Hierarchical Behavioral Partitioning for Multicomponent Synthesis[*]

Nand Kumar
Triquest Design Automation
San Jose, CA 95008
nand@triquest-da.com

Vinoo Srinivasan   Ranga Vemuri
Laboratory for Digital Design Environments, ECECS
University of Cincinnati, Cincinnati, OH 45221-0030
{vsriniva, ranga}@ece.uc.edu

Abstract

Packaging technology has tremendously improved over the last decade. Various packaging options such as ASICs, MCMs, boards, etc. should be well explored at early stages of the system-synthesis cycle. In this paper we present a hierarchical behavioral partitioning algorithm which partitions the input behavioral specification into a hierarchical structure and binds all elements of the structure to appropriate packages from a given package library. As an application to our partitioner, we integrated the partitioner with a high level synthesis tool to create an environment for multicomponent synthesis and hierarchical package design. We provide detailed partitioning algorithms and experimental results.

1 Introduction

High level synthesis converts a behavioral specification of a digital system into an equivalent RTL design that meets a set of stated performance constraints [1, 2, 3]. This RTL design can be partitioned into multiple segments to realize a multichip design. Partitioning RTL designs, however, has various drawbacks: (1) Control lines could be crossing segment boundaries; (2) Operators could be shared by operands in different segments, this results in poor performance due to interchip communication; (3) The design is fixed during synthesis and thus there is very little scope for circuit transformations to improve performance; (4) RTL designs are much larger than their behavioral counterparts, thus, the solution space increases rapidly with the size of the synthesized behavior, making the partitioning process very time consuming; and (5) Power estimation/measurement for RTL designs is too time consuming and not viable for very large designs.

Recent efforts in system-level synthesis have led to the development of high level synthesis systems that can produce multichip digital systems [4, 5, 6]. These systems, however, do not consider the impact of packaging on high level synthesis and hence designs produced by these systems cannot efficiently use available high performance packaging technology. For very large, performance critical designs, an efficient *hierarchical behavioral partitioner*, which fully explores various packaging options, is required to tackle the drawbacks of RTL partitioning. The inputs to the *Hierarchical behavioral partitioner* are: (1) a behavioral specification to partition; (2) parameterized register level component library characterized for area, delay, and switching activity; (3) package library with area, pins, switching activity, clock speed, and cost information for all packages; and (4) cost constraint $\mathcal{C}$, in dollars on the entire design. The output of the partitioner is: (1) a set of behavioral specifications, which together form the original specification; (2) a set of structures that realizes the hierarchical design; and (3) a binding of the behavioral specifications and the structures to appropriate cost effective packages from the package library.

The input behavioral specification (which may be given in VHDL) consists of a set of communicating and concurrently executing processes. This specification is internally represented as a process graph; with nodes in this graph representing the processes, and edges being communication channels. We formulate the hierarchical partitioning problem and propose a solution for the hierarchical partitioning and package binding problem. We show how our partitioner can be integrated with a high level synthesis tool to create an environment for multicomponent synthesis and hierarchical package binding. Experimental results for a number of designs are presented.

2 Problem Formulation

Definition 2.1 A *1-level partition* of a set $\mathcal{N}$ is a collection, $\mathcal{S}$, of nonempty sets (**segments**), such that:
- $\mathcal{S}$ is a collection of mutually disjoint sets, i.e., if $C \in \mathcal{S}, D \in \mathcal{S}$, and $C \neq D$, then $C \cap D = \phi$, and
- the union of $\mathcal{S}$ is the whole set $\mathcal{N}$, i.e., $\bigcup_{s \in \mathcal{S}} s = \mathcal{N}$. □

Definition 2.2 A *k-level partition*, $\mathcal{P}$, of a set $\mathcal{N}$ is a set of 1-level partitions $P_1, P_2, \ldots, P_k$ such that
- P_1 is a 1-level partition of $\mathcal{N}$, and
- for $1 \leq i < k$, P_{i+1} is a 1-level partition of P_i. □

Definition 2.3 A k-level partition of a graph $G = (N, E)$ is a k-level partition of N, where N is the set of nodes and E is the set of edges. □

The performance attributes of the nodes in the graph G and level 1 partition segments (each segment is viewed as a *sub-graph* of G or a subset of processes in the behavioral specification) in the graph are determined through scheduling and performance estimation of individual nodes or segments [12, 13, 15]. Thus for any *segment*, $s \in P_1$, the performance attributes $A(s)$, $H(s)$, $T(s)$, and $B(s)$ (area, switching activity, clock period and pin count respectively) are computed by the performance estimator built into the partitioning environment. This process is similar to the scheduling and performance estimation steps in high level synthesis [12, 15].

