WP248 (v1.0) 2007 年 7 月 31 日

移植到 Virtex–5 FPGA 的指南

作者： Brian Philofsky

当从以前的设计向 Virtex™–5 FPGA 平台进行代码移植或重定向时，需要考虑一些事项。本白皮书将确定并详述适合移植的指南。

Virtex–5 器件选择和重定向后分析

由于 Virtex–5 器件的基础架构与以往的 FPGA 器件不同，因此，要为特定设计选择合适的 Virtex–5 器件并非易事。大多数情况下，设计应采用类似的阵列大小（器件数量）并且比以前的目标器件至少低一个速度级别（如从中速级别到慢速级别）。但是，这种建议对于有些情况却并不适用。本节将介绍一些会影响 Virtex–5 FPGA 器件选择标准的设计风格和特征。

多寄存器设计

Virtex–5 器件名称（如 XC5VSX35T、XC5VLX85）是以每个 FPGA 中的可用逻辑相关数值为基础进行选择的。由于采用新的 6 输入 LUT，Virtex–5 器件的每个寄存器都具有更多的逻辑，因此，特定编号的 Virtex–5 器件拥有的寄存器数量比同一编号的 Virtex–4 器件大约少 30%。

例如，XC4VSX35 有 30,720 个 slice 寄存器，而 XC5VSX35T 只有 21,760 个。另一个例子是 XC4VLX80，它有 71,680 个 slice 寄存器，而 XC5VLX85 只有 51,840 个。对于采用不到可用 slice 寄存器的 60% 的设计，每个器件在 slice 寄存器总数方面的差别并不是问题。但是，当采用超过 slice 寄存器的 60% 的 Virtex–4 器件设计被重定向为 Virtex–5 器件设计时，可能需要更大型的 Virtex–5 器件。

精巧的旧架构

如果采用 Virtex–4 器件的设计能够高速（如 300MHz 或以上）运行，则使用 Virtex–5 器件时，该设计的性能不会有明显提升，LUT 利用率也不会降低。这是因为针对 4 输入 LUT 结构进行了优化的快速设计（同步对象之间具有很低的扇入逻辑和很少的逻辑级数）更可能形成 4 输入 LUT 到 6 输入 LUT 的一对一映射关系。这就意味着新的 Virtex–5 器件逻辑架构也很难降低 LUT 或逻辑利用率和提升性能。

经过多次实践发现，从 Virtex–5 器件内更大的 LUT 结构中受益最多的重定向设计，是那些以前运行速度相对较慢、带有多个逻辑级数和更大的 LUT– 寄存器比率的设计。在这些带有大扇入逻辑锥的设计中，6 输入 LUT 在减少逻辑级数数量以及构建逻辑功能所需的 LUT 数量上具有更大的潜力。当重定向为 Virtex–5 器件架构时，此类设计的 LUT 利用率会大大降低，性能也会大大提高。

在 Virtex–5 器件中获得更高的逻辑利用率

某些类型的逻辑功能可从 6 输入 LUT 中获得更多益处。例如，32 位 XOR 门需要 7 个 6 输入 LUT（有一些备用逻辑），而同样的功能则需要 11 个 4 输入 LUT（没有备用逻辑）。这说明所需的 LUT 数量减少了 36%。

但是，2:1 MUX 映射到 4 输入 LUT 中的方式与映射到 6 输入 LUT 的方式完全相同，因此采用 6 输入 LUT 就没有什么优势。一个 2 输入加法器的每位加法需要一个 4 输入 LUT 和一个 6 输入 LUT。较之以前的器件，软乘法器并没怎么从 Virtex–5 器件的逻辑结构中受益。因此，某些采用 MUX、加法器、软乘法器以及其他逻辑功能的设计并没有在 Virtex–5 器件上实现更高的利用率或性能。例如，DSP 设计一般将这些逻辑功能和大量流水线用作内核来完成大部分操作，因此，它们没有从 Virtex–5 器件的架构中得到太多益处。而某些嵌入式处理器的设计则从 Virtex–5 器件中得到了更多益处，因为它们：

