A Hardware Virtual Machine for the Networked Reconfiguration

Yajun Ha[1*], Patrick Schaumont[1], Marc Engels[1], Serge Vernalde[1],
Freddy Potargent[1], Luc Rijnders[1], Hugo De Man[1*]
[1]IMEC, Kapeldreef 75, Leuven 3001, Belgium
[*]Department of Electrical Engineering, K. U. Leuven, Leuven 3001, Belgium
yjha@imec.be

Abstract

Networked reconfiguration is an enabling technology for cost effective service deployment and maintenance. A hardware virtual machine to enable this networked reconfiguration is presented. An abstract FPGA model is the core of such a hardware virtual machine. Based on this abstract FPGA model, the traditional implementation flow of FPGA has been separated into two parts: the service provider's side and client's side. We show how to split the FPGA design flow to enable this networked reconfiguration and present first results in the efficiency of the bytefiles for a hardware virtual machine.

1. Introduction

It is expected that in the coming years, more and more new services will be developed and offered by the service providers to their clients. Although this is predicted to be a huge market, some enabling technologies are needed to reduce the business cost in new service deployment and maintenance. *Networked reconfiguration* [1] is one of such technologies. In the networked reconfiguration, as shown in Fig.1, the service deployment and maintenance are done in an easy way. Whenever one client wants to receive a new service, he will send a specific service request to the service provider. Once the service provider receives this request, it will send both hardware reconfiguration information and service data to the client. By preparing the reconfiguration information in a hardware bytecode format (which is abstract enough to be implemented on a wide variety of FPGA platforms, just like what Java bytecode [2] does for software), the service provider does not need to know on which kinds of reconfigurable resources (e.g., which series of FPGAs from which vendors) his hardware bytecode will be implemented . This hardware bytecode is transported from the service provider to the clients via the network. With the reconfiguration information contained in the hardware bytecode, the client is able to do customized data processing by first dynamically reconfiguring itself. The dynamic reconfiguration is achieved through the reprogramming of Programmable Logic Devices (PLDs) like FPGAs.

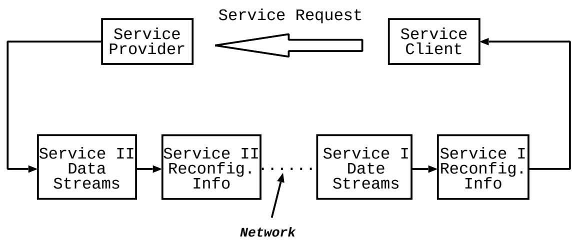


Figure 1. Networked Reconfiguration

To support the networked way of hardware reconfiguration mentioned above, the traditional FPGA implementation flow as shown in Fig. 2 has to be changed. In the networked reconfiguration, the application description and FPGA reconfiguration are no longer done in the same geographic site. Instead, service application is developed in the service provider's side, while the FPGA implementation is done in the client's side. This feature influences the FPGA mapping tools. To fully support FPGA implementation in the networking context, FPGA mapping tools have to do a tradeoff in their realization.

There are three options for the FPGA mapping tools. Firstly, the mapping tools can be totally put in the service provider's side. This is what Xilinx Online [3] adopted. It is easy to implement in theory, but very troublesome in maintenance. Since the service providers cannot limit and know what type of FPGA their respective client will use, they should maintain a large amount of FPGA CAD tools from different FPGA vendors, and create bitstream whenever a new client uses a new FPGA. As a second choice, the mapping tools can be totally put in the client's side. It is an easy way both for implementation and maintenance (from the point of view of the service provider), but too expensive for the terminals. Finally, as a third choice, mapping tools can be separated into two parts, partially in the service provider's side (Map_S), and partially in the client's side (Map_C). We call the Map_C block as *hardware virtual machine* (HVM), while Map_S block as *HVM-compiler*. As the benefits of this approach, the service providers only need to maintain a single or few FPGA CAD tools to distribute and update their new services, while at the same time, the client does a reasonable portion of the mapping task.

