Low-Cost Guaranteed-Throughput Dual-Ring Communication Infrastructure for Heterogeneous MPSoCs

Berend H.J. Dekens* Philip S. Wilmanns* Gerard J.M. Smit* Marco J.G. Bekooij*‡

* University of Twente
Department of EEMCS
Enschede, The Netherlands

‡ NXP Semiconductors
Eindhoven, The Netherlands

***Abstract*—Connection-oriented Guaranteed-Throughput (GT) mesh-based Networks on Chip (NoCs) have been proposed as a replacement for buses in real-time stream processing systems but are currently rarely used as hardware cost tends to be higher than conventional interconnects. Recently an interconnect with a ring topology was introduced as a low-cost alternative for use in medium scale homogeneous Multiple Processor System on Chip (MPSoC) designs. Cost-effective integration of stream processing accelerators would require an extension of this ring interconnect.**

We present a dual-ring communication infrastructure for heterogeneous MPSoC designs. Data and credits are transferred between tiles using their separate, oppositely directed, rings. The minimum throughput is determined by analysis of a Cyclo-Static Data Flow (CSDF) model for a system with communication between accelerators and processors.

The performance benefits and costs are evaluated by integration of our dual ring and an accelerator in a 16 core MPSoC which is mapped on a Virtex6 FPGA. On this MPSoC a real-time PAL video decoder is executed. A performance gain of a factor 3.6 was obtained at an increase in hardware cost of only 8.5%.

I. Introduction

Programming of homogeneous Multiple Processor Systems on Chip (MPSoCs) is usually much easier than the programming of heterogeneous MPSoCs. However, the use of general purpose processors for Digital Signal Processing (DSP) applications is inherently less efficient compared to the use of dedicated hardware. As such, the use of stream processing accelerators which contain dedicated logic can improve throughput. However, an interconnect which supports stream processing accelerators should not be so complex that it negates the energy and performance advantage gained from using accelerators. At the same time, such an interconnect should provide real-time guarantees as required by many stream processing applications, for example in the Software Defined Radio (SDR) domain.

Recently a connectionless ring interconnect was introduced as a low cost alternative for buses in medium scale homogeneous MPSoCs for real-time stream processing applications [1]. The concept of guaranteed acceptance removes the need for flow-control inside the write-only network. Instead, flow control is provided at the application level by a distributed communication protocol for FIFO channels.

In order to improve the efficiency and performance of the system from [1], we have extended this system with stream processing accelerators. Integrating stream processing accelerators in a memory mapped MPSoC introduces new issues as these accelerators usually work on a stream of data and as such have no notion of addresses. Communication with these accelerators requires an interconnect which supports hardware flow control.

In this paper we introduce a dual-ring communication infrastructure for heterogeneous MPSoCs with accelerators intended for stream processing applications. Hardware flow control is credit-based where the number of credits denote the depth of the buffer between a producer and a consumer. A producer consumes credits upon producing data and a consumer sends the producer a credit whenever it consumed data. As flow control is implemented in hardware by means of a small shell in each Network Interface (NI), the number of simultaneous FIFO channels for this implementation is limited. As such, all communication involving stream processing accelerators is handled in hardware, including streaming data from processor to accelerator or vice versa, while processor-to-processor communication is handled in software by the C-HEAP algorithm [2] which was modeled previously as an Synchronous Data Flow (SDF) graph [1]. We present a Cyclo-Static Data Flow (CSDF) [3] model for hardware based flow control in order to derive application level throughput guarantees. This model is valid for processor to accelerator communication when said processor only executes a single task. We show by means of a real-time Phase Alternating Line (PAL) video decoder application that adding the second ring and accelerator results in a significant improvement in throughput but results only in a relatively small increase of the hardware cost.

The outline of this paper is follows. In the next section we will introduce related work, Section III introduces the dual ring interconnect and our MPSoC system. Section IV presents the data flow model of the hardware flow control and in Section V hardware costs and performance is evaluated by means of a real-time PAL video decoder application. The conclusions are presented in Section VI.

This work is supported in part by the SenSafety project in the Dutch COMMIT program (commit-nl.nl) and the VAUDEO project from the second tender for the Dutch "Innovatie Maatschappelijke Veiligheid".

II. Related Work

Work in the field of integrating stream processing accelerators can be loosely grouped into three categories: interconnects supporting point-to-point streaming, wrappers for accelerators providing stream support and accelerators as co-processors.

Before considering interconnects, we first distinguish two major classes: *connection oriented* and *connectionless* interconnects. Connection oriented networks have separate connections between masters and slaves where properties can be specified for each individual connection [4]. In a connectionless network, streams are not separate and as such can influence each other which makes it hard to provide real-time guarantees [5]. We show that throughput guarantees can be given by making use of a CSDF model despite that our network is connectionless.

Ideally, data streams are transferred on their own dedicated FIFO channel such as provided by the AXI4 Stream Interconnect [6] and Avalon Streaming Bus [7]. However, when communication patterns between components are not known at design time, all-to-all connectivity is required which tends to result in high hardware costs.

Alternatively, circuit switched interconnects like a single layer bus, cross-bar or interconnects supporting virtual channels [8], [9], [10], [11] could be used to integrate stream processing accelerators in an MPSoC design as long as the setup of the point-to-point channels is handled elsewhere. For real-time applications support for Guaranteed-Throughput (GT) traffic is mandatory. Usually supporting all-to-all connectivity in such Networks on Chip (NoCs) results in high hardware costs where this is supported by default in our low cost dual ring interconnect.

