Windowed FIFOs for FPGA-based Multiprocessor Systems

Kai Huang David Grünert Lothar Thiele
khuang@tik.ee.ethz.ch davidgr@ee.ethz.ch thiele@tik.ee.ethz.ch
Swiss Federal Institute of Technology Zurich
Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zurich, Switzerland

Abstract

FPGA-based multiprocessor systems are viable solutions for stream-based embedded applications. They provide a software abstraction which enables coarse-grained parallel deployment on an FPGA chip. A widely used model for such a deployment is the class of Kahn process networks despite their limitation to pure FIFO communications. In this paper, a new mechanism denoted as windowed FIFO is introduced, extending the functionality for data transfer. The new concept allows non-destructive read, reordering, and skipping of data within a communication channel. We present the behavior, the software interface and the hardware design of this mechanism. We introduce our abstraction of WFIFO process network which is suitable for systematic and automated synthesis while still inheriting the nice property of Kahn process networks, i.e. being determinate. Also, we present illuminating examples to demonstrate the practicality of the outlined approach.

1. Introduction

The class of emerging multimedia and signal processing systems such as mobile-phones and set-top boxes demands high computing power. Different possibilities exist to meet these requirements, but multiprocessor architectures have turned out to be best suited with respect to their high computing power and low power consumption. Meanwhile, the FPGA technology available today allows the deployment of such multiprocessor architectures in a single chip. For instance, the Xilinx' Embedded Development Kit (EDK) [1] provides tools and a library of IP cores for multiprocessor developments on the Virtex [2] family of FPGAs. This environment integrates on-chip PowerPC cores, soft MicroBlaze cores, distributed memory, and customizable peripherals, which helps users to shift their focus from architecture details towards deploying application-level parallelism.

Kahn process networks (KPN) [3] are widely chosen for modeling multiprocessor applications at process-level because of their simple communication and synchronization mechanisms. A KPN model consists of a set of concurrent processes connected exclusively by point-to-point FIFO channels. Each process follows a sequential program and uses only FIFOs for data exchange. These FIFOs have unlimited size and a blocking read semantics that blocks the reading process as long as there are no data available. Given these properties, the KPN model is determinate, i.e., the functional input/output relation is independent of the timing of the processes. The intrinsic distributed control and distributed memory of the KPN model match as well the modern FPGA technique. A dedicated structure as shown in Fig. 1 which fully reflects the structure of a KPN can be mapped onto a single chip. In such a dedicated system, each processor has its own bus, and additional subsystems are used to implement the blocking read FIFO channels.

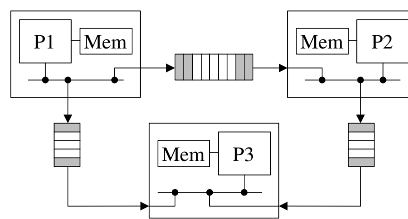

Figure 1. WFIFO example architecture.

Although the KPN architecture offers distributed control and a simple interface for programming, it also has limitations that make the implementation of stream-based applications difficult or inefficient. Three limitations that are of interest for such applications are addressed in this paper: 1) *Reordering*: A KPN communication channel behaves in a strict first-in first-out manner, which does not allow reading data in an order other than what they have been written. 2) *Non-destructive read*: A KPN communication channel does not allow reading the same data item more than once. After an item is read, it will be deleted from the channel and will not exist in the channel anymore. 3) *Skipping*: It is not possible to remove an item from a channel without reading it. Even if the channel contains unwanted data, all of which have to be read out for accesses of subsequent data.

In this paper, we introduce a novel FPGA-based solution denoted as windowed FIFO (WFIFO), extending the functionality for data transfer. The new concept tackles in a particular way the three limitations mentioned above, allowing non-destructive read, reordering, and skipping of data within a parameterizable length window in a communication channel. The contributions of this work can be summarized as follows:

- We introduce a WFIFO concept of communication for multiprocessor systems. This includes the definition of the software API, hardware, and the target architecture.
- We introduce our abstraction of WFIFO process networks which is applicable to systematic and automated synthesis

while still inheriting the nice property of a KPN, i.e. being determinate.