[*]This work was done at the University of Cincinnati and is part of the Multicomponent Synthesis System project sponsored in part by the U.S. Air Force Wright Laboratories under Contract No. F33615-91-C-1811, and the ARPA RASSP program monitored by the Wright Lab, US Air Force under contract no. F33615-93-C-1316.

We have a set of packages $p_1, p_2, p_3 \ldots p_n$ in a package library $\mathcal{L}$. Each package p has six attributes: $A(p)$, the area capacity; $H(p)$, the maximum switching activity; $T(p)$, period of the fastest clock allowed by the package; $B(p)$, the number of pins available in p; $C(p)$, the dollar cost of p; and $L(p) \geq 1$ is the level number of the package p. Level of a package is the level in the packaging hierarchy at which the package can be used. All bare-die packages are level one, ASICs and MCMs are level two, boards are level three, and so on. The *defining level* of a library is the smallest k such that no package in the library has level greater than k. For $i > 1$, packages with level i can contain only packages with level $i-1$ and level 1 packages contain the segments of the process graph. If p and q are two package instances then, $p \prec q$ denotes 'p contains q'.

Definition 2.4 For any instance, p, of a package from the package library $\mathcal{L}$:

If $2 \leq L(p) \leq k$:

(a) area cost of the package $a(p) = \sum_{p \prec q} a(q)$

(b) heat cost of the package $h(p) = \sum_{p \prec q} h(q)$

(c) pin cost of the package $b(p) = \sum_{e \in E} e$, e spans package instances p_a and p_b; such that: $(L(p_a) = L(p_b) = L(p) - 1) \wedge (p \prec p_a) \wedge (p \not\prec p_b)$

(d) clock period cost $t(p) = max_{p \prec q}(t(q))$

When $L(p) = 1$, the scheduler and performance estimator will determine the above costs based on the *level 1 segment* in p. □

Hierarchical Partitioning Problem: Given a process graph, $G = (N, E)$, a package library $\mathcal{L}$ with defining size k, and a cost constraint $\mathcal{C}$:
• find a (k-1)-level partition $\mathcal{P} = \{P_1, \ldots, P_{k-1}\}$ of G
• *Let* $P_k = \{s_k\}$; *where,* $s_k = \{s_{k-1} \mid s_{k-1} \in P_{k-1}\}$
that is, P_k contains exactly one segment (which in turn contains all the segments in P_{k-1}) to be mapped to a top most level package in the library.
• Now find a binding, $\mathcal{B}$, which for $1 \leq i \leq k$, binds each segment in P_i to some level i package instance from $\mathcal{L}$, such that
for each instance, p, of any package from $\mathcal{L}$:
$a(p) \leq A(p)$, $h(p) \leq H(p)$,
$b(p) \leq B(p)$, $t(S) \geq T(p)$.
subject to

$$Cost(\mathcal{P}) = \sum_{instance\,p} C(p); \quad Cost(\mathcal{P}) \leq \mathcal{C}.$$ □

3 The Behavior Level Hierarchical Partitioning Algorithm

The algorithm begins by partitioning the process graph and mapping partition segments onto available, cost-effective, bare-die packages. A graph is constructed from the partition generated at this level for further partitioning at the next higher level of packaging. The packaged partition segments form nodes in the new graph; edges of the current graph which connect nodes in different segments, form the edges of the new graph. At the next higher level of packaging, this new graph is partitioned and mapped onto cost-effective packages. This process continues until the packaging hierarchy is exhausted. If, at a particular level, no solution is found, we back-track to the previous level, tighten cost constraints, reconstruct the old partition and continue.

