Synthesis of Application Specific Programmable Processors

Kyosun Kim & Ramesh Karri
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst MA 01003

Miodrag Potkonjak
Department of Computer Science
University of California
Los Angeles CA 90095

Abstract: Synthesis of Application Specific Programmable Processors poses numerous new tasks on behavioral synthesis tools. We address some of them including **application bundling**. **Application Bundling** is a synthesis task where n control-data flow graphs are bundled into at most m groups, so that each application belongs to at least one group and throughput constraints for all applications are satisfied.
We have shown how a variety of application specific constraints such as manufacturing cost reduction and production risk reduction can be targeted during the synthesis process. The effectiveness of our approach is demonstrated on a number of real examples.

1 Introduction

The market which supplies integrated circuits (ICs) to the application specific consumer electronics, industrial electronics and communication and computer products has been historically sharply divided into two groups. While general purpose platforms (such as microprocessors, digital signal processors, video signal processors and microcontrollers) are highly flexible and have lower design turn-around times, dedicated ASICs are characterized by higher levels of achievable performance and low power. However, neither general purpose engines nor dedicated ASICs can, by themselves, provide the implementation properties required by modern electronic products. In fact, while multiplicity of standards, diverse quality-of-servers offerings, and rapidly changing transmission requirements mandate flexibility, portability and mobility imply a need for low power.
Over the last ten years field-programmable gate arrays (FPGA) have become a popular implementation medium. In fact, they are often the implementation medium of choice for control intensive applications that require low turn-around times, limited flexibility, and relatively high performance. FPGAs have caught the attention of the CAD research and development community [6]. Importance of application specific programmable processors (ASPP) market is underscored by a growth in the line of products offered by major semiconductor companies. For example, Motorola offers numerous DSP ASPPs [7].

DAC 97, Anaheim, California

Some important application scenarios wherein ASPPs are superior to both fully programmable and fully dedicated implementations are as follows. Modern applications usually require high performance, low power, and inexpensive multiple functionalities. Also, multiple standards may exist for the same service; code-division multiple access and time-division multiple access in wireless communications and NTSC, PAL, and SECAM standards for television. ASPP designs preserve all of the advantages of dedicated ASICs, while providing the demanded flexibility. It is possible to envision the use of ASPPs for power management in portable applications. ASPP-based designs provide the user with an option to adapt computations to a dynamically changing environment. For example, a power saving mode can be invoked by trading-off the speed of computation and communication or the quality-of-service.

2 Related Research

Reconfigurable computing is attracting a lot of attention recently. A fast growing billion dollar FPGA industry is supported by a number of commercial and research CAD tools [6]. A number of special purpose reconfigurable computers have been built. Early work in this direction includes the Splash [5] system. The Splash system enables reconfigurability to more than 100 different configurations which are well suited for several computational tasks in molecular biology. Several generations of data path reconfigurable video-processors with accompanying compilation support has been developed at UC Berkeley [14].

Behavioral synthesis has been an active research area for more than two decades [1, 4]. Recently, application specific instruction set processors (ASIP) [2, 3] received a great deal of attention. ASIPs are different from ASPPs in that they provide greater flexibility at the expense of low performance, higher cost and power, and a need for compilation support development.

One of the first partitioning techniques during high-level synthesis was proposed by McFarland [11]. The technique considered similarity of functions, common data carriers, and low level parallelism within single-stage clustering procedures. More recently, Lagnese and Thomas [10] generalized this work by considering multi-stage clustering for area reduction in behavioral synthesis. Other notable behavioral partitioning work includes [9, 12]. Application bundling differs in several ways from the traditional high level synthesis

partitioning. Partitioning distributes components of a single computational task across several ICs so as to optimize an objective such as the number of interconnections between the ICs. On the other hand, bundling is carried out on completely independent computations in such a way that cost (e.g. area or power) is minimized.