- We implement this concept in a VirtexII-Pro FPGA and the corresponding IP-block is fully compatible with the Xilinx EDK environment.
- We present illuminating examples to demonstrate the practicality of the outlined approach.

2. Related Work

Discussions on the extension/restriction of the KPN model have a long history and many publications are available. In [4], an implementation of a FIFO data-synchronization scheme is presented that can be used in the functional description and hardware realization for heterogeneous shared-memory multiprocessor systems. Since the presented approach targets shared-memory architecture, it not only has to clearly separate synchronization from data transportation but also cannot benefit from the nice properties of distributed control and distributed memory offered by modern FPGA techniques. In [5] which extends the way presented in [6], bunches of task-level interfaces are summarized for the structured design and programming of embedded multiprocess systems where our original idea is from. These interfaces are available as C/C++ API, which leaves the hardware support issue still open. Therefore there is no way to synthesize a system that uses these interfaces into an FPGA chip. In this paper, we give a more restrict definition and an FPGA-based implementation.

SHIM [7], a synthesizable design flow, uses a restricted KPN model, the communication of which is synchronous in the sense that both sending and receiving processes must agree when data are to be transferred. In [8], an exploration framework is presented to build FPGA multiprocessor systems for stream-oriented applications, generating homogeneous networks of MicroBlaze processors interconnected by buses and direct FSL links. [9] generates both homogeneous and heterogeneous FPGA synthesizable multiprocessor systems based on KPN models as well. All of these frameworks stick to a strict FIFO communication, therefore none of them can handle the three limitations mentioned above, i.e. reordering, non-destructive read, and skipping. Attempt trying to tackle the out-of-order access and multiplicity (non-destructive read) is presented in [10]. But the proposed methods either need special purpose memories or have to calculate a coordinate for each data item transferred. Even worse, dedicated control subsystems have to be generated for each communication channel.

3. The WFIFO Concept

A windowed FIFO (WFIFO) originates from a normal FIFO whereas it offers more functionality and flexibility. A normal FIFO behaves in a strict first-in first-out manner in which data must be read in the same order as they have been written. In addition, each data item written to the FIFO must be read exactly once. It is not possible to read the same item multiple times or to remove an item from the FIFO without reading it. Unlike a normal FIFO, the WFIFO supports out-of-order access within a continuous data segment located at the head or tail of the FIFO. These segments are called windows, which leads to the name windowed FIFO. The characteristics of the WFIFO can be summarized as followed:

- A WFIFO has two access ports, i.e. a read port and a write port, which support only read and write operations, respectively.
- An acquire operation to a WFIFO is mandatory before data transmissions.
- The acquired window can have an arbitrary size smaller than the size of WFIFO while the acquired read and write windows cannot overlap.
- Both blocking and non-blocking acquires are supported. The non-blocking means if the available data/spaces are less than the acquired window size, an error will return.
- The access to the window can be out-of-order and the read operation is non-destructive.
- A release operation is as well mandatory for synchronization after data transmissions.

A functionality abstraction of the WFIFO is shown in Fig. 2. The gray boxes located at both ports of the FIFO are in some sense reordering memories which support the new functionalities. At each port, a logic block controls the access of the reordering memory. For instance, when the logic of the write port receives an acquire-write instruction, it enables a write reordering memory. This memory has a continuous address space and a size as big as the required window. Subsequent write instructions are directed to this memory at a position indicated by an offset specified in the instructions. When a release instruction is received, writing is disabled and the data in the reordering memory will be merged to the FIFO without changing their order. Remind that Fig. 2 is only a functionality abstraction. In the WFIFO hardware implementation, a single memory block is used instead of the combination of a FIFO and reordering memories.

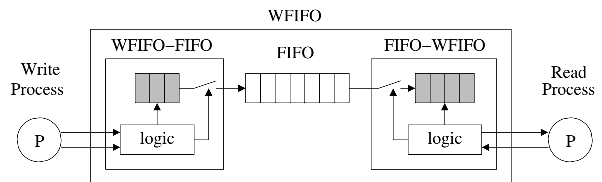

Figure 2. Functionality abstraction of the WFIFO. The reordering memories are indicated in gray.

