Definition and Solution of the Memory Packing Problem for Field-Programmable Systems

David Karchmer and Jonathan Rose
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario. Canada M5S 1A4

Abstract

This paper defines a new optimization problem that arises in the use of a Field-Programmable System (FPS). An FPS consists of a set of Field-Programmable Gate Arrays and memories, and is used both for emulation of ASICs and computation. In both cases the application circuits will include a set of memories which may not match the number and aspect ratio of the physical memories available on the FPS. This can often require that the physical memories be time-multiplexed to implement the required memories, in a circuit we call a memory organizer.

We give a precise definition of the packing optimization problem and present an algorithm for its solution. The algorithm has been implemented in a CAD tool that automatically produces a memory organizer circuit ready for synthesis by a commercial FPGA tool set.

1 Introduction

Field-Programmable Gate Arrays (FPGAs) are now widely used for the instant manufacturing of digital designs. Current FPGAs, however, do not have sufficient logic capacity for large designs, and so a system with multiple FPGAs is often needed. Such a *Field-Programmable System* (FPS) consists of FPGAs, some amount of memory and some type of programmable inter-chip connection mechanism. The term Field-Programmable System encompasses both FPGA-based compute engines [1, 2] and digital emulation systems [3, 4].

Memory is an essential part of almost any digital system, and so forms a key element of a Field-Programmable System. Each application, however, has vastly different memory needs. A telecommunication circuit, for example, may require many small queuing buffers, while a graphics engine will require a few larger frame buffers.

The subject of this paper is a method to efficiently map an application circuit's required set of memories into the available physical memories on an FPS. The context is a *low-cost* FPS in which there are a small number of pre-fabricated physical memories, but a large number of desired memories. The physical memories must be time-multiplexed to create the desired memories (*i.e.* each desired memory will have a different time slot in which it can access a separate portion of the physical memory).

The mapping becomes an optimization problem when the number of required memories exceeds the number of physical memories. There may be many different ways that the required (logical) memories can be packed into the physical memories. If the logical memories have different access time requirements, the optimization problem is to find a packing that meets the timing requirements. Furthermore, different packings will require different amounts of multiplexing and control, and so it is desirable to minimize the area devoted to this part of the circuit.

For example, consider an application in which there are five logical memories: There are three $3k \times 8$ memories, each with a required access time of $80ns$, one $4k \times 7$ memory with a required access time of $20ns$ and one $32k \times 8$ memory with a required access time of $100ns$. Assume that the FPS has three $32k \times 8$ physical memories, each with an access time of $20ns$. Since the logical memories are time-multiplexed, their final access time is approximately equal to the physical memory's nominal access time (in this case $20ns$) multiplied by the number of memories sharing the physical memory.

Figure 1a illustrates a packing in which all the $3k \times 8$ and $4k \times 7$ memories have a final access time of $40ns$ or less. This implementation does not meet the access time requirement of the $4k \times 7$ memory. Figure 1b

Permission to copy without fee all or part of this material is granted, provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1994 ACM 0-89791-690-5/94/0011/0020 $3.50

