A Universal Co-Processor for Workstations

Andreas Koch and Ulrich Golze

Abteilung Entwurf Integrierter Schaltungen (E.I.S.)
TU Braunschweig
D-38023 Braunschweig, Germany

Abstract

This paper considers the design and implementation of an FPGA-based configurable co-processor for general-purpose computers. Topics include the selection of suitable FPGAs as well as their interconnection. Various architectures are considered, one of which is examined in detail. It is based on the concepts of providing complex address generators separate from the data path and the availability of multiple concurrently accessible memory banks. The hardware and software interfaces to the host and operating system are outlined before a practical implementation and problems suitable for implementation on the proposed co-processor are discussed.

INTRODUCTION

While today's general purpose CPUs are becoming more and more powerful, various algorithms are nevertheless implemented more efficiently using dedicated hardware. This has lead to the development of specialized co-processors, circuits tailored to one problem domain or even a single algorithm. Co-processors currently in use range from sub-systems capable of handling numerical or graphical operations independently to specialized data transformations such as data encryption or compression. Apart from algorithms that are merely accelerated by utilizing such hardware, some real-time algorithms cannot be implemented efficiently without making use of dedicated processing units, since the employment of a sufficiently powerful general-purpose CPU, even if one is available, would not be cost-effective. The shifting of computation from the CPU to dedicated hardware evolved into the concept of hardware-software co-design by developing an algorithm and specialized hardware to execute its most time-consuming parts in parallel, always considering the dependencies between the hardware and software.

While the approach outlined in the previous paragraph can lead to remarkably powerful high-end systems employing co-processors to handle a broad variety of tasks, it is also impractical for use in systems positioned lower in the performance scale due to cost and other constraints (space, power, etc.). Furthermore, after an algorithm is committed to silicon, it is

no longer amenable to change, be it to correct an error or to alter the algorithm itself. Especially in the development stage, when the variation of algorithmic details and parameters can lead to a better understanding of the problem itself, this immutably of an ASIC usually precludes its use as a prototyping vehicle during the development process.

An FPGA-based co-processor can be used to address all of these points: since it can be dynamically reprogrammed, different co-processor functions can be time-multiplexed onto the same silicon, thus enabling even lower-end systems to profit from the presence of specialized hardware. While the performance of such a system will certainly not be on par with one using multiple specialized chips in parallel, a fact also compounded by the lower performance of configurable logic compared to dedicated silicon, it will usually still have numerous advantages over one relying solely on the CPU for computation and I/O purposes: though the circuits an FPGA is capable of implementing are far less complex than those realizable using custom silicon, FPGAs can often efficiently accommodate the inner loop of algorithms consisting of logical operations and integer arithmetic. Examples for this kind of algorithm include cryptological, data compression and verification as well as computer vision applications. Furthermore, the FPGAs can be dynamically reconfigured to address different requirements of the executing algorithm, somewhat easing the complexity restrictions on the circuits. Even if the performance of an FPGA based co-processor should prove insufficient for the real-world application, it can still serve as a useful prototyping and evaluation vehicle allowing to estimate the potential performance of an ASIC solution while still allowing modification of the algorithm under development.

The design, implementation and evaluation of such a general purpose FPGA co-processor on an expansion card for SBus workstations (such as the SUN SPARCstations) are some objectives of an ongoing research project at the department E.I.S. Our proposed architecture will be suited as a customizable co-processor for the acceleration of specific applications as well as an evaluation platform for solutions obtained using hardware-software co-design. In a secondary function, it will be able to act as a flexible I/O subsystem. The architectures we considered are based on a small number of FPGAs. While it is certainly possible to obtain more impressive results by employing dozens of large FPGAs in parallel, this approach is not economically feasible for general use. Our aim is to demonstrate that even a small co-processor can be used with good results when added to a standard workstation.

