CHOP: A Constraint-Driven System-Level Partitioner

BY

Kayhan Kucukcakar and Alice Parker

Technical Report CEng 90-26

Electrical Engineering Systems

University of Southern California

Los Angeles, CA. 90089-0781

CHOP: A Constraint-Driven System-Level Partitioner

Kayhan Küçükçakar and Alice C. Parker
Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089-0781

November 6, 1990

This work was supported by the Defense Advanced Research Projects Agency and monitored by the Federal Bureau of Investigation under contract No. JFBI90092. The views and conclusions considered in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Defense Advanced Research Projects Agency or the U.S. Government.

CHOP: A Constraint-Driven System-Level Partitioner

Abstract

This paper presents a novel constraint-driven system-level partitioning approach for behavioral specifications. The software supports a designer in partitioning behavioral specifications onto multiple chips while satisfying hard constraints which include individual chip areas, pin counts, system performance and system delay. The partitioning methodology implemented in CHOP is based on predicting the feasibility of tentative partitions, searching through potential implementations of partitions to find combinations which are feasible with system integration overhead taken into account. The results show the benefits of using such a software tool. The approach has as its immediate applications behavioral partitioning, system-level advising and task creation based on a custom-designed processor style.

1 Introduction

Most digital designs are too large to fit on a single chip. At present, much chip partitioning is performed manually by designers. With the increasing complexity of designs and quick turn-around time requirement, faster partitioning methods must be used.

The time and the way partitioning is performed is quite important. Partitioning to meet design constraints can be performed at the behavioral level, the RT (Register Transfer) level or the logic level. The overall synthesis process consists of a number of interacting tradeoffs and design decisions. The quality of synthesis results is highly sensitive to these design decisions. Therefore, if a design were synthesized as if it were a single chip design, but must later be partitioned onto multiple chips, the chances of synthesizing an inferior design would be considerably higher due to the fact that some of early design decisions were made without information about partition boundaries. If the partitioning were done after behavioral synthesis, it may not be possible to find any feasible partitions. In such a case, the synthesis process would somehow have to be repeated to find a feasible partitioning. Alternatively, if the partitioning were performed at an early stage, then different parts of the design space could be explored. If the partitioning were performed at the behavioral level without any implementation information, then determining the feasibility of any partitioning would involve a complete synthesis process. Both cases might suffer severely from long iteration times to reach a feasible solution.

All partitioning methods use some measures to determine the feasibility of a design or measures to evaluate the *quality* of the partitioning. At the RT or logic level it is relatively easy to use accurate measures because the subsequent changes to the global structure after partitioning are minimal. On the other hand, at the behavioral level, global information about the final design characteristics is non-existent when no structure exists. As a result, behavioral partitioning is further complicated by the need to finding accurate measures of partitioning quality.

We propose a constraint-driven partitioning methodology which is based on determining the feasibility of tentative partitions at the behavioral level using accurate prediction methods. The method does not suffer from long iteration times due the fact that fast predictors are used in place of synthesis tools to supply feedback to the partitioner. This methodology is illustrated in the prototype software, CHOP.

1.1 Related Research

Partitioning has been used at many stages of the digital design process. Partitioning of processors has been in existence for a long time. Floating-point co-processors, I/O peripherals and memory management units are among the most common partitions. A study on partitioning strategies for multi-chip VLSI microprocessors was performed in [12].

Partitioning of logic circuits in order to meet gate and pin count constraints of chips was described by Payne and vanCleemput in [10]. In this approach, a quality function was used to measure the costs of partitions, the compactness and the reliability of the overall design. Another reason to partition logic circuits is to keep the design size small enough for logic synthesis tools to work efficiently [1]. SPARTA [11] is a tool which evaluates an RTL design with a spreadsheet-like approach which checks if any area, power and pin count constraints are violated.

A manual partitioning method as a part of the behavioral synthesis process is described by Walker [13]. This approach allows the designer to synthesize manually partitioned designs. A partitioning algorithm using a multi-level clustering approach has been proposed to reduce the control overhead and routing area [2]. This approach uses the information about the graph topology and behavioral synthesis to generate the best intuitive partitioning to improve the quality of single-chip designs, but does not consider design constraints. A similar clustering-based partitioning algorithm has been successfully used in behavioral synthesis [6].

A heuristic for partitioning graphs with costs on the edges into subgraphs with specified sizes while trying to minimize the total cost of edges cut was proposed by Kernighan and Lin [4]. Although this heuristic can be applied successfully to partitioning of logic circuits or RTL designs, the model used in this approach is not directly applicable for partitioning of behavioral specifications due the fact that behavioral synthesis introduces significant and generally irregular sequential behavior into the design. This causes most final design characteristics to be a function of the sequential behavior introduced and hence a function of the structure as well as of the original behavior. Without having the results of behavioral synthesis or accurate predictions of these results, it is questionable if one can directly correlate "sum of costs of values cut" to the pin count requirement or "sum of sizes of operations in a partition" to the area of chips. Obviously, the areas of chips are consumed by not only functional units but also by registers, steering logic, controllers and wiring.

2 Partitioning Approach

The overall operation of the partitioner CHOP is shown in Figure 1. BAD is a Behavioral Area-Delay Predictor [5] which is embedded in the partitioner. The ovals indicate data and the rectangles indicate CHOP's and designer's actions.

2.1 Features

CHOP allows the designer to interactively create and modify partitions of behavioral specifications onto multiple chips. Each chip may have more than one partition which might or might not be implemented independently (our current prediction techniques assumes that partitions are synthesized separately). CHOP uses fast prediction methods; therefore it does not suffer from long iteration times. The partitioning is not performed in an abstract space; the approach deals with physical design parameters, constraints and goals.

