

Modeling Dynamic Partial Reconfiguration in the Dataflow Paradigm

Jonathan Piat

¹ CNRS, LAAS, 7 avenue du colonel Roche
F-31400 Toulouse, France

² Univ. de Toulouse, UPS, UTM, LAAS
F-31400 Toulouse, France
jpiat@laas.fr

Jeremie Crenne

¹ CNRS, IMS, 351 Cours de la Liberation
F-33405 Talence, France

² Univ. of Bordeaux, IPB, ENSEIRB-MATMECA
F-33405 Talence, France
jcrenne@enseirb-matmeca.fr

Abstract—An important number of studies have shown the benefit of dynamic partial reconfiguration in reconfigurable computing. Signal processing applications can make use of this technology in such a way that it allows greater flexibility, performances and cost reduction. However several points still need to be addressed and represent critical challenges. One of them concerns architectures modeling as abstraction is strongly required to help designers in building efficient designs. Dataflow is a well adopted modeling paradigm for signal processing application to allow early stage system properties evaluation.

This paper describes a first attempt to model dynamic partial reconfiguration in the dataflow paradigm. Our proposal leads to an efficient and simple approach suitable for signal processing systems.

I. INTRODUCTION

Dynamic partial reconfiguration (DPR) is an advanced hardware technique to reconfigure particular areas of a field-programmable gate array (FPGA) while the rest keeps running its execution. Digital signal processing (DSP) applications take advantage of reconfigurable computing [1] and DPR by changing tasks in the pipeline while keeping the overall system functional (Fig. 1).

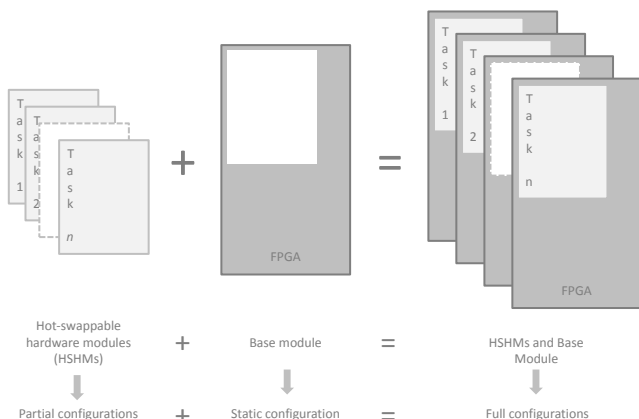


Fig. 1. Spatial and temporal partitioning of an application in a FPGA composed of one partially reconfigurable (PR) region and one static region

Technology limitations restricts reconfiguration both in term of granularity and placement but also takes time during

which part of the system is unavailable. As a consequence, using DPR comes with great challenges to ensure reconfiguration is performed at a convenient time and minimize its impact on the system latency and memory footprint.

Dataflow modeling is a well known paradigm to model signal processing applications and analyze properties of a system at compile-time. The application is captured as a network of actors exchanging tokens along communications edges. The behavior of an application is only studied regarding data dependencies in the model with only a little of knowledge about the actor itself (token production/consumption and guard conditions). Dataflow models handle dynamic behavior either regarding the worst case scenario or by considering the the dynamic behavior level at actor or network level. Such model can be used in a model-driven application design to ensure system properties in early design stages. As for now, no mean of DPR properties modeling exists in the dataflow paradigm. In this paper, we propose to extend the static dataflow model with additional properties to provide compile-time analysis of the DPR influence on the system. These new semantics allow data extraction such as DPR time slots and memory requirements.

This paper is organized as follows. We start by introducing DPR and signal processing in Sec. II. We then present the dataflow modeling paradigm in Sec. III. Sec. IV details the implementation of DPR in the dataflow paradigm and Sec. V describes the scheduling. We then review a SystemC simulation framework compatible with DPR dataflow model requirements in Sec. VI. Sec. VII highlights a practical case-study and we conclude with some final observations in Sec. VIII.

