Adaptive Fault Recovery for Networked Reconfigurable Systems

Weifeng Xu, Ramshankar Ramanarayanan, and Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003
{wxu, tessier}@ecs.umass.edu

Abstract

The device-level size and complexity of reconfigurable architectures makes fault tolerance an important concern in system design. In this paper, we introduce a fully-automated fault recovery system for networked systems which contain FPGAs. If a fault is detected that can not be addressed locally, fault information is transferred to a reconfiguration server. *Following design recompilation to avoid the fault, a new FPGA configuration is returned to the remote system and computation is reinitiated. To illustrate the benefit of this approach, we have implemented a complete fault recovery system which requires no manual intervention. An important part of the system is a timing-driven incremental router for Xilinx Virtex devices. This router is directly interfaced to Xilinx JBits and uses no CAD tools from the standard Xilinx Alliance tool flow. Our completed system has been applied to three benchmark designs and exhibits complete fault recovery in up to 12× less time than the standard incremental Xilinx PAR flow.*

1. Introduction

As the application space of digital systems has grown over the past decade to include a spectrum of applications across science and engineering, a strong desire for operational fault tolerance has developed. Although mission-critical systems frequently contain device-level redundancy to allow for continued operation in the presence of operational faults, other systems, with limited space for redundant devices, require on-line fault recovery to return components to functionality quickly and efficiently. Even though digital components, such as FPGAs, contain *internal* data path redundancy that could be harnessed to limit the effects of a single or small number of transient or permanent hardware faults, to date few *system-level* approaches that follow this model have been developed. For embedded systems, the development of an automated FPGA fault diagnosis and recovery system has been hampered by a limited amount of local computational resources. The recent availability of network access to computationally-superior resources has, in general, not been effectively leveraged for FPGA fault recovery.

To recover from detected faults, FPGA logic and routing configurations can be dynamically restructured to perform the same functional operations while avoiding faulty subcircuits [19]. The plentiful logic and routing resources found in contemporary FPGAs allow for reconfiguration around faulty lookup tables (LUTs), tracks, and connection points. Often, individual LUTs in a logic cluster are left unused during initial logic packing due to routing constraints. An unused LUT can be used in place of a faulty LUT by swapping intra-cluster functionality [19]. For routing resources, a faulty track or connection point can be avoided by rerouting the connection using previously unused resources. This process may involve ripping up and rerouting nets unaffected by the fault to create a feasible-route path. From a practical standpoint, logic and routing fault recovery must be timing-driven to preserve initial mapping performance. Additionally, FPGA recompilation for embedded systems must be performed quickly (e.g. seconds) to minimize system down time. Although current commercial FPGA CAD tools allow for incremental design placement and routing, the time required to perform these operations and to perform subsequent bitstream generation is often prohibitive (e.g. minutes).

In this paper, a fully-automated fault recovery system for networked FPGA systems is described. Our integrated recovery system harnesses the power of networked computing, including the Internet, to allow faulty systems (fault tolerant clients) access to computationally-superior processing resources (reconfiguration servers). These servers aid in the recovery effort of rebuilding a configuration bitstream. Following reconstruction of programming data, a new configuration is passed to a fault tolerant client via the network environment and remote computation is restarted. We have implemented this complete system using an FPGA-based

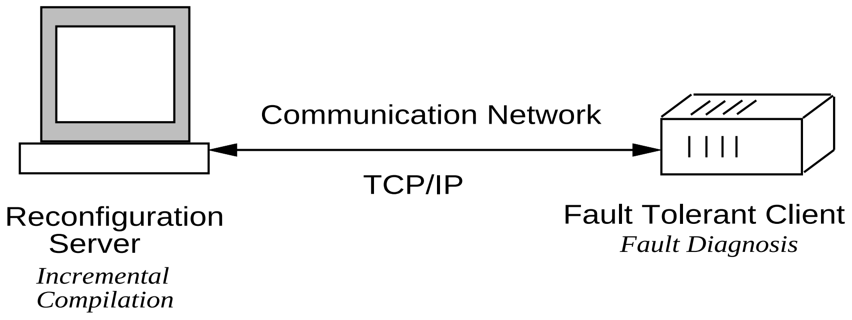

Figure 1. Networked Fault Tolerant System

card in a Windows NT-based PC (fault tolerant client) and a Sun Solaris-based workstation (reconfiguration server). To speed the recompilation effort, a self-contained CAD mapping system targeting commercial Xilinx Virtex devices has been developed. This system uses FlowMap [6] for technology mapping, a modified version of VPR [2] for place and route, and the JBits bitstream generator [11]. A timing-driven incremental FPGA router has been developed and integrated into VPR to overcome permanent faults in the FPGA routing fabric. Our approach is shown to allow for complete recompilation in less than 15 seconds for three benchmark applications. Our custom Virtex recompilation flow is shown to be nearly an order of magnitude faster on average than the standard Virtex place, route, and bitstream generation flow.

