Memory System for a Dynamically Adaptable Pixel Stream Architecture
Nicolas Ngan, Geoffroy Marpeaux, Eva Dokladalova, Mohamed Akil, François Contou-Carrère

To cite this version:
Nicolas Ngan, Geoffroy Marpeaux, Eva Dokladalova, Mohamed Akil, François Contou-Carrère. Memory System for a Dynamically Adaptable Pixel Stream Architecture. 20th International Conference on Field Programmable Logic and Applications (FPL’10), 2010, France. 10pp. hal-00622488

HAL Id: hal-00622488
https://hal-upec-upem.archives-ouvertes.fr/hal-00622488
Submitted on 21 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Memory system for a dynamically adaptable pixel stream architecture

Nicolas Ngan\textsuperscript{1,2}, Geoffroy Marpeaux\textsuperscript{1,2}, Eva Dokladalova, Mohamed Akil
\textsuperscript{1}Laboratoire Informatique Gaspard Monge, Equipe A3SI
\textsuperscript{2}Unite Mixte CNRS-UMLV-ESIEE (UMR 8049)
Emails: \{ngann, marpeaug, e.dokladalova, akilm\}@esiee.fr
François Contou-Carrère
\textsuperscript{2}Sagem, Massy-Palaiseau
Groupe SAFRAN, France
Email: francois.contou-carrere@sagem.com

\textbf{Abstract}—Nowadays, embedded vision systems have to face new hard requirements involved by modern applications: real-time processing of high resolution images issued by multiple image sensors. Recently, a new adaptable ring-based interconnection network on chip has been proposed. Based on adaptive datapath, it allows handling of multiple parallel pixel streams. In this paper, we present a new hierarchical memory system proposed for this adaptable ring-based architecture. The design of its different levels is discussed and we show how the memory system adapts dynamically with respect to the datapath and data access management in the interconnection network. We also present the timing performance and area occupation measured on an FPGA prototype.

\textbf{Index Terms}—memory, FPGA, adaptable, embedded, image processing, ring

\section{Introduction}

Modern embedded vision systems have to face growing demands in image sensor multiplication [1]. Moreover, they have to implement a wide variety of applications (i.e. zoom, image filtering, fusion, face detection). Those applications introduce constraints in the amount of data, the processing time and latency (i.e. the delay allowed from input to output). Also, the system has to support variable real-time constraints from 25 to 50 frames per second with an image size up to the 1080p HD resolution (1920x1080). Such requirements represent the challenge of how to design the system to run efficiently multiple-sensor applications.

If it has been demonstrated in [2] and [3] that the computing power does not represent a major problem, the bottleneck remains on the management of multiple streams and the efficient parallel data access. To illustrate, we can cite some critical applications such as image fusion [4], [5], multiscale approaches [6] and especially temporal processing with multiple delayed frames.

Recently, we have proposed a new adaptable ring-based interconnection network on chip [7]. The main principle at the system level is shown in Figure 1. An application loader precharges the application context into the processing system from the application memory. The system then dynamically adapt its datapath according to the running imaging application. This datapath adaptation allows handling parallel computation for multiple streams. It involves the need for a performant adaptable memory system. In particular, that memory system has to be adapted dynamically in datapath and data access.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{fig1.png}
\caption{Adaptable Ring-Based Architecture Context}
\end{figure}

In this paper, we propose a new memory system applied on the published adaptable ring-based architecture. That memory system is dynamically adaptable and it is organized in two levels: a frame level and a line level. The frame memory level allows data accesses by blocks of data and the line level allows random data accesses.

The paper is organized as follows. The Section II is a presentation of the Multi Data Flow Ring (MDFR) architecture. MDFR is an original adaptable ring-based interconnection network on chip. The dynamically adaptable memory system organized in two levels is described in Section III and IV. Results in space and timing performance are then presented for its first hardware prototyping in Section V.

\section{Adaptable Ring-Based Pixel Stream Architecture}

An imaging application is defined as one or a set of associated algorithms. We can cite for example a 2D convolution. An algorithm describes a sequence of operations to carry out by a specified task \(T_i\). Thus, the imaging application is described as a predefined sequence of a set of tasks \(\{T_i\}\). From data flow point of view, that sequence defines an association of different execution modes: pipeline, parallel and pipeline-parallel as shown in Figure 2. We call a pipeline execution mode a highly optimized datapath used to connect in a series computing ressources in order to process a given pixel stream. A parallel execution mode is considered when there are datapath divisions to have computing ressources processed.
tasks in parallel. Finally, a *pipeline-parallel* execution mode is a combination of *pipeline* and *parallel* execution mode.