The proposed partitioning approach is comprehensive. Detailed prediction techniques addressing most digital design aspects are used. The prediction techniques include: design style selection (pipelined/nonpipelined), module selection, operator, register and multiplexer allocation, wiring area/delay, controller area/delay, memory bandwidths, off-chip communication bandwidths, performance matching between independent datapaths, data

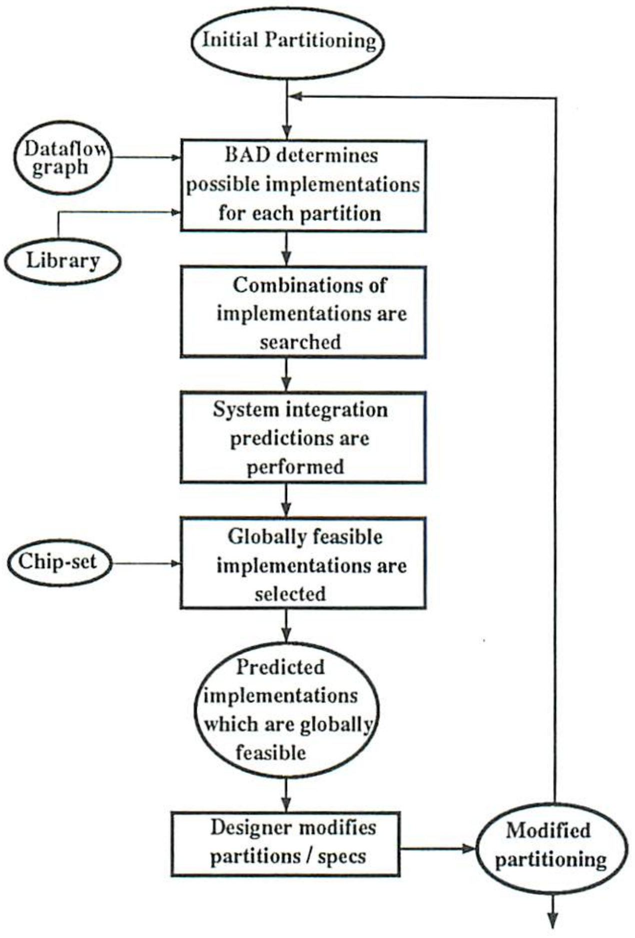

Initial Partitioning
Dataflow graph
BAD determines possible implementations for each partition
Library
Combinations of implementations are searched
System integration predictions are performed
Chip-set
Globally feasible implementations are selected
Predicted implementations which are globally feasible
Designer modifies partitions / specs
Modified partitioning

Figure 1: The overall operation of CHOP

buffering, and a distributed control mechanism. CHOP uses probabilistic methods to determine the feasibility of tentative partitions. The structure of the partitioner is such that inclusion of new predictors requires minimal effort.

The partitioning software can be instructed to discard any infeasible or inferior predicted designs immediately upon detection. This keeps the number of eligible predicted designs down, resulting in significantly faster execution speed and smaller run-time memory requirement. Keeping discarded predictions is only useful when the designer wants to see the entire design space explorable by the tool. CHOP performs the pruning of predictions which would lead to infeasible or inferior designs on two levels. The first level pruning happens before integrated partitioning predictions are performed. The predictions produced by BAD for each partition are examined and predictions which are infeasible or inferior are discarded. After the partitioning search and global predictions start, second level pruning is performed at various times to prune infeasible predictions.

When CHOP determines the feasibility of an implementation, it outputs the design decisions and prediction results. This provides a guideline for the designer to synthesize the predicted implementation. An example is given in Section 3.1.

2.2 Inputs

The input data required for CHOP currently consists of 6 groups:

- the behavioral specification in the form of a data flow graph (with added control constructs),
- a library of components,
- the chip set onto which the design is to be partitioned,
- on and off chip memory modules to be used and assignments of memory modules to chips,
- partitions and assignments of partitions to chips,
- tentative data path and data transfer clock cycle times, the architecture style, the feasibility criteria (to be discussed later), and the design parameters.

The library generally consists of more than one component which can implement each operation type. It is assumed that the memory hierarchy is designed prior to partitioning although, in practice, designers interleave iterations of memory and behavior partitioning, a step we intend to automate in the future. The chip-set information is in the form of actual chip packages to be used. The information about each chip includes the dimensions of the project area and the pin count of the chip, pad delays, and I/O pad area. The architecture style is to be compatible with the architecture style of the synthesis tools which will later be used to implement the hardware. The architecture style can allow either single-cycle or multi-cycle operations, and be pipelined or nonpipelined. The clock cycle is an input to the system. One reason for this is the fact that determination of the system clock cycle is also influenced by other design factors such as the interface to other system components. The second reason is due to allowing multi-cycle operations which, in turn, makes it computationally intensive to determine the best cycle time in parallel with partitioning.

The reason for having two distinct clocks is that the data processing rate can be different from the possible data transfer rate. Therefore, we assume two separate clocks for data path and data transfer in our prediction mechanism, but, in order to keep design complexity at a moderate level, both clocks in our model are to be synchronous with frequencies being multiples of the major clock frequency.

2.3 Restrictions

The behavioral specification should be free of inner loops. Inner loops with determinate iteration counts can be unrolled so that the resulting data flow graph is acyclic [7,9]. Stochastic delays for operations are not supported. The resynchronization (pipe flushing) rate is assumed to be zero. Some synthesis programs (e.g Sehwa [8]) use resynchronization rates during synthesis. Sehwa, for example, may produce results which are significantly different from the predictions if the resynchronization rate is non-zero.

The topology of graph being partitioned is also important when predictions are used. In order to have accurate results in the current approach, no two partitions should have mutual data dependency. The reason for this is that our partitioning methodology is based on independent predictions and implementations of each partition. Current prediction methods use the same synthesis methods as many synthesis programs available today. That includes the fact that all inputs to partitions are assumed to be simultaneously available before the execution starts. If there is mutual data dependency among any two partitions, then, the overall predictions (area, performance and delay) may not be accurate. But, this does not pose a serious problem for partitionability of specifications, since we allow multiple partitions on each chip, and cyclic data flow is allowed among chips (see Chip 4 in Figure 2 for an example of this).