The remainder of this paper is organized as follows. In Section 2, previous work related to fault recovery in reconfigurable systems is described. In Section 3, an overview of our experimental approach and system philosophy is provided. A complete, detailed description of our automated fault recovery system is given in Section 4. In Section 5, the incremental CAD environment used to support fault recovery is described. In Section 6, limitations of our current system are described. Experimental results validating our approach are show in Section 7. The paper concludes in Section 8 with a summary of conclusions and an overview of future work.

2. Related Work

Although a number of individual FPGA fault recovery tools involving incremental placement [8] [13] and incremental routing [9] [19] have been developed, little work has been performed to integrate them into a complete, automated fault recovery system. In Yu et al. [31], a fault recovery approach for single faults in a triple modular redundancy (TMR) system is detailed. State information from replicated hardware is transferred to the faulty component to continue operation. A similar approach, described in [32], trades area for fault coverage by reducing a TMR system to a dual modular redundancy system upon fault discovery. In Huang and McCluskey [17], an embedded, dual FPGA system is described. If one FPGA is determined to be faulty, the remaining FPGA attempts to return it to full functionality by shifting a column of logic blocks away from the faulty area. In Sinha et al. [23], performance is traded for fault tolerance in the PipeRench reconfigurable architecture by including built-in self test (BIST) circuitry in the architecture control circuitry. Unlike our approach, these previous efforts require additional system hardware to support fault tolerance. In Saxena et al. [21], a full fault recovery system is described including a multi-threaded processor, FPGA, memory and an I/O interface. Although the architecture appears to have the capability to automatically detect and recover from faults, a physical implementation of the system was not developed.

Network-based FPGA reconfiguration has been attempted in a number of recent projects. In Guccione et al. [12], an FPGA in a remote system takes the place of a standard network interface. The FPGA is capable of downloading new services and upgrades from the network. In Staicu et al. [24], a centralized job management system for reconfigurable computing systems is described. These systems are reconfigured over a network using job scheduling. A series of standard job management systems are analyzed. In Fallside and Smith [10], an FPGA-based system is directly connected to the Internet. Both FPGA configuration and application data are transfered to the FPGA via the network. None of these networked systems explicitly examines fault tolerance as a reconfiguration goal.

3. System Overview

A key to the successful implementation of an operational fault recovery system is access to computational resources that are sufficient to implement the recovery approach. A high-level model of a networked fault recovery system appears in Figure 1. The characteristics of each resource in the system can be described as follows:

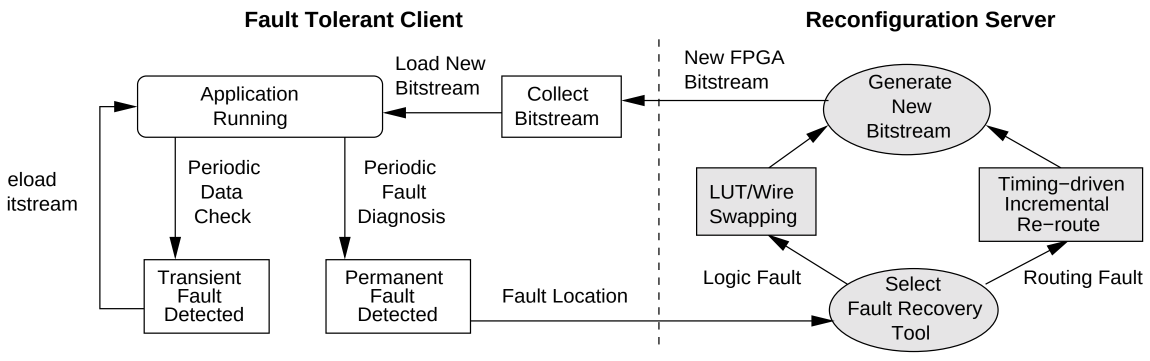

Figure 2. Fault Diagnosis and Recovery Steps

Fault tolerant client: This platform has a need for either uninterrupted system operation or minimized down-time if a hardware fault is detected. For fault tolerant systems with hard real-time constraints, hardware redundancy allows for periodic functional hardware test and fault recovery of system components with no system down-time. This constraint often means complete hardware redundancy of critical subsystems such as FPGAs and associated microprocessors and memories that might be used to configure them. For systems that can tolerate some system down-time, hardware redundancy is not needed to recover a faulty FPGA if the microprocessor and memories can continue functioning normally during the recovery effort and down time can be minimized. For FPGA-based systems, previously described techniques [15] [16] can be used to periodically diagnose permanent FPGA logic and interconnect faults. FPGA configuration SRAM bit transient faults (e.g. single SRAM configuration bit inversion) are successfully addressed by locally reloading the entire configuration bitstream into the FPGA. Due to a lack of local compute power, permanent logic and interconnect faults require the automatic transfer of the fault location to a computationally-superior reconfiguration server for fault recovery.

Reconfiguration server: This platform has the capability to perform recompilation of FPGA circuit designs. Information related to a detected fault is transferred over the network to this host system via standard communication protocols. For FPGA devices, incremental compilation approaches based on logic cluster LUT-swapping and incremental routing are used to generate replacement bitstreams. If incremental compilation fails, complete device re-place and re-route is required.