Setting constraints is an important step in the algorithm. Initially, on the first pass, overall area and switching activity constraints for the entire design are set to the minimum area and switching activity capacity of packages at the highest level in the package hierarchy (since, eventually, the design hierarchy needs to be mapped onto a package at the topmost level in the package hierarchy). The cost constraint is set by subtracting the cost of the smallest package at all levels of packaging above level 1 from the total cost constraint, $\mathcal{C}$. On subsequent invocations, if the algorithm is back-tracking, cost constraint is set by multiplying the previous iteration's cost for that level by a *constraint tighten factor* (CTF < 1). If the algorithm is not back-tracking, cost constraint is generated by subtracting the actual cost of packaging at lower levels of packaging and the projected packaging cost at higher levels (cost of smallest packages) from the total cost constraint, $\mathcal{C}$.

Algorithm 3.1 presents the hierarchical partitioning and package design algorithm (**HPP**). HPP has access to a multiway partitioning algorithm (**MP** – Algorithm 3.2). When partitioning at any level, HPP first extracts the graph $G = (N, E)$ to be partitioned using the *hierarchical netlist manager*. It also sets the cost, area, and switching activity constraints through Set_Constraint and then MP is invoked. MP explores the design space by constructing a set of alternative partitions; MP returns the first partition that satisfies constraints, or, in the absence of a constraint satisfying solution, returns the best cost solution from the set of partitions. MP returns a *status* flag along with a solution (partition with segments bound to packages). Based on the values of the *status* flag for the current and previous levels, HPP decides to proceed to the next higher level, back-track to previous level or terminate reporting failure.

The MP invokes a **K-way FM** Algorithm (KWAY – 3.3) to partition a graph into multiple segments with appropriate package bindings. K-way partitioning is carried out by repeatedly invoking *two-way* FM [11] on pairs of partition segments. To evaluate the cost of level 1 partition segments, the K-way FM invokes the *scheduler*, which estimates the performance attributes. Scheduling is the first important step in the high level synthesis process. The scheduler generates a time-stamped and partially bound data flow graph, that satisfies specified constraints. Scheduling determines execution speed of the synthesized design in terms of clock speed and number of clock cycles required to execute all operations. For a given parameterized component library, we can compute the area, average switching activity, and clock speed costs from the schedule produced by the scheduler. An implementation of Paulin's *force-directed list scheduling* [9], extended for communicating and concurrently executing processes [8], is used. Switching activity estimation technique has been reported in [7].

Algorithm 3.1 (HPP Algorithm: HierPartPack)

```
G: input graph (Behavioral specification)
P: package set
C: overall cost constraint on design
HN: hierarchical netlist manager
StatArr[k]: Status of partitioning at level k
BtkArr[k]: The number of back-tracks at level k
MaxBtk: The limit on number of back-tracks at any level
k: levels in package hierarchy, level: current level
area: overall area constraint
switch: overall switching activity constraint
cost: cost constraint at current package level

HierPartPack(G, P, C)
begin
  level ← 1      $G_{level}$ ← G      Solution ← null
  while level < k do
    Set_Constraint()
    (status, Solution) ← MP($G_{level}$, P(level), cost,
              area, switch, level)
    StatArr[k] ← status
    case status is
      SUCC:
        level ← level + 1
        HN :: read_partition(Solution)
        HN :: construct_netlist(level)
      BEST:
        if ((StatArr[level − 1] = SUCC) ∧
          (BtkArr[k] < MaxBtk)) then
            BtkArr[k] ← BtkArr[k] + 1
            level ← level − 1      /* back-track */
        else
            level ← level + 1
            HN :: read_partition(Solution)
            HN :: construct_netlist(level)
        end if
      FAIL:
        if ((StatArr[level − 1] = SUCC) ∧
          (BtkArr[k] < MaxBtk)) then
            BtkArr[k] ← BtkArr[k] + 1
            level ← level − 1      /* back-track */
        else
            return(null)
        end if
    end case
    $G_{level}$ ← HN :: read_netlist(level)
    /* retrieve next level netlist */
  end while
  return(Solution)
end
```

