Hierarchical Memory Mapping During Synthesis in FPGA-Based Reconfigurable Computers *

Iyad Ouaiss and Ranga Vemuri
Digital Design Environments Lab, University of Cincinnati
Cincinnati, OH 45221-0030, USA
{iouaiss, ranga}@ececs.uc.edu

Abstract

One step in the synthesis for FPGA-based Reconfigurable Computers (RCs) involves mapping the design data structures onto the physical memory banks available in the hardware. The advent of Xilinx Virtex-style FPGAs and of hierarchical memory schemes on reconfigurable boards introduced an added complexity to this mapping. The new RC boards offer a wealth of memory banks many of them on-chip (such as the BlockRAMs available in the Virtex architecture) and many of them offering variable number of ports and several depth/width configurations. Along with the external RAMs, a hierarchy of memories with varying access performances are available in a reconfigurable computer. It becomes critical to perform a good mapping to achieve optimal design performance. This paper presents an automatic memory mapping methodology which takes into account: the number of words and word size of design data segments and physical memory banks, number of ports on the banks, access latency of the banks, proximity of the banks to the processing unit, life cycle analysis of data segments, and it also incorporates configuration selection from the multiple configurations available in BlockRAMs of Virtex series FPGAs. In the case of multiple processing elements on board, the paper also provides a framework in which the task of memory mapping interacts with spatial partitioning to provide the best implementation.

1 Introduction

When designing a circuit in digital hardware (ASIC or FPGA), the task usually consists of mapping operations of the application onto the hardware logic and mapping the data structures onto physical memory banks of the hardware.

A synthesis process typically takes the design from a level of abstraction to a lower one getting the design closer to a hardware implementation. While doing so, it is important to intelligently assign data structures to physical memory banks in order to minimize the overall cost of memory accesses. This is especially the case in today's data-intensive applications that require a very high-speed data access and large storage area. In image processing and speech recognition applications, for instance, the memory component of an ASIC implementation could contribute to more than 75% of the entire design area! Such a high area has a direct effect on memory performance.

Memory mapping has been researched in the case of ASICs where the mapping consists of selecting memory components from a library and/or selecting where the memory components are placed and the way in which they are connected to the hardware logic. This view assumes that the hardware is custom-built for the application: starting with an application, the synthesis process selects the best physical memory components and interconnects them to the logic hardware in the most efficient manner. In that case, the physical constraints consist of area requirements, number of available pins, etc., and the selection of memory components is limited by the richness of the module library.

In reconfigurable computing, the problem is a little different: starting with a fixed hardware platform where only logic and few interconnection switches are programmable, the synthesis process tries to map, in the best possible way, the application onto the RC. Thus, the number of memory banks, the type of memory banks, and, to a certain extent, the way they are connected to the processing units, are usually fixed. Before the introduction of Xilinx Virtex-style FPGAs, the mapping was limited to the physical banks that are external to the processing units; and since the number of external physical banks was typically low, designers would perform the mapping manually or use a very simple assignment algorithm to do so.

However, in the case where a large number of physical memory banks is available, a manual assignment is very dif-

*This work is supported in part by the US Air Force, Wright Laboratory, WPAFB, under contract number F33615-97-C-1043.

Configuration Number	Depth (# words)	Word Size (# bits/word)
1	4096	1
2	2048	2
3	1024	4
4	512	8
5	256	16

Table 1. Xilinx BlockRAM Configurations

ficult. Furthermore, the Virtex-style FPGAs introduced three new complexities to the mapping: First, there is potentially a large number of on-chip memory banks (*BlockRAMs* for Xilinx Virtex devices [22], *Embedded Array Blocks* for Altera FLEX 10K devices [2], and *Embedded System Blocks* for Altera APEX E devices [1]). In the Xilinx Virtex FPGAs, this number ranges from 8 BlockRAMs for the XCV-50 device to 208 BlockRAMs for the XCV-3200E device! Second, the number of memory ports for each on-chip memory bank could be greater than one; two ports for every Xilinx Virtex BlockRAM, each port accessing the same physical space. And third, the depth/width ratio of each memory bank could be variable; for each Virtex BlockRAM, the five configurations are shown in Table 1. Taking into account these features, and adding the external memory banks to the on-chip banks, the problem of logical-to-physical memory mapping becomes quite complicated.