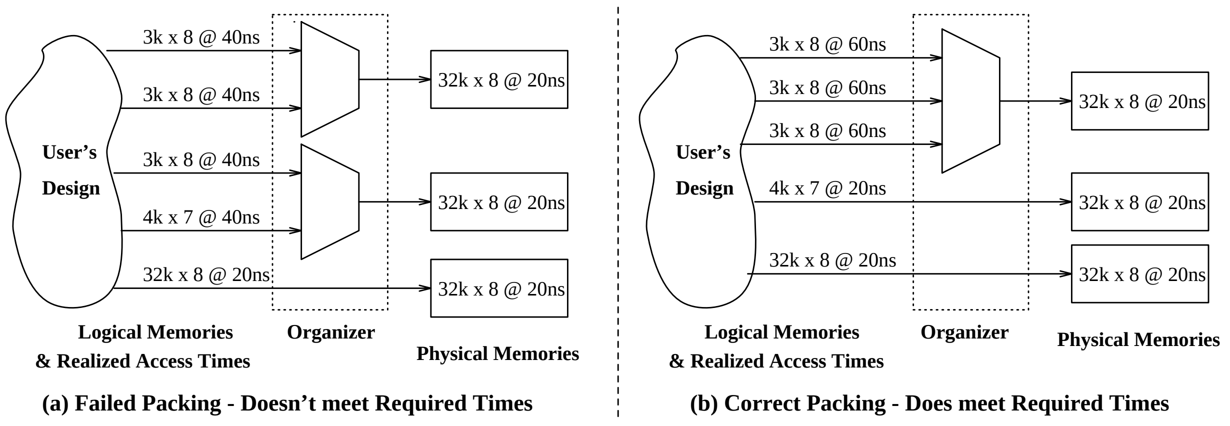

Figure 1: Possible Packing Solutions

shows the only possible packing that does meet all the access time constraints. The three $3k \times 8$ memories have a final access time of $60ns$ and both the $4k \times 7$ and the $32k \times 8$ memories have access time of $20ns$. Note that in some cases, a logical memory may be larger than an individual physical memory, and so will have to be partitioned into smaller pieces before it can be packed.

The multiplexer and controller that implements the packing will be referred to as a *Memory Organizer*. In a Field-Programmable System this circuit would be implemented using the FPGAs. This paper presents an algorithm to solve the optimization problem and describes a CAD tool which uses this algorithm to generate the memory organizer netlist. This tool has been used on an operational Field-Programmable System [5, 6].

This paper is organized as follows: Section 2 provides a precise definition of the optimization problem to be solved. Section 3 describes the memory organizer architecture, and the models needed to estimate the organizer speed and area. Section 4 gives an algorithm for the solution of the optimization problem. Section 5 presents several examples of the use of the CAD tool based on this algorithm.

2 Problem Definition

In this section we first describe our notation and then give a precise definition of the Field-Reconfigurable Memory packing problem.

There are three key parameters that characterize any memory: The *width* is the number of bits per memory word, the *depth* is the number of words in the memory, and the *access time* is the minimum amount of time that must pass between two consecutive memory access requests.

We will refer to each memory required by the user as a *Logical Memory*, abbreviated as LM. Each fixed available memory on the FPS is referred to as *Physical Memory* (PM). There are n logical memories in the set $L = \{LM_0, \ldots, LM_{n-1}\}$ and m physical memories in the set $P = \{PM_0, \ldots, PM_{m-1}\}$. The width, depth and access time of the j^{th} PM are denoted by Pw_j, Pd_j and Pt_j, and the width, depth and required access time of the i^{th} LM are denoted by Lw_i, Ld_i and Lt_i. If a logical memory is larger in depth or width than the physical memory, then it must be partitioned into pieces. Section 2.1 shows how this partitioning is done.

A *packing* is a partition of the LMs together with a mapping from each of the partitioned LMs to one of the physical memories.

The *occupancy* of a physical memory in a packing is the number of logical memories that are mapped into the physical memory. We denote the occupancy of PM_j by OC_j. For example, the occupancy of the topmost physical memory in Figure 1b is three.

A *legal packing* is a packing which meets the access times required by each of the logical memories. The access time of a realized logical memory is a function of the occupancy of the physical memory in which it resides, and of the delay introduced by the organizer multiplexer and control. If we denote the *organizer delay* of logical memory LM_i by OD_i, then a packing is legal if the following holds for all logical memories LM_i packed into physical memory PM_j:

$$Lt_i \leq OC_j \cdot Pt_j + OD_i \qquad (1)$$

If a logical memory is partitioned into smaller pieces in order to fit into a physical memory, then this access time requirement is also placed on the individual pieces.

The *area* of the organizer is the amount of logic used in the FPGAs to implement the multiplexing hardware and control. It depends on the number and size of the multiplexers in the organizer, which in turn depends on the occupancy of each physical memory, and the width and depth of the logical memories.