- 采用更少的寄存器（更少的流水线）
- 具有更多的扇入功能
- 采用更少的、在 6 输入 LUT 映射方面没有改进的功能

软件算法

当前的软件算法是为提供尽可能最高的性能而设计的，通常以牺牲面积或 LUT 数量为代价来实现。但是，随着软件越来越成熟，软件算法不断提高性能，同时也越来多地关注面积问题。因此，现在非常需要重定向现有的设计以提高性能。

从短期看，我们可以采取一些措施来改进有足够时间余度的设计的面积，但要牺牲其性能。首先，可以强加与实际时序目标相关的综合时序约束。这样一来，综合软件就可以采用节省空间的算法，同时仍然能够保持其性能。在没有时序约束的情况下，综合工具只能优化时序设计的各个部分，通常以牺牲面积为代价。其次，可以在 ISE™ 软件的映射部分范围内改变 LUT 和 slice 打包方式。该软件的 9.1i 版包含一个名为 **-lc** 的隐藏开关选项，它采用 LUT 压缩算法来减少设计所需的 LUT 数量。**-lc** 开关接受两个自变量：**auto** 和 **area**。当设置为 **-lc auto** 时，该软件会试图整合 LUT，而对时序没有太大影响。当设置为 **-lc area** 时，LUT 打包对时序会有较大影响。该算法可以用于提高在性能和功率上有改进空间的设计的 LUT 利用率。但是，我们不建议将该算法用于很难满足性能或功率预算的设计。另一种可以减少所需 LUT 数量的方法是 slice 压缩。这种方法通过在映射部分设置 **-c 1** 开关来实现，其采用了 LUT 和 slice 压缩，从而减少了需要的 slice 数量。但是，这通常都要以牺牲性能和功率为代价，可能比 **-lc area** 开关更严重。

采用旧核、EDIF 或 NGC 网表

强烈建议在 Virtex–5 器件中实现之前，重新生成或重新综合设计中所有旧核或黑匣子网表。大多数针对旧的 Spartan™ 系列和 Virtex 器件架构的网表在用于 Virtex–5 器件时都不会出现任何误差。但是，几乎在所有情况下，此类网表都会使得设计变得更慢和更大（需要更多逻辑级数和利用更多资源）。这要归因于基础架构（如，4 输入 LUT 和 6 输入 LUT）的不同，以及不同架构的时序和优化方面的差异。重定向这些黑匣子实例可确保它们针对 Virtex–5 器件架构进行了优化。

当重新综合无法进行的时候，另一种可选方法是在映射阶段采用全局优化（**-global_opt switch**）。虽然全局优化通常并不像直接重新综合黑匣子实例那样有效，但它允许工具重新综合或重组网表，从而实现更好的终端设计。使用全局优化的主要不足之处是延长了运行时间并且可能加重对内部逻辑通路的影响（重命名和重组），这会使设计调试变得更加困难。

采用物理约束

在重定向为 Virtex–5 FPGA 之前，应排除嵌入到现有设计的代码或网表中的所有 LOC、RLOC、BEL 约束或其它物理约束。对于以前的架构来说是最佳布局，而对于 Virtex–5 FPGA 来说可能就并非如此，因为它们在 slice 结构、CLB、RAM（LUT RAM 或 Block RAM）、逻辑、I/O 布局和时序上都存在差异。某些情况下，由于布局和坐标差异，可能会出现误差。但是，即使不出现误差，时序、密度和功率也有可能是次优的，除非物理约束被消除或针对新架构进行了更新。

采用控制信号

采用控制信号（控制时钟、置位、复位及时钟使能之类的同步单元的信号）会影响器件利用率。这对几乎所有 FPGA 技术都是如此。但是，Virtex–5 器件的结构差异使得选择和使用控制信号时的考虑事项发生了变化。本节将举例说明实现最佳器件利用率所需考虑的事项。

将推断寄存器初始化为已知逻辑值

当确定信号或寄存器时，所有推断寄存器都应被初始化为一个已知值。这可以实现更佳的器件利用率、性能和功率，并改进设计的验证。

限制使用低有效控制信号

出于以下原因，我们不建议采用带有低有效控制信号的推断元件或例示元件：

- Virtex–5 器件的粗粒 slice 成分
- Virtex–5 器件内 slice 控制信号上没有可编程逆变单元
- 层次化设计方法不允许跨越层次边界进行优化