Contention in such NoCs can prevent channels from being set up when a critical router is in use. The automatic serialization of the ring topology coupled with our arbitration policy prevents contention and provides GT support. Streaming data support is realized by a shell in the NI of the dual ring interconnect.

Rings have been used in commercial systems in recent years such as the Cell processor from IBM [12], the Intel Nehalem processor architecture [13] and more recently in the Maxeler data-flow computers [14]. The difference between our ring and existing implementations are the throughput and latency guarantees that our ring interconnect provides.

In [15], a ring network is introduced which proposes Rotating Time Division Multiplex Access (RTDMA) as a possible scheduling algorithm to divide bandwidth fairly between peripherals, which is similar to our arbitration policy. They support hardware flow-control but provide no back-pressure mechanism. Additionally, they do not support point-to-point data streams and are not work conserving.

Instead of using an interconnect with support for data streams, a wrapper can be placed between a stream processing accelerator and an interconnect in order to provide stream support. The C-HEAP [2] FIFO protocol can be implemented in such a wrapper. However, these wrappers have a significant hardware cost as their functionality is comparable to that of a Direct Memory Access (DMA) controller. Our hardware flow control uses static addresses for data and credit updates to prevent the need for dynamic address generation. By using C-HEAP only for software FIFO communication, we have maximum flexibility without high hardware costs.

A method to embed accelerators in a homogeneous memory mapped system is presented for the STHORM architecture [16]. Both accelerators and their wrappers have a notion of memory addresses and as such require logic for dynamic memory address generation. The use of static addressing in our shells for each accelerator results in low hardware costs.

An example of a solution which integrates fine-grained accelerators in the data path of a programmable processor is the Imagine stream processor [17]. This processor employs a crossbar for the communication between accelerators and the Arithmetic Logic Units (ALUs). The accelerators and ALUs execute in a Single Instruction Multiple Data (SIMD) fashion under the control of a program counter. In contrast, the accelerators and processors in our design are data driven which introduces additional scheduling freedom compared to the tight centralized control from Imagine.

The concept of "conservation cores" [18] is similar to co-processors. In this work parts of a software algorithm are synthesized into dedicated hardware blocks which are called conservation cores. The software offloads work to these co-processors wherever possible in order to reduce energy consumption. As the master processor has to wait for the co-processor to complete before it can advance the program thread, this system is instruction driven. Because our design is data driven, once again we have the freedom to perform processing data in pipelined fashion whenever possible.

III. Proposed Architecture

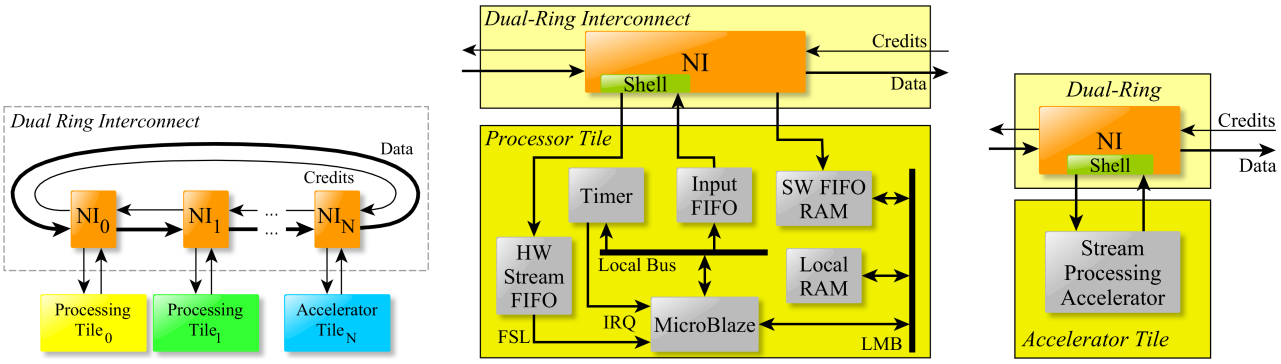
We will now describe the MPSoC system which contains the dual ring interconnect. This system was designed for heterogeneous processing applications where various processors and accelerators are employed to improve energy efficiency over homogeneous designs. Our architecture consists of various interconnected "tiles" which contain processing elements as is shown in Figure 1a.

A. Processor Tile

Figure 1b shows an overview of a processor tile. In our current prototype we have multiple processor tiles connected to the ring. Each tile contains a RISC MicroBlaze processor, timer for interrupts and local memory for instructions and data. Instruction and data caches are present to improve performance of larger programs. Each processor runs at the system clock speed of 100 MHz and executes a small real-time POSIX compliant kernel with multi-threading support. The local software FIFO memory is dual ported and is connected to the output port of the NI in order to receive data writes from remote tiles. These memories are connected to a Local Memory Bus (LMB) and are capable of single cycle reads or writes and as such also adhere to the guaranteed acceptance required for the ring.

The hardware stream FIFO in Figure 1b is used to receive a data stream from an accelerator tile, which is described in Section III-B. This FIFO is connected directly to the MicroBlaze processor by means of a Fast Simplex Link (FSL) [19] connection allowing for single cycle access to the data stream.

