

# A Modeling and Exploration Framework for Mapping of Linear Array of Tasks onto Adaptive Computing Systems<sup>1</sup>

Egor Andreev<sup>\*</sup>, Sumit Mohanty<sup>#</sup>, Viktor K. Prasanna<sup>#</sup>

<sup>\*</sup>Dept. of Computer Science

<sup>#</sup>Dept. of Electrical Engineering

University of Southern California

Los Angeles, CA 90089

{andreev, smohanty, prasanna}@usc.edu

## Abstract

*We discuss a modeling and design space exploration framework suitable for energy and latency efficient mapping of a class of applications onto adaptive computing systems. Reconfigurable devices, processors supporting dynamic voltage and frequency scaling, and memories supporting multiple operating states are some of the examples of adaptive computing systems (ACS). Such systems are ideal for low-power and high-performance implementation of embedded applications. We focus on the mapping of a class of signal processing applications, which can be represented as a linear array of tasks. Our framework provides support for modeling of such applications and the target ACS, specification of the mapping and design constraints, and selection of energy and latency efficient mappings through design space exploration. The framework is implemented using MILAN, an embedded system design environment. Exploiting the MILAN capabilities, we provide user friendly graphical interface for modeling, constraint specification, and design space exploration. Specifically, our framework uses DESERT, a constraint satisfaction tool, and HiPerE, a rapid performance estimation tool, to perform design space exploration. We illustrate the capability of the modeling and exploration framework through the design of an energy efficient beamformer application.*

## 1. Introduction

The ability to dynamically customize the architecture or the operating mode of an adaptive computing system to match the computation, data flow, and quality of service (QoS) requirements of an application has demonstrated significant performance benefits such as lower latency and energy dissipation compared to non-adaptive systems. Reconfigurable devices such as FPGAs, processors supporting dynamic voltage and frequency scaling, and memories with multiple operating states and banks that can be individually turned on/off are some of the examples of adaptive computing systems (ACS) [16][17]. An ACS can be a single device as described above or a combination of two or more devices (e.g. FPGA and a processor). An ACS allows the designer to change the operating state of the system to suit the target task. For example, one can reconfigure an FPGA or modify the operating voltage or frequency of a processor to suit the performance requirements of the mapped application task.

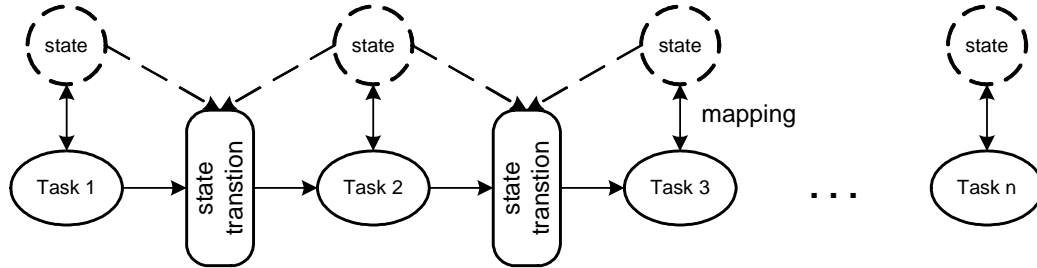
Synchronous data flow (SDF) graph is a well-known application model suitable for a large class of signal and image processing applications [4]. A simplified version of SDF is a linear data flow which models an application as an ordered set of tasks where each task can have at most one input and one output. Due to a simple and regular structure, linear data flow is well suited for formal algorithmic analysis and optimization [2][12]. Several applications of interest to military and general consumers such as automatic target recognition, automated object tracking, MPEG decoder/encoder, software defined radio, etc. can be modeled as a linear data flow graph [2][12][15]. Such applications, when implemented on adaptive computing systems (ACS), pose several design challenges including optimization of a single performance metric such as latency or energy or optimization of one metric while meeting a pre-specified requirement for another metric.

