# Marshall Plan Scholarship Research Report

## Design Space Exploration, Dynamic Management Policy Optimization, and Hardware Acceleration for DPD Architectures

Name: Lin Li
Home Institution: Department of Electrical and Computer Engineering, University of Maryland, College Park, Maryland, USA
Host Institution: Fachhochschule Salzburg, Austria

5th September 2017

## Abstract

My work during the research period aims at proposing a novel framework for design space exploration and dynamic management policy optimization of adaptive embedded signal processing systems. In the framework, Markov decision processes (MDPs) are applied in a hierarchical way to enable autonomous adaptation of embedded signal processing under multidimensional constraints and optimization objectives. Thus, we name the proposed framework Hierarchical MDP framework for Compact System-level Modeling (HMCSM). The applications are implemented using dataflow modeling techniques. The framework integrates automated, MDP-based generation of optimal reconfiguration policies, dataflow-based application modeling, and implementation of embedded control software that carries out the generated reconfiguration policies. HMCSM systematically decomposes a complex, monolithic MDP into a set of separate MDPs that are connected hierarchically, and that operate more efficiently through such a modularized structure.

The effectiveness of our proposed MDP-based system design framework was demonstrated through an adaptive wireless communication application: a digital predistortion (DPD) system, which benefits from an implementation that can dynamically generate optimal control policies and reconfigure itself according to the generated policies. Additionally, new features were added to the current LIghtweight Dataflow Environment with the Verilog language (LIDE-V) to improve the power efficiency of dataflow-based hardware design and support efficient hardware acceleration of the targeted DPD system in this work.

# Contents

# 1 Introduction

Modern signal processing applications impose increasing demands of adaptivity, flexibility and reconfigurability. The systems also have more and more complex design spaces that are composed of multidimensional design parameters. This trend presents challenges at many levels of system design, implementation and optimization. On one hand, adaptive signal processing systems must adjust to dynamically-changing environmental conditions, system status or user requirements; on the other hand, the systems must often satisfy stringent constraints on energy-efficiency and real-time performance.

In wireless communication systems, high-power transmitters suffer from nonlinearities due to power amplifier (PA) characteristics, I/Q imbalance, and local oscillator (LO) leakage. Digital Predistortion (DPD) is an effective technique to counteract these impairments. To help maximize agility in cognitive radio systems, it is important to investigate dynamically reconfigurable DPD systems that are adaptive to changes in the employed modulation schemes and operational constraints. To help maximize effectiveness, such reconfiguration should be performed based on multidimensional operational criteria. With this motivation, during the research period, I developed a novel framework, named Hierarchical MDP framework for Compact System-level Modeling (HMCSM), for the design space exploration and dynamic reconfiguration policy optimization of signal processing systems, and I demonstrated the application of this framework through the design of a reconfigurable DPD application.

Dataflow provides a formal mechanism for expressing the functionality of digital signal processing (DSP) applications, and facilitates exploration of system optimization methods to achieve efficient implementations (e.g., see [1]). Dataflow models of computation are widely used for design and implementation of DSP systems. These applications require a wide variety of platforms and design tools and impose different kinds of constraints on system performance. Dataflow, as an important form of model-based design, is effective in terms of retargetability of design processes across different platforms. To demonstrate the proposed design methodology and experiment with alternative DPD architectures, we apply dataflow modeling techniques and associated libraries and tools to the design and implementation of DPD systems.

Our work on DPD design space exploration and dynamic reconfiguration policy optimization builds on our previous work on multiobjective optimization methods for digital signal processing systems [2]. In this previous work, evolutionary algorithms are adapted according to design specifications and applied to search the multidimensional system design space for a set of Pareto-optimized configurations. Some of the optimized configurations in the obtained set are then selected and extracted. This subset of selected configurations provides the collection of system modes that will be integrated into the targeted implementation. The set of operating modes provided in the selected configuration set is made available during operation such that system trade-offs can be reconfigured among based on statically or dynamically generated control policies.

Given a set of extracted Pareto-optimized system modes (configurations), the HMCSM framework applies Markov decision processes (MDPs) to generate policies that control the switching among these modes under multidimensional constraints and optimization objectives. MDPs have been used in many application areas as a foundation for dynamic determination of system configurations in

stochastic environments. Representative areas include artificial intelligence [3], mobile systems [4], and wireless sensor networks [5].

Various methods have been developed to improve the practical utility of MDPs in complex design problems involving dynamically adaptive systems. For example, Boutilier et al. propose factored MDPs as a method for compact representation of large, structured MDPs [6]. Benini et al. introduce a finite-state, abstract system model for power-managed systems [7]. In their approach, the system and its external environment are modeled as a service provider and a service requester, respectively, in the format of Markov chains. Each of these Markov chains has a set of states and a matrix of state transition probabilities. The computed state-to-policy mapping is then stored in a local memory or controller and is used in real time to dynamically reconfigure the system according to its current state.

However, the complexity of the MDP algorithms in general grows exponentially with increases in the size of the state space. Jonsson and Barto present an algorithm that performs hierarchical decomposition of factored MDPs to help alleviate this growth in complexity [8]. Their approach to hierarchical decomposition systematically allows irrelevant state variables to be ignored. However, their development of hierarchical MDPs is focused on algorithms and theoretical analysis for state abstraction and MDP computation, and the connection to implementation of the hierarchical MDPs and application to real-world systems is not addressed. One objective of this work is to help bridge this gap in the context of embedded signal processing systems.

In particular, in this work, the MDP schemes presented in [9] and [8] are integrated. This results in a novel approach to formulating MDPs for policy optimization in embedded signal processing systems with complex state spaces, and stringent implementation constraints. In the proposed design framework, hierarchical MDPs are applied to decompose the modeling of the application and embedded processing system into multiple MDPs. Each smaller MDP is formulated using an approach similar to that developed in [7]. This hybrid MDP approach is referred to as the *Hierarchical MDP approach for Compact System-level Modeling* (*HMCSM*).