Based on the window operations, a WFIFO can perform out-of-order access (reordering), non-destructive read and skipping. Out-of-order access allows data items to be consumed in an order other than produced. Both the producer and the consumer can reorder the data within the acquired windows. Non-destructive read means that the same data item can be read an unlimited number of times as long as it is still in the read window. If the consumer does not read all data within the read window, this can be seen as skipping of unwanted data. Skipping makes sense if it is not possible for the producer to know if data will be required by the consumer or not. A sample application depicting the usefulness of the skipping is shown in Section 4.2.

3.1 WFIFO Process Network Model

A WFIFO process network derives from a KPN, which consists as well of concurrently running sequential processes that communicate exclusively through point-to-point communication channels. The difference is that the unbounded FIFO channels are replaced by bounded WFIFO channels. We will show that a WFIFO process network with both blocking read and blocking write semantics is determinate.

LEMMA 1. *A WFIFO process network with blocking semantics can be simulated by a KPN with blocking write semantics as well.*

PROOF. The fundamental difference between a WFIFO process network and a KPN with blocking write semantics is the replacement of FIFO channels with WFIFO channels. It must be shown that the windowed semantics can be implemented using a bounded FIFO channel. Given the property that the read and write windows do not overlap each other, the segment in between is strict FIFO semantics. The window mechanism can be transformed into a software implementation in the connected local processes. For the write process, when an acquire operation is invoked, a segment in the local memory with the same size as the acquired window is allocated. Subsequent write operations are all redirected to this segment until a release operation is received where all data within this segment will be written to the FIFO channel. For the read process, when an acquire is invoked, a segment in the local memory with the same size as the acquired window is allocated as well. Meanwhile, data as many as the acquired window can contain are removed from the FIFO to this segment. All subsequence read operations are redirected to this segment, i.e. reading from this piece of local memory, until a release operation where this segment will be freed. Under this transformation, a WFIFO process network with blocking semantics can be realized by a KPN with the same semantics. □

LEMMA 2. *A KPN with blocking write semantics can be simulated by a KPN with non-blocking write semantics.*

PROOF. A bounded FIFO implementation of a KPN leads to a blocking write semantics whereas a non-blocking write semantics of a KPN implies an unbounded size of FIFO channels. [7] presents a way how a buffer with rendezvous protocol can be implemented with unbounded FIFOs. Here a similar proof is made for bounded FIFOs that is not restricted to the rendezvous protocol. For any bounded FIFO F_i with size S_i in a process network, a pair of unbounded FIFOs is replaced, where one (forward) has the same direction as F_i and the other (backward) goes in the opposite direction. The backward FIFO is set to full initially, i.e. containing S_i data items. The access semantics is changed as followed: The writing process always reads an item from the backward FIFO before writing an item to the forward FIFO. The reading process writes an item to the backward FIFO immediately after reading an item from the forward channel. This backward channel mechanism guarantees that there are never more than S_i data items in the forward FIFO, which is tantamount to a bounded FIFO with size S_i. Under this transformation, a bounded FIFO can be implemented with a pair of unbounded FIFOs with non-blocking write semantics. Therefore, the lemma holds. □

THEOREM 1. *A WFIFO process networks with both blocking read and blocking write semantics is determinate.*

PROOF. Combining Lemma 1 and Lemma 2, a WFIFO process network with blocking semantics can be simulated by a KPN, therefore it inherits the properties of the KPN. The theorem holds. □

COROLLARY 1. *A KPN can be simulated by a WFIFO process network.*

PROOF. Fixing the window size to 1 data item, a WFIFO process network is equivalent to a KPN. □

From Corollary 1, we can find out that a WFIFO process network model is an extension of a KPN model where the first-in first-out semantics of a KPN FIFO is relaxed to bounded-size reordering windows at both ends of the FIFO channel. While gaining efficiency and flexibility from the bounded-size reordering windows from the programming point of view, an inevitable side effect is the artificial deadlock. Although how to decide the size is difficult in practice, at least the WFIFO model inherits the nice property of being determinate, i.e. giving correct result if deadlock does not occur.