This paper will consider the architecture of the proposed expansion card, its hardware and software environment and the practical implementation. We will also present some alternative solutions for the problems and examine our reasons for making specific design decisions. A sample application will be discussed and preliminary conclusions drawn.

FPGAS: PRACTICAL CONSIDERATIONS

Selection of suitable FPGAs

The requirement of dynamic in-system reconfigurability precludes the use of EPROM or AntiFuse-based FPGA technologies, rather an SRAM-based technology should be employed. The fulfilment of this requirement, combined with the local availability of CAD software, lead us to base our design on the Xilinx Logic Cell Array (LCA) series of FPGAs. As this choice is not based solely on the merits of the Xilinx chips but on external factors as well, it is not static. For example, it would be highly interesting to compare and contrast the Xilinx

LCAs with the AT&T Orca series of chips, whose architecture seems to be well suited to applications with a wide data path.

Initially, we intended to base our board on the Xilinx XC3195 type of LCAs for reasons of speed and software availability. However, an evaluation of the XC4k series conducted after XC4k-capable CAD tools became available at our institute gave us reason to reconsider our choice of LCA: while the XC31xx-3 series may be faster technologically than the XC40xx-5 series of chips (2.7 ns vs. 4.5 ns combinatorial CLB delay), the improved architecture of the XC4k series more than compensates for these differences for the class of applications considered (wide data paths and registers with logical and arithmetic operators). This, combined with the presence of features such as on-chip RAM and wide decoding capability, convinced us to use XC4010-5 LCAs in the main data path of our design. A pre-configured XC3142-3 type LCA would be employed as the master controller and bus interface.

Routing and integration issues

As always when attempting to integrate dynamically reconfigurable FPGAs into an otherwise static system, one of the main points to be considered is the fact, that different configurations loaded into the FPGAs usually lead to different pin-outs if the place and route process is allowed to run unconstrained. This is acceptable if only a single circuit is to be loaded into the FPGA at start-up, as in the case of the controller FPGA mentioned above. The user-configurable FPGAs, however, with their changing configurations, present a problem: changing pin-outs make them difficult to use with a fixed PCB layout. Two alternatives present themselves: establishing a dynamic mapping of FPGA pin-outs to PCB traces or locking FPGA pins.

Dynamically re-mapping changing FPGA pin-outs to fixed PCB wires can be done by interposing a configurable routing system for the signals between the user FPGAs and the fixed PCB layout. Such a system could, for example, consist of a cross-bar switch, another FPGA used solely for routing purposes or one of the recently introduced field-programmable interconnect components (FPICs) as offered by Aptix, for example. Since the number of lines to remap tends to be quite large in the applications we envision (32 bits of data, 20-32 bits of address bus plus assorted control lines), the limits of standard cross-bar switches as used in telecommunications applications are soon reached. Apart from that, their propagation delays of 25-40 ns place a considerable burden on the system performance as a whole. The use of another FPGA solely for routing purposes does not seem viable either: while the larger FPGAs have the I/O capability to establish a 64 to 64 mapping, for example, their internal routing architecture is also not up to the task of providing the desired performance. A cross-chip routing on an XC3195-3 chip, for example, has a worst case propagation delay of 37.1 ns. The last alternative, the use of FPICs, is the most attractive: low propagation delay times (in the range of 4-7 ns, depending on routing distance) combined with the ability to switch even very wide busses (Aptix offers 938 freely interconnectable pins) makes them an ideal solution to glue FPGAs onto a PCB. Unfortunately, practical considerations precluded (at least in our case) the use of FPICs. Due to them being state-of-the-art technology, the chips as well as the associated CAD tools are prohibitively expensive. However, we are closely following this development and will certainly consider using FPICs in future implementations when their prices come down.