II. DPR AND SIGNAL PROCESSING

FPGAs were introduced in the mid-1980s [2] and the idea of using FPGAs for parallel computation only is no more disseminated [3]. FPGA implementations of applications are now widely spread and can be classified into five large classes including signal processing, cryptography, arithmetic, scientific computing and networking, and are surveyed in [4]. The reconfigurable nature of SRAM-based FPGAs also enables application to change their configuration throughout their operational lifetime, at runtime. Implementation of hardware spatial computations and hardware organization can change during every single computations steps. This specialization to the

instant needs of the application improves overall performances by reducing the number of devices or the device size, thereby reducing, weight and energy [5] [6] [7].

One challenging limiting factor in the development of DPR tools is the identification of applications that can benefit from the approach to justify numerous extra design steps and run-time management overhead. This issue motivated the need for additional and comprehensive models and tools to help designers in identifying regions that could take advantage of dynamically hot-swapped hardware and could be implemented into a device using DPR and software to manage the wide deployment of device configurations at run time.

Reconfigurable hardware and DPR modeling emerged in previous literature by means of model driven engineering (MDE) methodology especially with unified modeling language (UML) extensions [8] [9]. DPR properties modeling in dataflow represents an opportunity to contribute to the wide adoption of dynamically reconfigurable designs in signal processing applications by leveraging technological limitations.

III. DATAFLOW MODELING

The dataflow model of computation aims at describing applications regarding data dependencies. The application is represented as a network of actions in a direct acyclic graph (DAG) allowing to get knowledge of intrinsic parallelism and memory requirement. The dataflow paradigm was formally introduced by Kahn in [10]. A Kahn process network (KPN) is made of actors connected by unbounded first-in first-out (FIFO) queues carrying data tokens. A data token is an atomic (can not be divided) chunk of data. An actor can thus be described as a functional process that maps a set of tokens sequence into another set of tokens sequence. The dataflow process network (DPN) [11] inherits its formal underpinning from Kahn process networks but associates a set of firing rules to each actor to give the necessary tokens input for an actor to be triggered. This allows further analysis of the network using a set of properties. For further compile-time analysis, the synchronous dataflow model (SDF) [12] takes the dataflow process network semantic and restricts its expressiveness by specifying the data production/consumption rate in an integer form for each actor interconnection. To compactly represents every connections between actors in the dataflow graph G a topology matrix T is specified. This matrix is one way to find the rate at which each actor should be fired. Such mandatory information allows to ensure the application to be deadlock free, to compute a statical schedule of the application and to compute memory requirements at compile time (Fig. 2).

One major drawback of SDF is related to hierarchy handling and dynamic reconfiguration of the network topology. Parameterized SDF (PSDF) introduced in [13] propose to enable network topology run-time reconfiguration by using a parameterized network (production/consumption are parameterized) whose parameters are solved at run-time. The SDF model does not specify how hierarchy should be handled and as a consequence the schedulability of a graph can only be established on a flat graph. The interface-based SDF (IBSDF) model solves this issue by introducing the concept of interface and define their behavior at the model level. The parametrized and interfaced dataflow meta-model (PiMM) [14]

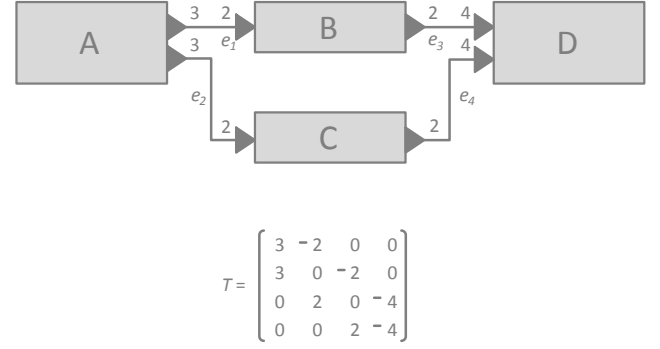


Fig. 2. A hypothetical SDF model of a dataflow graph G composed of four actors A, B, C and D connected by four edges e_1 , e_2 , e_3 , e_4 . Numbers at the inputs and output of an actor represent consumed and produced tokens respectively. Its corresponding topology matrix T is shown at the bottom. The row entries are the edges (connections) and the column entries are the vertices (actors)

later specified how this model could be extended to allow model composition.