In this paper, we consider the mapping of a linear array of tasks onto an ACS. An ACS is associated with several operating states. A mapping in our case refers to the identification of a set of operating states such that each task is associated with an operating state (Figure 1). Hence, the ACS may need to modify its operating state between the executions of two consecutive tasks. Each operating state is associated with certain amount of latency and energy cost for each task that can be executed in the state. State transition cost includes latency and energy dissipation. Such a model poses several design challenges such as optimization of a single performance metric (e.g. latency or energy) and optimization of one metric while meeting a pre-

---

<sup>1</sup> This work is part of the MILAN project, supported by the DARPA Power Aware Computing and Communication (PAC/C) Program under contract F33615-C-00-1633 monitored by Wright Patterson Air Force Base.

specified requirement of another metric. While many solutions exist for single metric optimizations using dynamic programming [3], optimizations that involve two metrics are shown to be NP-hard [10]. In addition, the design space increases exponentially as the number of tasks or operating states increases [9]. Therefore, it may not be practical to traverse the complete design space to identify the design that meets the performance requirements.



**Figure 1: Mapping of a linear array of tasks and state transitions**

For design space exploration, we follow a hybrid approach that uses DESERT, a constraint satisfaction tool, and HiPerE, a rapid performance estimation tool, to perform design space exploration. Design Space Exploration Tool (DESERT), developed at the Vanderbilt University, uses symbolic method based on Ordered Binary Decision Diagrams (OBDDs) for constraint satisfaction [11]. High-level Performance Estimator, HiPerE, is a rapid performance estimation tool developed at University of Southern California to enable rapid design space exploration [10]. HiPerE is used to evaluate the designs identified by DESERT.

We discuss the design and implementation of a framework that provides capabilities to model a linear data flow application and the target ACS, populate the model, and enable semi-automatic identification of the optimal design and its performance while meeting the requirements specified by the designer. The above framework is a part of a larger design environment, MILAN, which addresses various other issues related to complete system design. MILAN is a **Model based Integrated simuLatioN** framework for embedded system design and optimization through integration of various simulators into a unified environment [1][8]. The extension of MILAN, discussed in this paper, enables specification of an adaptive computing system and target application and performs design space exploration to identify the mapping based on pre-specified performance requirements.

The paper is organized as follows. Section 2 discusses some related works. Section 3 discusses the MILAN design framework. Section 4 discusses our approach for mapping a linear array of tasks onto ACS. Section 5 describes the design flow using MILAN. An example to demonstrate our design flow is discussed in Section 6 and we conclude in Section 7.

## 2. Related Work

There are a number of design tools for application design using adaptive computing systems. The Xilinx System Generator for DSP and Mentor Graphics FPGA Advantage provide support for design, simulation, and implementation of FPGA based systems [7][16]. However, while using these tools, the designer starts with a single conceptual design and explores various implementation choices for the initial design. Analytical modeling and optimization are not supported by any of these tools. Designers typically rely on pen-and-paper based analysis for modeling an optimization. There is also a lack of a general purpose tools for ACS that can handle different kind of operating states and state transition costs while mapping applications onto ACS.

In addition to the tools discussed above, many techniques for formal modeling and optimization of a single metric (latency or energy) while mapping applications onto reconfigurable computing systems have been proposed [2][3][12]. While these techniques are faster compared with our DESERT and HiPerE based approach, they are not suitable for multi-metric optimization. Similarly there have been many research works addressing optimal scheduling for processors supporting dynamic voltage scheduling (DVS) [6][14]. These solutions provide an analytical solution to identify energy optimal scheduling by exploiting DVS. In contrast, we model DVS as a generic property of ACS and provide a design framework to model the application and devices and explore the design space.

In this paper, we discuss a general purpose framework suitable for optimization of latency and energy (single metric as well as both the metrics) while mapping a linear array of tasks onto an ACS and the enhancements to MILAN that provide a semi-automated design environment to perform such modeling and exploration.

### 3. MILAN: A Model based Integrated Simulation Framework

MILAN is a model based integrated simulation environment for embedded system design and optimization through the integration of various simulators and tools into a unified environment [8]. Using the MILAN environment, the designer formally models the target application, underlying hardware, and constraints (latency, throughput, energy dissipation, etc.) through a graphical interface provided by MILAN. The models are stored in a model database. The model information is translated through model interpreters into suitable input formats required by the integrated simulators. MILAN adopts Model Integrated Computing (MIC) as the core design technology [1]. The Generic Modeling Environment (GME) is a configurable graphical tool suite supporting MIC [5]. GME allows the designer to create domain-specific models. A metamodel (modeling paradigm) is a formal description of model construction semantics. Once the metamodel is specified by the user, it can be used to configure GME itself to present a modeling environment specific to the problem domain. Every target system is specified in MILAN as a model. Model interpreters are the software components that translate the information captured in the models based on the input format required by the integrated tools and simulators.