With this notation, we can state a general version of the **Field-Reconfigurable Memory packing problem:**

Given: The sets $L = \{LM_0, \ldots, LM_{n-1}\}$ and $P = \{PM_0, \ldots, PM_{m-1}\}$ and associated width, depth and access time parameters.

Find: A legal packing of L into P, which minimizes the area of the organizer.

2.1 Practical Issues and Assumptions

To both create a practical organizer and to simplify the description in this paper, we make the following assumptions about the logical and physical memories:

1. **Homogenous Physical Memories**

   In the above definition each physical memory is allowed to be different. For simplicity we will assume all physical memories are identical. The notation for the width, depth and access time of all physical memories thus simplifies to Pw, Pd and Pt.

2. **Prevention of Word Sharing**

   Word sharing occurs when two or more logical memories share the same word in a physical memory. We will not allow this because it would cause a memory *write* to become significantly slower and more complex — since part of a word has to be preserved, each write must be preceded by a read to record the un-altered portion of the physical memory's word. Note that this assumption implies that more of the physical memory will be wasted.

3. **Depth Restricted to be a Power of Two**

   If the depth of each logical memory is anything other than a power of two, the addressing circuitry connecting the logical address to the physical memory address bus will require physical adders to create the correct offsets into the physical memory space. Since adders are expensive in space and time, we restrict all logical memory's depths to a power of two.

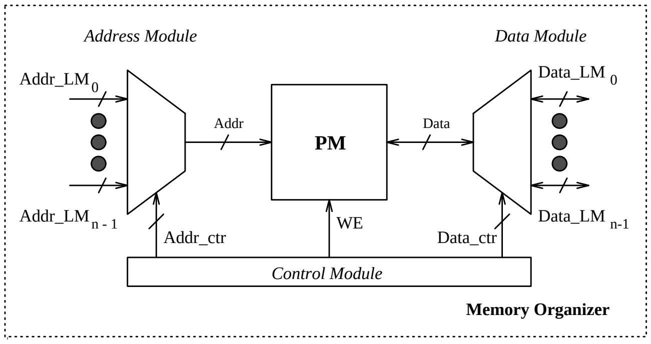

Figure 2: Block diagram of the Memory Organizer.

   If we restrict the offset to be a power of two value, we can replace the adders by simply including extra bits (*i.e.* logical ones or zeros) in the logical memories' address bus. Note that this assumption has the potential to waste part of the physical memory.

4. **Logical Memory Partition Limit**

   When a logical memory is larger than a physical memory, it is necessary to partition the logical memory into smaller pieces. For simplicity, we will partition logical memory LM_i only if:

   $$(Lw_i > Pw) \text{ or } (Ld_i > Pd) \qquad (2)$$

   From this point on we will assume that all logical memories have been partitioned in this way, and we will refer to the pieces simply as logical memories. Note that the CAD tool that generates the organizer does in fact generate the control and multiplexing to handle the larger memories.

3 Organizer Architecture

In this section, we describe the general structure of the organizer and derive models for the area and delay of the organizer hardware as a function of the occupancy of the physical memories and the width and depth of the logical memories. These models are used in the packing algorithm described in Section 4.

In the following discussion we describe area and delay models for an organizer for a single physical memory and some number of logical memories.

As illustrated in Figure 2, the organizer is divided into three modules: The Address module, which organizes the address busses, the Data module, which organizes the data busses and the Control module, which generates the timing and multiplexer control signals.

Some aspects of the model are independent of the FPGAs and memories in the actual Field-Programmable System. These *technology-independent* aspects will be described first. After describing the specific FPS that we have built, we will use it to give a technology-specific delay model.

3.1 Technology-Independent Area Model

Address Module

When more than one logical memory is mapped into a single physical memory, the logical memories' address buses are multiplexed to form a single physical memory address. The address module organizes these busses so that the access to the logical memories are mapped to non-overlapping portions of the physical memory.