To promote systematic derivation of embedded implementations using the HMCSM approach, we integrate the approach into the framework of dataflow-based design of signal processing systems. Model-based design in terms of dataflow graphs helps to ensure properties, such as determinacy, deadlock-free operation, and bounded memory requirements, which are of great importance in the reliable implementation of embedded signal processing systems (e.g., see [1]). Dataflow also orthogonalizes the implementation of individual functional components (actors) from the system-level control and coordination among the actors. This separation of concerns is especially useful because it enables efficient and reliable switching across different system-level configurations while reusing individual actors across the configurations. With these motivations, I develop in this work a dataflow-based framework for design and implementation of adaptive signal processing systems using HMCSM.

To demonstrate the efficiency and flexibility of the proposed design framework and the corresponding libraries and tools, we apply it to implement a DPD system that benefits significantly from adaptivity and run-time system reconfiguration. In the targeted DPD system, Pareto-optimized DPD parameters are derived subject to multidimensional constraints and predistortion trade-offs

are reconfigured among the different options to support efficient predistortion across time-varying operational requirements and modulation schemes. The design evaluation metrics (optimization objectives) targeted in the development of the DPD architecture are system energy consumption, adjacent channel power ratio (ACPR), and error vector magnitude (EVM).

Portions of our research on HMCSM are published in [10].

# 2 Background

In Section 1, MDP methods and dataflow-based design are introduced as two key foundations of the contributions in this work. In this section, I provide background in these areas that is relevant to development of the proposed HMCSM design framework.

## 2.1 MDP Methods

In Benini's work on MDP-based methods for system-level power management, the service provider, service requester, and power manager are defined as key system components. The policy is composed of a finite discrete sequence of decisions taken by the power manager. A generic deterministic stationary policy can be represented as a table with the rows representing all possible states and the columns representing all possible actions. The size of the policy table grows geometrically when the number of system states increases. As a result, the policies derived from the MDP techniques proposed in [7] are practical only for problems with relatively small numbers of system states.

A factored MDP only requires specification of the conditional probabilities with respect to dependent state variables, in contrast with traditional MDPs where the probabilities with respect to independent variables must also be specified. The resulting modeling components are smaller in size and their policies are more compact compared to traditional MDPs. A more detailed and systematic introduction to factored MDPs can be found in [6]. Based on the idea of factored MDPs, a number of factorization methods has been proposed (e.g., see [11]).

As mentioned in Section 1, hierarchical factored MDPs are explored by Jonsson and Barto [8]. They use a dynamic Bayesian network and a causal graph to identify relationships among state variables and construct a hierarchical MDP for a given policy optimization problem. In this work, we build on these theoretical foundations of hierarchical factored MDPs, and apply this class of MDPs to design and implementation of adaptive signal processing systems.

In our case study of a DPD architecture, we design and implement several hierarchical MDPs with different levels of design space complexity. This case study of a DPD architecture further demonstrates the utility of HMCSM MDPs. Specifically, through the case study, we demonstrate a systematic process for extending hierarchical MDPs to handle increasingly complex design spaces and explore trade-offs involving enhanced system performance (when more complex MDP formulations are applied) versus the design time and run time overhead of the more complex formulations. Details on our formulations of hierarchical MDPs for DPD architecture design are discussed in Section 6.5.

## 2.2 Dataflow-based Modeling and Design

An important contribution of this work is the integration of MDP-based design methods into a model-based design framework based on dataflow models of computation. In the form of dataflow that we apply, signal processing applications are modeled as directed graphs, called dataflow graphs, in which vertices (actors) represent computations of arbitrary complexity; edges represent first-in, first-out (FIFO) communication channels between actors; and actors represent discrete units of computation, called *firings*, that consume and produce well-defined amounts of data from and to the incident FIFOs [12].

Conceptually, data is encapsulated in objects called *tokens* as they pass through FIFOs from one actor to another. In signal processing oriented dataflow models, special attention is given to the rates at which actors produce and consume data to and from their ports, respectively. These rates are referred to as the production rates and consumption rates of the associated actor ports or incident edges. Collectively, production rates and consumption rates are referred to as *dataflow rates*. Analysis in terms of dataflow rates can be useful for many kinds of optimizations, such as those involving scheduling and memory management (e.g., see [1]). Different forms of dataflow have been proposed based on different restrictions on the dataflow rates or how dataflow rates across an actor or throughout a graph are related. Examples are cyclo-static dataflow [13], scenario-aware dataflow [14], and synchronous dataflow (SDF) [15].

In this work, a form of dataflow called *parameterized synchronous dataflow* (*PSDF*) is applied to demonstrate the model-based integration of the proposed MDP-based design techniques [16]. We use PSDF because it is useful in modeling dataflow graphs that have dynamically varying parameters. Quasi-static scheduling techniques have also been developed for these graphs that systematically derive *parameterized looped schedules* [17]. A parameterized looped schedule involves loops that iterate across subsets of actors, and have iteration counts that can be symbolic expressions in terms of static or dynamically-varying actor, edge or graph parameters.

Parameterized dataflow is applied due to its natural match with the objective of developing a model-based framework for adaptive signal processing. However, we envision that the framework can be adapted into modeling environments that are based on other parametric dataflow models, such as Boolean parametric dataflow [18] and the parameterized and interfaced dataflow meta-model [19]. Investigating such adaptations along with application of the distinguishing tools and analysis techniques available for such alternative models is an interesting direction for future work.

To implement PSDF-based models of adaptive signal processing systems, we apply the Lightweight Dataflow Environment (LIDE) [20]. LIDE is a software tool for dataflow-based design and implementation of signal processing systems. LIDE is based on a compact set of application programming interfaces (APIs) that is used for constructing and connecting dataflow actors, edges, and graphs. LIDE includes facilities for dynamically manipulating actor, edge and graph parameters. We use these facilities to incorporate PSDF semantics in the LIDE-based implementations that we develop when applying the HMCSM framework. Due to LIDE's lightweight and flexible feature, it can readily be targeted to different implementation languages such as C, CUDA and MATLAB [21]. When the system is implemented in a particular language XYZ, we refer to the result-

ing specialized version of LIDE as LIDE-XYZ. In this work, we apply LIDE-C, which is based on a C language implementation of the LIDE APIs. A useful direction for future work is to achieve hardware acceleration for DPD subsystems using a LIDE-based digital hardware design methodology. Details on this direction for future work are discussed in Section 5.