Let consider in the following of this paper that those tasks $T_i$ are implemented in a processing element $PE_i$ and that the *execution modes* are realized by a defined datapath.

![Fig. 2. Illustration of different execution modes](image1)

Depending on the application, those *execution modes* might also have an impact on the memory requirements. For instance in *parallel* execution mode, processed parallel streams might need to be synchronized and a memory is then required. The size of the memory can vary from a few lines to a complete frame. Also, each $T_i$ can require different manners to access data.

We distinguish two different *data access modes* in memory: data access by *lines* and data access by *pattern*. We define a data access by *lines* as a linear access and data access by *pattern* as a random access for accessing a specific block of data.

### A. Architecture description

The proposed architecture called *Multi Data Flow Ring* (MDFR) is described in Figure 3. It is a dynamically adaptable architecture for real-time imaging applications in embedded vision systems. It can be adapted in both datapath and memory accesses. The architecture contains two data router types: *Data Flow Routers* (DFR) and *I/O Stream Gate Manager* (SGM) nodes linked together forming a ring topology. We define a *ring channel* as a unidirectional communication link between routers. DFRs and SGMs are connected through four data ring channels: two rings oriented in clockwise direction and the two others in the inverse direction as shown in Figure 3. Those two directions allow the pixel stream to reach a PE through DFR from both directions in the shortest possible datapath.

Main datapath adaptations are made with the DFRs to realize application execution modes. A DFR is a router in charge of redirecting the pixel streams to its connected processing element $PE_i$ executing a task $T_i$. Its main actions are switching input pixel streams for its connected PE or forwarding it to the next DFR in the ring. It contains buffers to synchronize input and output streams, multiplexers, configuration registers and pixel counters to track the status of PE data output.

SGMs are the master units of the architecture. They control the pixel streams in the ring channels. A SGM is a particular data router that contains dedicated ports for external input and output streams. Each SGM has an access to the *Frame Buffer System* (FBS) which is in charge to store and manage the frames from pixel streams.

A SGM has two levels of memory (Figure 4). The first level is the *Line Memory System* (LMS) inside the SGM and the second level are the *Frame Memories* inside the *Frame Buffer System* (FBS).

The main controller inside the SGM can make a read or write request to the controller called *Frame Server* inside the FBS. Pixels stored in LMS are then transferred from the FBS or to the FBS. Each memory level has specific controllers ensuring memory accesses and datapath adaptations. Table I summarizes the memories characteristics (memory type, master, data access mode).

<table>
<thead>
<tr>
<th>Memory</th>
<th>Type</th>
<th>Master(s)</th>
<th>Data Access Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS</td>
<td>Local</td>
<td>SGM</td>
<td>Lines, Patterns</td>
</tr>
<tr>
<td>Frame Memories</td>
<td>Shared Global</td>
<td>SGMs</td>
<td>Lines</td>
</tr>
</tbody>
</table>

For the SGMs, the LMS is considered as a *local memory* and the FBS as a *shared global memory*. The LMS can be read in a predefined way to extract different patterns from input frames to be sent in ring channels (Figure 3). In the following of this paper, we focus on the memory system adaptations.
III. STREAM GATE MANAGER ARCHITECTURE

A SGM interfaces three types of pixel streams sources from ring channels ports, the external port or the FBS port. Let $K_c$ be the number of ring channels connected to the SGM, $K_{ext}$ the number of external input ports and $K_{fbs}$ the number of input ports from FBS. Consequently, the maximum number of input pixel streams $K$ to manage by the SGM would be:

$$K = K_c + K_{ext} + K_{fbs}$$

(1)

An overview of SGM architecture is given in Figure 5.

**A. Line Memory System Architecture**

The LMS architecture is presented in Figure 6. It allows both datapath and memory adaptations for six input pixel streams. Datapath adaptations are realized with four 2-input multiplexers, one $4 \times 4$ crossbar, and 4-input multiplexers. The LMS contains four memory units called Line Memory Channel (LMC). As shown in Figure 6, each LMC is connected to one ring channel at its output. Consequently, the Datapath and Memory Access Configurator (Figure 5) has to select the ring channel output with the crossbar (Figure 6). Note that pixel streams from ring channels can bypass directly the LMS without any stream modification from input registers to output registers. That possibility is important when a pixel stream has to cross several times SGMs in the ring for reusing or finding a particular PE.

