WP360 (v1.0) February 23, 2010

# Xilinx FPGA Embedded Memory Advantages

*By: Nick Mehta*

The Virtex®-6 and Spartan®-6 architectures feature flexible internal memory resources that can be configured in a variety of different sizes. This white paper details the available features, illustrating the wide array of memory sizes available and shows the trade-off of using different resources to perform memory functions of different sizes.

# Usage of Internal Memory Resources in Designs

Almost every design built within an FPGA requires the use of internal memory resources of some size for storage of coefficients, buffering of data, and a variety of other uses. Typical systems need a combination of small, medium, and large memory arrays to meet all of their requirements, with the overall power consumption of the memories, and therefore the FPGA, being a primary concern.

When laying out an FPGA, it is important to create a device that meets the majority of customer needs. If an FPGA is built with small, medium, and large memory resources that perfectly suit one application, the solution will be optimal for some customers while others wanting to use the same part will likely need to make considerable trade-offs.

Users trying to get the best value out of their FPGAs can be concerned about wasted bits in larger RAMs. Building finer RAM granularity, though, requires additional FPGA interconnect, at a cost. This white paper explains the trade-offs: why the cost of finer RAM granularity is often higher than the cost of otherwise wasted bits.

Figure 1 shows a theoretical distribution of small, medium, and large memory blocks within an FPGA (not drawn to any particular scale).

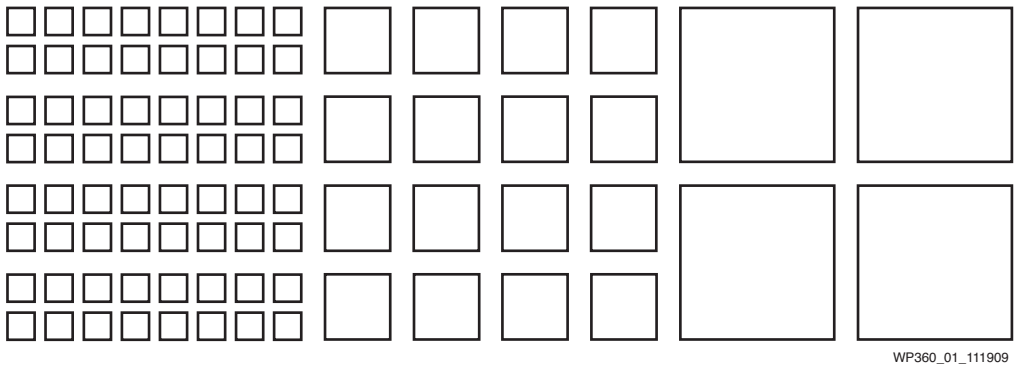


*Figure 1:* **Representation of Memory Arrays of Different Size**

A design that requires exactly this combination of blocks is able to make perfect use of the available resources (see Figure 2).

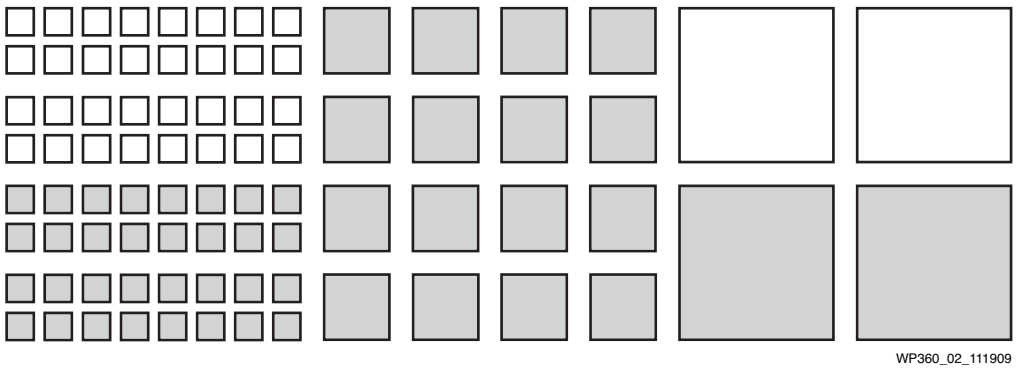


*Figure 2:* **Representation of Memories Mapping Perfectly into Available Resources**

However, imagine a scenario in which a user requires just four more of the medium sized resources. One possibility is to build the medium-sized memory array out of lots of small components, consuming lots of resources and incurring the complexity of

connecting them together. Another alternative is to use one large block as a medium block, rendering all the remaining resources in the large block unusable while keeping them powered on, therefore consuming power (see Figure 3).

