A Design Flow for Partially Reconfigurable Heterogeneous Multi-Processor Platforms

Li Jiashu, Anup Das and Akash Kumar
Department of Electrical & Computer Engineering
National University of Singapore, Singapore
{lijiashu, akdas, akash}@nus.edu.sg

***Abstract*—Modern multiprocessor systems-on-chip (MPSoCs) are expected to handle multi-application usecases. As the number and complexity of these applications scale, resource allocation to meet the application throughput requirement is becoming quite a challenge. In this paper, a complete design flow is proposed for partially reconfigurable heterogeneous MPSoC platforms. The proposed flow determines the minimum resources required to map and guarantee the throughput of applications in all use-cases. Further, a suitable mapping for each application is chosen so that energy consumption is minimized. Experiments conducted with a set of synthetic benchmarks and real-life applications clearly demonstrate the advantage of our approach over homogeneous or fully reconfigurable designs. The proposed design flow achieves more than 50% energy savings when the number of configurations is not optimized. With configuration-optimization, our flow results in 75% reduction in the number of configurations with 5% reduction in energy.**

***Index Terms*—Partially reconfigurable systems; heterogeneous systems; design-flow; multiple use-cases.**

I. Introduction

Modern embedded systems are usually required to execute multiple applications simultaneously [17]. As the complexity of these applications scales, a single general propose processor (GPP) is no longer capable of supporting all tasks. To accommodate ever increasing demand of applications and for the ease of scalability, multiprocessor systems-on-chip (MPSoCs) are becoming the obvious design choice in current and future technologies [4]. To meet the strict timing requirement of multimedia applications, system designers are resorting to heterogeneous architectures [22] because of the area and power penalties associated with homogeneous cores and GPP [12].

As the complexity of MPSoC scales, there is an increase in the number of applications enabled simultaneously (use-case[1]). Strict area and power budget coupled with requirement of low delay of use-case switching are becoming impossible to handle using static architectures. Partial reconfiguration (PR) has been proposed in recent years to improve the system performance as well as the resource utilization [6]. PR can have two interpretations in the context of MPSoC: 1) it can refer to a platform consisting of FPGA and ASIC design, where the FPGA logic can be reconfigured at runtime, and 2) it can refer to the process of reconfiguring only a part of the FPGA logic while the other parts continue operation. In this paper, benefits of both levels of PR are exploited.

There are many research works targeting design and mapping flow for partially reconfigurable platforms. These mappings are either not scalable with the number of applications (e.g. [8][13]) or they do not consider multiple-application use-cases (e.g. [1][14]). There are some research works focusing on multiple use-cases mapping flow. Area and energy are not considered in these works [22].

[1]Formally, an use-case is defined as a collection of multiple applications that are active simultaneously on an MPSoC

Contribution: In this paper a complete design flow for partially reconfigurable heterogeneous MPSoC platforms is proposed. The key contributions are the following.

- *Minimum Resource:* A design flow to determine the minimum resource requirement, which can support and guarantee throughput of all applications in all use-cases.
- *Combining Use-cases:* An algorithm to minimize reconfiguration delay by combining multiple use-cases into one configuration.
- *Energy Consumption Reduction:* A methodology to reduce the energy consumption by utilizing spare resources for each configuration.
- *Pareto Optimization:* Pareto optimizations to reduce data size and the time required for analysis.
- *Use-case Switching:* A method to speed up the reconfiguration process by utilizing the features of partial reconfiguration of modern FPGA devices.

The rest of the paper is organized as follows. Section II provides an overview of prior related works in this area. Section III introduces the application and hardware models used in our flow. Section IV describes the detailed mapping flow. Experimental results are provided in Section V. Section VI concludes the paper with key future directions.

II. Related Work

Shrinking time-to-market deadlines and strict area and power budget are contributing to the growing popularity of the hardware-software co-synthesis design space exploration methodology in modern MPSoCs. To better utilize the designed MPSoC platform and to reduce energy consumption, system designers need to explore different mapping alternatives of an application on the platform.

Most of the research works on application task mapping on MPSoC have focused on static techniques. These techniques determine the best placement of tasks on the cores at design-time. There are different mapping objectives studied in literature. An hardware-software approach with heterogeneous scheduling is proposed in [8]. Mappings are determined at design-time to minimize the dynamic power consumption. A mapping technique is proposed in [16] to reduce the task communication delay. There are also some studies to maximize the application throughput using satisfiability (SAT) techniques

[13]. A limitation of these static application mapping strategies is that only fixed architecture is supported and therefore a limited set of applications can be mapped on the platform.

