TKDM – A Reconfigurable Co-processor in a PC's Memory Slot

Christian Plessl and Marco Platzner
Swiss Federal Institute of Technology
Computer Engineering and Networks Lab
8092 Zurich, Switzerland
plessl@tik.ee.ethz.ch

Abstract

This paper presents TKDM, a PC-based high-performance reconfigurable computing environment. The TKDM hardware consists of an FPGA module that uses the DIMM (dual inline memory module) bus for high-bandwidth and low-latency communication with the host CPU. The system's firmware is integrated with the Linux host operating system and offers functions for data communication and FPGA reconfiguration.

The intended use of TKDM is that of a dynamically reconfigurable co-processor for data streaming applications. The system's firmware can be customized for specific application domains to facilitate simple and easy-to-use programming interfaces.

keywords: co-processor, DIMM module, data streaming

1 Introduction

FPGAs have proven successful for the implementation of co-processors that accelerate computationally intensive kernels. Examples for such co-processors can be found in the cryptography, signal processing, and pattern matching domains [7, 3, 8, 2]. Often such co-processors achieve high raw speedups compared to implementations on general-purpose CPUs. Despite these high raw speedups, the overall speedups taking into account the data communication between co-processor and host CPU as well as the reconfiguration time often turn out to be rather modest.

The main factor that tends to alleviate a co-processor's impact on the system-level performance is the *limited bandwidth and latency* of the communication between the host CPU and the co-processor. For applications that require excessive communication, e.g., data stream processing, an insufficient bandwidth becomes a severe bottleneck. Further, larger applications consist of a number of kernels. The dynamic configuration of these kernels onto the co-processor asks for low-latency communication.

This work aims at the development of an FPGA-based host/co-processor system for data streaming applications with high bandwidth and low latency communication. For the host processor, we are restricting ourselves to off-the-shelf computing systems such as personal computers (PC) or workstations.

PCs typically come with several standardized interfaces for hardware extensions: peripheral buses (Firewire, USB), I/O buses (PCI, AGP), and memory interfaces (SDRAM, DDR SDRAM, RAMBUS). The previously popular co-processor interfaces have vanished over the last couple of years as floating-point co-processors have diffused into the CPUs.

Most presented FPGA co-processors connect to the PCI bus. A PCI 33 MHz/32bit bus, for example, theoretically provides a maximum bandwidth of 132 MByte/s. However, the PCI bus often becomes a bottleneck as on one hand the usable bandwidth is lower and, on the other hand the bandwidth might be shared among several PCI I/O devices.

It can be observed that for any given computer generation the memory bus is always about one order of magnitude faster than the high-speed I/O buses. For example, PCI 33MHz/32bit with 132 MByte/s is about 8 times slower than a 133MHz SDRAM memory bus that provides 1064 MByte/s. In a more recent system a PCI 66MHz/64bit bus delivers 528 MByte/s compared to a dual channel PC400 DDR SDRAM memory bus with 2×3.2 GByte/s. While standards for faster memories have been rapidly included in desktop PCs and workstations, the enhanced PCI I/O bus standards (PCI 66, PCI X) are making their way so far only to high-end server machines.

Attaching FPGA-based coprocessors to a PC's memory bus instead of the I/O bus is appealing as it allows to leverage a much greater communication bandwidth and lower latency. This is exactly what the TKDM project strives for. We have built an FPGA co-processor module that attaches to the DIMM (dual inline memory module) memory bus and utilizes its full bandwidth. Compared to related work, the TKDM project described in this paper stands out by the following contributions:

- A versatile FPGA (Xilinx Virtex-II) co-processor board that attaches solely to the DIMM memory bus and can be mounted in any DIMM memory slot. The memory bus is used for both data communication and full/partial configuration and readback.

- The integration of the co-processor module into the Linux operating system. Linux libraries together with the coprocessor module's firmware form domain-specific abstractions that ease application mapping.