All data entering the ring passes through the shell in the NI. Data sent to a processor tile will use software flow control



(a) Abstract MPSoC architecture (b) Processor tile (c) Accelerator tile

Fig. 1. Overview of the MPSoC architecture and two types of tiles

while data for stream processing accelerators will use hardware flow control.

Each processing tile executes one or more tasks on a real-time POSIX compliant kernel. Usually accelerators are configured from a single processing tile when the system starts. Programs are compiled using the GNU Compiler Collection, allowing for easy development and programming.

B. Accelerator Tile

In Figure 1c an accelerator tile is shown. The interface between the NI and the accelerator is similar to the interface of a simple FIFO channel. At the input of the accelerator, a signal is set when the shell holds data from the interconnect. When the accelerator is ready to process this data, it reads the data from the NI shell output and sends a signal that it completed the read. At the output of the accelerator, the inverse is used. Flow control is implemented in hardware in the NI shell and as such this simple handshake interface to load and eject data from an accelerator is sufficient for any stream processing accelerator.

The accelerator has no notion of the underlying interconnect or memory addresses which are used to transfer packets between tiles. Accelerator configuration can be programmed over the ring by any processor, for example the signal gain on a level conversion accelerator.

In our system, stream processing accelerators are currently chained together based on a description written by a programmer which describes the data flows between tiles. A support library abstracts the exact interfaces used to set up and configure data streams, simplifying the use of accelerators. Processor tiles read from an accelerator by means of a library function which pulls data from the FSL FIFO and write data to accelerators using a similar function. Accelerators have no notion of which tile they are communicating with.

C. Data Ring

Our dual ring interconnect combines two uni-directional rings to support software and hardware based flow control. By only applying hardware flow control for data streams involving accelerators, we keep hardware costs low while the use of software flow control between software tasks maximizes flexibility in communication patterns. We will now describe the uni-directional ring [1] in more detail.

The packet switched uni-directional interconnect for all-to-all communication has low hardware cost. Our slot allocation algorithm for the ring, as briefly explained in Section III-D, provides GT traffic and makes the ring work conserving. All-to-all communication is supported without using further internal buffers, aside from registers between NIs. This results in low hardware cost and does not reduce predictability.

A ring interconnect consists of a number of NIs that are chained together from one to another, forming a ring. When a packet is received from a neighboring NI, it is either ejected when the current NI is its destination, or passed on to the next NI if it is not. Packets that are to be injected, enter the ring when it is available: which means it is injected when the current NI has no packet to pass along and the arbitration policy allows injection. Because packets have a single route between two tiles, it is guaranteed that data writes arrive in the same order as they were issued. As routing is trivial, we consider the "router" to be part of the NI.

To keep hardware costs low, our ring relies on the concept of guaranteed acceptance: connected tiles are require to accept and register any packet in a single cycle at any time, which is similar to many multi-layer buses. As such, data is ejected from the ring whenever it reaches its destination. This means that there is no need for back-pressure or packet deflection to manage packet flow within the ring. This also allows us to determine the transit latency precisely since we know for every packet how many hops it will need to reach its destination.

The nature of a ring provides serialization for all NIs, preventing the need for memory port arbitration. As input and output buffers can be shared between streams in our NIs, there is no limit to the number of simultaneous streams that can be multiplexed over the same NI.

We show that there is no need to support remote reads which keeps network complexity low. As such, tiles can only write to remote memories, which is no issue for our software and hardware FIFO channel implementations.

Figure 2 shows a schematic overview of a single NI. At the local input port of the network interface there is a small FIFO which has a capacity of one word for accelerators and is configurable for processor tiles. For processor tiles, an address is provided when writing data, for accelerators, only data is written. This FIFO holds tuples consisting of a network address and a data word. The depth γ of the FIFO is configurable

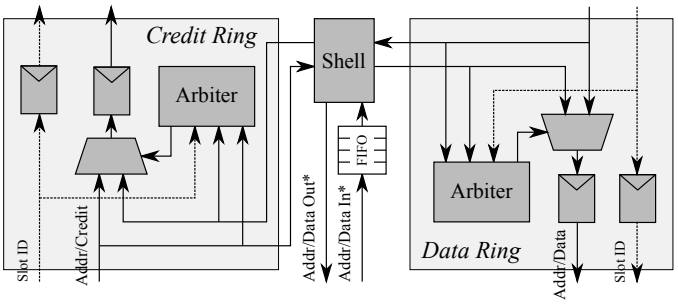


Fig. 2. Schematic overview of a single NI. *Accelerators only read and write data and omit addresses.

at design time. To prevent confusion we will refer to this hardware FIFO simply as "buffer" for the remainder of this paper. Whenever this buffer is full, any further writes from the local tile to the network will stall this tile.

D. Slot Allocation

Every NI passes an address and data to its neighbor every cycle, both of which may be empty. We consider these pairs unique and call them *slots* [1]. By giving every slot a unique and incrementing number as identifier, we can distinguish between every slot. By numbering NIs in the same manner, we can define the concept of an *owned* slot: when a slot arrives at an NI which has the same ID as the slot, this slot is owned by that NI. We define now the first rule of our arbitration policy:

Rule 1: *Every NI can always claim its own slot without having to check if it contains data: it can not be in use by another NI.*

We observe that if both slots and NIs are numbered in the same fashion, then every slot will reach its owner at the same time as the slots align with their initial NI. This happens once every N cycles, where N is the number of NIs forming the ring. From this we can conclude that available bandwidth is distributed fairly between all NIs and as such is $\frac{1}{N}$.