Expressivity, predictability and dynamicity levels of cited models are summarized in Fig. 3.

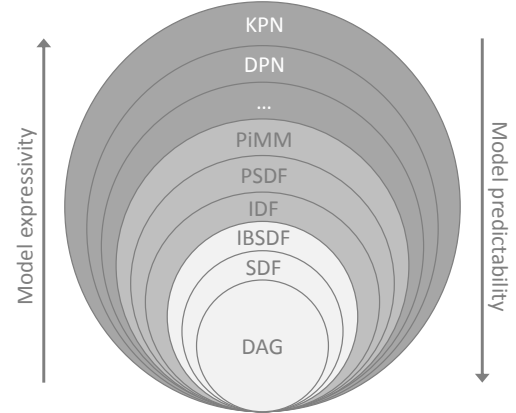


Fig. 3. Taxonomy of introduced models. Gray levels indicate the dynamic nature of a given model. The darker, the more the model can express dynamicity

IV. MODELING DPR IN THE DATAFLOW PARADIGM

DPR time overhead is high, usually several milliseconds, making it hard to handle critical real-time signal processing systems. Works such as in [15] improves the reconfiguration latency by using bitstream compression techniques. This allows to envision DPR usage onto uninterrupted data streams without losing any data. In order to use DPR in an efficient way, a high level study has to be done to make sure it is valuable for a given application. Modeling DPR at the dataflow level should help designers to make early choices regarding the relevance of using DPR.

Performing dynamic partial reconfiguration for a streaming application requires to perform the four following steps :

- 1) Enable the buffering of the incoming data for the actor to reconfigure
- 2) Trigger reconfiguration of the actor and wait for it to complete
- 3) Disable the buffering of the incoming data so the actor can resume
- 4) Wait for buffered data to be consumed to authorize new reconfiguration

Modeling the DPR can be performed at two distinct levels in the dataflow paradigm:

- 1) At atomic actor level. Only a single actor without any hierarchy is considered. Its behavior can evolve at run-time.
- 2) At network level. Reconfiguration considers a sub-graph of a dataflow actor. Its network topology can evolve over time.

A. Modeling DPR for atomic actors

The interface-based hierarchical synchronous dataflow (IB-SDF) model depicts dynamic behavior at the actor level while maintaining production/consumption rate (interface behavior). Such property helps to ensure that the change in the behavior won't impact the static application schedule. It is suited to represent a reconfigurable actor whose interfaces behavior will not evolve over time. The dynamic behavior of an actor is modeled as a reconfiguration overhead and an availability model. This reconfiguration overhead is expressed in actor firing time/latency unit. We also propose to introduce an availability model for the actor defining how often the actor is unavailable for computing due to ongoing reconfiguration. This availability is expressed in reconfiguration per firing and must be less than 1. This two associated metrics force the scheduler to extract data to construct a reconfiguration scheduler and define the impact of reconfiguration on system latency. A DPR actor is modeled as follows :

Let V be a IB-SDF actor with dynamic behavior. We define the DPR attributes of V as $DPR(V) = \langle o, a \rangle$ with o being the reconfiguration overhead and a being the availability model.

B. Modeling DPR at network level

As stated in Sec. III, a SDF model does not allow for dynamic behavior integration at network level. Instead, PSDF was later introduced to address this issue while maintaining compile-time schedulability by defining two distinct actor parametrization times : init and sub-init. The init phase ϕ_i is triggered at actor instantiation while the sub-init ϕ_s is triggered for each firing of the PSDF actor. These two phases have the following impacts on the application behavior :

- Actors parameters configuration : argument configuration only affects the actor behavior without affecting its production/consumption rates. This configuration can happen both at the sub-init and init phases of the PSDF graph execution.
- Network parameters configuration : network configuration can modify the production/consumption rates of the network. Token production/consumption rates

are captured as symbolic values in the model to allow schedulability check at compile-time. It can be resolved at run-time using tokens on parameter ports in the init phase of the PSDF run-time.

