# Inter-Process Communication using Pipes in FPGA-based Adaptive Computing

*Ming Liu*[‡†], *Zhonghai Lu*[†], *Wolfgang Kuehn*[‡], *Axel Jantsch*[†]

‡ II. Physics Institute
Justus-Liebig-University Giessen (JLU), Germany
{ming.liu, wolfgang.kuehn}@physik.uni-giessen.de

† Dept. of Electronic Systems
Royal Institute of Technology (KTH), Sweden
{mingliu, zhonghai, axel}@kth.se

***Abstract*—In FPGA-based adaptive computing, Inter-Process Communications (IPC) are required to exchange information among hardware processes which time-multiplex the resources in a same reconfigurable region. In this paper, we use pipes for IPC and analyze the performance in terms of throughput, throughput efficiency and latency in switching contexts. We also present two practical implementations using FPGA BRAM and external DDR memory. Experimental results expose the key role that context switching plays in determining the IPC performance at various pipe sizes and data rates.**

## I. Introduction

Inter-Process Communication (IPC) deals mainly with the techniques which facilitate communications between processes. In computer systems based on general-purpose processors plus Operating Systems (OS), different software processes have their individual address space which is protected from out-of-bounds access of others. Therefore IPC approaches are usually provided by OSes to exchange or share data among processes. Concrete mechanisms include *pipes/FIFOs*, *shared memory*, *sockets* based on memory access, and *message passing* based on network transmission, etc [1].

Adaptive computing based on FPGA Partial Reconfiguration (PR) technology tailors various computation algorithms to ambient conditions during system run-time. It intelligently manages on-chip computing resources and improves their utilization efficiency [2]. In contrast to conventional static configuration of FPGAs, PR provides the technical support for changing only a particular section of an FPGA design, while the remaining system is still in operation. Analogous to software multiprogramming on CPUs, the PR technology enables multitasking on limited FPGA resources by dynamically configuring or unloading processing modules according to computation requirements. In this context, each processing module is treated as a hardware process [3] for one specific algorithm. In correlated multitasking computation that has inter-module communication requirements, efficient IPC mechanisms are expected to exchange information among algorithms. In this paper, we borrow the concept of *pipe* from Unix-like OSes and investigate the queuing behavior of the communications between dynamically reconfigurable hardware processes on FPGAs.

The remainder of the paper is organized as follows: In Section II, related work is addressed in the area of FPGA-based self-adaptive multitasking. In Section III, we model pipe communications in both conventional static and modern dynamically reconfigurable designs. In Section IV, communication performance using pipes is analyzed and formulated in terms of throughput, throughput efficiency and latency. In Section V, we present two pipe implementations using FPGA Block RAM or external DDR memory. In Section VI, experimental results of performance measurements disclose the basic characteristics of using pipes in practical reconfigurable designs. Finally we conclude the paper and propose our future work in Section VII.

## II. Related Work

Several key aspects have been addressed in the FPGA-based design framework of self-adaptive multitasking systems: In [3], the authors propose the concept of hardware process on partially reconfigurable FPGAs, analogous to normal software processes in OSes. They also enhance a Linux kernel for supporting hardware processes and scheduling them together with software ones; In [4] and [5], the authors discuss context saving and restoring mechanisms, when multiple hardware processes are alternately loaded into or removed from the same reconfigurable slot on FPGAs; In [6], the authors study fast online placement and scheduling of tasks with known execution time on reconfigurable devices. Additional scheduling mechanisms and hardware resource management studies can be found in [7], [8] and [9]; In [10] and [11], the authors analyze the performance and estimate design costs for adaptive streaming applications. However, all the above cited researches do not take into account inter-module communications among hardware processes. The Erlangen Slot Machine (ESM) provides a hardware platform together with its corresponding development approach to flexibly relocate design modules in reconfigurable slots. In [12] and [13], the authors analyze signal routing dilemmas and optimize communications among different modules, which simultaneously appear in parallel in different configuration slots. However another challenging topic of communications among hardware processes that time-multiplex the same computing resources in sequence are not concerned. In this paper, we will address this issue and implement practical designs on FPGAs.

## III. IPC Queuing Models

### A. Static Process Model

Pipes and FIFOs (Also called named pipes. We use pipe as the general name.) are a basic IPC mechanism provided in all flavors of Unix OSes. They are best suited to implement producer-consumer interactions among processes. A pipe is a unidirectional channel: All data written by a process to the pipe is routed by the OS kernel to another process which can thus read it [1]. In FPGA-based designs, hardware pipes behave in an analogous manner as the software ones in OSes. Figure 1 shows consecutive pipe communications between algorithm modules. Conventionally all algorithm processors or algorithm steps are statically placed on the FPGA fabric. They work in parallel to process their respective input data streams, possibly generate output results, and pass IPC information to the next computation stage. In this model, intermediate pipes with buffering capability decouple and coordinate producers and consumers, if they do not generate and consume data at the same pace.

