An FPGA Wire Database for Run-Time Routers

Eric Keller and Scott McMillan

Xilinx Inc. 2100 Logic Drive, San Jose, CA 95124 (USA)
{Eric.Keller, Scott.McMillan}@xilinx.com

Abstract. Routing flexibility improves as FPGAs increase in size and density. While advantageous for applications, the routing resource software model requires significant hard disk and memory resources. As a result, run-time routers tend to devise alternate solutions to the flat graph model used by FPGA tools in order to accomodate the limited memory available in run-time systems. JBits, a run-time reconfiguration (RTR) tool suite from Xilinx ®, contains just this type of run-time router (JRoute). In order to accommodate the run-time memory limits placed on JRoute, JBits chose to take a different approach to storing circuit graphs. The solution, the JBits wire database, represents the wire connectivity of a device with Java objects. These objects are stored in a device generic manner that limits the references to repetitive routing structures. This and other memory reduction techniques implemented by the JBits wire database enable automatic run-time routing in RTR environments using JRoute. Beyond memory savings, the JBits wire database provides low-level control over individual wires, which is desirable in run-time routers. Also desirable in run-time routing is the ability to incrementally add and remove connections. This, along with the low-level control and the memory efficiency of the JBits wire database opens up new applications. Applications such as device testing, defect tolerance, and debugging are efficiently implemented using run-time routing with the JBits wire database.[1]

1 Introduction

As FPGA architectures continue to increase in density, a flat representation of the routing segments and switches consumes an inordinate amount of memory. This approach is common for static FPGA tool flows that run on systems abundant in memory and storage space. However, the memory limits placed on run-time routers in dynamic systems discourages the use of this single graph technique and uncovers the need for another methodology.

JBits[2] is a Java API that provides access into a Xilinx FPGA configuration bitstream, thus, enabling run-time reconfiguration (RTR). The JBits RTR environment contains a run-time router called JRoute[1] which is placed under the same limitations as any other dynamic router with regards to memory and storage. As a result, the JBits development suite saw an immediate need for a methodology that avoids the heavy memory and storage requirements of the flat graph representation.

The approach JBits uses to store connectivity information uses object oriented structures that take advantage of the repetition of identical routing structures tiled throughout

[1] This work was supported in part by DARPA in the Adaptive Computing Systems (ACS) program under contract DABT63-99-3-0004.

the FPGA device. While the compression of routing structures reduces memory overhead, the object oriented wire classes also give complete control over specific wire resources. This combination of memory efficiency and wire database control is very desirable in RTR applications.

Applications such as device testing [6], defect tolerance [7], and debugging [8] are ideal for taking advantage of the benefits offered with run-time routing. The memory efficiency of the JBits wire database enables fast implementation and modifications. The low-level control over individual wires allows for applications to specify wires to use and to avoid. The incremental routing allows for routes to be added and removed as the application executes. Each application has a dynamic nature to it. Defect tolerance, for example, uses the low-level control to tell the router to avoid using the defective wires. Incremental routing is then used to reroute any nets with defective wires on it.

In Section 2 background information on JBits and JRoute is presented. In Section 3 the JBits wire database is detailed. In Section 4 applications that utilize dynamic routing are summarized. In Section 5 a comparison is made between the wire database presented in this paper and other research. In Section 6 conclusions are made followed by future work in Section 7.

2 Background

2.1 JBits

JBits is a Java API that provides direct access into a Xilinx FPGA configuration bitstream. It allows a bitstream to be modified by defining an API allowing configuration of all routing and logic. The ability to modify the bitstream directly makes run-time reconfigurable applications possible.

A tile based approach is taken to bitstream manipulation by JBits where each tile represents some functional block of the FPGA (e.g. CLB tile, IOB tile, BRAM tile). While this approach is not unique, JBits varies from traditional techniques by building device specific tile structures at run-time. This methodology allows any tile associated information to be stored once separate from any device specific structures. Typically, standard tools store this type of data in a device specific file that duplicates this information for each identical tile.

Figure 1 provides a graphical representation of how JBits builds device specific tile arrangements at run-time. JBits assigns integer constants to each of the unique tile types in an architecture family (e.g. the Virtex™ family of devices). Each device tile array in an architecture family is stored statically using only these unique integers. When a user application constructs a new JBits object, the device (e.g. XCV1000) must be specified and used to select the stored device tile array.