3.2 WFIFO Software API

Writing to and reading from a WFIFO are two independent transactions which both follow a protocol. The protocol is defined by three subsequent steps. The enumeration below shows these steps for the write port of the WFIFO.

- Acquire a write window:
  WFIFO_ACQUIRE_WRITE(port, size)
  WFIFO_ACQUIRE_WRITE_NONBLK(port, size)

  The acquire-write instruction allocates a window of the *size* at the write port of the WFIFO. The *port* number identifies the WFIFO in case there is more than one WFIFO connected to the same processor. There are two versions of the acquire instruction, a blocking and a non-blocking one. Both versions execute the acquiring in case there is enough memory available. If there is not enough memory space, the blocking version blocks the calling process until enough memory space is available. The non-blocking instruction returns in any case, and signals with the return value if acquiring is successful or not, which is orthogonal to the non-blocking write semantics of a KPN where infinite memory is assumed. The acquire instruction is followed by step two or step three. Acquiring can only be executed if there is no write window at the write port.

- Write data:
  WFIFO_WRITE(port, offset, data)

  The write instruction writes the *data* to the write window at the position indicated by the *offset*. The instruction can be repeated an unlimited number of times. It is possible to write the same offset position more than once, which results in overwriting the old value. If an offset position is never written, its value is undefined. The write instruction can be followed by the next write instruction or by step three.

- Release the write window:
  WFIFO_RELEASE_WRITE(port)

  The release instruction terminates the writing phase and shifts the content of the window to an internal FIFO, from

where it can later be read. After releasing, no further writing to the window is possible. Before more data can be written to the channel, a new write window must be acquired.

The protocol for the read port is very similar. It also offers two acquiring functions — a blocking and a non-blocking one. The acquiring can be executed successfully if there are enough data in a WFIFO buffer for the read window. After acquiring succeeds, data can be read from the window. The release instruction is equivalent to deleting the data of the acquired size from the channel.

3.3 WFIFO Hardware Design

Using the Xilinx Embedded Development Kit (EDK) 7.1i [1], our WFIFO IP is implemented on a ML300 Evaluation board where a Virtex-II Pro FPGA chip (xc2vp7-ff672-6) [2] is integrated. The EDK offers two types of processor IP, i.e. MicroBlaze soft IP (MB) and IBM PowerPC 405 core (PPC), and three types of bus, i.e. On-Chip Peripheral Bus (OPB), Local Memory Bus (LMB) and Processor Local Bus (PLB). The OPB can serve both types of processors while the high speed PLB serves only the PPC.

An overview of the WFIFO architecture is given in Fig. 3. In order to construct the WFIFO IP generic enough for heterogeneous architecture, two modified Intellectual Property Interface (IPIF) are integrated inside that offers by Xilinx EDK a standardized IP interface at one side and a specific bus interface at the other side. With such a design it is possible to connect the same IP to different types of buses by means of different IPIF implementations. The WFIFO memory is implemented with a dual-port BlockRAM (BRAM) which is an on-chip parameterizable memory module available on all newer Xilinx FPGAs. The access of the BRAM is controlled by a state machine which is the heart of the WFIFO implementation.

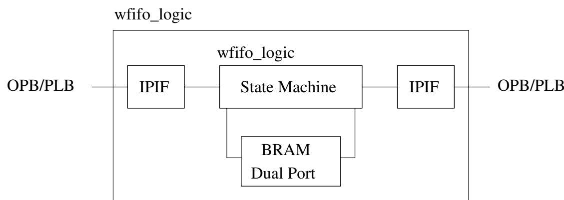

Figure 3. WFIFO architecture overview.

The architecture of the state machine is shown in Fig. 4. For efficiency reasons, the design implements two synchronized state machines. One state machine is connected to the write port and handles all write commands, while the other one does the same for the read port. Synchronization is achieved over a set of shared states. The reason for this design rather than a single state machine is that the implementation with the single state machine has to encode not only all possible read and write states but also all possible combinations of them.