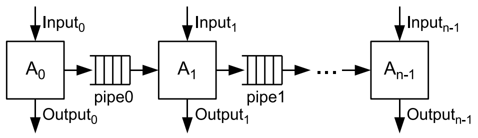


Figure 1. Consecutive pipe communications between algorithms or algorithm steps

## B. Reconfigurable Process Model

In adaptive computing scenarios in which multiple algorithm processors or algorithm steps timeshare the same resources in one PR Region (PRR), pipes can be used to bridge the communication of two modules activated at different time. In Figure 2, a reconfigurable design is demonstrated with the same dataflow as the static algorithm placement shown in Figure 1: Algorithm modules are sequentially loaded into the PR region for a period of time. They read and consume data from the previous-stage pipe, and store the generated inter-module information in the current-stage pipe for next-stage computation. For example after activated, algorithm A1 reads IPC packets of A0 from pipe0, and pass information to A2 via pipe1.

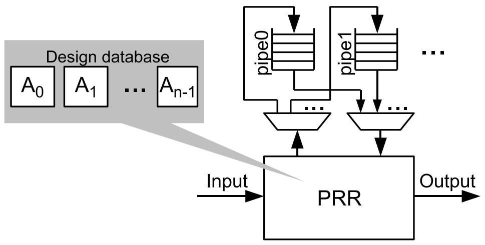


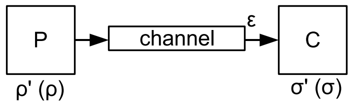
Figure 2. Pipe communications in PR designs

## IV. Performance Analysis

### A. Throughput and Throughput Efficiency

Referring to Figure 3, we introduce a metric **Throughput Efficiency (TE)** to measure the data delivery bandwidth efficiency of the communication channel in the producer-consumer model. TE is defined as the division of the effective (real) data delivery throughput ($\varepsilon$) via the channel and the minimum peak rate of data generation by the producer ($\rho$) and data consumption by the consumer ($\sigma$), as shown in Equation 1:

$$TE = \frac{\varepsilon}{Min(\rho, \sigma)} \tag{1}$$



ρ: peak data generation rate of Producer (P)
σ: peak data consumption rate of Consumer (C)
ρ': effective data generation rate of Producer
σ': effective data consumption rate of Consumer
ε: effective data delivery throughput via the channel

Figure 3. TE definition in the producer-consumer model

The minimum value of $\rho$ and $\sigma$ represents the physical limitation of data throughput in the system: The effective data delivery throughput can never exceed the smaller capability between the producer and the consumer. Therefore, the maximum value of TE is 1 (100%), which means that the communication channel is fully utilized and does not obstruct the data transmission between the producer and the consumer at all. If TE is smaller than 1, the real data delivery throughput does not reach the physical limitation, and there exists transmission efficiency loss in the channel.

In the static model as shown in Figure 1, intermediate buffers may coordinate the producer and the consumer to both work in the data generation or consumption rate of Min($\rho$, $\sigma$). For example if $\rho$ is larger than $\sigma$ (data generated faster than consumed), data will be backlogged in the buffer (pipe) and then either pause the producer from injecting or be lost. Hence the effective data generation rate ($\rho'$) is actually equal to $\sigma$ rather than the peak value $\rho$. Vice versa, it is also true for the effective data consumption rate ($\sigma'$). In this case, the real data delivery throughput $\varepsilon$ is equal to Min($\rho$, $\sigma$) and $TE_{stat}$ (static) is 1 which implies no obstacle from the channel for data transmission. It may also happen that the channel cannot even handle the data rate of Min($\rho$, $\sigma$), due to physical constraints such as bus conflicts or clock frequency limits. Thuswise $\varepsilon$, $\rho'$ and $\sigma'$ will all be smaller than Min($\rho$, $\sigma$) and $TE_{stat}$ is less than 1. By contrast in the reconfigurable model shown in Figure 2, the pipe must be firstly filled up by the producer and afterwards emptied by consumer readout, considering that the producer and the consumer multiplex the PR region and cannot work in parallel. Assuming the size of the pipe to be S, the effective data delivery throughput can be calculated with the transported data amount (equal to S) divided by the total time span for the producer to write the pipe till it is full ($T_{wr}$) and then for the consumer to read it ($T_{rd}$) till empty.

$$\varepsilon_{reconf} = \frac{S}{T_{wr} + T_{rd}} = \frac{S}{\frac{S}{\rho'} + \frac{S}{\sigma'}} = \frac{\rho'\sigma'}{\rho' + \sigma'} \tag{2}$$

Accordingly the throughput efficiency of the reconfigurable architecture is:

$$TE_{reconf} = \frac{\varepsilon_{reconf}}{Min(\rho, \sigma)} = \frac{\rho'\sigma'}{(\rho' + \sigma') \cdot Min(\rho, \sigma)} \tag{3}$$

If assuming an equivalent data generation and consumption rate and no channel constraints ($\rho$=$\sigma$=$\rho'$=$\sigma'$) for simplicity, $TE_{reconf}$ is equal to 1/2 which stands for half of the value $TE_{stat}$ in the static model. The reason comes from the fact that the producer and the consumer share the same PR region at different time, and thus the communication channel cannot be fully utilized.