# 3 LIDE-based Implementation

In this work, we employ LIDE as the implementation approach for dataflow modeling semantics. In the design of each actor, LIDE requires four basic interface functions, *construct*, *enable*, *invoke* and *terminate*. The construct function instantiates an actor and initializes its configuration. The enable function returns a Boolean indicator about whether or not the actor can be fired based on its current mode, and the current state of its input and output buffers. The invoke function executes the actor in its current mode. The terminate function performs operations for deallocating the actor when it is no longer needed in the enclosing system.

We target our implementation to the C language and extend LIDE-C by providing new capabilities to support hierarchical dataflow modeling and PSDF semantics. The key features and utilities added in the new version of LIDE-C are summarized as follows. For details on the original version of LIDE that we have extended as part of this work, we refer the reader to [22].

- A graph abstract data type (ADT) has been created to enable hierarchical semantics in LIDE-C. A graph ADT in LIDE-C contains a set of actors and edges in the graph and is associated with a scheduler that coordinates execution of actors in the graph when it is executed. An actor within the graph associated with a graph ADT instance $I$ can encapsulate another graph ADT instance $J$, thereby allowing for hierarchical design of dataflow graphs.

- The actor ADT has been extended to support implementation of hierarchical dataflow in LIDE-C. The address of an array $A$ that contains pointers to graph ADT instances has been added to the common data structure for actors so that an actor can be mapped to one or more subgraphs that are nested hierarchically within it. An actor can be mapped to different subgraphs based on different conditions. When an actor is associated with one or more subgraphs, the actor is called a *hierarchical actor*; otherwise, the array $A$ is empty, and the actor is called a *primitive actor*. An actor is executed from within graphs that contain it through a uniform interface regardless of whether the actor is hierarchical or primitive.

- Like a primitive actor, a hierarchical actor also has a set of modes, where each mode has fixed production and consumption rates that are associated with the actor ports. The enable function of a hierarchical actor checks the executions condition of the current mode in relation to the buffers in the enclosing graph that are connected to the input and output ports of the actor. The invoke function of a hierarchical actor executes the scheduler that is associated with the actor.

- To support the implementation of PSDF, an ADT for PSDF specifications has been developed in LIDE-C. The PSDF specification ADT inherits from the graph ADT. The three core components of a PSDF specification are modeled as three hierarchical actors (init, subinit, and body actors) in the PSDF specification ADT. Each of these actors encapsulates a subgraph that specifies the functionality of the corresponding PSDF specification component. The scheduler for PSDF specifications is defined according to PSDF semantics. A PSDF specification can be nested within a higher level graph by being encapsulated as a hierarchical actor.

These new features in LIDE-C enable important new capabilities for model-based design that are relevant to MDP-based design optimization for adaptive signal processing systems.

# 4 Hierarchical MDP Approach

## 4.1 HMCSM Framework

The HMCSM framework is illustrated in Figure 1. At design time, the application functionality is modeled using dataflow techniques, as illustrated in the top right region of the figure. The design process also involves modeling the environmental and system-level dynamics, and reconfiguration process in the form of a hierarchical MDP, as illustrated by the part of Figure 1 that is labeled Hierarchical MDP Subsystem. As part of this modeling process, Markov models are created of both the processing demands imposed on the system by the application, and the dynamics of the processing system components. In a classical MDP formulation, these elements are combined into a single MDP. In this work, I additionally explore the use of Hierarchical MDPs in comparison to a single MDP to address the scaling problems that are well known to be a major weakness of classical MDPs (see Section 1).

The single MDP is transformed into a hierarchy of multiple MDPs by first factoring the elements in the MDP based on their stochastic interdependencies. Once the MDP has been factored, it can be decomposed into sub-problems that can be independently solved by multiple MDPs arranged in a hierarchy. For details on the processes involved in factoring and transforming the original MDP, reader is referred to [8]. In the HMCSM framework, the factoring and decomposition are carried out by hand, using knowledge of the application domain and the processing system. An interesting future direction for this work is the development of tools to help automate the factoring and decomposition process.

The Configuration Control Machine (CCM), shown in the lower (run time) portion of Figure 1, is used to manage dynamic system-level reconfiguration throughout operation of the embedded signal processing system. Here, by *dynamic reconfiguration*, we mean changes to any system-level parameters, including software, platform, and algorithmic parameters, that can be manipulated while the system is executing. At run-time, the CCM executes periodically, where the period $T_c$ of its execution is a system parameter.

We refer to each periodic execution of the CCM as a *reconfiguration round*. During a given reconfiguration round, the CCM determines, based on the current environmental state and system state, whether or not to perform a dynamic reconfiguration operation. Furthermore, if the determination is to perform such

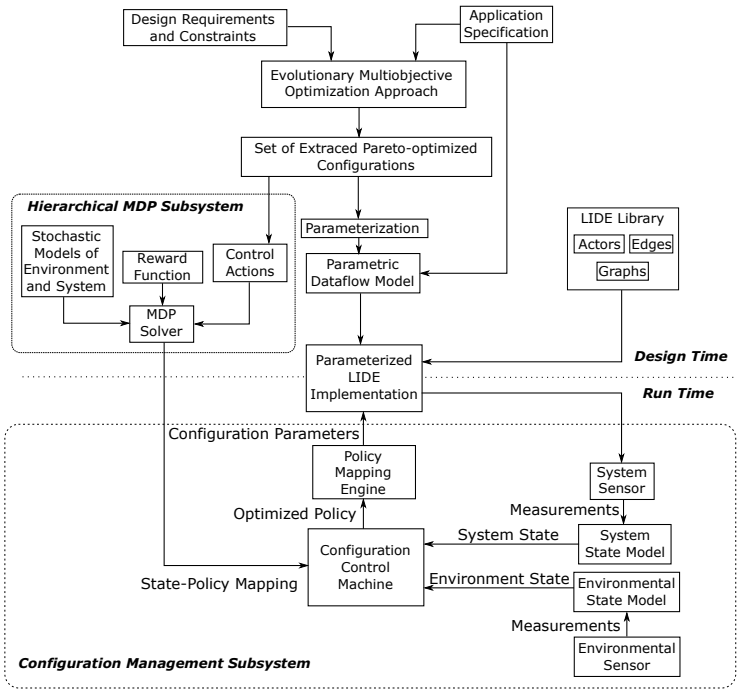


Figure 1: An illustration of the HMCSM framework for design and implementation of adaptive signal processing systems.