MILAN design flow consists of modeling, performance estimation, and design space exploration. The design process is initiated by modeling the application and the target architecture. Application modeling involves application specification as a data-flow graph with alternatives [1]. While MILAN supports specification of both synchronous and asynchronous data flow graph, for our purpose, we use the modeling capability for the specification of synchronous data flow graph. The functional specification of the target system specifies the structure of the data-flow graph and the choice of implementations specifies the alternatives. MILAN supports hierarchical modeling that enables hierarchical specification of the data flow graph making it easier to manage and analyze an application model. Resource modeling involves modeling of the target architecture. The modeling paradigm is based on the GenM model [10]. The user identifies key components and features of the target architecture that can be exploited for optimization and models them in MILAN. In addition, the user also models various states and state transition costs associated with different components such as reconfigurable devices, processors supporting DVS, and power aware memories [10]. Finally, the user describes possible mappings for each task alternatives and different performance or compositional constraints that the system needs to satisfy [9]. A mapping is a relation between an application task and a target processing component. Performance constraints are based on the latency and energy dissipation requirements given as input. Constraints on composition restrict the composition of alternate processing components. For example, if a certain operating state is chosen to execute a task the same operating state needs to be chosen for the following task. Before we can perform design space exploration, we need to populate the design space described above using the performance estimates for all the mappings specified in the model. MILAN is a simulator integration environment. Hence, if appropriate simulators are integrated, MILAN has the capability to perform automatic simulation (using specified implementation and sample input) and update the model for mapping using the simulation results.

Once the complete system is modeled, the user invokes the design space exploration (DSE) tools. A DSE tool rapidly identifies a set of designs that satisfy all the constraints. MILAN provides Design Space Exploration Tool, DESERT, as the primary DSE tool [11]. DESERT uses symbolic methods based on Ordered Binary Decision Diagrams (OBDDs) for constraint satisfaction. The output of DSE is evaluated using High-level Performance Estimator (HiPerE) tool, currently integrated into MILAN. HiPerE evaluates system-level energy dissipation and latency. In order to provide a rapid estimate, HiPerE operates at the task level abstraction of the application. In addition to the task execution cost, various other aspects considered by HiPerE for accurate performance estimation are data access cost, parallelism in the system, energy dissipation when a component is idle, and state transition cost. Our results for signal processing applications show that HiPerE estimates are within 8% of the estimates using low-level simulations [9]. Using HiPerE, the designs are evaluated and compared manually to identify the final design.

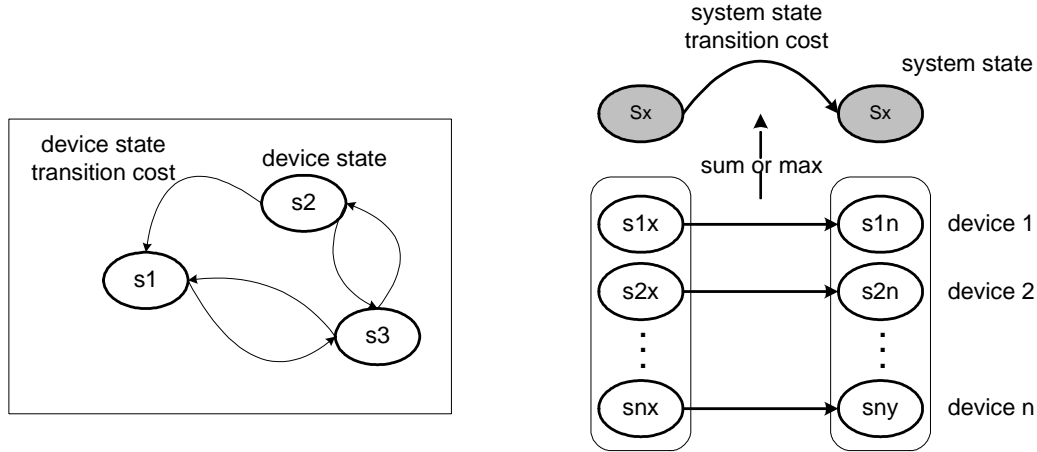
### 4. Mapping Applications onto ACS

Mapping applications onto ACS involves several steps. They are modeling the application and the target device(s), populating the model with required performance estimates, specification of constraints, and design space exploration to identify the required design. We will discuss our approach towards these steps.