某些情况下，由于将 LUT 用作逆变器，并且有时低有效控制信号还会对寄存器的打包带来额外限制，因此，器件利用率会下降。采用低有效控制信号还会对时序造成影响。可能的话，HDL 代码或例示元件内都应采用高态有效控制信号。当控制信号的极性无法在设计（如当其由外部、不可编程源驱动时）中进行控制时，应对代码最上层的信号进行转化，并且规定由逆变器驱动以得到相同极性（功能）的高态有效控制信号。

限制使用低扇出控制信号

不应对设计中的独特控制信号的数量进行分组或限制。而且，在设计中不应对低扇出时钟使能、置位或复位进行编码，原因如 “采用旧核、EDIF 或 NGC 网表” 中所述。就寄存器和 LUT RAM 而言，独特的低扇出控制信号会使得 slice 不能被充分利用，并且会对布局和时序造成负面影响。除非设计功能要求，否则不应在代码内实现置位、复位或时钟使能功能。

无需使用置位或复位

代码中不必要的置位或复位会阻止 SRL、RAM（LUT RAM 或 Block RAM）及其他逻辑结构的推理。要从该架构中获得最佳效率，只有在设计功能需要时才应对置位或复位执行编码操作。不需要时，不应对其进行编码操作。例如，当只用于寄存器初始化时，不需要复位，因为配置一完成，就会自动发生寄存器初始化。另一个例子是，当一个电路长时间保持空闲状态时，对输入寄存器进行简单复位最终会导致剩余电路上的数据丢失。类似的例子还有保持了数个时钟周期的复位状态的内部寄存器。这种情况下，复位过程中内部寄存器的数据会丢失，因此，无需复位。通过减少使用不必要的置位或复位可以提高器件利用率和性能。

DSP48E Slice 寄存器中的乘法器或加法器 / 减法器的置位

DSP48E slice 寄存器只包含复位，不包含置位。出于这一原因，除非有必要，否则不应在 DSP48E 中的乘法器、加法器、计算器或其他逻辑周围进行置位（其值等于施加信号值）编码。

使用同步置位 / 复位

如果置位或复位对电路的正常运行是必要的，则要对同步复位进行编码。同步置位 / 复位不仅提高了时序性能和稳定性，而且还在 FPGA 内实现了更小和更好的利用。同步置位 / 复位可实现更少的逻辑（更少 LUT），更少的打包约束，而且通常会使电路运行速度更快。如果无需将现有异步复位重新编码为同步复位，则可通过利用综合工具里的 “**treat asynchronous resets as synchronous（将异步复位用作同步复位）**” 开关，将异步复位视为同步复位。如果采用的综合工具是 Xilinx 综合技术（Xilinx Synthesis Technology，XST），则可以在 GUI 上找到这个开关。如果采用 Synplify 综合软件，该功能则以 TCL 命令的形式出现。采用同步复位对于减少资源和提高性能而言并不像重新编码一样有效。但是，它可以实现一些寄存器打包，这对于其他方法来说是不可能的。

采用时钟使能

当采用高扇出时钟使能时，不应手动分离或复制它们，而应作为单个时钟使能进行编码。如果由于时序或其他原因而需要复制，则应在综合工具中控制时钟使能。另一种方法是使用全局缓冲器（BUFG）来分配高扇出信号。但要小心操作，特别是当时钟使能源由全局时钟引脚以外的外部引脚驱动的时候。

控制集的分析和使用

如果因为使用独特的控制集（时钟、时钟使能、置位和复位）而有必要对设计进行分析，则 Map .mrp 报告会给出每个控制集的详细情况。要查看该控制集，则在映射过程中应启用 –detail 开关。这将丰富该报告第 13 部分的控制集信息。

如果如报告所述，大量独特的低扇出控制信号会使寄存器的利用率降低，那么可以在综合工具中采取措施来控制设计任何部分（如特定网或体系）或整个设计的控制信号的推断。禁用寄存器中的专用控制信号会实现更高的逻辑利用率或更高的逻辑打包。对于寄存器密集型设计而言，平衡这些综合控制的使用以使寄存器打包问题不会转变为 LUT 资源限制问题是非常重要的。