The remainder of this paper is organized as follows: Section 2 discusses related work. Section 3 introduces the TKDM hardware design and implementation. Section 4 presents the integration of the TKDM module into the Linux operating system and the construction of domain-specific firmware and libraries. The TKDM project is work in progress. Thus, Section 5 discusses the status of the project and gives early performance results for the TKDM platform. Section 6 concludes the paper and outlines future work.

2 Related work

To the best of our knowledge, there exist two related projects that tried to develop a PC memory bus based FPGA co-processor.

Leong at al. developed the *Pilchard* FPGA co-processor platform [4] that uses the PC100 or PC133 DIMM memory bus for data communication. Pilchard is a low-cost system based on a Xilinx Virtex or Virtex-E device that is attached directly to the memory bus. The board comprises an FPGA, a configuration PROM, output headers, and an Xchecker interface for programming the FPGA and the configuration PROM.

In 2001, a company called *Nuron* announced their adaptive computing boards, *AcB* and *AcB+* [1]. These boards also attached an FPGA to an SDRAM memory bus and featured 64MB of on-board memory. Further, reconfiguration support over the memory bus was announced. Although a small number of prototypes seems to exist, the product was never available to the public. In late 2001, Nuron was acquired by Intel.

TKDM has been inspired by Pilchard. However, in contrast to Pilchard the TKDM platform uses two FPGAs and four on-board memory banks. One FPGA connects to the memory bus and forms an abstraction layer that handles the DIMM bus protocol. The second FPGA is used to implement the actual user application. The configuration and readback of the second FPGA is performed via the memory bus. The memory richness allows TKDM to cover a wider range of co-processors, including dynamically reconfigurable designs where full or partial configurations are stored on the board.

3 TKDM Hardware

This section describes the hardware of the TKDM module. First, we list some important design goals. Then, we focus on the module's architecture and main components. Finally, we give some implementation details of the first produced version of TKDM.

3.1 Design goals

The design of TKDM was driven by several requirements. The primary requirement was to build a high-performance FPGA based co-processor platform with high bandwidth and low latency communication to the host CPU. The design has been inspired by Pilchard [4] and extends this concept in various ways:

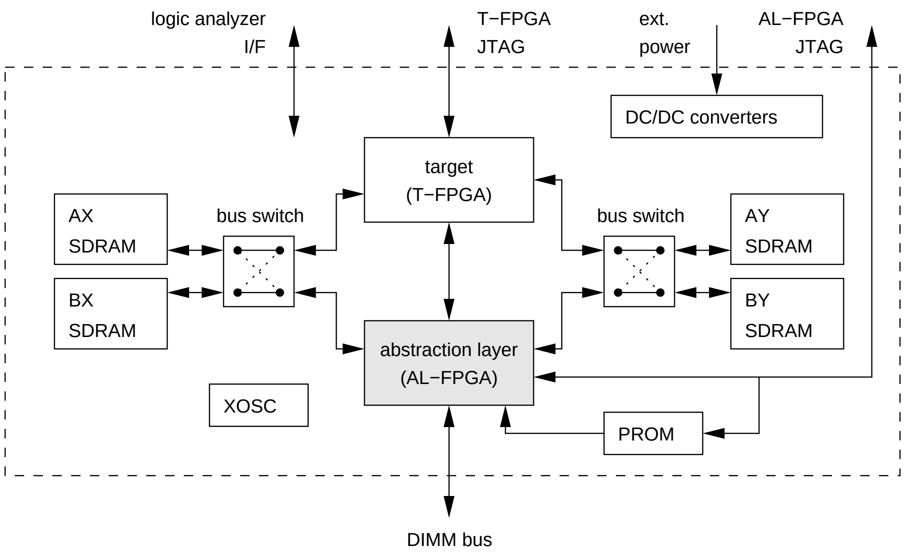
- The board should include several banks of on-board memory that can be used to buffer data as well as configurations. The memory should be accessible by the system's memory controller at full DIMM bus speeds.
- The FPGA implementing the co-processor should be configurable via the DIMM bus and from on-board memory. Both full and partial reconfiguration and readback has to be supported.
- While intended for data streaming applications, the TKDM architecture should provide sufficient flexibility to build customized, domain-specific firmware versions, e.g., for data encryption or audio processing. The environment must be ease-to-use by providing such domain-specific firmware and software libraries. The application developer should not have to know about the details of the DIMM bus protocols and how to access SDRAMs.