an operation, the CCM also determines the specific reconfiguration operation that is to be applied to the system. The blocks labeled System Sensor, System State Model, Environmental Sensor, and Environmental State Model represent measurements and models that are used by the CCM to determine the system and environmental state during a given reconfiguration round.

The block labeled *Control Actions*, in the design time portion, encompasses the set of possible reconfiguration operations that can be applied by the CCM in a given reconfiguration round. Examples of control actions in the context of HMCSM are changes to the type of digital filter that is applied to process a given signal in the application flowgraph, changes to the coefficients in a filter of a fixed type, and changes to the input port from which a given actor will read input data.

If $A$ denotes the set of control actions, then the CCM can be viewed as an implementation of a function $P : S_e \times S_s \rightarrow A$, where $S_e$ is the set of environmental states, and $S_s$ is the set of system states. This function $P$ is referred to as the *reconfiguration policy* or simply "policy" in an adaptive signal processing system that is developed using the HMCSM framework. In HMCSM, the policy is applied at run-time, and derived at design-time. It is derived using an MDP Solver, which is a software module that automatically generates optimized policies from MDP model specifications.

The Policy Mapping Engine, shown near the center of Figure 1, translates control actions into updates to dynamic parameters in the embedded software that achieve the intended actions. In the implementation of the HMCSM frame-

work, these parameter updates are made by setting appropriate variables in an implementation of the application dataflow graph that is developed using the Lightweight Dataflow Environment (LIDE) (see Section 2.2). This dataflow graph implementation is represented by the block labeled Parameterized LIDE Implementation, and the software tool that we use to construct this implementation is represented by the block labeled LIDE Library.

In the HMCSM framework, each control action is formulated in terms of specific changes to specific parameters in the parameterized dataflow application model $M$. In other words, a given control action $A$ can be represented as $A = \{(p_1, v_1), (p_2, v2), \ldots, (p_{N(A)}, v_{N(A)})\}$, where each $p_i$ is a distinct parameter in the application model $M$, each $v_i$ is an admissible value of parameter $p_i$, and $N(X)$ is the number of parameters in $M$ that are manipulated by a given control action $X$. Execution of the control action $A$ at run-time involves setting each parameter $p_i$ to the corresponding value $v_i$ such that subsequent operation of $M$ will be performed using these new parameter settings. Operation continues with the new parameter settings until a new control action is applied to the system (in some subsequent reconfiguration round).

The formulation of an MDP in HMCSM includes three main components, which are represented by the blocks in Figure 1 that are labeled Stochastic Models of Environment and System, Reward Function, and Control Actions. These components are developed by hand using well-established foundations of MDP modeling, along with domain knowledge of the targeted application.

We have discussed the Control Actions part of the MDP formulation earlier in this section. The Stochastic Models of Environment and System include, for each of the two models (environment and system), the definition of the state space and the state transition matrix (STM). The STM is a stochastic matrix that defines the probability of the transition to the next state given the existing state, and conditioned on a given action. Intuitively, the Reward Function maps state-action pairs into *scores* that assess the utility of performing the associated action (control action) during the given state. An approach is applied for incorporating multidimensional design objectives into the scores produced by the reward function. For details on this multidimensional reward function approach, the reader is referred to [9].

In summary, the HMCSM framework presented in this section provides a comprehensive methodology and supporting tools for design and implementation of adaptive embedded signal processing systems. The specific tools that are applied in the demonstration of the framework are MDPSOLVE [23] and LIDE, which correspond to the blocks labeled MDP Solver and LIDE Library, respectively, in Figure 1. However, the framework is not intended to be specific to these tools, and can readily be adapted to other tools for solving MDPs and implementing parametric dataflow graphs, respectively.

## 4.2 Example: Channelizer/receiver Application

In this section, I describe a detailed case study as an illustrative example of the proposed framework. In addition to providing a demonstration of the proposed HMCSM design framework on a practical, adaptive signal processing application, the case study also serves as a demonstration of a novel, application-specific, MDP-based system design. The example is an adaptive wireless communication receiver that dynamically optimizes its system configuration in response to

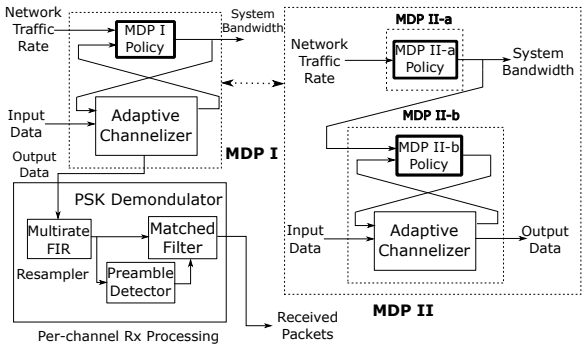


Figure 2: Block diagram of receiver signal processing and two MDP schemes.

changes in different use cases. As a key part of the receiver, the *channelizer* extracts multiple radio channels of distinct bandwidths from a digitized wideband input signal. Among various computing components of the receiver, the channelizer operates at the highest sampling rate in the system and accounts for most of the computational complexity and energy consumption [24]. By adapting the configuration of the channelizer based on the communication scenario, we seek to optimize its energy efficiency while ensuring that it extracts the number of channels that is required by the communication scenario at any given time. We design an HMCSM MDP to perform this adaptation, and apply the dataflow-based MDP implementation framework to realize the resulting adaptive signal processing on a state-of-the-art embedded processing platform.