要在 XST 应用中控制时钟使能，应在寄存器信号、模块或整个设计上使用 `use_clock_enables = no`。在 Synplify 软件中，可以利用 `syn_useenables = 0` 属性来在推断寄存器上实现该控制。也可以采用 TCL 命令，通过以下结构，按组、体系或全局来控制时钟使能：

```
# 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 考虑事项

要最大限度地利用 Virtex–5 器件架构中的 Block RAM 和 LUT，当通过推断、例示基元或 CORE Generator™ 软件重定向 Block RAM 和 LUT RAM 时必须了解某些考虑事项。如果 CORE Generator 软件被用于生成 RAM，应为 Virtex–5 器件重新生成内核，或对 RAM 重新编码，以便进行正确的综合推理。这两种方法通常都会带来较好的利用率和性能。但是，我们建议推测存储器（可能的话）来增进对代码、仿真以及未来代码可移植性的了解。

例示 RAM

本节中的建议适用于当 RAM 基元在设计中被例示或当无法为 Virtex–5 器件重新生成 CORE Generator 软件 IP 时的情况。这些建议也可由推断 RAM 的代码实现，特别是当采用综合属性来确定使用哪些 RAM 资源（如 `syn_ramstyle = blockram`）时。这些建议按 RAM 深度划分。RAM 深度通常是在确定使用哪些 RAM 资源时需要考虑的最主要因素。

深度大于 16 位但小于或等于 256 位

LUT 越大，Virtex–5 FPGA 中的 LUT RAM 就越深，在 Block RAM 和 LUT RAM 之间选择的标准与以前的 FPGA 器件不同。一般情况下，LUT RAM 用于 64 位或更小的存储器，除非某个目标器件缺少逻辑资源 (LUT) 和 / 或 SLICEM。不管数据宽度是多少，面向 64 位或更小的存储器的 LUT RAM 在资源、性能和功率上都更有效。对于深度大于 64 位但小于或等于 256 位的存储器，要确定使用哪种最佳资源取决于以下几个因素。第一，是否提供额外的 Block RAM? 如果答案是 “否”，则应采用 LUT RAM。第二，有哪些延迟要求? 如果需要异步读取能力，则必须采用 LUT RAM。第三，数据宽度是多少? 宽度大于 16 位应采用 Block RAM （ 如果有 ）。最后，有哪些性能要求? 与 Block RAM 相比，寄存器 LUT RAM 通常具有更短的时钟 – 输出延迟和更少的布局约束。

如果设计已经包含深度大于 16 位的例示 LUT RAM，则应采用更深的基元 （ 例如，RAM32X1S, RAM64X1S 等 ）。RAM16X1S 与 MUXF5 或其他逻辑一起使用不会被正确地重定向为自动使用深度更大的 LUT。在这种情况下，应修改代码以正确使用深度更大的基元。

512 位深度

当规定深度为 512 位时，如果两个端口宽度都为 18 位或更小，则以前架构中的例示 Block RAM 就可以有效地重定向。如果将其中一个或两个端口的数据宽度设为 36 位，Block RAM 通常可以有效地重定向。但是，这种情况需要考虑一些特殊事项。深度为 512 x36 位的 Block RAM 需要考虑这三种情况。

真正的双端口

如果 Block RAM 用于真正的双端口模式 （ 例如，双端口 RAM，其中一个或两个端口需要具有读写能力 ），则应考虑宽度。如果所有端口都需要 18 位或更小的数据，就可以有效地映射到 Virtex–5 器件的 Block RAM 中，虽然要进行一些修改。如果使用 Virtex–II 器件的 RAMB16_Sx_S36 或 Virtex–4 器件的 RAMB16，有一个或多个宽度参数设置为 36，则 Block RAM 需要占用整个 RAMB36 模块才能实现。要进行重定向，则应修改代码以便利用正确设置来推断 Block RAM 或例示 RAMB18 元件。或者，可将 RAMB16_Sx_S36 基元改为 RAMB16_Sx_Sy 基元，其中，x 和 y 都是 18 或更小。对于 Virtex–4 器件，应对源进行修改，以便将所有宽度参数都设为 18 位或更小。这可能是比较容易的途径，但是要实现这种改变却并不那么便捷。