Using the built-in synthesis tool within EDK 7.1i, the size of the WFIFO IP in terms of the Configurable Logic Blocks (CLB) – a.k.a. Slice, is 377, which is less than 1% of the total CLB (4928) available on the chip. In the pipeline mode 2, write and read operations take 5 and 6 clock cycles respectively, which is acceptable comparing to 2 cycles for both read and write afforded by the high-speed, dedicated point-to-point first-in first-out Fast Simplex Link (FSL) in the EDK library. The IP works can work

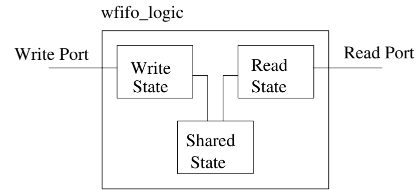

Figure 4. WFIFO state machine architecture.

up to 173.6 MHz limited by the IPIF component in the EDK. In order to support future automated system generation, the WFIFO IP is designed as a separate component fully compatible to Xilinx EDK environment and is parameterizable in terms of the BRAM size and the window width. Because of the space limit, we do not list all the details. The detailed statements and timing diagrams are described in [11].

3.4 Target System Synthesis

Synthesizing a WFIFO system starts with modeling the application behavior as a WFIFO process network. This resembles with modeling an application with KPN with the exception that normal FIFO channels are replaced by WFIFO channels. The application model includes the source code for the concurrent processes and the topology of the underlying process network. For the communication between the processes, WFIFO channels are used which are accessed using the WFIFO software API. Fig. 5 shows a topology of a WFIFO process network with three processes and sample source codes of the processes P_1 and P_2 are given in Listing 2.

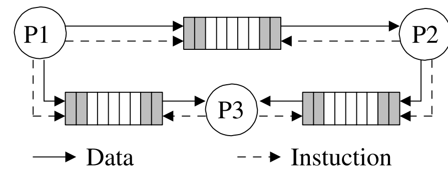

Figure 5. Example WFIFO process network with P1 as data source and P3 as data sink.

The second step is to build a target architecture. To reduce the complexity, the target architecture has a dedicated structure and is assembled from two building blocks shown in Fig. 6, one computation component and one communication component. The communication component is the WFIFO IP. The computation component combines a processor core, a bus and a piece of memory for instruction and data memory. WFIFOs are connected with the processors over a bus. WFIFO software API translates WFIFO instructions into bus transactions. The port numbers are used to the bus address calculation of the WFIFO. This translation from port number to bus address can be done at compile time.

To build a WFIFO system, a WFIFO process network needs to be mapped onto a target architecture, i.e., for each computation component onto which WFIFO processes are mapped as well as for each communication component onto which a WFIFO channel is mapped. In the current stage, we only consider a one-to-one mapping, i.e. mapping a WFIFO process to a computation

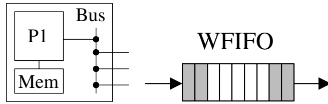

Figure 6. Building blocks for the WFIFO architecture. Mem is the instruction and local data memory of processor P1.

component. No mapping is required for WFIFO channels because each channel is mapped onto exactly one WFIFO hardware IP, the connections of which are well defined by the topology of the process network and the mapping of the processes. Fig. 1 depicts a resulting WFIFO system by a one-to-one mapping of the process network shown in Fig. 5.

4. Application Examples

In this section, we present illuminating application examples to demonstrate the practicality of the outlined approach.

4.1 Reordering and Multiple Read

Reordering and multiple read are common tasks in signal processing algorithms. A simple producer-consumer example is shown in this section. The original sequential code is in Listing 1 where the producer generates a two-dimensional array in the column order and the consumer read the array in the row order. Moreover, the consumer will read the second row of the array twice.

```
// P1: producer                  // P2: consumer
while(true) {                    while(true) {
  for (i=0; i<2; i++) {            for (j=0; j<2; j++) {
    for (j=0; j<3; j++) {            process(A[0][j]);
      A[i][j]=nextValue();           process(A[1][j]);
    }                                process(A[0][j+1]);
  }                                  process(A[1][j+1]);
  ...                              }
}                                  ...
                                 }
```

Listing 1. A sequential code for a simple reordering producer-consumer example.

To rewrite this piece of sequential code into a KPN becomes complicate. In the case of out-of-order access, either the producer or the consumer has to rearrange the execution order of its code. In the case of multiple accesses of a data item, either the producer has to send the data item again or the consumer has to store the data item somewhere in its local memory. All of these ad-hoc solutions are a bit involved from the programming point of view.