The area required for the multiplexer depends on the occupancy of the physical memory and the width of the maximum offset of the logical memories' address buses. If we denote by Lo_i the offset of the i^{th} LM mapped into the PM, the area of the address multiplexer for physical memory PM_j is given by:

$$A_{Addr} = [1 + \log_2(\max(Lo_0, \ldots, Lo_{OC_j - 1}))] \cdot MX(OC_j) \tag{3}$$

Here the function $MX(x)$ returns the number of logic blocks in the FPGAs needed to implement a one bit *x to one* multiplexer. The function depends on the type of FPGA being used.

Equation 3 illustrates that different packings will result in a different sized organizer: If there are wide address buses and narrow address buses in the set of logical memories, it is better to pack the wide addresses together and the thin addresses together because the address module area is determined by the maximum width bus in the physical memory.

Data Module

The data bus multiplexing is more complex than the address bus. For *write* accesses, the logical memories' data buses are multiplexed and passed through a tri-state driver into the physical memory's data bus. For *read* accesses, it is necessary to capture each logical memory's read data in a separate register.

Each bit in the data bus of all logical memories is multiplexed independently. For example, consider the following three logical memories: one $1k \times 3$, one $1k \times 6$ and one $1k \times 8$. In order to multiplex the data buses, we need a *3 to 1* multiplexer for data bits 0, 1 and 2 of all the logical memories and a *2 to 1* multiplexer for data bits 3, 4 and 5 of the $1k \times 6$ and $1k \times 8$ logical memories. Data bits 6 and 7 of the $1k \times 8$ logical memory are connected directly to the physical memory.

To calculate the area of the data multiplexer in the data module, we first sort the width of the logical memories and re-named them so that $Lw_0 \leq Lw_1 \leq \ldots \leq Lw_{n-1}$. To minimize the area, we first multiplex the first Lw_0 data bits of all the logical memories. Then we multiplex the $Lw_1 - Lw_0$ data bits of $LM_1, \ldots, LM_{n-1}$ and so on. The area of the organizer depends on the occupancy of the physical memory and the width of the logical memories. This is given by:

$$A_{D\_Mux} = \sum_{k=1}^{OC_j - 1} (Lw_k - Lw_{k-1}) \cdot MX(OC_j - k + 1) \tag{4}$$

The technology-dependent function MX is equivalent to the one used in equation 3. Equation 4 also illustrates that different packings will result in a different sized organizer: If the logical memories consists of both wide and narrow data busses, it is more area-efficient to pack narrow busses with wide busses, because most of the wide bus part will not require multiplexing. On the other hand, if the wide busses are packed together, then all bits will have to be multiplexed.

The area of the data registers in the data module only depends on the width of the logical memories. Hence, it is independent of the packing. This area is given by:

$$A_{D\_Reg} = \sum_{k=0}^{OC_j - 1} RG(Lw_k) \tag{5}$$

Here $RG(x)$ is a technology-dependent function that returns the number of FPGA logic blocks needed to implement a register of x bits.

Control Module

The Control module consists of a finite state machine and decoders that generate the timing, clocking and multiplexer control signals.

Although we will not describe the specifics of the control circuit, we include an expression for the area of the controller for completeness:

$$A_{Ctr} = CT(OC_j) + 2 \cdot RG(\log_2 OC_j) + DC(OC_j) \tag{6}$$

Here $CT(x)$ is a function that returns the number of logic blocks needed to implement a 0 *to* $x-1$ counter and $DC(x)$ is a function that returns the number of logic blocks needed to implement a $\log_2 x$ *to* x decoder. All of these functions return larger numbers as the occupancy of the physical memory increases and therefore depends on the packing.