如果真正的双端口 Block RAM 采用的端口宽度大于 18 位，就会占用整个 RAMB36。如果设计占用了所有 Block RAM 资源，找回 Block RAM 丢失部分的唯一方法是对 Block RAM 进行时分多路复用读写操作。时分多路复用操作不仅需要附加逻辑，而且还需要充裕的时序余量。因此，该技术只能用于时钟频率较低 （ 小于 150MHz ）、在 Block RAM 进 / 出通路上只有几纳秒时序余量的 Block RAM。如果时序余量不存在，则很可能要牺牲 Block RAM 的位数。

简易双端口

Virtex–5 器件的 Block RAM 可以有效地用于简易双端口模式（例如，一个端口只读，另一个端口只写）。但是，在某些重定向情况下，一个完整的 RAMB36 模块可以替代以前架构中使用的 RAMB16 模块。对于较老的 Virtex–II 器件的 Block RAM 基元（例如，RAMB16_S36_S36），9.2i 版以前的 ISE 软件无法确定是否将 Block RAM 用于简易双端口模式，因此，需将 Block RAM 映射到整个 RAMB36 中。将 RAMB16_S36_Sx RAM 替换为 RAMB18SDP 元件或适当的推断 Block RAM 或升级为最新的 ISE 软件版本即可找回失去的一半 Block RAM。同样，如果 Virtex–4 器件的 RAMB16 元件用于 9.2i 版以前的 ISE 软件，通过分析宽度属性，该软件就可以确定是否将 RAMB16 元件用于简易双端口。如果一个端口的 WRITE_WIDTH= 0，另一个端口的 READ_WIDTH =0，该软件就可以将 Block RAM 元件重新映射到 RAMB18SDP 元件中并充分利用 Block RAM。但是，如果端口宽度为非零值并且至少有一个宽度设为 36，那么整个 RAMB36 模块就可以用于映射 RAM 功能。出于这一原因，应对所有的 RAMB16 例示进行研究，以确定是否将它们用于简易双端口模式，如果是这样，所有未使用的端口都应将其宽度参数值设为 0。ISE 软件 9.2i 版及以后的版本都可以从其连接的方式来确定是否将 Block RAM 用于简易双端口模式，并在多数情况下采用适当的 Virtex–5 器件的 Block RAM 基元。

如果将 Block RAM 用于简易双端口模式，但是读写操作具有不同的数据宽度，其中一个数据宽度为 36，则可以从两者中选择一个。第一个选择是什么都不做；因此，需要占用整个 RAMB36 模块以实现该 Block RAM，而无需使用额外资源。第二个选择是例示 RAMB18SDP 元件，但必须添加额外的逻辑（多路数据和寻址逻辑）方可将静态 36 位端口宽度改为适当的尺寸。这个选择不仅要占用额外的逻辑（LUT），而且还会影响 Block RAM 的性能。因此，这两种实现方式的选择不仅取决于可用的资源，而且还取决于该 Block RAM 的性能要求。

单端口

如果 Block RAM 被用作单端口 RAM，并将输入和 / 或输出数据宽度设为 36 位，就可以在 Virtex–5 器件中有效实现之，但可能需要修改代码。如果例示为 RAMB16_S36（36 位单端口模式 RAMB16），Block RAM 应为例示的 RAMB18 元件所取代，其中，两个端口的读和写数据宽度都设为 18 位。两个端口的 CLK 引脚与同一个时钟源连接。如果需要单个（非字节使能）写使能 (WE)，则所有 WE 引脚都应与同一个源连接起来。ADDR 引脚应连接起来，以便将九个地址引脚与 Block RAM 的 ADDRA[14:6] 和 ADDRB[14:6] 连接起来。ADDRA[5] 引脚与逻辑 0 连接，ADDRB[5] 与逻辑 1 连接，从而在 RAM 的两个端口之间分配 1K 的地址空间。然后，36 位数据应与两个端口的输入和输出数据引脚连接，下面的 18 位位于端口 A 上，而上面的 18 位则位于端口 B 上。当采用这种方式连接时，Block RAM 就会高效地映射到 Virtex–5 器件中。