We now extend this policy to make the ring interconnect work conserving by using unclaimed slots.

Rule 2: *When a slot is empty, it can be used by any NI provided that the slot will not reach its owner before the data is delivered.*

This rule ensures that unclaimed bandwidth can be used while its guaranteed that any NI can always use its own slot.

As a result of this slot allocation policy, we can derive bounds for throughput and latency for single packets traversing one of the rings in our interconnect. These bounds will be used when modeling a FIFO channel between two tasks using either the software or hardware FIFO implementations.

E. Software FIFO

Our ring network is used with a distributed FIFO implementation [1] based on C-HEAP [2] using only writes to remote memory. This software FIFO provides support for back-pressure at the application level. This is used for the communication between parallel software tasks in stream processing applications that can be described by task graphs.

C-HEAP uses data containers which are an arbitrary number of words in size and works by means of distributing read and write pointers to both producer and consumer. These pointers are incremented in a round-robin fashion on the containers of the FIFO instance and are compared to determine how much data is buffered in a FIFO channel. The C-HEAP algorithm does not require locks, mutexes or atomic read-modify-write support. The only requirement is that reads and writes are in-order; they should arrive in memory in the same order as they were issued.

F. Credit Ring

We extend the interconnect consisting of a single, uni-directional data ring from [1] with a second ring for flow control for stream processing accelerators. To keep hardware costs low, it is undesirable to have large buffers at the inputs of accelerators to hold large tokens when processing can be fine grained. This means containers for the accelerator input FIFO could be as small as a single data word and buffers have to hold at least one such container. To prevent a large communication overhead on the data ring, we will use this second ring to send credits between consumers and producers.

By using a credit-based handshake [20], we implement flow control for accelerators. In this flow control scheme, we have a producer-consumer pair with a number of credits. The number of credits available at the producer determines how many data containers can be buffered at the consumer. When the producer wants to send data, it is required to have at least a single credit available. When no credits are available anymore, the producer has to wait for the consumer to free up its buffer. When a consumer consumed a data container, it sends a credit back to the producer to signal that it freed up space for a single data container.

In this case, the number of credits indicate the number of data containers that can be buffered at the input of an accelerator. As long as a producing accelerator has credits and input data, it can send data to another accelerator.

As credits are not unique, we only need to transfer the fact that a credit is present; this can be done in a single bit instead of 32 bits for a data word. As such, the credit ring requires less hardware than the data ring.

The credit ring has the same limitations as the data ring: high throughput and low latency is only possible when processing is kept local, i.e., data is sent to a neighbor which is only a few hops away. To keep latency low and prevent the needless intersection of credit streams, the credit ring transfers its payload in the opposite direction of the data ring. This means that if data is transferred H hops, a credit will traverse H hops back from the data consumer to the data producer.

G. Dual Ring Interconnect

Credit based flow control is implemented in the NI in a small shell which makes use of both rings, see Figure 2. This shell passes traffic for the software FIFO to the data ring. Data traffic to or from accelerators is subject to hardware flow control.

Each shell contains a single credit return address per input and single data forwarding address per output. As data arrives at the accelerator tile, the shell will pass it to the inputs of the accelerator. When the accelerator accepts the data at one of its inputs, a credit will be sent to the return address. Every input for an accelerator is buffered by the shell where at least a

single input token is buffered. Data produced by an accelerator is passed to the shell which will forward data to the data forwarding address as soon as credits are available. Note that these addresses are static and as such require no counter logic as is common for dynamic addressing. In this fashion, data is streamed from one accelerator to the next.

While flow control supports the use of multiple credits per stream, input buffers with a matching capacity at the accelerators are required. In the current implementation we use a fixed depth of a single credit per stream. This limits the maximum throughput between two tiles using hardware flow control but this is of little consequence for our current demonstrator as it requires a lower throughput.

IV. Data-Flow Model

The dual ring interconnect is used for software FIFO communication between processor tiles and hardware FIFO communication for all other combinations between tiles. The software FIFO communication has previously been modeled as an SDF graph [1].

An SDF model is a directed graph $G = (E, V)$ with *actors* $v \in V$ and directed *edges* $e \in E$. Each edge $e_{i,j}$ describes a unbounded queue for atomic data objects called *tokens* between actors v_i and v_j. The head and tail of each edge is annotated by *quanta* which denote the number of tokens an actor will consume or produce. Every actor has a *firing duration* ρ_v. An actor can *fire* when its incoming edges have at least the number of tokens as denoted by their quanta. To prevent concurrent firing where this is not desired, self edges are used with a single token to prevent overlapping firings of actors.

CSDF extends SDF by introducing the concept of *phases*. Each actor has a cyclic behavior during which its phases are fired. The firing duration for every phase p is denoted as $\rho_v(p)$. Both firing durations and quanta are expressed as a list of values with an equal length.

In this paper we define that actors in a CSDF graph do not have an implicit self-edge and as such phases could fire in concurrently. To prevent token reordering all phases of an actor that does not have a self-edge with one token must have the same firing duration. If parallel execution of phases of the same actor is not desired, a self-edge is added with a single token to the CSDF actor.