![Fig. 5. I/O Stream Gate Manager (SGM) architecture](image)

**B. Line Memory Channel Architecture**

Each LMC can accept up to one source type pixel stream from ring channel, external and FBS. A LMC can output pixel stream to a ring channel, to external output or to the FBS for writing. A LMC contains two main memories: Memory Up and Memory Down. Memory Up is used to store streams to
be sent to a ring channel and Memory Down is used either to output the stream to an external port or to write the buffered stream to the FBS. Thus, a pixel stream from ring channels can be oriented to any LMC to access an external port or to be written in FBS. The memory access adaptation is decoded by the Pixel Stream Extractor. As shown in Figure 7, it contains a Command Decoder unit which receives datapath and memory access requests from the Datapath and Memory Access Configurator in the SGM (Figure 5). Datapath is decoded to configure the input mux \( a \) and \( b \) to select the right pixel streams to process. The adress generator units for Memory Up and Memory Down are then configured to write input pixel streams linearly and to realize an access mode. An access mode by lines configures the adress generators unit to read the pixels linearly. Patterns are predefined for each address generators and memory addresses are computed according to the line width value of the frame. For instance, it can be configured to jump pixels for a fast frame downscaling or access particular block of pixels. Obviously, the pixel block size access is limited by the size of Memory Up or Down for a giving line width value and pixel granularity. A defined access mode configures also the Pixel Format units according to the pixel granularity in the stream such as 8, 16 or 32 bits only. The Pixel Format unit adapts the output pixels data of the Memory Up and Down to the ring channel data size by using byte masks and successive required concatenations. For instance, four 8-bits pixels to get the size of a 32-bits data in the ring channels.

B. Datapath and Memory Access adaptation control

LMS has two levels of datapath control. The first level is selecting the right LMC for pixel streams coming from ring channels and redirecting the LMC streams output either to a ring channel, to external or to FBS (Figure 6). The second level interests the selection of Memory Up and Memory Down inside a LMC (Figure 7). For an input pixel stream, there can be five datapath instruction requests from the PixelStream controller:

- FW(D) : Forward the input pixel stream to the direct associated ring channel
- FW(C) : Change the direction of the input pixel streams to another ring channel
- READ : Read another pixel streams in FBS to be processed
- STORE : Store the input pixel stream to FBS
- OUT : Exit the input pixel stream to the external port

Table II summarizes the cost in memory (Memory Up and Down) and datapath utilization (Ring channels) for an input pixel stream from a ring channel. The Datapath and Memory Access Configurator then takes a decision on the datapaths according to the maximum input streams capacity \( K \) and the availability of Memory Up and Down. After selecting the datapath, if a pixel stream has to be stored in Memory Up or Down, the Datapath and Memory Access Configurator has the possibility to define a specific access mode by lines or patterns by configuring the address generators in each LMC.

### IV. FRAME BUFFER SYSTEM ARCHITECTURE

A. Frame Slot Memory definition

We define a frame slot as a frame at a specific time index \( ts \). The time index \( ts \) represents the number of frames delayed from the current processed frame. Frames are stored in a frame memory at a physical memory address. A frame memory can then contains several frame slots.

Let \( M \) and \( N \) be the width and height of a frame slot. Let \( r \) be the pixel data size in bits. The memory block size \( S_F \) would be:

\[
S_F = M \times N \times r
\]  

We call frame slot memory the memory block of size \( S_F \) storing a frame slot. For example, the frame slot memory size for a complete HD greyscale image of 1920x1080 in 8 bits would be about 2 Mbits. Thus, let \( S_{LM} \) the size of the frame memory. The number \( P \) of available frame slot memory in the frame memory would be:

\[
P = \left\lfloor \frac{S_{LM}}{S_F} \right\rfloor
\]  

The Frame Buffer System (FBS) is based on the proposition to divide dynamically a frame memory in several frame slot memories. Each frame slot memory is identified by attributes such as the time index \( ts \), the pixel granularity \( id \) and the processing element \( n \) which has processed the frame slot.