Communication network: Communication between the fault tolerant client and the reconfiguration server is performed using standard TCP/IP protocols. Transport level processes on both systems provide the underlying communication mechanisms needed for status and configuration transfer.

A detailed flow of client-server interactions appears in Figure 2. For many computing platforms, and especially for time-critical systems, it is highly desirable to perform fault recovery in seconds rather than minutes or hours. As a result, complete device re-place and re-route is only considered as an absolute last option. The diagnosis and recovery effort for FPGA faults at the fault tolerant client is broken into three main parts: the diagnosis of the fault at the fault tolerant client and the transfer of fault information to the host workstation, the recompilation of the embedded program at the reconfiguration server to avoid the detected fault, and the transfer of the updated configuration bitstream information back to the fault tolerant client. Total roundtrip time for this fault recovery process should be a few seconds in most cases to minimize system down time. Transient faults are handled locally by reloading the bitstream into the FPGA.

4. Fault Diagnosis and Recovery System

Although our remote-processing approach could be implemented using dedicated inter-system network links, a shared-network based approach provides for needed system flexibility and scalability. To enable networked communication, a TCP/IP interface is used to allow for communication between a single host workstation server and the fault tolerant client. Although not included in experimentation, additional fault tolerant clients could be added to the system.

The hardware used to implement our fault recovery system includes:

- **Fault-tolerant client** - A TransTech DM11 signal processing board [27] containing a Xilinx XCV100-5 FPGA, a 200 MHz TMS320C6201, 32 Mbytes SDRAM, and 1 MB SRAM serves as the fault-tolerant client for this project. Although it is envisioned that some deployed fault tolerant systems would have a direct connection to the Internet, for this project the PCI-based DM11 was situated in a Windows NT-based PC.

Only the networking capabilities of the PC were used in the implementation of the networked fault tolerant system since the PC did not have sufficient processing or memory resources to perform FPGA incremental compilation.

- **Reconfiguration server** - A 440 MHz Solaris-based Ultra-10 Sun workstation with 768 MB main memory serves as the reconfiguration server. This server implements all initial and incremental CAD algorithms on the FPGA design and supports Internet based communication with the fault tolerant client

- **Communication network** - A 100 Mbit/s Ethernet connection between the fault tolerant client and the reconfiguration server is used to transport network data. During experimentation this shared connection was simultaneously loaded with normal laboratory data traffic.

4.1. System Initiation

Prior to normal system operation, a series of boot-up operations are required at both the reconfiguration server and the fault tolerant client. Upon server boot-up, a fault recovery process is initiated to continually evaluate client operating status. This server process communicates with the fault tolerant clients via a socket-based interface. In our implementation, a socket constructor from the *ActivePerl 5.8.0 INET* library [5] is used to set the socket and associated port. The reconfiguration server swaps processes to service alternate jobs until receiving a request from a remote fault tolerant client.

Upon fault tolerant client boot up, an initial FPGA configuration is loaded into the local FPGA and the WinNT *ActivePerl 5.8.0 INET* library [5] is used to form a socket connection from the client to the reconfiguration server. After bootup, a client process can provide periodic fault tests on the FPGA [16]. Since full FPGA fault diagnosis has not yet been implemented in our system, random FPGA fault locations generated by the local processor were used for system testing purposes.

4.2. Client-Server Interaction

Following client and server initiation, the fault recovery network connection remains dormant until the discovery of a permanent FPGA fault at the client. Upon fault detection, a series of steps are performed to promote fault recovery.

1. The location of the fault is transferred to the reconfiguration server via the pre-constructed TCP/IP client-server socket interface. This information includes the X, Y channel position and track number for routing channel faults and the position of the internal wire/transistor for logic cluster faults. The client-based data transfer utility enables this transfer using the local TCP/IP transport utility with the preassigned network socket.

2. Upon receipt of fault information from the fault tolerant client, the appropriate FPGA fault recovery tool (logic cluster LUT swapper or timing-driven incremental router) is initiated by the reconfiguration server. When a suitable replacement configuration bitstream has been generated, it is transferred to the fault tolerant client. Specific FPGA incremental routing techniques are detailed in Section 5.

3. The configuration manager process of the fault tolerant client receives the replacement configuration information and reprograms the FPGA configuration bitstream.

4. As a final step, FPGA execution in the fault tolerant client is reinitiated.

In some cases, due to the need to avoid faults, the fault-recovered FPGA configuration may not allow for design operation at the same clock speed as the original configuration. As a result, additional hardware requirements on the fault tolerant client, such as the presence of programmable system clock circuitry, may be needed. This latter feature has not yet been implemented in our system.

The fault recovery system operates without manual intervention. A system operator may be notified of a recovery effort via email or some other notification process. In cases of client performance reduction, it may be desirable to replace client hardware, if possible. Note that our automated fault response approach is similar in nature to the approach taken by fault tolerant computer manufacturers years ago [29]. For these systems, malfunctioning fault tolerant hardware would directly contact the manufacturer via a modem to report a specific problem. Replacement hardware was then shipped to the customer without their prior knowledge of a computing problem.

