WP248 (v1.0) July 31, 2007

Retargeting Guidelines for Virtex-5 FPGAs

By: Brian Philofsky

When migrating or retargeting code from a previous design into a Virtex™-5 platform FPGA, some considerations should be addressed. This white paper attempts to identify and detail the appropriate retargeting guidelines.

Virtex-5 Device Selection and Post-Retarget Analysis

Due to differences in the base architecture of the Virtex-5 device from previous FPGA generations, selection of the right Virtex-5 device for a given design is not always straightforward. In most cases, a design should fit in a similar array size (device number) and at least one slower speed grade to the previously targeted device (e.g., from mid speed grade to slow speed grade). However, there are some circumstances that can change that recommendation. This section describes some general design styles and characteristics that can influence the device selection criteria for the Virtex-5 FPGA.

Register-Rich Designs

Virtex-5 device names (e.g., XC5VSX35T, XC5VLX85) are selected based on a numerical value associated with the available logic in each FPGA. Since Virtex-5 devices have more logic per register due to the new 6-input LUT, a given Virtex-5 device number has approximately 30% fewer registers than the same Virtex-4 device number.

For example, the XC4VSX35 has 30,720 slice registers while the XC5VSX35T has 21,760. Another example is the XC4VLX80, which has 71,680 slice registers while the XC5VLX85 has 51,840. For designs that use less than 60% of the available slice registers, the difference in the number of total slice registers per device is not a concern. However, when a Virtex-4 device design that consumes greater than 60% of the slice registers is being retargeted to a Virtex-5 device design, a larger Virtex-5 device can be required.

Well-Crafted Older Architectures

Designs that run fast in a Virtex-4 device (i.e., at 300 MHz or above) see little improvement in performance or reduction in LUT utilization with a Virtex-5 device. This is because fast designs that are well-optimized to the 4-input LUT structure, with low fan-in logic and few logic levels between synchronous objects, are more likely to have one-to-one mapping from the 4-input LUT to a 6-input LUT. This means that there is little potential for LUT or logic reduction with the new Virtex-5 device logic structure, and thus, little improvement in performance.

Many times, the retargeted designs that benefit most from the larger LUT structure in Virtex-5 devices are those that previously ran relatively slowly, with many logic levels and larger LUT-to-register ratios. In these designs with large fan-in logic cones, there is greater potential for the 6-input LUT to considerably reduce the number of logic levels as well as the number of LUTs necessary to build a logic function. Such designs achieve a greater LUT reduction and performance boost when retargeting to the Virtex-5 device architecture.

Better Logic Utilization in Virtex-5 Devices

Some types of logic functions benefit from the 6-input LUT more than others. For example, a 32-bit XOR gate consumes seven 6-input LUTs (with some logic to spare) where the same function would require eleven 4-input LUTs (with no logic to spare). This represents a 36% reduction in the required number of LUTs.

However, a single 2-to-1 MUX maps into a single 4-input LUT in the same manner as a single 6-input LUT, so there is no advantage to using the 6-input LUT. A 2-input adder requires one 4-input LUT and one 6-input LUT per bit of addition. Soft

multipliers do not benefit very much from the Virtex-5 device logic structure over previous generations. Thus, certain types of designs that use MUXs, adders, soft multipliers, and other logic functions do not get better utilization or performance from Virtex-5 devices. For example, DSP designs generally use these logic functions and a lot of pipelining as the core for many of their operations and, therefore, do not realize many benefits from the Virtex-5 device architecture. Alternatively, some embedded processor designs realize more benefits from Virtex-5 devices because they:

- use fewer registers (less pipelining)
- have higher fan-in functions
- use fewer functions that do not get the mapping improvements for the 6-input LUT

Software Algorithms

Current software algorithms are designed to deliver the highest performance possible, often at the expense of area or number of LUTs. However, as the software matures, the software algorithms continue to improve performance, with a greater focus on area. Consequently, it is desirable to retarget an existing design to improve performance.