Dynamic mapping approaches consider application dynamism in resource allocation. There are some studies to perform resource-execution time trade-off analysis to determine mapping of applications (modeled using synchronous dataflow graphs) on MPSoC [22]. An energy-aware mapping heuristic is proposed in [14] that does not resort to resource reservation. Most of these dynamic techniques assume homogeneity of cores and can lead to large area and power overhead when they are applied to heterogeneous architectures.

There are a few mapping techniques for partially reconfigurable platforms. A mapping algorithm is proposed for NoC-based Heterogeneous MPSoC where tasks of an application are mapped into one reconfigurable fabric to reduce the communication delays [19]. However, throughput is not guaranteed in this technique. A design flow to exploit similarities in applications is proposed in [1]. Although the reconfiguration delay is reduced significantly, core selection is not performed according to the hardware platform and resource availability.

III. Application and Architecture Models

This section provides a brief overview of the application and the architecture model used in the proposed design flow.

A. Application Graph Model

Synchronous Data Flow Graphs (SDFGs, see [11]) are often used for modeling modern DSP applications and for designing concurrent multimedia applications implemented on a multiprocessor system-on-chip. The nodes of an SDFG are called *actors*; they represent functions that are computed by reading *tokens* (data items) from their input ports and writing the results of the computation as tokens on the output ports. The number of tokens produced or consumed in one execution of actor is called port *rate*, and remains constant. The rates are visualized as port annotations. Actor execution is also called *firing*, and requires a fixed amount of time, denoted with a number in the actors. The edges in the graph, called *channels*, represent data that is communicated from one actor to another.

Mathematically, an SDF graph is represented as a directed graph $G = (V, E)$, where $V = \{v_i | 1 \leq i \leq n\}$ represent the *actors* in G and the directed edges $E = \{e_{ij} | i, j \in [1, ..., n]\}$ represent the *channels*. Each communication channel e_{ij} links from source actor v_i to destination actor v_j. When an actor v_i starts its firing, it removes $Rate(ji)$ tokens from all $e_{ji} \in InC(v_i)$. The execution continues for $\tau(v_i)$ time units and when it ends, it produces $Rate(ik)$ tokens on every $e_{ik} \in OutC(v_i)$[2]. The repetation vector of an actor v_i is the number of times the actor is fired in one iteration of the SDFG. Each communication channel e_{ij} is also specified with a capacity, which is also known as the *buffer size*. It represents the maximum number of tokens that can be stored in the channel. Each channel e_{ij} may also contain a number of *initial tokens*.

Throughput is an important property of multimedia applications; it describes how fast those applications are able to run, and it is defined as the inverse of the average iteration time of an application. The technique of analyzing throughput of the SDFGs is described in [3].

[2] $InC(a)$ and $OutC(a)$ are respectively the incoming and outgoing edges of actor a.

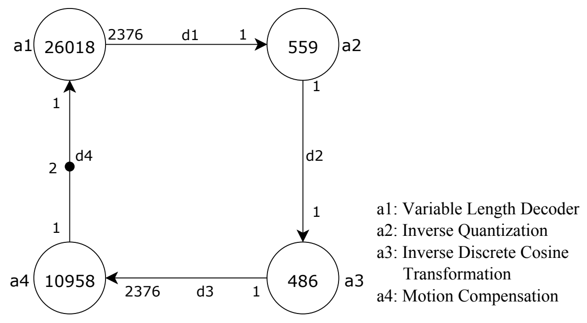

Fig. 1. SDF Graph of an H.263 Decoder

Figure 1 shows the SDFG of H.263 decoder [20] which is modeled using four actors *a1, a2, a3, and a4* and four communication channels *d1, d2, d3, and d4*. The rate of an actor is specified on its channel while its execution time[3] is indicated in the circle. An edge may also contain initial tokens which are indicated by bullets. Throughput constraint for an application is specified in the corresponding application model.

B. Hardware Platform Model

Partially reconfigurable heterogeneous MPSoC are used as the hardware platform for our flow. The advantages of partially reconfigurable platform over ASIC and fully reconfigurable FPGA design are the following:

- flexible as compared to ASIC
- faster than fully reconfigurable FPGA
- consumes less energy than fully reconfigurable FPGA

Partially reconfigurable architecture offers flexibility yet affordable solution for high-performance embedded systems.

Figure 2 shows the hardware model used in this work consisting of ASIC cores and reconfigurable area (RA). All ASIC cores are hard logic fixed during design process. These can be general purpose processors, application-domain specific processors (ADSPs) or any other ASIC designed cores. The RA consists of logic which can be re-programmed at run-time (eg. FPGAs). These are task-specific and usually bind to specific task in the application set. The allocation of actors to ASIC and RA is based on actor timing properties as well as the resource availability. By proper actor allocation, our flow minimizes the resource requirement.

