

IRS - A portable Interface for Reconfigurable Systems

Torsten Mehlan, Jochen Strunk, Torsten Hoefler, Frank Mietke, Wolfgang Rehm

Chair of Computer Architecture
Chemnitz University of Technology

{tome,sjoc,htor,mief,rehm}@cs.tu-chemnitz.de

Abstract

This work introduces the definition of a new Interface for Reconfigurable Systems (IRS). Since modern FPGAs support dynamic and partial reconfiguration, new core modules may be configured during normal operation. User applications are expected to benefit from the support of current FPGA hardware. To access this hardware there is a need for a well-defined interface. This paper proposes an Application Programming Interface (API) that meets the requirements of fast hardware access, portability and ease of use. Moreover the work explains how to implement the services for a maximum of portability.

1. Introduction

Modern FPGA devices like Xilinx Virtex-II Pro and Xilinx Virtex-4 provide the capability for partial reconfiguration. Users may download a hardware design to the FPGA without affecting other configurations on the chip. As long as the hardware still has resources available any core-modules can be configured. Other parts of the FPGA retain their configuration and continue normal work. This new capability provides services similar to other resources of traditional workstations and computers. Those resources comprise RAM, CPU time and network bandwidth. Thus the well-known usage model of conventional computing resources now can be extended to configurable hardware. To access partially reconfigurable FPGAs a user interface becomes necessary.

This work introduces an interface definition and the fundamental concepts of resource administration to support the proposed service. All design decisions underline the need for portability, performance and system security. The interface is implemented for Linux but can easily be ported to any other popular operating system.

2. Related Work

There are many efforts to enable the effective use of FPGAs in High Performance Computing. The proprietary products of Mitrion [6] provide functions similar to the proposed interface. Users are enabled to generate core modules from high-level descriptions of their algorithms. However, the administration of those modules and the interface are part of the integrated development environment. Thus, this approach is not suitable for a general framework for accessing FPGA resources by applications.

To enable typical users of large computing resources to leverage FPGAs there has to be some way to generate the appropriate core modules. Usually there is no experience in writing register transfer level (RTL) descriptions of the algorithms. Thus, many efforts are taken to synthesise high-level descriptions to the RTL level. Gupta et al. present a synthesis framework in [5]. This framework suits for applying optimization techniques to the synthesis of RTL VHDL code from algorithms written in C. A number of other works also deal with different optimization approaches for handling high-level descriptions such as [7], [2], [8].

In [11] the authors discuss a technique to determine optimized placements for dynamically reconfigurable FPGAs. Dynamically changing the configuration during execution may allow for resource sharing. Thus, the usage model of those hardware becomes even more close to the well known sharing of CPU time. More insight about the concept of time-multiplexed FPGAs is given in [10] and [9].

3. User Interface

The user interface has to meet the requirements of full functionality, simplicity, performance and portability. Moreover there should be a possibility to grant access rights in a fine grained manner. Thus the registration and use of hardware modules is separated. Once a hardware module is registered users with sufficient rights can use it until deregistration. Usually a privileged user cares for registering modules.

These tasks do not impose special performance requirements, but data transmission to and from the application needs to be as fast as possible. Thus, the interface maps the registers of the hardware module to the virtual address space of the process. Using this approach access to memory belonging to the module is directed to bus transactions to the FPGA. Once the address translation is cached in some TLB (Translation Lookaside Buffer) there is no overhead in communication. Furthermore, virtual memory management is implemented in almost every modern operating system. This approach delivers much more performance and portability than the use of packet based data transfers. Processing of headers and footers may consume CPU cycles where the approach of memory mapping does not need to perform these tasks. Conventional network communication benefits from mapping communication buffers into the virtual address space of user processes as well. The works [1], [3] and [4] discuss the advantages of this approach more deeply in the context of network communication.

Since data processing at the FPGA occurs concurrently to the computation at the CPU, the user application is likely to benefit from an interrupt mechanism. Alternatively, the application can read a register of the hardware module in a loop. This approach of busy waiting may impact the performance of the code especially if more than one compute intensive process competes for CPU time. The proposed interface to the FPGA hardware provides an interrupt registration mechanism. The user selects an interrupt number of the appropriate hardware module and specifies a handler function to be executed when the interrupt is signaled.