2.4 Method

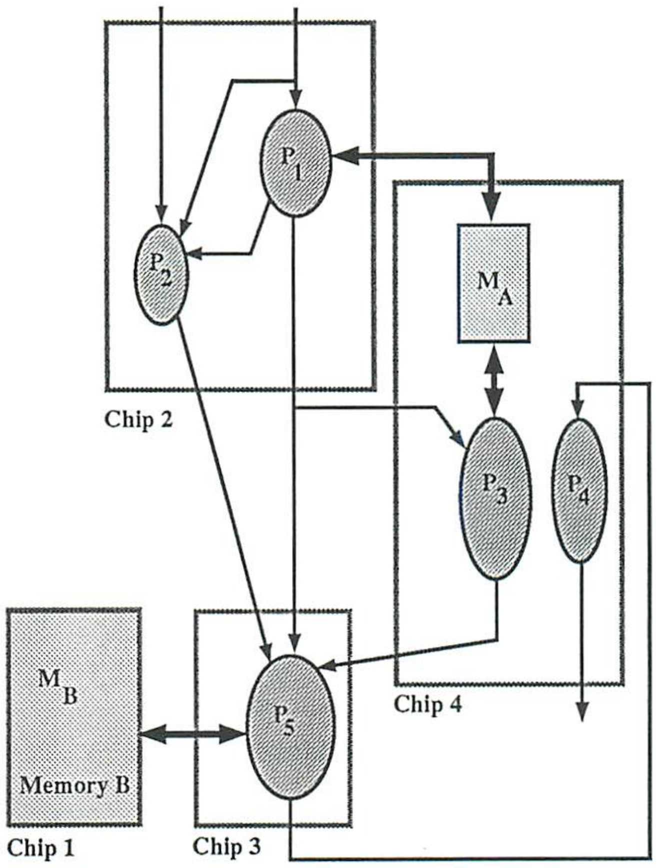


Figure 2: An example partitioning

The designer creates and modifies existing partitions and CHOP determines the feasibility of each tentative partitioning by searching for feasible implementations via predictions. An example tentative partitioning consisting of 5 partitions ($P1 - P5$), and 2 memory units (M_A and M_B) as a four-chip design is shown in Figure 2. It is important to note that

- there can be multiple partitions assigned to a single chip,
- partitions assigned to the same chip may or may not have dependencies on each other, as long as there are no cycles,
- memory blocks can be assigned to the same chips as partitions, and
- the use of off-the-shelf memory chips is allowed by CHOP. This also applies to any pre-designed implementation of any partition so that off-the-shelf data path parts can also be used.

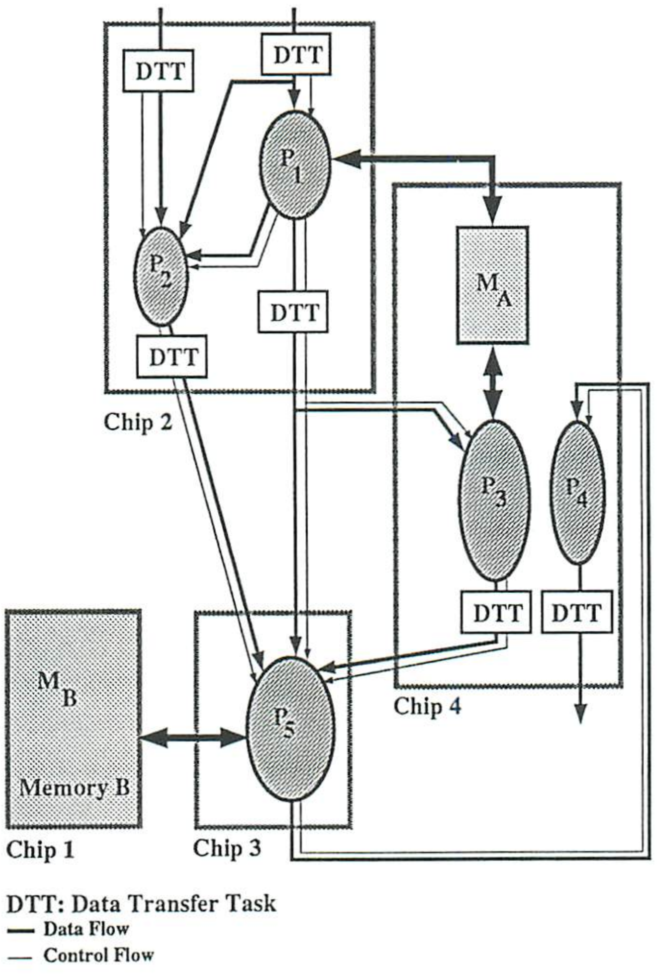
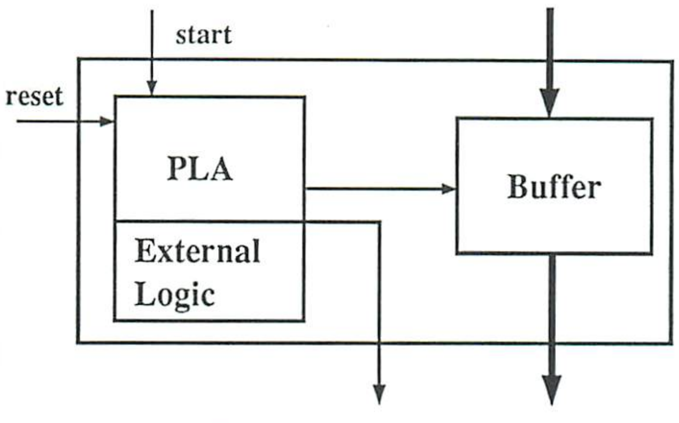
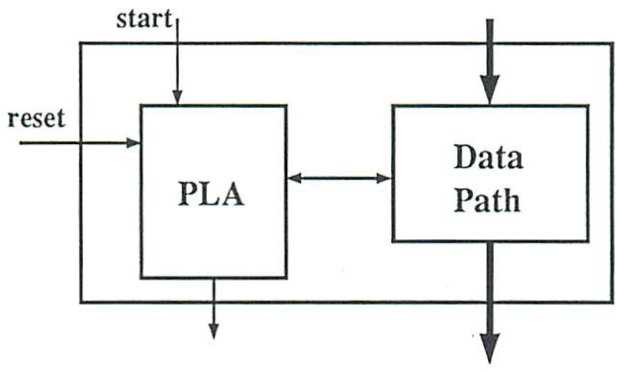


