XILINX®

Improving XC4000 Design Performance

Summary

This Application Note describes XC4000 architectural features that can be exploited in high-performance designs, and software techniques that improve placement, routing and timing. It also contains information necessary for advanced design techniques, such as floor planning, locking down I/Os, and critical path optimization.

LCA Family

XC4000

Demonstrates

High-performance XC4000 design

Introduction

Designers sometimes assume that the Xilinx FPGA architecture is a gate-array-like sea-of-gates and that, consequently, little or no architectural consideration is required during design. This approach is valid and is supported by XMAKE, the Xilinx fully-automated design procedure. It can, however, lead to inefficient designs.

The Xilinx FPGA architecture is very regular and gate-array-like, but it is not a simple sea-of-gates. An understanding of the architecture and the resources it provides can make designs become more efficient in both speed and density. This Application Note focuses on the features of the XC4000 architecture and its supporting software that improve design efficiency. It also describes advanced design techniques that extract the maximum performance from the architecture.

Some techniques described in this Application Note relate specifically to XACT v1.42. In a subsequent version, XACT 5, Hard Macros will be replaced by Relationally Placed Macros, and the operation of XACT Performance will change significantly.

XC4000 Architectural Features

XC4000 CLB Overview
The XC4000 CLB is shown in Figure 1. Key features are the three function generators and the two flip-flops. Unlike previous LCA devices, the F and G function generators do not share inputs, permitting them to operate totally independently, if required. The H function generator combines the F and G outputs with an additional H1 input.

The F-G-H combination can implement any function of five inputs. In addition, some functions of more inputs can also be implemented. Some functions of five inputs can be implemented using just an F-H or G-H combination.

The two flip-flops can store the function-generator outputs or a signal coming in on the DIN pin. If the H function generator is not in use, the H1 input can pass through the function generator and provide a second direct input to the other flip-flop. Since separate pairs of output pins are provided for the function generators and the flip-flops, the F and G function generators and the two flip-flops can operate independently.

Fast Carry Logic
In addition to implementing logic and providing storage, the XC4000 CLB contains dedicated hardware to accelerate the carry path of adders and counters, Figure 2. Using this feature, adders and counters are very fast and efficient, consuming a minimum number of CLBs.

While dedicated logic and interconnect are used to optimize the carry path, function generators are used to form sums from the operands and carries. In this way, two bits of arithmetic (or one bit, if so desired) can be implemented in each CLB. Since the dedicated carry logic can be configured in approximately 40 different ways, CLBs can be concatenated into a variety of arithmetic functions. The carry propagates either up or down a column of CLBs.

The dedicated carry logic is accessible via hard macros. Carry-logic hard macros are available in the Xilinx library, or may be defined by the user with a program called HMGEN*. The HMGEN package includes documentation which describes how to use both the HMGEN program and the dedicated carry logic.

For additional information on the dedicated carry logic see the Xilinx Application Notes Using the Dedicated Carry Logic in the XC4000 (XAPP 013) and Estimating the Performance of XC4000 Adders and Counters (XAPP 018).

*HMGEN is available free of charge from Xilinx. Contact the Xilinx Technical Support Hotline at 1-800-255-7778
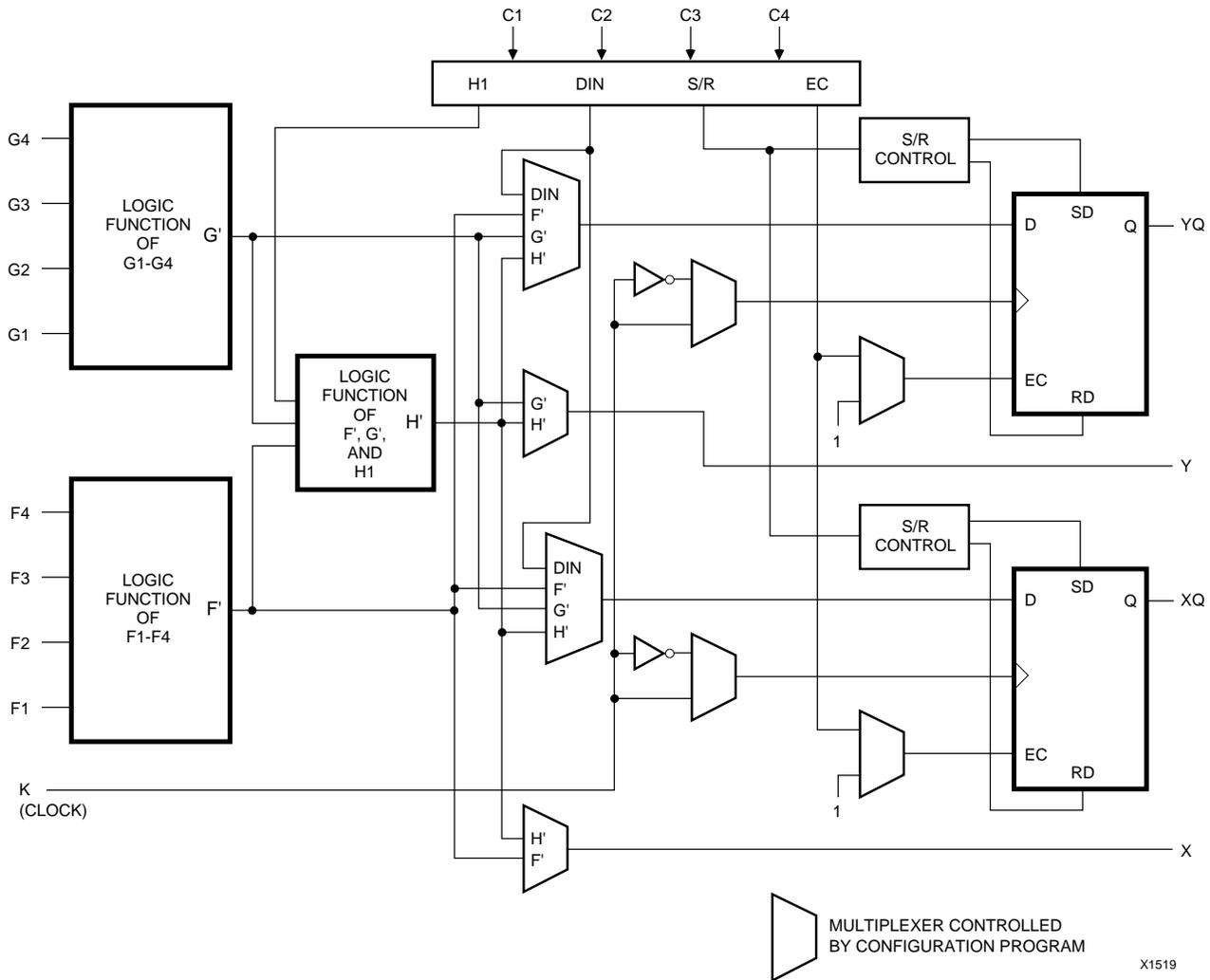
Figure 1. XC4000 Configurable Logic Block

On-Chip RAM

The XC4000 on-chip RAM significantly reduces the cost of data storage. Using this feature, up-to-64 bits of data can stored in a single CLB that otherwise could only store two bits in its flip-flops.

Any CLB can be configured as a RAM. In the RAM mode, the F- and G-function-generator look-up tables become writable, Figure 3. For a 16 x 2-bit RAM, the F and G function generators are used separately. For a 32 x 1-bit RAM, they are combined in the H function generator.

The F- and G-input pins act as memory address lines, just as they do in the non-RAM mode. Other CLB control pins, however, are redefined. For a 16 x 2-bit RAM, DIN and H1 become the two data inputs, while in the 32 x 1-bit configuration, DIN is the single data input, and D1 the fifth address bit. In both cases, S/R is the Write Enable (WE) input.

In the 16 x 2-bit mode, read data is available at the F- and G-function-generator outputs; in the 32 x 1-bit mode, it is

available at the H-function-generator output. Just as in the non-RAM mode, these function-generator outputs drive the X and Y output pins, or they can be registered in the flip-flops.

Some non-RAM functionality remains when CLBs are configured as RAM. In the 16 x 2-bit mode, the H function generator can be used to implement any function of the two RAM outputs. In the 32 x 1-bit mode, both the write and the read data can be captured in the flip-flops, since the flip-flops have access to the RAM input data on the DIN pin.

The XC4000 RAM function is extremely fast compared to monolithic SRAM devices that often have cycle times of 55 ns or longer. In those slower devices, 1 ns glitches in control signals can be tolerated. This is not the case in the XC4000 RAM, however, where cycle times are less than 10 ns. WE pulses as short as 1 ns are easily recognized, and good control-circuit design is essential.
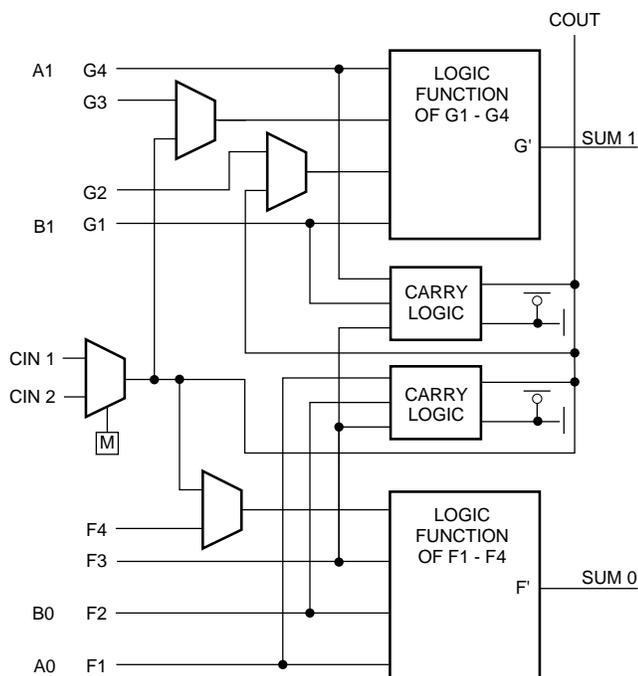
Figure 2. XC4000 Fast Carry Logic in Each CLB

Figure 3.   XC4000 CLB RAM Mode

Designing with the XC4000 SRAM is similar to designing with very fast monolithic SRAMs (<25 ns cycle time). Many factors, such as interconnect delays, that can safely be ignored in slower monolithic SRAM designs become critical. For a discussion of XC4000 RAM design, please refer to the Xilinx Application Notes Using the XC4000 RAM Capability (XAPP 031) and High-Speed RAM design in XC4000 (XAPP 042).

3-State Buffers
To facilitate on-chip multiplexed busses, the XC4000 architecture includes Longlines that can be driven by three-state buffers (TBUFs), Figure 4. Two TBUFs, located adjacent to each CLB, drive the horizontal Longlines immediately above and below the CLB.

The TBUF data inputs can easily be driven from the outputs of the associated CLB, but can also come from elsewhere. While any signal can be used to enable the TBUFs, if an enable is routed on a vertical Longline, it can be used to select a column-wise function to drive a horizontal bus.

Additional TBUFs are located near the Input/Output blocks (IOBs) on the left- and right-hand edges of the array. These permit IOBs to be included in a multiplexed bus, thus possibly extending an external bi-directional bus onto the chip.

TBUF Longlines can also be used to implement wired-AND functions. Optional resistive pull-ups at each end of a Longline cause it to go High while not being driven. Enabling any TBUF that has a logic Low at its input will cause the line to go Low, thus creating a wired-AND of the enable signals.

The TBUF Longlines are valuable routing resources, and should be used sparingly. Multiplexers and AND-gates with 16 or fewer inputs can often be implemented more efficiently using CLBs, thereby conserving Longline resources. TBUFs are rarely appropriate for multiplexers with four or fewer inputs.

Global Clock Buffers
Eight global nets run through XC4000 devices, potentially reaching every CLB, Figure 5. These global nets are optimized for the distribution of clocks and other time-critical or high-fanout signals.

Four of the eight are primary global nets that offer minimum delay and negligible skew. The other four are secondary global nets. Due to heavier loading, the secondary global nets introduce a slightly longer delay, about 1 ns more, and some additional skew.

Primary and secondary global nets are driven by BUFGP and BUFGS buffers, respectively, both of which can connect directly to pads. The use of a global net is specified in the schematic by using the appropriate buffer to drive the desired signal. BUFGP and BUFGS symbols are in the Xilinx library.

Switch matrices at the center of each column connect the eight global buffers to four vertical lines used to distribute global signals within the column. Figure 6 shows one of the switch matrices. Each vertical line can be driven by only one BUFGP, and each BUFGP drives a different vertical line. The BUFGSs, on the other hand, can each drive any or all vertical lines.

Consequently, BUFGSs are much more flexible in their routing. This flexibility is particularly valuable when routing
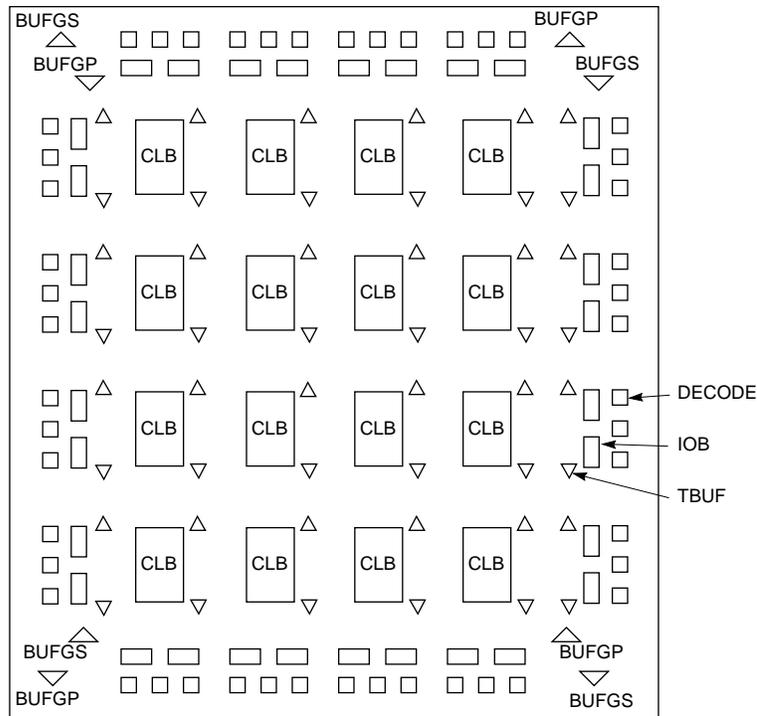
X5258

Figure 4. XC4000 TBUF Organization

non-clock global signals, since the vertical lines have limited connections to non-clock CLB pins. Being able to drive any, or even multiple vertical lines from a single buffer is a tremendous advantage.

The routing of non-clock global signals also benefits by using BUFGSs for clock distribution. A BUFGP would constrain the clock routing to the same vertical line in every column. With a BUFGS, however, any vertical line can be used for the clock, possibly freeing a critical line for non-clock routing. ***Whenever timing and skew requirements permit, BUFGSs should be used for distributing global signals.***

The number of clocks in a system should be minimized. Since only four clocks can be made available in a column of CLBs, a large number of clocks imposes considerable constraints on CLB placement. In particular, gated clocks should be avoided, unless absolutely necessary. In addition to consuming global nets, the gating logic introduces uncontrolled clock skew and the potential for clock glitches, both of which can cause a system to malfunction.

It is better to use a minimal number of clocks and disable the flip-flops when clocking is not required. Clock-enable signals can often be routed on local interconnect or regular Longlines, since they are not skew-critical.

Wide Decoders

Sometimes it is necessary to decode specific values from a large number of bits, e.g., when decoding a specific microprocessor memory address. To facilitate such decoding, dedicated wide-decoder functions are provided along each edge of XC4000 devices, Figure 7. These wide decoders are separate from the CLBs, and do not consume CLB resources.
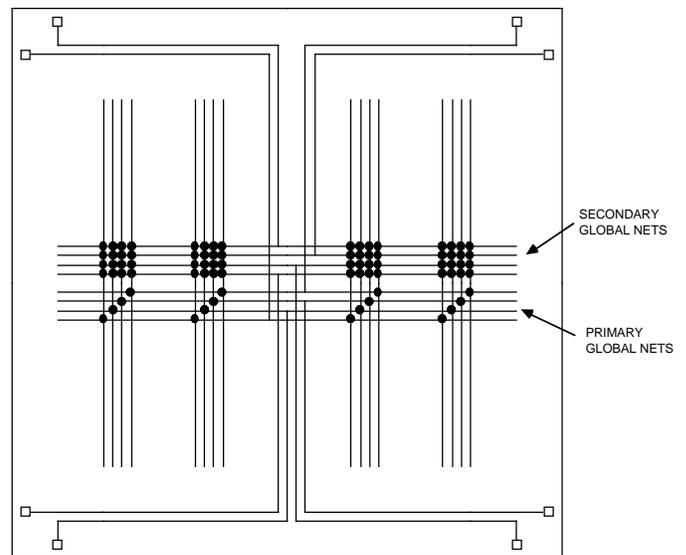
Figure 5. XC4000 Global Net Distribution

To CLB
Column

PIPs

From
Secondary
Buffer

From
Primary
Buffer
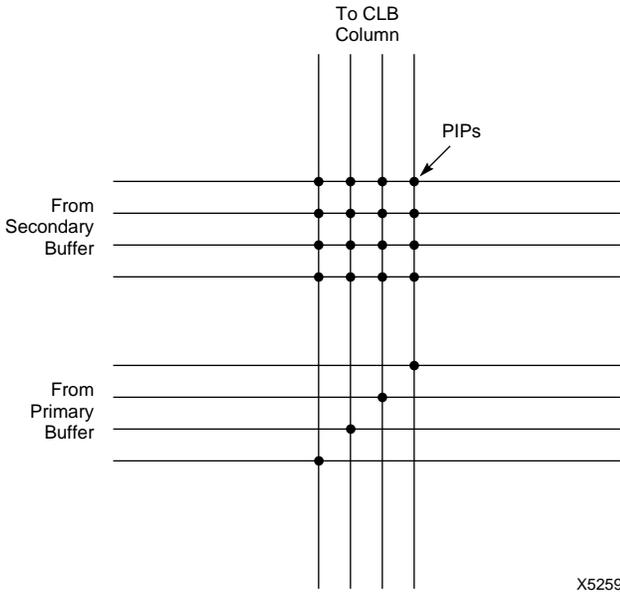
X5259

Figure 6. XC4000 Global Net Interconnection Matrix

On each edge of the chip, there are four decoders that share a common set of inputs. Potentially, there are three decoder inputs for each row or column of CLBs, one each from two adjacent IOBs and one from local interconnect. While the decoders share inputs, it is possible to decode different sets of bits on the same edge, since all inputs need not be used in every decoder. It is thus even possible to decode disjoint sets of bits on the same edge.

An XC4000, for example, has a 20 x 20 array of CLBs, and each edge has four wide decoders sharing 60 inputs. The decoders on one edge could decode a specific byte address and a specific word address from a 32-bit microprocessor bus, and, at the same time, decode two specific values from an internal 16-bit bus.

The wide decoders are implemented as wired-AND-gates. Resistive pull-ups at each end cause the output to go High

when all inputs meet their specified condition. Any input to the decoder that fails to meet its specified condition (High or Low) causes the output to be pulled Low.

Each decoder can be split at its center to make two half-sized decoders. There are, therefore, up to 32 decoders available. Dedicated decoders should only be used to decode ten or more inputs, since nine or fewer inputs can be decoded more efficiently using a single CLB.

Note that XC4000A devices have two wide decoders per edge, instead of the four found in non-A devices. The use of wide decoders is specified in the schematic by using the symbols DECODE4, DECODE 8, etc.

IOB Registers
The latches and registers located in the IOBs are an often-overlooked resource, Figure 8. They are ideal for synchronizing input and output signals, and can also provide additional storage for internal signals. Using IOB registers can significantly reduce CLB flip-flop utilization, and can also reduce routing congestion.

The use of IOB registers or latches is specified during design entry. They are represented by the schematic symbols OUTFF, INFF and INLAT.

Master Set/Reset.
XC4000 devices contain a global-set/reset (GSR) line. When GSR is asserted, every flip-flop in the LCA device is simultaneously set or reset. No general-purpose routing resources are consumed, however, since the GSR has its own dedicated routing. Setting or resetting all the flip-flops in this way is far more efficient that using the RD pins on individual CLBs.

Each flip-flop is either set or reset according to an attribute attached to it in the schematic. The default value for the

INTERCONNECT

IOB
.I1

IOB
.I1

A   C   B

$(\quad\quad C)$ .....
$(A \bullet B \bullet \overline{C})$ .....
$(A \quad \overline{B} \quad C)$ .....
$(\overline{A} \quad B \quad C)$ .....

X2627

Figure 7. **XC4000 Wide Decoder**
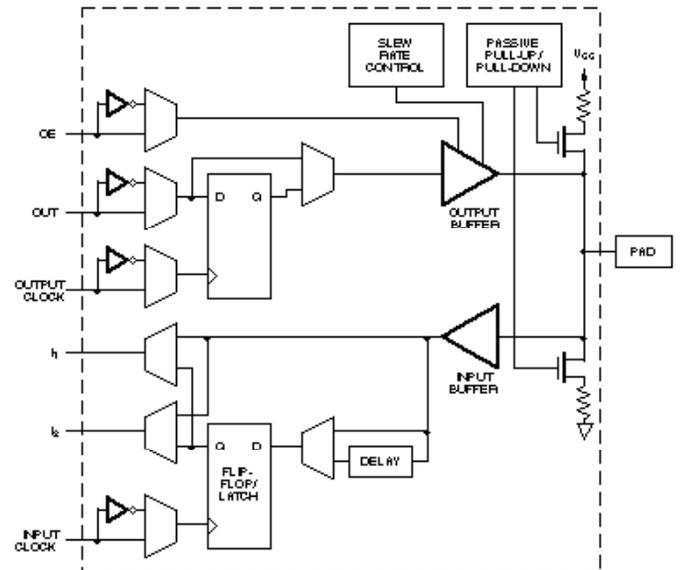
Figure 8. **XC4000 Input/Output Block**

X5260
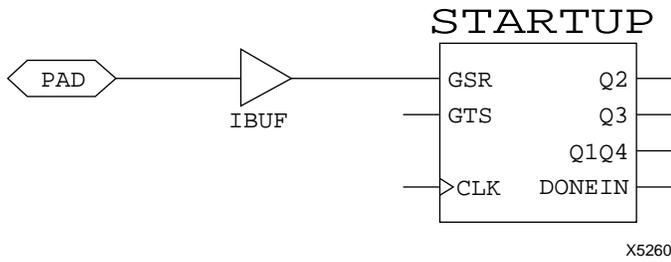
Figure 9. Schematic Symbols for Global Set/Reset

attribute is INIT=R, which causes GSR to reset the flip-flop. Changing the attribute to INIT=S causes the flip-flop to be set. The INIT attribute also determines the flip-flop state immediately after power-up.

To use GSR, the STARTUP primitive should be included in the schematic, and a pad connected to its GSR pin, Figure 9. Any user-I/O pin may be used, and, if necessary, the pad can be locked to a specific device pin, just like any other pad. If possible, however, PPR should be allowed to choose the location, since any unnecessary constraint reduces the freedom PPR has to implement the design, and potentially degrades the result.

Boundary-Scan Circuitry

In production, boards must be tested to assure the integrity of both the components and the interconnections. However, as integrated circuits become more complex and multi-layer PC boards become more dense, it is increasingly difficult to test assembled boards.

XC4000 helps solve this problem by providing boundary-scan test facilities. All user-I/O pins are fully testable, and the test protocols are compatible with IEEE Std 1149.1. Boundary scan does not detract from the capacity or capability of XC4000 devices, since it only uses dedicated logic.

External testing (EXTEST) is fully supported, and there is limited support for internal self-test. For more information on boundary scan in XC4000 devices, see the Xilinx Application Note Boundary Scan in XC4000 Devices (XAPP 017).

Advanced Design Guidelines

Design Partitioning

The first phase of the design implementation process is partitioning. LCA devices emulate logic using look-up tables, and it is necessary to divide the schematic into groups of gates that will fit into individual look-up tables. Inefficient partitioning can both decrease the performance of a design and cause it to use more CLBs than necessary.

Inefficient partitioning can have two causes. First, the logic may have been drawn in such a way that convenient partitioning boundaries do not exist. PPR does not split large gates into parts that can be absorbed into unused

X5261
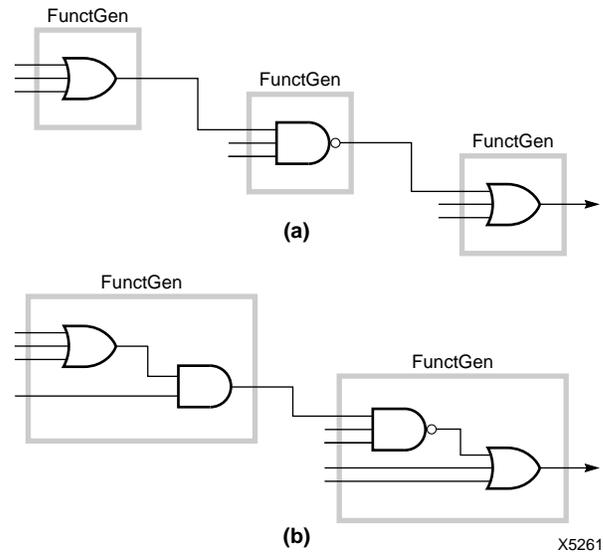
Figure 10. Improving Partitioning by Gate Decomposition

portions of surrounding function generators. Instead, it preserves gate boundary and may waste function generators. Figure 10 shows the same logic drawn in two ways; one way requires a cascade of three function generators, while the other way requires only two.

If a gate has a fanout greater than one, the partitioner always assigns the output of this gate to be the output of a function generator. PPR will not replicate the gate, even if the copies can be absorbed into other function generators. Figure 11 shows a second example where drawing the logic differently results in fewer CLBs and fewer levels of CLBs.

X5262
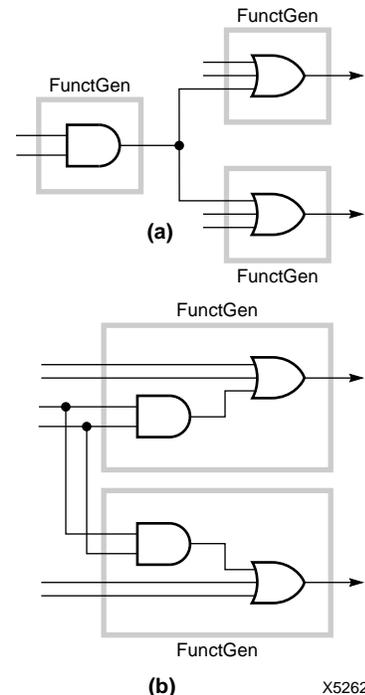
Figure 11. Improving Logic Partitioning by Gate Duplication

Note: (d) requires 2 FMAPs and 1 HMAP

X5263
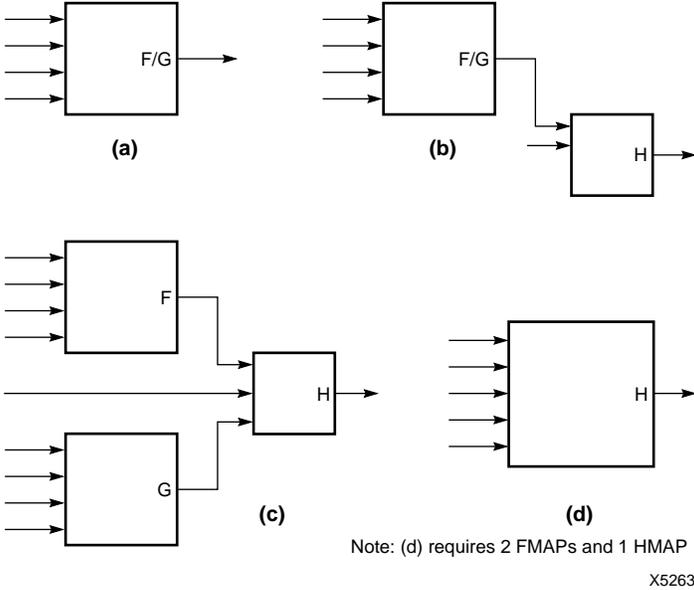
Figure 12. Preferred Functional-Block Sizes

Redrawing critical portions of an existing design can often improve its performance. It is much better, however, to draw the schematic initially in a way that guarantees a convenient partition. This is not as difficult as it might seem, and does not require gate-by-gate analysis or intimate knowledge of the partitioning algorithms.

Typically, the design process starts with a high-level block diagram, and progresses hierarchically through a series of increasingly detailed block diagrams as blocks are decomposed into smaller blocks. Eventually, the lowest-level blocks are translated into gates to create the logic diagram. The key to convenient partitioning is to structure the lowest level of blocks in such a way that it matches the XC4000 architecture.

Figure 12 shows four block structures that can conveniently be implemented in XC4000 devices. The four-input blocks correspond to F or G function generators, and the three-input blocks correspond to H function generators. The five-input block is a special case of the F-G-H combination. These blocks are characterized only by their number of inputs. Since a look-up table can emulate any function of its inputs, it is guaranteed that the logic required by each block will fit into a function generator, thus ensuring that a convenient partitioning exists.

This technique also facilitates performance estimates early in the implementation process. Each block in the block diagram corresponds to a delay specified in the XC4000 data sheet. With a simple routing allowance (add 50% to 100% of TILO per route), it is possible to estimate design feasibility. If necessary, structural de-

sign changes can be made before entering the schematic. For more advice on performance estimation, see to the Xilinx Application Note LCA Speed Estimation: Asking the Right Question (XAPP 011).

The second cause of apparent inefficient partitioning is the trade-off that PPR makes between area and speed. It is not possible for the built-in rule to match the requirements of every design. Partitioning can, however, be easily specified in the schematic using FMAPs and HMAPs, Figure 13.

X5272
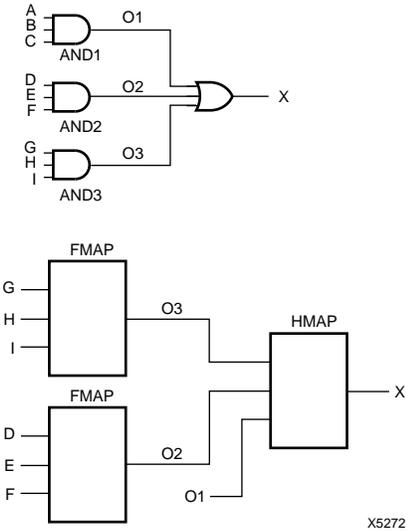
Figure 13. Using FMAPs and HMAPs

If an existing design is being optimized, isolated FMAPs and HMAPs can be added to improve critical paths. However, if the design technique described above is used, FMAPs and HMAPs can be included for the entire design to guarantee the expected partitioning. Little additional work is required since the FMAPs and HMAPs exactly match the blocks in the lowest-level block diagram.

Pipelining

Even with optimal partitioning, every logic function has a minimum achievable delay. If this minimum delay is too long, the function must be broken into smaller functions using pipelining.

Suppose a design needs to run at 20 MHz, and its worst-case Clock-to-Set-up delay is as shown in Table 1. Clearly, it does not meet its objective, and will only run up to 10 MHz. Of the 99.4 ns delay, 54.5 ns is attributable to block delays, and cannot be reduced by placement or routing improvements. In the unlikely event that all the routing delays could be reduced to the absolute minimum (~1.3 ns in XC4000-5), the worst-case Clock-to-Set-up delay of 66.2 ns would not still permit operation above 15 MHz. Using a more realistic routing estimate (50% to 100% of T_{ILO} per route), the design might run at only 10 – 13 MHz.

To achieve 20 MHz, there are four choices: use a faster device, improve the partitioning, restructure the logic to make the critical path less deep, or add a pipeline stage to break the function in two. While none to these may be possible in a particular case, pipelining is often the easiest choice.

Table 1. Worst-Case Clock-to-Set-Up Delay

Logical Path		Delay Cumulative
Source clock net : "CLK10MHZ" (Rising edge)		
From:BlkBLOCK1 CLOCK	to CLB_R13C12.YQ:	3.0 ns (3.0 ns)
Thru: Net NET1	to CLB_R13C11.C4 :	3.8 ns (6.8 ns)
Thru: Blk BLOCK2	to CLB_R13C11.XQ :	3.5 ns (10.3 ns)
Thru: Net NET2	to CLB_R23C11.F3 :	3.2 ns (13.5 ns)
Thru: Blk BLOCK3	to CLB_R23C11.X :	4.5 ns (18.0 ns)
Thru: Net NET31	to CLB_R18C9.C2 :	13.3 ns (31.3 ns)
Thru: Blk BLOCK4	to CLB_R18C9.XQ :	8.0 ns (39.3 ns)
Thru: Net NET4	to CLB_R13C10.F1 :	3.2 ns (42.5 ns)
Thru: Blk BLOCK5	to CLB_R13C10.Y :	7.0 ns (49.5 ns)
Thru: Net NET5	to CLB_R2C11.C4 :	8.7 ns (58.2 ns)
Thru: Blk BLOCK6	to CLB_R2C11.YQ :	8.0 ns (66.2 ns)
Thru: Net NET6	to CLB_R2C9.F3 :	3.7 ns (69.9 ns)
Thru: Blk BLOCK7	to CLB_R2C9.X :	4.5 ns (74.4 ns)
Thru: Net NET7	to CLB_R3C10.F4 :	1.6 ns (75.9 ns)
Thru: Blk BLOCK8	to CLB_R3C10.X :	4.5 ns (80.4 ns)
Thru: Net NET8	to CLB_R10C10.F3 :	4.8 ns (85.3 ns)
Thru: Blk BLOCK9	to CLB_R10C10.Y :	7.0 ns (92.3 ns)
Thru: Net NET9	to CLB_R9C11.G1 :	2.6 ns (94.9 ns)
To: FF Setup (D), Blk BLOCK10	:	4.5 ns (99.4 ns)
Target FFX drives output net "NET10"		
Dest clock net : "CLK10MHZ" (Rising edge)		

In an LCA device, every function generator has a flip-flop at its output. If this flip-flop is not being used for other purposes, it can be inserted into the logic path with minimal layout changes. There are difficulties, however. The latency introduced by pipeline flip-flops must be matched elsewhere both in data and control signals.

In the example, a pipeline flip-flop at the output of BLOCK5 would be most effective. The worst-case Clock-to-Set-up for the two halves of the function is 52.9 ns. This still does not meet the 20 MHz target, but is close enough to expect that routing changes could complete the job.

Of course, it is much better to anticipate such problems than solve them when the design is almost complete. The design technique discussed in the previous section provides delay estimates that are more than adequate for the detection of gross problems. Pipelining can then be built into the design, and any timing complications handled much more easily.

Floorplanning

Structured designs with a datapath-like organization and fixed data width can benefit greatly from simple floorplanning. Designs that use multiplexed busses are also candidates. In this section, floorplanning is discussed with respect to the absolute placement of logic functions within an LCA device. Relative placement control is discussed afterwards, in the next section on Hard Macros.

FPGA floorplanning is similar to planning a schematic diagram; an organization that permits gates to be connected conveniently on the schematic usually permits them to be connected conveniently in the FPGA. It must be remembered, however, that busses have to run though logic, not around it.

In most cases, the design itself suggests the floor plan. To minimize routing, bits with the same weight should be aligned in rows, and functions should be organized for the best data flow. Alignment of bits in rows is particularly important if a multiplexed bus is required, since horizontal TBUF Longlines must used for the bus.

If no obvious structure can be exploited, Floorplanning should be avoided. Arbitrary or poorly chosen placement constraints can hinder PPR, and lead to a worse result than if PPR were to run unconstrained.

In many instances of floorplanning, it is only necessary to consider the relative position of bits and functions. When planning busses, however, the absolute location can also be significant.

As described earlier, there are two TBUF Longlines per column of CLBs. Consequently, the widest bus that can be accommodated has twice as many bits as there are CLB rows in the array. This assumes that the busses require full-length Longlines. If a bus can be implemented using half-length Longlines, the center splitter can be opened, to accommodate twice as many bits.

Even if there are enough CLB rows to use full-length Longlines, it is good practice to use half-length lines wherever possible. The shorter lines have less capacitance and are, therefore, faster. In addition, the unused Longline halves are a valuable resource that can be used for other purposes.

Vertical Longlines run across the TBUF Longlines, and are often used for control signals. These vertical lines are also splittable, and, again, it is best to use half-length lines wherever possible. Floor plans should try to confine data paths to a single quadrant of the array.

While it is possible to use both halves of a TBUF Longline for two bits of a bus, this situation is far from ideal. Control signals must be duplicated in two columns since the Longline halves do not overlap. Functions like arithmetic carry that run across the bus are also complicated if they split between the two halves of the array. It is much better to use a device that is large enough to contain the full data path in one half of the array.

There is an IOB at each end of every TBUF Longline, and these IOBs have good connectivity to the Longlines. Consequently, I/O pins that need to connect to internal busses should be placed on the left or right sides of the chip, rather than on the top or bottom edges, where they lead to unnecessary routing congestion.

If signals are constrained to specific pins, the bit order of the pins should match the bit order of the bus. Again, routing congestion will result if the bit orders differ. Xilinx Hard

Figure 15. Efficient Placement in an XC4003

Macros have their bits in order with the MSB at the top, and if library macros are to be used, this standard should be adopted.

Consider the example shown in Figure 14. Two 8-bit registers, A[7:0] and B[7:0], and two 8-bit counters, C[7:0] and D[7:0], are multiplexed onto an 8-bit bidirectional bus, X[7:0], and routed to I/O pins. The registers each take four CLBs and should be arranged in columns. The soft macro counters also take four CLBs each. The part is an XC4003 with a 10 x 10 array of CLBs. Each quadrant is, therefore, a 5 x 5 array that can contain the logic.

Figure 15 shows a good placement that does not waste Longline resources. This placement constrains the bus and its multiplexed functions to one quadrant. It, therefore, only needs one set of horizontal and vertical Longlines. The bus I/Os are conveniently located to the left-hand end of the TBUF Longlines that drive them. A second choice for I/O placement would be at the right-hand end of the TBUF Longlines. This would, however, require the use of full-length TBUF Longlines.

Figure 16 shows an example of poor placement. It uses both halves of the horizontal Longlines to construct the 3-state bus, and additionally, might have to use both halves of the vertical Longlines to route the enable signals to the TBUFs. When the bus I/Os are located on the top or bottom of the graph (X6 and X7), routing them is difficult. X0-5 are not aligned with the bus, and require extra routing resources. This implementation would be slower than the previous one and consume more resources.

Figure 14. Bit Alignment for 3-State Bussing

X7 X6

X3
X2

| A7 | B7 | | C7 | D7 |
| A6 | B6 | | C6 | D6 |

X1
X0

| A5 | B5 | | C5 | D5 |
| A4 | B4 | | C4 | D4 |

| A3 | B3 | | C3 | D3 |
| A2 | B2 | | C2 | D2 |

| A1 | B1 | | C1 | D1 |
| A0 | B0 | | C0 | D0 |

X5
X4

X3264
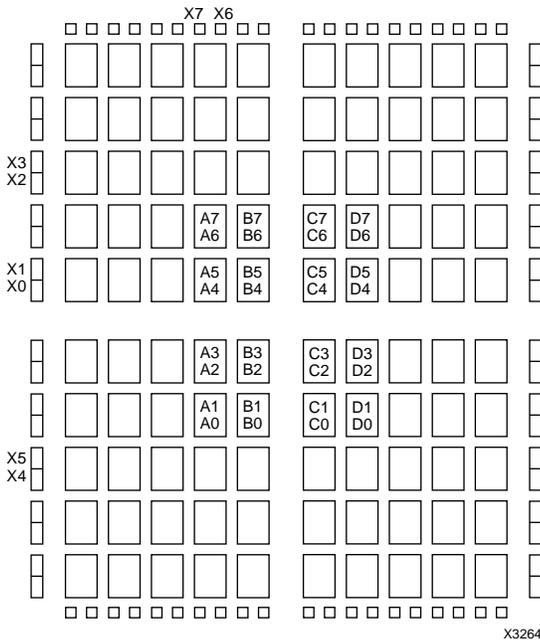
Figure 16. Inefficient Placement in an XC4003

Hard Macros

In the previous section, portions of the design were assigned fixed locations in the CLB array. If these fixed locations are chosen poorly, it can be more difficult for PPR to complete the design. There may, for instance, be many signals that must connect to the top of logic that is placed near the top of the array, thereby forcing PPR to generate the signals undesirably far from their destination.

This complication can be removed by creating Hard Macros. Hard Macros let the designer specify the placement of critical logic elements relative to each other, while allowing PPR the freedom to place the group where it is most convenient for the completion of the design. Unless specific I/O or Longline resources are needed, relative placement provides all the benefits of absolute placement without over-constraining PPR.

There are, however, factors that must be considered while creating Hard Macros. Firstly, all Hard Macros are rectangular; non-rectangular groups of CLBs convert to rectangular macros large enough to contain the original CLB group. In the process, empty CLBs are added that are not available to PPR, and will, therefore, remain unused in the final design. Holes in Hard Macros also remain unused. Leaving too many CLBs unused will cause a design to outgrow a device in which it would otherwise fit.

Secondly, the function generators in Hard Macros are closed, that is, no additional logic can be added into them. With a regular macro, an inverter driving a control pin will be absorbed into the macro, changing all the function generators to which the control pin connects but requiring no additional resources. With a Hard Macro, this is not possible

and an additional function generator, outside of the Hard Macro, might be necessary just to implement the inverter.

The third factor is pin-swappability. PPR often swaps function generator pins to improve routability. However, when a Hard Macro includes a carry-logic function, all the pins are locked on CLBs that use carry logic. Since PPR can no longer swap pins to improve routability, pins must be carefully selected when the macro is created.

Hard Macros are created from an unrouted LCA design that contains just the desired CLBs. The .lca file is converted to a Hard Macro using the program HMGEN. Also available from the Xilinx Technical Hotline is a Hard Macro Style Guide document that describes how to create macros that are compatible with those provided in the Xilinx library.

Locking Down I/Os

PPR permits the user to lock I/O signals to specific pins. Pin locking is sometimes necessary to ease congestion on the printed-circuit board, match the pin-out of an existing socket, or simply to allow printed-circuit-board design to start before the FPGA design is complete. Like any other constraint, however, pin locking limits PPR, potentially reducing performance or even preventing the design from routing completely.

Any locking of I/Os should be done as late as possible in the design process. If possible, 75 - 80% of the design should be completed before I/Os are locked. At this stage, the preferred pin locations for the FPGA design will be known, and a workable compromise between the needs of the FPGA and the needs of the printed-circuit board can be reached.

While it might be convenient to lock pins earlier, there is a danger that the pin constraints will prevent the FPGA design from completing successfully. When this occurs, the only solution is to remove some of the pin constraints, thus invalidating any printed-circuit-board design that has already been done.

Software Techniques

XACT Performance

With XACT Performance, the designer can specify timing objectives in XC4000 designs. These objectives are used by PPR primarily to optimize its use of routing resources. If a critical net needs a particular routing resource that has already been allocated to a non-critical net, the timing-driven router can choose to re-route the non-critical net, thus making the desired resource available. Timing objectives permit PPR to make such trade-offs intelligently, by telling it which nets are critical and how much freedom it has to slow non-critical nets.

XACT Performance does:

• Provide easily understood control of critical routing
• Allow PPR more freedom in critical areas

- Compensate for logic depth wherever possible
- Give early warning if timing expectations are unrealistic

XACT Performance does not:

- Perform delay matching
- Permit arbitrarily deep logic
- Guarantee that timing specifications will be met

This section provides hints and suggests options that can be used with XACT Performance. For a full description of XACT Performance that defines the terminology used in this section, see the *XACT Development System Reference Guide.*

Effective use of XACT Performance requires an understanding of two fundamental concepts, the Forward Tracing mechanism, by which net attributes are applied to paths, and the arbitration mechanism used when multiple attributes apply to the same path.

Forward Tracing: Although TIMESPEC attributes are applied to nets, they specify path delays between flip-flops, or between flip-flops and I/O. Each TIMESPEC applies to a group of flip-flops that is determined by tracing forward from the net with the TIMESPEC attribute. The TIMESPEC is applied to any flip-flop with an input that can be reached from the TIMESPEC net either directly, or through any depth of combinatorial logic.

Multiple Attributes: In many cases, more than one TIMESPEC will trace forward to the same flip-flop. Such conflicts are resolved according to the pin type on which the TIMESPEC arrives at the flip-flop. The arrival pin priority is shown in Table 2. By selecting an appropriate net for the TIMESPEC attribute, TIMESPEC priority can be set in the schematic. If more than one TIMESPEC has the same priority, the fastest TIMESPEC wins. Separate arbitrations occur for C2S, P2S and C2P specifications.

Once flip-flops have been assigned TIMESPECs, TIMESPECs can be assigned to paths. The TIMESPEC for a path is the faster of the TIMESPECs at its source and destination. If either the source or destination flip-flop does not have a TIMESPEC, the path does not have a TIMESPEC. This situation can only occur if default TIMESPECs have been set to IGNORE.

As stated above, XACT Performance operates primarily by allocating routing resources according to a net criticality. The simplest tactic that gives critical nets maximum access to routing resources is to minimize the criticality of non-critical nets. PPR, however, calculates default specifications for paths with no user TIMESPECs, and sometimes these defaults are unnecessarily demanding. Critical paths can benefit if the objectives of non-critical paths are reduced to match the design requirement.

In light of the "fastest wins" rule, the unattached attributes, DC2S, DP2S and DC2P, should be set to the slowest requirement for each path type. Other faster paths can then be specified explicitly. Alternatively, if there are "don't care" paths, the unattached attributes can be set to IGNORE.

Another way PPR creates default specifications is by modifying a flip-flop C2S specification to provide missing P2S or C2P specifications. Again, these defaults can be unnecessarily demanding. This problem is solved by providing the missing specifications, or disabling the default mechanism by setting EXTEND_C2S = FALSE, as described in the PPR Options section.

When PPR creates default specifications for a design, it does so by estimating a typical path delay based on logic depth in the design. The same estimate can be used in any TIMESPEC by setting the delay to AUTO, rather than a number of nanoseconds or megahertz.

AUTO is beneficial when a realistic estimate of achievable delay is not available. Overly loose specifications lead to unnecessarily slow results, since PPR stops improving a design once it has met the specified objectives. Impossibly short delay specifications are equally unproductive. As the optimizer tries to perform the impossible task, it will often create excessively long path delays, and overall, the design will be slower that one using AUTO delays.

Table 2.

Specification Level	Specification Method on a Schematic	Comments
1	Unattached attribute of type DP2S, DC2S or DC2P (the leading D is used to indicate a default attribute.)	Applies to all flip-flops in the design. Can be overridden by a Level 2 or Level 3 specification.
2	TS attributes of type C2S, P2S, or C2P plus corresponding flag attached to a net that can be traced forward to flip-flop clock pins.	Applies to all flip-flops whose clock pins are reached by the forward tracing mechanism. Can be overridden by a Level 3 specification.
3	TS attribute of type C2S, P2S, or C2P plus corresponding TS flag attached to a net that can be traced forward to flip-flop input pins other than clock pins.	Applies to all flip-flops whose non-clock pins are reached by the forward tracing mechanism. Overrides a Level 1 or Level 2 specification.

TIMESPEC

TS01=p2s:30
TS02=p2s:20
TS03=p2s:40
TS04=p2s:50

A0 — TS01
B1 — TS02
C2 — TS03
D3 — TS04

AND4

D

Clock

X3271
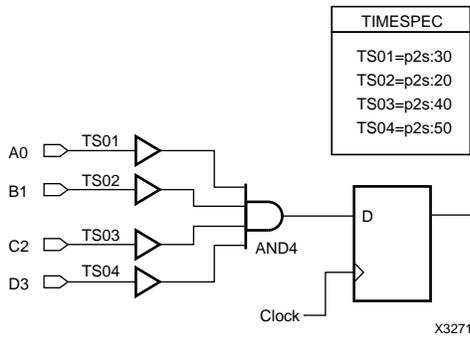
Figure 17. TIMESPEC Placement Where Some Specifications are Overridden

TIMESPEC

TS01=p2s:30:A*
TS02=p2s:20:B*
TS03=p2s:40:C*
TS04=p2s:50

A0 — TS01
B1 — TS02
C2 — TS03
D3 — TS04

AND4

D

Clock

X3272

Figure 18. TIMESPEC Placement Where Some Specifications are Lost

Whenever AUTO is invoked in a TIMESPEC on a net, a typical path delay is calculated based on only the logic to which the particular TIMESPEC applies. Each automatic TIMESPEC creates a different specification for the logic it controls, and consequently, multiple AUTOs lead to different speed objectives for different parts of the design. Exploiting these differences can benefit designs containing both simple logic that must be fast, and complex logic that can be slower.

Automatic TIMESPECs should placed in the design such that each TIMESPEC controls logic with a particular complexity and speed requirement. This arrangement causes the objectives for simple logic to be faster than those for complex logic, as desired. A single automatic TIMESPEC for the whole design, would set the objective according to the most complex logic, and no attempt would be made to make the simple logic faster.

A frequent cause for concern is the XACT Performance warning message [pa:SPEC_DOES_NOT_APPLY]. This does not necessarily mean that a specification was wrongly entered or has been ignored by XACT. As stated above, multiple TIMESPECs are often traced to a single flip-flop, where only one can be applied. After eliminating the redundant specifications, the arbiter documents its activity by issuing the warning.

The number of warnings is increased if the same TIMESPEC is applied to many nets and several of them trace to a single flip-flop. Each copy that arrives at the flip-flop is treated as a separate specification, thus increasing the number of specifications that must be eliminated and the number of warnings that are created. Careful placement of TIMESPECs will minimize the number of warning messages.

There are times, however, when P2S or C2P specifications can be completely overridden or ignored. In Figure 17, for example, all four TIMESPECs trace to the flip-flop D-pin, and the fastest, TS02, wins. TS01, TS03 and TS04 are overridden. Figure 18 shows an even worse example. Again, TS02 wins, but it is limited to paths from pads with name that start with "B" (the wild-card group B*). Consequently, the paths from A0, C2 and D3 are treated as unspecified, in spite of their TIMESPECs.

The solution is shown in Figure 19. When multiple P2S TIMESPECs are placed on a single net and they are qualified to operate with separate pads, all of the specifications are retained if the net wins in the arbitration. All specifications are then used by XACT Performance, each specifying the paths from the group of pads for which it is qualified.

In the example, there is only one net with TIMESPECs attached, and so it must win. All four specifications are retained since they come from the winning net. TS01 is used for A0, since it is qualified for the A* group. Similarly, TS02 is used for B1 and TS03 is used for C2. TS04 is used for all pads not in the A*, B* or C* groups. In this case, only D3 remains to be controlled by TS04. To simplify the schematic, the four TIMESPECs could be linked into a single specification, TS05, and this single TIMESPEC attached to the net in place of the other four.

When creating custom hard macros for use with XACT Performance, following these guidelines will help PPR analyze the design more effectively.

- If there is a clock pin on the Hard Macro, make certain that it is named C. This allows PPR to recognize the presence of flip-flops in the macro. If there is no clock pin on the hard macro, no other pin should be named

TIMESPEC

TS01=p2s:30:A*
TS02=p2s:20:B*
TS03=p2s:40:C*
TS04=p2s:50

A0
B1
C2
D3

AND4

TS01
TS02
TS03
TS04

D

Clock

X5264

Figure 19. TIMESPEC Placement Where All Specifications are Retained and Apply

C.
- Avoid creating hard macros with more than one clock, since PPR can recognize only one clock input per hard macro.
- Avoid combining clocked and non-clocked outputs on the same hard macro. When PPR detects a clock pin on a hard macro, it assumes that all outputs of hard macro are sourced by flip-flops.

Prior to PPR v1.31, two problems existed when using Hard Macros with XACT Performance. First, PPR was unable to analyze Hard-Macro timing adequately during the partitioning and placement phases of the implementation. This would result in early PPR predictions that timing requirements could be met, that were later proven untrue during routing. Second, analysis of false paths through carry logic would result in false notification that timing requirements could not be met.

Both problems have been resolved in PPR v1.31. However, there are still some difference between PPR and XDELAY timing analyses. These are described in the section on XDELAY.

When analyzing clock-to-set-up paths, PPR ignores clock skew. This is appropriate if clocks are distributed on global nets using BUFGP or BUFGS buffers. If clocks are routed on local nets, however, the skew can be significant, and must be anticipated when setting TIMESPECs.

To ensure correct operation, all completed designs should be carefully analyzed using XDELAY, and this is especially true when clock skew may be present. Since the -Analyze option of XDELAY does not consider clock skew either, a separate clock analysis must be performed. If PPR and XDELAY timing results differ, the XDELAY results should be used.

PPR Options
As its name suggests, PPR performs three principal functions: partitioning, placement and routing. In the partitioning phase, schematic gates are collected into function generator-sized groups. These groups and any associated flip-flops are then assigned to specific CLBs in the placement phase. First, an initial placement is created, typically using a Mincut algorithm. This placement is then improved by analyzing connection distances. Routing is an iterative process where early connections are often re-routed in order to free critical resources for nets that are routed later.

This section describes the more important PPR options and their recommended settings. At the end of the section, two examples of PPR settings are given.

DC2P = {(number), "auto", "ignore"}
DC2S = {(number), "auto", "ignore"}
DP2S = {(number), "auto", "ignore"}
DP2P = {(number), "auto", "ignore"}

Entry Method: command line, paramfile or /pprdx2x in XACTINIT.DAT file

The parameters are character strings that represent XACT Performance Default-Clock-to-Pad (DC2P), Default-Clock-to-Set-up (DC2S), Default-Pad-to-Set-up (DP2S) and Default-Pad-to-Pad (DP2P) specifications. The path-delay objectives may be specified in tenths of nanoseconds (0.1 to 3000.0), "auto" for individual automatic setting by PPR, or "ignore" to request timing be ignored for particular path types (except where explicit specifications are defined in the schematic).

For example, if there are no explicit TIMESPECs on P2P paths, setting DP2P to "ignore" will cause PPR to ignore all P2P paths in the design. If the design has no P2P requirements, this setting helps PPR meet the TIMESPEC requirements that are present. If DP2P is set to "ignore," explicit TIMESPECs that deal with P2P paths are honored, but only paths with P2P TIMESPECs are optimized; the delay could increase on other important paths. The default setting for these parameters is "auto".

Recommended setting:

ignore	if unconcerned with unspecified paths.
(number)	if trying to reach a specific delay.
auto	to let PPR decide a reasonable delay to try for.

extend_c2s={True, False}

Entry Method: command line, paramfile or /ppr/extend_c2s in XACTINIT.DAT file

When the variable is True, PPR automatically generates a P2S and C2P TIMESPEC for each C2S TIMESPEC without a corresponding P2S or C2P. The specifications are assigned reasonable values. To ignore P2S and C2P paths unless they have explicit TIMESPECs, set the variable to False. PPR can then concentrate on the C2S TIMESPECs, without optimizing P2S and C2P paths unnecessarily. The default setting is True.

Recommended setting:

False	To avoid unnecessarily demanding. automatic specifications.
True	To extend C2S specifications to P2S

and C2P paths.

ripup_allowance = (number)

Entry Method: /ppr/mxmazer/ripup_allowance = (number) in XACTINIT.DAT file

This option tells the router how many regressions to permit in the rip-up and retry process. Higher allowances increase the probability that a design will complete successfully, since the router tries longer. For example, if the router is close to its allowance and finds a minimum in the number of unroutes, it may stop. With a larger allowance, it might continue, and achieve a better result, although the number of unroutes may increase temporarily.

When running PPR multiple times to obtain the best placement, set the ripup_allowance low. Otherwise PPR can take a long time to produce a design that does not completely route anyhow due to a poor placement. By setting the parameter low, PPR can generate a result more quickly, and can go on to the next iteration of the loop.

If a design has hundreds of unroutes with a small ripup_allowance, increasing the allowance will probably not make it route completely. If the number of unroutes is 0-30, however, re-running PPR with a higher ripup_allowance will probably completely route the design. The default for this parameter is 2.

Recommended setting:

-1	Rips up and reroutes exhaustively, use if design will route or comes close to routing.
5-10	Use if unsure whether a design will route, or when running PPR loops.

improvecount = (number)

Entry Method: command line/paramfile

This option sets the number of iterations PPR makes to improve placement. The higher the number, the longer PPR attempts to improve the placement. If the placement ceases to improve, PPR stops with the best result it has obtained, regardless of the improvecount setting. Default = 3.

Recommended setting:

20	If runtime is not a factor.
6 - 8	Normally gives good results.

seeds_to_try = (number)

Entry Method: command line/paramfile

This option defines how many different seeds PPR tries in the placement improvement phase. For each of the specified number of seeds, PPR makes an initial placement and runs one improvement iteration. After this, the placement algorithm continues for (improvecount-1) iterations starting with the result that had the best score. Default = 1.

Recommended setting:

10	If runtime is not a factor.

3 - 4	Normally gives good results.

mincut_passes = (number)

Entry Method: command line/paramfile

This option sets the maximum improvement passes per partitioning attempt. Partitioning stops earlier if any pass shows no improvement. Default = 12.

Recommended setting:

20	If runtime is not a factor.
12	Reasonable value.

mincut_method = (number)

Entry Method: command line/paramfile

This variable selects the algorithm used during initial placement. The legal values are 0-3, and the default is method 3, which has proven to be best overall. On particular designs, however, methods 0-2 may yield better results. These other methods are most useful with designs that have low I/O utilization and/or large differences between flip-flop and function-generator utilization (check the PPR.LOG file for usage percentages).

Use these methods when placement achieved by method 3 is inadequate. There is no guarantee that the placement will improve, but a little experimentation can be very worthwhile. Default = 3.

Recommended setting:

3	Factory-tuned to normally yield good results.
0,1,2	Use experimentally to improve placement.

mincut_tries = (number)

Entry Method: command line/paramfile

This option sets the maximum number of initial configurations per partitioning step. While constructing a good initial partition, and improving upon it, a number of attempts are made at each partitioning step. The best result from these attempts is used in the design. Default = 2.

Recommended setting:

10	If runtime is not a factor.
2	For reasonable quality.

justflatten = (True, False)

Entry Method: command line/paramfile

When this option is set to True, the design is simply flattened into an .LCA file; the design is neither placed nor routed. Hard Macros, if they exist, are merged into the design. Using this option, a design is quickly translated into an .LCA file for back-annotation and unit delay simulation. This is

only necessary when there are hard macros in the design, and unit-delay simulation is required. Default is False.

Recommendations

The following sets of PPR options are recommended. For better results than are given by the default settings at the expense of a slightly longer run-time, use these values.

On the command line or in the paramfile:

    improvecount = 7
    seeds_to_try = 3
    extend_c2s = False
    DC2P = ignore   (if not concerned
    DP2S = ignore   with unspecified
    DP2P = ignore   paths.)

In the XACTINIT.DAT file:

/ppr/mxmazer/ripup_allowance = 8

For results that are potentially much better than those given by the default settings but with a much longer runtime, use the following values.

On the command line or in the paramfile:

    improvecount = 20
    seeds_to_try = 10
    extend_c2s = False
    mincut_passes = 20
    mincut_tries = 10
    DC2P = ignore   (if not concerned
    DP2S = ignore   with unspecified
    DP2P = ignore   paths.)

In XACTINIT.DAT file:

ppr/mxmazer/ripup_allowance = -1

The XDELAY Static Timing Analyzer

XDELAY provides static timing analysis of Xilinx FPGA designs. It analyzes all logic paths through a design, and produces a report that can be automatically analyzed for worst-case performance, viewed on-line, or stored as a file. XDELAY is documented in the *XACT Development System Reference Guide*.

Since, the XDELAY report for a single path can require many lines of text, and large designs can contain thousands of paths, XDELAY report files can be very large, sometimes occupying several megabytes of disk space. The size of these reports can be reduced significantly by using the filters provided.

Reports can be restricted to a single path type, clock-to-set-

up, for example, eliminating the reports on path types that are of no interest. For a worst-case-path analysis, setting the Delaygreater variable can eliminate paths that are too fast to be relevant. Localized analysis can be performed using the From, To or Ignorenet filters to limit the scope of the search.

PPR also provides static timing analysis as a part of its report, and sometimes there are differences between XDELAY and PPR timing results. These difference arise either because PPR does not trace certain paths, or because is analyzes some delays incorrectly.

XDELAY is the definitive Xilinx-FPGA timing analyzer, and its results should have more authority than those from any other source. XDELAY analysis should be part of the design process for all designs.

The following situations can result in PPR tracing paths differently than XDELAY.

- PPR does not trace through the asynchronous path from a reset-direct or set-direct pin to a flip-flop output. By default, however, XDELAY traces through this asynchronous path. The XDELAY option Flagblk CLB_Disable_SR_Q disables this tracing. See the item *PPR v1.30: Asynchronous Set/Reset Inputs Treated as Path Endpoints* in the release notes for more information.

- PPR treats RAM elements as combinatorial logic and continues tracing to the next clocked element or I/O pin. See the item *PPR V1.30: RAM Elements are NOT treated as Path Endpoints* in the release notes for more information.

- PPR does not trace through bi-directional IOBs as XDELAY sometimes does. See the item *XDELAY v4.30: Flagblk IOB_Disable_O_I and IOB_Disable_T_I Explained* in the release notes for more information.

The situations that causes PPR to analyze delays incorrectly are as follows.

- PPR always uses T_{IO1} as the delay through a TBUF. T_{IO1} is the delay from the TBUF I-pin to the output. Consequently, the delay from the T-pin of the TBUF will be incorrectly assigned the T_{IO1} delay. In some cases, T_{IO1} is less than the correct delay. WAND and WORAND elements are implemented with TBUFs, and are, therefore, also affected by this problem.

- If a BUFGP is sourced by internal logic, PPR must route that signal out through the dedicated IOB and back into the BUFGP input. In computing the delay of this path, PPR assumes that the output driver is configured for the FAST slew-rate, which is usually not the case. Thus, the delay may under-reported.

- Small discrepancies, typically less than 1 ns, appear in various places due to modeling differences between PPR and XDELAY/LCA2XNF.