3 ASPP Synthesis: Motivation

Consider three application control data flow graphs (CDFG) shown in figure 1. Furthermore, assume that all operations finish in a single cycle and that the applications have identical word lengths and have to be implemented in three clock cycles. If implemented as a

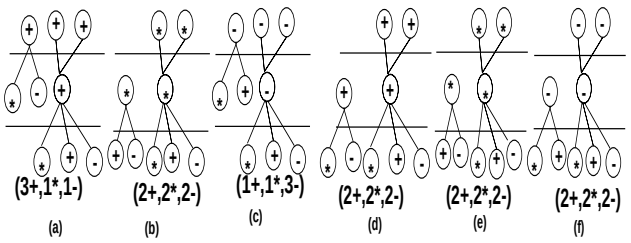

Figure 1: ASPP synthesis: a motivating example

dedicated ASIC, the first CDFG (shown in figure 1(a)) requires a resource allocation of **five functional units** namely {three adders, one multiplier, one subtract}. Similarly, the second and third CDFGs (shown in figures 1(b)-(c)) can be implemented as dedicated ASICs with resource allocations of {two adders, two multipliers, two subtracts} and {one adder, one multiplier, three subtracts} respectively.

A straightforward approach to implementing the three applications as a single ASPP involves superimposing the dedicated architectures. For example, since CDFG 1(a) requires three adders and since this is also the maximum number of adders required by any application, the synthesized ASPP will include at least three adders. Similarly, the ASPP will include at least three multipliers (due to CDFG 1(b)) and three subtracts (due to CDFG 1(c)). These maximum hardware requirements of individual CDFGs translate into an overall resource allocation of {three adders, two multipliers, three subtracts} for the synthesized ASPP.

Resource requirements of the ASPP can be reduced significantly by maintaining a global view of the resource requirements of all CDFGs at all times during the synthesis process. In fact, the three CDFGs introduced earlier can be implemented as an ASPP using a hardware allocation of {two adders, two multipliers, two subtracts} (a total of only **six functional units**) as shown in figure 1(d)-(f). Observe that **the allocated hardware is twenty five percent less when compared to the strategy described previously.**

4 Computational/Hardware Models

Our computational model for a single application is homogeneous synchronous data flow [8]. Within this model, a task is represented as a hierarchical CDFG $G(N, E, T)$, with nodes N representing the CDFG operations, and the edges E and T respectively the data and timing dependences between the operations.

In modern designs a variety of register file models have been used [13]. From among them we have selected the dedicated register file hardware model. This model clusters all registers in register files and each file is then connected only to the inputs of the corresponding execution units. An important benefit of the chosen hardware model is that it reduces the interconnect at the expense of additional registers.

5 ASPP Synthesis: Outline

Designing an ASPP for an incompatible set of applications will often times be counter-productive. For example, while grouping topologically incompatible computations on a single chip can translate into significant interconnect overhead, applications with incompatible hardware types can entail significant execution unit overhead. Consequently, applications with similar computation topology, hardware types, word length requirements, etc, are identified and **bundled** into compatible groups. Following application bundling, each bundle is synthesized into a separate ASPP. Since there are several applications in a bundle and since synthesis of an ASPP by simultaneously considering all the applications and their respective constraints is difficult, we propose to synthesize the ASPP from a bundle by considering one application at a time. Within such a methodology **ordering of applications** impacts the hardware overhead of the resulting ASPP. Ordering of applications within a bundle is followed by a novel **allocation, assignment and scheduling** of the applications in a bundle. These steps take into account (i) the allocated hardware due to the previously synthesized applications, (ii) the estimated hardware of the yet-to-be synthesized applications and (iii) the hardware required by the candidate application.

5.1 Application Bundling

During **application bundling** n CDFGs are bundled into at most m groups, so that each application belongs to at least one group and that some objective function is optimized. A myriad of issues should be considered during bundling.

Compatibility of Application Topologies: If applications with disparate topologies are implemented as an ASPP the attendant interconnect overhead will be significant. In the worst case, each hardware unit is connected with every other hardware unit, increasing the number of buses and multiplexers. Consequently, topological similarity between applications should be considered during bundling. In figure 2, application 2 can be bundled into an architecture implementing application 4 with almost no hardware overhead. This is because, the CDFG 2 is identical to a sub graph of the CDFG 4.

Resource Compatibility is an important issue during ASPP synthesis. For example, in figure 2, while applications 2, 3 and 4 use subtracts and multipliers,

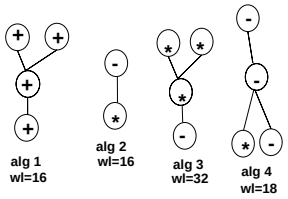

Figure 2: Illustration of important issues in ASPPs

application 1 uses adders. Consequently, bundling application 1 with either 2 or 3 or 4 does not yield justifiable benefit. In fact, it entails additional hardware overhead from the point of view of the bundled applications. On the other hand, based on the compatibility of their hardware types, applications 2, 3, and 4 are good candidates to be bundled in to an ASPP.