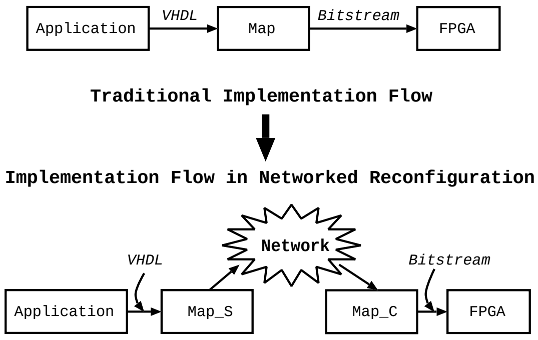

Application
VHDL
Map
Bitstream
FPGA
Traditional Implementation Flow
Implementation Flow in Networked Reconfiguration
Network
VHDL
Bitstream
Application
Map_S
Map_C
FPGA

Figure 2. Separation of Traditional FPGA Implementation Flow to Achieve Universal Networked Reconfiguration

This paper discusses the implementation of a hardware virtual machine for the networked reconfiguration. In the next section, a revised FPGA implementation flow to utilize the hardware virtual machine is reviewed. An abstract FPGA model is the core of the hardware virtual machine, and it works as the bridge between service provider and client. In section 3, the architecture details of this abstract FPGA model are described. Besides, reasons to choose some parameters of it are also introduced. In section 4, we will discuss the algorithms for mapping hardware bytecode back to a local FPGA architecture. Finally, experimental results are analyzed in section 5.

2. FPGA Design Flow in Networked Reconfiguration

To implement the networked reconfiguration, the FPGA design flow [4] is modified as in Fig. 3 [1]. The new implementation flow is divided into two parts. The service provider's part and the client's part. Considering the fact that we normally have a powerful service provider but a poor client with limited resources, most of the work load is assigned to the service provider. Although that may influence the efficiency of the whole implementation, it greatly eases the job for the client, and lowers its requirements for computing resources. On the other side, as had mentioned earlier, the mapping task cannot be totally assigned to the service provider, for the easy deployment and maintenance of new services. The implementation flow split-up should consider the compromise of performance for both service providers and clients.

In the service provider's side, an application design is first compiled and technology mapped to the abstract FPGA architecture model which will be discussed in the next section. Then it will be placed and pre-routed on the same model. The resulting physical design information will be written into a hardware bytecode file, and then will be transported via the network to the client. In the client's side, the received bytecode file is converted to the physical design information for that of the local FPGA architecture. The algorithms used to realize part of the converter will be discussed in section 4.