### 4.2.1 Adaptive Receiver Architecture

Our example is a wireless receiver for a Low Power Wide Area Network (LPWAN) used in a "Smart Cities" Internet of Things (IoT) application [25]. We propose an adaptive LPWAN receiver that dynamically adjusts the system bandwidth continually, and periodically transmits the new bandwidth setting to end nodes through the use of a downlink beacon. This case study implements the physical layer signal processing for such an adaptive receiver. The implemented architecture consists of reconfigurable channelizer and baseband processing algorithms. In this work, we build on the MDP-based dynamically reconfigurable channelizer presented in [9].

### 4.2.2 Application Specification

Figure 2 shows a block diagram of the adaptive receiver architecture that is investigated in this case study. The channelizer is used to separate the incoming wideband signal into multiple data streams, where each of the data streams is associated with a distinct channel. Each channel is then oversampled for symbol timing recovery, and then processed by a Generalized Likelihood Ratio Test (GLRT) detector, which looks for the transmission preamble. Once a detection is successful, a matched filter demodulator recovers the transmitted data and confirms it with an error detection function (e.g., CRC32).

We compare the relative merits of two separate MDP schemes, labeled MDP-I and MDP-II. These two schemes are illustrated together in Figure 2. MDP-I

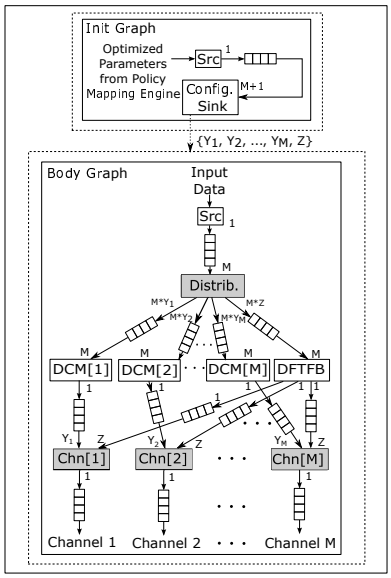


Figure 3: PSDF specification of channelizer subsystem.

consists of a single MDP employed for control of both the dynamic bandwidth as well as the channelizer processing configuration. MDP-II splits the modeling into two MDPs arranged in a hierarchy. These two MDPs are labeled MDP-II-a and MDP-II-b.

The channelizer can be implemented by one of two options, as detailed in [9] — a bank of $M$ polyphase decimators and mixers (labeled as DCM[1], DCM[2], ..., DCM[$M$]) or a Discrete Fourier Transform Filter Bank (labeled as DFTFB). Both options are capable of meeting the processing requirements of the channelizer subsystem.

However, they do so using different algorithmic means and the relative efficiency of one option compared to the other is highly platform- and implementation-dependent. Using our proposed MDP-based approach, the reconfiguration policy is systematically optimized for the exact processing characteristics of a specific platform (e.g., measured power consumption).

### 4.2.3 PSDF Model for Channelizer

We implement the channelizer subsystem as a PSDF design with dynamic parameters. Our PSDF model of the channelizer system is illustrated in Figure 3. Here, $M$ is a static parameter of the application that represents the total number of available channels.

At run-time, the MDP-generated reconfiguration policy determines how many channels to enable, based on the data rate, and whether to apply DCM processing or DFTFB processing. These policy decisions are used to manipulate a set of dynamic parameters $\{Y_1, Y_2, \ldots, Y_M\}$ that is associated with $M$ distinct DCM actors DCM[1], DCM[2], ..., DCM[$M$], respectively. The policy decisions are also used to manipulate a parameter $Z$ that is associated with an actor labeled DFTFB, which represents DFTFB processing on all of the enabled chan-

nels. Each of these $(M+1)$ parameters is binary valued. In particular, for each $i \in \{1, 2, \ldots, M\}$ if DCM processing is enabled, and the $i$th channel is enabled, then $Y_i = 1$ and $Z = 0$. Conversely, if DFTFB processing is enabled, then $Z = 1$, and $Y_i = 0$ for all $i$.

After each reconfiguration round, updated values of $\{Y_1, Y_2, \ldots, Y_M\}$ and $Z$ are propagated to the adaptive channelizer flowgraph through the Init Graph, as illustrated in Figure 3. The channelizer flowgraph is parameterized in terms of these $(M+1)$ dynamic parameters and encapsulated within the Body Graph, as shown in the figure. The Init Graph and Body Graph are modeling constructs in PSDF that are used, respectively, for reconfiguration functionality (determination and propagation of new parameter values), and core signal processing functionality associated with an application.

The Config. (configuration) Sink actor in Figure 3 is a special actor in LIDE that is used to propagate parameter updates from the Init Graph to the Body Graph in PSDF-based implementations. The shaded actors in the Body graph are dynamically parameterized actors. The expressions next to the input and output ports represent the consumption and production rates associated with the ports, respectively. The Distrib. (distributor) actor takes its input data and distributes copies of it to the appropriate subset of DCM/DFTFB actors depending on the current values of the dynamic parameters in the graph.

The actors labeled $\text{Chn}[1], \text{Chn}[2], \ldots, \text{Chn}[M]$ in Figure 3 copy data from their input ports to their output ports based on which (if any) input ports have nonzero consumption rate. These actors generate samples on each of the $M$ output channels of the channelizer subsystem.

We compare the relative merits of a conventional, monolithic MDP approach (labeled MDP-I) with our proposed hierarchical MDP approach (labeled MDP-II) to adaptive signal processing system design.

#### 4.2.4 MDP-I

In MDP-I, the (single) MDP state space consists of the instantaneous rate of uplink packets generated by all end nodes, and the configuration of the basestation processor (number of channels enabled and channelizer configuration).