#### 4.1 Modeling of ACS

A single device is modeled as a set of operating states and state transition costs (Figure 2). The operating states refer to operating frequency (voltage) in case of processors supporting dynamic voltage (frequency) scaling. For reconfigurable devices, operating states refer to configurations. Essentially, we model all possible operating states in which each tasks (of the application being mapped) can be executed. The edges between any two device operating states refer to the state transition cost in terms of latency and energy dissipation. For two or more devices, we use the following approach to define system operating states and state transition costs. Let's assume that we have  $n$  devices. We define a set of system states as a

set of n-tuples where each n-tuple refers to n states, one from each device. For a n-tuple with entries denoted as  $s_{ix}$ , where i can take values  $1 \dots n$ ,  $s_{ix}$  denotes a device operating state for device i. All the n-tuples are a unique combination of states. Figure 2 shows system operating states and their relation with the device operating states. For example, system operating state  $S_x$  consists of states  $s_{1x}, \dots, s_{nx}$  for n devices. The state transition costs for systems states is derived based on individual transition cost for the device operating states. For example, energy dissipation for a system operating state transition is the sum of individual device operating state transition cost for energy dissipation. In case of latency we assume parallel device state transitions and therefore the maximum value among all the transitions is used. Notice that, from modeling point of view, once the system operating states and the transition costs among them are identified they are fundamentally similar to device operating states and transitions among them. Therefore, throughout the paper we will use “operating state” as a reference to both device operating state and system operating state.



**Figure 2: Device states and system states and state transition costs**

#### 4.2 Modeling of a Linear Array of Tasks (LAT) and the Mapping

An application is modeled as a set of serially connected tasks (linear data flow graph). A task contains a set of sub-tasks that represent possible options for the implementation of the (parent) task. Sub-tasks represent all possible operating states of an ACS in which the task can be executed. Each option for a task contains necessary information to associate it with specific operating costs (latency and energy dissipation) and an operating state. We have extended the application model to model state transition. In order to do so, we introduce a pseudo task between each pair of consecutive tasks to represent state transition. This pseudo task is also associated with a set of choices or sub-tasks. These choices are the possible state transitions for the ACS.

Our modeling approach essentially transforms the application specification to a linear array of modules. A module can be a task or a pseudo task. Each module is associated with several choices which may differ in implementation costs which is latency and energy dissipation. The job of a design space exploration tool is to identify a single option for each module such that a given performance constraint is met. We will discuss performance constraints later in this section. While identifying a design, we have to ensure that the design is valid. While it is possible to choose any combination of choices for the tasks, the choice of implementation of the pseudo tasks cannot be chosen independently. The choice of implementation of a pseudo task depends on the choices made for the adjacent tasks. Such requirements are classified as compositional constraints. Our framework handles such requirements (see Section 5).

#### 4.3 Performance Constraints and Optimization Criteria

Our focus is single metric as well as multi-metric optimization. Single-metric optimization involves identification of the most efficient design based on a single performance metric. One example is to find the design with lowest energy dissipation. For multi-metric optimization we assume that we are expected to optimize on one metric while satisfying a requirement on all the other metric. For example, identifying the design with lowest energy dissipation that satisfies a given latency constraint is a multi-metric optimization problem. In this case, the latency constraint can be of the form “total latency of the application < 1000 ms”.

In addition, we may introduce additional implementation-specific constraints. These constraints may be a compositional constraint which indicates a dependency among the mappings of the tasks. One example may be that two tasks always have to be mapped to the same operating state.

#### 4.4 Using MILAN for Modeling and Exploration

MILAN already supports application modeling using the synchronous data flow graph which is used to model a linear data flow [9]. The MILAN resource model is extended to capture different operating states and state transition costs. The application model is associated with the resource model through mapping. A mapping of a task denotes execution of the task on a resource operating in a certain state. In MILAN, we model a complete design space. Therefore, with each task we associate all possible mappings. Once the design space is modeled, it is populated with the estimates for different parameters. These parameters include performance estimates for different mappings and state transition costs. The parameters are estimated using vendor provided data sheets for the target systems or using appropriate simulators.