5. Reconfiguration Server CAD Infrastructure

To facilitate high-speed FPGA fault recovery, a customized incremental FPGA CAD approach was deployed at the reconfiguration server. For many fault tolerant systems, fault recovery should occur as quickly as possible, preferably in a matter of seconds. Although network-based data transport generally meets this requirement, incremental FPGA placement and routing using commercial FPGA

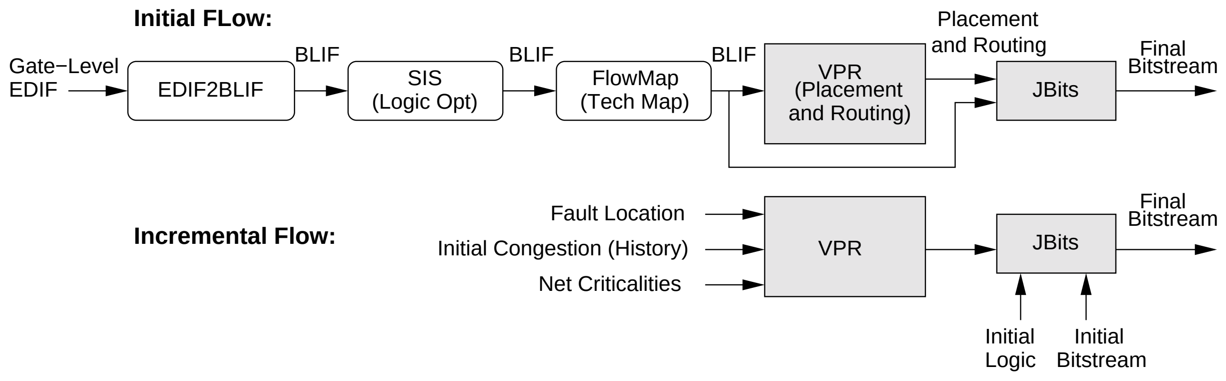

Figure 3. Academic FPGA Design Mapping Flow to Xilinx Virtex Devices

tools may take several minutes in the best case due to several factors including:

- A substantial portion of commercial FPGA tool start-up time is spent in the creation of a meticulously-detailed logic and routing graph. This graph must be sufficiently detailed to support all the technology mapping, placement, and routing features of the tool (e.g. block memory interfacing, phase-locked loop support, several levels of timing analysis) even if they are not used for a specific function, such as incremental placement and routing.

- Commercial FPGA tools provide very detailed error-checking mechanisms and hooks for a variety of user-accessible options. These features can delay the start-up time of the tool.

- Most commercial FPGA tools utilize an intermediate format (e.g. Xilinx NCD format) in performing intermediate CAD steps. The conversion from this format to the configuration bitstream is often lengthened by additional formatting and error-checking steps.

Additionally, commercial tools generally do not provide a time-efficient interface for users to feed results of non-commercial CAD tools into the error-checking and bit generation process.

To overcome these limitations, we have developed a complete *incremental* gate-level design mapping flow for Xilinx Virtex that is made **solely** from academic FPGA tools and the Xilinx JBits bitstream generator [11]. Since we found through experimentation that information from the initial design compilation is helpful in performing incremental compilation, we have also developed a complete *initial* design mapping flow that only uses academic tools and Xilinx JBits. A diagram of the flow for both initial and incremental compilation is shown in Figure 3. For designs originating at the RTL level, Altera MaxPlus2 [1] was used to convert designs to the gate level.

The tools listed for the two flows in Figure 3 include:

- **EDIF2BLIF** - This netlist tool converts a netlist in EDIF netlist format to BLIF netlist format [20].

- **SIS** - This well-known logic optimization tool [22] is used to convert arbitrary gate representations into minimized sum-of-product form.

- **FlowMap** - This academic technology mapping tool [6] packs logic gates into four-input lookup tables using a timing-driven cut-based mapping approach.

- **VPR** - The Versatile Place and Route (T-VPACK/VPR) tool suite [2] provides LUT packing into logic clusters, cluster placement, and inter-cluster routing for island-style FPGA architectures. As described in Section 5.1, significant effort was required to modify VPR to target a specific Virtex device.

- **VPR-to-JBits interface** - This Java-based interface parses logic, placement and routing information from FlowMap and VPR and converts the information into a series of Java calls for the JBits bitstream generator.

- **JBits** - This Java-based tool from Xilinx [11] creates a new Virtex bitstream or modifies an existing Virtex bitstream.

