A self-reconfiguring platform

Brandon Blodget[1], Philip James-Roxby[2], Eric Keller[2], Scott McMillan[1], and Prasanna Sundararajan[1]

[1] Xilinx, 2100 Logic Drive, San Jose, CA, 95124, USA
{brandonb, mcmillan, prasanna}@xilinx.com
[2] Xilinx, 3100 Logic Drive, Longmont, CO, 80503, USA
{jamespb, keller}@xilinx.com

Abstract. A self-reconfiguring platform is reported that enables an FPGA to dynamically reconfigure itself under the control of an embedded microprocessor. This platform has been implemented on Xilinx Virtex II^{tm} and Virtex II Pro^{tm} devices. The platform's hardware architecture has been designed to be lightweight. Two APIs (Application Program Interface) are described which abstract the low level configuration interface. The Xilinx Partial Reconfiguration Toolkit (XPART), the higher level of the two APIs, provides methods for reading and modifying select FPGA resources. It also provides support for relocatable partial bitstreams. The presented self-reconfiguring platform enables embedded applications to take advantage of dynamic partial reconfiguration without requiring external circuitry.

1 Introduction

This paper presents a self-reconfiguring platform (*SRP*) for Xilinx Virtex II^{tm} and Virtex II Pro^{tm} devices[1]. It begins by reviewing the motivation for developing the *SRP*, before presenting the first detailed description of the two core software components, the ICAP API and the Xilinx Partial Reconfiguration Toolkit(XPART). Recent improvements and revisions to the *SRP* hardware architecture are also described in more detail.

Dynamic reconfiguration and self-reconfiguration are two of the more advanced forms of FPGA reconfigurability. Dynamic reconfiguration implies that an active array may be partially reconfigured, while ensuring the correct operation of those active circuits that are not being changed. Self-reconfiguration extends the concept of dynamic reconfigurability. It assumes that specific circuits on the logic array are used to control the reconfiguration of other parts of the FPGA. Clearly the integrity of the control circuits must be guaranteed during reconfiguration, so by definition self-control is a specialized form of dynamic reconfiguration.

Both dynamic reconfiguration and self-reconfiguration rely on an external reconfiguration control interface to boot an FPGA when power is first applied or the device is reset. Once initially configured, self-control requires an internal reconfiguration interface that can be driven by the logic configured on the logic array. On Xilinx Virtex II and Virtex II Pro parts, this interface is called the internal configuration access port (ICAP)[2].

The hardware component of *SRP* is composed of the ICAP, control logic, a small configuration cache, and an embedded processor. The embedded processor can be Xilinx's MicroBlaze^{tm}, which is a 32-bit RISC soft microprocessor core[3]. The hardcore PowerPC on the Virtex II Pro can also be used as the embedded processor. The embedded processor provides intelligent control of device reconfiguration at runtime. The integration of this functionality is especially attractive for embedded systems. This lightweight approach maximizes flexibility while minimizing additional external circuitry.

The software component of *SRP* defines two APIs. The lower level one is the ICAP API. The ICAP API provides access to the configuration cache and controls reading and writing the cache to the device. The higher level API is Xilinx Partial Reconfiguration Toolkit(XPART). XPART is derived from the JBits API work[4]. Like the JBits API it abstracts the bitstream details providing seemingly random access to select FPGA resources. XPART also contains methods for relocating partial bitstreams.

SRP opens up a number of interesting possibilities. Firstly it gives more reconfiguration options to the designer. Adding *SRP* to a system allows an application to get reconfiguration data from any peripheral and partially reconfigure the device. For example if the system has a network connection partial bitstreams could be pulled off the network. Secondly the embedded processor could manipulate the data before reconfiguring the device. This could permit custom encryption or compression of bitstreams. Thirdly the XPART API enables fine grain reconfiguration control over select FPGA resources. This allows tuning of MGTs (multi-gigabit transceiver) , or constant folding achieved by modifying LUTs. Finally it is envisioned that a subsystem like *SRP* could play an important role in an embedded operating system running on a platform FPGA. *SRP* could help the OS manage hardware tasks. It could provide the capability to swap tasks in and out of hardware. It would also allow these tasks to be relocated to different regions on the device[5].

The paper is arranged as follows: Section 2 reviews previous work relating to *SRP*. Section 3 looks at the details of the ICAP and the reconfiguration mechanisms of the Virtex line of FPGAs. This provides the background necessary for an appreciation of the *SRP* hardware and software infrastructure that are described in sections 4 and 5. Sections 5.1 and 5.2 describe the APIs in the software layers. Section 6 presents future work and concludes the paper.