In the short term, some things can be done to improve area at the cost of performance in a design with adequate timing slack. First, synthesis timing constraints can be imposed that relate realistic timing objectives. In doing so, the synthesis software can apply area-saving algorithms in which performance can still be met. Without timing constraints, the synthesis tools have no choice but to optimize all parts of the design for timing, often at the expense of area. Second, the LUT and slice packing behaviors can be changed within the Map portion of the ISE™ software. The 9.1i version of the software contains a hidden switch option called **`-lc`** that employs a LUT compression algorithm to reduce the number of LUTs required in a design. The **`-lc`** switch accepts two arguments, **`auto`** and **`area`**. When set to **`-lc auto`**, the software attempts to combine LUTs with little impact on timing. When set to **`-lc area`**, LUT packing is done with a greater impact on timing. This algorithm can be used to get better LUT utilization for designs that have margin in performance or power. However, it is not recommended for use in designs that have difficulty in meeting performance or power budgets. Another method to reduce the required number of LUTs is slice compression. This is done by setting the **`-c 1`** switch in Map, which invokes both LUT and slice compression, resulting in fewer required slices. However, this is often at the cost of performance and power, perhaps even more so than the **`-lc area`** switch.

Use of Old Cores, EDIF, or NGC Netlists

It is highly recommended to regenerate or resynthesize any old cores or black box netlists in the design prior to implementation in the Virtex-5 device. Most netlists targeting older Spartan™ Generation and Virtex device architectures can be implemented without error when targeting Virtex-5 devices. However, in almost every case, such netlists can result in slower and larger implementations (with more logic levels and utilizing more resources, respectively). This trade-off is due to the differences in the underlying fabric architecture (e.g., 4-input LUT vs. 6-input LUT) as well as the timing and optimizations for the different architectures. Retargeting of these black box instances ensures that they are optimized for the Virtex-5 device architecture.

When resynthesis is not possible, an alternative is to use global optimization (**-global_opt switch**) during the Map phase. Although global optimization is

generally not as effective as direct resynthesis of the black box instance, it allows the tools to resynthesize and restructure the netlist, which can result in a better end implementation. The main disadvantages of using global optimization are increased runtime and possibly further obscurity (renaming and restructuring) to the internal logic paths, which can make design debug more difficult.

Use of Physical Constraints

Any LOCs, RLOCs, BEL constraints or other physical constraints embedded in the code or netlist of the existing design should be removed before retargeting to a Virtex-5 FPGA. An optimal placement for a previous architecture would likely not be optimal in a Virtex-5 FPGA due to differences in the slice structure, the CLB, RAM (LUT RAM or block RAM), logic, I/O layout, and timing differences. In some cases, errors can occur due to layout and coordinate differences. However, even if no errors occur, timing, density, and power could be suboptimal unless the physical constraints are removed or updated for the new architecture.

Use of Control Signals

The use of control signals (signals that control synchronous elements such as clock, set, reset, and clock enable) can have an impact on device utilization. This is true in almost any FPGA technology. However, the difference in the topology of the Virtex-5 device has changed some of the considerations in selecting and using control signals. This section illustrates some of these considerations necessary to achieve the best device utilization.

Initialize Inferred Registers to Known Logic Value

All inferred registers should be initialized to a known value when declaring the signal or register. This can result in better device utilization, performance, and power, and improves verification of the design.

Limit Use of Active-Low Control Signals

It is not recommended to use inferred or instantiated components with active-Low control signals due to the combination of:

- the Virtex-5 device's coarser granular slice composition
- the absence of a programmable inversion element on the slice control signals in the Virtex-5 device
- hierarchical design methods that do not allow optimization across hierarchical boundaries

In certain situations, device utilization can decrease due to the use of a LUT as an inverter and the additional restrictions of register packing sometimes caused by active-Low control signals. Timing can also be affected by the use of active-Low control signals. Active-High control signals should be used wherever possible in the HDL code or instantiated components. When a control signal polarity cannot be controlled within the design (e.g., when it is driven by an external, non-programmable source), the signal in the top-level hierarchy of the code should be inverted and active-High control signals that are driven by the inverter to get the same polarity (functionality) should be described.