The action space consists of the next configuration of the basestation processor (number of channels enabled and channelizer configuration).

The STM for the packet generation rate is computed a priori from the traffic rate measured in a design-time simulation. The STM for the processing system is obtained from the dynamics of the implementation on the reference platform.

The MDP reward metric is a linear combination of two competing metrics: the probability of packet collision and the power consumption expended in basestation processing. This linear combination of conflicting metrics tasks the MDP with finding an optimal trade-off at any time based on relative weightings that are provided by the designer for the two metrics.

#### 4.2.5 MDP-II

In MDP-II, the control task is split across two hierarchically arranged MDPs: MDP-II-a and MDP-II-b.

MDP-II-a is used to determine the optimal number of channels to enable at a given time, while being agnostic to what processing configuration is actu-

ally used in the receiver to implement that configuration. MDP-II-b is used to determine the optimal processing configuration for an exogenously specified number of channels.

### 4.2.6 Implementation

The configurable components of the processing system were implemented using the method detailed in Section 4.1, and deployed to a Raspberry Pi 3 Model B computing platform. Each of the available configurations was run in a test mode, and the average processing power consumed was measured and tabulated as shown in Table 1. The device we used for measuring power consumption is the Tektronix Keithley Series 2280 Precision Measurement DC Power Supply. The measurements were then provided as input to the MDP models, and used to generate control policies using these empirically measured characteristics of the processing system.

Table 1: Platform measurements.

| Processing Configuration | Number of Channels Processed | Average Power |
|---|---|---|
| DCM | 1 | 1.4406 W |
| DCM | 2 | 1.4781 W |
| DCM | 3 | 1.5203 W |
| DCM | 4 | 1.5660 W |
| DCM | 5 | 1.6025 W |
| DCM | 6 | 1.6524 W |
| DCM | 7 | 1.7013 W |
| DCM | 8 | 1.7453 W |
| DFTFB | 8 | 1.6754 W |

### 4.2.7 Simulation Results

We performed a physical layer signal processing simulation in MATLAB to generate uplink traffic from 1,000 end nodes. The simulation compared the use of the two (adaptive) MDP Schemes and also (non-adaptive) cases where only the fixed channelizer configurations were used. Each run of the simulation generated results of the form shown in Figure 4. In this figure, the MDP (MDP-I) is in control of the number $R$ of receiver channels enabled. Note that when the traffic rate is high, the system increases $R$ in order to reduce packet collisions.

The two competing MDP schemes, MDP-I and MDP-II, produced the same control policy. However, a key difference is that the hierarchical MDP reduced the model size from 1.63MB to 265kB, which is a factor of 6.1 times smaller than the original size. This reduction becomes especially relevant when housing the MDP model and policy generation code on the processing platform itself. Such an *embedded MDP* realization is useful because it allows the MDP and generated policy to adapt dynamically based on learned characteristics of the operating environment. Integration of embedded MDP techniques into the HMCSM framework is a useful direction for future work.

MDP-II also provides a benefit in the execution time required for the MDP solver to compute the optimal policy. We applied the MATLAB-based open source solver called MDPSOLVE [23]. The execution times were measured as 294ms for MDP-I, 50.8ms for MDP-II-a and 41.5ms for MDP-II-b.

As a result, in a deployment with a fixed processing system that periodically re-computes the control policy in response to a changing external environment, the hierarchical MDP scheme reduces the solver time from 294ms to 50.8ms, which is a factor of over 5.7X smaller.

All of the simulation runs that we carried out are compared in Figure 5. Different simulations for the MDP-generated policy were carried out using different relative weightings for the two performance metrics. Simulations were also carried out for fixed signal processing configurations. The square-shaped points in Figure 5 correspond to DCM-based channelizer operation with the number of enabled channels ranging from 1 to 8. The fixed configuration DFTFB case is represented by the triangle-shaped point in the top left region of the figure. In the context of all of the design points evaluated and the two performance metrics, we see from the figure that the MDP-based approach generates a Pareto front. This demonstrates that the adaptive reconfiguration capability provided by the MDP leads to better performance in comparison with fixed configurations for each of the available signal processing algorithms.

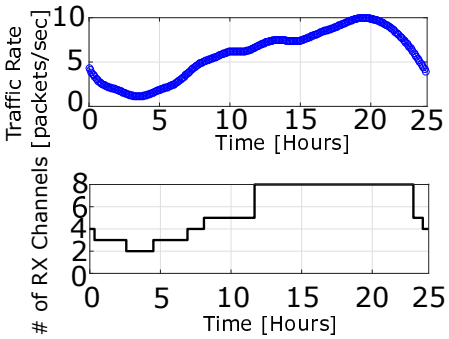


Figure 4: Simulation results for MDP-I.

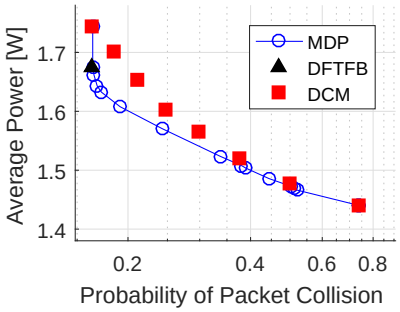


Figure 5: Comparison among MDP-generated policies and fixed-configuration designs.

# 5 Extensions to the LIDE-V Library

Based on the analysis in [26], where a dataflow model is constructed for the DPD algorithm in [27], most of the computation and energy consumption is concentrated in the filter actors. Thus, in our future work, we plan to implement all the predistortion filters in the filtering stage on an FPGA platform to accelerate the data processing. To facilitate the power efficient hardware implementation of DPD systems, we have extended the LIDE-V library to integrate low power techniques.

In LIDE-V, the enable, invoke and scheduling functions for an actor are implemented as three coupled Verilog modules, which we refer to as the actor enable module (AEM), actor invoke module (AIM) and actor scheduling module (ASM), respectively [28]. Dataflow edges are implemented as dataflow edge modules (DEMs) to provide communication channels for connections between actors. Since DEMs buffer data through a first-in first-out protocol, we also refer to them simply as FIFOs.