IV. Design Flow

Figure 3 shows the two-phase design flow proposed in this work. The first phase is the resource optimization phase where the target MPSoC platform is determined to minimize the resource usage. The flow then advances to the energy-delay optimization phase where techniques are developed to minimize energy and reconfiguration delay of the designed MPSoC. The flow proposed here is motivated by the strict area budget often imposed in system specification; energy and reconfiguration delay are set at a priority lower than the area.

[3] The execution time of the actors is measured on ARM Cortex A9 procesor

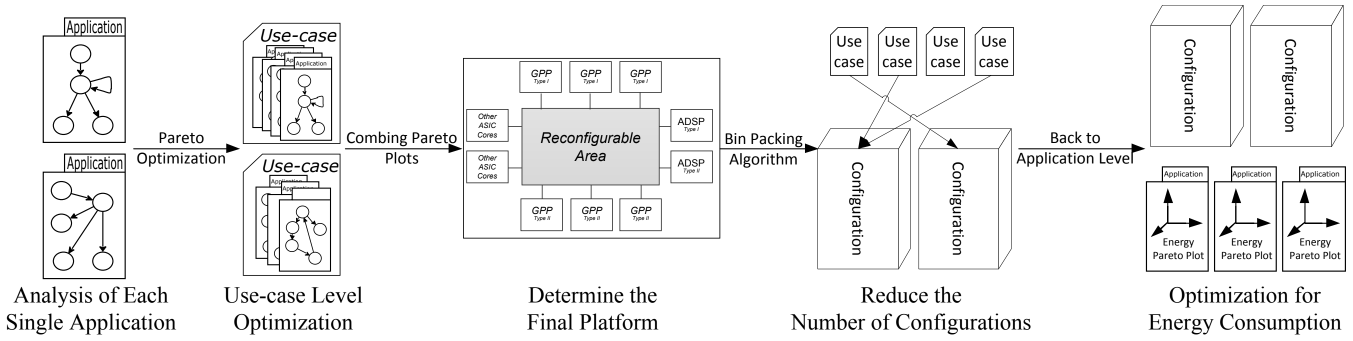

Fig. 3. Proposed Mapping Flow

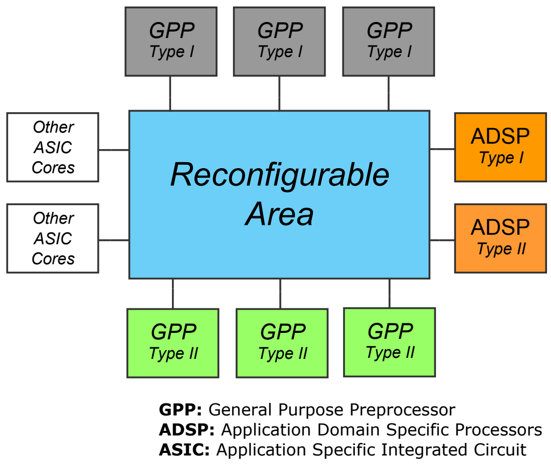
GPP Type I
Other ASIC Cores
Reconfigurable Area
ADSP Type I
ADSP Type II
GPP Type II
GPP: General Purpose Preprocessor
ADSP: Application Domain Specific Processors
ASIC: Application Specific Integrated Circuit

Fig. 2. Conceptual Hardware Model (MPSoC)

The first step in the flow is the analysis for every application to determine resource usage and energy consumption for every possible mapping on the given architecture. The set of optimal points are retained. The flow then advances to the use-case level; the Pareto plots of different applications within a single use-case are added together, and the resource usage for each use-case is calculated. The maximum resource requirement among all use-cases is selected as the architecture resource requirement. Once the target architecture is finalized, multiple use-cases are packed into one configuration to reduce the reconfiguration delay. Finally, spare resources for each configuration (if any) are utilized to minimize energy consumption.

A. Pre-processing

Applications to be mapped on our platform can have varying set of requirements. Some use-cases can be trivial or unrealistic. To tackle these scenarios, a pre-processing step is needed before the actual design flow.

1) Identify Unrealistic Use-cases: Unrealistic use-cases refer to those which are unlikely to occur in the real life such as playing mp3 while talking on a mobile phone. In the pre-processing step, unrealistic use-cases are identified early in the design cycle to prevent unrealistic outputs.

2) Identify Trivial Use-cases: A use-case is defined as trivial if it is a proper subset of some other use-case [9]. Formally, for any use-case U_i, if U_i is trivial, then there exists another use-case U_j, so that all applications which are included in U_i are supported by U_j as well. Discarding trivial use-cases minimizes the reconfiguration delay of switching from a trivial use-case to its super-level use-case and vice-versa.

3) Same Application with Different Throughput Constraints: In embedded MPSoCs with multiple applications it is possible to have one application with different throughput requirements within a use-case. An example of such use-case is picture-in-picture (PiP). Here, the video decoding for two different programs (channels) have different throughput requirements due to different resolution. In our flow, they are treated as two different applications because of the different performance and potentially different hardware requirements.