The above analysis is for the ideal situation. In practice, context switching overhead ($T_{csp}$ for switching to the producer and $T_{csc}$ for to the consumer) of dynamically reconfiguring each algorithm module should also be included in the total time span. Thus the data delivery throughput and the throughput efficiency in the reconfigurable model can be revised as:

$$\varepsilon_{reconf} = \frac{S}{T_{wr}+T_{rd}+T_{csp}+T_{csc}} = \frac{\rho'\sigma'}{\rho'+\sigma'+\dfrac{(T_{csp}+T_{csc})\cdot\rho'\sigma'}{S}} \quad (4)$$

$$TE_{reconf} = \frac{\varepsilon_{reconf}}{Min(\rho,\sigma)} = \frac{\rho'\sigma'}{(\rho'+\sigma'+\dfrac{(T_{csp}+T_{csc})\cdot\rho'\sigma'}{S})\cdot Min(\rho,\sigma)} \quad (5)$$

## B. Latency

The latency (L) of an IPC packet is the time slice from its generation by the producer till it is received by the consumer. In the static process model, the latency is simply the time needed by the consumer to fetch the packet from the pipe. Assuming that the producer and the consumer are ideally coordinated at the same pace and channel constraints are excluded ($\rho$=$\sigma$=$\rho'$=$\sigma'$), the pipe channel will behave much like a direct connection with a delay unit and the latency can be highly minimized. While in the reconfigurable model, the latency consists of three compositions, as demonstrated in Figure 4. Firstly the packet is injected by the producer into the pipe. It stays in the pipe until the pipe is full. Afterwards the context switching reconfiguration starts and the consumer is to be activated. Finally the IPC packet is received by the consumer only after all other packets before it are read out from the pipe. Equation 6 lists the three latency compositions of the *i*th packet in the reconfigurable design, in which the pipe can hold *n* packets in total:

$$L_i = L_{wr\_i} + T_{csc} + L_{rd\_i} \quad (i \in [0, n-1]) \quad (6)$$

In periodic traffic patterns for both the producer and the consumer, Equation 6 will be elaborated as:

$$\begin{aligned} L_i &= L_{wr\_i}+T_{csc}+L_{rd\_i} = T_{wr}\cdot(1-\frac{i+1}{n})+T_{csc}+T_{rd}\cdot\frac{i+1}{n} \\ &= \frac{S}{\rho'}\cdot(1-\frac{i+1}{n})+T_{csc}+\frac{S}{\sigma'}\cdot\frac{i+1}{n} \quad (i \in [0, n-1]) \end{aligned} \quad (7)$$

With a coordinated data generation and consumption rate and without channel constraints ($\rho$=$\sigma$=$\rho'$=$\sigma'$), Equation 6 can be further simplified as:

$$L_i = T_{wr} + T_{csc} = T_{rd} + T_{csc} = \frac{S}{\rho} + T_{csc} = \frac{S}{\sigma} + T_{csc} \quad (8)$$