As a consequence, the dynamic aspect of the dataflow implies scheduling/partitioning and memory footprint changes. To limit PSDF init impact on the overall application, it can be decided to use parameters to emulate the behavior of the integer-controlled dataflow (IDF) model. In that case, the model is composed of multiple exclusive paths that are activated depending on parameters values. The scheduler schedules every independent path separately, without recomputing the schedule at runtime. We limit ourselves to that specific case so that we have a compile-time knowledge of the model schedule.

To separate scheduling concerns, PSDF makes use of hierarchy to enclose PSDF domains and limits the impact of sub-init and init procedures. Modeling reconfiguration as a PSDF actor becomes trivial and is seen as an atomic actor with previously stated attributes.

V. SCHEDULING DATAFLOW AND DPR

In the following section we only consider the case of a single DPR actor in the application. The proposal will later be extended to handle multiple actors. Applying clustering techniques to group DPR actors allows to deal with application including multiple reconfigurable actors.

A. Compute buffering requirements on DPR actor inputs

Incoming data of a DPR actor need to be buffered as the data sources cannot be stopped. A good balance between latency and memory footprint is achievable by setting buffer placement and dimension. Buffering data streams at the input of the actor to reconfigure may however not give the best memory performance. To use as less memory as possible, a decision is required to inform the model where the buffering has to be added. The example in Fig. 4 considers an SDF network with actor D being an actor to reconfigure. In this example buffering can happen directly at D inputs (buffering spots 1, 2) or earlier in the dataflow (buffering spot 3). Depending on edges properties of the network, an earlier buffering in the data stream may allow to achieve a smaller memory footprint.

This computation is performed by analyzing all incoming edges of the DPR actor and recursively evaluating the memory cost of the input path. With $c(e)$ being the token consumption of the incoming edge e and o_v being the reconfiguration overhead for the actor v expressed in number of firings. The buffer size of an input edge of a DPR actor is computed using the following equation :

$$buffer_{size} = (c(e) \times o_v)$$

The buffering cost b_{cost} at a given distance $dist$ from the edge source $src(e)$ to the edge sink $sink(e)$ is expressed in number of vertices from the DPR actor and is performed by the following recursive cost evaluation algorithm, where E is

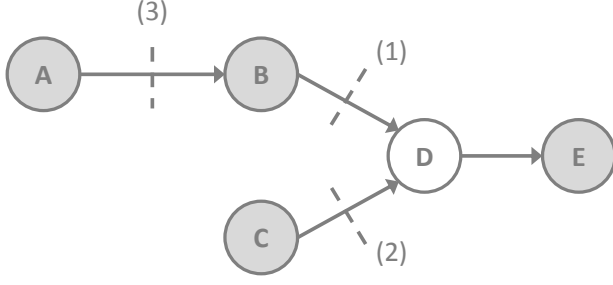


Fig. 4. An abstract DPR dataflow model with five actors A, B, C, D and E. One DPR actor D (unfilled white circle) and three potential buffering spots 1, 2 and 3 (dashed lines) are depicted. Other actors are represented with filled gray circles

the edge set and $p(e)$ is the token production :

```

1: Function  $buffer_{cost}(E, e, v, o_v, dist)$ 
2: if  $dist > 0$  and  $[e \in E | sink(e) = v] \neq \emptyset$  then
3:   for all  $e \in E | sink(e) = v$  do
4:      $b_{cost} \leftarrow b_{cost} + buffer_{cost}(E, e, src(e), o_v \times$ 
        $(c(e)/p(e)), dist - 1)$ 
5:   end for
6:   return  $b_{cost}$ 
7: else
8:   return  $o_v \times p(e)$ 
9: end if
10: EndFunction

```

This function can be used in minimization procedures such as simulated annealing (SA) [16] or genetic algorithm (GA) [17] to find the best buffering spot in the graph. However, it only supports a fixed distance in a multiple incoming path graph and needs to be improved for the multi-path case.