3.2 Module architecture

The block diagram of the TKDM is shown in Fig. 1. The module comprises two FPGAs, four banks of SDRAM, two bus switches, and further supporting components such as configuration PROM and DC/DC converters. The two FPGAs are denoted as abstraction layer FPGA (AL-FPGA) and target FPGA (T-FPGA).

Memory banks The four banks of SDRAMs are grouped into two banks attached to the left of AL-FPGA and T-FPGA (AX and BX SDRAMs) and two banks attached to the right (AY and BY SDRAMs) as shown in Fig. 1. Each bank features a 16MByte, 32bit-wide, 133MHz SDRAM.

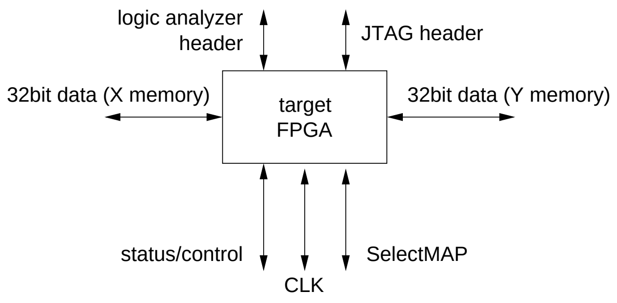
Figure 1. TKDM module architecture



Bus switches The data buses from the FPGAs to the memories are routed via high-speed, low-voltage bus switches. These switches can either connect the data buses straight-through or cross-over. For example, the bus switch on the X side can connect either the AL-FPGA with the BX SDRAM and the T-FPGA with the AX SDRAM (straight-through), or the AL-FPGA with the AX SDRAM and the T-FPGA with the BX SDRAM (cross-over). Both bus switches are controlled independently by the AL-FPGA.

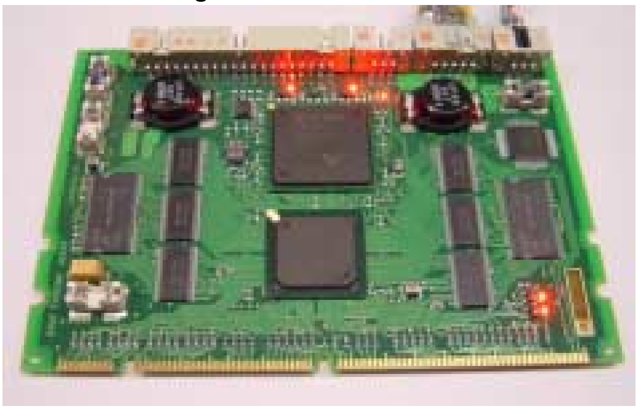
Abstraction layer FPGA The abstraction layer FPGA is the only component in our design that connects to the DIMM bus, i.e., all the time-critical circuitry dealing with the DIMM bus protocol is implemented in the AL-FPGA. The AL-FPGA is responsible for transferring data from/to the DIMM bus to/from the memories or the T-FPGA. To this end, the AL-FPGA contains memory controllers for the SDRAMs. These memory controllers are also used to control data transfers between the memories and the T-FPGA. A co-processor in the T-FPGA thus relies on the AL-FPGA for accessing the memories. While this design decision increases the complexity of the AL-FPGA, it also shows two advantages. First, all memory control and address lines connect only to the AL-FPGA and are not switched which simplifies the board design. Second, the T-FPGA is relieved of implementing SDRAM memory controllers which greatly simplifies application design and provides a maximum of T-FPGA resources for the application. The circuitry implemented in the AL-FPGA forms the module's firmware which handles all data transfers, T-FPGA configurations and readback, and other control functions.