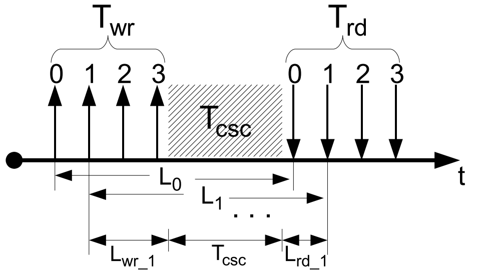


Figure 4. Packet latency in the reconfigurable model

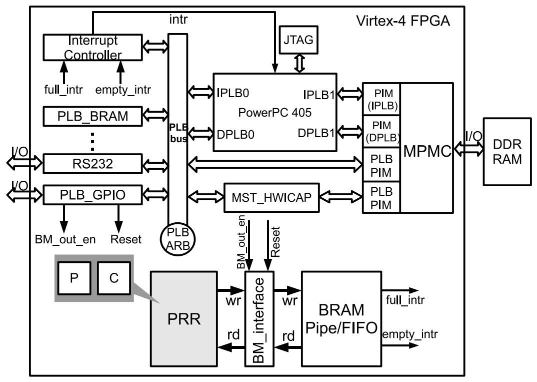


Figure 5. Reconfigurable design using BRAM_pipe

## V. Hardware Implementation

We implement the reconfigurable producer-consumer model on a Xilinx Virtex-4 FX60 FPGA. The pipe is realized by finite-depth FIFO devices using the Block RAM (BRAM) resource on the FPGA or the external DDR SDRAM. On-chip BRAM provides more efficient data transmission bandwidth, while DDR enables larger buffering capability. The reconfigurable implementation using BRAM_pipe is illustrated in Figure 5, in which a complete computer system manages the producer and the consumer configurations in the PR region. Dynamic partial reconfiguration is technically realized by initiating the MST_HWICAP core [14] to load partial bitstreams into the FPGA configuration memory. Switching to the consumer or the producer is respectively triggered by pipe full or empty interrupts, which are detected by the scheduler program running on the PowerPC 405 processor. Between the PR region and the rest static design, a Bus Macro (BM) [15] interface is inserted to lock the signal routing. The interface receives I/O controls to isolate the unpredictable module output signals (with BM_out_en) during active reconfiguration, and to solely reset the newly loaded module (with Reset) after its reconfiguration. More details on the adaptive design framework can be referred to in [16]. When using DDR to implement the pipe (see Figure 6), an interface design (ddr_pipe_interface) is required to realize DDR accesses. The interface provides the bus master functionality to collect produced IPC packets into the DDR memory and direct them to the consumer, via a port of the Multi-Port Memory Controller (MPMC). Both BRAM_pipe and the DDR memory together with its controller feature a data bus width of 64 bits at a system clock of 100 MHz.

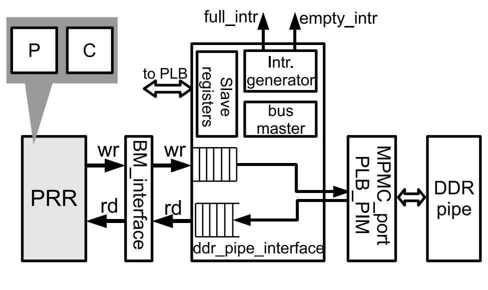


Figure 6. Reconfigurable design using DDR_pipe

|  | BRAM_pipe |  |  |  | DDR_pipe (with ddr_pipe_interface) |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|
|  | 1 KB | 4 KB | 16 KB | 64 KB | 1 KB | 4 KB | 16 KB | 64 KB | 1 MB | 16 MB |
| 4-input LUTs | 63 out of 50560 (0.12%) | 59 (0.12%) | 69 (0.14%) | 200 (0.40%) | 971 (1.92%) |  |  |  |  |  |
| Slice Flip-Flops | 53 out of 50560 (0.10%) | 63 (0.12%) | 73 (0.14%) | 118 (0.23%) | 829 (1.64%) |  |  |  |  |  |
| Block RAM | 2 out of 232 (0.86%) | 2 (0.86%) | 8 (3.45%) | 29 (12.5%) | 2 (0.86%) |  |  |  |  |  |
| DDR SDRAM | / | / | / | / | 1 KB | 4 KB | 16 KB | 64 KB | 1 MB | 16 MB |