B. Frame Buffer System architecture

The FBS architecture is presented in Figure 8. It contains a Frame Server which receives commands (read or write) from SGMs and controls the adaptation units. The adaptation units are the The Memory Access Controller and the Data Output Controller. The Memory Access Controller generates the addresses to access the frame memories and the Data Output Controller adapt dynamically the frame memories output datapaths with a crossbar to reach any SGMs.

The FBS contains as many frame memory as the number of SGMs in the ring (See Figure 8 with four SGMs). Each SGM is connected to its private frame memory in writing. The set of frame memories are designed to get a private access in writing but a shared access in reading for all SGMs.

Each frame memory is divided in frame slot memories and those slots are managed by the Frame Slot Manager. All frame slots memory attributes (time index \( ts \), the pixel granularity \( id \) and the processing element \( n \)) are registered.

<table>
<thead>
<tr>
<th>Memory Up</th>
<th>Memory Down</th>
<th>Chan Out</th>
<th>I/O Out</th>
<th>FBS In</th>
<th>FBS Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW(D)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FW(C)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>READ</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STORE</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>OUT</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### TABLE II

**Datapath and Memory Costs for an input pixel stream from Ring Channel**
in Frame Register Tables. As shown in Figure 9, the Frame Register Tables contain the addresses connections between frame memory slots and the true physical memory addresses. The Frame Slot Manager unit registers in a free frame slot memory each incoming frame slot to the FBS. The Frame Register Tables are dynamically updated by the Frame Slot Manager. If the frame slot size (i.e. $S_F$) is modified, it can reallocate dynamically the frame slots memory by updating the physical memory address table.

The FBS architecture has several advantages. First, the physical memory addressing is completely transparent for the SGMs. Frame read or write requests to the Frame Server are done indirectly with the frame slot attributes and not with the physical memory address. Thus, it allows reusability and evolution to any frame memory size in the FBS. The frame management stays completely independent. The Frame Register Tables size only depends on the defined number of frame slot memory to manage and not the memory size. Secondly, it allows processing a full frame by blocks sequentially or in parallel. If the incoming frame size is higher than the defined frame slot memory size, then it is automatically divided and considered as several lower frame slots to be processed. It is particularly interesting for processing elements limited in frame size. Thus, imaging applications can also be accelerated by propagating the pixel stream to each SGM and duplicating the frame to each frame memory. Each SGMs can then access in parallel to separate parts of the frame considered as lower frame slots. For instance, an HD $1920 \times 1080$ image can be considered as sixteen $480 \times 270$ frame slots to be processed.

C. FBS Datapath and Memory Access control

The FBS uses adaptation algorithms for reading and writing on the different memories. It contains a reading queue in case of simultaneous reading requests on the same frame memory.

For writing a frame, the Frame Server has to decode the attributes of the requested frame slot (the time index $t_s$, the pixel granularity $id$ and the processing element $n$) and the pixel stream size $q$ (number of pixels). It asks the Frame Slot Manager for a free frame slot memory. From the frame slot attributes, the Frame Slot Manager can address the Frame Register Tables and returns to the Frame Server the physical memory address of the last free slot memory (Figure 9). The physical memory address is then transmitted to the Memory Access Controller with $q$ to initialize the writing address generator unit of the frame memory. During the writing, the Frame Slot Manager updates the table by linking the frame slot memory address to the attributes.

For reading a frame slot, the Frame Server has to decode also first the attributes to get the physical memory address. If the frame memory is still busy, it puts the read command in the reading queue. If the frame memory is available, then the Frame Server configures the Data Output Controller to adapt the frame memory output datapath. The reading address generator unit of the frame memory is configured with the physical memory address and the parameter $q$.

V. HARDWARE PROTOTYPING

The presented adaptable memory system has been prototyped on an Altera FPGA Stratix III EP3SL150. For evaluation of this implementation, we focus on the memory system adaptation cost in area (logic elements) and time (latency). We consider working with frame slots of size $10 \times 100$ Bytes (B). The whole system is working at the same frequency (maximum frequency of $210$ MHz) and we use the on-chip memory to implement the memory system. We define the ring channels data size to 32 bits and the instruction memory to contain $8 \times 1$ B instructions.