Little work has been done to automatically assign data structures to complex RC systems; furthermore, features of on-chip Virtex-style FPGAs have not yet been incorporated in the synthesis process. This paper exposes the different features of a complex, hierarchical hardware memory structure and evaluates the integer-linear programming approach to memory mapping.

The rest of this paper is organized as follows: Section 2 discusses previous work performed in memory mapping. Section 3 describes the inputs to the environment in which memory mapping is performed. Section 4 depicts the Integer Linear Programming approach for memory mapping. Section 5 shows some results obtained and a discussion of the approach. Finally, Section 6 concludes the paper by presenting ongoing work.

2 Previous Work

There exist several studies on utilizing memory modules in data path synthesis. As mentioned in Section 1, the majority of the memory mapping studies focuses on the ASIC implementation. The tools pick a set of physical memory modules from a library of available banks and select the interconnection structure to connect the processing units to the memory banks.

Several studies were conducted in the realm of high-level synthesis where cliques of the design variables are partitioned to form data segments. Some researchers performed this task without taking into consideration the interconnection structure of the hardware [3], while others consider the cost of interconnection during variable grouping for multi-port memory banks [19].

An integer linear programming model is used in [13, 8] to group registers and form multi-port memory modules. Instead of dealing with fixed hardware configurations, these studies construct the hardware based on minimizing the needs of memory banks and minimizing the cost of interconnection.

In the above references, interconnection cost is one of the most important constraints, and the research aimed at minimizing this cost as well as minimizing the number of physical memory banks . With on-chip memory and fixed external memory banks and interconnection structures, the problem becomes different. On-chip memory does not require off-chip communication thus reducing external interconnections. Furthermore, given a fixed memory structure on an RC board, minimizing the number of required memory banks might not be the best mapping solution; as long as the mapper does not exceed the physical storage area, it should be allowed to use as many banks as it sees fit.

It is worth mentioning that, contrary to the older works mentioned above, FPGA on-chip memory banks are targeted in [12, 18]. However, the banks are used for implementing logic. These studies address technology mapping of logic onto on-chip memories, whereas the storage capability of the banks is not considered.

In [17], memory mapping for FPGAs with on-chip memories is addressed; however, only single-ported memory banks are assumed. The same technique was improved in [21] so that the mapping caters to recent FPGAs containing dual-ported on-chip banks. In both works, the focus is on hardware containing a single type of memory banks (either single or dual ported).

Given data structures and access constraints to these structures, [5] finds a legal packing of the logical segments into the physical segments while minimizing the area. Again, since the storage area in the RC framework is fixed, it might not be beneficial to minimize the area. On the other hand, instead of meeting access constraints to the data structures, our work targets on minimizing all memory accesses. The work can be extended in the event hard access constraints must be met on some data structures.

As classified in [15], the problem of *logical-to-physical memory mapping* [17] can be divided into two steps: Firstly, translating the storage requirements onto *logical memories*; i.e. forming the data structures needed by the design. Secondly, mapping the logical memories onto the *physical memories* of the hardware; i.e. assigning the data structures to memory banks. In [15], the mapping of logical memories

is onto physical memories chosen from a library; however, in this paper, the mapping is onto a fixed architecture dictated by the RC board.

An analysis of several memory mapping studies is presented in [15]. The authors compare the techniques based on the number and the type of logical memories and physical banks considered simultaneously. In addition, the book by Catthoor et al. [6] and the book by Panda, Dutt, and Nicolau [16] provide an excellent source for topics in memory system optimization, exploration, and management.

3 Problem Formulation

Several features exist for different types of RC boards. This section generalizes the approach of memory mapping by targeting a flexible hierarchical memory structure.

A memory mapper must take as an input both the architecture of the target RC board as well as a description of the design to be synthesized and mapped onto the board. For this paper, it is assumed that the RC board contains only one processing unit. As part of future enhancement, this work will be extended to multi-processing units where logic placement and pin constraints during routing will be addressed.

3.1 Architecture Description

The RC board architecture is described by a collection of *memory types*. There could be several instances of each memory type, but all instances share the same storage and access speed specifications, and share the same proximity and ease of access from the processing unit.