Figure 3: Task Graph for the example partitioning

When the information about partition and memory block assignments is available, data transfer tasks are created by CHOP to transfer data among partitions as shown in Figure 3. This process involves determining the manner and the amount of data to be transferred, reserving enough pins for control signals to assure proper communication between distributed controllers and also for other necessary signal pins which are not shared (Select, R/W lines for memory blocks). CHOP assumes that a special purpose hardware unit will be used to implement each task, as shown in Figure 4; i.e. data transfer modules will be designed to implement data transfer tasks and PUs (Processing Units) will be used to implement partitions.



Data Transfer Module (DTM)



Processing Unit (PU)

Figure 4: The architectural building blocks to implement tasks

At this stage, the processing requirements (but not the resource requirements or timing characteristics) of all tasks are known. As a first step, CHOP predicts possible individual implementations for each partition P_i by calling BAD. BAD generates predictions for several implementations using detailed prediction techniques. The results from BAD include completely specified characteristics (area, performance, delay) and memory bandwidth requirements for each memory block (I/O operations are modeled as memory-mapped I/O) for a number of predicted designs. BAD considers pipelined and non-pipelined design styles, includes all possible module-set combinations, considers serial-parallel tradeoffs and performs detailed predictions on register and multiplexer allocation, PLA-based controller area, and standard cell routing area, as well as the additional delays introduced to the clock cycle (register, multiplexer, wiring and PLA delays). The results from BAD have being tested using the ADAM Synthesis tools [3] and have been very accurate so far.

When multiple predicted implementations (with different area-delay characteristics) exist for partitions, selecting only one implementation (one predicted design) for each partition while satisfying global design constraints and optimizing a design goal is a hard problem. The problem is similar to the module selection problem in high-level synthesis but it is more complicated. The timing of each partition must be compatible with other partitions so that the flow of data between partitions is efficient and smooth. Also, multiple predictions residing on a single-chip have to jointly meet the area limitations of the chip which invalidates some combinations of implementations. Global implementations with very different design characteristics may be obtained for each combination of partition implementations. But, it may not be possible to accurately predict the global design characteristics without tying down the characteristics of each partition which make up the global design. In order to predict the characteristics of global implementations, we use searching techniques for possible combinations of local implementations followed by the system integration predictions and feasibility analysis for each combination explored.

The designer may choose between two separate heuristics at run-time. These have been devised to search the possible combinations of partition implementations. Neither of the heuristics can be claimed to be better than the other in terms of the quality of results or run-time but they explore the design space differently.

The first heuristic is based on explicit enumeration. The heuristic searches all possible combinations of implementing the global design (partitioning), given the predicted implementations of individual partitions. If N_i represents the number of predicted implementations for partition P_i, then, the number of possible combinations of global implementations, N is given as:

$$N = \Pi_i N_i$$

where i is varies over all partitions. The heuristic assumes that the performance of each combination is upper bounded and set by the slowest partition implementation in the combination. If any 2 or more partition implementations in a global implementation have pipelined design styles and different data rates, then the global implementation is feasible due to a data rate mismatch. On the other hand, faster nonpipelined implementations can be used along with slower pipelined implementations. Although it may seem that this enumeration considers all possibilities, unfortunately, there are multiple ways of integrating the partitions considered in each combination, and the heuristic does not examine all ways. As a result, even this enumeration is a heuristic.

The second heuristic tries to find the minimum system delay for each feasible performance value (each feasible initiation interval in terms of the clock cycle). For each feasible initiation interval, the heuristic starts with the fastest predicted implementation for each partition and iteratively considers more serial implementations of partitions residing on chips whose area constraint is violated. Selection of more serial implementations is done in such a way that the incremental system delay caused by serialization is minimized. This selection generally favors the serialization of off-critical-path partitions (off-critical with respect to the current global implementation considered). The outline of the algorithm is given in Figure 5.

Given a selection of predicted designs (one predicted implementation per partition) from any selection method described above, the system integration predictions (prediction

```
Procedure find_feasible_implementations_for_the_partitioning

Let $W_i$ be the index for predicted implementation of partition $P_i$.
Let $L_i$ is the initiation interval of $W_i$.
Let $Q$ be the list of candidate partitions for more serialization.
Sort all predicted implementations for all $P_i$ in increasing order first for the initiation interval
and then for the circuit delay.
for each feasible initiation interval $l$
  for all $P_i$
    Initialize $W_i$ to the first (fastest) predicted implementation in the sorted prediction list of $P_i$
  Advance each $W_i$ until $L_i \geq l$ or $W_i$ is a non-pipelined implementation with $L_i \leq l$
  while TRUE
    If $\neg$ ($\exists$ a $W_i$ for all partitions)
      break
    Estimate system integrations using $l$ and implementations pointed by $W_i$
    If the estimation result is feasible
      Record the overall prediction results.
      Set $Q$ to nil.
    else
      Set $Q$ to the list of partitions residing on chips whose area
        constraint is violated by the last system integration prediction.
    For each partition in list $Q$
      Move $W_i$ forward (serialize) in its list.
      Find the expected system delay using the urgency scheduling for chip pins
        and memory blocks
      Restore the old value of $W_i$
      Record the partition number for the minimum system delay.
    If $\exists$ a partition number recorded above
      Serialize the partition. (Move $W_i$ one element forward in its list)
End.
```

Figure 5: A heuristic algorithm using an iterative approach

of overall design characteristics including data transfer module implementations, overhead for the clock cycle, overall system performance and delay) are performed as described in the next section. System integration overhead is highly dependent on the characteristics of the selected implementations for partitions and the chip-level constraints.