Word length Compatibility: Most synthesis systems circumvent the problem of disparate word length requirements within an application by assuming uniform word lengths for all hardware used in the implementation. Such a uniform word length assumption is justifiable during dedicated ASIC synthesis based on a simplicity-of-implementation argument. However, it is untenable within the context of ASPP synthesis. Firstly, individual applications may have different precision requirements. Furthermore, assuming a uniform word length across all applications in a bundle entails an enormous hardware overhead.

Impact of Precision on Interconnect Overhead: Two data transfers can be merged onto the same bus without any multiplexer overhead if the length of the fractional parts of the destinations are identical.

The Bundling Framework: A probabilistic rejectionless framework is used for application bundling. First, applications are bundled randomly. Based on the incompatibility between the applications and the bundles, the algorithm proceeds in a way similar to probabilistic iterative techniques. A source bundle is randomly chosen, probabilistically favoring bundles with incompatible applications. From such a bundle, an incompatible application is probabilistically selected and moved to another bundle where applications are compatible with the selected application. The hardware area of all bundles is then computed, and the current bundling configuration is saved if it is the best one so far. This continues until no more improvement is obtained for a given number of iterations.

5.1.1 Minimization of Manufacturing Cost

The manufacturing cost of implementing an application i as a dedicated ASIC includes the fixed and production volume dependent costs. The fixed costs include the non-recurrent engineering cost ($NREC_i$) and the cost of engineering for design, test and verification(EC_i). Assume that the cost of a unit area of silicon implementation is α and the area of the ASIC implementing the application i is $Area_i$. The cost of manufacturing x_i units of the ASIC is $NREC_i + EC_i + \alpha \cdot Area_i \cdot x_i$. The total cost of manufacturing n applications as dedicated ASICs is,

$$\sum_{i=1}^{n} \{NREC_i + EC_i + \alpha \cdot Area_i \cdot x_i\}$$

Alternately, consider implementing $\sum_{i=1}^{n} x_i$ ASPPs each of which supports these n applications. Of these data paths, x_1 can be configured to implement application 1, x_2 can be configured to implement application 2, and so on. Let $NREC_{ASPP}$, EC_{ASPP} be the non-recurrent and other engineering costs associated with the manufacturing of these ASPPs. The cost of manufacturing these $\sum_{i=1}^{n} x_i$ ASPPs is:

$$NREC_{ASPP} + EC_{ASPP} + \alpha \cdot Area_{ASPP} \cdot \sum_{i=1}^{n} x_i$$

There is a point (in the number of ASPPs) at which the manufacturing cost of implementing the tasks as an ASPP equals the manufacturing cost of implementing each of the tasks as a dedicated ASIC. This is called the **break even point** and is a function of the relative proportion β_i $(= x_i / \sum_{j=1}^{n} x_j)$ of the constituent applications in the production lot. Assume that the engineering costs of an ASPP is equal to the sum of the engineering costs of implementing the constituent applications as dedicated ASICs. Also assume that the non recurrent engineering cost of the ASPP and the dedicated ASICs are identical. The break even point (BE_{ASPP}) then becomes:

$$BE_{ASPP} = \frac{1}{\alpha} \cdot \frac{(n-1) \cdot NREC}{Area_{ASPP} - \sum_{i=1}^{n} \beta_i \cdot Area_i}$$