For each memory type, a *number of instances* tells the mapper how many instances of each type exist on the board. The *number of ports* of a type is one if the memory is a single-ported memory, two if dual-ported, etc. As shown in Figure 1, the depth/width ratio of a memory could be variable; the *number of configurations* for each type is the number of possible settings of each port of that type.

The *number of words* (depth) and the *number of bits per word* (width) of a type are unique numbers if only one configuration exists. Otherwise, these are equal-length lists of numbers describing the possible configurations of the depth/width ratio. Entry i of the depth list together with entry i of the width list correspond to configuration i. It is assumed that the capacity of each configuration is a constant; i.e.

$$\forall i \in configurations:$$

$$depth(i) * width(i) = capacity(memory) \qquad (1)$$

The access latency for each type of memory is variable; the *read latency* is the number of clock cycles required after performing a read and before getting valid data out of the memory bank. Similarly, the *write latency* is the number of clock cycles required after performing a write operation and before the data is correctly stored in the memory bank.

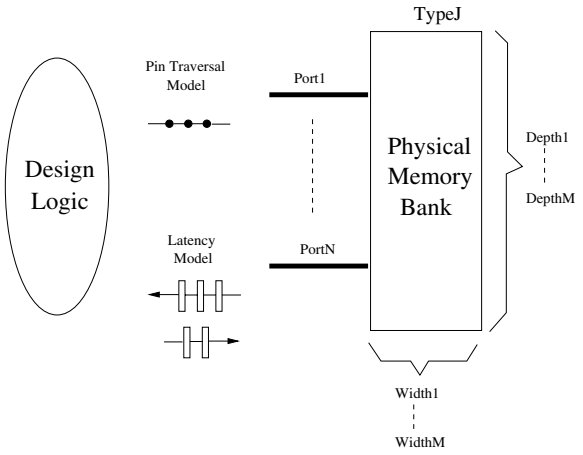

Figure 1. Generic Memory Bank

Finally, with respect to the physical location of the memory bank, the *number of pins traversed* depicts the proximity of the physical memory bank to the processing unit. If a bank is on-chip, zero pins are traversed. If an off-chip bank is directly connected to the memory, two pins are traversed. If an indirect connection exist between the processing unit and the external memory bank, then additional pins are traversed. In general, the aim is to map data structures to physical banks that are as close as possible to the processing unit; the further away they are, the larger the impact on the overall memory access performance.

A generic physical bank is shown in Figure 1. The latency model captures the number of read and write clock delays and the pin traversal model captures the number of pins traversed between the processing unit and the memory bank. In the Figure, bank type J is an N-ported memory, has M different configurations: depth*1*/width*1*, depth*2*/width*2*, ..., depth*M*/width*M*.

3.2 Task graph Description

On the design side, a description of the data structures is required. Since this work focuses on placing the data structures on the physical banks, it is assumed that the structures are already formed.

For each data segment in the design, the *number of words* (depth) in the segment and the *number of bits per word* (width) are required. A footprint analysis of the memory accesses could tremendously help in guiding the mapping process: e.g. data segments that are extensively accessed should be assigned to faster and closer physical banks. In this paper, it is assumed that the footprint analysis is not available. Instead, frequency of accesses is based on the

size of each data segment. The assumption that a large data segment would be accessed more frequently than a smaller segment is not very accurate; however, it is used as a rough metric in the mapping process.

3.3 Conflict Description

During synthesis of a design, scheduling determines the *life times* [7, 4] of the variables and data structures. This life cycle analysis could further improve the memory mapping since segments that can overlap could be placed in the same storage area, thus decreasing the total storage requirement. For this purpose, the mapper needs to know which data segments life cycles overlap. A set of *conflicting pairs* captures this requirement; pair (L1, L2) means that data segment L1 cannot share storage space with segment L2.

Note that on one extreme, if no conflicting pairs are given, all segments could be ideally mapped to the same physical bank. On the other end of the spectrum, if all conflicting pairs exist, no overlapping of storage space can take place.

4 ILP Formulation

A 0-1 Integer Linear Programming (ILP) model is presented in this work. This approach yields an optimal solution to the mapping problem, however, it typically takes a very long time to execute and might not finish execution within reasonable time for larger problems.

