Practical Experiences with the SPARXIL Co-Processor

Andreas Koch[1] and Ulrich Golze
Technical University of Braunschweig, Department for VLSI Design
Gaußstr. 11, D-38106 Braunschweig, Germany
{koch,golze}@eis.cs.tu-bs.de

Abstract

This paper examines the use of compact FPGA-based configurable processors as an alternative to ever-higher powered general purpose CPUs. It describes sample applications in which even a very simple configurable processor outperforms all but very fast general purpose CPUs. Performance data is given for DES encryption, labeling objects in black-and-white images, and LZW decompression. In particular, the design of the labeling co-processor is presented.

1. Introduction

In the quest for ever more computation capacity, popular solutions include faster general-purpose CPUs, and faster special-purpose processing units (DSPs, specialized multimedia processors, etc.). Both of these approaches share a fixed internal architecture, and are manipulated solely by software.

Another, still somewhat exotic alternative is to employ a reconfigurable processor having a completely flexible internal architecture that can be finely adapted to each problem. Such a processor can be realized using Field Programmable Gate Arrays (FPGAs), a class of chips that allow the composition of arbitrary digital circuits from basic logic cells at runtime. An algorithm for a configurable processor thus consists of a mix of hardware (the internal architecture) and software (driver programs or programs executing on the specialized architecture).

2. SPARXIL architecture

Many configurable processors employ dozens of FPGAs to hold large circuits, and rely on massively parallel processing. While these systems achieve very impressive computational power (even when compared to supercomputers) for algorithms like DNA matching and fingerprint analysis [1], they are often not economical for widespread deployment.

Our SPARXIL architecture for a configurable co-processor was developed with slightly different aims: It should consist only of few FPGAs, and dispense with the expensive programmable routing network that is often found in the larger systems. However, it should still be able to efficiently and flexibly support the acceleration of algorithms over conventional processors. The following is a brief overview over the architecture, [2] offers a more detailed perspective on the individual design decisions faced.

2.1. Data and address operations

SPARXIL consists only of three Xilinx XC4010 FPGAs, and two 256k x 32-bit memory banks (Figure 1). This FPGA type can hold up to 7k-20k gate equivalents, which are partitioned into 400 configurable logic blocks (CLB). Since the XC4010 (which was among the largest FPGAs available in 1993) has I/O capacity for only one regular 32-bit data path, we assigned the invidual chips to specific tasks: Each of the memory banks has a dedicated FPGA as address generator (A-FPGA). The data busses of both memories are connected to the third FPGA, which operates as data processing unit (D-FPGA). Each A-FPGA can access the data bus of its memory, and also the D-FPGA, but only the centrally placed D-FPGA has simultaneous access to the data in both memory banks.

This allows us to implement the required wide data path in the D-FPGA, and shift all addressing logic to the two A-FPGAs containing data paths for address manipulation. Since we now have dedicated chips for address operations, we can easily implement complex address generators offering, for example, fast indirect addressing and address arithmetic capable of increment/decrement and scaled indexed addressing as well as the generation of non-linear address sequences. The latter can be especially helpful for the efficient implementation of certain matrix operations [3].

[1] Supported by a HSP-III post-doctoral fellowship of the DAAD.

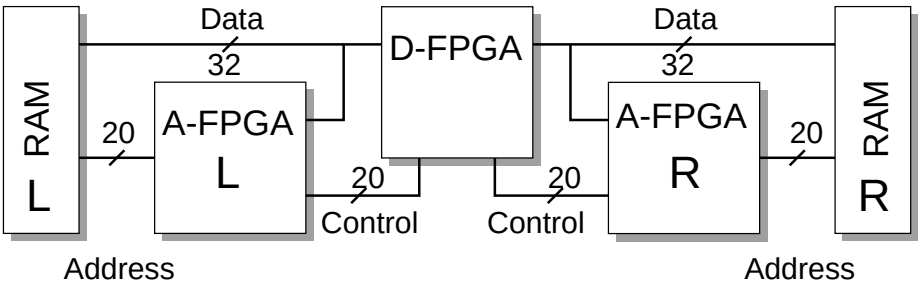


Figure 1. SPARXIL architecture

2.2. Inter-FPGA communication

A dedicated set of 20 control lines each allow the communication between the A-FPGAs and the D-FPGA. These control lines are used to implement user defined communication protocols between D- and A-FPGA suited to the currently running circuit. As long as the communication between D- and A-FPGAs uses only the dedicated control lines, and no indirect addressing takes place, the data bus is available to transfer information between the RAM and the main data path in the D-FPGA.