Using a WFIFO, the code as shown in Listing 2 becomes straightforward. The consumer first acquires a read window from the WFIFO, then it can read the array in arbitrary order no matter in what kind of order the producer generates the array. Meanwhile, the consumer reads 8 data items within the acquired window from the WFIFO although the producer only writes 6 data items. In addition, the storage of the array shifts from the local memory of the process to the WFIFO, which can reduce the local memory size of the process as a side effect.

```
// P1: producer
while(true) {
  WFIFO_ACQUIRE_WRITE(0, 6);
  for (i=0; i<6; i++) {
    WFIFO_WRITE(0, i, nextValue());
  }
  WFIFO_RELEASE_WRITE(0);
  ...
}

// P2: consumer
while(true) {
  WFIFO_ACQUIRE_READ(0, 6);
  WFIFO_READ(0, 0, tmp); process(tmp);
  WFIFO_READ(0, 3, tmp); process(tmp);
  WFIFO_READ(0, 1, tmp); process(tmp);
  WFIFO_READ(0, 4, tmp); process(tmp);
  WFIFO_READ(0, 1, tmp); process(tmp);
  WFIFO_READ(0, 4, tmp); process(tmp);
  WFIFO_READ(0, 2, tmp); process(tmp);
  WFIFO_READ(0, 5, tmp); process(tmp);
  WFIFO_RELEASE_READ(0);
  ...
}
```

Listing 2. The corresponding WFIFO code for this simple reordering producer-consumer example.

4.2 Skipping

A possible application for the WFIFO concept is in the image filtering, where an image is divided into slides or macroblocks then processed by a multiprocessor architecture in parallel. For such an application it is typical that at a certain point of the processing, information from the neighboring processors is required. For this type of data exchange, WFIFOs can be applied. Instead of an image processing algorithm, we discuss here a parallelization of the game of life [12] which has similar requirements but is less complex.

The "world" for the game of life is made of a rectangular array of cells which are either dead or alive. The state of a cell in the next cycle depends on its actual state and on the states of its eight immediate neighbors. A live cell with more than three live neighbors dies. A dead cell with exactly three live neighbors becomes alive, otherwise it remains dead.

The game of life can be parallelized by segmenting the world into horizontal slides. Each processor computes the next states for one such slide. For the cells at the boundaries of a slide, the processor requires cell states from the neighbor to make a decision for the next cycle. There are conditions where a processor can make such a decision without knowing the state of the cells from neighbor slide. Fig. 7 shows two such situations where the states of the cells from neighbor are not required. The key point here is that only the consumer of the data can decide if it requires the data or not while the producer cannot and has to send the data all along. With a channel that does not allow skipping, all states of the cells next to the boundaries of the tile must be written to the channel and they must be read by the neighbor processor. With a WFIFO buffer, only writing is mandatory. After acquiring a read window, the reading processor is free to decide whether it reads the data or not.

We built a two-processor system, trying to simulate a 14×24 world, the initial state of which is shown in Fig. 8. We split

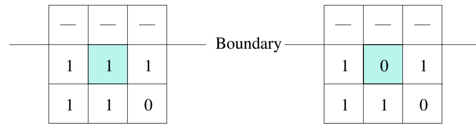

Figure 7. Two situations where the next state of the cell indicated in gray is known without reading the neighbor states beyond the boundary.

the world into two equivalent slides, the upper and lower part corresponding to the figure, each of which will be separately processed by a MicroBlaze processor. In each cycle, the two processors exchange the states of the neighboring two lines in their own slides to compute the states of the joint boundary lines for the next cycle. We run the system for one hundred iterations where 5% of the total reads from the WFIFO connecting to the adjacent processor is avoided. The unused data in the WFIFO has just simply been skipped. It is also easy to observe the amount of skipping read increases as the live cells at the boundary increased. Consider an extreme case where cells at the four rows along the boundary are all alive, the skipping read can reach 15%.