5.1. VPR Modifications to Support Xilinx Virtex

As shown in Figure 4, the VPR place and route tool flow targets FPGAs which represent a generalized FPGA model [3]. The relative run-time speedup of the tool compared to commercial FPGA place and route tools is largely attributable to the high-level definition of a specific FPGA logic and routing *tile* which is replicated in two dimensions to form a two dimensional grid [3]. Although it was possible to retain the VPR model and its associated speed advantage for our Virtex-based work, it was necessary to integrate several Virtex-specific features into VPR:

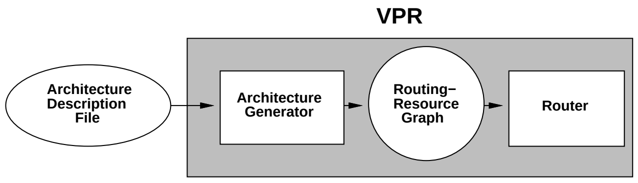

Figure 4. A concise VPR architecture description is converted to a detailed routing graph [3]

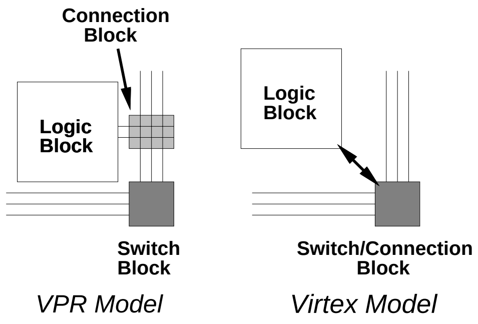

Figure 5. VPR and Virtex FPGA Tiles

1. The standard VPR model assumes a basic *switch box/connection box* model [4] that has been used in FPGA research for the past decade. As shown in Figure 5, for Virtex, these blocks are combined into a single structure with irregular connection patterns, making it more difficult to define wire-to-wire connections succinctly. VPR was modified to include the more recent interconnection model.

2. The VPR model assumes that internal logic cluster connectivity is very regular (e.g. all cluster inputs drive all LUT inputs and all LUT/FF outputs feed back to all LUT inputs). These connections are sparsely populated inside Virtex clusters (CLBs) requiring an extension of the VPR routing graph into each cluster.

3. Virtex routing tracks that span multiple clusters change their *relative* track position (e.g. they shift up one track in the channel) after passing each cluster to facilitate layout and to provide a breadth of connectivity to clusters and other routing tracks. This (X, Y) position-dependent connectivity for tracks was added to VPR.

Although not shown in Figure 3, to validate the accuracy of our modified VPR for Virtex, an interface was created between the generated design logic, placement, and routing files and Xilinx PAR using the Xilinx XDL interface. This interface allows users to generate Xilinx Alliance internal format files (NCD) for error-checking purposes. Each configuration that was read into PAR was shown to meet all error-checking and simulation requirements. Note that the XDL interface was used only for checking; JBits was used for system bitstream generation.[1]

5.2. Timing-Driven Incremental Routing

While previously-described LUT-swapping approaches [19] can be used to recover from intra-cluster logic faults, a new *timing-driven* incremental router was needed to overcome permanent routing track and interconnect transistor faults in the interconnection fabric. To date, few effective incremental routing techniques have been developed. In Dutt and Verma [28], a channel-based refit algorithm based on incremental global, then detailed, routing is described. In Emmert and Bhatia [9], a re-route approach for FPGAs is described that re-routes nets following logic block movement. This approach does not consider the removal of previously-routed nets that block fault recovery. The JRoute router [18] is also limited by this issue. Although JRoute allows for individual net routing that could be used to overcome single interconnect faults, the lack of a net rip-up mechanism makes its use impractical for densely-routed designs. In Lakamraju and Tessier [19], a routability-based version of the PathFinder algorithm [7] was implemented to overcome faults in an island-style architecture. None of these approaches offer timing-driven incremental routing.

In the event of interconnect segment failure or if logic cluster exchange approaches are ineffective, incremental routing techniques are needed to re-connect nets affected by faults. To maximize net routability, we have based our timing-driven incremental router on a timing-driven version of the PathFinder negotiated congestion algorithm, a widely-used maze-routing algorithm. PathFinder is a multi-iteration maze router that re-routes each net in sequence for each iteration. The routing search for each net evaluates a series of routing nodes (cluster pins and intra- and inter-cluster wire segments) each of which has been assigned a node cost value, c_n, based on the following equation [4, 7]:

$$c_n = Crit(i,j) \times delay(n) + [1 - Crit(i,j)] \times (1 + H_n) * (1 + P_n) \quad (1)$$

where $delay(n)$ is the delay of the node, P_n is the *present* cost of the node, based on the number of nets currently assigned to the node, and H_n is a *history* cost value indicating that a node, while perhaps uncongested presently, was overused during one or more previous iterations. $Crit(i,j)$ is the criticality of a net segment (i,j) targeted to a node defined as:

$$Crit(i,j) = 1 - \frac{slack(i,j)}{D_{max}} \quad (2)$$

[1]All VPR for Virtex and associated files are available for free on-line at: http://www-unix.ecs.umass.edu/˜wxu/jbits/ .

```
R: Design nets to be re-routed.
MaxIter: Maximum re-route iterations.
Iter: Current iteration number.

Remove faulty nodes from routing graph
Initialize $H_n$ and $Crit(i,j)$ to values from initial route.
Add nets affected by faults to R.
While nets in R $> 0$ and Iter $<$ MaxIter
    Order nets by criticality.
    For each unrouted net
        Maze-route net using node values $H_n$, $P_n$.
        Update $P_n$ values.
    Endfor
    Update $H_n$ values
    Remove nets with no overused nodes from R.
    Add nets with congested nodes to R.
EndWhile
```

Figure 6. Timing-Driven Incremental Routing Algorithm

where D_{max} is the critical path delay and $slack(i,j)$ is the delay slack of the critical path net segment (i,j) [4]. By performing multiple iterations with a non-decreasing history value, shortest-path net routes can be guided away from congested device areas to areas with available routing resources.