MILAN already integrates DESERT an ordered binary decision diagram based design space exploration tool [11]. Given a design space and performance constraints, DESERT explores the design space and identifies the designs that meet the performance constraints. However, using DESERT, it is not possible to directly model state transition costs. Therefore, we have developed a technique which combines application modeling and constraint specification to model the optimization problems for ACS. We introduce a pseudo task between each pair of tasks to model state transitions. Each choice of mapping for the pseudo task uniquely corresponds to a possible system-state transition. Along with performance constraints (e.g. latency < 1000 mili sec.), DESERT also supports compositional constraints such as if a certain mapping is selected for a task, then a specific mapping should be selected for another task. We use compositional constraints to ensure that a valid system-state transition is selected based on the mappings identified for the tasks preceding and succeeding a system-state transition. Following modeling, DESERT is invoked to identify the designs that meet the constraints. DESERT does not identify a single optimal design. Instead, based on the constraints specified, DESERT identifies a set of designs that meet the constraints. Therefore, we use High-level Performance Estimator (HiPerE) to evaluate the pruned design space [10]. HiPerE evaluates the designs identified by DESERT based on their performance estimate. We have developed a design browser that provides a friendly graphical interface to compare the designs based on their performance and identify the optimal design.

The MILAN environment provides a user friendly graphical interface for modeling and constraint specification. It is also semi-automated in the sense that once the design space has been specified it performs optimization and automatically indicates the optimal design. If a simulator is integrated it is also possible to specify high-level implementation (e.g. C or VHDL) of a task and appropriate input stimulus that can be used to simulate the task to generate the performance estimates and automatically populate the model. In terms of speed, our solution is very fast. While it is not possible to derive such complexity bounds when DESERT is used for optimization, our experience shows that DESERT can handle traversal of a design space of size  $10^3$  to  $10^4$  in approximately 10 to 15 seconds.

### 5. Design Flow using MILAN

Using GME 2000 configured with the MILAN metamodel, design and optimization of a linear array of tasks (LAT) applications can be performed in five simple steps discussed below. GME 2000's user-friendly GUI allows the user to perform most of the actions associated with modeling by just dragging & dropping visual objects.

*Note: A version of MILAN that supports our approach is available for download at <http://www.isis.vanderbilt.edu/Projects/milan/downloads.asp>.*

#### Step 1

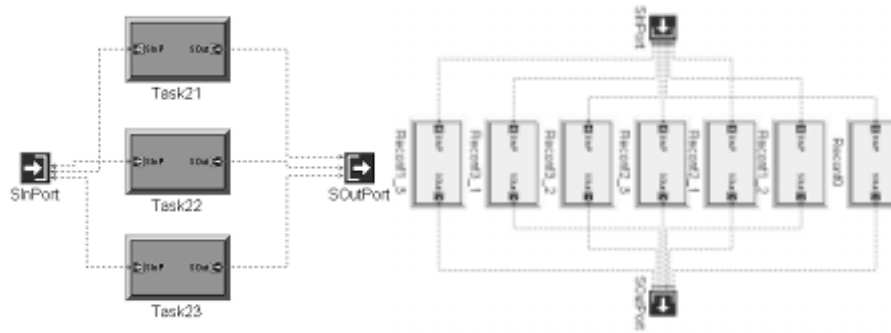


**Figure 3: Application modeling. Tasks are the lighter shaded boxes, and state transitions are the darker ones. Dataflow is indicated with directed connections (black arrows).**

As discussed in the previous section, we model a LAT application as a set of serially connected modules. We introduce a pseudo task between each pair of tasks to model state transitions (Figure 3). In short, an application is modeled as a linear array of serially connected modules with a pseudo module (or pseudo task) between each pair of consecutive application tasks to

model state transitions. From the modeling point of view, there is no difference between a task and a pseudo task. Both are SyncAlternative models. We have only used different colors to indicate the difference. In fact, while using MILAN, the default color is yellow. All the tasks should be connected in a linear fashion (as shown using the black arrows) to indicate the data flow. The small squares are known as ports and they are used to connect different modules. Modeling involves dragging & dropping appropriate modules from the Component view to the Model view and connecting them into a dataflow schema.

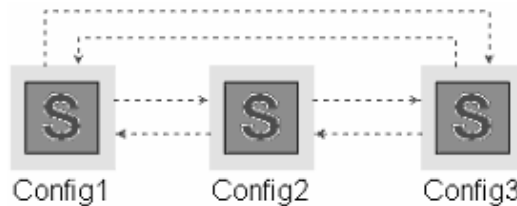
**Step 2**



**Figure 4: Task and state transition modeling. The dark boxes are the options of a task and light boxes are possible state transitions (options for a pseudo task)**

In this step, we model the options of implementation for tasks and state transitions. In terms of MILAN, an option is one of the possible implementations of a module. An option for a task is a possible operating state of the target ACS in which the task can be executed. An option for a state transition module is a possible transition that may occur. The model is then populated with estimations of execution and state transition costs. The performance estimates are stored in the Mapping aspect as a field in the model Configuration. Similar to the previous step, there is no difference between the choice for task implementation alternative and the alternatives for state transitions.