Algorithm 3.2 (Multiway Partitioning Algorithm)

```
G: input graph, P: package set
p: individual package from P
area: overall area constraint
switch: overall switching activity constraint
C: cost constraint on design
level: level in package hierarchy

MP(G, P, C, area, switch, level)
begin
  min_seg ← max(area/max_area(p), switch/max_switch(p))
  max_seg ← num_cell(G) /* # of nodes in graph */
  best_cost ← ∞      status ← FAIL      Solution ← null
  for num_seg = min_seg to max_seg do
    Best ← KWAY(G, P, num_seg, level)
    /* generate first partition */
    num_fm_ite ← 1      num_fm_imp ← 1
    status ← check_constraint(Best, area, switch, C)
    while (status ≠ SUCC ∧ num_fm_ite < MAX_FM_ITE ∧
      num_fm_imp < MAX_FM_IMP)  do
      S ← KWAY(G, P, num_seg, level)
      status ← check_constraint(S)
      num_fm_ite ← num_fm_ite + 1
      best_cost_kway ← cost(Best)
      if (status = SUCC) ∨ ((status = BEST) ∧
        (cost(S) < best_cost_kway)) then
          Best ← S
      end if
      if (cost(S) < best_cost_kway) then
          num_fm_imp ← 1
      else
          num_fm_imp ← num_fm_imp + 1
      end if
    end while
    if status = SUCC then
      return (status, Best)
    elsif (status = BEST) ∧ (cost(Best) < best_cost) then
      Solution ← Best
      best_cost ← cost(Best)
    end if
  end for
  return(status, Solution)
end
```

4 Multicomponent Synthesis

The multicomponent synthesis approach is demonstrated in figure 1. We integrate our hierarchical partitioning environment with a high level synthesis system to produce multicomponent designs with packaging hierarchy. First, the partitioner hierarchically partitions the input behavioral specification and binds segments at each level to appropriate packages. Multicomponent synthesis is carried out by synthesizing all level 1 partition segments (set of interacting behavioral processes) using a high level synthesis tool. We call this integrated system, **MSS** (Multicomponent Synthesis System) [10]. Design tradeoffs are performed by considering various partitions and carrying out scheduling and performance estimation on proposed partition segments. The performance attributes of the synthesized RTL designs are determined and compared against the capacity and cost constraints imposed by the packages they are bound to. Also, a *global controller* is automatically placed on a partition segment and interconnected with the RTL design segments. The global controller is placed on a partition segment whose package has the most unused space. Details of the controller model to support multicomponent partitioning are discussed in [13, 14, 16].

At the end of multicomponent synthesis and hierarchical package design we have a multicomponent design composed of interacting RTL design segments. The behavioral partitioning phase produces multiple behavior segments that are completely synthesized to RTL designs using a high level synthesis system such as DSS [12, 13]. Also produced is a hierarchical structural design (the leaf nodes in this design are the individual RTL designs) that is mapped onto efficient cost-effective packages from a package library. We functionally validate our approach by simulating the hierarchical RTL design and the input behavior for the same set of test vectors and comparing their outputs.

5 Results

We present results for a number of examples to demonstrate the validity of our behavioral partitioning approach for multicomponent synthesis and hierarchical package design. Details of a few packages from our library is shown in Table 1. Table 2 presents details of the number of lines of code in behavior level VHDLspecification and the number of processes for each of our examples.

Algorithm 3.3 (k-way FM Algorithm: KWAY) .

```
G: graph $G = (V, E)$
P: available set of packaging options
S: $\{s_1, s_2, \cdots, s_n\}$ a partition of G with k segments

KWAY(G, P, k, level)
begin
  Best ← initialize()   /* create initial partitions */
  if level = 1 then    /* pure behavior specification
                        - estimate attributes */
     for all $s \in Best$ do
        Schedule/Performance Estimate s
        and generate A(s), H(s), B(s), and T(s)
     end for
  end if
  best_cost ← 0     S ← null     cont_part ← TRUE
  ite_cnt ← 1     imp_cnt ← 1
  for all $s \in Best$ do     /* map partition segment
                             to package and find cost */
     best_cost ← best_cost + cost($\mathcal{B}$(s))
  end for
  while cont_part = TRUE do
     for i = 1 to k−1 do
        for j = i+1 to k do
           $two\_way\_fm(s_i, s_j)$
        end for
     end for
     if level = 1 then   /* pure behavior specification
                          - estimate attributes */
        for all $s \in S$ do
           Schedule/Performance Estimate s
           and generate A(s), H(s), B(s), and T(s)
        end for
     end if
     curr_cost ← 0
     for all $s \in S$ do      /* map partition segment
                            to a package in P and find the cost */
        curr_cost ← curr_cost + cost($\mathcal{B}$(s))
     end for
     ite_cnt ← ite_cnt + 1
     if curr_cost < best_cost then
        imp_cnt ← 1      Best ← S
        /* save best partition seen so far */
     else   imp_cnt ← imp_cnt + 1    end if
     if ite_cnt = MAX_ITE ∨ imp_cnt = IMP_CNT
       then   cont_part ← FALSE    end if
  end while
  return(Best)          /* retrieve best partition */
end
```