Limit Use of Low Fan-Out Control Signals

The number of unique control signals in the design should not be grouped and limited. Also, several low fan-out clock enables, sets, or resets should not be coded in the design for the same reasons explained in "Use of Old Cores, EDIF, or NGC Netlists." Low fan-out, unique control signals can result in underutilized slices in terms of registers and LUT RAM and can have negative impacts on placement and timing. A set, reset, or clock enable should not be implemented in the code unless it is required for the active functionality of the design.

Unnecessary Use of Sets or Resets

Unnecessary sets and resets in the code can prevent the inference of SRLs, RAMs (LUT RAMs or block RAMs), and other logic structures that would otherwise be possible. To get the most efficiency out of the architecture, sets and resets should only be coded where they are necessary for the active functionality of the design. They should not be coded where they are not required. For example, a reset is not required when it is only used for initialization of the register because register initialization occurs automatically upon completion of configuration. Another example is when a circuit remains idle for long periods and a simple reset on the input registers eventually flushes out the data on the rest of the circuit. A similar example is in the case of the inner registers when the reset is held for multiple clock cycles. In this case, the inner registers are flushed during reset; the reset is, therefore, not necessary. By reducing the use of unnecessary sets or resets, greater device utilization and improved performance can be achieved.

Sets for Multipliers or Adders/Subtractors in DSP48E Slice Registers

DSP48E slice registers contain only resets and not sets. For this reason, unless necessary, a set (value equals one upon an applied signal) should not be coded around multipliers, adders, counters, or other logic that can be implemented within a DSP48E slice.

Use of Synchronous Sets/Resets

If a set or reset is necessary for the proper operation of the circuit, a synchronous reset should always be coded. Synchronous sets/resets not only have improved timing characteristics and stability but can also result in smaller, better utilization within the FPGA. Synchronous sets/resets can result in less logic (fewer LUTs), fewer restrictions on packing, and, often, faster circuits. If it is not desired to recode existing asynchronous resets to synchronous resets, the asynchronous resets can be treated as synchronous resets by using the **treat asynchronous resets as synchronous** switch, if available, in the synthesis tool. If Xilinx Synthesis Technology (XST) is the synthesis tool, this switch is available in the GUI. If Synplify synthesis software is being used, this function is available as a TCL command. This is not as effective as recoding to use a synchronous reset in terms of reducing resources and improving performance. However, it does allow for some register packing where it would not be possible otherwise.

Use of Clock Enables

When high fan-out clock enables are used, they should not be manually split or replicated but coded as a single clock enable. If replication becomes necessary for timing or other reasons, it should be controlled within the synthesis tool. Another method is to use a global buffer (BUFG) to distribute the high fan-out signal. However,

this should be done with caution, especially when the clock enable source is driven by an external pin that is not specified to a global clock pin.

Analysis and Use of Control Sets

If it becomes necessary to analyze the design for the use of unique control sets (groups of clocks, clock enables, sets, and resets), the Map `.mrp` report can give details about each control set. To view the control sets, the -detail switch should be enabled during Map. This populates section 13 of the report with the control set information.

If it is suspected from the report that a large number of unique, low fan-out control signals are causing low register utilization, steps can be taken within the synthesis tools to control the inference of control signals for either part of the design (i.e., a particular net or hierarchy) or globally in the design. Disabling the use of the dedicated control signals in the registers can cause greater logic utilization and higher register packing can be realized. For register-intensive designs, it is important to balance the use of these synthesis controls to not shift from a register-packing issue into a LUT resource limitation.