JBits uses this device tile array to reference the information that has been stored based upon the unique tile types of an architecture family. As an example, when an applications reads a bitstream, JBits will utilize the device tile array to set pointer values to bitstream locations associated with specific tiles. This provides the information necessary to perform fast run-time calculations that will locate resource configuration "bits" in the bitstream.

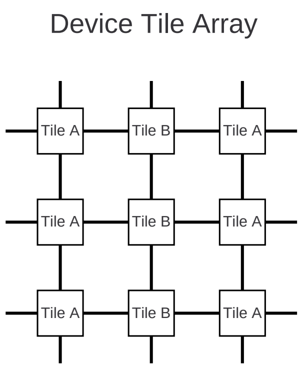

Fig. 1. With JBits, each identical tile in the tile array accesses the same circuit graph in memory (the intra-tile connectivity). In this example there are only 2 unique intra-tile connection graphs.

2.2 JRoute

The JBits tool suite contains a run-time router (JRoute) that utilizes the wire database presented in this paper. While the JBits wire database presents router developers with an interface into the device connectivity, JRoute[1] provides the application developer with a dynamic, incremental run-time routing API. This combination of layered APIs allows the user to determine the level of control needed for a specific application.

JRoute provides different levels of abstractions in a route request. The user has the ability to auto-route a net, define a template for a generic route or fully specify a net. JRoute tracks resource utilization during route request processing and allows future removal of the completed routes. These features give the user complete control over the JRoute incremental routing process and, as with the wire database, the API allows flexibility in the underlying routing algorithms.

3 The JBits Wire Database

3.1 Tile Based Representation

The wire database used by JBits stores routing graphs based upon unique tiles as shown in Figure 1. Storing these graphs, independent of device, results in a significant storage reduction since identical tiles are duplicated thousands of times for just one device. For example, in an XCV1000 device, there are 6,144 (64 X 96) CLB related tiles. This causes the CLB tile routing graphs to be duplicated 6,144 times for just the XCV1000 device when using traditional methods. If the other Virtex [3] devices are included, for this same tile, the factor will be raised to over 20,000. This factor will continue to grow with the newer architectures.

The reduction in database size certainly reduces the storage required and this benefit will extend to memory resources as well. In the context of memory, the JBits approach has some additional techniques for minimizing memory utilization to be discussed in

Section 3.2. In addition to being stored only once per tile, each routing resource wire is stored as a Java object completely separate from all other wires in a specific tile. These techniques take advantage of Java's on demand loading and garbage collection.

3.2 Object Oriented Representation

Figure 2 shows an example of a wire object in the JBits wire database. The stored part of the wire discussed presently consists of the intra-tile connectivity shown in CLB 0. Each wire is represented by a Java object that will be loaded on demand, which is a benefit of using Java. Since many designs do not require every wire to be instantiated, this methodology will result in additional memory savings. For example, the implementation of the Smith-Watermann algorithm [16] required only 1,136 out of 2,424 wire objects to be instantiated. This leads to an additional 50% reduction in the memory required beyond those savings obtained from the tile based approach already described.

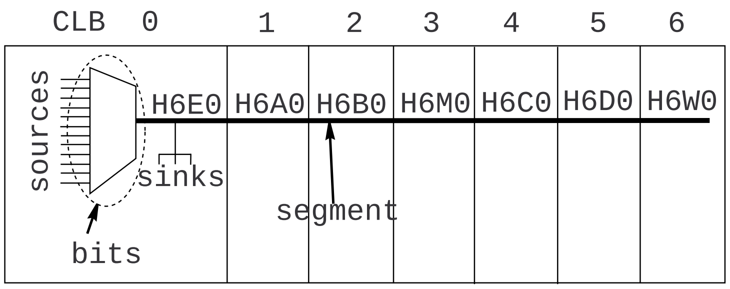

Fig. 2. Diagram showing a hex east 0 wire object. Shown are the intra-tile source and sink connectivity information and the inter-tile segment that crosses over 6 CLB tile boundaries. The "bits" shown represent the necessary configuration "bits" needed to connect any of this wires sources to this wire object.