大于 16K 的深度

如果构建一个深度大于 16K 的 Block RAM，这个深度更大的 Block RAM 存储器可生成效率和性能都更高的 Block RAM。如果这个深度更大的 Block RAM 用两个级联 Virtex–4 器件的 RAMB16 基元定义，软件就能够自动地将 Block RAM 重定向到 RAMB36 元件。但是，如果这个深度更大的 Block RAM 采用以前的 RAMB16_S1 基元描述和 / 或需要 64K 的深度，则 Block RAM 应被重新编码为推断代码，或者应对 RAMB36 或一对级联 RAMB36 进行例示以构建合适的 RAM 结构。这充分利用了 Virtex–5 器件更深的 Block RAM 结构。

对推断 Block RAM 的当前限制

大多数推断 Block RAM 的实现对于 Virtex–5 FPGA 架构都是最理想的。本节将讲述在采用 Synplify 8.8 或 XST 9.1i 软件对 Block RAM 进行推断时受到的限制。

512 x 36 单端口 RAM

对于 XST 和 Synplify 软件，在 RAMB36 中实现数据宽度大于 18 位但小于或等于 36 位、深度为 512 位或更小的单端口 Block RAM（实际上还可以在 RAMB18 中实现）。在校正之后，采用这种类型和尺寸的 Block RAM 时，才可以采用 “单端口”例示方法。

简易双端口 Block RAM

Synplify 软件无法为数据宽度大于 18 位但小于或等于 36 位、深度为 512 位的简易双端口模块正确推断 RAMB18SDP。应参考最新版本的工具文件，以确定这一限制是否仍然存在。但是，更宽的 RAMB36SDP（大于 36 位但小于 72 位）的推断进行得很顺利。

具有不同的读、写或端口数据宽度值的 Block RAM

XST 和 Synplify 无法推断带有不同数据高宽比的 Block RAM，无论是同一端口的读和写，还是端口 A 和 B 上的不同宽度。必须对此类 Block RAM 进行例示以便融入设计。

Block RAM 深度不是 2 的整数幂

如果 Block RAM 的深度没有规定为或不需要达到 2 的整数幂（例如，512、1024，2048 等），CORE Generator 软件有时可以实现更小的 Block RAM 设计。

利用 Block RAM 实现最佳器件利用率的其他考虑事项

要从 Virtex–5 器件的 Block RAM 结构中得到最多益处，需要了解其他方面的考虑事项，特别是在该设计以前面向比 Virtex–4 器件更老的技术。

输出寄存器

Virtex–4 器件包含一个用于 Block RAM 的输出寄存器。但是，在 Virtex–4 器件架构以前创建的设计和许多其后创建的设计都没有采用该特性。采用输出寄存器可大大提高 Block RAM 的性能（时钟 – 输出），同时还会提高功率和器件利用率。如果一个设计被从 Virtex–4 之前的架构移植到 Virtex–5 器件中，则应重新检查代码，以便弄清输出寄存器是否可以融入设计中。

字节宽写使能

Virtex–4 器件还具有字节宽写使能功能。这一特性对 Block RAM 的存取和器件时用有利。因此，在重定向 Block RAM 时，应了解和实现字节宽写使能功能。

不同的读 / 写高宽比

Virtex–4 和 Virtex–5 器件的 Block RAM 不仅在其两个端口上而且在读写上都具有不同的数据宽度。数据宽度不同会占用更多 Block RAM，并且允许使用未使用全存储器阵列来访问更多存储器的实例。

FIFO 逻辑

如果设计实现了由 CORE Generator 软件创建的或来自推断逻辑的 FIFO，则应该用 Virtex–5 器件中的专用 FIFO 逻辑来替换 FIFO。这种替换方案不仅节省了逻辑资源，而且还简化了 FIFO 逻辑的实现和时序。

ECC 逻辑