Figure 2. TKDM target FPGA schematic



Target FPGA The T-FPGA implements the actual co-processor application. The interface of the T-FPGA to the rest of the system is kept simple, as shown in Fig. 2. The T-FPGA can access the data lines for both SDRAM groups (X side and Y side). The bus switches that are controlled by the AL-FPGA determine which of the memories (A or B, respectively) are actually being connected to the T-FPGA ports. As mentioned above, the generation of SDRAM addresses is done by the AL-FPGA. There are 42 bidirectional wires running between T-FPGA and AL-FPGA. Seven of these signals are routed to global clock inputs of the T-FPGA which allows to feed the T-FPGA with clock signals generated by digital clock managers (DCM) in the AL-FPGA. All wires that are not used for routing clocks can be used to implement domain-specific functions. For example, a firmware for encryption applications could use these wires to provide encryption keys to the co-processor in the T-FPGA.

Figure 3. TKDM module



Reconfiguration During normal operation the T-FPGA is configured by the AL-FPGA via SelectMAP. The AL-FPGA can also perform partial reconfiguration and read-back. The configuration data for the T-FPGA is either received via the DIMM bus or read from an on-board memory. For fast reconfiguration, several T-FPGA bitstreams can be held in the on-board RAM. The AL-FPGA is configured from the configuration PROM at startup. Both FPGAs and the configuration PROM can alternatively be reprogrammed via a JTAG interface.

3.3 Implementation

Fig. 3 shows the first TKDM version that has been implemented on an impedance-controlled, 6-layer PCB board. The board's dimensions are 13$\times$9 cm which is about 2.5 times the height of a regular DIMM memory module.

The AL-FPGA is a Xilinx Virtex-II XC2V1000 in FG456 package which provides not only sufficient logic capacity for complex firmware but also the more than 300 I/O pins required to attach the DIMM bus, on-board RAMs and the T-FPGA. The Virtex-II digital clock managers (DCM) have been proven very useful for coping with the stringent timing constraints demanded by the DIMM bus. The AL-FPGA is configured from a Xilinx XC18V04 PROM at startup. Additionally, the JTAG ports of the AL-FPGA and the PROM build a chain that is accessible through an external header. This JTAG chain is used for reprogramming the PROM as well as for in-circuit debugging using Xilinx ChipScope.

The T-FPGA is also a Virtex-II device. Virtex-II devices in fine-pitch BGA packages are footprint-compatible to some extent. We have designed the board such that any Virtex-II in an FG676 package (i.e., XCV2V1500/2000/3000) or an FG456 package (i.e., XC2V250/500/1000) can be used without any modification of the PCB. This allows for building TKDM variants ranging from low-cost, low-capacity modules to more expensive high-end variants.

TKDM uses four Micron MT48LC4M32B2 SDRAM devices for on-board memory running at 133 MHz. The board further features two high-efficiency switched DC/DC converters to generate the 3.3V for FPGA I/Os, SDRAMs, and bus-switches as well as the 1.5V FPGA core voltage. The power can either be taken from the DIMM bus or from an external 5V power connector. For debugging purposes, 16 general purpose I/O pins of the T-FPGA are routed to an external connector.

4 System Integration

We have integrated TKDM with the Linux operating system on a Intel Pentium based PC. We have chosen Linux mainly due the the free availability of the kernel source code and the good documentation on writing device drivers. Fig. 4 outlines the system integration of the TKDM module.