If the number of control lines is insufficient, the data busses may be employed for A- to D-FPGA signaling. In general, this prevents memory operations during such a control information transfer, but SPARXIL allows a compromise: If the full 32-bits of the memories are not needed for a specific application, the unused data bits may be used as control bits without hindering 16- or 8-bit memory operations.

2.3. Host interface

SPARXIL attaches to a host workstation. User software transfers data to and from the co-processor by advising the host memory management unit (MMU) to map the card onboard RAM into the user process' virtual memory space, allowing the user to access card RAM without regard to its physical location. Since all accesses are still supervised by the MMU and operating system, the memory protection remains inviolate. Thus, algorithms executing on the co-processor cannot bypass system security measures.

2.4. Automatic reconfiguration

While SPARXIL can cache four entire co-processor configurations on-board, and initiate a reconfiguration without host intervention, this feature has not been used in any application to date (all fit in a single configuration each). Furthermore, the need to preserve the FPGA-internal states in the memory banks, and the slow reconfiguration speed of roughly 70ms, make this operation less useful practically than initially conceived. The later constraint could be lifted by the use of more recent FPGAs (such as the Xilinx XC6200 series [4]), which allow reconfiguration up to 1000 times as fast [5].

3. Sample applications

In order to evaluate the efficiency of this architecture, we are in the process of implementing various sample designs, some of which will be discussed in the next sections. Since SPARXIL is based on 1993's technology (Xilinx XC4010PG191-5 FPGAs), performance data will also be given for a hypothetical implementation using the faster XC4010EPG191-1 speedgrade currently available. Furthermore, note that with todays' technology, a single XC4036XL FPGA already has more gate capacity than the entire co-processor, and a single state-of-the-art XC4085XL chip would more than double SPARXIL's capacity.

4. Simple DES encryption

Our first trial circuit is a naive implementation of UltraFastCrypt (UFC) [6], a software version of DES. The UFC processor is a direct mapping of the algorithm to hardware, no pipelining or other architectural optimizations have been performed.

Profiling discovered that the algorithm spends 84.9% of its execution time in a single function that is suitable for implementation on SPARXIL hardware. The function operates on five lookup tables, needs complex addressing and contains tight inner loops handling 32 bit wide data.

It fits the SPARXIL architecture quite well: Due to the reliance of UFC on fast table lookups (precomputed values for S-box usage), the tables can be distributed over both memory banks to allow two parallel memory accesses per clock cycle. Each of the A-FPGAs contains an address generator that indexes the lookup tables. The two data words obtained in this manner are then combined in the center D-FPGA by the XOR-operation typical for DES. Due to the

Platform	@MHz	μs	%SPARXIL-1 speed
SPARXIL-5	16.5	188	49%
SPARXIL-1	34.6	90	100%
SPARC	25	1110	8%
SuperSPARC	85	282	32%
UltraSPARC-II	296	68	132%

Figure 2. DES performance

very simple control flow (only nested loops), each FPGA contains a local controller without any inter-FPGA communication. The circuit was entered in schematic entry. When mapped, both address generators (symmetrical) use 137 CLBs each, the data path uses 198 CLBs.

The performance of the circuit is shown in Figure 2. Note that only the very recent UltraSPARC-II CPU beats even this trivial SPARXIL design.

5. Labeling objects in B/W images

Our next project was more ambitious: We designed a coprocessor for the recognition and labeling of objects (contiguous regions of adjacent pixels) in black-and-white images, a task commonly occurring in computer vision applications (e.g., in robotics). The algorithm implemented is based on [7]. A more detailed description of the circuit including Verilog source code can be found in [8].