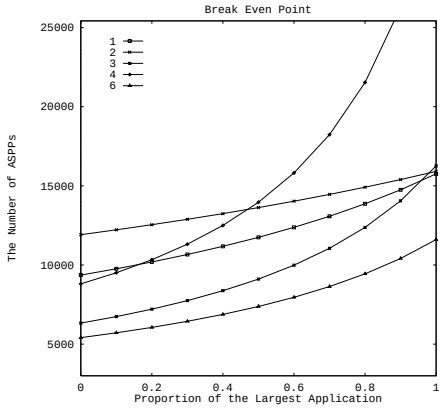

Figure 3: A break even analysis of ASPP designs

In figure 3, the break even point (below which an ASPP implementation is more economical than the dedicated implementations) is plotted as a function of β_i for five ASPPs. Even when such a simple analysis is used, the break even point between the ASPP and the dedicated implementations is of the order of thousands of units. The break even points in figure 3 are obtained assuming $\alpha = 0.5$ \$/mm^2, NREC = \$30,000. A more rigorous

analysis has confirmed that these break even values are of the order of tens of thousands of units. The area of the ASPP should not be so large as to increase its variable costs beyond the savings in the fixed initial cost. Therefore, the overall area of ASPPs must be minimized. Let $h_t(g)$ be the number of units of type t and $Area_t$ be the area of a unit of type t. If bundle B_i has n_i applications, then the cost of manufacturing these ASPPs (assuming equal quantities of each application) is:

$$Cost_i = n_i \cdot \sum_{t \in T} \max_{g \in B_i} h_t(g) \cdot Area_t$$

Problem: Given N applications, each with its own execution time bound and hardware requirements when implemented as a dedicated ASIC, partition the applications into M bundles minimizing the overall area of the ASPPs programmed for the N applications.

An application bundle will have the smallest area if the applications in the bundle are compatible with one another. This is possible if (i) they are topologically similar, (ii) have identical hardware usages, and (iii) have similar word lengths requirements.

Based on these observations we have developed a measure to identify bundles with incompatible applications. $\sum_{t \in T} h_t(g_j) \cdot Area_t$ is an estimate of the area of an application, g_j, and $\sum_{t \in T} \max_{g \in B_i} h_t(g) \cdot Area_t$ is an estimate of the area of a bundle, B_i. The larger the difference between these areas, the more incompatible the application is with the remaining applications in the bundle. This incompatibility can be obtained as:

$$Incompatibility_{i,j} = \sum_{t \in T} \mid \max_{g \in B_i} h_t(g) - h_t(g_j) \mid \cdot Area_t$$

and is used to weight the candidate solutions.

5.1.2 Bundling for Risk Reduction

In order to reduce the risk of overproduction and lost revenues, ASPPs can be used to piggyback smaller, compatible applications onto a large, primary application. If the primary application succeeds in the marketplace, all of the ASPPs can be programmed to execute this application. However, if the primary application does not yield the expected revenues, the ASPPs can be programmed to execute alternative applications.

The area of the ASPP must not be much larger than that of the dedicated ASIC implementation of the primary application. If g^i_{prim} is the primary application in a bundle B_i, then the area overhead of the ASPP is:

$$Overhead_i = \sum_{t \in T} \max_{g \in B_i} h_t(g) - h_t(g^i_{max}) \cdot Area_t$$

The first term is the maximum hardware requirement of type t in bundle B_i, and the second term is the hardware requirement of type t of g^i_{prim}. The incompatibility between bundle B_i and any application g_j is:

$$OverheadDist_{i,j} = \sum_{t \in T} \max(0, h_t(g^i_{max}) - h_t(g_j)) \cdot Area_t$$

and is used to weight the candidate solutions.

5.2 ASPP Allocation and Scheduling

The ASPP assignment, allocation and scheduling algorithm is outlined in figure 4. An initial allocation, A for the bundle, G is derived in step 1. Beginning with the most critical application, a feasible ASPP solution for the entire bundle is obtained in steps 2-8. From the total hardware allocated to the bundle, the hardware allocation T for the candidate application is obtained in step 3. Steps 4-6 constitute the synthesis loop. Assignment and scheduling of the candidate application are carried out using this allocation. If the allocated hardware is not sufficient, it is upgraded. The upgraded hardware in T is reflected in the overall allocation in step 8. In step 7, any subset of the current allocation is checked for feasibility. Since the criticality of the applications changes dynamically with the changes in allocation, application ordering is included in the loop. In the global redundancy removal phase, the applications are also ordered dynamically, and **downgrading** of a hardware type aborts as soon as an application fails to tolerate it.

```
G={g |applns in bundle}, A={a | hardw alloc}
T={t | temp hardw alloc}, C[type] = hardw criticality
ASPPSynthesis(G) {
1: A ← InitialAllocation(G, φ)
 /* Find a Feasible Solution */
2: while ((g ←ApplicationOrdering(G, ∀types, A)) ≠ φ)
3:  T ← InitialAllocation({g}, A)
4:  while (AssignAndSchedule(g, T, &C) = FAIL)
5:   type ← MostCritHWTypeToUpgrade(C)
6:   T ← T ∪ { a ← Upgrade(g, A, type) }
7:  RemoveRedundancy(g, T, C)
8:  A ← A ∪ T
 /* Remove Global Redundancy */
9: while ((type ←LeastCritHWTypeToDowngrade(C)) ≠ φ)
10:  while ( (g ← ApplicationOrdering(G, type, A) ) ≠ φ)
11:  T ← Downgrade({g}, A, type)
12:  if (AssignAndSchedule(g, A, &C) = FAIL) break
13:  if (g = φ) A ← Downgrade(G, A, type) }
```