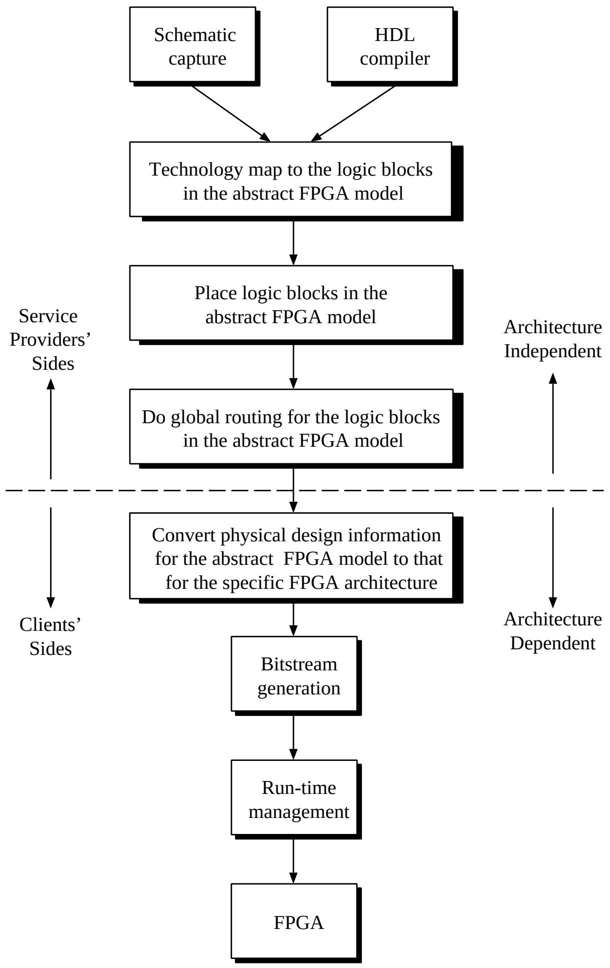

Schematic capture
HDL compiler
Technology map to the logic blocks in the abstract FPGA model
Place logic blocks in the abstract FPGA model
Do global routing for the logic blocks in the abstract FPGA model
Convert physical design information for the abstract FPGA model to that for the specific FPGA architecture
Bitstream generation
Run-time management
FPGA
Service Providers' Sides
Clients' Sides
Architecture Independent
Architecture Dependent

Figure 3. An implementation flow for the FPGA applications in networked reconfiguration

3. Abstract FPGA Model

In the previous section, a modified FPGA design flow has been described. It explains the application context of the hardware virtual machine. In this section, we will focus on the description of an abstract FPGA model, which is the core of the hardware virtual machine. Additionally, hardware bytecode will be introduced. They are the intermediate mapping results on the server side. Implementation of the abstract FPGA model converter is the topic of the next section.

For the definition of the abstract FPGA model, we start from the observation that, all commercial FPGAs are composed of three fundamental components: *logic blocks*, *I/O blocks* and *programmable routing*. A circuit is implemented in an FPGA by programming each of the logic blocks to implement a small portion of the logic required by the circuit, and each of the I/O blocks to act as either an input pad or an output pad, as required by the circuit. The programmable routing is configured to make all the necessary connections between logic blocks and between logic blocks and I/O blocks.

Corresponding to commercial FPGAs, the abstract FPGA model (see Fig. 4) also contains three blocks. They are *abstract logic block*, *abstract routing architecture*, and *abstract I/O pad*.

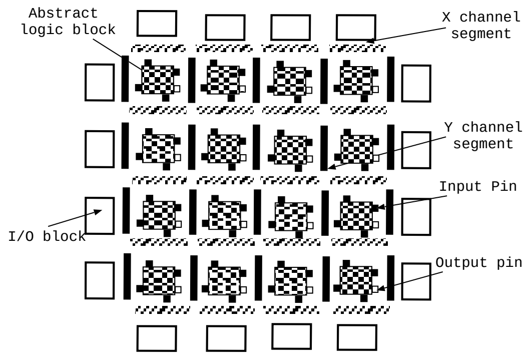


Figure 4. An abstract FPGA architecture

3.1. Abstract Logic Block and I/O Block

The logic block used in an FPGA strongly influences the FPGA speed and area-efficiency. While many different logic blocks have been used in FPGAs, most current commercial FPGAs are using logic blocks based on *look-up tables* (LUTs). Accordingly, in our research, we will only build abstract logic block for those LUT-based FPGAs.

One abstract basic logic block is shown in Fig. 5. It contains a K-input LUT (we choose K=4 in the example) to implement combinational logic, a D Flip-Flop to implement sequential logic, and a multiplexer to choose output [8]. The pin locations of an abstract logic block are chosen randomly, because they will be mapped to the real pin locations of the client FPGA later.

Combination of abstract basic logic blocks can be used to describe a series of commercial logic blocks, such as those used in Altera 8K and 10K FPGAs [5], Xilinx 4000 and 5200 series [4].