Our modeling environment can also capture additional information about the target application and devices. These information are not directly used by our design space exploration scheme but can be used as references. Modeling of operating states and state transition costs is one such example. In fact, we have actually captured this information in the application model shown in Figure 4. However, modeling the same information as shown in Figure 5 makes it easy to understand.



**Figure 5: Model of operating states and state transition costs**

Similarly, while we are just interested in the execution costs for each choice of implementation for a task, we can also capture the mapping information in the model. Figure 6 shows a model that captures the information that a task Task2 is mapped onto a device, FPGA device 1, which is configured as Config1. This model can be accessed in the Mapping aspect inside the model Configuration.



**Figure 6: The “Task2” box indicates a task option, the “FPGA device 1” box indicates the target device onto which the task is mapped, and the “Config1” box indicates an operating state**

### Step 3

In this step, we specify all the constraints. Constraints ensuring selection of the appropriate state transitions need to be specified. These constraints are of the form: “Task A is executed in State 1 and Task B is executed in State 2” implies “State transition between Tasks A and B is implemented by Option “Transition from State 1 to State 2”. Such constraints have to be specified for each combination of operating states for a pair of tasks. We also specify the overall latency and energy. The overall constraints are of the form: “total latency < 1000 ms”. We need to specify such constraints even for the metrics that are to be optimized. This is because DESERT only supports performance constraints of the form “<”, “=”, and “>” only. There is no support for specifying minimum or maximum. Therefore, care should be taken that for the metrics for which we are optimizing the constraints should not be so tight that no designs get selected. DESERT allows relaxing or tightening of constraints. So an iterative approach can be taken.

### Step 4

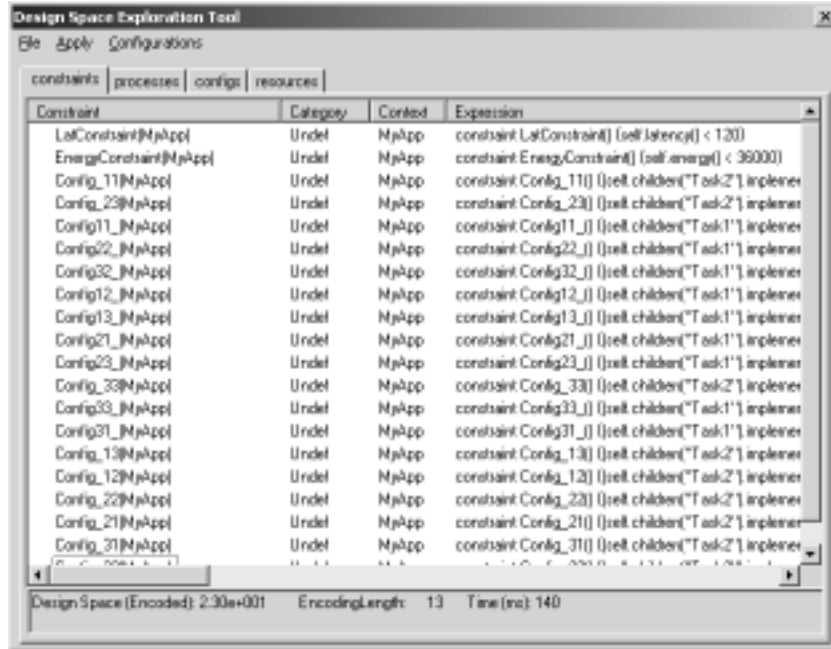


Figure 7: DESERT interface for constraint satisfaction

Once modeling and constraint specification is over, DESERT can be invoked. The initial DESERT window shows all the constraints specified in the model. While you can choose to apply some or all the constraints, in our approach we generally apply all the constraints. DESERT outputs the designs that satisfy the constraints. If the number of resulting designs is too big (small) then tighten (loosen) the constraints by going back to Step 3 and invoke DESERT again.

### Step 5

DESERT does not perform optimization and hence cannot identify the optimal design. Therefore, we use HiPerE which can evaluate the designs selected by DESERT and identify the optimal design. MILAN includes a design browser that reads in the output file generated by DESERT and allows a platform to apply HiPerE and compare designs based on latency and energy. HiPerE also provides an activity report per design evaluated to detail mappings and state transitions. Using the design browser the designer can choose the optimal design.