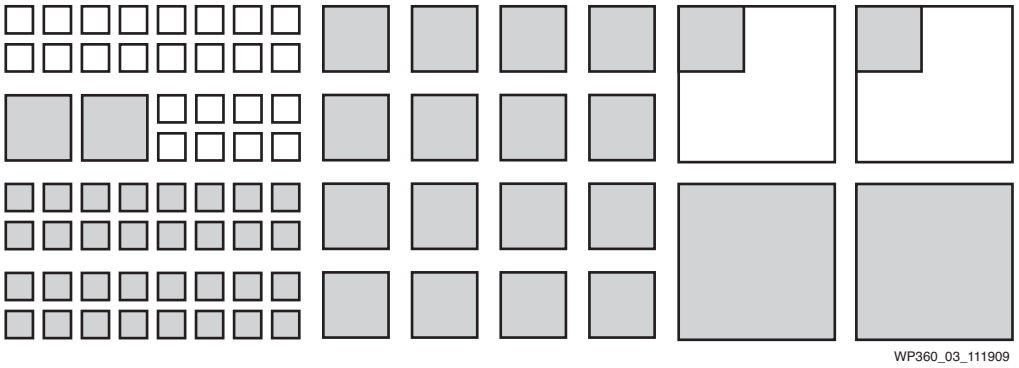


*Figure 3:* **Representation of Memories Having to Use Non-Ideal Resources**

The challenge for an FPGA manufacturer is to build devices with the most flexible combination of memory resources, giving all users the ability to fit their desired memory array size into the device while achieving the desired performance—without wasting vast quantities of resources and power.

# FPGA Memory Resources

Xilinx® FPGAs all use a variety of memory resources to give the best-in-class combination of flexibility and low cost—or cost per bit. Virtex-6 FPGAs and Spartan-6 FPGAs leverage similar building blocks.

It is evident that building memory resources to fulfill the requirements of every user is a difficult challenge. The solution implemented in the Xilinx FPGA is to create blocks called block RAM (see Figure 4) that can either be combined together to make larger arrays or divided to make smaller arrays. The ability to combine block RAMs with 6-input look-up tables (LUTs) in the FPGA logic as small memory arrays provides the user the most flexible resources from which to create their various-sized memory arrays.

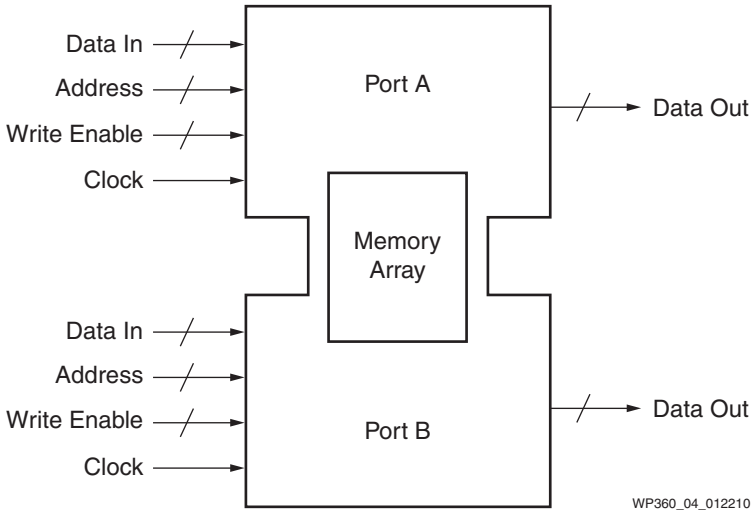


*Figure 4:* **A Block RAM**

## Block RAM

Every Virtex-6 FPGA has between 156 and 1,064 dual-port block RAMs, each capable of storing 36 Kb, 32 Kb of which is allocated to data storage and, in some memory configurations, an additional 4 Kb allocated to parity bits. Each block RAM has two completely independent ports that share nothing but the stored data.

Each port can be configured as:

- 32K x 1
- 16K x 2
- 8K x 4
- 4K x 9 (or 8)
- 2K x 18 (or 16)
- 1K x 36 (or 32)
- 512 x 72 (or 64)

Each block RAM can be divided into two completely independent 18 Kb block RAMs that can each be configured to any aspect ratio from 16K x 1 to 512 x 36. When a 36K block RAM is split into two independent block RAMs, each of the two independent block RAMs behaves exactly like a one 36 Kb block RAM, just half the size.