2.5 System Integration Overhead

System integration predictions basically involve predicting data transfer module characteristics and, of course, the performance and delay characteristics of the overall system. Since chips have hard pin-count constraints, data transfer times are functions of the amount of data to be transferred and the availability of pins. It is assumed that the maximum possible bandwidth is used for each data transfer and each data transfer task and P_i should wait until its required bandwidth (for pins and memory) is available [1]. The bandwidth for each data transfer task is defined as the minimum bandwidth of all chips involved in the data transfer. During bandwidth calculations, the effects of simultaneous memory I/O on pin usage are also taken into account. This bandwidth is then used to calculate the duration (delay) of each data transfer task.

Having delays of all tasks (data transfer tasks and partitions), an urgency scheduling is performed to confirm feasibility of sharing the data pins of chips as well as to keep memory accesses to each memory block feasible while reaching the minimum overall system delay. The urgency measure is based on the actual critical path delays of tasks and is similar to urgency measures used in [8].

The resulting task schedule is used in determining the characteristics of data transfer modules. One data transfer module per data transfer task has to be placed on all chips involved in the data transfer (one in the output mode, the rest in the input mode). Each output data transfer module may contain a wait until enough pins are available for the data transfer, followed by the data transfer. Each input data transfer module has a data transfer which may be followed by a wait time to hold the data until the destination partition can accept the data. The buffer size, B needed for each data transfer module is predicted as

$$B = D \times (\lceil \frac{W}{l} \rceil + \frac{X}{l})$$

where D is the size of data to be transferred, W is the wait time, X is the data transfer time, and l is the initiation interval considered. The second term corresponds to the buffer requirement while the data is being transferred. During the data transfer, the buffer requirement is not necessarily the maximum due to the stair-like nature of the storage requirements.

Although individual partitions may have pipelined and nonpipelined implementations co-existing in the final design, the overall process (including the data transfer tasks) is always assumed to be pipelined, considering nonpipelined partitions as special cases. Therefore, the wait time for a data transfer task could be longer than the initiation interval of

[1] Of course, the actual implementation might produce more complicated pin allocation and hence better performance than CHOP predicts.

the system. On the other hand, the data transfer time for each data transfer task cannot be longer than the initiation interval of the system in order not to cause data clashes (Pin counts are hard constraints which cannot be changed by CHOP).

The data transfer module controller should be active from the time there is a start signal for new data transfer until the time the data transfer module relinquishes control of the data to the destination partition. Of course, when the wait time of a data transfer task is longer than the initiation interval, the corresponding data transfer module is active at all times. The wait and data transfer times are used to predict the number of inputs, outputs and product terms of a PLA to control the data transfer, from which PLA size and delay are predicted by the same methods used in BAD.

2.6 Feasibility Analysis

When the individual characteristics of PUs and data transfer modules are available, the feasibility of the partitioning is checked. The feasibility criteria is same as the criteria used in BAD. All prediction results (in the form of a triplet: a lower bound, a most likely and an upper bound value) are stored in a statistical environment, and the feasibility analysis is done with the probabilistic methods used in BAD. The details can be found in [5].

The feasibility analysis is performed for each chip area constraint by considering the area taken by PUs, data transfer modules residing on each chip, and multiplexing to share the data pins of each chip. The performance and delay characteristics of PUs, chip packaging and data transfer modules as well as delays of pin multiplexing logic are used to calculate the delay introduced into the clock cycle. The clock cycle time is adjusted and feasibility of the performance and the system delay are checked.

2.7 Partitioning Modification

After the feasibility of a partitioning is checked, the partitioning process can be continued as the designer modifies the specifications and constraints, based on the feedback from the CHOP partitioner. Modifications can be grouped into 4 major groups:

Behavioral Partitions: The changes can be as simple as operation migrations from partition to partition, or migration of partitions from chip to chip. It is also possible to decrease the size of partitions (by increasing the number of partitions) to make use of the unused space left on chips. The granularity of partitions is a tradeoff issue by itself because too fine granularity as well as too coarse granularity can reduce the design quality.

Memory blocks: The assignments of memory blocks can also be changed to possibly decrease the number of off-chip memory accesses. The design of memory partitioning and behavioral partitioning are highly interrelated and unless both can be performed simultaneously, interleaving memory and behavioral partitioning must be performed.

Target chip set: The feasibility of behavioral partitioning is also highly dependent on the target chip set for partitioning. Less area available for the design generally translates into slower more serial designs. By the same token, the less the number of pins available for data transfer, the slower the data transfer. Modifications in target chip-set characteristics affect the feasibility of the behavioral partitioning. Target chip characteristics generally dictate the overall manufacturing cost of the design.

Constraints: High performance and tight delay constraints translate into more parallel designs which are larger in size. High performance constraints also cause the I/O pin usage to increase, which in turn, makes some implementations infeasible.