Our timing-driven incremental router extends the original PathFinder approach by using history and criticality values *from the initial route* to guide incremental re-route. Additionally, nets unaffected by operational faults may be ripped up to remove blockages that may inhibit routing for fault-affected nets. Given the small number of reroutes required, we found that the periodic run-time update of criticality $Crit(i,j)$ values during incremental re-route was unnecessary.

The outer-loop of the routing algorithm used to re-route nets over multiple iterations is shown in Figure 6. Unlike traditional PathFinder formulations, in our formulation only a subset of nets are re-routed in each iteration. Following each iteration, nets associated with overused nodes are designated for rip-up, history values are updated, and ripped-up nets are re-routed in the following iteration. We have found a maximum iteration count of about 30 to be appropriate in determining success or failure of re-routing. To achieve accelerated routing speed, an A* search parameter was added to the PathFinder cost function, as in [19] [25] and [26], to promote depth-first search behavior without the loss of routing quality. The route sequence for sources and sinks of individual nets, indicated as **Maze-route** in Figure 6, is described in detail in [26].

6. System Limitations

The effectiveness of our fault recovery approach is limited by the ability of our system to locate specific permanent interconnect and logic faults in an FPGA device. Although a fully-functional fault diagnosis system has not yet been implemented for Virtex XCV100-5 devices, initial results indicate the approximate amount of time required to fully diagnose faults in the device.

In previous work [15] [16], it was shown that the presence of an interconnect fault can be detected (fault testing) and the specific location of a fault can be determined (fault diagnosis) for generic cluster-based architectures. Both test and diagnosis procedures are performed using a similar approach. A portion of the FPGA device that is known to be working is configured to generate test patterns and to collect results for a specific FPGA tile under test. FPGA reconfiguration is used to generate multiple test patterns for each tile and to evaluate multiple tiles. In general, it was shown that fault diagnosis time is about three times longer than fault test time for a given FPGA device.

Recently, the fault test approach outlined in [16] was modified [14] to target commercial Virtex devices. Through experimentation using a Virtex-based board, it has been determined that a full fault test of all FPGA single-length interconnect tracks, including required reconfigurations, takes about 1.5 seconds. Based on our previous work [16] and these initial results, we estimate that a full fault test of all FPGA logic and interconnect in a Virtex XCV100-5 will require about 2 seconds and a full fault diagnosis will require about 6 seconds. Note that fault diagnosis is only necessary if the fault test fails.

7. Results

To judge the performance of our fault recovery system, three benchmark circuits, listed in Table 1, were used. These designs were synthesized from RTL to gate-level form using Altera MaxPlus2 and then mapped using the flow shown in Figure 3. All designs were successfully mapped to the Virtex XCV100-5 on the TransTech DM11 board using our new initial and incremental design flow. $FIR16$ is a 16-tap, 16-bit FIR filter. TEA is an implementation of the Tiny Encryption Algorithm [30], a low-overhead encryption approach. $FMUL$ contains two 8-bit IEEE floating point multipliers. Each Virtex CLB contains four LUT/FF pairs. The XCV100-5 contains a total of 600 CLBs. The minimum cycle times (T_{min}) in Table 1 indicate

Design	CLBs	LUTs	FFs	I/Os	T_{min}
FIR16	465	1859	256	32	127 ns
TEA	109	433	72	32	39 ns
FMUL	227	902	126	32	96 ns

Table 1. Benchmark Statistics

Task	Time (s)		
	FIR16	TEA	FMUL
Client-to-server transfer	0.001	0.001	0.001
Server preprocessing	0.001	0.001	0.001
Incremental CAD	13	12	12
Server-to-client transfer	0.03	0.03	0.03
Client reconfiguration	0.47	0.47	0.47
Total	14	13	13

Table 2. Total time to perform networked fault recovery

the best-possible performance achieved during initial-pass mapping with our VPR-based flow.

The total time for system recovery from single-error faults is shown in Table 2. Incremental CAD was performed on a 440 MHz Ultra 10 Sun Workstation with 768 MB of memory. These values are the average of fifty random single-error recoveries initiated by the fault tolerant client. In all cases it was possible to recover from the single fault and maintain initial-map design performance with our new VPR-based tool set (modified VPR version 4.30). As expected, the incremental CAD portion of the flow required most of the recovery effort time.