A. SGM implementation

All the SGM control units (Pixel Stream controller, Frame Reader, Frame Writer and Datapath and Memory access configurator) are implemented as Finite-State Machines (FSMs). The Pixel Stream controller contains 16 B FIFO memories to store pixels during adaptations. The Memory Up and Down in LMCs are implemented as 32-bits dual-port memories to fit $2 \times 100$ B (two lines). Table III shows the implementation results for one SGM with the parameters $K_c = 4$, $K_{ext} = 1$ and $K_{fbs} = 1$.
Note that the results for the control units depend only on the parameters $K_c$, $K_{ext}$ and $K_{fbs}$. Table IV shows the SGM latency in clock cycles for one pixel stream in a pipeline execution mode and for each adaptation requests described in Table II.

### TABLE IV

**SGM: LATENCY (IN CYCLES) FOR ONE PIXEL STREAM IN PIPELINE**

<table>
<thead>
<tr>
<th>Instructions</th>
<th>LMC</th>
<th>Adaptation</th>
<th>Memory</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW(D)</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>FW(C)</td>
<td>$2 + \alpha$</td>
<td>6</td>
<td>$\beta$</td>
<td>$8 + \alpha + \beta$</td>
</tr>
<tr>
<td>READ</td>
<td>$2 + \alpha$</td>
<td>8</td>
<td>$\beta$</td>
<td>$10 + \alpha + \beta$</td>
</tr>
<tr>
<td>STORE</td>
<td>$2 + \alpha$</td>
<td>8</td>
<td>$\beta$</td>
<td>$10 + \alpha + \beta$</td>
</tr>
<tr>
<td>OUT</td>
<td>$2 + \alpha$</td>
<td>6</td>
<td>$\beta$</td>
<td>$8 + \alpha + \beta$</td>
</tr>
</tbody>
</table>

To cross a SGM, each stream has to wait the datapath and memory adaptations inside the SGM. The fastest adaptation is forwarding directly (FW(D)) the stream because the datapath in LMC is just modified at the first level in 2 clock cycles. If the stream has to be stored an LMC (i.e. with instructions FW(C), READ, STORE, OUT) then it adds 4 clocks cycles to communicate with the Pixel Extractor and wait for its handshake (signal `ack` in Figure 6 and 7). Reading or writing to the FBS adds 2 clock cycles to initialize the Frame Reader and Writer (Figure 5). $\beta$ is the time for the address generators to be initialized for the access pattern. In our case, $\beta = 1$ for a linear access and $\beta = 3$ for accessing a block. The time $\alpha$ is the latency for buffering the pixels in LMC according to the access mode. For instance, for accessing the frame each two lines of 100 pixels, $\alpha = 50$ clock cycles. Note that using parallel or pipeline-parallel execution mode only impact the datapath latency for FW(D) to 6 because the stream has to be stored in LMC in order to use a second ring channel.

#### B. FBS implementation

For the FBS implementation, we use four frame memory implemented as 32-bits dual-port memories of size 16384 B to fit 10 frame slots of $10 \times 100$ B. Frame Server and Frame Slot Manager are implemented as FSMs. The Memory Access controller is a FSM with several counters for each address generators units. Table V presents the FBS implementation results for managing 10 Frame slots. Note that the size of frame slot does not have any impact on the size of control units, datapath adaptation unit and memory access controller.

Table VI shows the FBS latency for read and write requests. Checking the Frame Register table requires 3 clock cycles to get the translation between the attributes to the physical memory address. A register store the last free frame slot to accelerate the writing. $\delta_1$ is the waiting time for reading if the frame memory is busy (from 0 to 250 clock cycles equivalent of reading one $10 \times 100$B frame). $\delta_2$ is the waiting time for writing if the Frame Buffer is still updating the Frame Register Tables (from 0 to 30 clock cycles for 10 slots).

From those results, we can conclude that the memory system latency cost for an input stream to cross a SGM is from 4 to 14 clock cycles. The latency cost for the FBS is 8 clock cycles and it can be overlapped by the SGM adaptation latency. Obviously, the more combined instructions or number of input streams are used and the more efficient the adaptation is. For long pixel stream pipeline, thoses latencies become negligible and the ring structure remains performant. Besides, those small latencies allows to have reduced sized FIFO buffers for each SGM in the ring.

### VI. CONCLUSION

In this paper, we have presented a new adaptable hierarchical memory system to address streaming imaging applications with several pixel streams. It can then be adapted dynamically to manage different frames from image sensors or delayed frames stored in memory. The memory system adaptation is both based on datapath and memory accesses to satisfy each execution mode. A first FPGA prototype has been presented and evaluated. Future works will focus on power consumption.

### REFERENCES