The advantage of using this approach is that it exposes the constraints and objectives of the problem and pinpoints areas that introduce added complexities to the model. The ILP method is used here as a backbone solution to the problem where an interaction with heuristical approaches would result with fast, low-cost solutions.

It should be noted that linearization techniques are used at times when non-linear constraints arise. A survey of linearization approaches can be found in [14].

For the formulation presented in this section, we assume the following notation. There are M data structures:

$$DS = \{DS_1, DS_2, \ldots, DS_M\}$$

to be mapped onto N different types of physical memory banks:

$$PB = \{PB_1, PB_2, \ldots, PB_N\}$$

Note that N is not the number of available banks on the RC system. There could be multiple instances of each type of memory bank.

For each logical data structure d, we have:

$$\begin{cases} D_d & \text{Number of words in segment } d. \\ W_d & \text{Number of bits per word in segment } d. \end{cases}$$

For each type of physical memory bank t, we have:

$$\begin{cases} I_t & \text{Number of banks of type } t. \\ P_t & \text{Number of ports in a bank of type } t. \\ C_t & \text{Number of depth/width configurations in a bank of type } t. \\ D_t & \text{Array of number of words in a bank of type } t. \\ W_t & \text{Array of number of bits per word in a bank of type } t. \\ RL_t & \text{Read latency in number of clock cycles.} \\ WL_t & \text{Write latency in number of clock cycles.} \\ T_t & \text{Number of pins traversed from the processing unit to a bank of type } t. \end{cases}$$

where the depth/width ratio variables are:

$D_t = \{d_1, d_2, \ldots, d_{C_t}\}$ and: $W_t = \{w_1, w_2, \ldots, w_{C_t}\}$

Finally, there are Q conflict pairs in the design where each associates two logical structures: (DS_x, DS_y) where: $x \neq y$.

The following are some simplifying assumptions made in this paper; they can be easily alleviated by adding new constraints and/or new modeling variables to the formulation:

- All ports of the physical banks are assumed to be read/write ports. Overlooking capacity constraints, any data structure could be mapped to any port of a physical bank.
- If a multi-ported physical bank has several depth/word configurations, it is assumed that each port can have its own configuration setting. This is in agreement with the Virtex BlockRAM architecture. Note that forcing all ports to have the same configuration setting, simplifies the ILP formulation.
- A data structure that is too deep or two wide to fit on any single physical bank will be mapped onto more than one bank. The mapper should select banks having the same type in order to preserve similar access latency: an access to any word in a data structure has the same latency irrespective of the location of the word within the structure.

Finally, the remaining notations pertain to the 0-1 variables used in the model. Z_{dt} associates a data structure to a memory type:

$$Z_{dt} = \begin{cases} 1 & \text{if data structure } d \text{ is assigned to some instance of bank type } t. \\ 0 & \text{otherwise.} \end{cases}$$

Z_{dt} is used to force an oversized data structure to be split across banks of the *same type*. Similarly, X_{dtip} associates a

data structure to a specific physical bank:

$$X_{dtip} = \begin{cases} 1 & \text{if data structure } d \text{ is assigned to port } p \\ & \text{of instance } i \text{ of bank type } t. \\ 0 & \text{otherwise.} \end{cases}$$

And, only for multi-configuration physical banks (i.e. $C_t > 1$), Y_{tipc} sets a specific configuration to a port of a memory bank:

$$Y_{tipc} = \begin{cases} 1 & \text{if configuration } c \text{ is selected for port } p \\ & \text{of instance } i \text{ of bank type } t. \\ 0 & \text{otherwise.} \end{cases}$$

4.1 Constraint Formulation

The following are some of the important constraints in the memory mapping problem.