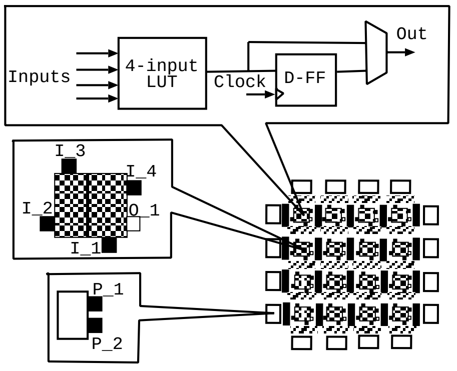


Figure 5. A basic logic block in the abstract FPGA.

Abstract I/O blocks, in many ways, can be treated as a simpler logic block. It comprises M pads (we choose M=2 in the example) which are programmed to be either input or output pad. Similarly as the pins in an abstract logic block, I/O pad locations can be chosen freely.

The position for an abstract logic block or I/O pad is specifically defined in the coordinate system as shown in Fig. 6.

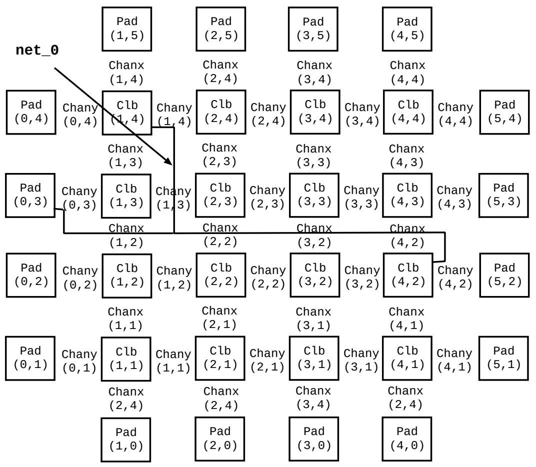


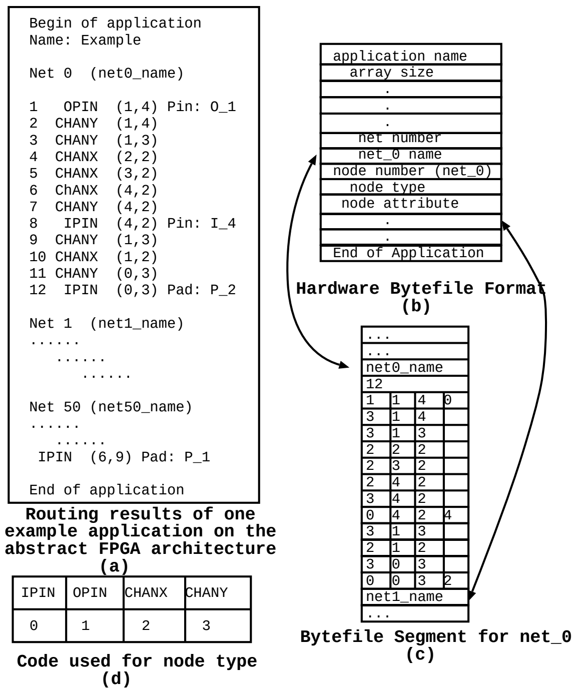
Figure 6. The coordinate system for the abstract FPGA architecture

3.2. Abstract Routing Architecture

Based on their routing architecture, commercial FPGAs can be classified into three groups. The FPGAs of Xilinx, Lucent are *island-style* FPGAs, while Actel's FPGAs are *row-based*, and Altera's FPGAs are *hierarchical*. The routing architecture in our abstract FPGA model can describe all of the three groups. It consists of an array of X channel segments and Y channel segments like in Fig. 4. Any path between two pins or one pin and one pad can be abstractly described by the combination of X and Y channel segments. The position for a X or Y channel is specifically defined in the coordinate system as shown in Fig. 6. Part of Fig. 7 shows an example to describe the *net_0* in Fig. 6.