To control clock enables within the XST application, `use_clock_enables = no` should be used on either a register signal, module, or globally on a design. In the Synplify software, this control can be done on inferred register instances using the `syn_useenables = 0` attribute. Alternatively, using TCL, the clock enables can be controlled by groups, hierarchy, or globally by using this construct:

```
# Globally remove all CE inference
define_scope_collection yourname {find -hier -seq *}
define_attribute {$yourname} syn_useenables {0}

# Remove CE inference at a hierarchy
define_scope_collection yourname {find -seq
{hierarchy1.hierarchy2.*}}
define_attribute {$yourname} syn_useenables {0}

# Remove CE inference for a group of like instances
define_scope_collection yourname {find -hier -seq -inst
{inst_prefix_name*}}
define_attribute {$yourname} syn_useenables {0}
```

RAM Considerations

To maximize the use of block RAMs and LUTs in the Virtex-5 device architecture, certain considerations must be understood when retargeting block RAM and LUT RAM by inference, instantiated primitive, or CORE Generator™ software. If the CORE Generator software is used for RAM generation, the core should be regenerated for the Virtex-5 device, or the RAM should be recoded for proper synthesis inference. Either method often gives good results for utilization and performance. However, it is recommended to infer memory where possible to improve understanding of the code, simulation, and future portability of the code.

Instantiating RAMs

The recommendations in this section are for cases in which RAM primitives are instantiated in the design or when it is not possible to regenerate CORE Generator software IP for Virtex-5 devices. These suggestions should also be implemented by

code that infers RAM, especially when using synthesis attributes to guide which RAM resources are used (e.g., `syn_ramstyle = blockram`). The suggestions are divided by RAM depth, which is generally the biggest factor in determining which RAM resource to use.

Depths Greater Than 16 Bits but Less Than or Equal To 256 Bits

Due to the larger LUTs and, thus, deeper LUT RAMs in the Virtex-5 FPGA, the criteria for choosing between a block RAM and LUT RAM are different compared to those for previous FPGA generations. In general, a LUT RAM should be used for all memories 64 bits or less, unless there is a shortage of logic resources (LUTs) and/or SLICEMs for the particular target device. Using LUT RAMs for memory depths of 64 bits or less, regardless of data width, is generally more efficient in terms of resources, performance, and power. For depths greater than 64 bits but less than or equal to 256 bits, the decision on the best resource to use depends on several factors. First, are extra block RAMs available? If the answer is no, LUT RAM should be used. Second, what are the latency requirements? If it is necessary to have asynchronous read capability, LUT RAMs must be used. Third, what is the data width? Widths greater than 16 bits should probably use block RAM, if available. Last, what are the necessary performance requirements? Register LUT RAMs generally have shorter clock-to-out and fewer placement restrictions than block RAMs.

If the design already contains instantiated LUT RAMs with depths greater than 16 bits, the deeper primitive (e.g., RAM32X1S, RAM64X1S, etc.) should be used. RAM16X1Ss used in conjunction with MUXF5s or other logic will not be properly retargeted to use the greater depth LUT automatically. In such a case, the code should be modified to properly use the deeper primitives.

Depths of 512 Bits

When specifying a depth of 512 bits, instantiated block RAMs from previous architectures can be efficiently retargeted if both port widths are 18 bits or less. If one or both port data widths are set to 36 bits, the block RAM can often be efficiently retargeted. However, special considerations exist for this scenario. Consider these three cases for block RAM depths of 512 x 36 bits.

True Dual-Port

Width should be considered if a block RAM is used in true dual-port mode (i.e., dual-port RAM with one or both ports requiring both read and write capability). If 18 bits or less of data is required on all ports, this can be efficiently mapped to a Virtex-5 device block RAM, albeit with some modification. If a Virtex-II device RAMB16_Sx_S36 or a Virtex-4 device RAMB16 is used with one or more WIDTH parameters set to 36, the block RAM consumes an entire RAMB36 block to be implemented. To allow a retarget, the code should be modified to either infer the block RAM or instantiate a RAMB18 component with the proper settings. Alternatively, the RAMB16_Sx_S36 primitive can be changed to a RAMB16_Sx_Sy primitive, where both x and y are 18 or less. For a Virtex-4 device, the source should be modified to have all WIDTH parameters set to 18 bits or less. This can be the easier route but is the less portable way to implement this change.