Table I
RESOURCE UTILIZATION OF BRAM_PIPE AND DDR_PIPE ON VIRTEX-4 FX60 FPGA

|  |  | Peak data gen./cons. rate ($\rho$, $\sigma$) | Pipe size (S, depth×width) | Partial bitstream size (BS)[1] | Measurement time | Delivered data amount | Data delivery throughput ($\varepsilon_{reconf}$) | Throughput Efficiency ($TE_{reconf}$) |
|---|---|---|---|---|---|---|---|---|
| Test 1 | 1.1 | 800 KB/s | 128×8B=1KB | 27 KB | 1.427s | 500 KB | 350.39 KB/s | 0.438 |
|  | 1.2 | 800 KB/s | 512×8B=4KB | 27 KB | 5.270s | 2 MB | 379.51 KB/s | 0.474 |
|  | 1.3 | 800 KB/s | 2K×8B=16KB | 27 KB | 20.62s | 8 MB | 387.97 KB/s | 0.485 |
|  | 1.4 | 800 KB/s | 8K×8B=64KB | 27 KB | 82.07s | 32 MB | 389.91 KB/s | 0.487 |
| Test 2 | 2.1 | 8 MB/s | 128×8B=1KB | 27 KB | 0.273s | 500 KB | 1.83 MB/s | 0.229 |
|  | 2.2 | 8 MB/s | 512×8B=4KB | 27 KB | 0.656s | 2 MB | 3.05 MB/s | 0.381 |
|  | 2.3 | 8 MB/s | 2K×8B=16KB | 27 KB | 2.194s | 8 MB | 3.65 MB/s | 0.456 |
|  | 2.4 | 8 MB/s | 8K×8B=64KB | 27 KB | 8.339s | 32 MB | 3.84 MB/s | 0.480 |
| Test 3 | 3.1 | 80 MB/s | 128×8B=1KB | 27 KB | 0.159s | 500 KB | 3.14 MB/s | 0.039 |
|  | 3.2 | 80 MB/s | 512×8B=4KB | 27 KB | 0.197s | 2 MB | 10.15 MB/s | 0.127 |
|  | 3.3 | 80 MB/s | 2K×8B=16KB | 27 KB | 0.350s | 8 MB | 22.86 MB/s | 0.286 |
|  | 3.4 | 80 MB/s | 8K×8B=64KB | 27 KB | 0.966s | 32 MB | 33.13 MB/s | 0.414 |
| Test 4 | 4.1 | 800 MB/s | 128×8B=1KB | 27 KB | 0.143s | 500 KB | 3.50 MB/s | 0.0044 |
|  | 4.2 | 800 MB/s | 512×8B=4KB | 27 KB | 0.151s | 2 MB | 13.25 MB/s | 0.0166 |
|  | 4.3 | 800 MB/s | 2K×8B=16KB | 27 KB | 0.165s | 8 MB | 48.48 MB/s | 0.0606 |
|  | 4.4 | 800 MB/s | 8K×8B=64KB | 27 KB | 0.228s | 32 MB | 140.35 MB/s | 0.1754 |
| Test 5 | 5.1 | 8 MB/s | 128×8B=1KB | 80 KB | 0.500s | 500 KB | 1.00 MB/s | 0.125 |
|  | 5.2 | 8 MB/s | 512×8B=4KB | 80 KB | 0.882s | 2 MB | 2.27 MB/s | 0.284 |
|  | 5.3 | 8 MB/s | 2K×8B=16KB | 80 KB | 2.418s | 8 MB | 3.31 MB/s | 0.414 |
|  | 5.4 | 8 MB/s | 8K×8B=64KB | 80 KB | 8.563s | 32 MB | 3.74 MB/s | 0.468 |

Table II
MEASUREMENT RESULTS OF THE PIPE PERFORMANCE ON THE RECONFIGURABLE IMPLEMENTATION

Table I lists the resource utilization of BRAM_pipe and DDR_pipe implementations. We see from the numbers that BRAM_pipe consumes the expensive BRAM resource on the FPGA to construct the FIFO. BRAM provides more efficient data delivery bandwidth, but only small pipe sizes. LUT and Flip-Flop utilizations in BRAM_pipe are negligible. DDR_pipe requires a fixed overhead of the interface design (ddr_pipe_interface). Its resource utilization is small (all below 2%) and acceptable in practical reconfigurable designs. The pipe sizes are flexibly adjustable in a wide range in the DDR memory.

## VI. EXPERIMENTAL RESULTS

### A. Throughput and TE Results

To verify the formulas derived in Section IV, we did experiments to measure the performance and explicitly demonstrate the characteristics of pipes. For analysis simplicity, we select periodic traffic patterns and matching data generation and consumption capabilities ($\rho = \sigma$), with which TE is both analyzed and measured as 1 in the static producer-consumer design. On the reconfigurable implementation using BRAM_pipe, we measure the performance at four rates, specifically 800 KB/s, 8 MB/s, 80 MB/s and 800 MB/s. Results are listed in Table II as the function of data generation/consumption rates ($\rho, \sigma$) and pipe sizes (S). With the same partial bitstream size of 27 KB, Test 1 - 4 correspond to slow, medium, fast, and extra-fast IPC data rates. In each test, four levels of pipe sizes are varied for measurements. We record the delivered data amount as well as the time span, and calculate the effective data delivery throughput ($\varepsilon$) from their division. According to the definition formula in Equation 1, we also derive the TE values of using BRAM_pipe in reconfigurable computing, and list the results in the last column of the table. To study the effect of different partial bitstream sizes (BS), which result in different context switching overhead $T_{csp}$ and $T_{csc}$, we did also measurements with a larger bitstream size of 80 KB in Test 5 at a modest data rate of 8 MB/s (Normally IPCs do not feature too high rates, as incoming data streams do.). We draw TE curves in Figure 7 for all the tests, and observe that the measured throughput efficiency of the pipe may approach to the theoretical value of 0.5 at the end of large pipe size. In addition, a lower data rate leads to more efficient bandwidth utilization of the pipe channel. Both reasons come from the approach to reduce the context switching frequency and decrease the proportion of context switching overhead in the overall measurement time.