In order to keep hardware costs low, inputs of an accelerator tile should have a very small FIFO capacity as we assume that most stream processing accelerators can work on individual data words. Processing tiles on the other hand, transfer larger chunks of data to minimize overhead resulting from the acquire and release calls on containers in the software FIFO administration.

If we would attempt to capture the exchange of data containers between a processor tile and an accelerator tile in a simple SDF model containing a *Producer-Consumer* pair of actors, we see that this model is not accurate because it models that all S words are produced at once when the task v_P finishes its execution instead of word by word during the execution of the task. The result is shown in Figure 3 and contains actors v_P and v_{Acc} where v_P produces data in FIFO containers of size S words and v_{Acc} consumes data one word at a time. The depth of the FIFO between both actors is expressed in data words and denoted by α. When the hardware FIFOs

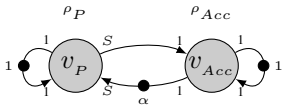


Fig. 3. Potentially dead-locked model for accelerator communication

should be kept small for cost reasons, it is desirable to make α smaller than S. However in this case the model will deadlock.

We will now introduce a new model for temporal analysis of the hardware FIFO communication. To this end we derive an CSDF model which can be used to determine whether the requirements are satisfied for a specific real-time stream processing application. We need CSDF as we just showed that SDF is not sufficiently expressive to accurately model the communication between a processor and an accelerator.

In our CSDF model, the incremental transference of data v_P to v_{Acc} is modeled by the explicit release of individual data tokens where every token denotes a data word. In Figure 4, actor v_P produces S times a tokens (notation $\langle S \times 1 \rangle$), each of which is released explicitly after each phase. Note that the sum of all firing durations for each phase of v_P from Figure 4 is identical to the firing duration of v_P from Figure 3 and for simplicity we can assign the last phase the summed duration. At the start of the firing of v_{Acc}, one token will be consumed and at the completion one credit token is emitted.

Note that if $\alpha = 1$, parallel execution is impossible as v_P and v_{Acc} will execute one after another. As such, we assume that $\alpha \geq 2$ where increasing FIFO capacity improves pipelining and throughput. We will show that a minimum FIFO buffer capacity can be computed given a minimum throughput requirement.

In our system a processor will stall if a write can not complete. During a stall the processor does not execute instructions. A write can not complete if the buffer in the network interface is full. How often this buffer is full during a write depends on the actual moments in time that v_{Acc} consumes its data. Because our dataflow analysis only determines the latest possible consumption moments, the actual consumption moments are not known at design time and therefore the total time that the processor will stall is unknown. As a consequence it is not possible to determine at design time how many cycles are left for other tasks executing on the same processor. Therefore, for predictability reasons we do not allow sharing of a processor by multiple tasks if one of these tasks communicates with an accelerator.

At the completion of every phase of v_P a write is issued which may stall the processor. However, as writes only occur at the end of a phase and not during its firing, the processor will never stall during the firing of a phase. As such the firing durations of all phases of v_P are not affected by the stalling of the processor.

When we consider a FIFO channel between accelerators, a producing accelerator would write output word by word to the consuming accelerator which will send a credit for every

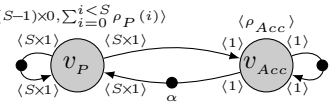


Fig. 4. CSDF model of hardware FIFO communication

accepted word. As such, the model from Figure 4 can also be applied to accelerator to accelerator communication when $S = 1$.

One type of data exchange between tiles that not has been covered so far, is streaming data from an accelerator to a processor tile. As shown in Section III-A, data arrives at a processor tile in a hardware FIFO which is connected to the shell from the NI like an accelerator. Data delivered to the hardware FIFO is directly available to a task running on the processor. As such, from a modeling perspective, this case is no different than inter-accelerator communication.

A. Channel Description

The model from Figure 4 only describes the communication between a producer v_P and consumer v_{Acc} where the interaction at the end points are described. As such the interconnect that actually transports the data is assumed to be ideal: it introduces no latency and has unlimited bandwidth. In an actual system, data is transferred over an interconnect which introduces latency and imposes a bandwidth limit. These properties can be described by a Latency-Rate Server model [21] which is shown in Figure 5.

By adding two actors in both the data and credit path, we can express latency and rate for both channels. The latency actor v_L from Figure 5 introduces a delay expressing the worst case latency experienced by any packet that traverses the interconnect to an arbitrary tile. This includes all wait times like access delays and queuing effects and thus requires a deterministic interconnect which can provide such guarantees. Because there is no queuing in the network itself, the latency for a packet going from v_P to v_C in Figure 5 is $\rho_L + \rho_R$. Therefor, the firing duration of ρ_L should be the worst case latency reduced by the firing duration of ρ_R.

The second actor v_R is used to model the maximum rate or bandwidth of the interconnect. In the ideal case, both the firing duration ρ_R and latency ρ_L are 0 which implies unlimited bandwidth and no packet latency. As this is never the case, the self-loop on v_R determines the rate at which tokens are passed on to v_C. As v_R can fire once every ρ_R time units, its maximum throughput will be $\frac{1}{\rho_R}$.

We will now use the properties from the dual ring interconnect as described in Section III to obtain the CSDF model for hardware FIFO communication on the dual ring interconnect. As both credits and data traverse the interconnect, both paths in the model use a Latency-Rate Server to describe the channel as is shown in Figure 6.