Conversely, if the user requires larger memory arrays, two adjacent 36 Kb block RAMs can be configured like one cascaded 64K x 1 dual-port RAM without any additional logic.

Similarly, the Spartan-6 FPGA family provides block RAM components. Each is capable of storing 18 Kb, 16 Kb of which is allocated to data storage and, in some memory configurations, an additional 2 Kb allocated to parity bits. Each 18 Kb block RAM can be divided into two independent 9 Kb memory arrays. Every Spartan-6 FPGA has between 12 and 268 block RAMs, resulting in up to 4,824 Kb of memory. When used as single 18K block RAMs, each component in the Spartan-6 architecture can be configured as:

- 16K x 1
- 8K x 2
- 4K x 4
- 2K x 9 (or 8)
- 1K x 18 (or 16)
- 512 x 36 (or 32)

When used as two independent 9 Kb block RAMs, each can be configured to any aspect ratio from 8K x 1 to 256 x 36.

The block RAM components in both Virtex-6 and Spartan-6 architectures can be configured in Single Port, Simple Dual Port, or True Dual Port modes. Additionally, data can be read from the block RAM in one of three ways: READ_FIRST, WRITE_FIRST, or NO_CHANGE mode. For more information on the permitted configurations and write modes, refer to the *Spartan-6 FPGA Block RAM Resources User Guide* [Ref 1] and the *Virtex-6 FPGA Memory Resources User Guide* [Ref 2].

### Splitting Block RAM

If the user only requires Single Port memories, rather than implementing full True Dual Port functionality, it is possible to divide the block RAMs into smaller memory

arrays. When a block RAM is in True Dual Port mode (the default mode), Port A and Port B can implement separate, independent delay line memories, Single Port memories, or ROMs by connecting the most significant bit of the ADDRA address bus to $V_{CC}$ and the most significant bit of the ADDRB address bus to ground, thus creating two Single Port block RAMs. For example, a RAMB16BWER, the 18K block RAM primitive in the Spartan-6 architecture can be split into two 9K Single Port block RAMs by splitting the address space in this way. Likewise, the RAMB8BWER, the 9K block RAM primitive in the Spartan-6 architecture can be split into two 4K Single Port memories.

The same method can be applied to the block RAM primitives in the Virtex-6 architecture. The RAMB36E1, the 36K block RAM primitive can be divided into two 18K Single Port memories, and the RAMB18E1 can be divided into two 9K Single Port memories. Using this method, it is possible to create four delay line memories per block RAM array in both the Virtex-6 and Spartan-6 architectures. To implement delay lines in the block RAM in this fashion, READ_FIRST or Read-Before-Write mode must be used. (In Read-Before-Write mode, data previously stored at the write address appears at the outputs while input data is being stored in the memory.) If implementing Single Port memories, there is no such limitation on the permitted modes; any supported mode (READ_FIRST, WRITE_FIRST, or No_Change) can be used, and the different memories within one block RAM can have different port widths. The resulting memory schemes perform one operation per clock cycle for each port, and therefore four operations per block RAM per clock cycle.

## Synchronous Operation

Each memory access, whether a read or a write, is controlled by the clock. All inputs, data, address, clock enables, and write enables are registered. Nothing happens without a clock. The input address is always clocked, retaining data until the next operation. An optional output data pipeline register allows higher clock rates at the cost of an extra cycle of latency. During a write operation, the data output can reflect either the previously stored data, the newly written data, or remain unchanged.

## Byte-wide Write Enable

The byte-wide write enable feature of block RAM gives the capability to write eight bit (one byte) portions of incoming data. There are up to four independent byte-wide write enable inputs to the true dual-port RAM. Each byte-wide write enable is associated with one byte of input data and one parity bit. This feature is useful when using block RAM to interface with a microprocessor. Byte-wide write enable is further described in the *Spartan-6 FPGA Block RAM Resources User Guide* [Ref 1] and the *Virtex-6 FPGA Memory Resources User Guide* [Ref 2].

## Error Detection and Correction

Each 64-bit-wide block RAM in the Virtex-6 architecture can generate, store, and utilize eight additional Hamming-code bits, and perform single-bit error correction and double-bit error detection (ECC) during the read process. The ECC logic can also be used when writing to or reading from external 64/72-bit-wide memories. This works in simple dual-port mode and does not support read-during-write.