2 Related work

Dynamic reconfiguration implies that an active array may be partially reconfigured, while ensuring the correct operation of those active circuits that are not being changed. Much research has been done on dynamic reconfiguration which shows it can be used to reduce circuit complexity, increase performance and simplify system design[6].

Self-reconfiguration is a special case of dynamic reconfiguration where the configuration control is hosted within the logic array that is being dynamically reconfigured. The part of the array containing the configuration control remains unmodified throughout execution. A formal definition of self-reconfiguration is presented in [7]. There are several desirable features of such an arrangement. Firstly the control logic is as close to the logic array as possible, thus minimizing the latencies associated with accessing the configuration port. Secondly, fewer discrete devices are required, reducing the overall system complexity[8].

For a device to support self-reconfiguration it must be dynamically reconfigurable. A second desirable, but not required, characteristic is the device provides internal access to the configuration port. This way, the configuration stream does not have to exit the chip and use board level resources to gain access to the configuration port. The Xilinx XC6200 family of devices first provided both of these features, newer devices from Xilinx such as the Virtex II and Virtex II Pro families are both dynamically reconfigurable and provide access to the configuration port to internal logic.

The first references to self-reconfiguration using FPGAs in the literature was in [9] where a small amount of static logic is added to a reconfigurable device in order to produce a self-reconfiguring processor. The Flexible URISC[10] defined an abstract model of virtual circuitry that had self-configuring capability. A pattern-matching algorithm was used to investigate the viability of systems that exhibit self-control of reconfiguration management[8]. A number of applications mapped using self-reconfiguration have been shown to improve performance over existing approaches[9][11].

Xilinx application note 662 describes a method for using self-reconfiguration on a Virtex II Pro device to update MGT (multi-gigabit transceiver) parameters[12]. These attributes must be modified to optimize the MGT signal transmission prior to and after a system has been deployed in the field. The work presented in this paper extends the framework described in Xilinx application note 662 to enable a general purpose self-reconfiguring platform.

The XPART software layer is derived from Xilinx's JBits API work. XPART provides a lightweight, minimal set of JBits API features implemented in the C language. XPART also provides some basic functionality for supporting relocatable modules. A relocatable module is a partial bitstream that can be relocated to multiple places on the FPGA. This functionality has also been called Dynamic Hardware Plugins(DHP) and was used for implementing multiple networking applications in hardware for high-performance programmable routers[13]. The methods that XPART provide are very similar to PARBIT (PARtial BItfile Transformer)[14]. The tool JBitsDiff also provides the ability to create relocatable modules directly from bitstreams[15]. The added benefit XPART provides is these functions can run on an embedded processor.

3 Virtex I/II/II Pro Configuration Architecture

SRAM based FPGAs are configured by loading application specific data into configuration memory. All Virtex^{tm} series devices (Virtex I, Virtex II and Virtex II Pro) have their configuration memory segmented into *frames*. These devices are partially reconfigurable and a frame is the smallest unit of reconfiguration. There are multiple frames per CLB column. For example Virtex devices have 48 frames per CLB column. Frames are one bit wide and differ in length depending on the number of CLB rows in the device. For example an XC2V40 has 832 bits per frame and an XC2V1000 has 3392 bits per frame.

Virtex II and Virtex II Pro devices have an internal reconfiguration access port (ICAP) which can be controlled by internal FPGA logic. The ICAP interface is a subset of the SelectMAP^{tm} interface. Figure 1 shows a comparison between the two interfaces. The ICAP interface has fewer signals than the SelectMAP interface because it does not have to do full configurations and it does not have to support different configuration modes. It also differs in that the SelectMAP bi-directional *D* port is split into an *I* data port and an *O* data port. One other item to note is the functionality provided by the ICAP *CE* pin is equivalent to the SelectMAP *CS* pin.

The eight bit data *I* port on the ICAP allows faster reconfiguration than serial modes of reconfiguration. The maximum frequency that the SelectMAP and ICAP interfaces can be clocked at without checking the BUSY signal is 66MHz[16].