The new features added to the LIDE-V library can be summarized as the following:

- We have integrated into the LIDE-V library two low power methods: asynchronous design with multiple clock domains and clock gating.
- We have integrated support for hierarchical actor design in LIDE-V. This advancement provides useful new features in LIDE-V for design of complex applications and reuse of dataflow subgraphs.

We have demonstrated the flexibility and power efficiency of the extended LIDE-V library through a deep neural network application for vehicle classification. Our envisioned future work in this direction is to apply the updated LIDE-V library to implement the DPD system designed in this work.

# 6 Case study: Digital Predistortion Application

## 6.1 Background

In wireless communication systems, I/Q mismatch, power amplifier (PA) nonlinearities, and signal leakage in the local oscillator (LO) are implementation-related problems that must be addressed before the direct-conversion principal can be deployed. In the frequency domain of the transmitted signal, the effects of these impairments are translated as power leakage into adjacent channels. DPD is a widely investigated technique (e.g., see [26, 29–32]) to counteract such impairments by applying carefully-calculated distortion to the signal prior to transmission.

## 6.2 Adaptive Dataflow-based DPD Architecture

The DPD architecture developed in this report is based on the algorithm presented in [27]. This DPD algorithm operates in two stages. In the *coefficient estimation* stage, the DPD filtering coefficients are estimated. The estimated coefficients are then employed in the *DPD filtering* stage for actual predistortion of the input signal.

The structure of the predistortion filtering system is shown in Figure 6. The DPD system is split into two branches, namely direct and conjugate predistortions. The output of the predistortion filter can be expressed as

$$z_n = \sum_{p \in I_P} f_{p,n} \star \psi_p(x_n) + \sum_{q \in I_Q} \bar{f}_{q,n} \star \psi_q(x_n^*) + c' \ , \tag{1}$$

where $\star$ denotes convolution; $x_n$ and $x_n^*$ are the direct and conjugate input samples, respectively; $I_P$ and $I_Q$ are the employed sets of direct and conjugate term orders, respectively; $\psi_p$ and $\psi_q$ are *polynomial basis functions* for the direct and conjugate branches, respectively; $f_{p,n}$ and $\bar{f}_{q,n}$ are the FIR filter coefficients for the direct and conjugate polynomials, respectively; and $c'$ is the LO leakage compensation component. The maximum polynomial order used can be different for the direct and conjugate branches of the predistorter [27].

Given $r \in \{p, q\}$, the polynomial basis function $\psi_r$ can be expressed as

$$\psi_r(x_n) = \sum_{k \in I_r} u_{k,r} |x_n|^{k-1} x_n, \ r \in I_R \ , \tag{2}$$

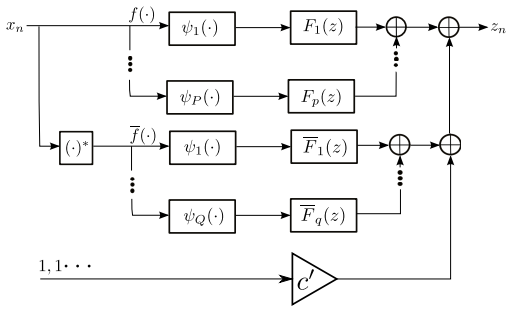


Figure 6: Predistorter structure for the joint predistortion of PA and I/Q modulator impairments.

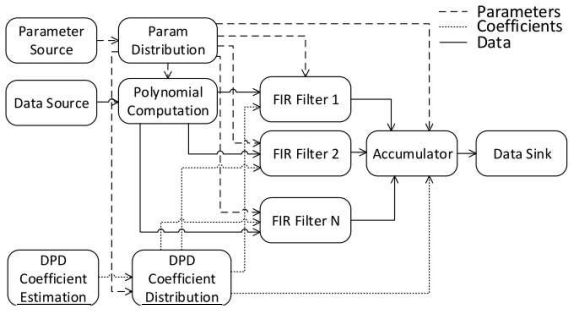


Figure 7: Dataflow model of DPD coefficient estimation.

where $I_R$ denotes the set of term orders employed in the given DPD configuration ($I_R = I_P$ if $r = p$, and $I_R = I_Q$ if $r = q$); $I_r$ denotes the subset of $I_R$ that contains only term orders up to $r$ in $I_R$; and $\{u_{k,r}\}$ denotes the polynomial weights. Here, given a polynomial $\rho = a_0 + a_1 x + \ldots + a_n x^n$, we define each monomial $a_i x^i$ to be a *term* of $\rho$, and we define $i$ to be the associated *term order*. According to [27], only odd-order polynomials are used to avoid the computation of the square-root within $|x_n|^{k-1}$, which is a computation-saving option that has been applied in the proposed implementation.

## 6.3 Application Specification

Dataflow modeling of the DPD is complex because of the high degree of parameterization. As mentioned in 6.2, the DPD architecture has been split into two parts, the DPD filter and DPD coefficient estimation, as shown in Figure 7 and Figure 8, respectively. The interconnection between the two diagrams can be inferred from the placeholder actors DPD Coefficient Estimation and DPD Filter in Figure 7 and Figure 8. Also, the Parameter Source and Param Distribution actors have same functionalities in the two dataflow models and have been replicated for design simplicity.

The implementations of the actors in the two dataflow graphs are summarized as follows.

- **Source and Sink:** source actors are used for file reading. They read a file

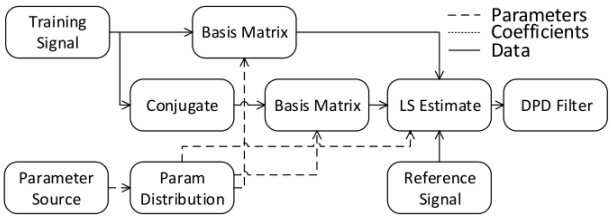


Figure 8: Dataflow model of the predistortion filter.

that contains data and move the acquired values to their output FIFOs. Similarly, sink actors consume data from their input FIFOs and write it to an output file.