## FIFO Controller

The built-in FIFO controller in the Virtex-6 architecture for single-clock (synchronous) or dual-clock (asynchronous or multirate) operation increments the internal addresses and provides four handshaking flags: full, empty, almost full, and almost empty. The

almost full and almost empty flags are freely programmable. Similar to the block RAM, the FIFO width and depth are programmable, but the write and read ports always have identical width. First-word fall-through mode presents the first written word on the data output even before the first read operation. After the first word has been read, there is no difference between this mode and the standard mode.

## Distributed RAM

The logic of the FPGA architecture consists of, among other elements, 6-input LUTs. The LUTs are arranged in groups of four and combine to make a slice. There are two types of slices in the Virtex-6 architecture, SLICEM and SLICEL, and three types of slices in the Spartan-6 architecture, SLICEM, SLICEL and SLICEX. LUTs in a SLICEM, which compose 25–50% of the total number of slices in the Virtex-6 and Spartan-6 architecture, can be implemented as a synchronous RAM resource called a distributed RAM element. Each 6-input LUT can be configured as a 64 x 1-bit RAM or two 32 x 1-bit RAMs. The 6-input LUTs within a SLICEM can be cascaded to form larger elements up to 64 x 3-bit in simple dual-port configuration or 256 x 1-bit in Single Port configuration. See Figure 5.

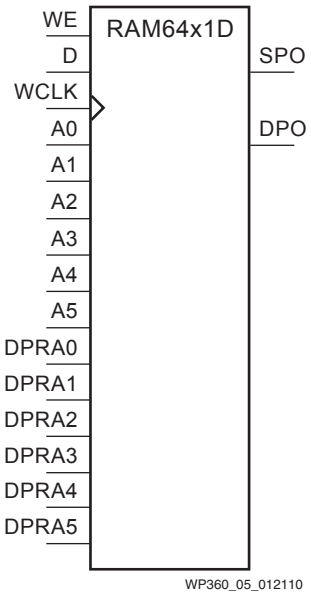


*Figure 5:* **64-Deep by 1-Wide Dual Port Static Synchronous RAM**

Distributed RAM modules are synchronous (write) resources. A synchronous read can be implemented with a storage element or a flip-flop in the same slice. By placing this flip-flop, the distributed RAM performance is improved by decreasing the delay into the clock-to-out value of the flip-flop. However, additional clock latency is added. The distributed elements share the same clock input. For a write operation, the Write Enable (WE) input, driven by either the CE or WE pin of a SLICEM, must be set High.

## A Typical User Design Utilization

Table 1 shows the resource mapping in a typical high memory utilization design targeting a Virtex-6 SX315T FPGA. This data is based on a real-world example from a user concerned about wasted bits.

*Table 1:* **Resource Mapping in a Typical High Memory Utilization Design**

| Requirement | Implementation |
|---|---|
| 16 memories at (32K x 32) | 512 block RAMs (36K configuration) |
| 300 memories at (512 x 18) | 150 block RAMs (18K configuration) |
| 100 memories at (64 x 4) | 100 SLICEMs configured as distributed RAM |

The multiple different resources take advantage of the ability to configure the block RAM resources in different sizes.

## The Cost of Infinite Granularity

Having established that the Xilinx FPGA architectures offer a great variety of different memory depth/width granularities, it is important to understand the trade-offs involved in adding finer granularity to the architecture. If, for example, the Virtex-6 FPGA 36K block RAM is divided into not only two 18K blocks but further into four 9K true dual-port blocks, there is a penalty to pay. Doubling the number of unique memories within a single block RAM means that the maximum number of signals to be routed into and out of each block RAM also needs to double, which, in turn, necessitates double the number of interconnect resources (or tiles) to accommodate routing of the ~400 signals.

For example, an 8K block RAM can be configured as 16 bits wide and 512 bits deep, requiring 16 data lines and 9 address lines for a total of 25 input signals. Then divide the 8K block RAM into two 4K block RAMs. Each of these block RAMs can be configured as 16 bits wide and 256 bits deep. This configuration requires 16 data lines and 8 address lines per block, for a total of 24 input signals per 4K block RAM or 48 signals total. The 48 signals are roughly twice the 25 input signals needed for the 8K block RAM.

The effect of doubling the number of interconnect tiles associated with each block RAM increases the silicon area by 25% — a penalty that is applied to all blocks regardless of the configuration in which they are used. Therefore, the ability of every block RAM to be configured as four 9K blocks means that each block increases from four area units to five area units. On the positive side, it allows smaller memories to take advantage of the smaller block size (see Table 2).