Virtex I/II/II Pro FPGAs require a pad frame be added to the end of the configuration data. This pad frame flushes out the reconfiguration pipeline. Therefore to write one frame to the device it is necessary to clock in two frames, the data frame plus a pad frame. Thus the minimal time to reconfigure over ICAP a single XC2V40 frame at 66MHz is 3.2us. It would take 13us to reconfigure a single XC2V1000 frame at 66MHz.

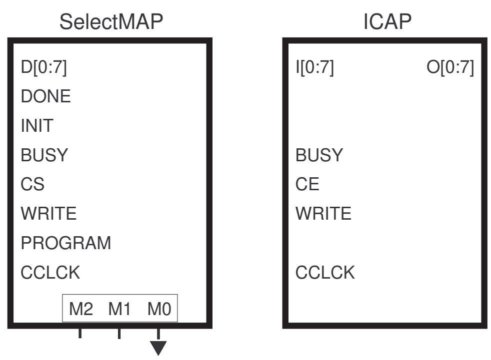

Fig. 1. SelectMap vs ICAP

4 The SRP Hardware Architecture

SRP's hardware component is composed of the ICAP, some control logic, a small configuration cache, and an embedded processor. These peripherals communicate over the CoreConnect^{tm} Open Peripheral Bus (OPB)[17]. Figure 2A shows a block diagram of the current hardware subsystem implementation.

In this implementation a 32 bit register is used to interface to the ICAP port. Figure 2A shows how this register is mapped to the ICAP signals. The control logic for reading and writing data to the ICAP is implemented in a low level software driver. This driver defines methods for reading and writing to the ICAP interface register. There are also methods for reading and writing blocks of data to the ICAP. This methodology is similar to the XVPI register JBits SDK used to access the SelectMAP interface[18].

The BlockRAM (BRAM) shown is Figure 2A is used to cache configuration data. To keep the *SRP* hardware as lightweight as possible only one BRAM is used. One Virtex II BRAM can store 16K bits of data. Since the largest Virtex II Pro device, the XC2VP125, has a frame length of 11K bits, one BRAM can easily store a whole frame of even this largest Virtex II Pro device.

This hardware system has been implemented on both Virtex II and Virtex II Pro devices. On the Virtex II device the MicroBlaze soft processor was used. On Virtex II Pro the embedded PowerPC was targeted. The Xilinx Embedded Developer Kit (EDK) and Xilinx 5.1i ISE tools were used to build these hardware platforms.

The next generation of the hardware system will provide much better performance. Figure 2B shows this system. This will be achieved via a few changes. Firstly the ICAP control logic will move from being in software (hardware drivers) to being implemented directly in hardware. This move also means there will be less communication over the system bus. Secondly, the configuration cache BRAM will be moved inside the ICAP control peripheral. This will allow the dual ported nature of the BRAM to be exploited. One port of the BRAM will be exposed to the system bus. The other port will be accessed by the ICAP control logic. This way the ICAP control logic will not take up system bus cycles to transfer data to and from the configuration cache. The embedded processor will still be able to access the configuration cache BRAM over the system bus. Thirdly, the ICAP control peripheral will be a master peripheral. This way the peripheral will be able to fetch configuration data directly from external memory without requiring the processor be involved. Lastly, we plan to implement the peripheral as a Xilinx IPIF (IP Interface) peripheral. IPIF peripherals can interface to both the OPB and the faster PLB (Processor Local Bus).

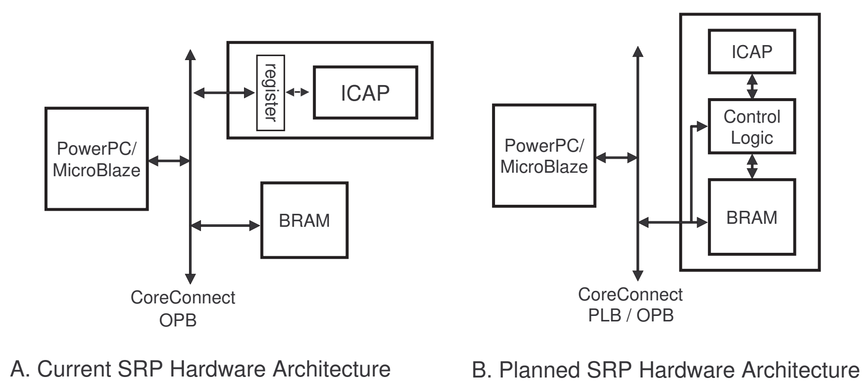

Fig. 2. Curent and Planned SRP Hardware Architecture

5 The SRP software architecture

The software components of *SRP* are designed in layers. Figure 3 shows the different software layers. The software layers are divided into hardware dependent and hardware independent parts. This division is enabled by the ICAP API. This API defines methods for transferring data between the configuration cache implemented in BRAM and the active configuration memory. It also provides methods for accessing the configuration cache.