3 Results

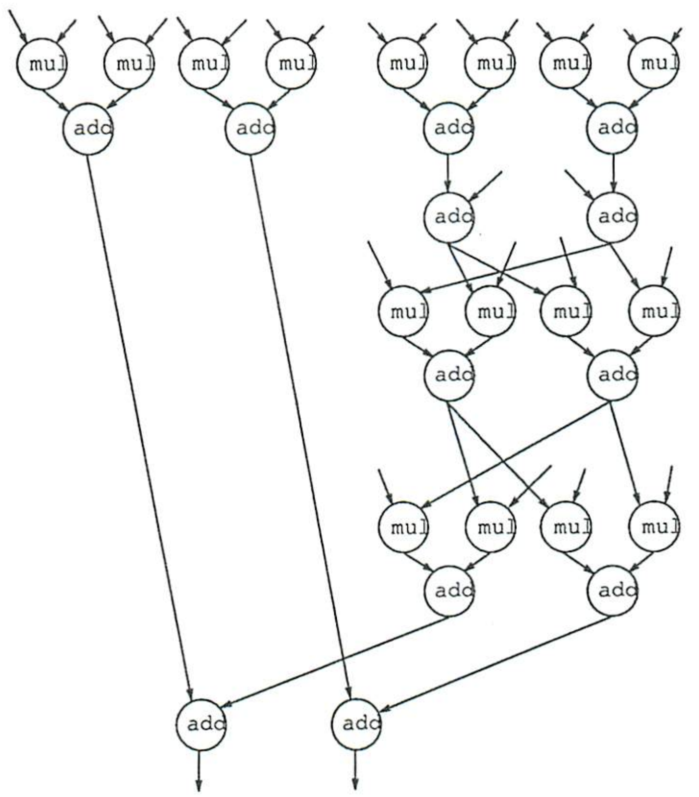


Figure 6: AR lattice filter data flow graph

Two sets of experiments have been performed using an AR lattice filter element shown in Figure 6. This simple example was chosen due to space limitations in this paper. The AR filter does not have any memory or I/O operations and unfortunately, it does not demonstrate all features of the partitioner. The design library used for the experiment is shown in Table 1. The library has a 3 micron technology which is consistent with the technology assumed by BAD. Table 2 shows the set of chip packages considered in the experimentation. The main clock cycle from which the data path and the data transfer clocks will be derived was set to $300ns$. All CPU times reported are in seconds on a Solbourne Series 5e/900 (Sun 4 compatible).

Module Name	Type	Bit Width	Area mil^2	Delay ns
add1	Addition	16	4200	34
add2		16	2880	53
add3		16	1200	151
mul1	Multiplication	16	49000	375
mul2		16	9800	2950
mul3		16	7100	7370
register	Register	1	31	5
mux	2:1 Multiplexer	1	18	4

Table 1: Library used in the experiments

We first set the performance and delay constraints to $30000ns$. The feasibility criteria were defined as follows: If a predicted design has a probability of 100% of satisfying the performance (initiation interval) and chip area constraints, and a probability of 80% of satisfying the system delay (delay from inputs to outputs) constraint, then the predicted design is considered feasible.

Three partitioning schemes were evaluated using different design parameters. The first partitioning had a single partition, the second had two partitions (a horizontal cut from the middle of the graph), and the third had three partitions of approximately equal size. In all cases, each partition was manually assigned to a separate chip.

No	Width mil	Length mil	Number of Pins	Pad Delay ns	Pad Area mil^2
1	311.02	362.20	64	25.0	297.60
2	311.02	362.20	84	25.0	297.60

Table 2: A subset of MOSIS Standard Chip Packages

3.1 Experiment 1

In the first set of experiments, the design space was explored using a single-cycle-operation architecture style which is a widely used style among current datapath synthesis approaches. The data path clock is set to be 10 times slower than the main clock cycle and the data transfer clock is set to be the same speed as the main clock. The performance and delay constraints set earlier, correspond to an initiation interval and system

Partition Count	Total number of predictions	Number of feasible predictions
1	111	5
2	207	25
3	236	32

Table 3: Statistics on the results from BAD for experiment 1

delay of 10 data path cycles which may not require a very highly parallel or pipelined implementation.

In the first set of experiments,the goal was to search for the fastest feasible design. As the first step, the feasibility of a single-partition implementation was checked. Then, given a feasible predicted design, the feasibility of faster designs was explored using multiple partitions on multiple chips. The feasible and non-inferior predicted designs are shown in Table 4. Table 3 shows the statistics on the predicted implementations returned by BAD for each partitioning considered in Table 4. In Tables 4 and 6, the type of heuristic used for a given run is shown under the field called H. E is for the enumeration based heuristic and I is for the iterative heuristic results.

Partition Count	Package Type	H	CPU Time	Partitioning Imp. Trials	Feasible Trials	Initiation Interval	Delay	Clock Cycle ns
1	2	E	0.07	5	1	60	67	312
1	2	I	0.06	13	1	60	67	312
2	2	E	0.59	156	2	30	57	310
	2					20	79	309
2	2	I	0.21	9	2	30	57	310
	2					20	79	309
2	1	E	0.43	156	2	30	59	310
	1					20	80	309
2	1	I	0.22	9	2	30	59	310
	1					20	80	309
3	2	E	1.98	1050	1	30	77	308
3	2	I	0.27	9	1	30	67	308

Table 4: Results of experiment 1

It can be seen from Table 4 that two times higher performance can be obtained easily by doubling the available chip area. If slightly longer system delays can be tolerated, then up to 3 times performance increase is possible.

The effect of pin counts of chip packages can be seen in Table 4. Using 64 rather than

84 pin chip packaging causes a slight increase in the system delay of the predicted designs, mainly due to longer data transfer times of inputs and outputs.

It can also be seen from Table 4 that partitioning a design onto more and more chips in order to improve the performance or system delay characteristics may not always be possible. Partitioning a design onto more chips generally increases the usage of chip pins to transfer data between the chips and chip pins become the bottleneck in high-performance designs.