Move Machine: The instruction set of the Move Machine controls instruction and data flow. It does not compute any data values. ALU operations are assumed to be memory mapped. **Fifo:** Fifo models a producer consumer problem. **Shuffle:** The Shuffle is a high speed reconfigurable 32 bit shuffle-exchange network for parallel signal processing. The Shuffle exchange is a commercial product of Texas Instruments, Inc. **dyn** is a five process description that monitors and maintains the dynamic length and maximum length to which a queue in a producer-consumer problem grows. **alu** is a nine process description of an arithmetic logic unit. *dyn1-dyn10* and *alu1-alu5* are multiple processing elements generated by making multiple instantiations of dyn and alu respectively.

5.1 Multicomponent Synthesis and Hierarchical Package Design

Tables 3 and 4 present results of multicomponent synthesis and hierarchical package design for the design examples in Table 2 with the package library shown in Table 1. For the smaller examples (Move Mc - dyn2), Table 3 presents the mapping of segments to packages at three levels of design hierarchy. Due to lack of space

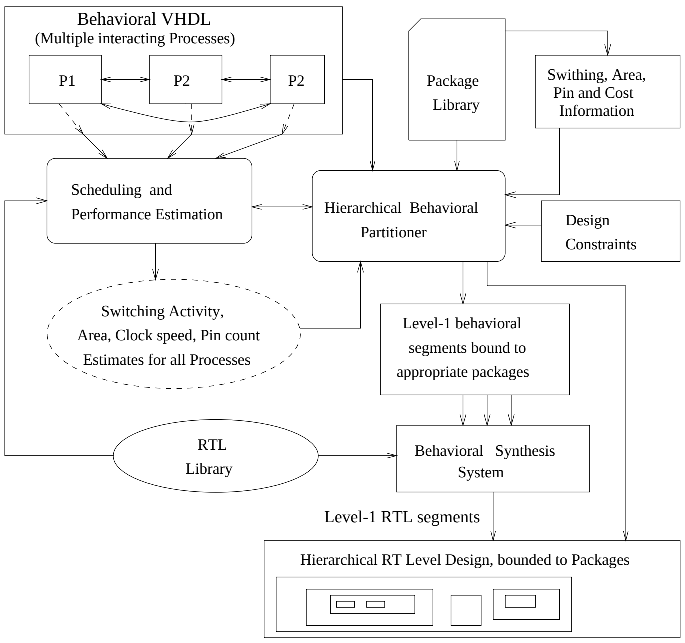
Behavioral VHDL
(Multiple interacting Processes)
P1
P2
P2
Package Library
Swithing, Area, Pin and Cost Information
Scheduling and Performance Estimation
Hierarchical Behavioral Partitioner
Design Constraints
Switching Activity, Area, Clock speed, Pin count Estimates for all Processes
Level-1 behavioral segments bound to appropriate packages
RTL Library
Behavioral Synthesis System
Level-1 RTL segments
Hierarchical RT Level Design, bounded to Packages

Figure 1. Hierarchical Behavioral Partitioning for Multicomponent synthesis

we are not able to show this result for the other larger designs. Table 4 presents the following results for all designs in Table 2: (1) Number of back-tracks taken by the algorithm / BTK(Max. back tracks allowed); (2) Actual design cost/constraint; and (4) execution time.

For each example, the cost constraint was progressively tightened until the algorithm failed to find a cost-satisfying solution. In all cases, if a constraint-satisfying solution existed, it was discovered by the algorithm. For smaller examples, this was verified by manual examination. The results establish the validity of the algorithm. An interesting observation that vindicates our choice of the back-tracking algorithm is that in all our examples the most times the algorithm ever back-tracks is three (Table 4). This is because the algorithm back-tracks only if it can potentially find a solution with better cost and, also, the algorithm converges to a constraint-satisfying solution fairly rapidly.