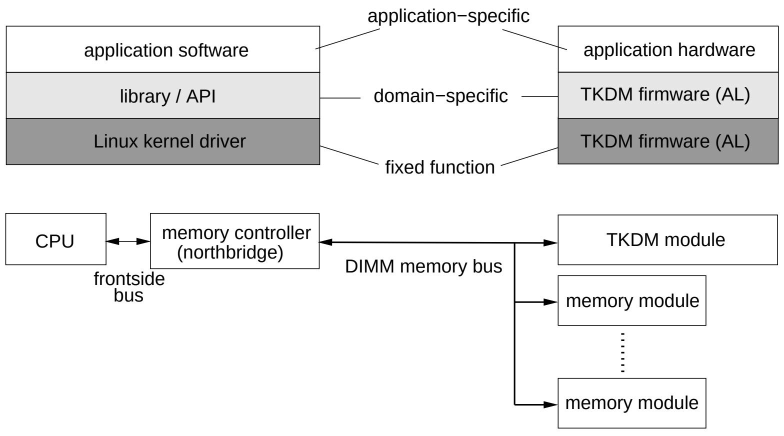
The TKDM environment comprises a software integration and a hardware integration. Both parts are layered and split into a fixed-function part and domain-specific part. The software layers on the host CPU consist of a *Linux kernel driver* and an accompanying *library*. The kernel driver provides low-level access to the firmware functions on the TKDM module. The library forms a domain-specific application programmer interface (API) and encapsulates calls to the kernel driver. Similarly, the hardware layers on the TKDM consists of a *fixed-function firmware* and a *domain-specific firmware* for the AL-FPGA. Finally, an application consists of *application software* running on the PC and the actual co-processor, the *application hardware*, mapped to the TKDM module.

4.1 Linux kernel driver and fixed-function firmware

Linux kernel driver The TKDM Linux driver has two main purposes, setting up the PC's memory controller at startup time such that the TKDM modules can be accessed by the operating system, and providing low-level access to the TKDM firmware functions at runtime.

When a PC is booted, the BIOS detects all memory modules in the system. For each DIMM slot, the BIOS probes the on-board identification PROM which must be present on any compatible DIMM memory module. This PROM informs the BIOS about the type, the size, and the timing parameters of the memory module. This information is used to initialize the PC's memory controller which then maps all memories found into the CPU's address space. The operating system is booted after this intialization phase and relies on the memory configuration detected by the BIOS. In a

Figure 4. TKDM system integration



typical PC, the memory controller is integrated in the so-called northbridge which is a part of the chipset on the PC's motherboard.

The TKDM is a memory-mapped I/O device instead of a memory, and is therefore not equipped with an identification PROM. Consequently, the module is not visible to the BIOS and, thus, also not known to the operating system at startup. The first task of the TKDM Linux kernel driver is to probe all DIMM memory slots for which the BIOS could not detect memory. If a TKDM module is found in such a slot, the driver re-programs the PC's memory controller such that the TKDM module is assigned a certain physical memory area. The next step is to map this physical memory area into the Linux address space as an uncachable memory area. Allowing the memory controller to cache the TKDM memory space could potentially result in undesired read/write behavior as the exact order of memory accesses is undetermined.

During runtime, the Linux kernel driver sends commands to the TKDM firmware using memory mapped I/O. The driver provides a number of functions that provide convenient wrappers to the firmware functionality.

Fixed-function firmware The TKDM module emulates a memory module with four memory banks. Bank 0 implements a set of memory-mapped control and status registers. Banks 1 and 2 are used to access the on-board SDRAMs for the X side and Y side, respectively. Finally, bank 3 directly maps internal BlockRAM of the AL-FPGA to the corresponding address space. The Linux driver communicates with the module by issueing commands to the control registers in memory bank 0. Data is transferred by writing/reading to/from memory banks 1 and 2. The main function groups of the fixed firmware on the TKDM module are:

- *Reconfiguration modes* control the access to the configuration port of the T-FPGA. The functions include reconfiguration, re-initialization, and readback. Both full and partial bitstreams are supported. The configuration data can either be streamed from the host CPU (via a memory-mapped register in bank 0) or read from any on-board SDRAM.