RAMB36SDP 的 ECC 功能应用于自检和校正 Block RAM 数据。ECC 逻辑未必会降低利用率。但是，如果这个功能对于设计而言非常重要，则在设计的 Block RAM 重定向过程中应使用这一功能。

其他基元重定向的考虑事项

一些为以前的架构例示的器件基元不会被自动重定向为 Virtex–5 器件架构，或者在重定向后存在明显的差异。

重定向这些基元时要考虑到一些可能的变化。

FIFO16

Virtex–5 器件的标志逻辑（空、满）FIFO 的功能与 FIFO16 逻辑的不同，但 FIFO16 是自动重定向的。如果设计包含 FIFO16，则 FIFO16 应被手动替换为 FIFO18，以确保综合前 RTL 仿真结果与最终设计的功能相匹配。如果未替换 FIFO16，则应对带有自动替换的 FIFO18 的设计进行全功能仿真。这是为了确保 FIFO 标志的功能变化不会对设计造成负面影响。

BUFR

Virtex–5 器件的 BUFR 的延迟与 Virtex–4 器件中的 BUFR 延迟不同。对于大多数带有自由运行时钟的设计而言，这没有任何区别。但是，缓冲器启动时，延迟方面的差异会产生影响。在时钟可以停止和重启缓冲器的情况下，延迟的变化会对设计产生影响。要更新元件的综合前功能以使 Virtex–5 器件具有适当的延迟，则应更新 BUFR 例示，以增加适当的 SIM_DEVICE generic （使用 VHDL 语言）或参数（使用 Verilog 语言）。如需正确的例示模板，请参考 ISE 软件语言模板。不管 SIM_DEVICE 的设置如何，实现后门级仿真利用 SIM_DEVICE 属性的适当值进行更新，门级仿真应反映正确的延迟功能。出于这一原因，我们强烈建议进行门级功能仿真，以确保功能不受延迟差异的影响。

DCM_ADV

大多数情况下，很明显是针对 Virtex–5 器件而重定向 DCM_ADV 的。但是，在使用动态重配置端口 (DRP) 的情况下，地址映射从 Virtex–4 变为 Virtex–5 器件。如果 DCM 未使用 DRP，则无需进行其它修改。但是，对于使用了 DRP 的 DCM 而言，应了解 DCM_ADV 的 DPR 端口的寻址和行为方面的差异，如果有必要，则应修改设计以便消除这些差异。

BUFGMUX

如果设计包含 BUFGMUX，它会自动重定向到 BUFGCTRL。除了元件的延迟差异，该自动重定向与 Virtex–4 FPGA 中的重定向类似。当采用单个自由运行时钟时，延迟差异不会影响功能。但是，在停止或切换了时钟的情况下，时序可能会稍有不同。因此，BUFGMUX 应由 BUFGMUX_CTRL 或 BUFGCTRL 替换，并且 / 或者应进行门级时序仿真，以确保延迟差异不会影响设计的最终功能。

MUXCY、XORCY、MULTAND、MUXF5 和 MUXF6

由于基础架构方面的差异，则不会以对面积、密度或时序而言最有效的方式重定向 MUXCY、XORCY、MULTAND、MUXF5 和 MUXF6。应重新评估包含这些例示元件的设计，以确定包含这些元件的代码部分是不是可以推断的或被 Virtex–5 器件架构中的等效结构基元替换。

这些基元是自动重定向的。

PMCD

如果设计包含相位匹配的时钟分频器（PWCD），则它应被手动替换为正确配置的锁相环（PLL）。可以通过使用 PLL Architecture Wizard 创建正确的 PLL 例示来减少工作量。

RAMB4

如果设计包含原来的 Virtex 器件或 Spartan–II 系列中的 RAMB4 基元，那些 Block RAM 基元就不会被自动重定向；必须手动地将其改为推断代码（优先）或 Virtex–5 器件支持的例示基元。不支持该重定向的主要原因是将 4K Block RAM 直接重定向到 18K RAM 的总体效率不佳。

结论

总体上而言，无需对设计进行太多规划或修改即可实现设计移植。要充分利用 Virtex–5 FPGA 架构的基本创新，应考虑其它的一些因素，以便为特定设计实现最佳密度、性能和功率。

修订历史

本技术文档的修订历史如下表所示。

日期	版本	修订
2007 年 7 月 31 日	1.0	Xilinx 最初版本。

Notice of Disclaimer