## 6. Illustrative Example

The PASTA application design problem is to identify an energy efficient mapping of an automated target recognition (ATR) application onto a heterogeneous embedded system while meeting the given latency constraint [13]. The underlying architecture for the PASTA project is already specified [13]. Hence, MILAN is used to identify an energy efficient mapping of the ATR application onto the PASTA hardware.

The hardware includes sensor(s), a processor, several microcontrollers, memories, and a radio. Each component can be independently turned on or off. In addition, the processor (Intel PXA 255) supports voltage and frequency scaling [17]. The

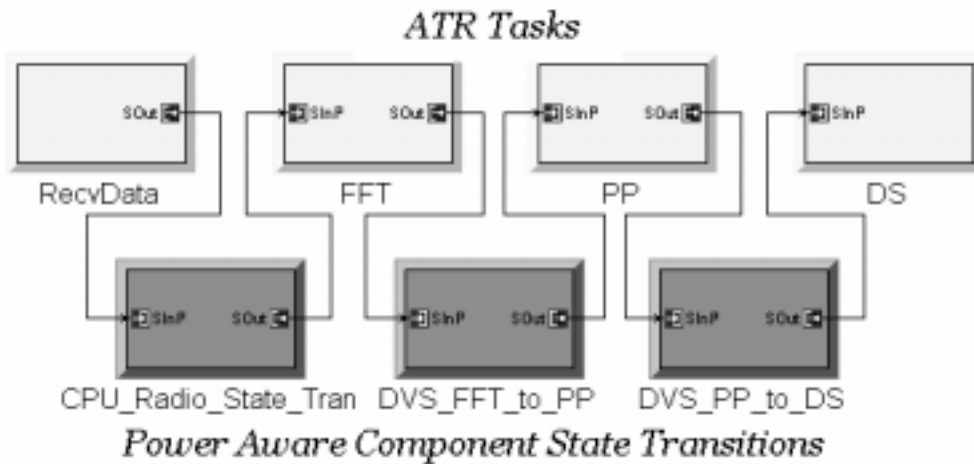
beamforming application consists of a linear array of 4 tasks. The first task is *receive data* which is mapped onto the radio and the microcontroller. The last three tasks that compute beamforming are *FFT*, *peak-pick*, and *delay sum*.

The design problem for PASTA involves identification of the operating state of each component for each task such that the complete ATR application dissipates the minimum energy while satisfying the latency requirement. All the components in the PASTA stack have at least two operating states; ON and OFF. There is a constant amount of time and energy spent to switch on each component. In addition, the processor has 6 different operating states. They are (99,50), (199,50), (299,50), (199,99), (299,99) and (399,99) where first value is the operating frequency of the processor and the second is the operating frequency of the bus. Tasks can be mapped onto the processor or the microcontroller. When mapped onto a processor, the task can be executed in a certain operating state and the performance of the mapping depends on the operating state. Transition between any two operating states also involves time and energy costs which depend on the source and destination states. However, we noticed that the transition costs between different operating states of the processor are negligible except one transition which involves changing operating frequency of the bus. Table 1 shows the energy and latency values in micro joule and micro seconds respectively for different options available for the tasks and the state transitions. Transition1 refers to the startup of Intel PXA 255 after receive data. The two options being, PXA is transitioning from idle to active or shut down to active. Two choices for receive data are if the processor is dissipating energy (by idling) or not. The six choices for FFT and Delay sum refer to execution of the tasks in each of the six operating state. The DVS columns refer to frequency transition costs when there is no transition (0,0) or if there is a transition from (f,50) to (F,99) where f is 99,199, or 299 and F is 199, 299, or 399.

The overall design space consists of 2,300 designs. We used simulators for Intel PXA 255 and the microcontroller to estimate performance of all the mappings. The start-up costs and state transition costs are estimated based on the data sheets provided by the vendors [17].