5.1. Algorithm

Figure 3. Sample input image

The algorithm expects as input a monochrome image composed of 16-bit pixels. The value '1' marks a set foreground pixel, '0' indicates a blank background pixel (Figure 3). The entire object labeling proceeds in three phases:

1. **Pixel labeling** (or just labeling). Here, we aim to label *adjacent* pixels by assigning them the same object ID. To this end, we scan an operator window (Figure 4)

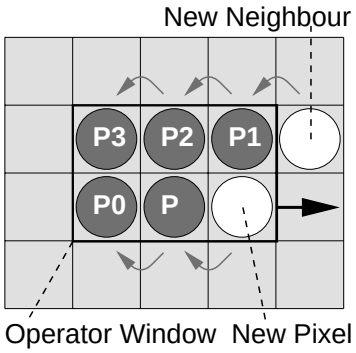


Figure 4. Operator window

from left to right, and top to bottom, across the image. Whenever we find a new solitary pixel at the center P of the operator window, we assign it a new object ID i, and make an entry $m[i] = i$ in the mapping table. If we find multiple adjacent pixels in the window, we assign the current pixel P the minimum object ID j of all already labeled pixels in the window. We remember the previous (larger) object IDs k of the already labeled pixels together with the new (smaller) value j of P in a mapping table as $m[k] = j$ to discover transitive adjacency relations later. Applied to the sample input image in Figure 3, this yields the pixel labeling shown in Figure 5, and the mapping table in Figure 6.

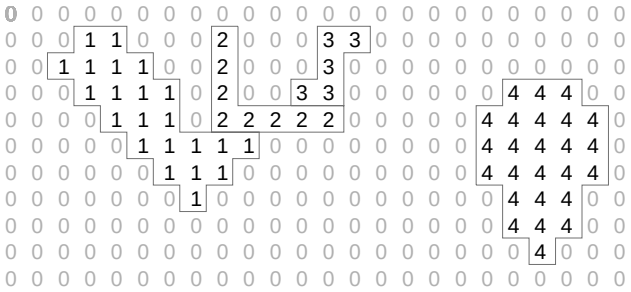


Figure 5. Pixel labeled input image

2. **Transitive flattening**. In this phase, we flatten each entry in the mapping table into its smallest transitively

Object ID	Adjacent to
1	1
2	1
3	2
4	4

Figure 6. Mapping table after pixel labeling

Object ID	Adjacent to
1	1
2	1
3	1
4	4

Figure 7. Flattened mapping table

reachable object ID (Figure 6). For the example, this creates the flattened mapping table of Figure 7.

3. **Object merging**. Finally, we use the flattened mapping table to merge object segments that were transitively adjacent into single objects, assigning them the smallest object ID found during flattening (Figure 8).

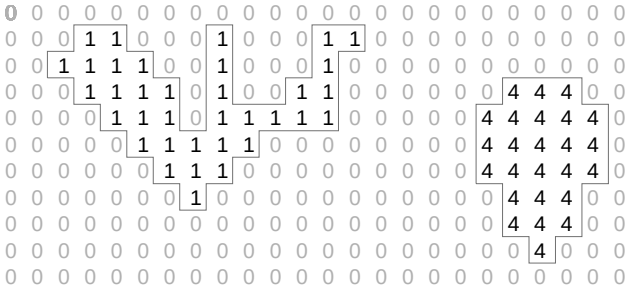


Figure 8. Merged object segments using flattened mapping table

As output, the image has been labeled in-place: Each of the 16-bit words now contains the object number this pixel belongs to. Furthermore, the number of labeled objects can be determined by reading the mapping table m and counting the number of entries with $m[i] = i$ (in the example, 2).

5.2. Hardware implementation

In order to exploit the parallelism inherent in the algorithm, and to allow efficient pipelining, the following partitioning was chosen: The image is held in the left memory bank, while the mapping table occupies the right one. This assignment allows the pipelined schedule shown in Figure 9 for pixel labeling. The flattening and merging phases do not profit from the separate memories, their schedules are simply sequential.