- **Uniqueness constraints:** Each data structure should be mapped to exactly one *type* of physical bank:

$$\forall d \in DS, \quad \sum_{t \in PB} Z_{dt} = 1$$

- **Port constraints:** Each data structure should be mapped to at most one port of an instance:

$$\forall d \in DS, \quad \forall t \in PB \quad \forall i, 1 \leq i \leq I_t : \quad \sum_{p=1}^{P_t} X_{dtip} \leq 1$$

- **Capacity constraints:** Each data structure should be fully stored on the hardware. In the case of physical banks with *fixed configuration*:

$$\forall d \in DS, \forall t \in PB,$$

$$\sum_{i=1}^{I_t} \sum_{p=1}^{P_t} X_{dtip} = \left\lceil \frac{D_d}{D_t[1]} \right\rceil * \left\lceil \frac{W_d}{W_t[1]} \right\rceil * Z_{dt}$$

where the ceiling operator refers to the closest, greater or equal to the power of two. This equation is based on the assumption of Equation 1. In other words,

$$\forall i, \ , \forall j, \ 1 \leq i, j \leq C_t : \quad D_t[i] * W_t[i] = D_t[j] * W_t[j]$$

For physical banks with multiple configurations, the formulation is more complex. First, each data structure must fit in the storage area assigned to it:

$$\forall d \in DS,$$

$$\lceil D_d \rceil * \lceil W_d \rceil \leq \sum_{t \in PB} \sum_{i=1}^{I_t} \sum_{p=1}^{P_t} X_{dtip} * D_t[1] * W_t[1]$$

Second, the depth of each segment must be less than the combined depth of the assigned banks:

$$\forall d \in DS, \quad D_d \leq \sum_{t \in PB} \sum_{i=1}^{I_t} \sum_{p=1}^{P_t} \sum_{c=1}^{C_t} D_t[c] * Y_{tipc}$$

Third, the width of each segment must be less than the combined width of the assigned banks:

$$\forall d \in DS, \quad W_d \leq \sum_{t \in PB} \sum_{i=1}^{I_t} \sum_{p=1}^{P_t} \sum_{c=1}^{C_t} W_t[c] * Y_{tipc}$$

Finally, by keeping the last two formulations linear, the following constraint must be added:

$$\forall d \in DS, \quad \forall t \in PB, \quad \forall i, 1 \leq i \leq I_t,$$

$$\forall p, 1 \leq p \leq P_t, \quad \forall c, 1 \leq c \leq C_t : \quad Y_{tipc} \leq X_{dtip}$$

- **Lifecycle conflict constraints:** When there is a conflict between two data structures, they cannot be mapped onto the same port of a physical bank:

$$\forall t \in PB, \quad \forall i, 1 \leq i \leq I_t, \quad \forall p, 1 \leq p \leq P_t,$$

$$\forall q, 1 \leq q \leq Q : \quad X_{q[1]tip} + X_{q[2]tip} \leq 1$$

Note that if all data structures conflict with each other, the formulation can be simplified by condensing the above into a single constraint:

$$\forall t \in PB, \quad \forall i, 1 \leq i \leq I_t, \quad \forall p, 1 \leq p \leq P_t :$$

$$\sum_{d \in DS} X_{dtip} \leq 1$$

- **Configuration uniqueness constraints:** In multi-configuration banks, at most one configuration can be selected at a time:

$$\forall t \in PB, \quad \forall i, 1 \leq i \leq I_t, \quad \forall p, 1 \leq p \leq P_t :$$

$$\sum_{c=1}^{C_t} Y_{tipc} \leq 1$$

- **Physical storage constraints:** In multi-ported banks, the sum of the storage space mapped to every port must fit within the bank. For single configuration banks, the constraint can be formulated as:

$$\forall t \in PB, \quad \forall i, 1 \leq i \leq I_t :$$

$$\sum_{p=1}^{P_t} max(\sum_{d \in DS} X_{dtip} * \lceil D_d \rceil) \leq D_t$$

And for multi-configuration banks, the constraint becomes:

$$\forall t \in PB, \quad \forall i, 1 \leq i \leq I_t :$$

$$\sum_{p=1}^{P_t} max(\sum_{d \in DS} X_{dtip} * \lceil D_d \rceil * \lceil W_d \rceil) \leq D_t * W_t$$

In the above two equations, the $max()$ operator can be dropped if the design assumes that all data structures conflict with each others (i.e. no space overlapping). This makes the formulation much simpler since it avoids any non-linearity in the model.

Note that when the number of physical banks is very large, the model might not need to consider all available banks. An upper bound on this number, for each type of physical banks, can be used instead. Also, several other simplifications could be made in order to speed the execution while still producing good results.