3.3. Hardware Bytecode Format

Based on the coordinate system and paths that *net_0* use in Fig. 6, the routing information for *net_0* can be described by Fig. 7(a). This text description of routing information is then converted into a binary bytefile segment (Fig. 7(c)) according to the format defined in Fig. 7(b). Fig. 7(d) shows the code used to represent different kinds of routing resources in bytefile.



```
Begin of application
Name: Example

Net 0  (net0_name)

1   OPIN  (1,4) Pin: O_1
2  CHANY  (1,4)
3  CHANY  (1,3)
4  CHANX  (2,2)
5  CHANX  (3,2)
6  ChANX  (4,2)
7  CHANY  (4,2)
8   IPIN  (4,2) Pin: I_4
9  CHANY  (1,3)
10 CHANX  (1,2)
11 CHANY  (0,3)
12  IPIN  (0,3) Pad: P_2

Net 1  (net1_name)
......
   ......
      ......

Net 50 (net50_name)
......
   ......
 IPIN  (6,9) Pad: P_1

End of application
```

Routing results of one example application on the abstract FPGA architecture (a)

application name
array size
.
.
.
net number
net_0 name
node number (net_0)
node type
node attribute
.
.
End of Application

Hardware Bytefile Format (b)

...			
...			
net0_name			
12			
1	1	4	0
3	1	4	
3	1	3	
2	2	2	
2	3	2	
2	4	2	
3	4	2	
0	4	2	4
3	1	3	
2	1	2	
3	0	3	
0	0	3	2
net1_name			
...			

Bytefile Segment for net_0 (c)

IPIN	OPIN	CHANX	CHANY
0	1	2	3

Code used for node type (d)

Figure 7. The hardware bytecode for the abstract FPGA architecture

4. Abstract FPGA Model Converter

The abstract FPGA model converter fulfills the task to interpret abstract hardware bytecode into local FPGA programming files, where the hardware bytecode is received from the service provider via the network. The converter partitions its job into eight steps, *logic block rebinding*, *local routing modification (LRM) for illegal tracks*, *coordinate adjustment*, *pin reassignment*, *local routing modification (LRM) for pins*, *I/O pad reassignment*, *LRM for I/O pad*, and *detailed routing*.

4.1. Logic block rebinding

In the logic block rebinding phase, the abstract FPGA model should be mapped to the local logic block architecture (shown in Fig. 4). Depending on how large the local logic block is, several 4-input logic blocks in Fig. 4 will be combined to make a new big logic block. After this technology mapping, some of the channels disappear in the new architecture. For example, if two abstract logic blocks are combined, there will be one X-channel or one Y-channel removed (see Fig. 8(a)). If four abstract 4-input logic blocks are combined, there will be two X-channel and two Y-channel removed (see Fig. 8(b)).

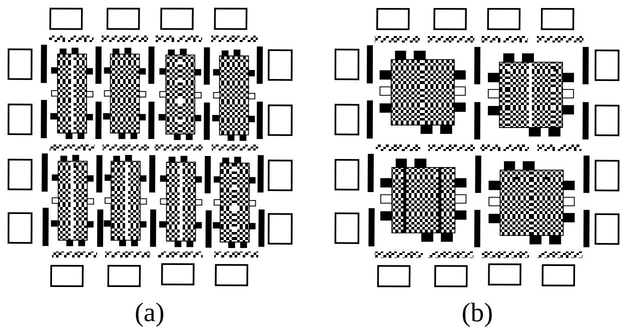


Figure 8. Channel disappear when (a) Two logic block combined to one local logic block. (b) Four logic block combined to one local logic block.

4.2. LRM for illegal tracks and coordinate adjustment

If the path of a net in the hardware bytefile passes any channel that is illegal after logic block rebinding, it should be modified to use legal channels. Fig. 9 illustrates two such situations.