Figure 4: ASPP assignment, allocation & scheduling

5.2.1 Initial Allocation

Let a^i_j be the minimum bound on the necessary amount of hardware of each type j for the i^{th} application of a bundle. For each hardware type j and for each application i of a bundle, relaxation based scheduling techniques are used to derive an estimate of a^i_j. For an application bundle, a global min bound $a_j = \max_{i \in G} a^i_j$ is then used as the initial allocation for the j^{th} hardware type. This is because there will be **at least** one application in the bundle that requires at least these many hardware units of type j. For example, if the three applications in figure 1 are implemented as a single ASPP, the minimum hardware requirements are two adders (due to application 1), two multipliers (due to

application 2) and three subtractors (due to application 3).

5.2.2 Application Ordering

A good synthesis solution can be found by scheduling applications one after the other. The order of applications is important because if the applications detecting the allocation shortage are scheduled first, the synthesis process can terminate early. Intuitively, applications with either critical hardware requirements or with expensive hardware requirements should be scheduled first. Also, applications that are less critical vis-a-vis their hardware requirements can exploit the extra hardware afforded by those applications that have been scheduled previously.

5.2.3 Hardware Criticality

We have developed a hardware criticality measure that accounts for the fact that the slack time of a node affects those of its adjacent nodes. And this effect propagates through the CDFG. Nodes that create slack and nodes that consume slack are identified for each node and used to calculate their mobilities as follows:

$$M_i = (S_i - \min_{j \in fanin(i)} S_j) + (S_i - \min_{j \in fanout(i)} S_j)$$

where S_i is the slack of node i. If node i creates slack time and propagates it to the ancestors (descendants), the first (second) difference in the equation is positive. If the node consumes the slack time propagated from its ancestors (descendants), it is negative. The hardware criticality of type t is then computed as $C_t = -\sum_{type(i)=t} M_i$.

5.2.4 Hardware Upgrading/Downgrading

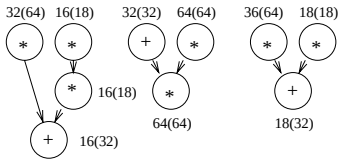

Figure 5: Upgrading and Downgrading

Within the context of ASPP synthesis, a straightforward allocation scheme that starts with an initial allocation and adds hardware until a feasible solution is found will result in excessive hardware overhead. For example, consider the three CDFGs shown in figure 5 annotated with their word length requirements. Assuming that these CDFGs are scheduled in three clock cycles, a straightforward implementation of an ASPP will require four adders (with word lengths 32, 32, 18, and 16) and four multipliers (with word lengths 32, 32, 18 and 16).

Uniform word length assumption used previously in literature is expensive as well. In the above example, the uniform word length assumption results in two 32-bit adders and two 64-bit multipliers. However, by observing that a higher precision hardware unit can be allocated to a lower precision node, the hardware overhead can be reduced to two multipliers (with word lengths 18 and 64) and two adders (with word lengths 18 and 32). In the process, 14 adder bits and 46 multiplier bits are saved. Based on these observations we have developed a parsimonious ASPP hardware allocation strategy called **upgrading**. First, a unit of type j with a larger word length is selected. Otherwise, a unit of type j is selected and its word length **upgraded**. Otherwise, a new hardware unit is added. **Downgrading** is the exact opposite of **upgrading**.