4.2 Objective Formulation

The objective of the ILP model is to optimize the performance and minimize the interconnection cost of the memory assignment. The cost function takes the form:

$$minimize[Cost_1 * \alpha_1 + Cost_2 * \alpha_2 + \ldots + Cost_n * \alpha_n]$$

where α_i is a weight coefficient used to normalize $Cost_i$ with respect to all other cost components.

The objective formulation in an ILP model is compact and allows easy expansions. Three cost components are depicted below. They try to cover the speed performance and the pin limitation constraints while assuming that the depth of a data segment is proportional to the number of times the segment is accessed:

- **Latency cost:** Assuming the number of reads is equal to the number of writes for every data structure:

$$\sum_{d \in DS} \sum_{t \in PB} Z_{dt} * D_d * [RL_t + WL_t]$$

- **Pin delay cost:** Assuming the number of pins traversed from the processing unit to reach the memory bank is inversely proportional to the clock speed:

$$\sum_{d \in DS} \sum_{t \in PB} Z_{dt} * D_d * T_t$$

- **Pin I/O cost:** The larger the width of a data structure the more pins it will need in the event of off-chip physical banks:

$$\sum_{d \in DS} \sum_{t \in PB} Z_{dt} * W_d * T_t$$

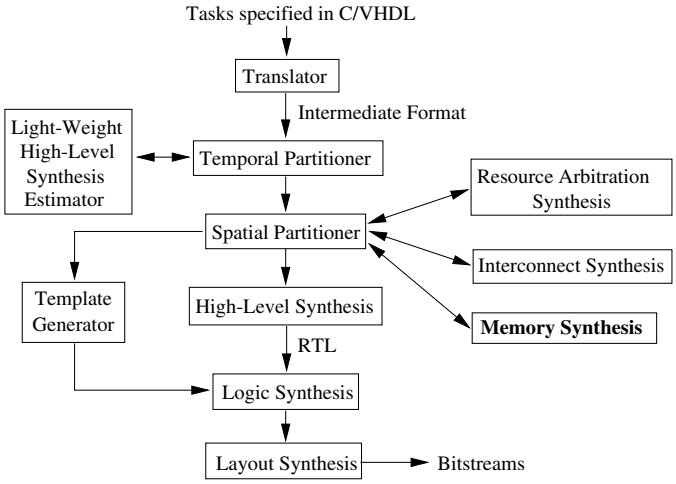

Figure 2. Interaction with the Synthesis Flow

5 Results and Discussion

Figure 2 shows the interaction of the memory mapper/synthesis tool with an existing synthesis and partitioning tool for multi-FPGA RC boards [10]. In this paper, since a single processing unit is assumed, the interaction with the memory synthesis tool can take place during temporal partitioning or during high-level synthesis.

The ILP model presented in the previous section was executed for designs of different sizes. CPLEX, a commercial linear programming solver [11], was used. Both the number and types of logical segments as well as physical banks were varied. Given the constraints and objective functions, the quality of the mapping produced was optimal. Table 2 shows the execution time on a SUN Ultra-30 (248MHz with 128MB RAM) for designs of various sizes: For logical memories, the number of segments represents the main complexity parameter in the ILP formulation. Similarly, for physical memories, the three complexity parameters are: the total number of physical banks, the total number of ports summed over all instances of all bank types, and the total number of possible configuration settings summed over all multi-configuration ports of all bank types. The execution time is given in seconds.

It is clear that for large designs, the ILP formulation becomes impractical. However, based on the current RC technology and current applications, the ILP execution is satisfactory in several cases. It can also be integrated with heuristical approaches to limit the search space. By choosing proper heuristics, the quality of the final mapping could be preserved to a large extent while gaining execution speed. The ILP formulation used to obtain the results in Table 2 contains all constraints stated in this paper; it did, however, assume that all data structures had conflicting life

Logical Memories	Physical Memories			Execution Time
#segments	Total #banks	Total #ports	Total #configs	(in seconds)
22	13	25	50	1.03
32	23	45	100	4.84
32	45	77	150	13.21
42	45	77	150	20.00
32	23	45	100	9.42
32	65	105	150	28.57
32	180	265	375	114.37
72	65	105	150	71.07
72	180	265	375	313.65
132	180	265	375	833.42

Table 2. ILP Execution Times

cycles, thus no overlapping of memory segments. A few additional constraints were also introduced and the formulation of some constraints was slightly modified to enhance the ILP execution speed.