In FPGA-based adaptive computing, context switching overhead consists of Partial Reconfiguration (PR) time for switching IP cores and software overhead, such as the time needed to stop the BM outputs and to reset the newly loaded module [16]. To investigate context switching overhead in depth, we compare Test 2 and Test 5 with the same data rate, and profile the composition of the overall measurement time. From the results shown in Figure 8(a) for Test 2.4 and 8(b) for Test 5.4, we observe that with a modest data rate of 8 MB/s and a large pipe size of 64 KB, effective pipe write and read operations dominate the proportion of the overall measurement time (98.24% and 95.67% respectively). Therefore the measured throughput efficiency is very close to

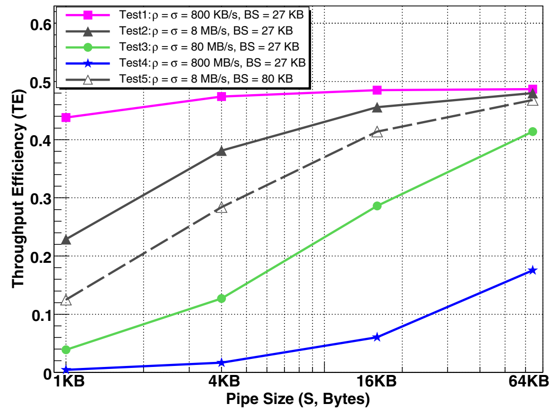
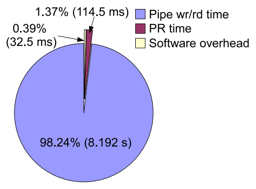
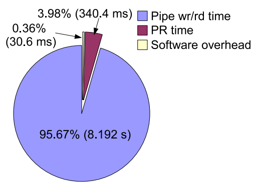


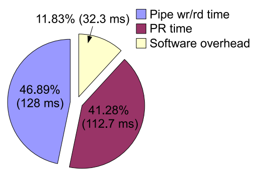
Figure 7. TE measurements on BRAM_pipe



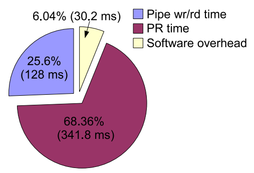
(a) Test 2.4



(b) Test 5.4



(c) Test 2.1



(d) Test 5.1

Figure 8. Composition of the measurement time

the theoretical value of 0.5. The context switching overhead is very small in this case. Moreover from these two subfigures, we also see the PR time as well as its proportion in Test 5.4 (340.4 ms, 3.98%) larger than in Test 2.4 (114.5 ms, 1.37%), due to the larger partial bitstream size. The software overhead is very similar (∼30 ms) in both tests. By contrast, we observe much larger overhead proportions of 53.11% and 74.4% respectively in Test 2.1 and 5.1 with a small pipe size of 1KB, as shown in Figure 8(c) and 8(d). Because of the small buffering capability, frequent context switching increases the overhead proportion and less efficiently utilizes the pipe to transport IPC data. Test 5.1 is even worse than 2.1, due to its larger partial bitstream size.

Throughput and TE are also measured on DDR_pipe. Results are illustrated in Figure 9. We see that DDR_pipe can improve the data delivery throughput and TE by further increasing the pipe size to 1 MB and 16 MB. With the same pipe sizes from 1 KB to 64 KB, DDR_pipe achieves similar results as BRAM_pipe, with the only exception in the test for extra-fast data rate of 800 MB/s. This is due to the situation that the memory controller bandwidth is saturated by the too high data rate and generates bottleneck to DDR_pipe accesses. By contrast, BRAM_pipe provides sufficient bandwidth and achieves higher TE than DDR_pipe at the extra-fast IPC data rate.