If the true dual-port block RAM uses any port width greater than 18 bits, an entire RAMB36 site is consumed. If the design runs out of block RAM resources, the only way to reclaim the lost portion of the block RAM is to time-multiplex the read and write operations to the block RAM. This time-multiplexing not only requires extra logic to be described but also a significant timing margin. In general, therefore, this

technique can only be used on block RAMs with slow clocks (slower than 150 MHz) with a few nanoseconds of extra timing margin on the paths entering and/or exiting the block RAM. If the timing margin does not exist, it is likely that the block RAM bits need to be sacrificed.

Simple Dual-Port

A Virtex-5 device block RAM can be used efficiently in simple dual-port mode (i.e., one port read-only, the other port write-only). However, under some retargeting scenarios, a full RAMB36 block can be used in place of a RAMB16 block from a prior architecture. In the case of older Virtex-II device block RAM primitives (e.g., RAMB16_S36_S36), the ISE software prior to version 9.2i does not have the ability to determine whether the block RAM is used in simple dual-port mode and, thus, maps the block RAM into an entire RAMB36 site. The lost half of the block RAM can be reclaimed by replacing the RAMB16_S36_Sx RAM with either a RAMB18SDP primitive or with the proper inference block RAM, or upgrading to the latest version of the ISE software. Similarly, if a Virtex-4 device RAMB16 component is used in a version of ISE software prior to 9.2i, the software can identify whether the RAMB16 component is used as a simple dual-port by analyzing the WIDTH attributes. If one port has WRITE_WIDTH = 0 and the other port has READ_WIDTH = 0, the software can remap that block RAM component into a RAMB18SDP and get efficient use of the block RAM. If, however, all port widths have a non-zero value and at least one WIDTH is set to 36, an entire RAMB36 block is used to map the RAM function. For this reason, all RAMB16 instantiations should be investigated to determine whether they are used in simple dual-port mode, and if so, all unused ports should have their WIDTH parameters set to zero. ISE software version 9.2i and later can identify a block RAM being used in simple dual-port mode from the manner in which it is connected and, under most circumstances, use the proper Virtex-5 device block RAM primitive in its place.

If the block RAM is used in simple dual-port mode but with different data widths for the read and write operations, and one of those data widths is equal to 36, one of two choices can be made. The first is to do nothing; thereby, an entire RAMB36 block is consumed to implement this block RAM and no additional resources are used. The second alternative is to instantiate a RAMB18SDP component, but additional logic (multiplexing data and addressing logic) must be added to change the static 36-bit port width to the appropriate size. This alternative not only consumes additional logic (LUTs) but can affect the performance of the block RAM. Thus, the choice between these two implementations depends not only on the available resources but also the performance requirements for this block RAM.

Single-Port

If the block RAM is used as a single-port RAM with the input and/or output data width set to 36 bits, it can be efficiently implemented in a Virtex-5 device but can require changes to the code. If instantiated as a RAMB16_S36 (RAMB16 in 36-bit single-port mode), the block RAM should be replaced by an instantiated RAMB18 component in which read and write data widths for both ports are set to 18 bits. The CLK pins for both ports are tied to the same clock source. If a single (non-byte enable) write enable (WE) is desired, all WE pins should also be tied to the same source. The ADDR pins should be connected so that the nine address pins are tied to ADDRA[14:6] and ADDRB[14:6] of the block RAM. The ADDRA[5] pin connects to a logic zero and the ADDRB[5] ties to a logic one, thus splitting the 1K-deep address space between the two ports of the RAM. The 36 bits of data should then be connected to the input and output data pins for both ports, with the lower 18 bits residing on port

A and the upper 18 bits on port B. When connected in this fashion, the block RAM should efficiently and equivalently map into a Virtex-5 device.