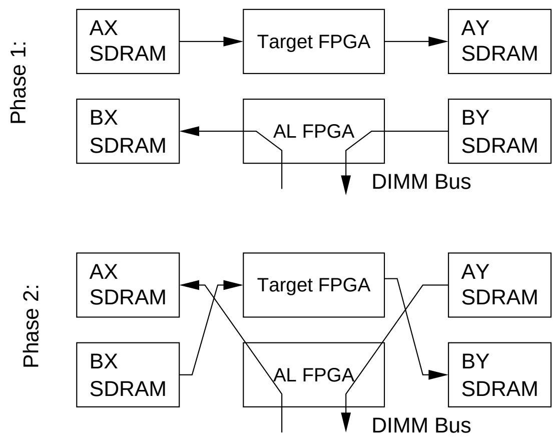
- *Switch modes* control the settings of the two bus switches. The bus switches for the X and Y side can be controlled independently.

- *Memory access modes* control the behavior of the memory controllers for the four banks of on-board memory. The DIMM bus interface cannot access the memory modules on the board because the RAMs are not directly connected to the DIMM bus. Any data transfer is managed by DMA controller channels that are a part of the memory controller implemented in the AL-FPGA fixed firmware. These DMA channels are used for data transfers from the CPU to the on-board RAMs and vice versa, as well as from the on-board RAMs to the T-FPGA and vice versa.

  As an example, consider a case where the CPU wants to transfer 100 words to the AX SDRAM starting at address 500. A corresponding write request is written into a control register in bank 0 of the AL-FPGA. The firmware decodes the request and programs a DMA channel for the AX memory with the base address of 500. After that, the firmware will transfer any subsequent data incoming on bank 1 to the AX memory, automatically incrementing the address.

- *Status modes* are used to retrieve status information for the various modes and the system, e.g., configuration state of the T-FPGA, state of the memory controllers, state of the DMA channels, and the firmware version.

Figure 5. TKDMstreaming firmware, interleaved processing



4.2 Domain-specific firmware and libraries

The TKDM concept allows to capture functionalities that many applications share in so-called domain-specific firmware and software libraries. A typical example for a domain-specific functionality is the handling of large data blocks, including data partitioning and merging, transfer to memories, etc. So far we have focused on firmware and libraries for the support of one-dimensional data streams, the application domain TKDM was designed for.

Domain-specific firmware for stream processing A stream co-processor continuously applies the same function on a large set of input data [5]. Internally, most stream processors make heavy use of deep pipelines and parallel execution. Our TKDM streaming firmware exploits the TKDM architecture which allows to overlap data processing in the T-FPGA with data transfers to and from the memories. Fig. 5 sketches the basic principle. Data processing alternates in two phases: In phase 1, data is read from the AX SDRAM, processed by the T-FPGA, and the results are written to the AY SDRAM. Meanwhile, the CPU reads back the results from the previous iteration stored in the BY SDRAM and writes the input data for the next iteration to the BX SDRAM. Then, the bus-switches are set to the cross-over mode and the T-FPGA starts processing data from the BX SDRAM to the BY SDRAM while the CPU reads the results from AY SDRAM and fills new data into the AX SDRAM.

Fig. 6 presents a schematic view of the streaming firmware and its combination with an *application hardware (core)*. In this figure, the application hardware is shown in the shaded box labelled 'streaming application'. Besides the application hardware, the T-FPGA includes two additional units, an input buffer and an output buffer. The buffers can be considered as a leight-weight interface to the domain-specific firmware. The application core reads data from the input buffer and writes the processed data to the output buffer, controlled by a few signals. The input buffer asserts `bNotEmpty` whenever there is valid data in the input buffer. Upon reading one data item from the input buffer, the application core signals `read`. There are no processing rates constraints imposed on the application core, the reading of input values can be delayed by an arbitrary number of cycles. The protocol for writing data to the output buffer is similar. The output buffer asserts `bNotFull` to denote that additional data can be buffered. When the application core writes data to the output buffer, the signal `write` is asserted.