When the abstract logic blocks are rebound to make local logic blocks, and illegal tracks have been removed, the coordinate system of the abstract FPGA is converted to that of

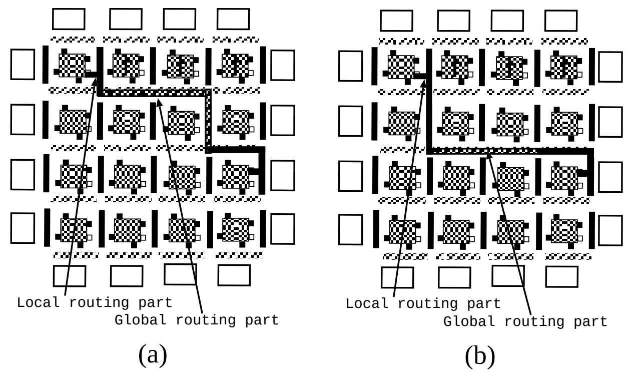


Figure 9. Example of routing modification for illegal track . (a) The original global connection in the abstract routing architecture. (b) Modified global connection in the local routing architecture.

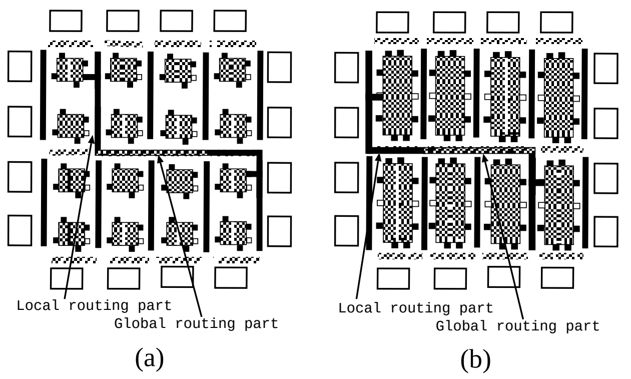


Figure 11. Example of pin local routing modification. (a) The original connection in the abstract logic block. (b) Mapped connection in the local logic block.

the local FPGA coordinate system. The routing information is now represented on the local coordinate system.

4.3. Pin reassignment and LRM for pin

In the pin reassignment phase, the pin location in each new logic block should be reassigned. This pin reassignment is based on the pin location information contained in several abstract logic blocks which composed this new block. Fig. 10 is used to illustrate this.

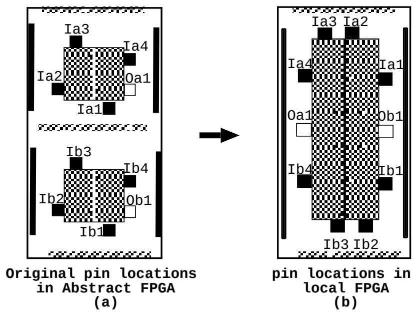


Figure 10. An example of pin reassignment

The results from the pin reassignment phase provide the basis to the local routing modification (LRM) for the pins. Because although pin location may change from one side to another, this change can be updated through local routing modification. Compared to a complete rerouting algorithm, LRM can greatly reduce the computation complexity. Fig. 11 is used to illustrate this.

4.4. Pad reassignment and LRM for pad

Similar to the objective and approach in pin reassignment, I/O pad will be reassigned to accommodate the difference between abstract I/O pad and local I/O pad. In the LRM for pad phase, the converter does the job quite similar to what it does in the LRM for pin phase. The only difference is that it is the pad that has been adjusted.

4.5. Detailed routing

In the detailed routing phase, the path of each net will be accurately assigned to each track. Because the global routing information contained in the hardware bytefile has given an easy start for the detailed routing, detailed routing is simplified to a track scheduling task.

5. Experimental Results

Bytefiles for ten large MCNC benchmark circuits have been generated. Each of the MCNC benchmark circuits was synthesized with the SIS [6] tool and then technology mapped to our abstract FPGA architecture using Flowmap [7] and VPACK [8]. The outputs of VPACK are then passed to VPR [8], which generates the placement and global routing information on our abstract FPGA architecture. This placement and global routing information is finally converted to our bytefile using the format defined in Fig. 7.