Depths Greater Than 16K

If building a block RAM with depths greater than 16K, the new deeper block RAM memories can create a more efficient and higher performance block RAM implementation. If the deeper block RAM is described using two cascaded Virtex-4 device RAMB16 primitives, the software should be able to automatically retarget the block RAM to a RAMB36 component. If, however, this deeper block RAM is described with older RAMB16_S1 primitives and/or a depth of 64K is desired, the block RAM should be recoded to inference code, or the RAMB36 or a pair of cascaded RAMB36s should be instantiated to build the proper RAM structure. This makes full use of the Virtex-5 device's deeper block RAM structures.

Current Limitations on Inferring Block RAMs

Most inferred block RAM implementations are optimal for the Virtex-5 FPGA architecture. This section attempts to explain the current known limitations in trying to infer block RAMs using either Synplify 8.8 or XST 9.1i software.

512 x 36 Single-Port RAM

For both XST and Synplify software, a single-port block RAM with a data width greater than 18 bits but less than or equal to 36 bits, and a depth of 512 bits or less, is implemented in a RAMB36 where it can be implemented in a RAMB18. Until this is corrected, the instantiation method in "Single-Port"should be adopted when using block RAMs of this type and size.

Simple Dual-Port Block RAM

Synplify software does not properly infer the RAMB18SDP for simple dual-port block RAMs with a data width greater than 18 bits but less than or equal to 36 bits, and a depth of 512 bits. The documentation for the latest version of the tool should be consulted to determine if this limitation still exists. However, the inference of the wider RAMB36SDP (greater than 36 bits but less than 72 bits) does work properly.

Block RAMs With Different Read, Write, or Port Data-Width Values

XST and Synplify software cannot infer block RAMs with different data-width aspect ratios, either on the read and write of the same port or different widths on ports A and B. Such block RAMs must be instantiated to be incorporated into the design.

Block RAM Depth Not a Power of Two

If the block RAM depth is not defined as or desired to be a power of 2 (e.g., 512, 1024, 2048, etc.), CORE Generator software can sometimes implement a more area-efficient block RAM implementation.

Other Considerations for Best Device Utilization with Block RAMs

To get the most out of the block RAM structures in the Virtex-5 device, other considerations should be understood, especially if the design was previously targeted to a technology older than the Virtex-4 device.

Output Register

Virtex-4 devices contain an output register for the block RAM. However, designs created prior to the Virtex-4 device architecture and many created after it do not use this feature. Use of the output register can significantly improve performance (clock-to-out) of the block RAM, while also improving power and device utilization. If a design is ported into the Virtex-5 device from an architecture prior to the Virtex-4 device, the code should be reexamined to see if the output register can be incorporated into the design.

Byte-Wide Write Enables

Virtex-4 devices also have byte-wide write enables. This feature can be beneficial to the block RAM access and utilization for the device. Thus, byte-wide write enables should be understood and implemented when retargeting block RAMs.

Different Read/Write Aspect Ratios

Virtex-4 and Virtex-5 device block RAMs can have different data widths, not only on the two ports of the block RAM but on reads and writes as well. The different data widths utilize more of the block RAM and allow use cases that do not use the full memory array to access more memory.

FIFO Logic

If the design implements FIFOs that are either created by CORE Generator software or from inferred logic, the FIFOs should be replaced with the dedicated FIFO logic available in the Virtex-5 device. This replacement not only saves on additional logic resources but also simplifies the implementation and timing aspects of the FIFO logic.

ECC Logic

The ECC capability of the RAMB36SDP should be used to self-check and correct block RAM data. ECC logic is not necessarily considered to be a real utilization savings. However, if this capability is important, it should be used during the block RAM retargeting of the design.