As it can be seen from Table 4, a large number of possible implementations were searched by CHOP. The run-time for this search was extremely short, mainly due to the fact that the partitioner was instructed to discard partitioning implementations as soon as it detected infeasibility or inferiority. In order to show the size of the design space explored by CHOP, we ran CHOP again by requesting to keep all implementations (no pruning) it encounters during the same search performed to construct Table 4. The designs encountered are shown in Figure 7. The CPU time spent to generate these predictions for a total of 13411 (699 unique) designs shown in Figure 7 was 61.40 seconds, showing the advantage of the pruning techniques used in CHOP.

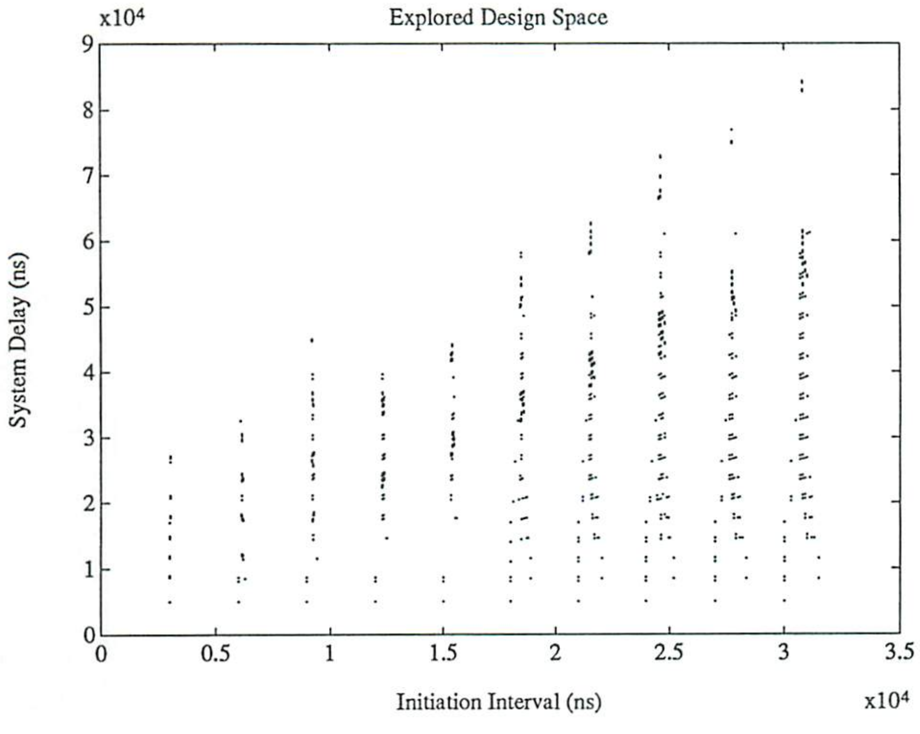


Figure 7: Designs considered during experiment 1

In order to illustrate the guidelines provided to the designer by CHOP, consider the feasible design predicted on the third line of Table 4. The predicted initiation interval (in terms of the clock cycle), system delay and clock cycle time for the design are 30, 57 and 310 respectively. CHOP has reached to this prediction, by selecting:

- Partition 1

  - a non-pipelined design style with 2 stages,
  - module library of add2 and mul3 for additions and multiplications,
  - 3 adders and 4 multipliers,
  - 104 bits of registers for the data path,
  - 349 1-bit 2-to-1 multiplexers, and
  - other design decisions/characteristics.
- Partition 2
  - a non-pipelined design style with 3 stages,
  - module library of add2 and mul3 for additions and multiplications,
  - 2 adders and 3 multipliers,
  - 56 bits of registers for the data path,
  - 283 1-bit 2-to-1 multiplexers, and
  - other design decisions/characteristics.

Similar predictions are also output for each data transfer module. The designer can guide the synthesis tools to reach these expected design characteristics.

3.2 Experiment 2

In the second experiment, we wanted to explore the feasibility of faster and more efficient designs by reducing the inefficiencies which might have been caused by discretization due to the slow data path clock. Therefore, the architecture style is set to include multi-cycle operations. The data path and data transfer clocks are set to be the same speed as the main clock. The performance constraint is tightened to $20,000ns$ since we were trying to find faster designs and there was no need to search the parts of the design space which contain slower designs.

It must be noted that using a clock cycle of $300ns$ and performance constraint of $20,000ns$, approximately 60 possible initiation intervals are considered for each implementation. Taken together with a library which allows up to 9 module-set configurations for implementation of each partition, searching the design space is an ambitious goal. The feasible and non-inferior predicted designs are shown in Table 6. Table 5 shows the statistics on the predicted implementations returned by BAD for each partitioning considered in Table 6.

It can be seen from Table 6 that a multi-cycle-operation architecture allows a more efficient use of a faster clock (reducing the discretization caused by the synchronous clocking style) resulting in higher performance designs.

In order to show the size of the design space explored in the second experiment, we tried to run CHOP again while requesting it to keep all implementations it encounters (without any pruning) during the same search performed to construct Table 6. Unfortunately, we were unable to do so due to swap space problems. Figure 8 shows implementations encountered, considering only a 1-partition implementation. This corresponds to the first two lines of Table 6. The CPU time spent to generate these predictions for a total 21828

Partition Count	Total number of predictions	Number of feasible predictions
1	656	3
2	1437	24
3	1818	43

Table 5: Statistics on the results from BAD for experiment 2

Partition Count	Package Type	H	CPU Time	Partitioning Imp. Trials	Feasible Trials	Initiation Interval	Delay	Clock Cycle *ns*
1	2	I	0.44	99	1	40	47	400
1	2	E	0.23	3	1	40	47	400
2	2	I	1.41	97	2	20	76	385
	2					22	44	384
2	2	E	1.25	143	3	20	76	385
	2					21	58	384
	2					22	45	384
3	2	I	1.82	50	1	20	46	374
3	2	E	3.51	2912	1	16	38	374

Table 6: Results of experiment 2

(8764 unique) designs shown in Figure 8 was 65.89 seconds. But, CHOP implicitly explored the huge design space we are unable to show here by immediate pruning of predictions which would be infeasible or inferior. The faster the data path clock, the more design possibilities exist for a given set of design constraints. Of course, using a slower data path clock and tightening the performance and delay constraints would speed up the predictions as well as reducing the run-time memory requirement.