Since none of the dynamic routing options investigated seemed viable, we had to consider the other alternative, constraining the place and route process to lock FPGA signals

to specified pins of the package and thus to the fixed PCB traces. This practice of locking I/O Blocks (IOBs) is frowned upon by Xilinx, since it hampers the place and route algorithm and may cause the automatic design implementation (ADI) process to fail, even when an unconstrained design could be realized successfully. However, when the LCA (both XC3k and XC4k) architecture is examined closely, implicit constraints become obvious: for example, since the only tri-state buffers (TBUFs) available on the chips are located adjacent to the horizontal long line (HLLs) interconnections, all bi-directional chip-wide busses must use these HLLs, imposing constraints on block placement as well as routing. Thus, the general layout of a chip-wide data bus is already pre-defined by the LCA architecture itself. By choosing a suitable package, it can be guaranteed that all HLLs are accessible from the outside using bonded IOBs. Furthermore, as long as the design to be realized does not closely approach the maximum capacity of the LCA in question, the effects on implementability and performance seem to be negligible. Thus, this scheme was chosen to connect the FPGAs to the other components on the PCB. However, we concede that this approach, especially the performance penalty incurred, needs further study and is inelegant compared to a dynamic I/O remapping using FPICs.

SBus FEATURES

The SBus was developed by SUN Microsystems as an I/O bus to replace the VME bus in its desktop workstations. While it is predominantly used in SUN SPARC-based workstations and their compatibles, the bus is processor independent. It is a high bandwidth/low latency design supporting multiple bus masters and automatic virtual-physical address translation as well as fast burst data transfers. Further features include flow control and retry mechanisms for slave devices, dynamic bus sizing, central bus arbitration with geographic card addressing and flexible interrupt management. The bus has a CMOS-compatible interface and is based on simple, synchronous protocols.

CO-PROCESSOR ARCHITECTURE

Fundamental design decisions

Early in the design process, it had to be decided whether to include on-board RAM on the expansion card or to rely on the host RAM. Technically, both alternatives are viable since the SBus with its multiple master and direct virtual memory access (DVMA) methods allows any card to execute master protocols, thus accessing system memory as well as other SBus devices. Since the virtual to physical address translation is performed by the bus, it does not add to the complexity of the card.

However, after considering the application domain of the card, the decision to employ a sufficiently large on-board RAM enabling the card to run without bus accesses, was easily made. First, the maximum sustained transfer rate of the SBus is around 80 MB/s when using 16 word 32 bit burst-transfers (for complexity reasons, we disregard 64 bit transfers) and far less for single word transfers (at least four cycles overhead per word). SPARCstation 1+ workstations, as available at our site, are even slower, peaking at around 50 MB/s burst transfer. On the other hand, the optimum bandwidth of an economically feasible on-board

memory with a ≈30 ns cycle time (including glue logic) is ≈120 MB/s based on 32 bit wide access paths and does not incur further delay due to bus protocol.

The comparably slow performance of the SBus with regard to a pure memory bus should not be construed against the SBus in general. It has been developed as a general purpose I/O bus. For CPU(s) to memory interconnect, SUN has implemented the MBus, which is optimized for this kind of application and offers peak transfer rates of up to 320 MB/s. Unfortunately, most SPARCstations are currently not ready to accept co-processor cards on the MBus, thus we have to abuse the SBus as a processor-to-processor and processor-to-memory-bus.

While much of the complexity of SBus mastership is delegated to the central SBus controller, being a bus master still requires a more intricate knowledge of the bus protocol (bus sizing etc.) than performing only slave duties. This fact combined with the figures on bus bandwidth lead us to design the card as an SBus slave device with on-board memory which can be accessed by the host (or other bus masters) using fast burst transfers. Furthermore, since bus accesses are no longer needed while the card is running, the bus can be used by the host for other tasks in parallel which also improves performance of the whole system.

Another fundamental decision made at this stage was caused by the small (compared to ASICs) maximum complexity of the FPGAs considered. Our basic design calls for a 32 bit wide data path combined with 20-32 bit addressing capability. Keeping in mind our planned on-chip topology (regular layout with bi-directional busses based on HLLs) and the desire to avoid multiplexing data- and address lines (to minimize circuit complexity and maximize performance), we needed LCAs with at least 52 HLLs. Unfortunately, these circuits (XC4016 and XC4020) are only slowly becoming available. Thus, we decided to assign the tasks of data and address management to different but closely coupled LCAs.