B. Analysis of Applications

The second stage of our design flow enumerates every mapping of an application to determine if the throughput constraint is met. All mappings satisfying the throughput requirement are stored along with their resource usage and energy consumption. If no such mapping is found, the corresponding application is discarded.

1) Time Complexity Analysis: The time complexity of the analysis stage is proportional to the number of mappings evaluated. This is computed based on the fact that actors of an application can be mapped on a dedicated core or on a GPP. Defining $P(x)$ as the number of possibilities for mapping x actors on x GPPs; the total number of mappings M_n for an application with n actors is given by

$$M_n = \sum_{k=0}^{n} \binom{n}{k} P(k) \tag{1}$$

According to [7], $P(x) = B_x$, the bell number, which is given by

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k \tag{2}$$

and

$$P(x) = B_n \sim \frac{1}{\sqrt{n}} \left[\lambda\left(n\right)\right]^{n+\frac{1}{2}} e^{\lambda(n)-n-1} \tag{3}$$

Thus,

$$M_n = B_{n+1} \sim \frac{1}{\sqrt{n+1}} \left[\lambda\left(n+1\right)\right]^{n+\frac{3}{2}} e^{\lambda(n+1)-n-2} \tag{4}$$

Clearly, M_n grows exponentially with the number of actors n for a specific application. The number of mappings for applications with $5, 10, 15$ actors are $203, 678570, 10480142147$ respectively.

2) Speed Up Possibilities: To identify speed-up possibilities, the following definitions are introduced.

Definition 1: (ACTOR LOAD) *Load of an actor is defined as the amount of computation required for that actor in a period and is given by the following formula.*

$$ActorLoad(i) = \frac{ExecTime(i) \times RepVec(i)}{Period}$$

Definition 2: (PROCESSOR LOAD) *Load of a processor is defined as the sum of actor loads for all the actors mapped onto the processor.*

Two methods are proposed to speed-up the analysis stage.

a) Processor Load Based Pruning: Processor load defines the utilization of a processor necessary to satisfy the throughput requirement of the actors mapped to the processor. If a processor load is more than one, the actors mapped to the processor will never be able to meet their throughput requirements and therefore, these mappings need to be discarded. The first speed-up process involves pruning of such mappings to retain those for which the processor load is less than one.

b) Load Balancing: As shown in Equation 4, the number of possible mappings grows exponentially with the number of actors in the application. A load balancing strategy is therefore proposed to distribute actors evenly on GPPs. Thus a single mapping configuration is analyzed for a specific number of GPPs rather than analyzing all the possibilities. With this, Equation 4 can be re-written with $P(x)$ replaced by x when load balancing algorithm is applied.

$$M_n = \sum_{k=0}^{n} \binom{n}{k} \times x = 2^{n-1} \times n \tag{5}$$

Although, the number of mappings still grows exponentially with the number of actors, the growth is slower than Eqn. 4.

3) Pareto Optimization: Pareto optimization is performed on the mappings obtained after initial screening. Optimal mappings in terms of throughput, resource usage and energy consumption are recorded which reduces the data size and the analysis time. According to [15], the Pareto dominance and Pareto optimal can be defined as follows:

Definition 3: (PARETO DOMINATE) *Given vectors* $u = \{u_1, u_2, u_3, ..., u_n\}$ *and* $v = \{v_1, v_2, v_3, ..., v_n\}$, *$u$ is said to dominate v if and only if* $\forall i \in \{1, ..., n\}, u_i >= v_i$ *and* $\exists i \in \{1, ..., n\}, u_i > v_i$.

Definition 4: (PARETO OPTIMAL) *A solution $x_u \in U$ is said to be Pareto optimal if and only if there is no solution $x_v \in U$, where x_v dominates x_u.*

Definition 5: (PARETO SUBOPTIMAL) *A solution $x_u \in U$ is said to be Pareto suboptimal if and only if $\exists x_v \in U$, where x_v dominates x_u.*

Since Pareto suboptimal points are equal or inferior to their respective dominant points in all aspects, a solution containing those suboptimal is definitely inferior to another solution containing its dominant points instead. Thus, Pareto suboptimal points are removed at every stage to reduce the data size and hence shorten the time required for further processing.