Hierarchical RTL Partitioning: We also developed a Hierarchical RTL partitioner [14] as an alternate approach. Here, we synthesize the input behavior and then partition the resulting RTL design. Table 5 presents the results of the hierarchical RTL partitioning for the designs in table 2. Blanks indicate that the input design was too large to be handled by the RTL partitioner. For each example, the dollar cost solution is bold-faced if it is better than the behavioral counterpart. RTL partitioning yields better designs for smaller examples where the number of synthesized RTL components is relatively small (< 200). For larger examples multicomponent synthesis executes much faster and also out-performs RTL partitioning in terms of the quality of solutions produced. The behavioral partitioner produced better quality results faster than RTL partitioner because papritioning at the process level and following our multicomponent synthesis approach

L(p)	Name	A(p)*	B(p)	H(p)+	T(p)−	C(p)#
1	Tiny1	5	40	50	50	400
1	Tiny2	5	40	60	50	500
1	Small1	15	40	150	50	800
1	PGA-2	15	84	300	50	1300
1	PGA-6	20	169	1000	50	1800
2	PI-1	6	40	50	50	250
2	Cer-1	15	40	200	50	500
2	PGA-1C	12	84	220	50	800
2	PGA-5C	20	169	1000	50	1500
2	MCM-3	400	169	3000	75	20000
3	Board-1	300	80	2000	100	300
3	Board-2	400	80	3000	100	400
3	Board-6	1000	128	12000	100	1200

* : sq. mm; + : 1000 node switches; - : ns; # : $

Table 1. Package Alternatives

Example	Num Lines (VHDL)	Num Proc
Mv Mc	75	3
Fifo	65	3
Shuffle	472	5
dyn1	132	5
dyn2	254	10
dyn3	376	15
dyn4	498	20
dyn5	620	25
dyn6	742	30
dyn7	864	35
dyn8	986	40
dyn9	1108	45
dyn10	1230	50
alu1	100	9
alu2	188	18
alu3	276	27
alu4	364	36
alu5	452	45

Table 2. Design Data for Examples

avoids the various drawbacks of RTL partitioning as mentioned in section 1.

Hierarchical Package Design without Integrated Scheduling: Since scheduling and performance estimation are time consuming, we modified KWAY-FM by replacing the schedule and performance estimation steps by approximations for area and switching activity. In this approach, individual processes are first scheduled and performance estimated. Then, for level 1 segments, the area and switching activity costs of the individual processes in the segment are summed to obtain the total area and switching activity of the overall segment. These numbers are then adjusted by a small percentage (10-30%) to take into account the possible sharing of resources if the processes had been actually scheduled together[14]. Table 6 presents results of hierarchical partitioning and package binding without an integrated scheduling and performance estimation step.The better dollar cost for each example is bold-faced. *Invalid* indicates that at least one of the partition segments at level 1 does not fit on available packages; thus, the design is not valid. The approach with scheduling out-performs the approximation method, especially for the larger designs. However, (a) execution time for the approximation method is very small; and (b) the estimated cost of packaging the designs are fairly close to the solutions reported by the algorithm with embedded scheduling algorithm. This observation indicates that the approximation algorithm should be used to quickly generate approximate dollar cost constraints to be imposed on the rigorous algorithm.

Example	Segments and Mapping (s_i–p_i) Level-3	Level-2	Level-1
Mv Mc	s_{21}–Board-1	s_{11}–PGA-5C	s_1–PGA-6 EXE
		s_{12}–PGA-1C	s_2–PGA-1 FET, DEC
Fifo	s_{21}–Board-1	s_{11}–PI-5	s_1–Small1 FIFO PRODUCER CONSUMER
Shuffle	s_{21}–Board-2	s_{11}–PGA-4C	s_1–PGA-4 shuffle-1
		s_{12}–PGA-4C	s_2–PGA-4 shuffle-2
		s_{13}–PGA-4C	s_3–PGA-4 shuffle-3
		s_{14}–PGA-4C	s_4–PGA-4 shuffle-4
		s_{15}–PGA-4C	s_5–PGA-4 output
dyn1	s_{21}–Board-1	s_{11}–Cer-3	s_1–Small3 s1_p_1,s1_p_pt s1_p_sl,s1_p_2 s1_p_st
alu1	s_{21}–Board-1	s_{11}–Cer-2	s_1–PGA-1 s1_nbp,s1_nap s1_np,s1_outp
			s_2–Tiny1 s1_mp,s1_ap s1_op
		s_{12}–PI-1	s_3–Tiny1 s1_dp,s1_sp
dyn2	s_{21}–Board-1	s_{11}–Cer-3	s_1–Small-1 s2_p_sl,s2_p_pt s2_p_2
			s_2–Tiny1 s2_p_st,s1_p_st
		s_{12}–PI-5	s_3–Small1 s1_p_sl,s1_p_pt s1_p_1,s1_p_2