Referring to the same design example, the impact amounts to the smaller 512 x 18 memories consuming fewer resources if they are able to be packed efficiently into adjacent 9K blocks. However, the larger blocks all become 25% larger, adding a significant area penalty.

*Table 2:* **Area Impact on Typical High Memory Utilization Design**

| Resources and Silicon Area Used | | |
|---|---|---|
| | **If Supporting 36K/18K** | **If Supporting 36K/18K/9K** |
| 16X (32K x 32) | 512 blocks of 4 area units | 512 blocks of 5 area units |
| 300X (512 x 18) | 150 blocks of 4 area units | 75 blocks of 5 area units |
| | | |
| Difference | | +512 area units<br>-(3 X 75) area units |
| Result | | +287 area units |

Thus, if the block RAMs in the Virtex-6 FPGA can be divided into four independent memories and the user design can take full advantage of these smaller granularity blocks, this typical design still consumes a far greater area than the current 36K/18K configuration.

It is, however, unlikely that a design can take full advantage of having four independent 9K true dual-port memories by packing them into the same 36K block RAM due to the routing congestion of trying to connect so many signals to a single location. The area penalty in Table 2 is, therefore, a best case scenario; in reality, not all 36K block RAMs would be populated by four 9K memories.

## Impact on Device Resources

The impact of the block RAM resources increasing in size can have one of three results in context of overall device resources:

- Keep the device size the same and lose block RAM bits
- Keep the number of block RAM bits the same and reduce the quantity of other resources, i.e. CLBs
- Leave all resources as they are and allow the device to become larger

There are obvious penalties associated with each of these choices:

- Losing block RAM bits is not ideal because most users are utilizing the majority of the available memory. If the block RAM are all 25% larger, the number of blocks in a Virtex-6 SX315T FPGA decreases from 704 to 563, equating to a reduction from 25,344 Kb to 20,268 Kb.
- A 25% increase in block RAM size means the loss of one column of CLBs per block RAM column. The Virtex-6 SX315T FPGA has fifteen block RAM columns, so in this scenario, it loses fifteen columns of CLBs from the device, equating to a loss of 7,200 CLBs, which translates into approximately 92,000 logic cells. This would have the impact of reducing the SX315T from 315,000 logic cells to 223,000 logic cells, resulting in a much less capable device at the same cost.
- Keeping all the resources the same and allowing the silicon area to increase has several drawbacks. The obvious first issue is that a larger device means a more expensive device, but the physically larger silicon also has a significant impact on power consumption.

## Software

XST inference and CORE Generator™ software flows are available.

### Inferring RAMs with XST

XST infers block RAM if the read address is registered inside the module. Conversely, it infers distributed RAM if the RAM network uses an asynchronous read operation. The ram_style attribute can be used to specify whether block RAM or distributed RAM is used.

When the RAM network is larger than is supported by one component, multiple block RAM can be used. The default strategy is to optimize for performance. In this strategy, small RAMs are implemented using distributed RAM. RAM networks can also be optimized for power and area; these are discussed in the Building RAMs with CORE Generator Software section.

Anything smaller than 128-bits deep or 512-bits depth x width is implemented in distributed RAM unless the user specifies otherwise with the ram_style attribute.

To ensure the most efficient use of both block RAM and distributed RAM resources, XST first implements the largest RAMs in block RAM, then places the smaller RAMs in block RAM if there are still block RAM resources available. XST can also pack small single-port RAMs into a single block RAM. Refer to the *XST User Guide for Virtex-6 and Spartan-6 Devices* [Ref 3] for more information.

ROMs can be inferred by the use of large Case Statements.

XST can also implement Finite State Machines (FSM) and logic in block RAM to maximize the available logical resources. See the *XST User Guide* for more information.

## Building RAMs with CORE Generator Software

The CORE Generator tool has three algorithms by which it can optimize block RAM networks. Minimum Area scheme uses the fewest possible resources (block RAMs) but also minimizes output multiplexing to maintain the highest performance at the smallest area (see Figure 6). Low Power scheme can use more block RAM but ensures the fewest blocks are enabled during each read and write operation. This can result in a small quantity of additional decoding logic on the enable signal but it is a small real estate penalty compared to the power saving made.