6 Experimental Results

no	appln	(N,E)	word len'	crit' path	avail time	area (mm^2)
1	8IIR	(41,54)	23	18	21	38.50
2	ADAPT	(24,24)	16	6	9	21.91
3	ARAI	(51,63)	22	8	10	34.77
4	CASC	(34,47)	12	10	10	11.34
5	DIF	(57,66)	22	10	12	30.99
6	DIR	(124,138)	22	9	22	53.05
7	DIT	(58,70)	22	9	11	43.39
8	FFT8	(30,31)	16	5	6	11.21
9	FIR20	(32,42)	16	3	12	18.03
10	GM1M	(20,27)	20	14	14	21.81
11	IIR7	(29,39)	24	10	10	32.26
12	IIR8	(39,57)	11	9	12	9.97
13	LDILP	(14,18)	16	6	6	9.31
14	LEE	(57,66)	22	10	12	31.51
15	MCM	(102, 111)	22	9	12	35.26
16	PR1	(56,65)	22	8	20	52.16
17	PR2	(66,73)	22	9	10	38.50
18	VOLT	(30,39)	24	12	12	28.47
19	WANG	(56,65)	22	8	22	31.64
20	W'LET	(53,65)	16	14	14	18.68
21	WDF5	(23,29)	16	12	12	9.80
22	WDF7	(31,37)	22	12	12	28.68
23	WDF9	(23,28)	26	9	9	19.62

Table 1: Example Applications

The ASPP synthesis techniques proposed in this paper were validated on the set of DSP, video, control and communication applications summarized in table 1. For each application, columns 3-6 show the number of nodes (N), the number of edges (E), the word length, the critical path, and the input time constraint, respectively. The last column is the area in mm^2 of a dedicated ASIC implementation.

Application bundling was invoked on this set of applications to minimize the manufacturing cost. Constraints were imposed on the number of chips on which these applications should be implemented, and the applications that cannot be implemented on the same IC. For example, incompatibility constraints were imposed on the nine applications (3, 5, 6, 7, 14, 15, 16, 17 and 19) implementing the discrete cosine transform to prevent them from being bundled together. Table 2 summarize the results with three different constraints on the number of ICs. Column 2 shows the applications in each bundle. Figures 6 (a,b,c) are the layouts of the PR2 and ADAPT ASICs and their ASPP implementation. The ASPP is only 10.5% larger than PR2 (the

larger of the ASICs).

# IC	Bundles	Area		
		ASIC	ASPP	Red'n
10	{10,14,20,22},{15}, {1,6},{4,8,21},{2,17}, {16},{7,23},{5}, {9,11,13,18,19},{3,12}	452.13	265.31	41.3%
11	{13,21,23},{9,15},{8}, {1,10,14},{4,17},{16}, {2,6},{5,11,18},{3}, {12,19,20,22},{7}	452.13	280.19	38.0%
12	{14,23},{11,12,18}, {1,6},{4,8,13,21},{7}, {2,9,20},{5,10,22},{15}, {16},{3},{19},{17}	452.13	298.61	34.0%