4 Conclusions

A constraint-driven behavioral partitioning methodology has been presented. When the constraints are removed, then the entire explorable design space for the partitioned design can be predicted. The fast feedback from CHOP allows a designer to search for several alternative implementation schemes. A tool like CHOP is extremely beneficial to speed up the system-level design process, and possibly helps to increase the quality of designs.

The methodology presented in this paper is also suitable as a system-level advisor. The designer can easily check the effects of system-level decisions in real-time. The tool will also be extremely helpful in understanding immature areas of system-level automation,

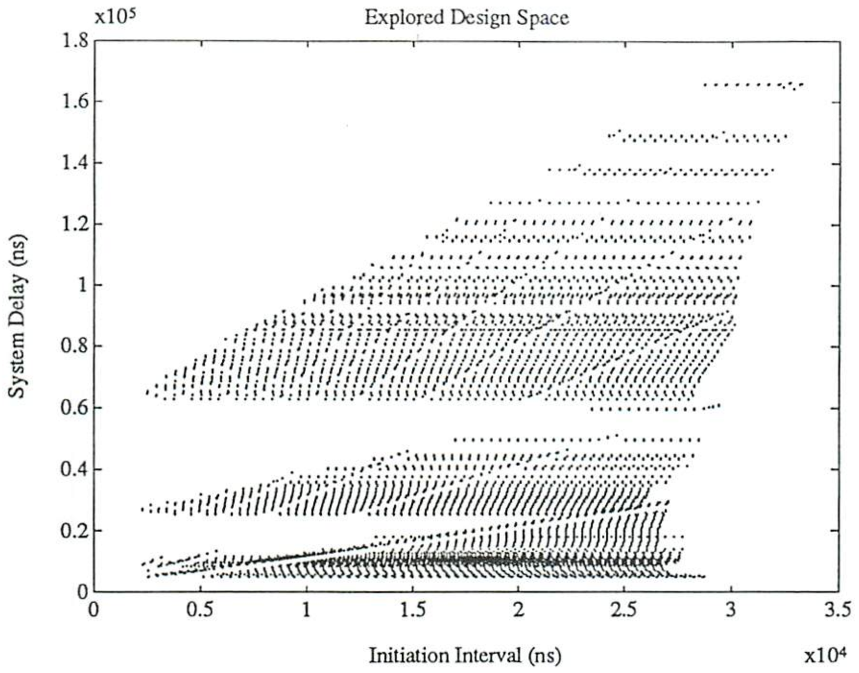


Figure 8: Some of designs considered during experiment 2

especially exploring partitioning methodologies, effects of high-level transformations, and effects of memory hierarchy.

5 Future Research

An immediate task is to synthesize and layout some partitioned designs. Although the prediction results are accurate, the prediction methods have to be upgraded to keep up with newer and improved methods used in behavioral synthesis. One example is the consideration of individual inputs with unique arrival times in scheduling.

The partitioning methodology does not consider the overhead for testability. In order to synthesize highly testable designs while still satisfying design constraints, the testability overheads for area, delay, performance and pin count have to be considered in the prediction mechanism. Also, the partitioning methodology currently works with area, delay, performance and pin count characteristics and needs to be extended to include power consumption constraints.

References

[1] Raul Camposano and R. K. Brayton. Partitioning before Logic Synthesis. In *Proceedings of International Conference on Computer Aided Design*, IEEE, November 1987.

[2] E. Beth Dirkes. Architectural Partitioning for System Level Design. In *Proceedings 26th Design Automation Conference*, ACM/IEEE, June 1989.

[3] R. Jain, K. Kucukcakar, M. J. Mlinar, and A. C. Parker. Experience with the ADAM Synthesis System. In *Proceedings 26th Design Automation Conference*, ACM/IEEE, June 1989.

[4] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, 49(1), January 1970.

[5] K. Kucukcakar and A. C. Parker. BAD: Behavioral Area-Delay Predictor. *IEEE Journal of Solid-State Circuits, First Special Issue on Microelectronic Systems*, 1990. Submitted.

[6] M. C. McFarland. Computer-Aided Partitioning of Behavioral Hardware Descriptions. In *Proceedings 20th Design Automation Conference*, ACM/IEEE, June 1983.

[7] N. Park. *Synthesis of High Speed Digital Systems.* PhD thesis, University of Southern California, August 1985.

[8] N. Park and A. C. Parker. Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications. *IEEE Transactions on Computer Aided Design*, 7(3), March 1988.

[9] P. G. Paulin and J. P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASIC's. *IEEE Transactions on Computer Aided Design*, 8(6), June 1989.

[10] Thomas S. Payne and W. M. vanCleemput. Automated Partitioning of Hierarchicaly Specified Digital Systems. In *Proceedings 19th Design Automation Conference*, ACM/IEEE, June 1982.

[11] Martin L. Resnick. SPARTA : A System Partitioning Aid. *IEEE Transactions on Computer-Aided Design*, CAD-5(4), October 1986.

[12] Brian Schmult. *Partitioning Strategies for Multi-chip VLSI Microprocessors.* Technical Report CMUCAD-84-26, Department of Electrical and Computer Engineering, Carnegie-Mellon University, February 1984.

[13] R. A. Walker. *Design Representation and Behavioral Transformation for Algorithmic Level Integrated Circuit Design.* PhD thesis, Department of Electrical and Computer Engineering, Carnegie-Mellon University, April 1988.