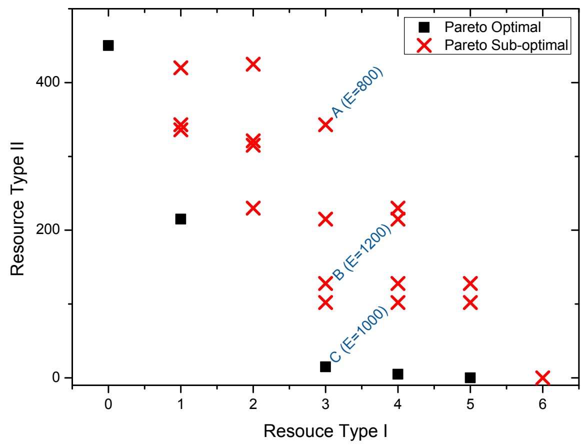

Fig. 4. Pareto Optimization

Figure 4 plots the two dimensional Pareto space considering two different resource types[4]. The bold block points in the figure represent the area optimal (Pareto optimal) designs considering the two types of resources. The points marked with cross are sub-optimal points in terms of design area. However, some of these sub-optimal points (point A and B in the figure) can be Pareto optimal when some other metric (energy for example) is considered. The Pareto optimization phase is re-visited after the MPSoC architecture is determined to explore some of these design points (both optimal and sub-optimal) for potential energy optimization.

C. Use-case Level Optimization

Once the application-level Pareto optimization phase completes, use-case level optimization is performed. Applications with shared resources are hard to analyze. Authors in [10] present a method to predict the performance when multiple applications are mapped onto shared resource, but provides no hard timing guarantee. In our flow, it is assumed that the applications within a use-case will not share the same processor. Thus, the resource usage for every single application in a particular use-case is added up to form the resource requirement for that particular use-case. This is equivalent to adding the Pareto points. All possible combinations from the Pareto plots need to be considered in this process. Pareto optimization is again performed on the set of points thus obtained.

D. Determine the Final Platform

Resource requirement of the final platform is selected based on the minimum hardware set required to support a use-case. As no two use-cases can run concurrently, the minimum set of hardware supporting all use-cases is to be selected.

A cost function C is proposed in our flow to solve this problem. If $C_n(x)$ is defined as the cost to have x unit

[4]The number of dimensions of the Pareto space is equal to the number of resource types considered in the MPSoC platform.

of Resource Type n, the total cost of the solution can be expressed as

$$C = \sum_{k=1}^{n} C_k(x_k) \tag{6}$$

where $\{x_1, x_2, ..., x_n\}$ indicate the usage for resource type $1, 2, ...n$ for a particular solution. The cost function is user defined and is used to determine the final platform.

E. Reduce the Number of Configurations

A methodology is proposed to divide the use-cases into different sets, so that each set can be packed into one configuration and its resource usage does not exceed the size determined in the previous stage. This reduces use-case switching time[5].

The problem of combining use-cases into one configuration is modeled as a bin-packing problem, where every bin is a configuration and objects represent the use-cases [5]. Bin packing problem being NP-hard, three approximation algorithms are proposed to solve the problem. The performance of these algorithms are evaluated in Section V.

a) Greedy Algorithm: The greedy approach selects the largest set of the use-cases which can be fitted into the device at every stage. The algorithm finishes when all the use-cases are assigned with a configuration.

b) First Fit: The First Fit (FF) algorithm labels the configurations as $c_1, c_2, ..., c_n$, and the use-cases as $u_1, u_2, ..., u_n$. For every use-case u_x, the algorithm packs it into configurations by the order of 1 to n. u_x will be packed into the least index i such that c_i can contain u_x.

c) First Fit Decreasing: The First Fit Decreasing (FFD) algorithm is a variant of First Fit. It reorders all the use-cases according to their resource usage. Let r_x be the resource usage for use-case u_x. The sorted use-cases will satisfy $r_i \leq r_{i+1}$, for $1 \leq i < n$. After sorting, the First Fit algorithm is applied.

F. Optimization for Energy Consumption

As established in Section IV-B, the first phase of the design flow involves selection of mappings to optimize the resource usage. Energy is not taken into consideration in the Pareto optimization step. However, once the final platform is determined, it is necessary to explore possibilities of energy minimization by re-looking at the mappings with energy as another metric in conjunction with resource usage. This motivates re-visiting the Pareto optimization step of the design flow (see Figure 4). All points in the Pareto space are marked with the corresponding energy numbers (shown in the figure in parenthesis against some of the points – *A*, *B* and *C* for example). As can be seen from the figure, point *A* is Pareto optimal in terms of energy and suboptimal in terms of resource usage, point *C* is Pareto optimal in terms of both energy and resource usage while point *B* is suboptimal considering both the metrics.

Energy optimization for each configuration is performed by selecting one mapping for each application within the configuration such that the total energy of the configuration is reduced, honoring the resource usage pre-determined in the Pareto-optimization step of Section IV-B.

[5]If two use-cases are in the same configuration, the time required to switch between them is much reduced since only the communication layer needs to be re-programmed.