All functions of the interface are listed below (The return code "int" is not displayed):

- `IRS_ModuleRegister(char *filename, unsigned long *key, unsigned int flags)`
- `IRS_ModuleUnregister (unsigned long ModKey, unsigned int flags)`
- `IRS_ModuleOpen (unsigned long ModuleKey, unsigned long *ModuleID, unsigned int flags)`
- `IRS_ModuleClose (unsigned int ModuleID, unsigned int flags)`
- `IRS_ModuleAddress (unsigned int ModuleID, unsigned long **ModuleUserAddress)`
- `IRS_ModuleInterruptHandler (unsigned int ModuleID, int InterruptNumber, IRS_INTR_CALLBACK callback)`
- `IRS_ModuleInterruptState (unsigned int ModuleID, int InterruptNumber, int state)`

- `IRS_ErrorString (int ErrorNumber, char *buffer)`

Modules become available through a call to `IRS_ModuleRegister`. Users can configure the FPGA with registered modules using the function `IRS_ModuleOpen` and the appropriate identification key. This operation returns an individual module ID that is used to refer to this particular module instance. The application requests the start address of the mapped hardware registers by calling the function `IRS_ModuleAddress`. Finally interrupt handlers may be registered using the function `IRS_ModuleInterruptHandler`.

4. Concept of FPGA administration

The functions of the API have to be implemented at different levels. User applications need a conventional library that contains user level code and serves as wrapper to the appropriate system calls. All tasks that possibly affect system integrity have to be accomplished in a privileged context. Thus, an extension of the operating system has to care for the most critical parts of FPGA reconfiguration and user authentication. The current implementation uses a Linux kernel module to support those tasks. Some functions being less critical can also be done by a daemon process running with different privileges than the user application.

4.1. Kernel Module

The operating system extension of the IRS interface benefits from the ability to dynamically load modules into the kernel of current operating systems. The IRS kernel extension authenticates requests from user processes. Thus, only the application that requested a particular hardware configuration is allowed to access the associated registers and interrupts. Moreover, the kernel module is responsible for the configuration of the FPGA and low level interrupt handling. Two distinct kernel interfaces provide access to those services. The first interface is directly used by the IRS user level library to request module registration and configuration as well as address mapping and interrupt handler registration. The second one serves as interface to the module agent.

Any request for the configuration of a specific hardware module results in a kernel transition. The kernel module validates the parameters and sends a request to the module agent running as a daemon process. The module agent is aware of the configuration of user accounts, access rights to specific modules and quotas. After checking user permissions the module agent either sends a deny of the request to the kernel or acknowledges the request and sends the configuration bit stream belonging to the appropriate mod-

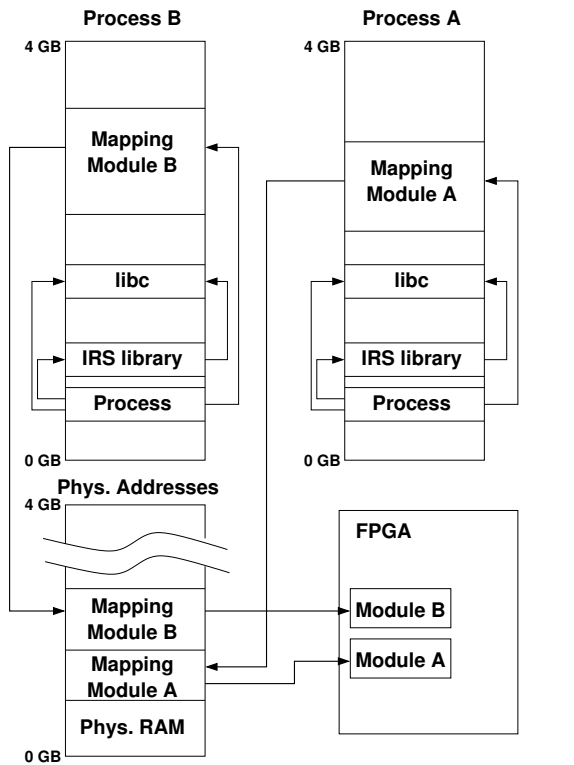


Figure 1. Mapping of the hardware interfaces to the virtual address space of processes

ule. Finally the kernel takes the configuration from the user space and performs the actual configuration of the FPGA.

The kernel module also performs the mapping of the hardware module registers into the virtual address space of the user application. Therefore it implements a memory mapping function that maps the physical address of the appropriate register space. After validation of the address parameter the page tables of the process are adjusted and the virtual address is returned. The entire interface of the hardware module has to reside in a contiguous area of memory. The size of the mapping and the physical address are maintained by the kernel module, thus, allowing for the check of the user supplied parameters. Usually the bus addresses of the FPGA reside far above the border of physically available RAM. Figure 1 shows the relationships between virtual addresses, physical addresses and FPGA hardware interfaces.