While this function minimize buffering with the selection of optimal spots, the DPR scheduling must be performed to evaluate the buffer size $buffer_{size}$.

B. DPR scheduling in dataflow

DPR could be modeled directly in the dataflow model but would greatly decrease model readability. This issue is addressed by choosing to schedule a dedicated DPR layer aside the basic dataflow schedule. The reconfiguration layer is in charge of defining the reconfiguration instant for each DPR actor, either based on user defined reconfiguration scheme or on network tokens in the case of PSDF model. When a reconfiguration needs to be triggered for a given actor, it performs the following actions :

- 1) Enable buffering of DPR area
- 2) Wait for actor to complete on-going process
- 3) Trigger reconfiguration
- 4) Wait for reconfiguration to complete
- 5) Disable buffering of DPR area

Because reconfiguration can be considered as a single resource being shared between every DPR actors, it needs to

be statically scheduled to guarantee application latency. The reconfiguration layer schedule is extracted from the network specification and executed as a two phases process: 1) enable buffering for each DPR actor before executing graph schedule and 2) trigger sequentially each DPR in parallel with dataflow graph execution.

VI. SIMULATION OF DPR AND DATAFLOW IN SYSTEMC

In [18], we proposed a simulation framework for dataflow applications based on PREESM [19]. This framework uses SystemC as a simulation engine for co-design applications allowing to perform early functional simulation of the application model and to pave the way of implementation. The SystemC based simulation uses the EIDF model to simulate SDF and PSDF networks and will evolve to support additional models.

The enable-invoke dataflow (EIDF) model of computation was introduced in [20] as a superset of dataflow models for unified simulation. Each actor in the model is represented by an *enable* rule and an *invoke* action. The enable rule checks the actor firing rules against available tokens in the interface while the invoke action process inputs tokens to produce outputs tokens. The actor consumption/production rate can vary over-time and depends on tokens values. The execution model relies on a scheduler responsible for polling the enable port of every actors of the network and for triggering invoke methods accordingly. Most of the dataflows can be modeled and simulated in a unified framework using this model.

The SystemC simulator is generated from a graphical representation of the model and allows to evaluate application functional behavior. To simulate DPR at the atomic actor level, we propose to use the enable function of the actor to implement the availability model. Additionally, a *configure* port is added to the actor to trigger the configuration of its behavior (Fig. 5).

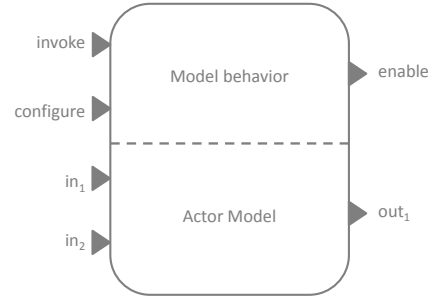


Fig. 5. Each DPR actor makes use of EIDF computation model

The current state of implementation does not allow to fully execute the SystemC generated code. When DPR will be fully implemented, PREESM will allow us to specify the actor network, and run functional simulation of the actor network with DPR. As for now, we do not target RTL code generation with PREESM framework.

VII. CASE STUDY

In this section we apply the DPR dataflow modeling to a signal processing example as shown in Fig. 6. The goal of this

application is to process pixels from a pixel source and extract their correlation with a given set of known image features. Seven actors are presented :

- *Image* : outputs every pixel data of an image of width W and height H .
- *Pixel_{source}* : emulates a camera that stream one pixel at a time.
- *Block_{5×5}* : takes one pixel and outputs its 5×5 neighborhood. The neighborhood is composed of pixels registered in the actor internal state.
- *Feat_{source}* : provides a given set of characteristic image descriptors.
- *Correl* : takes a 5×5 image block and computes a correlation score against *Feat_{source}* outputs.
- *Broadcast* : sends its input tokens to the N iterations of the *Correl* actor. Its purpose is to force the scheduling of the graph to a specified behavior.
- *Score* : stores every *Correl* score at the end of computation.