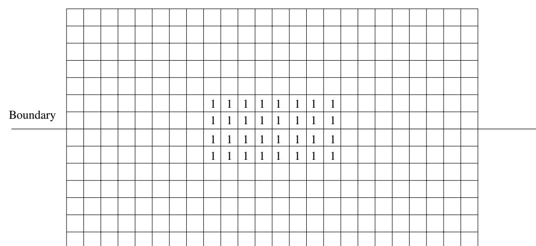

Figure 8. Initial state of the experiment.

The achievement from this simple application seems trivial because a data item to be read is only 1 byte. When data items are however at the block/macroblock level of images, the skipping functionality can reduce enormous amount of data transmissions and I/O operations.

5. Conclusion and Future Work

In this paper, We introduce the WFIFO mechanism and present the model, the software interface, and the hardware implementation of this mechanism. Without loosing favorable properties of the traditional KPN model, the WFIFO offers additional Reordering, non-destructive read, and skipping mechanisms. Although leaving the usage of these three mechanisms into the hands of the designers might cause uncertainty, the WFIFO however simplifies the implementation of applications and makes this concept applicable for a wider range of problems.

In the future, we plan to integrate this work into the ESPAM [9] design flow in order to achieve automated generation of both heterogeneous and homogeneous systems. We will also try to automate the parameters of the IP with respect to concrete applications and conduct sophisticated case studies, e.g. packets transmission in VoIP, as a reviewer suggested.

6. Acknowledgement

This research has been funded by European Integrated Project SHAPES under IST Future Emerging Technologies - Advanced Computing Architecture (ACA). Project number: 26825. A special thanks to Dr. Todor Stefanov for kindly lending the FPGA board.

7. References

[1] "Xilinx Platform Studio and the Embedded Development Kit, EDK version 7.1i edition.." `http://www.xilinx.com/ise/`.

[2] "Alpha Data Parallel Systems, Ltd.." `http://www.alpha-data.com/adm-xpl.html/`.

[3] G. Kahn, "The semantics of a simple language for parallel programming," in *Proceedings of IFIP Congress 74, Stockholm, Sweden, August 5-10 1974*, pp. 471–475.

[4] O. P. Gangwal, A. Nieuwland, and P. Lippens, "A scalable and flexible data synchronization scheme for embedded hw-sw shared-memory systems," in *Proceedings of ISSS, October 1-3, 2001, Montral, Qubec, Canada.*

[5] P. van der Wolf, E. de Kock, T. Henriksson, W. Kruijtzer, and G. Essink, "Design and programming of embedded multiprocessors: an interface-centric approach," in *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, (New York, NY, USA), pp. 206–217, ACM Press, 2004.

[6] E. A. de Kock, J.-Y. Brunel, K. A. Vissers, P. Lieverse, P. van der Wolf, W. M. Kruijtzer, W. J. M. Smits, and G. Essink, "Yapi: Application modeling for signal processing systems," in *Proceedings of 37th Conference on Design Automation (DAC'00)*, pp. 402–405.

[7] S. A. Edwards and O. Tardieu, "Shim: A deterministic model for heterogeneous embedded systems.," in *Proceedings of the ACM Conference on Embedded Software (Emsoft), Jersey City, NJ, September 2005.*

[8] Y. Jin, N. Satish, K. Ravindran, and K. Keutzer, "An automated exploration framework for fpga-based soft multiprocessor systems," in *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 273–278, 2005.

[9] H. Nikolov, T. Stefanov, and E. Deprettere, "Multi-processor system design with espam," in *CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, (New York, NY, USA), pp. 211–216, ACM Press, 2006.

[10] A. Turjan, B. Kienhuis, and E. Deprettere, "Classifying interprocess communication in process network representation of nested loop programs," *IEEE Trans. on Embedded Computing Systems*, 2006.

[11] D. Grünert, "Window based fifos for communication in on-chip multiprocessor systems," Master's thesis, ETHZ, Sep. 2006.

[12] B. Gennart and R. Hersch, "Computer-aided synthesis of parallel image processing applications," in *Proceedings Conf. Parallel and Distributed Methods for Image Processing III, SPIE Int. Symp. on Opt. Science, Denver, July 1999, SPIE Vol-3817, 48-61.*