To support the delivery of interrupts to the application the kernel module has to implement waiting for hardware interrupts. The kernel module manages for each interrupt of the FPGA to which application it belongs and whether it was enabled or disabled. When an interrupt is enabled the kernel initializes an object (e.g. an operating system

semaphore) to block the execution of a user level thread. As soon as the associated interrupt occurs the thread wakes up and is scheduled for execution. Thus the application has the ability to react on interrupts of the hardware module.

4.2. Module Agent

The module agent is responsible for the administration of accounts, access permissions to hardware modules and quotas. Each hardware module is associated with a configuration file containing the configuration bit stream and meta-information. For instance some hardware modules may be exclusively available to a particular group of users. The module agent reads the configuration files and manages the availability of resources to users and user groups.

To restrict access to registered hardware modules the module agent manages an access control list. For each hardware module the list of users and groups is given that are allowed to configure the FPGA. Moreover the hardware modules have a size supporting the enforcement of quotas. The module agent is aware of all registered hardware modules and interacts with the kernel part of IRS. It runs as a daemon process and waits for a signal from the kernel to perform the validation of a request. On success the module agent passes the configuration to the kernel and continues sleeping.

4.3. User Library

The user level library can be linked either dynamically or statically. The library serves as front end to the IRS kernel extensions and the IRS daemon process. Though the library never calls the daemon process directly its services are used by the kernel module in response to service requests. The interface provided to the applications comprises the functions described in section 3. Given the presence of appropriate configurations of hardware modules the applications acquire access to modules via a unique key. The processing of any opening request is done by the kernel module through interaction with the module agent daemon. A successful opening operation results in the configuration of the FPGA. Subsequently, the application uses the function `IRS_ModuleAddress` to gain access to the register set of the core module. The application views the hardware interface as contiguous area located in the local address space. Any data structure may be used to cast this interface into an appropriate representation. The registration of an interrupt handler results in actions performed by the library and the kernel. The library has to create a new thread dedicated to the execution of the handler function. The kernel module associates a semaphore with this thread.

The user process resolves all functions, types and constants of IRS during the link step. The functions and types of IRS are selected with respect to portability issues. User

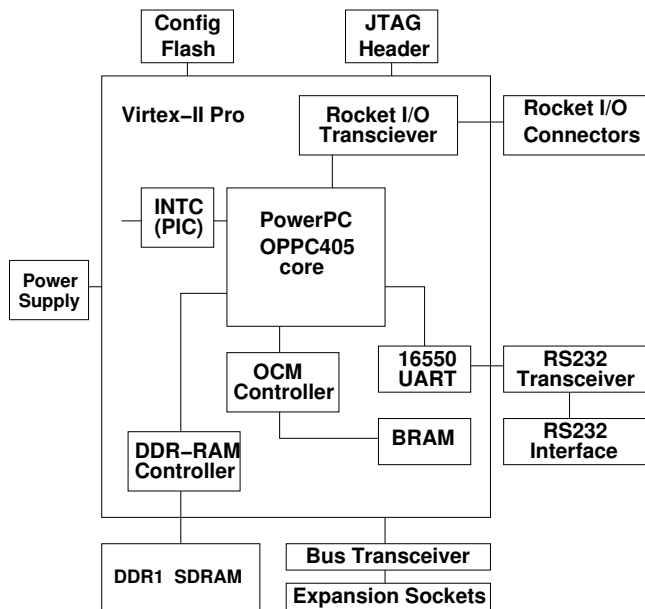


Figure 2. GDx Virtex-II Pro Development Board

applications may be compiled using dynamic linking, and the appropriate association takes place after spawning instances of the program. Usually the underlying interface to the operating system may change without affecting binary applications as long as the definitions of IRS are preserved. This requirement implies the need for a well defined interface supporting today's demands and future developments. We believe that IRS meets all the requirements of stability, functionality and simplicity where user applications can rely on.

5. Proof of Concept

To validate the concepts of IRS we used the FPGA-Board illustrated in figure 2. The FPGA contains a PowerPC CPU core running a Linux operating system. The remaining parts of the FPGA implement necessary infrastructure (e.g. RAM controller, interrupt controller) and are used to provide reconfigurable hardware to user space applications. Among the FPGA there are 64 MByte of DDR1-RAM, a JTAG interface, RS232 converter and Rocket I/O connectors mounted.