3.2 Technology-Dependent Delay Model

In this section we describe the *Transmogrifier-1* Field-Programmable System built at the University of Toronto [5, 6], and use it to create a technology specific delay model. The TM-1 consists of four Xilinx 4010 FPGAs [7], two Aptix Field-Programmable Interconnect Components (FPICs) [8] and four Motorola's MC62110 $32k \times 9$ synchronous SRAM chips. Figure 3 gives a block diagram of the TM-1. Two 40-pin connectors, carrying 72 bi-directional signals, are connected to a SUN SPARCstation through an additional Xilinx 4010 programmed to act as a communication controller.

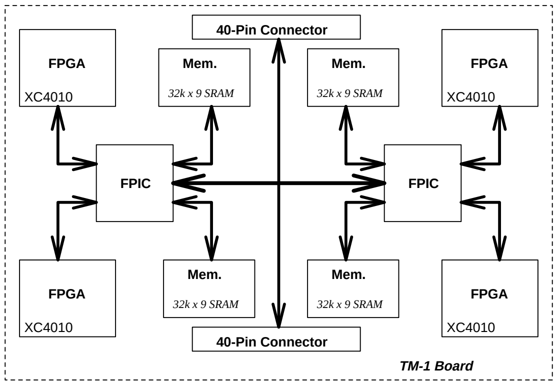

Figure 3: Block Diagram of the *Transmogrifier-1*.

Finding an exact delay model for the Organizer is difficult because the circuits implemented in most FPGAs have delays that are hard to predict. We now present a worst-case delay model for our implementation in the TM-1.

The control module uses two states for every physical memory access and so the best possible access time for LM_i packed in PM_j is:

$$Lt_i = 2 \cdot (OC_j \cdot Pt + OD_i) \qquad (7)$$

For the TM-1 FPS, the access time of the physical memories, Pt, is equal to $20ns$. The delay due to the programmable interconnect component incurs an additional $15ns$. Thus, the physical memory access time is $35ns$.

The Organizer Delay for LM_i, OD_i, is a function of the packing. It depends on the number of logic blocks

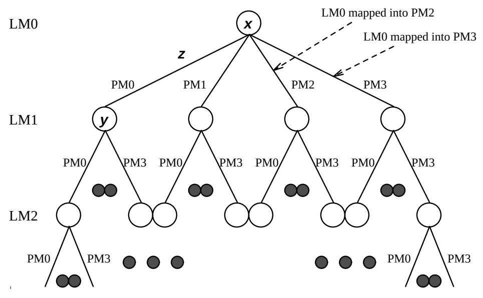

Figure 4: Decision Tree of Packing Solutions.

in the critical path and the routing needed to connect this blocks. For simplicity, we assume that the routing delay is constant and that all the logical memories mapped into one physical memory have the same delay. We assume that the critical path includes the counter in the local control module, the multiplexer in the address module and the output buffer.

If we assume $8ns$ delay per Xilinx XC4000 family logic block, (*i.e.* the $6ns$ nominal delay plus $2ns$ for routing), then the counter and multiplexer have a worst-case delay of $8ns$ each for $x = 2, 3, 4$ and of $16ns$ for $x = 5, 6, 7, 8$. The output buffer's delay is approximately equal to $10ns$.

With the above data and Equations (7), the final delay model is given by,

$$Lt_i = \begin{cases} 138 \cdot OC_j, & \text{if } OC_j = 2, 3, 4 \\ 170 \cdot OC_j, & \text{if } OC_j = 5, 6, 7, 8 \end{cases} \qquad (8)$$

This model assumes that the entire Organizer is placed in a single FPGA, and not split across multiple FPGAs. This is a reasonable assumption, as the organizer is small.

4 Memory Packing Algorithm

In this section we present an algorithm which solves the constrained problem defined in Section 2. The memory mapping problem can be solved using a Branch and Bound algorithm [9]. Figure 4 illustrates the branch and bound decision tree that represents all the possible solutions to the memory mapping problem. The nodes represent the logical memories. The edges represent the physical memories into which the logical memories are packed. For example, the decision tree in Figure 4 has two nodes, x and y, joined by edge z. The partial solution represented by node y means that LM_0 was mapped into PM_0.