We have created two implementations of the ICAP API. The first uses the SRP hardware described in the previous section to enable embedded applications to readback or modify the active configuration of the FPGA. The second implementation emulates the active configuration store and the configuration cache entirely in software. This emulated version of the ICAP API can then be compiled for a Windows or Unix workstation.

The Xilinx Partial Reconfiguration Toolkit (XPART) is built on top of the ICAP API. Thus XPART is hardware independent and can be compiled for an embedded system, or for a Windows or Unix system. This portability also applies to applications that use XPART and/or the ICAP API.

One advantage of this portability is one can do a degree of debug on a standard workstation before moving to the target embedded platform. It is also possible to use XPART on a workstation, manipulate partial bitstreams, then write them to the computer's hard drive instead of the physical FPGA device.

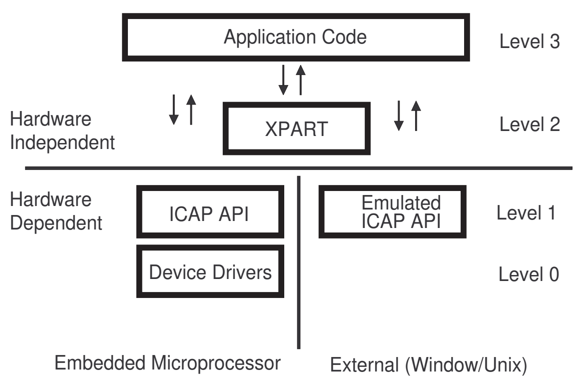

Fig. 3. SRP Software Layers

5.1 The ICAP API

The ICAP API defines methods for accessing configuration logic through the ICAP port. The main methods move data between the configuration cache (BRAM) and the active configuration memory (the device). Other methods allow the processor to read and write to the configuration cache. Finally the setDevice() method indicates which device is being targeted. This method allows the designer to retarget their application to a different device by changing one line in the code. All Virtex II and Virtex II Pro family members are supported. Table 1 gives an outline of the methods in the ICAP API.

Routines	**Description**
setDevice()	Indicates the target device.
storageBufferWrite()	Writes data to the configuration storage buffer
storageBufferRead()	Reads data from the configuration storage buffer
deviceWrite()	Transfers specified number of bytes from storage buffer to ICAP_IN
deviceRead()	Transfers specified number of bytes from ICAP_OUT to storage buffer
deviceAbort()	Aborts the current operation
deviceReadFrame()	Reads one or more frames from the device into the storage buffer
deviceWriteFrame()	Writes one or more frames to the device from the storage buffer
setConfiguration()	Loads a configuration from memory
getConfiguration()	Writes current configuration to memory

Table 1. The ICAP API

5.2 XPART - Xilinx Partial Reconfiguration Toolkit

The Xilinx Partial Reconfiguration Toolkit (XPART) has been built on top of the ICAP API. The purpose of this toolkit is to allow embedded processors to modify resources and relocate modules. Currently this toolkit has four methods. Table 2 gives an overview of these methods.

XPART provides functions for on the fly resource modification. This is done through the getCLBBits() and setCLBBits() methods. These methods abstract the bitstream and provides seemingly random access to selected resources. Currently XPART has the capability to read or modify all Virtex II CLB logic and routing resource. It currently does not have access to IOB, DCM, MGT, BRAM or other non CLB resources.

The strategy employed is to do a read/modify/write of configuration data. The following lists the steps that setCLBBits() would perform to update a resource like a LUT:

1. Calculate the target frame
2. Find LUT bits in target frame
3. read target frame from device and put in storage buffer using deviceReadFrame()
4. Modify the LUT bits in the storage buffer using writeStorageBuffer()
5. Reconfigure the device with new LUT bits using deviceWriteFrame()