Note the following:

- The number of data structures can be larger than the total number of available physical banks. There are several ways of coping with this problem. First, if a physical bank has more than one port, then the mapper could assign more than one logical segment to this bank (one logical segment per port). Second, if there are no life cycle conflicts between two or more logical segments, then they could be mapped to the same port of a bank. Third, two or more conflicting data segments could be mapped to the same port of a physical bank provided an arbitration mechanism is introduced (See Section 6). The mapper decides to arbitrate segments based on the added area and delay estimates due to arbitration.

- A footprint analysis or software profiling information depicting the number of memory accesses is not available to the mapper. As a rough guide, the number of words in each data structure was assumed to be proportional to the frequency of accesses. A software profiling approach could lead to a lower cost implementation since the mapper would have knowledge of the number of memory accesses for every data structure.

- The current work did not take into account constraints posed by the data structures. In other words, there was no way of specifying a performance constraint on specific segments. At the cost of added complexity, the ILP formulation could handle such constraints.

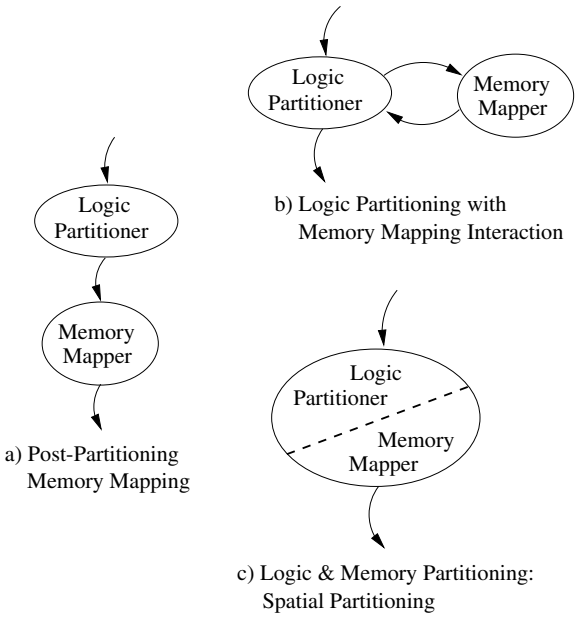

Figure 3. Partitioning and Memory Mapping

6 Ongoing Work

In a RC system containing multile processing units, logic partitioning is the task of mapping computational operations to the processing units. In this case, the memory mapping process can occur at different stages of synthesis. Figure 3 shows three ways of incorporating memory mapping with logic partitioning:

- Figure 3a depicts a *post-partitioning* scheme where memory mapping occurs after the logic partitioning is complete. The logic partitioner assumes that at least one memory mapping is possible but does not take into account how and where data structures are placed on the board.

- Figure 3b shows an *interaction* between the logic partitioner and the memory mapper. In this scenario, the logic partitioner contemplates logic partitions while taking into account possible memory assignments. Depending on how many times the logic partitioner invokes the memory mapper, a *lighter weight* memory mapper technique could be implemented in order to speed the interaction. A lighter weight mapper could be one with fewer, or more relaxed, constraints.

- In Figure 3c, the processes of logic partitioning and of memory mapping are merged. The resultant spatial partitioning engine considers both logic and data structures at the same time. Both problems are tackled as part of the formulation or of the heuristic algorithm.

The three views are currently being assessed. In this paper, since a single processing unit was used, the three views are equivalent to the post-partitioning approach where memory mapping occurs after logic partitioning.

As part of ongoing and future work, the following issues are being considered:

- In the case of a single processing unit, all design logic is mapped onto one hardware area, and all logic areas are assumed equidistant from each physical bank. The model need to be enhanced to support multiple processing units.
- The interconnection structure connecting the processing unit to the physical banks is fixed. In practice, some RC boards might offer limited programmability. New constraints should be added to support programmable interconnect. Insight to the interconnection synthesis problem can be found in [20].
- Arbitration is not taken into consideration in this paper. In other words, if two logical segments conflict, they will be mapped on two different ports. Support for arbitration could be based on the arbitration scheme introduced in [9]. This scheme interacts with the synthesis process to provide performance estimates due to arbitration and introduce arbiter tasks in the design.
- It is seen that heuristic approaches could be used to prune the design space before or while running the ILP model. Such approaches include limiting the number of available physical banks and pre-processing of the input logical segments.
- Finally, an interfacing mechanism between the memory mapper and the synthesis tools will allow the design space exploration phase to take into account different memory assignments.