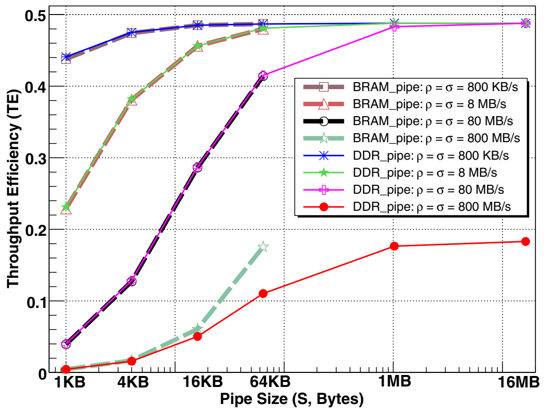


Figure 9. TE comparison of BRAM_pipe and DDR_pipe

## B. Latency Results

Due to the time multiplexing feature of computing resources in reconfigurable designs, IPC packets cannot be directly transmitted to the consumer and they have to be contained in the pipe until context switching to the consumer process. The latency evaluation has been discussed in Section IV-B, and we did also practical measurements on the FPGA design. In the experimental setup for BRAM_pipe and DDR_pipe, the producer transmits time tags as IPC packets into the pipe, and the consumer receives them and calculates latency from the time differences with a global timer. Large amounts of IPC packets were sampled, and the arithmetic average results are listed and illustrated in Table III and Figure 10 for pipes with different sizes. We observe that the latency of IPC packets deteriorates along with the increment of the pipe size (approximately linearly), as well as the decrement of the data rate. With the same pipe sizes, DDR_pipe results in slightly higher latency than BRAM_pipe, due to the complex DDR access mechanism: IPC packets need more clock cycles to be stored into or read out of the DDR memory. The latency of DDR_pipe obviously deviates from BRAM_pipe only at the extra-high data rate of 800 MB/s, due to the saturated memory bandwidth.

There exists a pair of contradictions between throughput and latency in the reconfigurable design: To shrink context switching overhead and increase throughput efficiency, the pipe size is expected to be maximized. However a large pipe size leads also to a large latency meanwhile. Therefore the pipe size should be cautiously determined according to concrete system requirements such as communication throughput or realtime requirements.

## C. Result Matching with Formulas

All practical measurement results on throughput, TE, and latency are verified to match very well the theoretical analysis using derived formulas in Section IV. As an instance

[1]Due to the implementation differences, the sizes of partial bitstreams for different designs may slightly fluctuate close to the listed values in the table.

| Peak data rate / Pipe size | BRAM_pipe |  |  |  | DDR_pipe |  |  |  |
|---|---|---|---|---|---|---|---|---|
|  | 800 KB/s | 8 MB/s | 80 MB/s | 800 MB/s | 800 KB/s | 8 MB/s | 80 MB/s | 800 MB/s |
| 1 KB | 1.431 ms | 0.266 ms | 0.150 ms | 0.138 ms | 1.483 ms | 0.274 ms | 0.153 ms | 0.143 ms |
| 4 KB | 5.269 ms | 0.654 ms | 0.188 ms | 0.142 ms | 5.323 ms | 0.658 ms | 0.192 ms | 0.154 ms |
| 16 KB | 20.631 ms | 2.185 ms | 0.343 ms | 0.156 ms | 20.683 ms | 2.195 ms | 0.345 ms | 0.195 ms |
| 64 KB | 82.070 ms | 8.328 ms | 0.954 ms | 0.217 ms | 82.123 ms | 8.338 ms | 0.960 ms | 0.358 ms |
| 1 MB | / | / | / | / | 1.311 s | 131.219 ms | 13.248 ms | 3.627 ms |
| 16 MB | / | / | / | / | 20.972 s | 2.097 s | 209.856 ms | 55.930 ms |

Table III
LATENCY MEASUREMENTS ON BRAM_PIPE AND DDR_PIPE

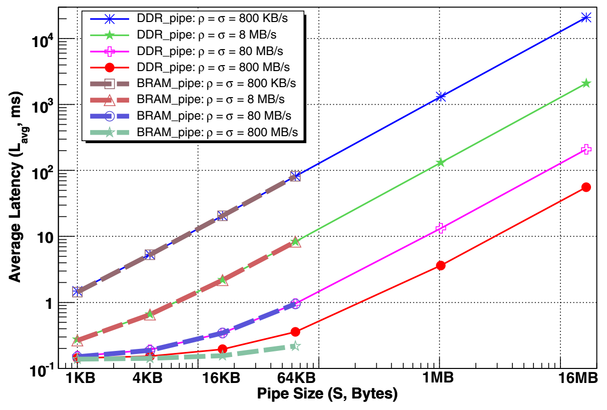


Figure 10. Latency measurements on BRAM_pipe and DDR_pipe

in Test 2.1 (S=1 KB), the producer and the consumer are coordinated at a same data rate of 8 MB/s. The dedicated BRAM_pipe write and read interfaces provide sufficient bandwidth. Thus both the producer and the consumer can effectively work at the peak data rate ($\rho'$=$\sigma'$=$\rho$=$\sigma$). We measured the time overhead switching to the producer ($T_{csp}$) and to the consumer ($T_{csc}$) respectively as 150.37 and 135.50 $\mu$s. Thus according to Equation 4 and 5, we theoretically evaluate $\varepsilon_{reconf}$ and $TE_{reconf}$ as 1.86 MB/s and 0.233 respectively. Both values match very well the measured results of 1.83 MB/s and 0.229 listed in Table II. With Equation 8 the latency is calculated as 0.261 ms, also close to the 0.266 ms in Table III. Other tests share the same analysis procedure except the ones for DDR_pipe at the extra-high data rate of 800 MB/s. In that case, the memory controller bandwidth is saturated and hence the effective data generation or consumption rate has to be lowered below the peak rate ($\rho'$<($\rho$=$\sigma$), $\sigma'$<($\rho$=$\sigma$)). Accordingly a pair of smaller $\rho'$ and $\sigma'$ results in less $\varepsilon_{reconf}$, $TE_{reconf}$ and larger $L_i$.