The timing-driven incremental routing portion of the recovery effort using our new VPR-based flow is significantly faster than re-routing each design from scratch or attempting to use the automated incremental reroute capabilities in Xilinx PAR (Alliance version 2.1). As shown in Table 3 for incremental single fault recovery, our incremental VPR-based approach is up to 12$\times$ faster than Xilinx PAR. For the Xilinx PAR experiments, the faulty track was designated as unusable in a Xilinx NCD file. The PAR router was then run to reconnect the affected net. The initial VPR design placement and routing files were read into the XDL interface for these experiments. The time necessary to designate the fault in the NCD file or to read files from the XDL interface were not included in the Xilinx total. The VPR-based incremental route approach was deployed as shown in Figure 3. The amount of time required to place and route designs from scratch using both PAR and VPR is provided for reference. Note that the fault recovery time required for each design (about 12 seconds each) is only slightly larger than the estimated combined fault test time (2 seconds) and fault diagnosis time (6 seconds) described in Section 6.

Experimental testing of our system has demonstrated a number of additional points:

- The timing-driven incremental router was used to reroute designs with route fault counts ranging from 1 to 50. Although all faults could be overcome, incremental route time increased from 8 to 10s for the three designs as the number of faults increased. Note that the emergence of 50 permanent routing faults in a previously fault-free device is unlikely to occur in practice.
- For route fault counts ranging from 1 to 50, initial-design performance could be maintained for fault counts up to 14. Design performance degradation ranged from an average of 1% to 5% for fault counts between 15 and 50.
- Once bitstream generation was complete, the entire, new configuration bitstream (about 100 KB) was returned to the fault tolerant client. JBits analysis shows that for each fault about 24 configuration bits are changed, on average. This analysis indicates that in congested network settings it may be more advantageous to merge the netlist changes at the client rather than at the server.
- It was determined that the use of initial-route history values and criticality values for timing-driven incremental routing reduced the overall route time by about 30%.

8. Future Work

The work outlined in this paper is an attempt to automate the fault recovery process for remote FPGA systems. Once a fault is detected, fault information is transfered over a network to a computationally-powerful reconfiguration server. Following reconfigurable design recovery, a replacement bitstream is returned to the fault tolerant system via the network. A necessary part of this system was the development of an academic-tool based FPGA mapping tool set for Xilinx Virtex devices. The use of these tools gave us the flexibility and compilation speed needed to complete single fault recovery for FPGA designs with no manual intervention.

A number of improvements are required to enhance system functionality. A complete Virtex fault diagnosis tool is currently being tested. The functionality of the reconfiguration server needs to be expanded to allow for the simultaneous service of multiple clients. The depth-optimal version of FlowMap has not yet been integrated into our tool flow, limiting the possible design performance achieved by our VPR-based place and route flow for Virtex.

Time (s)	FIR16		TEA		FMUL	
	Custom	Xilinx	Custom	Xilinx	Custom	Xilinx
Incremental Routing	8	152	8	43	8	76
Bitstream Generation	3	15	2	11	2	15
Promgen	2	2	2	2	2	2
Total	**13**	**168**	**12**	**56**	**12**	**93**
From-scratch	67	170	32	65	22	102

Table 3. Task time for timing-driven incremental routing and bitstream generation

9. Acknowledgments

This work was sponsored by National Science Foundation grants CCR-0081405 and CCR-9988238 and with support from Xilinx Corporation. The authors wish to acknowledge the efforts of Eric Sondhi, Andrew Laffely, Srini Krishnamoorthy, Ramaswamy Ramaswamy, Vibhor Garg, Dereck Fernandes, and Ian Harris in the completion of this project.

References

[1] Altera Corporation. *MaxPlus2 Users Guide*, 2002. http://www.altera.com/.

[2] V. Betz and J. Rose. VPR: A New Packing, Placement, and Routing Tool for FPGA Research. In *Proceedings, International Conference on Field Programmable Logic and Applications*, pages 213–222, Oxford, UK, Sept. 1997.

[3] V. Betz and J. Rose. Automatic Generation of FPGA Routing Architectures from High-Level Descriptions. In *Proceedings, ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 175–184, Monterey, CA, Feb. 2000.

[4] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Boston, MA, 1999.

[5] M. Brown. *ActivePerl Developer's Guide*. McGraw Hill, New York, N.Y., 2000.

[6] J. Cong and Y. Ding. FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs. *IEEE Transactions on Computer-Aided Design*, 13(1):1–12, Jan. 1994.

[7] C. Ebeling, L. McMurchie, S. Hauck, and S. Burns. Placement and Routing Tools for the Tryptych FPGA. *IEEE Transactions on VLSI Systems*, 3(4):473–482, Dec. 1995.

[8] J. M. Emmert and D. Bhatia. Partial Reconfiguration of FPGA Mapped Designs with Applications to Fault Tolerance and Reconfigurable Computing. In *Proceedings, International Conference on Field Programmable Logic and Applications*, pages 141–150, Oxford, England, Sept. 1997.