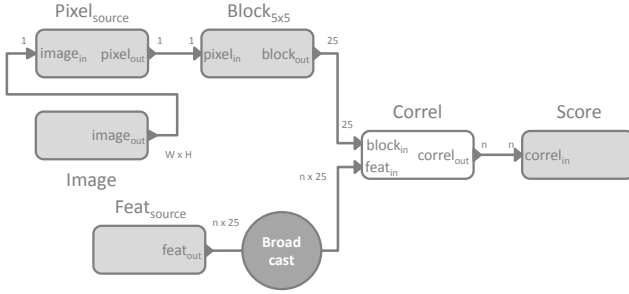


Fig. 6. Model of the case study correlation example composed of seven actors. The DPR actor *Correl* is in white while other actors are in gray. Only the *Broadcast* actor is shown as a darker gray circle

In this application, a number of correlated features for each frame is configured. To make sure the required number of features can be tracked, the correlation method needs to be adapted to consume less resources when a large number of features are tracked and can consume more resources to track less feature with greater quality. The correlation method used in this application ranges from binary robust independent elementary features (BRIEF) [21], sum of absolute differences (SAD) and zero-mean cross-correlation (ZCC). In this dataflow model, the correlation actor *Correl* is configured with an partial reconfiguration overhead attribute *ovhd* equal to 10000 time units and an availability attribute *avbl* of 0.66.

In this example the *Correl* actor can execute at 1.5 times the speed of the pixel source to tackle the unlimited buffer problem. Thus, if the correlation scheme gets modified for an new image, it will take 2 images before the buffer is fully consumed and a new reconfiguration can occur. This limit the reconfiguration period to a reconfiguration every 3 images at most. In our case we decided to fix the *avbl* at 0.66 meaning that we authorize a reconfiguration every 3 frames (actor is

unavailable during the reconfiguration time so 1 frame every 3 frames).

The *avbl* attribute defined here, means that our actor is available 66% of the time, so the reconfiguration will occur at most every three images. This schedule is pictured in Fig. 7.

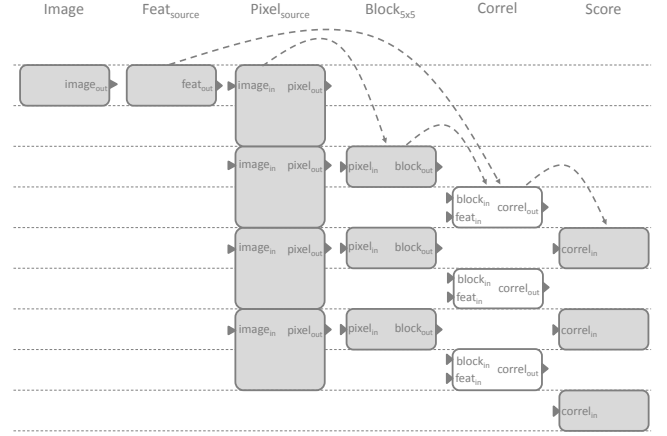


Fig. 7. Scheduling of the case study. Thick gray lines underline data dependencies between actors

Based on this model and its schedule, we first identify DPR properties of the graph. One aspect to evaluate is where to place every buffer to save tokens while the actor is being reconfigured. Directly buffering the *Correl* actor input requires a buffer of size $(100000 \times 25) = 2500K$ tokens. If we follow the incoming path of the *Correl* actor, we determine the buffering at the *Block_{5×5}* input to requires a buffer of size $((100000 \times 25)/25)/1 = 100K$ tokens. It greatly minimizes the buffering. The reconfiguration is limited to happen at a maximum frequency of once every two frames (modeled by the availability attribute) to ensure that buffer does not need to be extended. The frequency could be extracted from the graph and implementation attributes and is limited by the reconfigurable actor as it must process all the buffered data before a new reconfiguration. This depends on the actor implementation and its throughput, and the reconfiguration time overhead. In our case, for a *Correl* actor with a throughput equal to the pixel source throughput, reconfiguration would never happen with bounded memory. Automatic analysis of this maximum reconfiguration frequency could be performed at the scheduling step. Figure 8 shows the DPR schedule of the case study when the reconfiguration happens at the init phase ϕ_i .