Figure 3 illustrates how the wire objects can be manipulated by a router. Step 1 starts off with an output XQ wire object. The XQ object is queried in step 2 for its sink wire objects (OUT0, OUT1 and OUT7 are shown). If one of the objects needed has not been previously accessed, it is dynamically instantiated and loaded into memory. In this example, the OUT0 wire object's connect methods are used to configure the OUT0 mux to provide a route from XQ to OUT0. In step 3, a similar process occurs and E2 is chosen to be the next hop in the route from XQ.

In the example so far the intra-tile connectivity has been considered without mention of wires that cross inter-tile boundaries. Step 4 shows what happens when wires cross tile boundaries. Besides querying E2 for sink objects, in step 4 E2 is also probed for its tile to tile segment connectivity. Then, as seen in step 5, the path route moves from E2 to an adjacent tile where E2 changes names to W2. This process can continue in any manner the controlling process desires and when a route is complete, Java can then garbage collect any inter-tile segments created while producing the route.

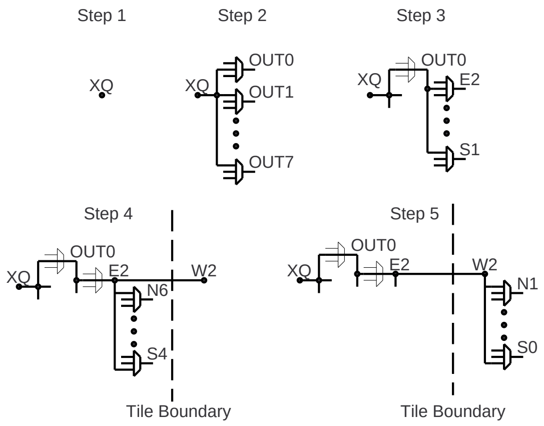

Fig. 3. Step by step creation of a route.

3.3 Routing Utilization Numbers

Table 1 presents several designs that fill up a Virtex device. The metric used in the table refers to the number of wire segments used compared to the total number of wire segments on the entire chip. Another commonly used metric counts the number of used switches as a percent of the total number of switches. With the JBits wire database representation, the former is the more appropriate method as it sheds more light on exactly how much benefit the greedy on demand loading provides.

Table 1. This shows, for several applications, the amount of routing resources that are used as a percent of the total number of wires available.

	Device	Routing Utilization	Logic Utilization
Rijndael	XCV1000	40.9%	99%
Smith-Watermann	XCV1000	15.1%	99%
Firewall	XCV1000	18.4%	95%

As shown in Table 1, the routing intensive Rijndael application [9] utilizes only 41 percent of the wire segments. The other applications utilize less and are representative of common designs. As a result, the Java on demand class loading creates memory savings since not all the wire objects need to be loaded for each application.

3.4 Extra Processing Cycles

While there is an obvious benefit in storing wire database information in tile specific files, this means of data representation is not complete. The main piece missing from the stored data is the inter-tile connectivity, which is an integral part of device specific circuit graphs. Figures 2 and 3 show the inter-tile connectivity represented in the JBits wire database as a segment. Since commercial FPGAs lack homogeneous tile arrays, storing inter-tile connectivity statically would require a device dependent flat graph approach. Instead we chose to generate this information at run-time in order to save on storage and memory utilization. As a result, the JBits wire database requires extra processing cycles not needed in the flat graph approach.

JBits attempts to provide the ability to handle these extra processing cycles in a number of ways. A cache can be enabled with a specific size so commonly used resources will not require segment regeneration for each occurrence of a tile wire segment. The application has complete control over the size of the cache and whether or not it is enabled, providing the user with the power to match the application to the resources of the system. Additionally, there may be a point in small parts where the flat graph memory requirements more closely match those of the ideas presented in this paper. In this case, the JBits API in no way prevents an implementation using a device dependent flat graph approach from being inserted in place of the wire database described in this paper. The analysis on where this breakpoint occurs is left as future work.

4 Application Examples

This section examines some applications that can benefit from automatic run-time routers to show the usefulness of enabling dynamic routing in run-time systems. Since these systems require low memory overhead in the underlying structures to become practical, these examples reinforce the need for a low memory wire database solution.

4.1 Defect Test

Device testing is an important field. Whether it is quality assurance from the manufacturer or diagnostic tests in the field, device testing is necessary to ensure a working chip. The JBits wire database has the ability to specify the exact resources to use for a given net [6]. Combining full control of wire resources with run-time control of the FPGA routes allows for programmatic control of fault detection techniques. As a result, a programmer can create automated fault isolation tools.