Table 2: Bundling to minimize manufacturing cost

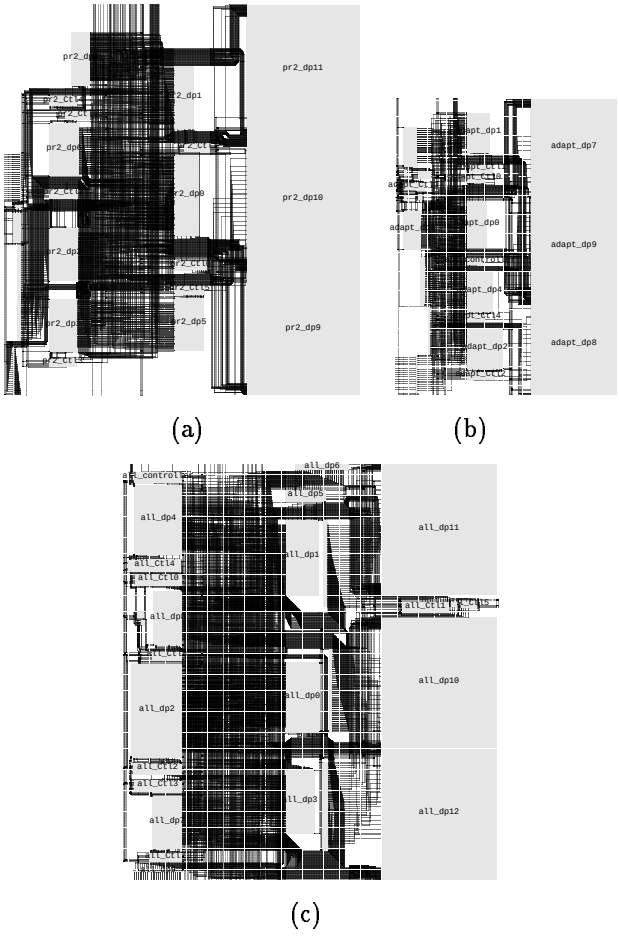

Figure 6: Layout of (a) PR2 ASIC, (b) ADAPT ASIC, and (c) {PR2,ADAPT} ASPP

The results targeting risk reduction are shown in table 3. The overhead is minimized by bundling applications whose hardware requirements are a subset of the hardware requirements of the largest design. In row 2 of table 3, four bundles have a single application. This is because the applications are incompatible with the remaining applications.

# IC	Bundles	ASPP Area	Total Overhead
10	{1,14},{6,23},{15,20,22},{7,9} {4,19},{2,10,17},{5,8,12}, {16},{3},{11,13,18,21}	280.25	10.05
11	{23},{15},{6},{1},{7,9,10,13} {4,14},{8,16},{2,17,21}, {5,11,18},{3,12,20},{19,22},	302.69	2.36
12	{23},{15,22},{8,14},{1}, {5},{4,19},{16},{2,6},{17}, {3,9,13},{11,18,21},{7,10,12,20},	309.85	0.00

Table 3: Bundling for risk reduction

7 Conclusions

We introduced a new synthesis approach to ASPPs. In addition to allocation and scheduling algorithms which resolve a number of generic and unique ASPP design optimization aspects, we addressed a new synthesis problem unique to ASPPs: application bundling.

References

[1] G. De Micheli, Synthesis and Optimization of Digital Circuits, Mc Graw Hill, 1994.

[2] R. Leupers, W. Schenk, P. Marwedel, "Retargetable Assembly Code Generation by Bootstrapping", Int'l Symposium on High-Level Synthesis, 1994.

[3] P.G. Paulin, C. Liem, T.C. May, S. Sutarwala, "CodeSyn: A Retargetable Code Synthesis System", Int'l Symposium on High-Level Synthesis, 1994.

[4] M. C. McFarland and A. C. Parker and R. Camposano, "The high-level synthesis of digital systems", Proc of IEEE, Vol. 78, No. 2, pp. 301-318, 1990.

[5] M. Gokhale, et al., "SPLASH: A Reconfigurable Linear Logic Array", ICPP, 1990.

[6] A. El Gamal, J. Rose, A. Sangiovanni-Vincentelli, "Synthesis Methods for Field Programmable Gate Arrays", Proc. of IEEE, pp. 1013-1029, 1993.

[7] G.D. Hillman: "DSP56200: An Algorithm-Specific Digital Signal Processor Peripheral", Proc. of IEEE, pp. 1185-1191, 1987.

[8] E. A. Lee and D. C. Messerschmitt, "Static Scheduling of Synchronous Data flow Programs for Digital Signal Processing", IEEE Trans. on Comp, pp. 24-36, 1987.

[9] R. Gupta, G. De Micheli, "Partitioning of Functional Models of Synchronous Digital Systems", ICCAD, pp. 216-219, 1990.

[10] E.D. Lagnese, D.E. Thomas, "Architectural Partitioning for System Level Design", DAC, pp. 62-67, 1989.

[11] M.C. McFarland, "Computer-Aided Partitioning of Behavioral Hardware Descriptions", DAC, pp. 472-480, 1983.

[12] F. Vahid, D.D. Gajski, "Specification Partitioning for System Design", DAC, pp. 219-224., 1992.

[13] J. Rabaey, et al., "Fast Prototyping of Data Path Intensive Architectures", IEEE Design & Test, Vol. 8, No. 1, pp. 40-51, 1991.

[14] A.K. Yeung, J.M. Rabaey, "A 2.4 GOPS data-driven reconfigurable multiprocessor IC for DSP", ISSCC, pp. 108-109, 1995.