**Table 1: Energy, latency values for different options for tasks and state transitions**

Recv. data	Transition1	FFT	DVS	Pick-peak	DVS	Delay sum
113377, 655360	1100,10000	107989,431958	0,0	1983,7933	0,0	57989,231956
172359,655360	2700,15000	90665,215870	1100,10000	1259,1982	1100,10000	46868,115919
		64772,143937				34782,77292
		90665,215870				46868,115919
		64772,143937				34782,77292
		68556,107962				36813,57974



**Figure 8: Model of the PASTA application. Modules in the upper row are the tasks and modules in the lower row are the pseudo tasks for state transition**

Design space exploration involved DESERT and HiPerE. For design space exploration, the latency constraint was assumed to be < 1 sec. DESERT initially pruned the design space to 10 designs based on the latency constraint. DESERT does not include the performance of the sensor, radio, and the memory while evaluating the design space. Hence we used HiPerE to identify the design with minimum energy dissipation. In the resulting design, the components which are idle are switched off. For example, while the radio is receiving data, the processor is turned off and is turned on only when data is ready for

processing. The design also maps the false-alarm detection task onto the microcontroller as while latency is higher compared with the PXA processor, energy dissipation is lower.

**Table 2: Sample results using our approach**

<b>Constraint</b>	<b>Energy (micro Joule)</b>	<b>Latency (micro sec)</b>
Optimize energy	216890	888571
Optimize latency	278987	823278

Table 2 shows some sample results. We have identified two designs with minimum latency and energy dissipation. The table shows the latency and energy dissipation values for both the designs.

## 7. Conclusion

We presented a design framework to identify optimal mapping of applications, modeled as a linear data flow, onto adaptive computing systems. The strength of the framework is to be able to handle a general class of ACS system and a GUI based user-friendly design flow. This framework enables modeling of the target application and ACS device, specification of performance and composition constraints, and design space exploration. While the approach is slower than a dynamic programming based approach to find optimal design, it is able to handle multi-metric optimizations. In fact, the same modeling support can be used to drive an optimization tool that is based on dynamic programming based approach. We are currently focusing on a integrated environment that would support both the presented approach and a dynamic programming based approach for identification of optimal mapping of a linear array of tasks onto ACS.

## References

- [1] Agrawal, A. Bakshi, J. Davis, B. Eames, A. Ledeczi, S. Mohanty, V. Mathur, S. Neema, G. Nordstrom, V. Prasanna, C. Raghavendra, M. Singh, "MILAN: A Model Based Integrated Simulation for Design of Embedded Systems," Language Compilers and Tools for Embedded Systems, 2001.
- [2] K. Bondalapati and V. K. Prasanna, "Loop Pipelining and Optimization for Reconfigurable Architectures," Reconfigurable Architectures Workshop (RAW), May 2000.
- [3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, "Introduction to Algorithms," MIT Press and McGraw Hill, 2001.
- [4] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow," Proceedings of IEEE, Vol. 75, September 1987.
- [5] Generic Modeling Environment (GME). <http://www.isis.vanderbilt.edu/projects/gme/>
- [6] S. Huang, C. Chen, M. Ahmadi, and P. Mokrian, "An Optimal Variable Voltage Scheduling," The 2002 45th Midwest Symposium on Circuits and Systems, 2002.
- [7] Mentor Graphics FPGA Advantage. <http://www.mentor.com/fpga-advantage/>
- [8] Model-based Integrated Simulation. <http://milan.usc.edu/>
- [9] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis, "Rapid Design Space Exploration of Heterogeneous Embedded Systems using Symbolic Search and Multi-Granular Simulation," Language Compilers and Tools for Embedded System, 2002.
- [10] S. Mohanty and V. Prasanna, "Rapid System-Level Performance Evaluation and Optimization for Application Mapping onto SoC Architectures," IEEE Intl. ASIC/SOC Conference, 2002.
- [11] S. Neema, "System Level Synthesis of Adaptive Computing Systems," Ph.D. Dissertation, Vanderbilt University, Department of Electrical and Computer Engineering, May 2001.
- [12] J. Ou, S. Choi, and V. K. Prasanna, "Performance Modeling of Reconfigurable SoC Architectures and Energy-Efficient Mapping of a Class of Applications," IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), April 2003.
- [13] Power Aware Sensing Tracking and Analysis. <http://pasta.east.isi.edu/>
- [14] A. Sinha and A. P. Chandrakasan, "Energy Efficient Real-Time Scheduling," IEEE/ACM International Conference on Computer Aided Design, 2001.
- [15] Software Defined Radio. SDR Forum. <http://www.sdrforum.org/>
- [16] Xilinx System Generator for DSP. <http://www.xilinx.com/>
- [17] XScale: Intel PXA255 Processors. <http://www.intel.com/design/intelxscale/>