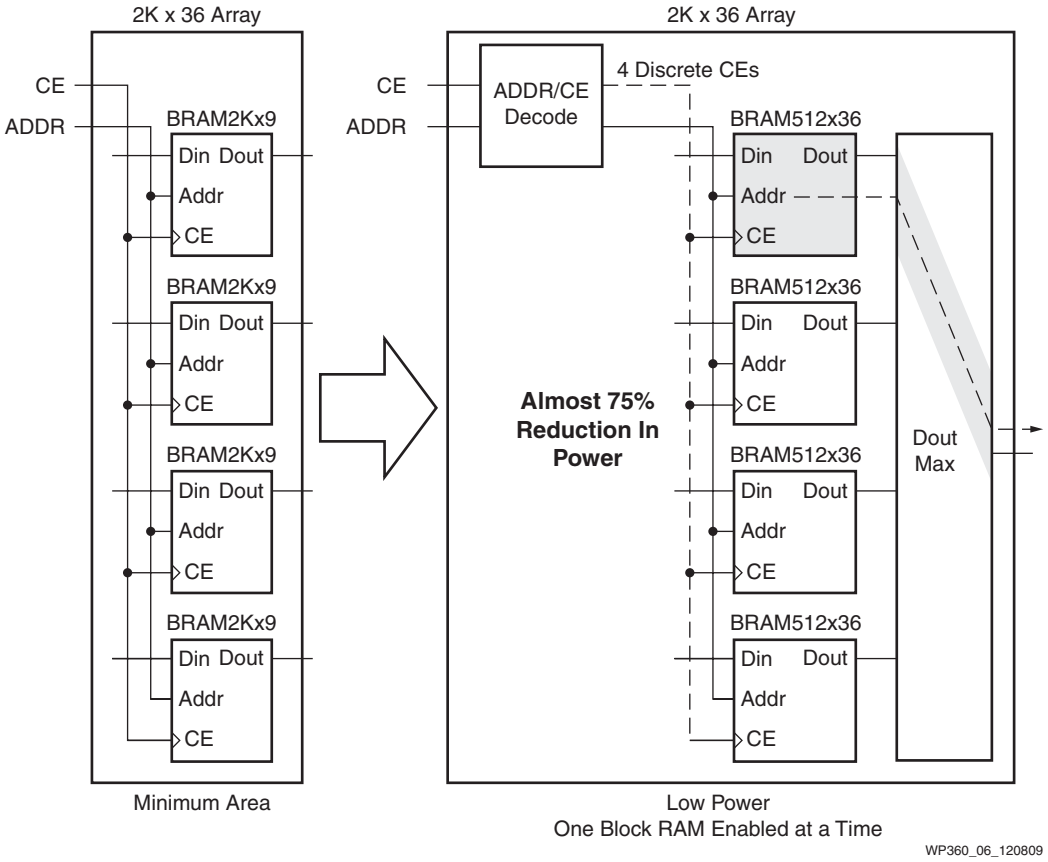


*Figure 6:* **CORE Generator Software Method of Lowering Total Block RAM Power Consumption**

The third optimization scheme is Fixed Primitive, in which the user can choose a specific primitive, e.g., 4K x 4, from which to build their RAM network. It is up to the user to decide which primitive is best for their application.

The CORE Generator tool also provides an option to register the outputs of the RAM network to improve performance. If multiple block RAMs are used in the network, the

user can choose whether to register at the output of each block RAM primitive or at the output of the core.

# Conclusion

Combining block RAM that can be configured in a variety of data width/depth combinations with distributed RAM to support smaller memory arrays offers the most flexible way of building memories of different sizes at the lowest cost per bit of memory. Adding additional functionality to a component, such as adding ports to a block RAM, can initially appear to be the best solution if that exact functionality is required. However, there are area penalties to both the component and its corresponding interconnect for adding new features. These area penalties have a significant impact on device resources. Many years of experience developing and building FPGA embedded memories have led to efficient solutions across a wide range of applications.

# References

This white paper uses the following references:

1. UG383, *Spartan-6 FPGA Block RAM Resources User Guide.*
2. UG363, *Virtex-6 FPGA Memory Resources User Guide.*
3. UG687, *XST User Guide for Virtex-6 and Spartan-6 Devices.*

# Additional Resources

The following resources provide additional information useful to this white paper:

- UG364, *Virtex-6 FPGA Configurable Logic Block User Guide*.
- UG384, *Spartan-6 FPGA Configurable Logic Block User Guide*.
- DS512, *Block Memory Generator v3.3*.
- DS322, *Distributed Memory Generator v4.3*.

# Revision History

The following table shows the revision history for this document:

| Date | Version | Description of Revisions |
|---|---|---|
| 02/23/10 | 1.0 | Initial Xilinx release. |

# Notice of Disclaimer

MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE INFORMATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS.