

AN EFFICIENT RUN-TIME ROUTER FOR CONNECTING MODULES IN FPGAS

Jorge Surís, Cameron Patterson and Peter Athanas

Configurable Computing Lab
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA 24061
email: jasuris@vt.edu, cdp@vt.edu, athanas@vt.edu

ABSTRACT

It is often desirable to change the logic and/or the connections within an FPGA design on-the-fly without the benefit of a workstation or vendor CAD software. This paper presents a dynamic router for Xilinx FPGAs, designed to run on stand-alone embedded systems. With information obtained from Xilinx's XDL tool, a compact routing database for the Virtex-II/IIP/4 devices is built which only requires 96 KB of storage. A channel routing algorithm is used because of its deterministic execution time and because all routing resources in the channel are available. Sample channels are routed with the router and compared with the Xilinx PAR tool. Improvements in both execution time and in memory usage of several orders of magnitude are observed.

1. INTRODUCTION

With the introduction of bus macro-based partial reconfiguration [1], Xilinx provides a more flexible option that allows the designer to separate the static and dynamic portions of the design by pre-allocating slots where the dynamic modules will reside. Slots are areas in the reconfigurable fabric reserved for the exclusive use of dynamic modules. By providing the ability to reconfigure each slot separately, slots can function independently of each other. A partial bitstream is created for every module that may occupy a slot. All communication to and from a dynamic module must be routed through the provided bus macros which serve as the bridge between regions. Using this scheme, a partial bitstream is generated for every module that can inhabit a slot rather than a full bitstream for every supported combination of modules.

Although a flexible environment, the partial reconfiguration flow does have its limitations. Slots for dynamic module placement are allocated at design time, making the number of modules that can be instantiated at the same time a compile-time constant. The size of a slot is also fixed to the footprint of its largest module. This may prove wasteful if the same region is used for both large and small modules.

Another limitation is the static communication scheme requiring all modules sharing a single slot to have the same input and output ports.

Contemporary RTR frameworks lack the ability to establish connections between modules at run-time, and hence have limited potential. For example, a module with new functionality may not be able to be integrated into the system if its interface does not conform to the interface defined at design time. Future RTR frameworks should include a router capable connecting dynamic modules to each other with no previous knowledge of what their interface is. This paper presents a new dynamic router, developed for the Wires-on-Demand (WoD) RTR framework, for the connection of run-time instantiated modules [2]. Because ports are assumed to be registered, the routing algorithm need not be timing driven, allowing the use of an algorithm that exhibits low execution times.

2. DYNAMIC ROUTING

FPGA routing optimization continues to be an active area of research, and the difficulty lies in improving routing execution time without affecting system performance. Research on static routing in ASICS and FPGAs has concentrated on minimizing the propagation delay associated with critical paths. Because static routing is performed at compile-time, the execution time and storage requirements of the router need not be bounded. This leads to the development of algorithms such as PathFinder [3] that use graphs representing the routing resources of the entire device and go through several routing iterations to determine the best possible routing for critical paths. With current routing-rich devices, the memory required for these graphs can often exceed 100 MB. The iterative nature of compile-time routing algorithms also leads to execution times measured in minutes or hours for dense designs targeting large parts.

2.1. JBits

Dynamic routing arose from the desire to change a design once it has gone through the vendor place and route tools. The first significant effort in dynamic routing was accomplished with JRoute, which was part of the Xilinx Bitstream Interface (XBI) package [4] within JBits. JBits consists of a set of Java application programming interfaces (APIs) that allow a user to programmatically alter a bitstream. Routing was accomplished by controlling individual Programmable Interconnect Points (PIPs), or by extending the Router class to implement a custom routing algorithm. Because JBits was designed to allow the implementation of logic as well as routing, it has the capacity to control many of the configurable elements in the device. This has resulted in a large database and memory usage upward of 100 MB. With the release of JBits version 3.0, support for JRoute was replaced with a wire database and an API for constructing routes. It was not until 2002 that a new router was developed for JBits as part of the ADB wiring database [5]. Routing information was compressed in relation to JRoute, reducing the access time and storage requirements. Because the database still contains the information for all the PIPs in a device and each device is different, memory requirements are still measured in tens of MBs. Lastly, ADB does not currently support the Virtex-4 family of devices.

2.2. Block-based Routing

Block-based routing addresses the difficulty of routing by simplifying the problem [6]. Similar to Xilinx's partial flow, slots for dynamic modules are allocated at design time, with a fixed interface to static logic. Pre-defined blocks are used to establish a vertical routing channel along the entire height of the slot, connecting the slot to static logic. Other blocks, containing horizontal as well as vertical routing, are used to connect dynamic modules to the routing channel. Using a combination of these two block types, modules can be connected to the static logic. This methodology allows for a more flexible placement of modules in the slot, but limiting routing to specific blocks or templates results in reduced flexibility, making techniques such as signal jogs and bit shifts difficult to implement.

2.3. Wires-on-Demand

Building on top of Xilinx's partial reconfiguration flow, the Wires-on-Demand framework addresses some of the partial flow's shortcomings. Rather than use multiple reconfigurable slots, one large *sandbox* region is defined that contains all dynamic modules. Traditional bus macros are used to define the interface between the static and dynamic logic. All dynamic modules are wrapped at design time to anchor and buffer all ports at known locations. The wrapper also

ensures that all modules are self-contained by confining all logic and internal routing inside of the module. Using meta-data stored in a module database, the placer finds the best location for each module, ensuring access to any required special resources such as Block RAM (BRAM) or DSP blocks. A router is then used to connect ports between modules. By expanding the dynamic region to encompass all modules, only the interface to the dynamic region is fixed.

3. WIRES-ON-DEMAND ROUTER

Clear goals have to be set when designing a router, as an improvement in one property can adversely effect another. This is evident in the relationship between execution time and path propagation delay, which is the time required for a change at the input of a signal to be reflected at all its outputs. If minimal critical path delay is desired, iterative algorithms such as VPR [7] and PathFinder may be required, which have non-deterministic execution time. Because one of the target platforms for WoD is embedded systems, the primary design goal was to minimize memory usage and execution time, with the propagation delay minimization as a secondary goal. If all module inputs and outputs are registered, the maximum propagation delay of routed signals need only be less than one clock cycle. This permits timing to be traded off for improvements in execution time and memory usage [8].

3.1. Routing Database

As mentioned in Section 2, dynamic routers such as JBits require large databases to function properly. This is largely due to the support of local, chip-wide and logic routing, which requires knowledge of all the routing resources. By limiting the scope of the router to local routing between modules, routing resources with sparse connectivity, long spans, or with specialized purposes such as carry logic can be ignored, resulting in a reduced database size with little effect on the quality of results.

For the devices targeted in this work (Xilinx's Virtex-II/IIP/4) the `double` and `omux` routing resources are used. A `double` segment spans three interconnect tiles stopping at every tile. Interconnect tiles contain switch matrices connecting logic elements and other resources to the routing architecture. The middle point can drive an input of every look-up table (LUT) in a configurable logic block (CLB), and a segment that travels in each orthogonal direction. End points share the same connectivity as the middle points, but have an additional connection to other segments traveling in the same direction. Figure 1 shows a simplified view of the arrangement of `doubles` in which the lines inside the CLBs are possible connections that can be established. Note that `doubles` travel in four directions: north, south, east

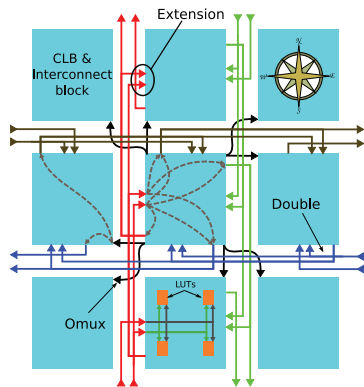


Fig. 1. Simplified Virtex-II/IIP/4 routing architecture.

and west. Omux segments, which can also be seen in Figure 1, are used despite their limited connectivity since they provide direct connections to all eight neighboring tiles.

Size is not the only factor in the decision to build a custom database. One of the goals when designing WoD is to create a framework that does not depend on mainstream tools for run-time operations, and has the capability to run autonomously. This objective came out of a desire to run the framework on platforms that are not supported by the vendor place and route tools, such as an embedded PowerPC processor. To that end, a generic database has been built that can be compiled into any C/C++ program. The resulting router can be run on any platform which has support for C++ and the Standard Template Library (STL).

Current Xilinx devices have the feature that all tile types use a similar switch matrix. Only resources that share the same connectivity on all the switch matrices are used in the WoD router. With this, all of the information for one such block per device family is stored, reducing the size of the database. The standard implementation tools cannot take full advantage of these similarities because they use all routing resources. Information is stored for each switch matrix instance using flat databases (.nph files). As a result, Xilinx's routing resource databases are device-specific and rather large.

In the WoD routing resource database, information is maintained for all doubles and omuxes that can start, stop or end at an interconnect block and the PIPs connecting them. Inputs and outputs for LUTs and flipflops are also stored. For each routing segment, the following information is stored:

- *Wire Index*: a unique identifier used to find entries in the database.
- *Type*: the type of segment (i.e. double, omux, F-LUT input, etc...).

- *Direction*: indicates the direction this segment is traveling (i.e. north, south, east, west).
- *Point*: if this is the beginning, middle or end of the wire segment.
- *Segment Index*: used to differentiate between wires that have the same type and direction.
- *Extensions*: a list of wire segments that connect to this segment in another switch matrix. For example, the middle point of a double would be an extension of the beginning. Figure 1 points out two such extensions.
- *Connections*: a list of the wire segments in the interconnect block that the wire segment can drive. Connections are identified by their wire index.

For current Xilinx devices, approximately 218 entries are stored in the WoD database with a total size of 9.6 KB per device family. In contrast, Xilinx's nph files for the Virtex-4 family range in size from 3.6 to 19.5 MB for a total size of 150 MB.

The source information for building the database is obtained from the Xilinx Design Language (XDL) tool. A Perl script generates an XDL PIP report that contains all the connectivity information for the specified device. Information is then extracted from the report by the script, and an array of C/C++ structures used to represent the information detailed above is created and placed in a header file. The resulting header file can be compiled into any programs that requires the routing information.

3.2. Routing Algorithm

Several algorithms were considered for the WoD router. VPR and PathFinder provide timing-driven results, but their execution time is non-deterministic due in part to their iterative approach, and their memory requirements are exceedingly large. First a flat graph of all architectural routing resources is read, and all signals are initially routed allowing segment contention. Paths that are being used by more than one signal are assigned higher costs and the signals are re-routed. This process is repeated until all signals have been routed with no conflicts.

Channel routing is a simpler problem. Instead of building graphs representing all routing resources, a limited list of possible paths, called tracks, are used to route all signals. Because the number of tracks is fixed, channel routing has the attractive attributes of deterministic execution time and limited memory requirements. The segmented nature of FPGA routing resources lends itself well to channel routing due to a large number of tracks in a condensed space. The flexibility provided by the connectivity of double segments is well suited for creating tracks in a channel. The

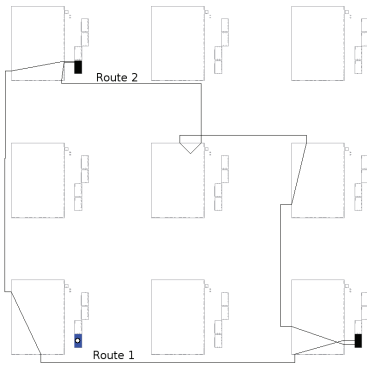


Fig. 2. Two versus three segment routes.

mid point of a segment can be used to create a track that spans two interconnect blocks, or a track can be made to span the entire width of the channel by stitching the segments together. For these reasons the WoD router uses channel routing.

3.3. Track Allocation

Most channel routers assume that the start and end points of a signal lie on opposite sides of the channel. If this is the case, the choice of which track to use within the channel is only important as far as it affects the router's ability to route other signals. Regardless of which track is used, the signal consists of three segments: one to get to the channel, the track used in the channel, and one to exit the channel.

Extending the channel to include the module boundaries increases the number of tracks available to the router and makes the choice of track much more relevant. This can be done since all modules are self-contained and ports are anchored to a boundary region devoid of module-related logic or routing, allowing the use all routing resources beginning in the boundary region that do not flow into the module. By using the tracks in the module boundaries, routes connecting two modules can be accomplished without the use of any additional area.

If the track chosen lies in the same CLB row as the source or sink of the signal, the route can be completed using two segments instead of three. Figure 2 shows a two segment and a three segment signal. These two signals have nearly identical starting and end points but the propagation delay for Route 2 is nearly 65% higher than for Route 1. The router uses this knowledge when allocating tracks, attempting to use those in the same CLB row as the signal source or sink before attempting to use any others. As with segment allocation, tracks are allocated in a greedy manner and signals routed first have a higher probability of finding a track that allows a two-segment route.

3.4. Memory Usage

Beyond the routing database, the router need only keep track of which supported segments are in use by each switch matrix. To accomplish this, an `Interconnect` class is used with functions to check whether a segment is in-use, or to set a specific segment as in-use. The x and y coordinates of the switch matrix in the device it represents are stored in the object, as well as a short integer *indicator* for every represented segment. These indicators are allocated in the constructor. Because the router assumes that all resources in the channel are reserved exclusively for its use, all indicators are initialized to zero. Objects of the `Interconnect` class are allocated for every switch matrix in the channel, allowing the router to track the status of every represented segment in the channel. A segment can be used to route a signal if either the indicator for the segment is zero, or if marked with the same signal number as the current signal being routed. The latter allows re-using segments in multi-sink signals.

The memory footprint of routers using a flat graph of routing segments and PIPs grows in proportion to the area being routed. By using tracks instead of routing graphs, memory is not needed to map all possible paths, just available segments. Because the number of segments is constant in each interconnect block, the memory requirements for channel routing increase linearly with the height and width of the channel.

3.5. Optimizations

Even though the WoD router does not try to minimize the propagation delay of routed signals, timing cannot be ignored. To assess the timing results, version 8.2.03i of the PAR tool was run for several common scenarios. Patterns were identified and heuristics developed to incorporate these patterns as templates into the signals routed by the WoD router. The track allocation algorithm previously discussed is one such heuristic. Another optimization is to use a combination of segments types that ensures each segment is used in its entirety. For example, if a signal needs to travel a length of three CLBs it is usually preferable to use an `omux` and a `double` rather than using two `doubles`. The multi-directional nature of the `omux` segments is also exploited. When a signal's route must take a 90° turn, the router attempts to use an `omux` segment with a diagonal extension. This reduces the path length of the signal and may reduce the delay. The use of a diagonal extension can be seen in Route 2 in Figure 2.

4. DESIGN EXAMPLES

Although there are several benchmarks used for gauging the performance of a router, none of them address the specific

	Wires-on-Demand Router				Xilinx's PAR			
	Max Delay (ns)	Storage Used (KB)	Exec Time (ms)	Avg. # of Segments	Max Delay (ns)	Storage Used (KB)	Exec Time (ms)	Avg. # of Segments
DES Key	1.366	257	1.2	5.6	1.002	199,680	20,000	5.1
Direct Bus	0.337	140	1.2	2.0	0.406	224,256	17,000	2.0
Inverted Bus	0.918	177	1.6	3.3	0.756	225,280	17,000	2.9
Shifted Bus	0.876	178	1.6	3.4	0.764	225,280	17,000	3.0
Byte Swap	0.927	177	1.4	3.3	0.746	225,280	17,000	2.9
Bit Swap	0.427	140	1.0	2.0	0.428	225,280	17,000	2.0
Bifurcated Bus	1.366	299	2.3	6.4	1.080	225,280	17,000	6.2

Table 1. Performance comparison between Wires-on-Demand router and Xilinx's PAR.