VIII. CONCLUSION

This paper introduces an ongoing work to use dataflow modeling for application using DPR on FPGA. The proposed model allows to evaluate and optimize buffering for signal processing application dealing with streaming data source. While the proposal is not yet complete, it already allows to evaluate reconfiguration costs to determine the relevance of DPR for a application. Using SystemC, the reconfiguration can also be simulated to measure how DPR affect system performance and check the model functional behavior.

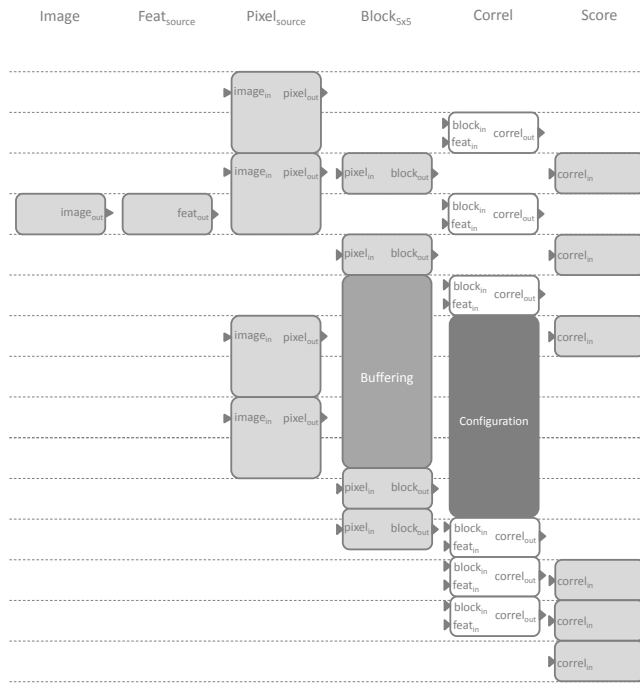


Fig. 8. Scheduling with DPR happening at ϕ_i

The proposed method will be applied to drive the design of a visual odometry system on a FPGA-based platform supporting reconfiguration. For this application, reconfiguration is a key feature to allow real-time performance while maintaining a good quality of result.

Future work on this topic will include improving SystemC code generation and automatic generation of DPR scheduling to produce code that can be both simulated and synthesized for and on FPGA technology. A knowledge of physical metrics such as resource, placement and routing overheads would help to indiscriminate or discriminate the usage of DPR in a given application context. More work will also be conducted on the model to allow the specification of more challenging applications : multiple node reconfiguration and clustering of reconfigurable actors.

REFERENCES

- [1] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: A survey," *J. VLSI Signal Process. Syst.*, vol. 28, no. 1/2, pp. 7–27, May 2001. [Online]. Available: <http://dx.doi.org/10.1023/A:1008155020711>
- [2] W. S. Carter, K. Duong, R. H. Freeman, H.-C. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo, and S. L. Sze, "A user programmable reconfigurable logic array," in *Proceedings of the IEEE Custom Integrated Circuits Conference*. IEEE, May 1986, pp. 233–235, first peer-review, public description of a commercial FPGA.
- [3] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Comput. Surv.*, vol. 34, no. 2, pp. 171–210, June 2002. [Online]. Available: <http://doi.acm.org/10.1145/508352.508353>
- [4] K. Pock, R. Tessier, and A. DeHon, "Birth and adolescence of reconfigurable computing: A survey of the first 20 years of field-programmable custom computing machines," in *Highlights of the First Twenty Years of the IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2013, pp. 3–19.