Table 3. Multicomponent Synthesis with Hierarchical Package Design Results

Note: s–p denotes the mapping of segment s onto package p from the package library. Also, at level 1, number of processes on each partition segment are presented.

6 Conclusions and Discussion

We have presented a hierarchical behavioral partitioning and package design algorithm. We demonstrated a methodology to integrate our partitioner with a high level synthesis tool to create a multicomponent synthesis and hierarchical package design environment, MSS (Multicomponent Synthesis System) [10]. MSS takes as input a multi process VHDL behavior, a parameterized component library, a package library, and an overall cost constraint on the design and generates a hierarchical RTL design while simultaneously constructing a physical package hierarchy for the design.

We presented results to evaluate the performance of the approach with respect to the quality of designs produced and execution times for a number of design examples. Hierarchical RTL partitioning and package design yields good results for examples where the number of RTL components in the synthesized design are less than 200. When partitioning at the RTL netlist level, the design architecture is frozen (during high level synthesis). Alternate multichip designs cannot be explored during hierarchical RTL partitioning, whereas

MSS explores the design space by considering alternate implementations during high level synthesis. Also, thermal profiling of RTL designs is too time consuming and is not viable for large designs. For almost all the examples, MSS produces better results and executes much faster than the hierarchical RTL partitioning. For smaller designs, scheduling overhead can be reduced through approximate estimation procedures to evaluate the cost of level 1 segments form individual process costs. From the results, we infer that the hierarchical behavioral partitioning is both a suitable and a viable approach to multicomponent synthesis and hierarchical packaging.

References

[1] M.C. McFarland, A.C. Parker, and R. Camposano, "Tutorial on High-Level Synthesis," *Proc. 25th Design Automation Conference,* pp. 330–336, June 1988.

[2] M.C. McFarland, A.C. Parker, and R. Camposano, "The High-Level Synthesis of Digital Systems," *Proc. of the IEEE,* Vol. 78, No. 2, pp. 301–318, Feb. 1990.

[3] R. Camposano, "From Behavior to Structure: High-Level Synthesis," *IEEE Design & Test of Computers,* pp. 8–19, Oct. 1990.

[4] K. Kucukcakar, "System-Level Synthesis Techniques With Emphasis on Partitioning and Design Planning," *Ph.D. Dissertation,* Dept. of Electrical Engineering-Systems, University of Southern California, CA, Oct. 1991.

[5] F. Vahid and D.D. Gajski, "Specification Partitioning for System Design," *Proc. 29th Design Automation Conference,* pp. 219–224, June 1992.

[6] R. Gupta and G. De Micheli, "Partitioning of Functional Models of Synchronous Digital Systems," *Proc. ICCAD-90,* Santa Clara, pp. 216–219, Nov. 1990.

[7] Nand Kumar, Srinivas Katkoori, Leo Rader and Ranga Vemuri, "Profile-Driven Behavioral Synthesis for Low Power VLSI Systems", *IEEE Design & Test of Computers*, pp. 70-84, Fall 1995.

[8] R. Dutta, "Distributed Design-Space Exploration for High-Level Synthesis Systems," *Master's Thesis,* Dept. of Electrical and Computer Engineering, University of Cincinnati, OH, 1991.

[9] P.G. Paulin and J.P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Trans. Computer-Aided Design,* Vol. 8, No. 6, pp. 661–679, June 1989.

[10] R. Vemuri et al, "An Integrated Multicomponent Synthesis Environment for Multichip Modules," *Computer,* pp. 62–74, April 1993.