[9] J. M. Emmert and D. Bhatia. Incremental Routing in FPGAs. In *Proceedings, IEEE International ASIC Conference*, pages 302–305, Sept. 1998.

[10] H. Fallside and M. Smith. Internet Connected FPL. In *Proceedings, International Conference on Field Programmable Logic and Applications*, pages 48–57, Villach, Austria, September 2000.

[11] S. Guccione, D. Levi, and P. Sundararajan. A Java-Based Interface for Reconfigurable Computing. In *Proceedings, Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference*, Sept. 1999.

[12] S. Guccione, D. Verkest, and I. Bolsens. Design Technology for Networked Reconfigurable FPGA Platforms. In *Proceedings, Design, Automation and Test in Europe*, pages 994–997, March 2002.

[13] F. Hanchek and S. Dutt. Methodologies for Tolerating Cell and Interconnect Faults in FPGAs. *IEEE Transactions on Computers*, 47(1):15–32, Jan. 1998.

[14] I. G. Harris. Personal communication. University of Massachusetts, Amherst.

[15] I. G. Harris and R. Tessier. Diagnosis of Interconnect Faults in Cluster-Based FPGA Architectures. In *Proceedings, IEEE/ACM International Conference on Computer-Aided Design*, pages 472–475, San Jose, CA, Nov. 2000.

[16] I. G. Harris and R. Tessier. Testing and Diagnosis of Interconnect Faults in Cluster-Based FPGA Architectures. *IEEE Transactions on Computer-Aided Design of Electronic Systems*, 21(11):1337–1343, Nov. 2002.

[17] W.-J. Huang and E. J. McCluskey. Column-Based Precompiled Configuration Techniques for FPGA Fault Tolerance. In *Proceedings, IEEE Symposium on Field-Programmable Custom Computing Machines*, Rohnert Park, CA, April 2001.

[18] E. Keller. JRoute: A Run-Time Routing API for FPGA Hardware. In *Proceedings, 7th Reconfigurable Architecture Workshop*, pages 874–881, Cancun, Mexico, May 2000.

[19] V. Lakamraju and R. Tessier. Tolerating Operational Faults in Cluster-Based FPGAs. In *Proceedings, ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 187–194, Monterey, CA, Feb. 2000.

[20] P. Leventis. Using Edif2Blif, Version 1.0. *University of Toronto, Department of Electrical Engineering Technical Report*, June 1998.

[21] N. Saxena, S. Fernandez-Gomez, W.-J. Huang, S. Mitra, S.-Y. Yu, and E. J. McCluskey. Dependable Computing and Online Testing in Adaptive and Configurable Systems. *IEEE Design and Test of Computers*, 17(1):29–41, Jan. 2000.

[22] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. *SIS: A System for Sequential Circuit Analysis*. Memorandum No. UCB/ERL M92/41, Electronics Research Laboratory, University of California, Berkeley, May 1992.

[23] S. Sinha, P. M. Kamarchik, and S. C. Goldstein. Tunable Fault Tolerance for Runtime Reconfigurable Architectures. In *Proceedings, IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 185–194, Napa, CA, April 2000.

[24] A. V. Staicu, J. R. Radzikowski, K. Gaj, N. Alexandridis, and T. El-Ghazawi. Effective Use of Networked Reconfigurable Resources. In *Proceedings, Military Applications of Programmable Logic Devices*, Laurel, MD, Sept. 2001.

[25] J. Swartz, V. Betz, and J. Rose. A Fast Routability-Driven Router for FPGAs. In *6th International Workshop on Field-Programmable Gate Arrays*, Monterey, Ca, Feb. 1998.

[26] R. Tessier. Negotiated A* Routing for FPGAs. In *Proceedings: Fifth Canadian Workshop on Field-Programmable Devices*, pages 14–19, Montreal, Quebec, June 1998.

[27] TransTech Corporation. *DM11 Data Sheet*, 2000.

[28] V. Verma and S. Dutt. A Search-Based Bump-and-Refit Approach to Incremental Routing for ECO Applications in FPGAs. In *Proceedings, IEEE/ACM International Conference on Computer-Aided Design*, pages 141–151, San Jose, CA, Nov. 2001.

[29] S. Webber and J. Beirne. The Stratus Architecture. In *Proceedings, 21st International Symposium on Fault-Tolerant Computing*, Montreal, Quebec, June 1991.

[30] D. Wheeler and R. Needham. TEA: A Tiny Encryption Algorithm. In *Proceedings, Fast Software Encryption: Second International Workshop*, pages 363–366, Leuven, Belgium, Dec. 1994.

[31] S.-Y. Yu and E. J. McCluskey. On-line Testing and Recovery in TMR Systems for Real-Time Application. In *Proceedings, IEEE International Test Conference*, pages 240–249, Baltimore, MD, Oct. 2001.

[32] S.-Y. Yu and E. J. McCluskey. Permanent Fault Repair for FPGAs through Graceful Degradation. In *Proceedings, IEEE International Conference on Dependable Systems and Networks Fast Abstracts*, Goteborg, Sweden, July 2001.