The decision tree is traversed depth-first from the root. Pruning occurs in the following manner: assume that node x is the current node and has child node y connected by edge z. The tree is pruned at node y if LM_x cannot be mapped into PM_z, which occurs when any one of the following is *not* true:

1. LM_x physically fits in the size remaining in PM_z.
2. After placing LM_x into PM_z, the required access time of LM_x (Lt_x) is achieved.
3. By placing LM_x into PM_z, the LMs already placed into PM_z also meet their required access times.

If the bottom of the tree is reached then a legal packing is found. The algorithm continues to traverse the tree to find the solution with the minimal area as determine by the area model described in Section 3.

The tree is also pruned using a bounding function on the area of the partial solutions: this lower bound is calculated as the area needed to implement an Organizer for all the logical memories already mapped in the sub-tree above the current node. If there are n logical memories and m physical memories, the worst-case complexity of the algorithm is m^n. However, in most of the cases, a large portion of the tree is pruned and only a small fraction of the tree is visited

5 Results

The above algorithm has been implemented in a CAD tool called **MemPacker**. It is currently targeted towards the TM-1 FPS described in Section 3.2. The inputs to MemPacker are the logical memory parameters (width, depth and access time) and the output is the FPGA design of the organizer. MemPacker produces a Xilinx 4000 series netlist format (XNF) file that can be directly synthesized by the native Xilinx tools to produce a programming bit stream. We note that this tool saves the designer a significant amount of time by automatically generating the memory multiplexing logic and control. Manual generation of such circuits may take many hours.

Architecture Exploration

MemPacker can be used to explore the architectural space of a design from the perspective of the memory access times. It is often true that memory access times are the limiting factor in the overall speed of an application [2]. MemPacker can be used to determine the minimum access time for a set of logical memories implemented on a set of physical memories by iterating the algorithm with successively smaller required access times. The iteration prior to the one in which the algorithm fails to find a legal packing gives the minimum access time for all of the memories. The algorithm will also determine the smallest sized organizer that will achieve maximum performance.

Table 1 gives a set of example applications for which the maximum operating frequency of the memories has been determined. In this example, we assume that all of the memories will have the same access time. MemPacker is iteratively invoked to determine the smallest possible access time. The first column of Table 1 gives the source and/or name of the circuit from which the set of logical memories was derived. The second column describes the set of logical memories. These are packed into the TM-1, which consists of four $32k \times 8$ physical memories. Column three indicates the number of pieces the logical memories are partitioned into, as described in Section 2.1. Column four gives the minimum area that the organizer achieved, in terms of the number of Xilinx 4000-series logic blocks. Column five gives the maximum operating frequency achievable if the memory access time is the limiting factor in the system performance. This illustrates how MemPacker can be used to explore the performance and area costs of different memory architectures.

Area Dependency on the Packing

In Section 3 we discussed the effect of the packing on both the address and data module area. Here we illustrate these effects by an example. Consider the following set of logical memories: four $4k \times 8$, two 32×3, two 64×2 and two $8k \times 2$. These will be packed into the same physical memories as above.

Using a naive bin-packing algorithm such as Best-Fit Decreasing [10], these memories would be packed as illustrated in Figure 5a. The area cost of this packing is 84 Xilinx 4000-series logic blocks. Using the algorithm described in Section 4, the packing illustrated in Figure 5b results. The area cost of the latter packing is only 78 logic blocks. The reason for the major difference is a better matching of address and data busses within each physical memory to minimize the amount of multiplexing. Note that both packings achieve the same minimum access time over all the memories. If this constraint is relaxed and the minimum area solution is generated, the packing of Figure 5c results, which has an area cost of only 66 logic blocks. This solution is 21% smaller than the naive bin-packing solution.