The PowerPC core connects to other system components via two different bus architectures. The first bus type is named Processor Local Bus (PLB) and connects directly to the CPU. This bus has 32 bit address width, 64 bit data width and runs at 100 MHz clock frequency. The second

bus is named On-Chip Peripheral Bus (OCB) and needs a bridge to connect to the PLB. All components are either connected to the PLB or the OCB.

This board was connected via a serial cable to an NFS volume containing home directories and a boot image of Linux. To enhance the number of interrupt lines we implemented an additional interrupt controller working behind the standard one. All core modules of user applications are connected to the enhanced interrupt controller supporting 128 interrupt lines. These interrupts are signaled via interrupt 3 of the standard controller.

The kernel support is done by a module exporting the devices `/dev/irs` and `/dev/irsa`. The first one serves as interface to the user level library while the second one interacts with the module agent. Finally, we implemented the privileged module agent as daemon process and the user library. Using this infrastructure we were able to load configuration streams down to the reconfigurable area of the FPGA. Using the advanced interrupt controller the core modules were able to signal completion events to the user application without exhausting the limited resources of the original controller. A small test configuration performing simple mathematical functions showed that the implementation of IRS works as expected.

6. Conclusion

This work presented an interface for accessing dynamically reconfigurable hardware by user applications. New generations of FPGA devices are expected to support partial reconfiguration during operation without affecting other core modules. To exploit this feature from user space there is a need for a reliable interface definition. The proposed interface IRS serves as efficient and portable facility to reconfigurable hardware. This work presented the precise layout of IRS and explains how the appropriate services can be implemented in a portable fashion. Finally a proof of concept was given.

References

- [1] M.A. Blumrich, C. Dubnicki, Edward W. Felten, Kai Li, and M.R. Mesarina. Virtual-Memory-Mapped Network Interfaces. *IEEE Micro*, 15(1):21–28, February 1995.
- [2] L.C.V. dos Santos and J.A.G. Jess. A Reordering Technique for efficient Code Motion. In *Proceedings of the Design Automation Conference*, New Orleans, USA, June 1999. ACM Press.
- [3] C. Dubnicki, Liviu Iftode, Edward W. Felten, and Kai Li. Software Support for Virtual Memory-Mapped Communication. In *Proceedings of the 10th International Parallel Processing Symposium*, Honolulu, Hawaii, USA, April 1996. IEEE Computer Society.

- [4] Edward W. Felten, R.D. Alpert, A. Bilas, M.A. Blumrich, D.W. Clark, S.M. Damianakis, C. Dubnicki, Liviu Iftode, and Kai Li. Early Experience with Message-passing on the SHRIMP Multicomputer. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture (ISCA '96)*, Philadelphia, PA, USA, May 1996. ACM Press.
- [5] Sumit Gupta, Nikil Dutt, Rajesh Gupta, and Alex Nicolau. SPARK: A High-Level Synthesis Framework For Applying Parallelizing Compiler Transformations. In *Proceedings of the International Conference on VLSI Design*, New Delhi, India, January 2003. IEEE Computer Society.
- [6] Mitronics Inc. The Mitrion Software Development Kit. <http://www.mitronics.com/msdk.shtml>.
- [7] I. Radivojevic and F. Brewer. A new symbolic Technique for Control-dependent Scheduling. *IEEE Transactions on Computer Aided Design*, 15(1):45–57, January 1996.
- [8] Minjoong Rim, Yaw Fann, and Rajiv Jain. Global scheduling with code-motions for high-level synthesis applications. *IEEE Transactions on Very Large Scale Integration Systems*, 3(3):379–392, September 1995.
- [9] Steve Trimberger. Scheduling Designs into a Time-Multiplexed FPGA. In *Proceedings of the 1998 ACM/SIGDA sixth International Symposium on Field Programmable Gate Arrays*, Monterey, California, USA, February 1998. ACM Press.
- [10] Steve Trimberger, Dean Carberry, Anders Johnson, and Jennifer Wong. A Time-Multiplexed FPGA. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, USA, April 1997. IEEE Computer Society.
- [11] Guang-Ming Wu, Jai-Ming Lin, and Yao-Wen Chang. An Algorithm for Dynamically Reconfigurable FPGA Placement. In *Proceedings of the International Conference on Computer Design*, Austin, TX, USA, September 2001. IEEE Computer Society.