Filling the input buffer and emptying the output buffer is done at maximum speed by the streaming firmware. The buffer status signals (`bufferNotFull` and `buffer-NotEmpty`) are fed to finite state machines in the AL-FPGA that control the DMA channels which actually access the memories. The data streaming firmware greatly eases application design as it totally hides the complexity of accessing SDRAM including refreshes, page management, dealing with latencies, etc.

Libraries Application sofware accessing the TKDM board is supposed to use a library rather than calling the TKDM Linux driver directly. The library provides a set of high-level functions (API) that are mapped to the appropriate kernel driver calls. The Linux library for streaming application contains functions for initializing the TKDM hardware, the download of the desired application hardware, and for the buffer management that allows for interleaved processing (see Fig. 5). The application software does not have to deal with controlling bus-switches and setting up DMA transfers.

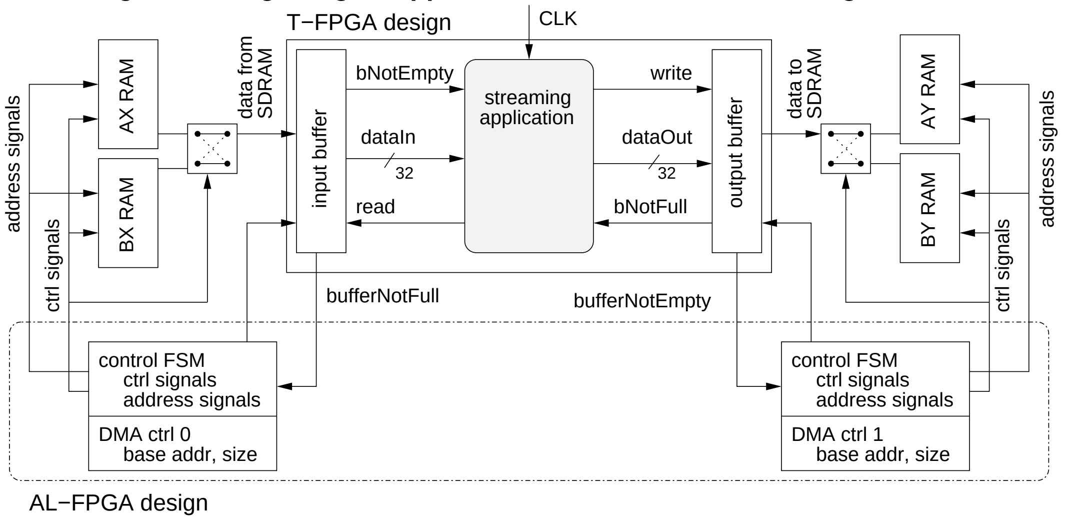
5 Project Status

TKDM is work in progress. A first TKDM version as described in Section 3.3 has been produced and basic tests have been conducted. The module is functionally fully operational and currently we are in the process of optimizing the different layers of firmware and support software, i.e., Linux driver and libraries.

5.1 AL-FPGA performance

First benchmarks showed the following throughputs: 128 MB/s for writes from CPU to TKDM and 53 MB/s for reads to CPU from TKDM. These figures were taken on a host computer with an Asus P3B motherboard and an Intel

Figure 6. Integrating an application into the TKDM streaming firmware



Pentium 500 MHz CPU with an Intel i840 chipset. Caching has been disabled for the address space used by TKDM. The motherboard supports PC100 SDRAM memory technology. We have benchmarked the read/write performance from the host CPU to the AL-FPGA by reading and writing 1024 words to TKDM memory bank 3, which is mapped to internal BlockRAM by the AL-FPGA. Reading/writing the 1kByte data block is iterated 256 times with four 64-bit transfers per iteration. The transfer commands were in-line assembly coded and used an MMX register to transfer 64 bits at once.