G. Switching Among the Configurations

To switch between configurations during runtime, reconfiguration bitstreams are required to re-program the reconfigurable devices. As generating bitstreams at runtime can be very costly, our flow generates all reconfiguration bitstreams at design-time and stores them on the device.

Modern FPGA devices support reconfiguring only a part of the FPGA keeping other logic unchanged. Partial reconfiguration is preferred over full reconfiguration when there are only few changes in the logic, as partial bitstreams are smaller and take less time to reconfigure[6]. In this paper, three methods are proposed to switch among configurations:

a) Complete Reconfiguration: In this technique, the reconfigurable area is completely re-programmed every time user switches configuration. There are n bitstreams required for n different configurations supported on the target MPSoC. However, one drawback of this approach is the long reconfiguration delay causing user to wait longer before a system can be used again.

b) Customized Transition: The idea of customized transition is to provide customized partial bitstreams for each possible transition among the use-cases. This method achieves the shortest reconfigurable delays as the bitstream will only reconfigure the differences between the configurations. However, considering the possibility of switching between two arbitrary configurations, $2 \times \binom{n}{2}$ bitstreams are needed to be stored in the system. This impacts storage requirements.

c) Base Configuration: In this technique, a *base* configuration is created and every other configuration is considered as a variant from this configuration. Figure 5 shows the concept of this method. The *base* configuration is a collection of the frequently used cores and aims to have minimum differences from other configurations. Thus for each configuration, only two bitstreams are required – one to reconfigure from the *base* configuration to the current configuration and the other to revert back to the *base* configuration. By using this method, the total number of bitstreams required is reduced to $2 \times n$.

One of the three techniques is selected based on the application scenario. Configurations requiring frequent switching prefer customized transition bitstreams; complete reconfiguration is used for configurations with lots of uncommon cores; the *base* configuration method provides a compromise between the other two methods and can be applied for other configurations.

V. Experimental Results

Experimental results are obtained by applying the proposed design flow on a set of synthetic benchmarks and a multimedia application case-study. The improvements on the resource usage, as well as the effectiveness of techniques used in our flow are described in this section.

The set of synthetic benchmarks consists of 10 applications randomly generated by the SDF^3 tool [21]. The actor count of each application ranges from 5 to 7. 80 use-cases are generated for the benchmark, and 3 to 5 applications are randomly placed into each of the use-cases. For real application case study, a multimedia application set is formed with 6 applications[7] –

[6]Configuration time is directly proportional to the size of the bitstream [6]

[7]Benchmarks from [21][2]

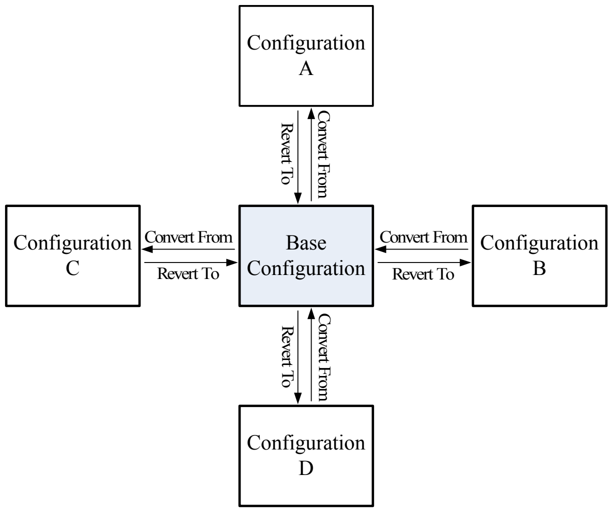

Fig. 5. The Base Configuration Scheme

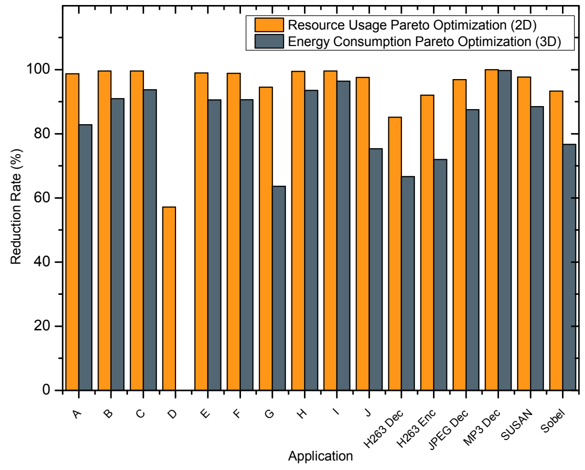

Fig. 6. Effectiveness of Pareto Optimization

TABLE I
Comparison of Resource Requirement of the Resulting Device

	Our Design Flow		**Ref. Flow 1** [18]	**Ref. Flow 2** [1]
Resources	**GPPs**	**RA Size**	**GPPs**	**RA Size**
Synthetic	6	1125	FAIL	2319
Multimedia	3	400	FAIL	1000
Synthetic Relaxed Constraints	8	289	11	1889

two video CODECs (H.263 encoder, H.263 decoder), three image processing units (JPEG decoder, SUSAN, and Sobel) and one audio decoder (MP3 decoder). The application set includes a total of 10 use-cases with applications containing 4 to 14 actors.