4.2 Defect Tolerance

After isolating a fault at run-time, the next possible step using the wire database would be to tolerate these defects [7]. Since the developer has complete control over each and every wire, once a defect is found, the control program can mark specific wires as bad and remove them from JRoute's list of usable wires. As a result, the defect can be ignored and routed around at run-time by JRoute.

4.3 RCAM

Another application that demonstrates run-time routing is the run-time reconfigurable content addressable memory (RCAM) [5]. The RCAM makes use of the run-time router to change the priority encoder. JRoute changes priorities by unrouting and rerouting the nets from the match unit to the priority encoder. Since these routes are modified at run-time, benefits in area reduction and clock frequency can be obtained in the resulting circuit.

4.4 Debugging

Debugging FPGA designs form an interesting class of debugging techniques. One technique to observe internal FPGA signals is to instrument the design with extra circuitry that routes a signal to an unused IOB or to extra circuitry implementing a logic analyzer. Typically this is done by adding in the extra debug circuitry at design time [11]. However, Graham, et al. have demonstrated that adding in the extra circuitry at run-time is possible with the use of JRoute [8].

4.5 RTPCores

Run-time parameterizable (RTP) [4] cores provide a way to instantiate high level circuit descriptions with a parameter determined at run-time. RTP cores can be added, removed or relocated at run-time. Because of these needs, a run-time router is required to perform dynamic modification of routes.

4.6 Partial Configuration

One topic that has received much attention is how to partition designs to make use of partial configuration. The need to specify areas of the device that cannot be used by the tools for placement or routing is required. This is not an optimal solution as it reduces the number of routing resources available. However, when using a dynamic router, like JRoute, a partitioning tool can avoid the step of constraining routes from a partitioned area[10]. For situations where the run-time overhead of auto-routing is not possible, another solution is to avoid using specific wires instead of every wire in a given area. The JBits wire database and JRoute provide the ability to mark wires as used so that the run-time router avoids specific wires without requiring the developer to completely avoid all routes within a specific region. The configurable blocks can be swapped in and out with the internal routing fully specified since the static part of the circuit does not use those routing resources.

5 Comparison to Other Run-Time Approaches

This section contrasts a number of different approaches to solving the memory problem associated with the single graph database in run-time routing. Depending upon the application requirements for memory, time to switch route matrices and route versatility, any one of these methods can be considered the better choice. The approach described in this paper focuses on reducing memory overhead, maintaining automatic run-time routing versatility and fast communications between the route points, but at the expense of the speed of interconnect switching. Other approaches have different tradeoffs.

Brebner and Donlin present multiple techniques for addressing run-time routing issues in the context of Swappable Logic Units (SLUs) [12]. In one approach, the SLUs are connected via a parallel harness resulting in fast communication and low memory overhead, but limited route versatility and slow switchiing speeds. The parallel harness, for example, can be organized in a 2D systolic array and then switched to a hypercube later. The main intent, however, is for the parallel harness arrangement to change infrequently.

The Ultimate RISC (URISC) [13][14] provides an additional solution to this run-time routing dilemma. The URISC has one instruction that moves memory to memory allowing a software controlled routing program to move data. In the context of the SLU, the hardware would have a sea of SLUs with IO registers mapped into the FURI memory map. Routing would occur when the software control program moves data from the memory of one SLU to another. This method has some memory overhead and high route versatility, but the tradeoff results in slow switching and slow communication.

Eggers et. al. [15] created a crossbar on an FPGA fabric for an ultrasonic probe. The methodology used for dynamically modifying the crossbar was very similar to another technique presented in [12]. The run-time routing software is created specifically for the target application. This method has good switching speeds, minimal memory overhead, fast communication speeds but limited versatility. If additional routes are required, the user will have to modify the run-time routing program and the crossbar circuit to handle the new routes.

6 Conclusions

The JBits wire database reduces memory and storage requirements using a few different methodologies. While the idea of representing wires in a device independent manner decreases both storage and memory needs, additional benefit is realized by storing the tile dependent wires as objects separate from other wires resident in the same tile. This combination, of tile based graphs and greedy Java class loading, can lead to significant paring of both storage and memory consumption.