5.3. Design flow

A C implementation of the algorithm was used for reference during the entire design cycle. Next, an RTL Verilog model was composed. It describes the fundamental units of

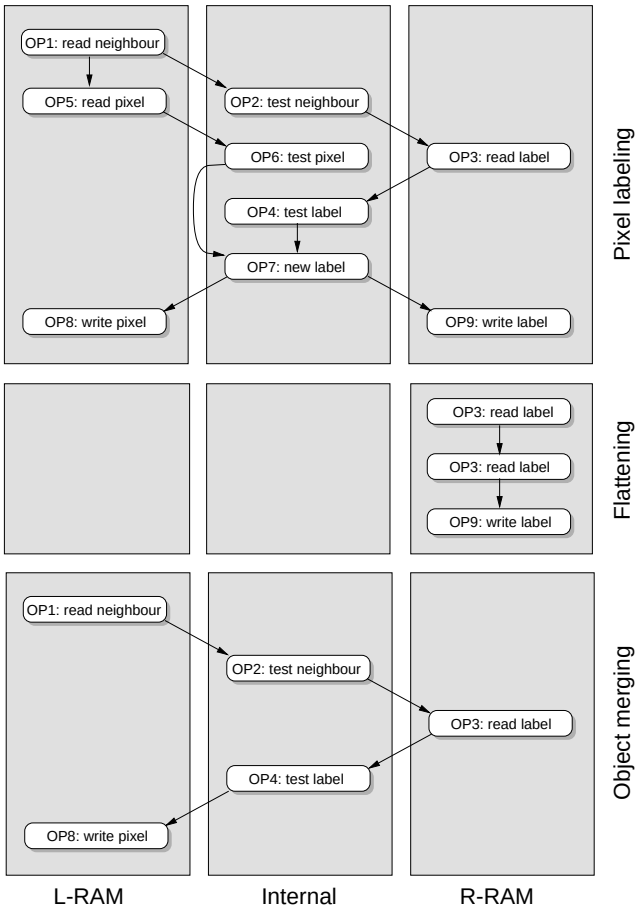
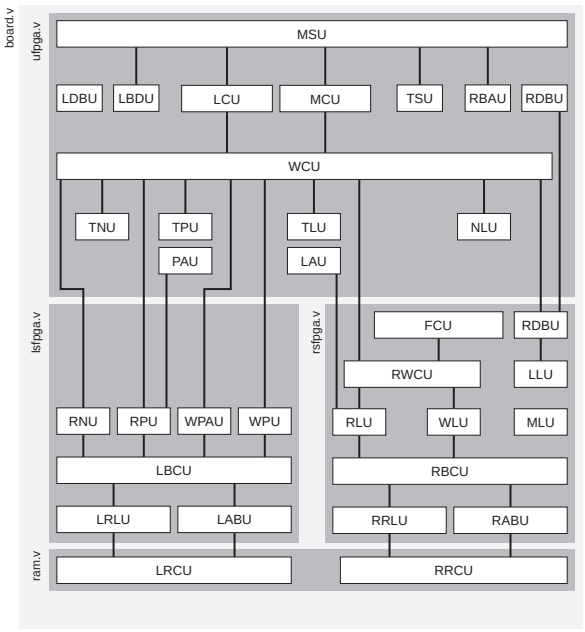


Figure 9. Schedules

the design , and was used to test the pipelining and memory access schemes.

This model was then refined by considering the actual SPARXIL architecture: The functionality is now distributed over the D- and A-FPGAs: The logic dealing with the image has been put in the left A-FPGA, while the logic dealing with the mapping table is implemented in the right A-FPGA. This assignment puts logic close to the data it processes (Figure 10). In this design, the number of inter-FPGA signals exceeded the number of control lines. But, since all data processed is only 16 bits wide, the unused part of the 32-bit inter-FPGA data busses were available for increased control bandwidth.

