 2K deep
- 9 bits wide, 4K deep
- 4 bits wide, 8K deep

But the controller can also use only half of the block RAM and leave the other half to be used as general-purpose block RAM. The FIFO options are then:

- 36 bits wide, 512 deep
- 18 bits wide, 1K deep
- 9 bits wide, 2K deep
- 4 bits wide, 4K deep

In all cases, the FIFO write and read ports have identical width. Unequal width would complicate the interpretation of full/empty flags and is therefore not implemented in Virtex-5 devices.

Soft FIFO controller cores have been available for many years, but the dedicated FIFO controller offers three advantages:

- Higher performance, since dedicated logic is naturally faster than programmable logic
- Smaller size and lower power consumption, as it uses no fabric resources, CLBs, nor additional interconnects
- Guaranteed functionality and performance without any design effort

Write and read clocks can have arbitrary or undefined phase and frequency relationships. But for proper flag operation, both clocks should be free-running.

The challenging aspect of FIFO design is the reliable generation of status flags (FULL, EMPTY, and ALMOST\_FULL or ALMOST\_EMPTY) when write and read clock frequencies are unrelated. The trailing edges of these flags are inevitably generated by the “wrong” clock domain and must be re-synchronized to the proper clock domain. For a detailed explanation, visit [www.sunburst-design.com/papers/CummingsSNUG2002SJ\\_FIFO2.pdf](http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO2.pdf). (See sidebar, “Verifying the EMPTY Flag Synchronization.”)

The FIFO controller offers two new options: first-word fall through (FWFT) and synchronous operations.

After a first entry has been written into an empty FIFO, the EMPTY output goes low (inactive), indicating that the read port is allowed to enable its read clock and thus cause the data word to appear at the output. This might be described as a “pull” operation. Data appears at the output after the next enabled read clock.

In FWFT, the newly written data word appears automatically at the output simultaneously with EMPTY going inactive. This might be called a “push” operation.

You can configure the FIFO for either of these modes. The difference is visible only at the read data output of the very first

## Verifying the EMPTY Flag Synchronization

We tested the EMPTY synchronization logic exhaustively by writing data into the FIFO at 200 MHz and reading it out at 500 MHz, which makes it go EMPTY soon after each write cycle. This exercised the detection logic and re-synchronized the trailing edge of EMPTY 200 million times a second.

More specifically, we wrote an ascending data sequence at 200 MHz and read it out at 500 MHz. We wrote the output data directly into a second FIFO at the same 500 MHz. We then read the second FIFO out at the original 200-MHz rate.

The combined dual FIFO forms a synchronous system but with asynchronous data transfer between the two halves. When we synchronously subtracted the input data from the output data, the difference was constant, indicating flawless transfer at the 500-MHz read/write rate and no flag synchronization problem – even at this high rate.

When two clock frequencies are uncorrelated, each read clock cycle has a different phase relationship with respect to the write clock. During any second, the active read clock edge steps across the ~5 ns write clock period in ~200 million different phase orientations, thus creating a timing granularity of 0.025 femtoseconds. This resolution is millions of times better than any conventional deterministic test methodology can possibly achieve.

We ran this test for several weeks, with more than  $10^{14}$  operations, without any errors.

word after the FIFO has been empty. In subsequent operations, no behavioral difference exists between the two modes.

### *Asynchronous vs. Synchronous Operation*

The main purpose of most FIFOs is to bridge between independent clock domains; most FIFO applications thus use separate and uncorrelated clocks for writing and reading. Because the trailing edge of each flag must be re-synchronized to the opposite clock, there is an unavoidable one-clock-period ambiguity about the delay of the rising flag edge. This can increase the delay for flags going inactive and thus can cause a very small performance loss. The operation is less predictable, but only while the FIFO recovers from the empty or full condition.

In certain applications, there is only one clock domain, and write and read clocks are therefore identical. In that case, you can (optionally) set the mode to “synchronous.” This eliminates re-synchro-

nization circuitry, reduces the trailing flag delay, and completely avoids the delay uncertainty. The performance improvement is very small.

### **External Memory**

When a design needs multiple megabytes of memory, it is best implemented in external DRAM devices. High-performance SDRAM controllers can pose a design challenge, but Xilinx offers several application notes and well-documented cores and evaluation boards that implement several different memory controller designs.

### **Conclusion**

Virtex-5 memories and memory interfaces have come a long way since the first LUT RAM appeared in the XC4000 family 16 years ago. Versatile dual-ported block RAMs with FIFO and ECC options simplify system design, while well-documented interface designs allow for unlimited expansion with external DRAMs.