The memory elements that define the content of a LUT in Virtex II are all located in one frame. This is not true with all resources. Some resources have their configuration bits scattered across multiple frames. The setCLBBits() and getCLBBits() methods are optimized so they will not have to read or write the same frame twice during the same call. Figure 4 shows sample code that updates the content of a LUT.

```
#include <XPART.h>
#include <LUT.h> /* Bitstream resource library for LUTs */

int main(int argc, char *args[]) {
   char* value;
   int error, i, row, col, slice;
   setDevice(XC2VP7); // Set the device type

   ... [initialize row,col,slice and value to desired values] ...

   error = setCLBBits(row, col, LUT.RES[slice][LE_F_LUT], value, 16);
   return error;
} /* end main() */
```

Fig. 4. Code to update LUT contents

XPART provides some basic functionality for supporting relocatable modules. A relocatable module is a partial bitstream that can be relocated to multiple places on the FPGA. XPART provides two methods for dealing with relocatable modules. The two methods are setCLBModule() and copyCLBModule(). The setCLBModule() method works on regular partial bitstreams that contains information about all of the rows in the included frames. It works by modifying the header of the partial reconfiguration packet. Figure 5A illustrates the setCLBModule() command.

The copyCLBModule() function copies any sized rectangular region of configuration memory and writes it to another location. The copied region contains just a subset of the rows in a frame. This allows the designer to define dynamic regions that have static regions above or below it. Figure 5B shows an illustration of copyCLBModule().

The copyModule() function employees a read/modify/write strategy like the resource modification functions. This technique enables changing select bits in a frame and leaving the others bits to their current configured state. The Virtex II provides glitchless configuration logic, meaning if a bit stays the same between two configurations no glitch will occur. For example a frame may be modified that contains a routing resource. If the value of the bits controlling that routing resource remain the same between the old and new configurations, no glitch will occur on that routing resource.

Routines	**Description**
getCLBBits()	Reads back the state of a selected CLB resource.
setCLBBits()	Reconfigures the state of a selected CLB resource.
setCLBModule()	Place the module at a particular location on the device
copyCLBModule()	Given a bounding box copy it to a new location on the device

Table 2. XPART Methods

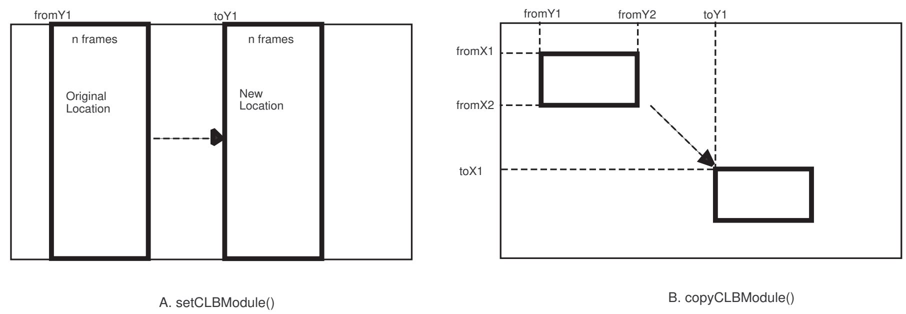

Fig. 5. Illustration of Module Methods

6 Future work and conclusions

Several extensions and applications of the system are in progress. As noted earlier we are working on a more efficient implementation of our *SRP* hardware architecture. In the current implementation the ICAP control logic is implemented in software via device drivers. This implementation requires 10ms to modify a LUT value on an XC2V1000 device with MicroBlaze running at 50MHz. The next version of the *SRP* hardware should allow us to clock configuration data into and out of the device at the maximum no handshake frequency of 66MHz. Modifying one LUT on the XC2V1000 will take 13us for the read, negligible time for the modify, and 13us for the write for a total of approximately 26us. This is over two orders of magnitude faster than the existing *SRP* hardware architecture.

We are currently using *SRP* to develop a reconfiguration controller for a high I/O reconfigurable crossbar switch[19]. The original reconfiguration controller was implemented using external logic and memory. The data for the reconfiguration controller was generated offline using JBits SDK. *SRP* should allow us to implement the reconfigurable crossbar and controller on the same chip.

In conclusion, we have described a self-reconfigurable platform. *SRP* is an intelligent subsystem for lightweight reconfiguration of Xilinx Virtex II and Virtex II Pro FPGAs in embedded systems. The system enables self-reconfiguration under software control within a single FPGA. *SRP* has a layered hardware and software architecture that permits a variety of different interfaces to maximize flexibility and ease-of-use.