The hardware model consists of one type of GPP and reconfigurable area. Each actor in the application set can be mapped into two types of cores: GPP or a dedicated core in the reconfigurable area. Most of the execution time, area and power values are from the SDF^3 online models and are normalized with respect to GPP.

Experiments are conducted on a 3.1GHz Intel workstation running Linux. SDF^3 tool is used in this experiment to analyze the throughput of the applications.

A. Execution Time

The execution time of the proposed design flow are *6 minutes* and *82 minutes* for synthetic and multimedia application sets respectively. The application analysis stage is the most time-consuming part of the design flow and consumes $> 99.9\%$ of the total time. This is due to the analysis of large number of mapping possibilities. Execution time of the remaining stages in the flow is only *160 ms* and *535 ms* for synthetic and multimedia sets respectively.

B. Resource Requirement of the Resulting Device

Table I compares the resource utilization of our flow with other state-of-the-art design flows [18] and [1]. The flow proposed in [18] is targeted at minimizing the resource requirements for homogeneous platforms. Since all the tiles are identical, GPP is used as the processing element. However, this design flow fails to complete for both sets of applications considered due to the tight timing constraints of some applications. The flow is therefore compared with ours using a set of applications with relaxed timing constraints. Result shows that our design flow achieves 27% reduction in the number of GPPs with an extra reconfigurable area overhead of 289 units in comparison with the flow in [18].

The flow proposed in [1] maps multiple applications into heterogeneous MPSoCs on fully reconfigurable hardware architectures. The difference of this approach with ours is that in [1] resource requirement is determined for every application prior to the flow thereby having minimal resource for every application. Experiments with real and synthetic applications result in reduction of the size of reconfigurable area by 51% to 85% as compared to [1]. These savings translate to the reduction in the number of GPPs.

C. Pareto Optimization

Figure 6 shows the effectiveness of Pareto Optimization. Pareto Optimization for resource usage is performed in 2 dimensions (2D). Only the number of GPPs and the RA size are considered in the optimization process. For energy consumption, the Pareto Optimization is performed in 3 dimensions (3D) taking GPP, RA size and energy consumption into consideration.

D. Use-case Reduction and Combination

Figure 7 shows the effectiveness of use-case combination technique proposed in our flow for a synthetic set of applications. As can be seen from the figure, our technique achieves a reduction of more than 60% in the number of configurations. This greatly reduces the overall time required to switch between use-cases.

Except for greedy algorithm, all other algorithms finish execution within *10 ms*. The greedy algorithm takes *38 seconds* due to the increase in the number of possible sets. However, when use-case reduction is applied, the execution

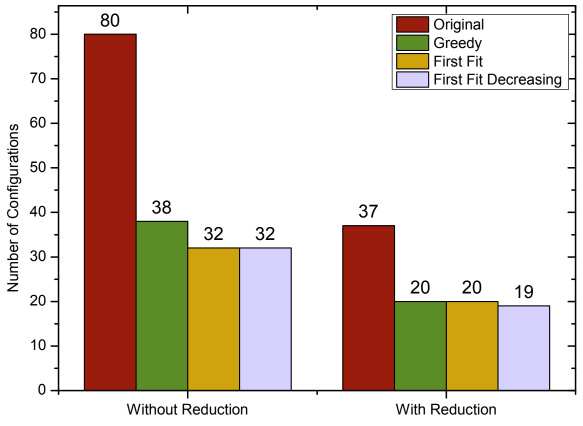

Fig. 7. Use-case Reduction and Combination for Synthetic Benchmarks

TABLE II
NUMBER OF CONFIGURATIONS / ENERGY CONSUMPTION TRADE-OFF

		Algorithm	No. of Config.	Energy Reduction Rate	
				Maximum	Average
Synthetic Benchmarks					
Without	**Reduction**	**Original**	80	94.35%	51.49%
		Greedy	38	79.89%	20.45%
		FF	32	10.33%	2.94%
		FFD	32	22.25%	1.97%
With	**Reduction**	**Original**	37	72.97%	41.17%
		Greedy	20	50.01%	11.41%
		FF	20	46.46%	10.95%
		FFD	19	68.88%	5.53%
Multimedia Case Study					
Without	**Reduction**	**Original**	10	72.97%	41.17%
		Greedy	5	18.09%	7.01%
		FF	5	18.09%	7.01%
		FFD	5	25.65%	8.52%
With	**Reduction**	**Original**	5	69.94%	30.10%
		Greedy	4	18.09%	8.76%
		FF	4	18.09%	8.76%
		FFD	4	18.09%	8.76%

time of the greedy algorithm reduces to *20 ms*. Thus, use-case combination technique proposed in this paper is effective in reducing the number of configurations and hence the overall performance of our algorithm.