From Section III-D we see that a ring with N tiles will fairly distribute all available bandwidth under full load and as such the worst case throughput will be $\frac{1}{N}$. The worst case latency for the transfer of a data packet over the ring is the combination of the stall time before a packet is admitted onto the ring and the traversal time on the ring itself. The worst

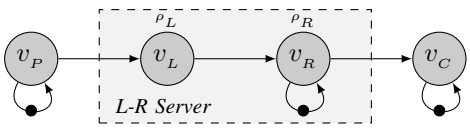


Fig. 5. HSDF model of a Latency-Rate Server

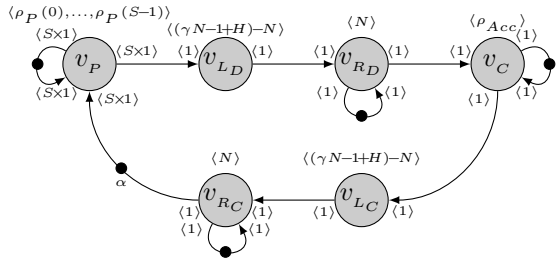


Fig. 6. CSDF model of hardware FIFO including channel characteristics

case stall time occurs when the input buffer at the NI is filled with γ data words and we missed the owned slot of the NI by one cycle. After $N-1$ cycles, the first data packet is injected into the owned slot from the buffer and every N cycles after that, the next data word is injected into the owned slot on the ring until all data is transmitted. As such the stall time is $\beta = (N-1) + (\gamma-1)N = \gamma N - 1$. The worst case latency for a single packet is as such the sum of stall time β and the traversal time based on the number of hops H a packet takes until it reaches its destination: $\beta + H = \gamma N - 1 + H$. As credit tokens traverse the dual ring in the opposite direction of the data, the latency for both data and credit packets is identical.

B. Guaranteed Throughput

In order to determine exact throughput and latency bounds, we need to transform the CSDF model from Figure 6 to its Homogeneous Synchronous Data Flow (HSDF) counterpart. HSDF is a simplified form of SDF where all quanta are one.

The model from Figure 6 does not contain the stall time introduced by the network at the producer when the input buffer at the NI is full. When we are only interested in the worst case stall time before a packet is admitted into the input buffer at the NI, we see that within $N-1$ cycles a packet from the input buffer is injected into the ring. One cycle later the input buffer can accept a new packet in the input buffer. As such, in the worst case, at the end of each phase of v_P, writing a single data word stalls the processor N cycles.

This stall time can be aggregated as the total stall time for any actor v during the write of S words:

$$\varphi = N \cdot S \tag{1}$$

This wait time can be incorporated into the original firing duration of actor v: $\hat{\rho} = \rho + \varphi$. As φ directly influences the firing duration of v_P, it should be kept small. This can be done by reducing the capacity of the input buffer γ at the NI. However, lowering γ can make the average case throughput worse by increasing the amount of stall cycles.

In order to determine the worst case throughput, we need to transform the model from Figure 6 from CSDF to its HSDF counterpart [3] in order to determine its Maximum Cycle Mean (MCM). The throughput is the inverse of the MCM. The MCM is the maximum of all Cycle Means (CMs) of every simple directed cycle C in the HSDF graph G:

$$\lambda^* = \max_{\forall C \in G} \{\lambda(C)\} \tag{2}$$

Where the Cycle Mean (CM) of a cycle C is defined as:

$$\lambda(C) = \frac{w(C)}{\tau} \tag{3}$$

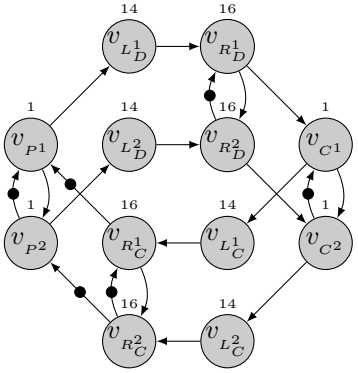


Fig. 7. HSDF obtained after transformation of the CSDF model from Figure 6

Here $w(C)$ is the sum of all firing durations of all actors on C and τ is the sum of number of tokens on all edges in C. The cycle with the largest CM is called the *critical cycle*.

In the transformation from CSDF to HSDF of the model for hardware FIFO communication, the value of α and S determine the number of actors in the resulting model and the critical cycle through them. We will show the transformation for the case where $\alpha = 2$ and $S = 2$ and derive the MCM. As the resulting HSDF model will have many cycles, rather than obtaining the MCM symbolically, we will present a numerical example. We use the following parameters: input buffer $\gamma = 1$, worst case traversal time is $H = 15$ for traffic on a ring with $N = 16$ tiles. Additionally we define that $\forall i \bullet \rho_{P}(i) = \rho_{Acc} = 1$, where i denotes all phases for v_{P}.

We transform the CSDF model to a HSDF model [3], which is shown in Figure 7. Using the parameters as specified before, we can now numerically calculate the firing duration for each actor. We find that two cycles have the maximum CM and as such are critical:

$$\begin{aligned} c_1 &= \left(v_{P^1}, v_{L_D^1}, v_{R_D^1}, v_{C^1}, v_{L_C^1}, v_{R_C^1}, v_{P^1}\right) \\ c_2 &= \left(v_{P^2}, v_{L_D^2}, v_{R_D^2}, v_{C^2}, v_{L_C^2}, v_{R_C^2}, v_{P^2}\right) \end{aligned} \tag{4}$$

The MCM can now be determined:

$$\lambda^* = \lambda(c_1) = \lambda(c_2) = \frac{62}{1} \tag{5}$$

The transformation of the CSDF model for other values of S and α is done in a similar fashion. For completeness, we show the MCM for various combinations of S and α. The resulting MCMs are presented in Table I where we can see that the dual ring is rate limited in this example by v_{R_D} when $\alpha = 4$. Latency introduced by the distance between v_P and v_C influences throughput only when $\alpha < 4$. As such, when an accelerator tile has a FIFO buffer of four words, our guaranteed lower bound on throughput is reached and further increasing α can only increase best effort throughput for this network size.