This allows us to implement the required wide data path in the main FPGA (which will be called user FPGA from now on) and shift all addressing logic to one or more FPGAs containing data paths for address manipulation (termed service FPGAs). Since we now have dedicated LCAs 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. We feel that this is a very natural way of partitioning the target algorithms we envision. The communication between user and service FPGAs will be facilitated by allowing both chips access to the shared data bus and a relatively small (≈10) number of dedicated bi-directional control lines (see Figure 1).

These control lines, which are located on the unoccupied HLLs of the user FPGA and various other IOBs on the north and south sides of the LCA (reachable using uncommitted vertical long lines, VLLs) can be used to implement user defined communication protocols between user and service FPGA suited to the currently running circuits. They might be used to select address registers inside the service FPGA or cause specific operations such as increment or decrement of addresses. They could also be used to signal the loading of a specific address register from the data bus or select a read or write RAM access and its transfer size. In reverse, the service FPGA can signal the overflow/underflow of various address counters back to the user FPGA. As long as the communication between user and service 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 user FPGA.

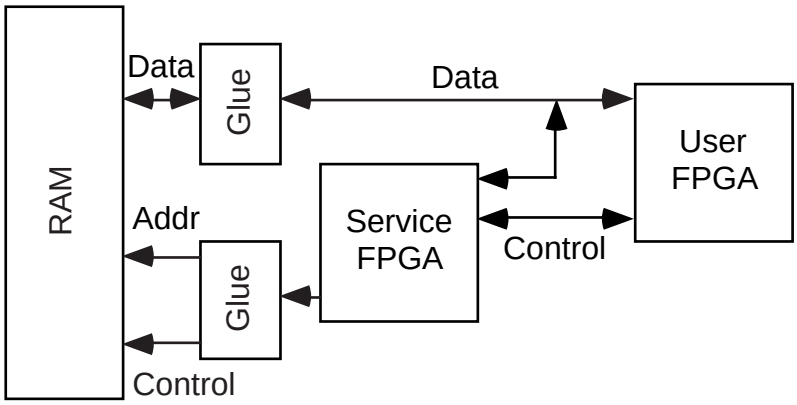

Figure 1 FPGA/RAM interface

Architectures considered

With the basic parameters of the card fixed (on-board RAM, separate address and data FPGAs), we will now briefly highlight some of various architectures considered (Figure 2) before we justify choosing one specific architecture. As mentioned in the introduction, our approach is to use a few, tightly integrated FPGAs capable of housing only the inner loop of the target problems instead of partitioning the whole algorithm into a large array of FPGAs. Figure 2 depicts only the general organization of the architectures. Missing from all diagrams are components such as the SBus interface, glue logic, FPGA control lines and an optional external I/O bus.

Architecture A is the simplest one still fulfilling our basic parameters. While it is very easy to implement, it has only limited gate capacity and is somewhat uninteresting to experiment with. It offers no distinct advantages over a fast CPU.

Architecture B expands upon A by adding a second RAM sub-system. The gains offered by this approach far outweigh the added complexity. Apart from the additional gate capacity afforded by the second service FPGA, this co-processor can parallely access data in two different RAM banks. This makes architecture B far more interesting to experiment with, since it offers a distinct advantage over conventional CPUs, which can only use one RAM access path at a time.

Architecture C adds more gate capacity by duplicating the user FPGA. Both user FPGAs could communicate via connected HLLs, thus still complying with our proposed internal organization. A disadvantage of this approach is, that for an user FPGA to access a non-adjacent RAM bank, data has to be routed through the other user FPGA, thus incurring a massive routing delay (as explained earlier).