E. Energy Consumption Optimization

Table II shows the number of configurations and energy savings obtained using our technique. As can be seen from the table, a trade-off can be observed. With 76.25% reduction of configurations, our flow achieves an average energy reduction of 5.53%. However, with no optimization of the configuration, 51.49% energy savings is obtained. The choice is left to system designer to select the proper number of configurations meeting the energy budget of the system.

VI. CONCLUSION AND FUTURE WORK

This paper presents a complete design flow to map multiple applications and use-cases onto a partially reconfigurable heterogeneous MPSoC Platform. The hardware platform proposed in this flow has a mixture of ASIC cores and Reconfigurable Areas, and it provides a flexible but yet affordable solution for high performance embedded systems.

The primary aim of our proposed flow is to determine the minimum resource requirement for the resulting device, which can support and guarantee throughput for applications in all use-cases. This is achieved by analyzing all mapping possibilities of each application. The flow also explores the possibilities of minimizing the use-case switching time by packing more use-cases into one configuration. Energy consumptions are then optimized using spare resources in each configuration. The technique of Pareto optimization is widely employed in this flow to reduce the data size and the time required for further analysis. Some heuristic algorithms for use-case combining are proposed, implemented and discussed in this paper. Results show the effectiveness of those algorithms.

In future, we plan to develop more heuristic algorithms and reduction methods to speed up the analyzing process of the flow, so that it is able to handle more complex application models and use-cases. Floor planning aspect for the reconfigurable area can also be explored in future.

REFERENCES

[1] I. Beretta *et al.* A Mapping Flow for Dynamically Reconfigurable Multi-Core System-on-Chip Design. *TCAD*, 2011.
[2] Electronic Systems Group at TU/e. MAMPS Website. http://www.es.ele.tue.nl/mamps/, 2010. [Online; Accessed 18-March-2012].
[3] A. Ghamarian *et al.* Throughput analysis of synchronous data flow graphs. In *ACSD*, 2006.
[4] M. Hubner. *Multiprocessor System-on-Chip: Hardware Design and Tool Integration*. Springer Verlag, 2010.
[5] D. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 1974.
[6] C. Kao. Benefits of partial reconfiguration. *Xcell journal*, 2005.
[7] J. Katriel. On a Generalized Recurrence for Bell Numbers. *Journal of Integer Sequences*, 2008.
[8] M. Kim *et al.* Energy-aware cosynthesis of real-time multimedia applications on MPSoCs using heterogeneous scheduling policies. *TECS*, 2008.
[9] A. Kumar *et al.* Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA. *TODAES*, 2008.
[10] A. Kumar *et al.* Iterative Probabilistic Performance Prediction for Multi-Application Multiprocessor Systems. *TCAD*, 2010.
[11] E. Lee *et al.* Synchronous data flow. *Proceedings of the IEEE*, 1987.
[12] J. Leijten *et al.* Prophid: a heterogeneous multi-processor architecture for multimedia. In *ICCD*, 1997.
[13] W. Liu *et al.* Efficient SAT-Based Mapping and Scheduling of Homogeneous Synchronous Dataflow Graphs for Throughput Optimization. In *RTS*, 2008.
[14] M. Mandelli *et al.* Multi-task dynamic mapping onto NoC-based MPSoCs. In *SBCCI*, 2011.
[15] R. Mouhoub *et al.* Multiprocessor on chip: beating the simulation wall through multiobjective design space exploration with direct execution. In *IPDPS*, 2006.
[16] S. Murali *et al.* Bandwidth-constrained mapping of cores onto NoC architectures. In *DATE*, 2004.
[17] Semiconductor Industry Association. International Technology Roadmap for Semiconductors (ITRS), 2007 edition, 2007.
[18] A. Shabbir *et al.* An MPSoC design approach for multiple use-cases of throughput constrainted applications. In *CF*, 2011.
[19] A. Singh *et al.* Mapping Algorithms for NoC-Based Heterogeneous MPSoC Platforms. In *DSD*, 2009.
[20] A. Singh *et al.* A hybrid strategy for mapping multiple throughput-constrained applications on MPSoCs. In *CASES*, 2011.
[21] S. Stuijk *et al.* SDF3: SDF for free. In *ACSD*, 2006.
[22] S. Stuijk *et al.* A Predictable Multiprocessor Design Flow for Streaming Applications with Dynamic Behaviour. In *DSD*, 2010.