V. Evaluation

In this section we will evaluate the architecture implementation of the proposed interconnect by means of a demonstrator application. We will consider the hardware costs of both rings and accelerator shells compared to the size of processors and accelerators in the system.

		Container size in number of words (S)				
		1	2	3	4	5
FIFO capacity (α)	1	62	124	186	248	310
	2	31	62	93	124	155
	3	$\frac{62}{3}$	$\frac{124}{3}$	62	$\frac{248}{3}$	$\frac{310}{3}$
	4	16	32	48	64	80
	5	16	32	48	64	80

TABLE I

MCM for various S and α on for hardware FIFO communication between a processor and accelerator.

Data Ring:	2.1%	Credit Ring:	1.6%
Shells:	4.8%	AM Decoder Accelerator:	33%

TABLE II

Relative logic usage on a Virtex-6 FPGA of the NI components and accelerator compared to a MicroBlaze processor

A. Hardware Costs

We implemented our dual ring interconnect for an MPSoC with 16 CPU cores and one accelerator on a Xilinx ML-605 prototyping board. Previously, the single ring was shown to scale linearly with the number of processor tiles and occupy only ∼2% of the total hardware costs in the design [1].

When we consider our dual ring interconnect, the NI comprises of the interface for both the data and credit ring and the shell which implements hardware flow control. In Table II, we present the logic usage for our dual ring interconnect normalized to the size of a single MicroBlaze processor. The ring routers for both the data and credit ring use 3.7% of the amount of logic needed for a single processor. As we want every processor to be able to communicate with an accelerator, the shell is part of the NI. The combination of both ring routers and shell results in a NI which is 8.5% in size compared to a single processor.

B. PAL Video Decoder

In order to evaluate the performance of the proposed architecture we implemented a PAL decoder in software on a heterogeneous 16 core design. The PAL standard describes a field interlaced, 25 frames per second color video signal, usually in combination with FM modulated audio [22]. The signal consists of an Amplitude Modulation (AM) luminance signal (Y) and a quadrature modulated color difference signal (R-Y and B-Y) at 4.43 MHz from the luminance carrier.

Every horizontal line is separated by a sync pulse which uses a lower signal value than the rest of the inverted luminance signal, as shown in Figure 8. It is customary to normalize the input signal so that the luminance ranges from 0.0 to 1.0 where 0.3 and lower are used for line syncs. Each frame consists of two interleaved fields. All fields are separated by a number of synchronization pulses.

Our parallel decoding algorithm is depicted as a task graph in Figure 9. The actors in this task graph correspond with tasks where each task is using a dedicated processor. Some tasks are suitable for task level parallelism and are duplicated to improve application throughput. The decoder finds the sync pulses denoting the start of a fields, deinterlaces two fields into a complete frame and outputs it to the screen.

Most tasks from Figure 9 are simple signal processing tasks without complex control, field and type detection are control heavy tasks which are computationally bound and can not be duplicated and thus limit throughput of the entire decoder.

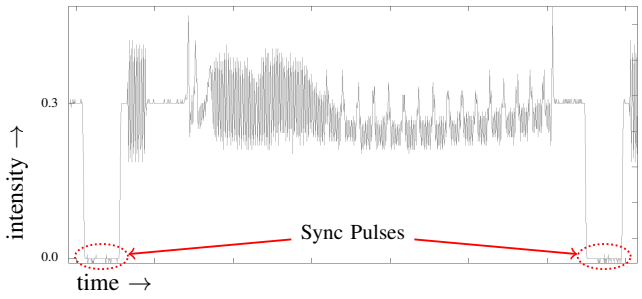


Fig. 8. Complete PAL line with two low valued synchronization pulses

However, in order to reach a speed of 3 MS/s or 12 MB/s, the "AM Demod" task has to be executed on four processors in parallel. This software task was replaced by a single accelerator tile containing a hardware decoder which freed up four processors. In Figure 8 this can be seen where the "AM Demod" tasks are replaced by the single, dashed task depicting the accelerator. The throughput over this accelerator is bounded by the speed of the software "ADC" and "Level Detect" tasks and improves throughput between those two tasks from 3 MS/s to 11 MS/s.

This is a 366% increase in throughput from the addition of a single accelerator which is only $\frac{1}{12}$ the size of the four processors it replaces.

VI. Conclusion

In this paper we presented a low cost dual-ring communication infrastructure for a real-time heterogeneous multi-core stream processing architecture for SDR applications. Our dual ring interconnect realizes low hardware cost by sharing buffers within a NI and provides automatic serialization while providing bandwidth and latency guarantees. The work conserving scheduling policy of the ring interconnect allows tasks to use the reserved but unused bandwidth of other tasks which can improve average throughput at the application level.