Between each design phase, the models were verified against each other, and the C reference. The second model (written in a style amenable to synthesis) was then submitted to the Synopsys FPGA Compiler for implementation. The resulting netlist was placed and routed by the Xilinx XACT tool suite. Post-layout simulations against the second Verilog model demonstrated the correct operation of



FCU	flattening control unit	LABU	left address bus unit
LAU	label address unit	LBCU	left bus control unit
LBDU	left bus data unit	LCU	labeling control unit
LDBU	left data bus unit	LLU	last label unit
LRCU	left RAM control unit	LRLU	left RAM logic unit
MCU	merging control unit	MLU	max label unit
MSU	mode selection unit	NLU	new label unit
PAU	pixel address unit	RABU	right address bus unit
RBAU	right bus address unit	RBCU	right bus control unit
RDBU	right data bus unit	RLU	read label unit
RNU	read neighbour unit	RPU	read pixel unit
RRCU	right RAM control unit	RRLU	right RAM logic unit
RWCU	right work control unit	TLU	test label unit
TNU	test neighbour unit	TPU	test pixel unit
TSU	taken signal unit	WCU	work control unit
WLU	write label unit	WPAU	write pixel address unit
WPU	write pixel unit		

Figure 10. Hardware architecture

the entire circuit. The complete design uses 185 CLBs in the left A-FPGA, 344 CLBs in the D-FPGA, and 124 CLBs in the right A-FPGA.

The performance data for the labeling problem is shown in Figure 11. The more intricate hardware design bears fruit, in that even a 1993's SPARXIL-5 easily beats even a state-of-the-art CPU.

6. Further examples

We are currently in the process of implementing co-processor designs for LZW decompression (promising preliminary performance data in Figure 12), and an accelerator for the TIERRA [9] artificial life simulator.

7. Conclusion

SPARXIL demonstrates that even simple FPGA-based reconfigurable processors can offer considerable speedups over conventional CPUs. When high-performance computation without the inflexibility of custom silicon is required, it might be worthwhile to consider an economical FPGA-based co-processor implementing problem-specific architectures, instead of a more expensive standard CPU with hardwired internals.

References

[1] Buell, D.A., Arnold, J.M., Kleinfelder, W.J., "Splash 2 – FPGAs in a Custom Computing Machine", IEEE Computer Society Press, 1996

[2] Koch, A., Golze, U., "A Universal Co-Processor for Workstations" in *More FPGAs*, ed. by Moore, W., Luk,W., Oxford 1994, pp. 317-328

[3] Ast, A., Hartenstein, et al., "Novel High Performance Machine Paradigms and Fast-Turnaround ASIC Design Methods", Proc. 2nd FPL, Vienna, 1992

[4] Xilinx, Inc., "XC6200 Field Programmable Gate Arrays", *http://www.xilinx.com/partinfo/6200.pdf*, 1997

[5] Xilinx, Inc., , "Introducing the XC6200 FPGA Architecture", *http://www.xilinx.com/xcell/xl18/xl18-22.pdf*, 1997

[6] Glad, M, "GNU crypt 2.0.4", *ftp://ftp.ifi.uio.no/pub/-gnu/glibc-crypt-2.0.4.tar.gz*, 1997

[7] Wahl, F.M., "Digitale Bildverarbeitung", Springer, 1989

[8] Meyer, K., "Entwurf eines FPGA-basierten Co-Prozessors zur Objekt-Etikettierung in der Bilderkennung", Diploma Thesis, Tech. Univ. Braunschweig, Germany, 1997

[9] Ray, T.S., "An Approach to the Synthesis of Life.", in M. A. Boden (ed.), The Philosophy of Artificial Life., Oxford University Press, 1996

Platform	@MHz	ms	%SPARXIL-1 speed
SPARXIL-5	10.9	74	43%
SPARXIL-1	25.2	32	100%
SPARC	25.0	3044	1%
SuperSPARC	85.0	480	8%
UltraSPARC-II	296.0	151	21%

Figure 11. Object labeling performance

Platform	@MHz	ms	%SPARXIL-1 speed
SPARXIL-5	9.5	492	67%
SPARXIL-1	14.2	329	100%
SPARC	25.0	5151	6%
SuperSPARC	85.0	1012	33%
UltraSPARC-II	296.0	266	124%

Figure 12. LZW decompression performance