To make a comparison with commercial FPGA bitstream files, Xilinx 4000 series have been chosen to implement the same benchmark circuits. Results for both bytefiles and bitstream files are summarized in table 1. It reveals that, bytefiles are normally larger than bitstreams although the bytefile contains less and abstract information. That is because

Circuit	#Gates	#FF	Device	Bitstream (bits)	Bits Per Gate	Bytecode (bytes)	Bytes Per Gate
alu4	3915	0	XC4005E	95,008	19	314,178	80
apex4	3206	0	XC4005E	95,008	19	289,186	90
bigkey	4675	224	XC4005E	95,008	19	324,638	68
clma	22136	33	XC4025E	422,176	17	1916,162	87
dsip	4330	224	XC4005E	95,008	19	273,462	63
elliptic	9475	1266	XC4010E	178,144	18	712,704	75
ex5p	2768	0	XC4003E	53,984	18	248,990	90
s298	5049	8	XC4006E	119,840	20	360,090	71
s38417	16911	1463	XC4020E	329,312	16	112,296	66
spla	10182	0	XC4013E	247,968	19	880,228	86
Average					18		78

Table 1. Bytecode vs bitstream comparison

in a bitstream, the coordinate information for one routing resource (X-channel or Y-Channel) is implicitly indicated by the bitstream sequence, but the abstract bytefile has to explicitly specify it using four or more bytes. Since our project is now not going to the optimization phase, we only use simple and direct way to write abstract routing information into bytefile (see Fig. 7). We expect the bytefile size can be greatly decreased by doing encoding.

We are now working to build a reference implementation for the client converter to measure T_a and T_b (Let T_a be the time for a client to get global routing results from the bytefile, T_b be the time for a client to get global routing results from the scratch). The client converter uses the algorithms introduced in section 4. It is predicted that T_a will be much faster than T_b, because placement and global routing tasks are much more computation intensive than local routing modification [9].

6. Conclusions

We have developed an abstract FPGA model and a split-up implementation flow to realize a hardware virtual machine. By introducing the local routing modification (LRM) methods, most of the time consuming mapping tasks (placement and global routing) have been assigned to the service providers, and only reasonable portions of the mapping task are done by the client terminals.

Acknowledgments

The authors would like to acknowledge the valuable discussions with Professors Francky Catthoor (K.U.Leuven / IMEC), Rudy Lauwereins (K.U.Leuven) and many people in the DESICS division of IMEC. The research was supported by the flemish IWT under the ITA-II Multiservice Network Technologies project.

References

[1] Y. Ha, P. Schaumont, L. Rijnders, S.Vernalde, F. Potargent, M. Engels, and H. De Man, A scalable Architecture to Support Networked Reconfiguration, *Proceedings of IEEE ProRISC*, The Netherlands, November 1999, pp. 677-683.

[2] J. Meyer, T. Downing, *Java Virtual Machine*, O'Reilly & Associates, Inc, 1997.

[3] R. Sevcik, Internet Reconfigurable Logic, *white papers*, Xilinx Inc, 1999.

[4] Xilinx Inc., *The Programmable Logic Data Book*, 1999.

[5] Altera Corporation, *Data Book*, January 1998.

[6] E. M. Sentovich et. al, SIS: A System for Sequential Circuit Analysis, *Technical Report NO. UCB/ERLM92/41*, University of California, Berkeley, 1992.

[7] J. Cong and Y. Ding, Flowmap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Design, *IEEE Trans. on CAD*, 1994, pp.469-485.

[8] V. Betz and J. Rose, VPR: A New Packing, Placement and Routing Tool for FPGA Research, *Int. Workshop on Field-Programmable Logic and Applications*, 1997, pp. 213-222.

[9] C. Ebeling, L. McMurchie, S. A. Hauck and S. Burns, Placement and Routing Tools for the Triptych FPGA, *IEEE Trans. on VLSI systems*, Dec 1995, pp.473 - 482.