System	Logical Memories	Number of subdivided LMs	Area (XC4000 CLBs)	Max Op. Freq
Viterbi decoder	three 28x16, one 28x3	7	53	3.6MHz
Neural Network Chip	16x80, 160x8, 16x16, 32x8	14	121	1.8MHz
Fast Divider	2048x56, 4096x12	9	96	2.4MHz
DMA Chip for LAN	15x24, 16x4, 256x32	8	67	3.6MHz
Industrial Example 1	six 88x8, one 64x24	9	87	2.4MHz
Industrial Example 2	three 736x16	6	52	3.6MHz

Table 1: Maximum Operating Frequency and Area for Example Circuits.

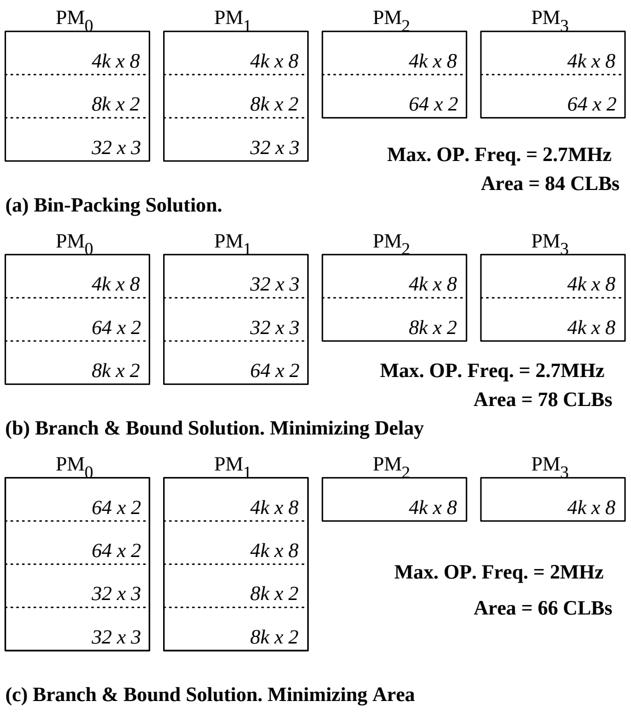

Figure 5: Area and Delay for different packings.

6 Conclusions

This paper motivates and defines the memory packing problem for Field-Programmable Systems. Memory packing is necessary when the number of application logical memories exceeds the number of physical memories. Because different packings result in both different access times and area requirements, it is an optimization problem to select the fastest and most area-efficient packing. This paper has presented a precise definition of this problem and an algorithm for its solution. The resulting CAD tool, **MemPacker** was used to synthesize area-efficient and delay-minimal packings for a set of application circuit examples.

References

[1] J. Arnold, D. Buell, and E. Davis, "Splash 2," in *4th. Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 316–322, 1992.

[2] P. Bertin, D. Roncin, and J. Vuillemin, "Programmable Active Memories: A Performance Assessment," in *Research on Integrated Systems: Proceedings of the 1993 Symposium*, MIT Press, 1993.

[3] S. Walters, "Computer-aided prototyping for ASIC-Based systems," *IEEE Design and Test of Computers*, pp. 4–10, June 1991.

[4] R. Tessier, J. Babb, M. Dahl, S. Hanono, and A. Agarwal, "The virtual wires emulation system: A gate-efficient asic prototyping environment," in *FPGA 94*, February 1994.

[5] D. Galloway, D. Karchmer, P. Chow, D. Lewis, and J. Rose, "The Transmogrifier: The University of Toronto Field-Programmable System," Tech. Rep. 306, CSRI, University of Toronto, 1994.

[6] D. Karchmer, "A Field-Programmable System with Reconfigurable Memory," Master's thesis, University of Toronto, June 1994.

[7] Xilinx Inc., San Jose, CA, *XACT Development System*, October 1992.

[8] Aptix Corporation, San Jose, CA, *Aptix System Data Book*, November 1993.

[9] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley & Sons, 1990.

[10] M. Garey and R. Graham, "Worst-case analysis of memory allocation algorithms," in *4th. Annual ACM Symposium on Theory of Computing*, 1972.