Architecture D attempts to solve this problem by adding a data routing network between the two user FPGAs. This network, which could be implemented using Quality Semiconductor QuickSwitches, for example, could dynamically route data from one user FPGA to the other user FPGA or directly to a non-adjacent RAM bank. The disadvantage of this approach is its complexity, which will complicate the PCB layout considerably by adding more layers.

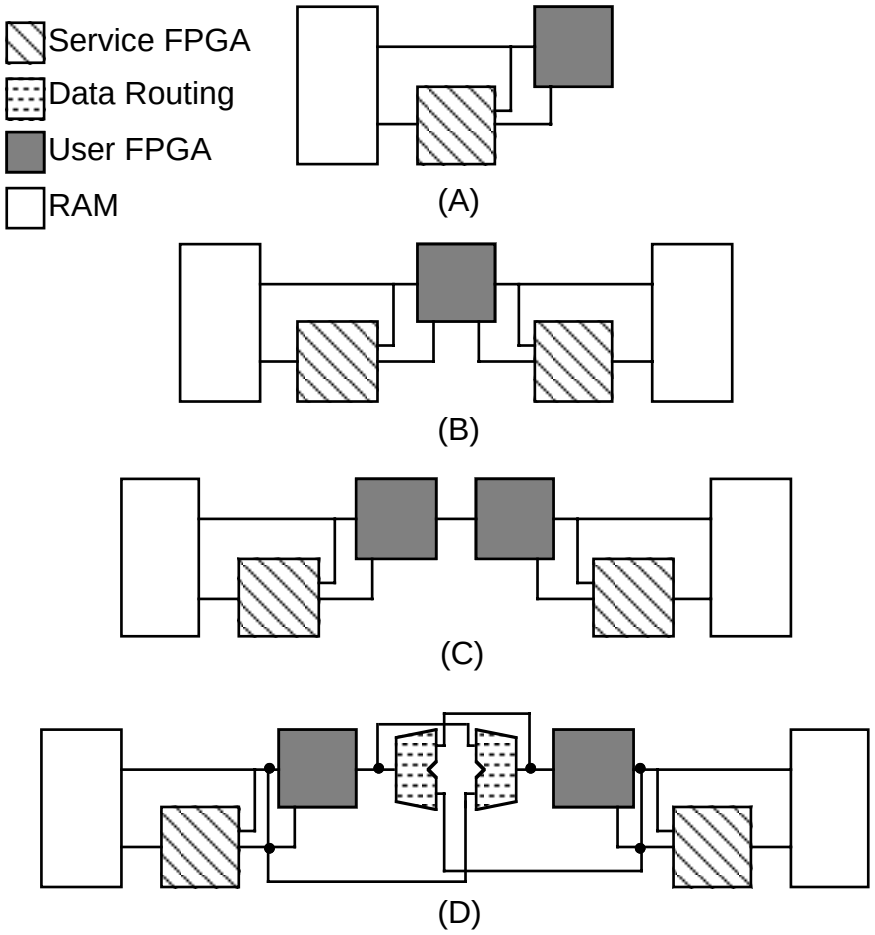

Figure 2 Some co-processor architectures

Architecture implemented

We chose to implement architecture B, since it represents a good compromise between PCB complexity and computing power. The architecture's two RAM access paths are very well suited to the LCA's internal routing organization, since the HLLs of the XC4k series can be split in the middle, thus creating two completely parallel processing units with independent bi-directional busses which can communicate via CLBs at the split point. In order to conserve FPGA capacity for user applications, standard data routing and RAM access control logic has been factored out of the FPGAs and moved into external circuits. For example, a byte and short word steering logic for 8 and 16 bit wide RAM accesses is available as well as RAM access arbitration between the SBus and the FPGAs. Furthermore, the RAM banks can be switched to a mode that allows a write access from the SBus to be applied to both banks simultaneously. This feature can be used to reduce initialization time, for example, when identical lookup-tables have to be set up in both RAM banks for increased parallelism.