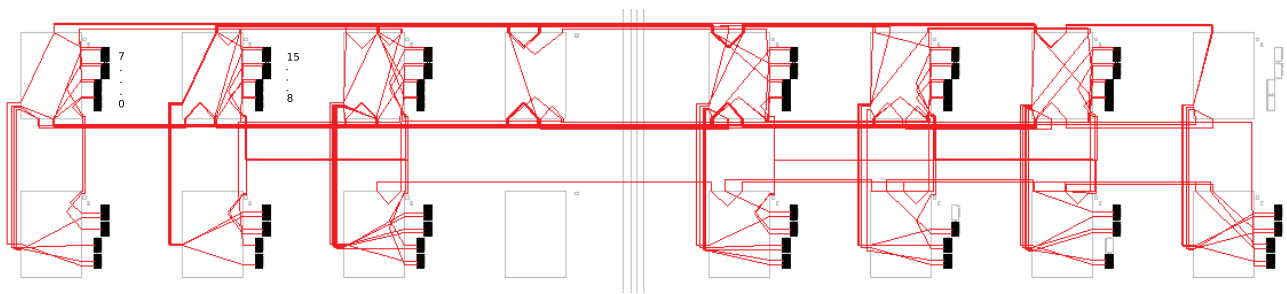


Fig. 3. DES key schedule channel.

problem of routing signals between modules in a rectangular channel devoid of other routing. Lacking a suitable benchmark, the following scenarios were chosen to evaluate the WoD router on a Xilinx Virtex-IIP XC2VP30-5FF896C FPGA:

- *DES key schedule stage:* Data Encryption Standard (DES) was considered to be a difficult test since the permutation of the key's bits between stages creates a large number of signal jogs.
- *Direct Bus:* Routes a 32-bit bus, where the source and the sink of every signal are vertically aligned.
- *Inverted Bus:* Routes a 32-bit bus, where the bits of the bus are in inverted order.
- *Shifted Bus:* Routes a 32-bit bus, where the input bus is shifted two CLBs in relation to the output bus.
- *Byte Swap:* Routes a 32-bit bus, where the order of the bytes is inverted.
- *Bit Swap:* Routes a 32-bit bus, where the order of the bits of the bus are inverted in two-bit pairs.
- *Bifurcated Bus:* Routes a 32-bit bus, where the output bus is connected to two input busses.

The resulting channel from the DES sample design can be seen in Figure 3. The two thin vertical columns in the center

of the channel are BRAM columns with their corresponding switch matrices to the left. Though the router cannot connect signals to the BRAMs, it can be seen that the routing resources of the BRAM interconnect tile are utilized. This is true for any special column in the device.

After a signal has been successfully routed, the configuration changes required by this signal are recorded in the bitstream using custom software. This is done by turning on the PIPs that establish the connections between the routing segments used. The equations of sink LUTs are also changed to ensure they act as passthroughs for the input pin used by the router.

In Section 3.2, routing success is defined as creating paths between modules with a propagation delay of less than one clock cycle. As a measure of the WoD router's effectiveness, a timing analysis was done on hard macros produced from the routed channels, with the Xilinx FPGA Editor tool. The channels were then routed with PAR version 8.2.03i and its performance was analyzed. Table 1 compares the performance of both routers for each of the sample channels. Both routers were run on a dual 2.8 GHz Pentium 4 PC with 1 GB of memory running a Debian distribution of Linux kernel 2.6.18. The maximum delay achieved was 1.366 ns, which represents a 36% increase from that achieved with PAR, while the average increase in delay was only 15%. It is important to note that the delay of the DES channel is below the requirements for a system running at 600 MHz, which is well above the operating frequencies of most designs.

The average number of segments used per signal was also determined as an additional metric of the router's effectiveness. The design produced by PAR for the DES channel used an average of 5.1 segments per signal, while the design produced by the WoD router for the same channel used an average of 5.6 segments per signal. The fact that the average number of segments per signal is similar for all the design samples may indicate that the increase in delay is not due to the choice of segment type, but the choice of specific segments. Two segments which may seem to be identical can have different propagation delay characteristics due to the physical layout in the silicon. For example, ten eastbound double segments have a starting point at each interconnect block, but segment one may have a larger propagation delay than segment four. PAR uses this knowledge to choose segment four over segment one for critical paths. On the other hand the WoD router considers segment one and four to be equivalent.