## VII. CONCLUSION AND FUTURE WORK

In this paper, we investigate an inter-process communication mechanism using pipes for FPGA-based adaptive computing. Dynamically reconfigurable hardware processes multiplex the same computing resources at different time, and pass their IPC information through pipes. The pipe performance is theoretically analyzed in switching contexts, in terms of throughput, throughput efficiency and latency. We also present two practical pipe implementations using FPGA Block RAM or external DDR memory. Experimental results on the performance measurements disclose the basic characteristics of this communication mechanism, which can act as important reference criteria when pipes are practically parameterized in adaptive designs.

In the future work, the latency of IPC packets is expected to be reduced with more intelligent mechanisms, and to be properly coordinated with throughput requirements in realtime applications.

## ACKNOWLEDGMENT

This work was supported in part by BMBF under contract Nos. 06GI9107I and 06GI9108I, FZ-Juelich under contract No. COSY-099 41821475, HIC for FAIR, and WTZ: CHN 06/20.

## REFERENCES

[1] D. P. Bovet and M. Cesati, “Understanding the Linux Kernel, 3rd Edition”, O’REILLY & Associates, Inc., ISBN: 0-596-00565-2, Nov. 2005.

[2] P. Master, “Adaptive computing makes efficient use of silicon”, *EE Times*, Feb. 2002.

[3] H. K. So, A. Tkachenko, and R. Brodersen, “A Unified Hardware/Software Runtime Environment for FPGA-based Reconfigurable Computers using BORPH”, *In Proc. of the International Conference on Hardware/Software Codesign and System Synthesis*, Oct. 2006.

[4] H. Kalte and M. Porrmann, “Context Saving and Restoring for Multitasking in Reconfigurable Systems”, *In Proc. of the International Conference on Field Programmable Logic and Applications*, Aug. 2005.

[5] C. Huang and P. Hsiung, “Software-controlled Dynamically Swappable Hardware Design in Partially Reconfigurable Systems”, *EURASIP Journal on Embedded Systems*, Jan. 2008.

[6] X. Zhou, Y. Wang, X. Huang, and C. Peng, “Fast On-line Task Placement and Scheduling on Reconfigurable Devices”, *In Proc. of the International Conference on Field Programmable Logic and Applications*, Aug. 2007.

[7] H. Walder and M. Platzner, “Online Scheduling for Block-partitioned Reconfigurable Devices”, *In Proc. of the Design Automation and Test in Europe Conference and Exhibition*, Dec. 2003.

[8] H. Walder and M. Platzner, “Non-preemptive Multitasking on FPGAs: Task Placement and Footprint Transform”, *In Proc. of the 2nd International Conference on Engineering of Reconfigurable systems and Architectures*, Jun. 2002.

[9] C. Steiger, H. Walder, and M. Platzner, “Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-Time Tasks”, *IEEE Transactions on Computers*, Nov. 2004.

[10] J. Zhu, I. Sander, and A. Jantsch, “Performance Analysis of Reconfiguration in Adaptive Real-time Streaming Applications”, *In Proc. of the Workshop on Embedded Systems for Real-time Multimedia*, Oct. 2008.

[11] I. Sander, et al., “High-Level Estimation and Trade-Off Analysis for Adaptive Real-Time Systems”, *In Proc. of the Reconfigurable Architectures Workshop*, May. 2009.

[12] C. Bobda, M. Majer, A. Ahmadinia, T. Haller, A. Linarth and J. Teich, “The Erlangen Slot Machine: Increasing Flexibility in FPGA-based Reconfigurable Platforms”, *In Proc. of the IEEE International Conference on Field-Programmable Technology*, Dec. 2005.

[13] S. Fekete, J. van der Veen, M. Majer, J. Teich, “Minimizing Communication Cost for Reconfigurable Slot Modules”, *In Proc. of the International Conference on Field Programmable Logic and Applications*, Aug. 2006.

[14] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, “Run-time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration”, *In Proc. of the International Conference on Field Programmable Logic and Applications*, Aug. 2009.

[15] M. Huebner, T. Becker, and J. Becker, “Real-time LUT-Based Network Topologies for Dynamic and Partial FPGA Self-Reconfiguration”, *In Proc. of the Symposium on Integrated Circuits and System Design*, Apr. 2004.

[16] M. Liu, Z. Lu, W. Kuehn, S. Yang, and A. Jantsch, “A Reconfigurable Design Framework for FPGA Adaptive Computing”, *In Proc. of the International Conference on ReConFigurable Computing and FPGAs*, Dec. 2009.