References

1. Blodget, B., McMillan, S., Lysaght, P.: A lightweight approach for embedded reconfiguration of fpgas. In: Design, Automation and Test in Europe (DATE03), IEEE (2003) 399–400
2. Xilinx, Inc.: Xilinx 5.1i Libraries Guide. (2002)
3. : Xilinx web site. http://www.xilinx.com/ipcenter/processorcentral/microblaze (2003)

4. Sundararajan, P., Guccione, S.A., Levi, D.: JBits: Java based interface for reconfigurable computing. In: 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies (MAPLD99), Laurel, MD (1999)
5. Nollet, V., Coene, P., Verkest, D., Vernalde, S., Lauwereins, R.: Designing an operating system for a heterogeneous reconfigurable soc. In: RAW'03 workshop. (2003) (Accepted)
6. Compton, K., Hauck, S.: Reconfigurable computing: A survey of systems and software. In: ACM Computing Surveys, Vol 34, No. 2. (2002) 171–210
7. Sidhu, R., Prasanna, V.K.: Efficient metacomputation using self-reconfiguration. In: Field Programmable Logic and Applications (FPL02), Springer (2002) 698–709
8. McGregor, G., Lysaght, P.: Self controlling dynamic reconfiguration: A case study. In: Field Programmable Logic and Applications (FPL99), Springer (1999) 144–154
9. French, P.C., Taylor, R.W.: A self-reconfiguring processor. In: IEEE Symposium on File-Programmable Custom Computing Machines (FCCM93), Napa Valley, California (1993) 50–59
10. Donlin, A.: Self modifying circuitry - a platform for tractable virtual circuitry. In: Field Programmable Logic and Applications (FPL98), Springer (1998) 199–208
11. Sidhu, R.P.S., Mei, A., Prasanna, V.K.: Genetic programming using self-reconfigurable fpgas. In: Field Programmable Logic and Applications (FPL99), Springer (1999)
12. Eck, V., Kalra, P., LeBlanc, R., McManus, J.: In-circuit partial reconfiguration of RocketIO attributes. Xilinx Application Note XAPP662, version 1.0, Xilinx, Inc. (2003)
13. David E. Taylor, Jonathan S. Turner, J.W.L.: Dynamic hardware plugins (DHP): exploiting reconfigurable hardware for high-performance programmable routers. In: Open architectures and Network Programming Proceedings, IEEE (2001) 25–34
14. Horta, E., Lockwood, J.W.: PARBIT: a tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (FPGAs). Technical Report WUCS-01-13, Washington University in Saint Louis, Department of Computer Science (July 6, 2001)
15. James-Roxby, P., Guccione, S.A.: Automated extraction of run-time parameterisable cores from programmable device configurations. In: IEEE Symposium on File-Programmable Custom Computing Machines (FCCM00), Napa Valley, California, IEEE Computer Society (2000) 153–161
16. Xilinx, Inc.: Virtex-II Platform FPGA User Guide. (2002)
17. : IBM web site. http://www.chips.ibm.com/products/coreconnect (2003)
18. Sundararajan, P., Guccione, S.A.: XVPI: A portable hardware/software interface for virtex. In: Reconfigurable Technology: FPGAs for Computing and Applications II, Proc. SPIE 4212, SPIE – The International Society for Optical Engineering (2000) 90–95
19. Young, S., Alfke, P., Fewer, C., McMillan, S., Blodget, B., Levi, D.: A high i/o reconfigurable crossbar switch. In: IEEE Symposium on File-Programmable Custom Computing Machines (FCCM03), Napa Valley, California, IEEE Computer Society (2003)
20. Carmichael, C.: Virtex FPGA series configuration and readback. Xilinx Application Note XAPP138, version 1.1, Xilinx, Inc. (1999)
21. Lim, D., Peattie, M.: Two flows for partial reconfiguration: Module based or small bit manipulations. Xilinx Application Note XAPP290, version 1.0, Xilinx, Inc. (2002)
22. McMillan, S., Guccione, S.A.: Partial run-time reconfiguration using JRTR. In: Field Programmable Logic and Applications (FPL00), Springer (2000) 352–360
23. Sidhu, R., Prasanna, V.K.: Fast regular expression matching using fpgas. In: IEEE Symposium on File-Programmable Custom Computing Machines (FCCM01), Rohnert Park, California (2001)
24. Horta, E.L., Lockwood, J.W., Taylor, D.E., Parlour, D.: Dynamic hardware plugins in an FPGA with partial run-time reconfiguration. In: Design Automation Conference (DAC), New Orleans, LA (2002)