This reduction in memory aids dynamic routers by allowing them to have full access to and complete control of routing resources in a run-time environment. While methodologies for enabling run-time routing already exist, the techniques described in this paper focus on enabling generic and automatic routing as opposed to speed. This benefits applications requiring these full and automatic features and not necessarily applications requiring low latency changes to the bitstream.

7 Future Work

As future work, a detailed analysis of the memory usage numbers by the JBits wire database is necessary. Additionally, a comparison of different programming language implementations of JBits would be beneficial. For example, a C/C++ version of JBits could compare the advantages of C/C++ speed and memory utilization versus Java's ability to do on demand loading of the wire classes. Finally, an analysis of possible benefit of these wire database ideas to traditional static routers is desired.

References

1. E. Keller, "JRoute: A Run-Time Routing API for FPGA Hardware," *7th Reconfigurable Architectures Workshop*, Lecture Notes in Computer Science 1800, pp 874-881, Cancun, Mexico, May, 2000.
2. S. A. Guccione and D. Levi, "XBI: A Java-based interface to FPGA hardware," *Configurable Computing Technology and its uses in High Performance Computing, DSP and Systems Engineering*, Proc. SPIE Photonics East, J. Schewel (Ed.), SPIE - The International Society for Optical Engineering, Bellingham, WA, November 1998.
3. Xilinx, Inc., *The Programmable Logic Data Book*, 1999.
4. S. Guccione and D. Levi, "Run-Time Parameterizable Cores," *Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications,* Lecture Notes in Computer Science 1673, pp 215-222, 1999.
5. P.B. James-Roxby and D.J. Downs, "An Efficient Content-Addressable Memory using Dynamic Routing," *2001 IEEE Symposium on Field-Programmable Custom Computing Machines*, Rohnert Park, California April 29 - May 2, 2001
6. Steven Guccione, Scott McMillan, and Prasanna Sundarajaran, "Testing FPGA Devices using JBits," 4^{th} Military and Aerospace Programmable Logic Devices (MAPLD) International Conference, Laurel, Maryland, September 2001.
7. Prasanna Sundarajaran and Steven Guccione, "Run-Time Defect Tolerance using JBits," *ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays* (FPGA 2001), Monterey, CA, February 2001, pp 193-198.
8. Paul Graham, Brent Nelson, and Brad Hutchings, "Instrumenting Bitstreams for Debugging FPGA Circuits," *2001 IEEE Symposium on Field-Programmable Custom Computing Machines*, Rohnert Park, California April 29 - May 2, 2001.
9. Scott McMillan and Cameron Patterson, "JBits Implementations of the Advanced Encryption Standard (Rijndael)," *Proceedings of the the 11th International Workshop on Field-Programmable Logic and Applications*, Lecture Notes in Computer Science 2147, pp 162-171, 2001.
10. Phil B. James-Roxby and Daniel J. Downs, "Cores and Anti-Cores: Using JBits as Part of a Mainstream Design Flow," *8th Reconfigurable Architectures Workshop*, San Francisco, CA, April 27, 2001.
11. Xilinx, Inc. *ChipScope Software and ILA Cores User Manual*, v 4.0, October 2001.
12. Gordon Brebner and Adam Donlin. "Runtime Reconfigurable Routing," *5th Reconfigurable Architectures Workshop*, Lecture Notes in Computer Science 1388, 1998.
13. Jones, "The Ultimate RISC," Computer Architecture News 16 3, June 1988, pp 48-55.
14. A. Donlin, "Self Modifying Circuitry - A Platform for Tractable Virtual Circuitry," *Proceedings of the Eighth International Workshop on Field Programmable Logic and Applications*, Lecture Notes in Computer Science 1482, pp 199-208, 1998.
15. Fast Reconfigurable Crossbar Switching in FPGAs, H. Eggers, P. Lysaght, H. Dick, G. McGregor, "Fast Reconfigurable Crossbar Switching in FPGAs," *Proceedings of the the 6th International Workshop on Field-Programmable Logic and Applications*, Lecture Notes in Computer Science 1142, pp 297-306, 1996.
16. T. F. Smith and M.S. Waterman. 1981. Identification of common molecular subsequences. Journal of Molecular Biology 147: 195-197.