In order to run each FPGA configuration at its optimum speed, the board also includes a programmable clock generator capable of generating frequencies from a few hundred kHz up to 80 MHz with fine tuning to almost any target frequency in between. Thus, FPGA applications can be clocked independently of the SBus clock: slower for more complex circuits, faster for simpler designs.

Configuration

While further refining the architecture, we investigated several alternatives on how to configure the user and service FPGAs (the SBus interface/card controller FPGA is configured by serial PROM). The option of directly configuring the FPGAs via the SBus was quickly discarded, since the slow FPGA configuration data rate of max. 1 MB/s would hog the bus for an inordinate amount of time. The next alternative, storing configuration data in one or both of the user RAM banks, becomes less attractive when the waste of space in high speed RAM for such a low speed application is considered. Additionally, this approach complicates the external addressing logic and renders the user memory model inelegant (non-contiguous). The solution we chose is to add a dedicated RAM bank, which can use far slower RAM than the two main banks (85 vs. 20 ns cycle time), solely for the purpose of holding configuration data. This implementation leaves two banks of contiguous memory at the user's disposal while considerably simplifying the addressing and data routing logic. The configuration RAM (termed C-RAM) can be loaded far more efficiently by the SBus than the FPGAs themselves, the implemented transfer rate of more than 8 MB/s is quite adequate for the task. Since the C-RAM has sufficient space for multiple complete configurations for all three FPGAs plus appropriate parameters for the programmable clock (see Figure 3), later reconfigurations using the stored data place no load on the SBus at all.

While the current configuration can naturally be selected by user software, an important feature of the architecture is the ability of the co-processor to reconfigure itself at run-time without intervention of software or the host CPU. The user FPGA signals the controller FPGA to initiate reconfiguration by requesting a load of one the bitstreams stored in the C-RAM. The controller FPGA then enables the appropriate address lines on the C-RAM and sets up a data routing appropriate for reconfiguration before it initiates the configuration process itself. This capability exemplifies our approach of sharing the same hardware not only between different programs and algorithms, but also between phases of a single algorithm, thus maximizing the efficiency of co-processors with limited gate capacity.

Only one of the three run-time configurable FPGAs has a direct parallel connection to the configuration RAM. This chip is loaded in master parallel mode. The other FPGAs are daisy-chained to the master and are configured using slave serial mode. The programmable clock is loaded directly by the controller FPGA, which performs the appropriate C-RAM accesses.

The current design only allows the complete re-configuration of all three FPGAs. Later implementations might be modified to selectively load specific LCAs while leaving the others unmodified. The ability to partially reconfigure FPGAs while the untouched portions of the chips continue processing could also prove beneficial to our concept of maximizing hardware usage. Unfortunately, few FPGA chips currently available provide this capability.

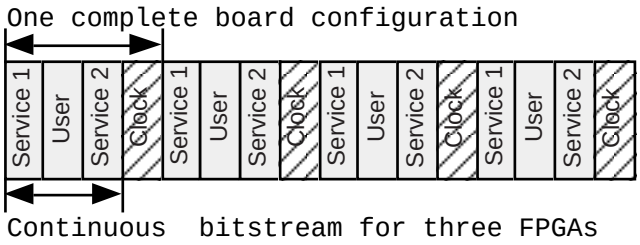

Figure 3 C-RAM organization

SOFTWARE INTERFACE

The Unix operating system

The primary operating system for the host machine (SUN SPARCstation 1/1+/2 or compatible) is a Unix variant named SunOS. Since Unix is a multi-user, multi-tasking OS using virtual memory, it has built-in access protection and addressing requirements which must be observed when adding hardware to the host.

User software communicates with the co-processor through a device driver, which is integrated into the OS kernel. Only this driver may directly access the card, thus conflicts caused by multiple processes trying to use the same card concurrently can be avoided. Furthermore, the driver supervises the configuration process, ensuring that only complete bitstreams are downloaded into the C-RAM. Another service provided by the driver is the ability for the user software and the co-processor to execute in parallel, with the co-processor notifying user programs of the end of calculations through a software interrupt signal.