- [5] K. Bondalapati and V. K. Prasanna, "Dynamic precision management for loop computations on reconfigurable architectures," in *FCCM*. IEEE Computer Society, 1999, pp. 249–. [Online]. Available: <http://dblp.uni-trier.de/db/conf/fccm/fccm1999.html>
- [6] J. Liang, R. Tessier, and D. Goeckel, "A dynamically-reconfigurable, power-efficient turbo decoder," in *FCCM*. IEEE Computer Society, 2004, pp. 91–100. [Online]. Available: <http://dblp.uni-trier.de/db/conf/fccm/fccm2004.html>
- [7] R. Garcia, A. Gordon-Ross, and A. D. George, "Exploiting partially reconfigurable fpgas for situation-based reconfiguration in wireless sensor networks," *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, vol. 0, pp. 243–246, 2009.
- [8] J. Vidal, F. de Lamotte, G. Gogniat, J.-P. Diguët, and P. Soulard, "Uml design for dynamically reconfigurable multiprocessor embedded systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2010, pp. 1195–1200. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1870926.1871215>
- [9] S. Guillet, F. de Lamotte, N. L. Griguer, r. Rutten, G. Gogniat, and J.-P. Diguët, "Designing formal reconfiguration control using uml/marte," in *ReCoSoC*, L. S. Indrusiak, G. Gogniat, and N. S. Voros, Eds. IEEE, 2012, pp. 1–8. [Online]. Available: <http://dblp.uni-trier.de/db/conf/recosoc/ReCoSoC2012.html>
- [10] G. Kahn, "The semantics of simple language for parallel programming," in *IFIP Congress*, 1974, pp. 471–475. [Online]. Available: <http://dblp.uni-trier.de/db/conf/ifip/ifip74.html>
- [11] E. A. Lee and T. Parks, "Dataflow process networks," in *Proceedings of the IEEE*, 1995, pp. 773–799.
- [12] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. 36, no. 1, pp. 24–35, Jan. 1987. [Online]. Available: <http://dx.doi.org/10.1109/TC.1987.5009446>
- [13] B. Bhattacharya and S. Bhattacharyya, "Parameterized dataflow modeling for dsp systems," *Signal Processing, IEEE Transactions on*, vol. 49, no. 10, pp. 2408–2421, Oct 2001.
- [14] K. Desnos, M. Pelcat, J.-F. Nezan, S. Bhattacharyya, and S. Aridhi, "Pimm: Parameterized and interfaced dataflow meta-model for mpsoes runtime reconfiguration," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII)*, 2013 International Conference on, July 2013, pp. 41–48.
- [15] F. Duhem, F. Muller, and P. Lorenzini, "Farm: Fast reconfiguration manager for reducing reconfiguration time overhead on fpga," in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. Lecture Notes in Computer Science, A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, and T. El-Ghazawi, Eds. Springer Berlin Heidelberg, 2011, vol. 6578, pp. 253–260. [Online]. Available: <http://dx.doi.org/10.1007/978-3-642-19475-7-26>
- [16] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *SCIENCE*, vol. 220, no. 4598, pp. 671–680, 1983.
- [17] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.
- [18] J. Piat, D. Marquez-Gamez, and M. Devy, "Embedded vision-based slam: A model-driven approach," in *Design and Architectures for Signal and Image Processing (DASIP)*, 2013 Conference on, Oct 2013, pp. 284–289.
- [19] M. Pelcat, J. Piat, M. Wipliez, J. F. Nezan, and S. Aridhi, "An Open Framework for Rapid Prototyping of Signal Processing Applications," *EURASIP Journal on Embedded Systems*, vol. vol 2009, p. pp14, 2009. [Online]. Available: <http://hal.archives-ouvertes.fr/hal-00429312>
- [20] W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya, "Functional dif for rapid prototyping," in *In Proceedings of the International Symposium on Rapid System Prototyping*, 2008, pp. 17–23.
- [21] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ser. ECCV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 778–792. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1888089.1888148>