References

[1] Altera Corporation. "APEX 20K Programmable Logic Device Family Data Sheet", March 2000.

[2] Altera Corporation. "FLEX 10K Embedded Programmable Logic Family Data Sheet", May 2000.

[3] C. J. Tseng and D. Siewiorek. "Automated Synthesis of Data Paths in Digital Systems". In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 5, pages 379–395, July 1986.

[4] D. D. Gajski, N. D. Dutt, A. C. Wu, and S. Y. Lin. *"High-Level Synthesis, Introduction to Chip and System Design"*. Kluwer, 1992.

[5] D. Karchmer and J. Rose. "Definition and Solution of the Memory Packing Problem for Field-Programmable Systems". In *Proceedings of International Conference on Computer Aided Design*, pages 20–26. ACM Press, November 1994.

[6] F. Catthoor, et al. *"Custom Memory Management Methodology"*. Kluwer, 1998.

[7] G. De Micheli. *"Synthesis and Optimization of Digital Circuits"*. McGraw-Hill, 1994.

[8] I. Ahmad and C. Y. Chen. "Post-Process for Data Path Synthesis". In *Proceedings of International Conference on Computer Aided Design*, pages 276–279. ACM Press, 1991.

[9] I. Ouaiss and R. Vemuri. "Efficient Resource Arbitration in Reconfigurable Computing Environments". In *Proceedings of Design Automation and Test in Europe*, pages 560–566. IEEE Computer Society Press, April 2000.

[10] I. Ouaiss, S. Govindarajan, V. Srinivasan, M. Kaul, and R. Vemuri. "An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures". In *Proceedings of the 5th Reconfigurable Architectures Workshop*, pages 31–36. Springer, March 1998.

[11] ILOG Incorporation. *"Using the CPLEX Callable Library"*. http://www.cplex.com.

[12] J. Cong and K. Yan. "Synthesis for FPGAs with Embedded Memory Blocks". In *Proceedings of International Symposium on Field Programmable Gate Arrays*, pages 75–81. ACM press, February 2000.

[13] M. Balakrishnan, et al. "Allocation of Multiport Memories in Data Path Synthesis". In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 7, pages 536–540, April 1988.

[14] P. Hansen, B. Jaumard, and V. Mathon. "Constrained Nonlinear 0-1 Programming". In *ORSA Journal of Computing*, volume 5, pages 97–119, 1993.

[15] P. Jha and N. Dutt. "High-Level Library Mapping for Memories". In *ACM Transactions on Design Automation of Electronic Systems*, pages 566–603. ACM Press, July 2000.

[16] P. R. Panda, N. Dutt, A. Nicalau. *"Memory Issues In Embedded Systems-On-Chip"*. Kluwer, 1999.

[17] S. Wilton. *"Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory"*. PhD thesis, University of Toronto, 1997.

[18] S. Wilton. "Heterogeneous Technology Mapping for FPGAs with Dual-Port Embedded Memory Arrays". In *Proceedings of International Symposium on Field Programmable Gate Arrays*, pages 67–74. ACM press, February 2000.

[19] T. Kim and C. L. Liu. "Utilization of Multiport Memories in Data Path Synthesis". In *Proceedings of the 30th Design Automation Conference*, pages 298–302. ACM Press, June 1993.

[20] V. Srinivasan, S. Radhakrishnan, R. Vemuri, and J. Walrath. "Interconnect Synthesis for Reconfigurable Multi-FPGA Architectures". In *Proceedings of the 6th Reconfigurable Architectures Workshop*, pages 597–605. Springer, April 1999.

[21] W. Ho and S. Wilton. "Logical-to-Physical Memory Mapping for FPGAs with Dual-Port Embedded Arrays". In *Proceedings of International Workshop on Field-Programmable Logic and Applications*, pages 111–123. Springer, September 1999.

[22] Xilinx, Inc. "Virtex 2.5V Field Programmable Gate Arrays", September 2000.