[11] C.M. Fiduccia and R.M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *Proc. 19th Design Automation Conference,* pp. 175–181, June 1982.

[12] J. Roy, N. Kumar, R. Dutta, and R. Vemuri, "DSS: A Distributed High-Level Synthesis System," *IEEE Design & Test of Computers,* pp. 18–32, June 1992.

[13] J. Roy, "Parallel Algorithms for High-Level Synthesis," *Ph.D. Dissertation,* Dept. of Electrical and Computer Engineering, University of Cincinnati, OH, Feb. 1993.

[14] N. Kumar, "High Level VLSI Synthesis for Multichip Designs" *Ph.D. Dissertation,* Dept. of Electrical and Computer Engineering, University of Cincinnati, OH, Oct. 1994.

[15] R. Dutta, J. Roy, and R. Vemuri, "Distributed Design-Space Exploration for High-Level Synthesis Systems," *Proc. 29th Design Automation Conference,* pp. 644–650, June 1992.

[16] N. Narasimhan, J. Roy, and R. Vemuri, "Synchronous Controller Models for Synthesis from Communicating VHDL Processes," *Proc. Ninth International Conference on VLSI Design,* pp. 198-204, Jan. 1996.

Example	Num BkTrk/ BTK	Cost/Constraint ($)	Exec Time (s)
Mv Mc	1/10	5600/5000	6
Fifo	0/10	1550/3000	2.7
Shuffle	0/10	13900/1200	59.8
dyn1	1/10	1900/2000	3.6
alu1	1/10	3100/2500	100.7
dyn2	2/10	3350/3200	212.7
dyn3	0/10	5000/5000	126.1
alu2	1/10	6700/5000	412.8
dyn4	0/10	6350/800	229.3
dyn5	0/10	8350/8000	349.5
alu3	0/10	12700/8000	579
dyn6	1/10	9850/9000	1470.7
dyn7	2/10	11200/10000	3141
alu4	3/10	14100/15000	1549.4
dyn8	1/10	11850/12000	1863.5
dyn9	1/10	13800/13000	3684.1
alu5	2/10	17750/18000	1626.4
dyn10	2/10	16850/15000	6452.2

Table 4. Hierarchical Behavioral Partitioning and Package Design

Example	Num RTL Comp	Btk/ BTK	Cost ($)	Exec Time (s)	Cost ($) Constr.
Mv Mc	53	0/10	**4250**	13.2	5000
Fifo	76	0/10	1750	6.4	3000
Shuffle	379	-	-	-	12000
dyn1	128	0/10	**1550**	11.9	2000
alu1	65	0/10	**1900**	6.5	2500
dyn2	234	0/10	6200	6560	3200
dyn3	334	0/10	53000	113272	5000
alu2	123	0/10	**5400**	2976	5000
dyn4	-	-	-	-	8000
dyn5	-	-	-	-	8000
alu3	161	0/10	**10850**	6251	8000
dyn6	-	-	-	-	9000
dyn7	-	-	-	-	10000
alu4	205	0/10	53600	109850	15000
dyn8	-	-	-	-	12000
dyn9	-	-	-	-	13000
alu5	-	-	-	-	18000
dyn10	-	-	-	-	15000

Table 5. Hierarchical RTL Partitioning and Package design

Example	Btk/ BTK	Cost ($)	Exec Time (s)	Cost ($) Constr.
Mv Mc	1/10	6500	3	5000
Fifo	0/10	1550	1.1	3000
Shuffle	0/10	13900	29.8	12000
dyn1	0/10	1900	1.4	2000
alu1	0/10	3550	11.3	2500
dyn2	1/10	3600	9	3200
dyn3	0/10	Invalid	5.8	5000
alu2	1/10	6800	76.2	5000
dyn4	0/10	7150	10.3	8000
dyn5	0/10	Invalid	12.4	8000
alu3	1/10	**11250**	248.9	8000
dyn6	0/10	Invalid	26	9000
dyn7	1/10	11850	252.5	10000
alu4	1/10	Invalid	77.8	15000
dyn8	1/10	Invalid	438.9	12000
dyn9	2/10	Invalid	708.1	13000
alu5	1/10	Invalid	1092	18000
dyn10	2/10	Invalid	875	15000

Table 6. Partitioning without Scheduling