Software tasks communicate using a software FIFO implementation which only requires the data ring. Accelerator integration is supported by means of credit-based hardware flow control which uses a second ring for transferring credits.

We derived a CSDF model which describes a hardware FIFO between two tiles. This model can be used to determine the required FIFO capacity in order to reduce the effects of network latency on application throughput.

We found that our dual ring interconnect occupies 8.5% of the resources in a 16 core MPSoC design when implemented on an FPGA. We evaluated our design by means of a PAL video decoder application. When only software tasks are used, all 16 processors are used. Four processors could be freed up by using a single accelerator for the AM demodulation which at the same time improved maximum throughput by 366%.

The presented results indicate the potential of the dual ring communication interconnect for use in heterogeneous medium scale multiprocessor systems for real-time stream processing applications.

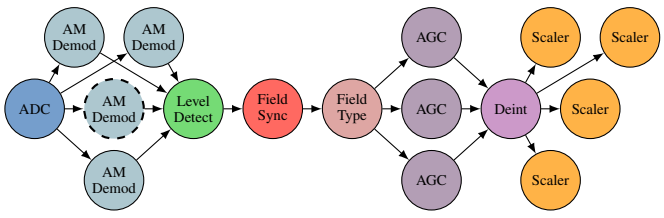


Fig. 9. Task graph of our PAL decoder application

References

[1] B. H. J. Dekens, P. Wilmanns, M. J. G. Bekooij, and G. J. M. Smit, "Low-Cost Guaranteed-Throughput Communication Ring for Real-Time Streaming MPSoCs," in *Proc. Conf. on Design & Architectures for Signal & Image Processing (DASIP)*. Europe: ECSI, 2013, pp. 239–246.

[2] A. Nieuwland and J. Kang, "C-HEAP: A heterogeneous multi-processor architecture template and scalable and flexible protocol for the design of embedded signal processing systems," *Design Automation for Embedded Systems*, vol. 7, no. 3, pp. 233–270, 2002.

[3] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, 1996.

[4] E. Rijpkema *et al.*, "Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip," in *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*. Washington, D.C., USA: IEEE Computer Society, 2003, pp. 350–355.

[5] M. Harmanci, N. Escudero, Y. Leblebici, and P. Ienne, "Providing QoS to connection-less packet-switched NoC by implementing diffserv functionalities," in *Proc. Int'l Symposium on Systems on Chip (SoC)*. New York, NY, USA: IEEE, 2004, pp. 37–40.

[6] Xilinx, *UG761 AXI Reference Guide*, 2011, vol. 761.

[7] Altera, *Avalon Interface Specifications*, 2013, no. May.

[8] D. Liu, D. Wiklund, and E. Svensson, "SoCBUS: The solution of high communication bandwidth on chip and short TTM," in *Proc. Real-Time and Embedded Computing Conference (RTECC)*, 2002.

[9] P. Wolkotte, G. Smit, G. Rauwerda, and L. Smit, "An Energy-Efficient Reconfigurable Circuit-Switched Network-on-Chip," in *Proc. Int'l Parallel and Distributed Processing Symposium (IPDPS)*. New York, NY, USA: IEEE, 2005, p. 155a.

[10] K. Goossens and A. Hansson, "The Aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proc. Design Automation Conference (DAC)*, 2010, pp. 306–311.

[11] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens, "A TDM NoC supporting QoS, multicast, and fast connection set-up," in *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2012, pp. 1283–1288.

[12] M. Kistler, M. Perrone, and F. Petrini, "Cell Multiprocessor Communication Network: Built for Speed," *IEEE Micro*, vol. 26, no. 3, pp. 10–23, May 2006.

[13] S. Kottapalli and J. Baxter, "Nehalem-EX CPU Architecture," *Hot chips*, pp. 1–19, 2009.

[14] M. J. Flynn, O. Pell, and O. Mencer, "Dataflow supercomputing," in *Proc. Int'l Conf. on Field Programmable Logic and Applications (FPL)*. New York, NY, USA: IEEE, Aug. 2012, pp. 1–3.

[15] M. Panić *et al.*, "On-chip ring network designs for hard-real time systems," in *Proc. Int'l Conference on Real-Time Networks and Systems*. New York, NY, USA: ACM Press, 2013, p. 23.

[16] F. Conti, A. Marongiu, and L. Benini, "Synthesis-friendly techniques for tightly-coupled integration of hardware accelerators into shared-memory multi-core clusters," in *Proc. Int'l Conf. on Hardware/software codesign and system synthesis (CODES+ISSS)*. New York, NY, USA: IEEE, Sep. 2013, pp. 1–10.

[17] B. Khailany *et al.*, "Imagine: media processing with streams," *IEEE Micro*, vol. 21, no. 2, pp. 35–46, 2001.

[18] G. Venkatesh *et al.*, "Conservation cores," in *Proc. Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. New York, NY, USA: ACM Press, 2010, p. 205.

[19] Xilinx, *LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c)*, 2010.

[20] N. Kung and R. Morris, "Credit-based flow control for ATM networks," *IEEE Network*, vol. 9, no. 2, pp. 40–48, 1995.

[21] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit, "Modelling run-time arbitration by latency-rate servers in dataflow graphs," in *Proc. Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*. New York, NY, USA: ACM Press, 2007, p. 11.

[22] ITU-R, "Rec. ITU-R BT.470-6: Conventional Television Systems," Tech. Rep., 1998.