5.2 Current Limitations

The current throughput limitation is due to the specifics of the host CPU and its memory controller (Intel i840 chipset). An analysis of the DIMM bus traffic shows that although the memory controller issues memory bursts of length 4, only the very first word of each burst is used and the following 3 words are masked out. The resulting throughput is significantly lower than the theoretical limit of the PC100 SDRAM interface, which amounts to 800 MB/s. We assume that disabling the cache is the cause for this behavior. The cache line size for the Pentium3 processor is 256 bits. A cache line is usually read/written by bursts of length 4. For cachable address regions we always observed burst reads/writes.

Ongoing work investigates the use of DMA memory transfers and several caching options. One specific option is to activate caching and issue cache flushes whenever the order of accesses to the memory mapped registers in the AL-FPGA is of importance. However, Pentium3 processors only allow to flush the whole cache which makes this method quite inefficient. Since Intel Pentium4 allows for a more fine-granular control of the cache, e.g., flushing single cache lines, we have already started moving our development to a Pentium4 based system.

6 Summary and Future Work

In this paper we have presented the TKDM platform that attaches a reconfigurable co-processor to the DIMM memory bus. We have shown the hardware architecture and it's integration with the Linux operating system. The TKDM project is work in progress; especially the domain-specific firmware and libraries for streaming applications are still under development.

While a first test has validated the functionality of the hardware, the fixed-function firmware and the Linux kernel driver, further work remains to be done toward the analysis and optimization of the performance and the implementation of complete applications.

On-going and future work includes:

- Improving the communication performance by porting the fixed-function firmware to a Pentium4 motherboard with a 133MHz DIMM memory bus.
- Investigating whether the advanced cache control of Pentium4 and DMA transfers can increase the communication bandwidth.
- Completing the data streaming firmware and software libraries.
- Implementing complete applications to study the co-processor's impact on the system level performance

Acknowledgments

We would like to thank Andreas Schweizer for his excellent work with the design, implementation, and test of the TKDM that he did during his Master's thesis. A detailed report on the TKDM hardware architecture is available in [6].

This work is supported by the Polyproject *Wearable Computing* at ETH Zurich.

References

[1] www.xilinx.com/prs_rls/nurondw.htm.

[2] J. Gause, P. Cheung, and W. Luk. Reconfigurable shape-adaptive template matching architectures. In K. Pocek and J. Arnold, editors, *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE Computer Society Press, 2002.

[3] S. Hauck. The Roles of FPGA's in Reprogrammable Systems. *Proceedings of the IEEE*, 86(4):615–638, Apr. 1998.

[4] P. Leong, M. Leong, O. Cheung, T. Tung, C. Kwok, M. Wong, and K. Lee. Pilchard - a reconfigurable computing platform with memory slot interface. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE Computer Society, April 2001.

[5] O. Mencer, H. Huebert, M. Morf, and M. J. Flynn. StReAm: Object-Oriented Programming of Stream Architectures using PAM-Blox. In *Proceedings of the International Workshop on Field-Programmable Logic and Applications (FPL)*, 2000.

[6] A. Schweizer. Reconfigurable Computing on a DIMM module. Master's thesis, Computer Engineering Lab, ETH Zurich, Switzerland, March 2003.

[7] D. Tong, P. Lo, K. Lee, and P. Leong. A System Level Implementation of Rijndael on a Memory-Slot based FPGA Card. In *Proceedings of the 2002 International Conference on Field Programmable Technology (FPT)*, pages 102–109, 2002.

[8] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard. Programmable Active Memories: Reconfigurable Systems Come of Age. 4(1):56–69, March 1996.