Table 1 also demonstrates the large difference in memory requirements. For the sample designs PAR required an average of 216 MB of storage partly because it uses a flat graph of all routing resources in the device. On the other hand, the WoD router only required an average of 195 KB of storage because the database is built into the executable and no graphs are read or constructed at run-time. Integrating the database into the executable does increase the size of the executable, but the increase is negligible compared to the reduction of three orders of magnitude in the storage used.

A considerable decrease in execution time was also observed, attributable to the routing algorithm used. PAR uses an iterative algorithm where some signals are routed multiple times with the goal of obtaining optimal routing for critical paths. When signal density is high, as in the DES design example, segment reuse may increase the number of iterations required to finally obtain a routing with no segment reuse. Because the WoD router uses the first available route found and does not allow signal reuse at any point, execution time is not affected by signal density.

5. CONCLUSIONS

Dynamic routing has historically been the most difficult part of run-time reconfigurable systems. Many systems avoid the problem by making all routing between modules static. Others such as JBits require large databases or a complex infrastructure such as a Java Virtual Machine. This paper presents a viable alternative to vendor place and route tools for dynamic global routing between modules. Exploiting the repetitiveness of the routing fabric in current Xilinx devices allowed the creation of a database with a size several orders of magnitude smaller than those used by vendor tools. Avoiding large resource graphs reduced memory usage by three orders of magnitude compared to PAR. The determin-

istic nature of channel routing, which is the basis of the WoD router, led to execution times that are four orders of magnitude less than PAR. The WoD router yielded path delays comparable to PAR for the sample designs.

The router presented in this paper allows for the development of flexible reconfigurable systems. By providing the ability to connect modules at run-time with little to no restrictions, the modules and how they connect are no longer static parts of the design. The router's modest resource requirements and independence from vendor tools means that it can be used in platforms ranging from embedded systems to hybrid supercomputers.

6. ACKNOWLEDGEMENTS

This work is supported by the United States Air Force under Contract Number FA8651-06-C-0126. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force.

7. REFERENCES

- [1] Xilinx Inc., *XAPP290: Two flows for partial reconfiguration: Module based or difference based*, September 2004.
- [2] P. Athanas, J. Bowen, T. Dunham, C. Patterson, J. Rice, M. Shelburne, J. Surís, M. Bucciario, and J. Graf, "Wires on Demand: Run-time communication synthesis for reconfigurable computing," in *FPL 2007: Int. Conf. on Field Programmable Logic and Applications*, pp. 513–516.
- [3] L. McMurchie and C. Ebeling, "PathFinder: a negotiation-based performance-driven router for FPGAs," in *FPGA '95: Proc. of the 1995 ACM Third Int. Symp. on Field Programmable Gate Arrays*, 1995, pp. 111–117.
- [4] S. A. Guccione and D. Levi, "XBI: A Java-based interface to FPGA hardware," in *Configurable Computing: Technology and Applications, Proc. SPIE 3526*.
- [5] N. Steiner and P. Athanas, "An alternate wire database for Xilinx FPGAs," in *FCCM '04: Proc. of the 12th Annual IEEE Symp. on Field-Programmable Custom Computing Machines*. IEEE Computer Society, 2004, pp. 336–337.
- [6] M. Hübner, C. Schuck, and J. Becker, "Elementary block based 2-dimensional dynamic and partial reconfiguration for Virtex-II FPGAs," in *IPDPS*. IEEE, 2006.
- [7] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Field-Programmable Logic and Applications*, W. Luk, P. Y. Cheung, and M. Glesner, Eds. Springer-Verlag, Berlin, 1997, pp. 213–222.
- [8] C. Mulpuri and S. Hauck, "Runtime and quality tradeoffs in FPGA placement and routing," in *FPGA '01: Proc. of the 2001 ACM/SIGDA Ninth Int. Symp. on Field Programmable Gate Arrays*. New York, NY, USA: ACM Press, 2001, pp. 29–36.