Co-processor memory map

User software transfers data to and from the card by advising the host memory management unit (MMU) to map the card on-board 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, the memory protection remains inviolate. Thus, algorithms executing on the co-processor cannot bypass system security measures.

The memory layout the card presents when mapped in is summarized in Table 1. The ID-EPROM is needed by the SBus boot sequence to recognize the card's presence and contains, among other data, the name of the driver to use. Next in address space are the different RAM banks, both for configurations and user data. At the upper end reside the card control and status registers.

These registers enable the driver to query the card for its current status (busy, configuring, ...), to select one of the downloaded configurations for execution, to determine the reason for a card-generated interrupt (done, address error etc.), to send data to the programmable clock and to start/stop card operation.

In addition to the driver, the software interface consists of a library of C routines which enable user programs to easily control the co-processor without regard to SBus protocols, virtual memory conventions or layout of the card address space. However, this interface is mainly concerned with administrative functions, user programs have full flexibility in implementing their own communication protocols. Figure 4 presents a trivial example how

Table 1 Co-processor memory map

Address Range	Name	Access	Bit Width
000000 - 007FFF	ID EPROM	R	8
100000 - 17FFFF	C-RAM	R/W	8
200000 - 2FFFFF	User RAM Bank 1	R/W	32
300000 - 3FFFFF	User RAM Bank 2	R/W	32
400000	STATUS register	R	4
400001	CONFIG register	R/W	2
400002	IRQWHY register	R	3
400003	CLKPRG register	W	4
400004	CMD register	W	3
400005	MIRRMEM register	W	1

programs can execute an algorithm on the co-processor card. Note that, in order to improve readability, all error checking and advanced features, such as asynchronous execution and concurrent access to the two user RAM banks, remain unused.

As demonstrated, the data types and organization of parameters passed between the program and the co-processor are completely user-defined and can be matched to the specific application. Since the host MMU has mapped-in the requested amount of card RAM, the communication between program and co-processor takes place transparently.

PRACTICAL IMPLEMENTATION

The prototype card currently being implemented is organized as two connected two layer double-width PCBs, with one level dedicated to the SBus interface, the other one to the FPGAs and RAM. We found this structure preferable to a single multi-layer PCB for economical as well as for debugging reasons.

As mentioned above, the data and address data paths are implemented in three Xilinx XC4010-5 FPGAs, the board itself is controlled and connected to the SBus by a XC3142-3

```
#include "fpga.h"

typedef struct
   {
      int a, b, x;
   } FPGAParams;

main()
{
   FPGA fpga;
   FPGAParams *params;

   fpga = fpgaOpen(fpgaAny);
   fpgaLoadCRAM(fpga, "adder32.cop");
   params = (FPGAParams *)
      fpgaMapBank(fpga, fpgaBank0,
               sizeof(FPGAParams));

   params->a = 35;
   params->b = 7;

   fpgaStart(fpga, fpgaSync);

   printf("%d\n", params->x);

   fpgaClose(fpga);
   exit(0);
}
```

Figure 4 Sample co-processor application: adding two integers

FPGA. We initially intended to use an EPLD to house the SBus interface, but the added complexity of supporting some SBus functions (especially concerning the intricacies of the burst transfer mode) made a small FPGA more attractive.

The on-board user memory consists of two 256Kx32bit SRAM modules with 20 ns cycle time, the C-RAM uses a single 512Kx8bit SRAM module with 85 ns cycle time. This module can house four complete board configurations including the clock data. SRAM is used for its speed as well as its simpler integrability (no DRAM controller is needed).

Data routing functions are provided by wide bus transceivers in ABT technology on the SBus side and integrated pass transistor networks (QuickSwitches) between the FPGAs and RAM banks.