Other Primitive Retargeting Considerations

Some device primitives instantiated for previous architectures are not automatically retargeted to the Virtex-5 device architecture, or they are retargeted with some notable differences.

These primitives are retargeted with some possible changes to be considered.

FIFO16

The flag logic (EMPTY, FULL) for the Virtex-5 device FIFO can function differently from the FIFO16 logic, but the FIFO16 is automatically retargeted. If the design contains a FIFO16, the FIFO16 should be manually replaced with a FIFO18 to ensure that the pre-synthesis RTL simulation matches the functionality of the end design. If the FIFO16 is not replaced, a fully functional simulation of the design with the automatically replaced FIFO18 should be run. This is to ensure that the functionality changes of the FIFO flags do not have an adverse effect on the design.

BUFR

The BUFR for the Virtex-5 device has different latency than that in the Virtex-4 device. For most designs with a free-running clock, this should make no difference. However, this difference in latency can affect when the buffer starts up. In the cases where a clock can stop and restart using this buffer, the latency change can have an effect on the design. To update the component's pre-synthesis functionality with the proper latency for the Virtex-5 device, the BUFR instantiation should be updated to add the proper SIM_DEVICE generic (in VHDL) or parameter (in Verilog). The ISE software language templates should be referred to for the proper instantiation template. Regardless of the setting for SIM_DEVICE, the post-implementation gate-level simulation is updated with the proper value for SIM_DEVICE attribute, and gate-level simulations should reflect the proper latency functionality. For this reason, it is highly recommended to run a gate-level functional simulation to ensure that functionality is not affected by this latency difference.

DCM_ADV

In most cases, the DCM_ADV is transparently retargeted for the Virtex-5 device. However, in the cases where the dynamic reconfiguration port (DRP) is used, the address mapping is changed from Virtex-4 to Virtex-5 devices. If the DCM does not use the DRP, no additional modification is necessary. But for DCMs that use the DRP, the differences in the addressing and behavior of the DRP port for the DCM_ADV should be understood and, if necessary, the design should be modified to account for these differences.

BUFGMUX

If the design contains a BUFGMUX, it is automatically retargeted to a BUFGCTRL. This automatic retargeting is similar to that in a Virtex-4 FPGA except for latency differences in the component. When using a single free-running clock, this latency difference should not affect the functionality. However, in the case where a clock is stopped or switched, the timing can be slightly different. Therefore, the BUFGMUX should be replaced either with a BUFGMUX_CTRL or BUFGCTRL, and/or a gate-level timing simulation should be done to ensure that the latency differences do not affect the resulting functionality of the design.

MUXCY, XORCY, MULTAND, MUXF5, and MUXF6

Due to differences in the underlying architecture, the MUXCY, XORCY, MULTAND, MUXF5, and MUXF6 are not retargeted in the most efficient way for area, density, or timing. Designs that contain these instantiated components should be reevaluated to determine whether the code section containing these components can be inferable or else replaced with the structural equivalent primitives from the Virtex-5 device architecture.

These primitives are automatically retargeted.

PMCD

If the design contains a phase-matched clock divider (PMCD), it should be manually replaced with the appropriately configured phase-locked loop (PLL). The effort for this should be lessened by using the PLL architecture wizard to create the proper PLL instantiation.

RAMB4

If the design contains RAMB4 primitives from the original Virtex device or Spartan-II families, those block RAM primitives are not automatically retargeted; they must be manually changed to either inference code (preferred) or an instantiation primitive supported by the Virtex-5 device. The primary reason why this retargeting is not supported is the gross inefficiency associated with a direct retarget of the 4K block RAM to the 18K RAM.

Conclusion

In general, design retargeting can happen without much planning or modification of the design. To get the best use of the underlying innovations of the Virtex-5 FPGA architecture, however, some additional considerations should be taken into account to get the best density, performance, and power for a given design.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/31/07	1.0	Initial Xilinx release.

Notice of Disclaimer