The programmable clock is implemented by an IC originally intended as a graphics clock generator for video interfaces. This circuit can easily be configured to generate virtually any frequency ranging from a few hundred kHz up to 80 MHz and is far superior for our application than any solution based on a custom-designed discrete PLL.

In order to test our implementation of the SBus protocol, the finished card will be examined in a hardware tester combined with a compiler that allows the entry of SBus transactions in a high-level language. These transactions are translated into the tester-specific format and downloaded for execution. The results uploaded by the tester are then annotated back into the high-level source. This setup should simplify debugging considerably over the standard procedure using logic analyzer and oscilloscope.

SUITABLE APPLICATIONS AND IMPLEMENTATION PROCEDURE

Applications currently being investigated for successful implementation on our proposed architecture include general purpose algorithms such as data compression as well as more specialized circuits dealing with computer vision and system security.

For example, one of the applications we examined is a fast encryption algorithm which can be used to “break” the UNIX password system. Profiling tools discovered that this algorithm spends 84.9% of its execution time in a single function. One iteration of this function's main loop takes 72 clock cycles on a SPARC CPU. A hypothetical implementation of this operation on our co-processor should take only 16 cycles per iteration. However, the practically achievable speed-up of a real implementation will ultimately depend on the clock frequency of the FPGA-based solution, which has to compete with the software version executing on a CPU running at 25-40 MHz.

In the current absence of CAD tools tailored to our co-processor architecture, we are using the Xilinx X-BLOX synthesis tool to implement the designs. While this tool is easy to apply and quickly produces usable results, the generated designs do not map perfectly into the co-processor's intended internal architecture. In order to maximize the performance of the board, a later re-design of the circuits developed on the LCA level (using tools such as Xilinx XDE) seems in order. These circuits, having a topology optimized to the co-processor structure and interconnect organization, can then be used to evaluate the performance of the target algorithms.

FUTURE DIRECTIONS

While we are completing the initial prototypes of hardware and software, various areas for future work have been noted. Should our architecture prove successful in benchmarks using optimized circuits, we will endeavor to create CAD tools assisting a designer to map his designs onto the co-processor architecture. Such tools might include services to aid the designer with the floor-planning of hard macros, which are then used to implement larger elements of the data path. It is also possible to conceive a special version of the PPR program which is familiar with the intended topology of the co-processor and maps circuits accordingly.

On the hardware side, it would be most desirable to integrate all SBus interface and data routing functions into a single ASIC, simplifying the card considerably. This might also reduce the power consumption and the form factor from a double-width to a single-width SBus card. As mentioned earlier, alternative implementations of co-processor cards based on results obtained using the initial prototype (maybe employing different FPGAs) might also be considered.

We are also looking forward to the educational possibilities offered by a working configurable co-processor: it could be the perfect teaching vehicle for hardware-software co-design, featuring both a sophisticated software as well as hardware interface. Student labs could consist of a general problem specification which had to be implemented initially as a software-only solution to validate the algorithms proposed, is then profiled to determine bottlenecks and finally partially shifted onto a problem-specific co-processor. This procedure should be interesting for CS as well as EE students, enabling them to experience the best of both worlds when designing interacting programs and circuits while still considering the constraints of an existing environment (Unix and co-processor architecture).

CONCLUSIONS

While it is still too early to draw definite conclusions from the project, our initial simulation results look promising, even when using non-optimized CAD tools. Furthermore, the profiling and evaluation of various algorithms for suitability to being partially implemented in hardware has shown the usefulness even of small configurable co-processors integrated into a general purpose computer.

REFERENCE

Ast, A., Hartenstein, R., Kress, R., Reinig, H., Schmidt, K., "Novel High Performance Machine Paradigms and Fast-Turnaround ASIC Design Methods: a Consequence of, and, a Challenge to, Field-programmable Logic", *2nd Intl. Workshop on Field-Programmable Logic and Applications,* Vienna, 1992