- **Param Distribution:** This actor distributes updated parameter values to parameterized actors.
- **Conjugate:** The conjugate actor computes the complex conjugate of the received sample value and writes it to its output FIFO.
- **Basis Matrix:** This actor performs the basis matrix computation introduced in Section 6.2.
- **LS Estimate:** This actor performs the least squares (LS) estimation operation.
- **DPD Coefficient Distribution:** The actor reads and distributes estimated filter coefficients to FIR filters.
- **Polynomial Computation:** This actor computes the polynomial basis function according to Equation 2 for both non-conjugate and conjugate branches of the DPD filter.
- **FIR Filter:** This actor performs the FIR filtering operation part of Equation 1.
- **Accumulator:** This actor combines the output from all FIR filters and adds the LO leakage compensation component according to Equation 1.

## 6.4 PSDF Model

The DPD architecture introduced in Section 6.3 is implemented as a PSDF design with dynamic parameters. A PSDF model of the DPD system is illustrated in Figure 9. The DPD system is parameterized by the number of direct predistortion filters, the number of conjugate predistortion filters, and the order of each deployed filter. Each of these parameters can be reconfigured at run-time.

At design time, we determine the maximum number of available filters for direct and conjugate predistortion, labeled $M$ and $N$, respectively. Thus, in Figure 9, there are $(M+N)$ FIR filters in total, labeled FIR Filter 1, FIR Filter 2, ..., FIR Filter $(M+N)$. At run-time, the MDP-generated reconfiguration policy determines the current system configuration, which includes how many

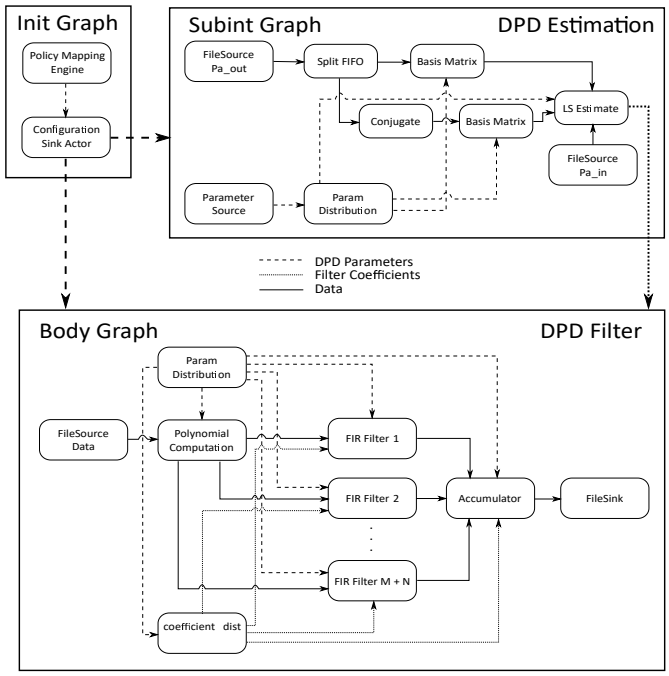


Figure 9: PSDF model of DPD architecture.

filters are deployed for direct and conjugate predistortion, respectively, and the corresponding order of each deployed filter. These policy decisions are used to manipulate a set of dynamic parameters $\{P, Q, F_1, F_2, \ldots, F_M, \bar{F}_1, \bar{F}_2, \ldots, \bar{F}_N\}$, where $P$ and $Q$ are the number of deployed filters for direct and conjugate data processing, respectively; $F_1, F_2, \ldots, F_M$ are the orders of the $M$ direct data filters; and $\bar{F}_1, \bar{F}_2, \ldots, \bar{F}_N$ are the orders of the $N$ conjugate data filters. Note that when a filter is not deployed in the current configuration, the corresponding order of that filter is set to 0.

After each reconfiguration round, updated values of $\{P, Q, F_1, F_2, \ldots, F_M, \bar{F}_1, \bar{F}_2, \ldots, \bar{F}_N\}$ are propagated to the adaptive DPD flowgraph through the PSDF init graph. The DPD flowgraph is parameterized in terms of these dynamic parameters and encapsulated within the subinit graph and the body graph. The subinit graph is a modeling construct in PSDF that can change body graph parameters just before each execution of the body graph. Examples of such parameters include the coefficients of FIR filters and the phases of downsamplers and upsamplers.

In our PSDF-based DPD architecture design, the DPD coefficient estimation subsystem is mapped to the subinit graph. This subsystem estimates coefficients for the FIR filters that are deployed in the current configuration. The computed coefficients are then propagated to the body graph as parameter updates. Given the system configuration parameters from the init graph and the filter coefficient parameters from the subinit graph, the body graph performs the predistortion of input data based on the updated DPD parameter settings. In the body graph, the first $(P + Q)$ FIR filters are switched on for predistortion filtering and the rest of the filters are disabled.

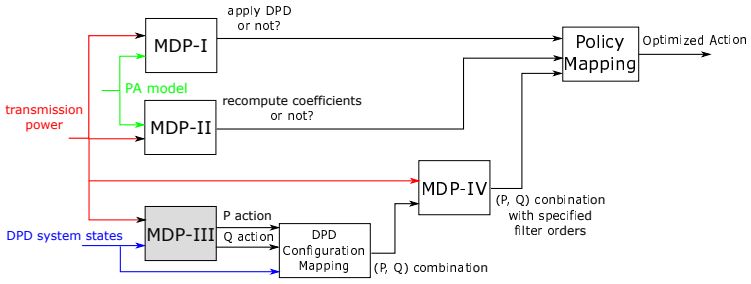


Figure 10: Hierarchical MDP scheme for DPD system.

## 6.5 Hierarchical MDP Models for DPD System Optimization

In this work, we apply the hierarchical MDP approach introduced in Section 4.1 to the dynamic management of the DPD system. The proposed hierarchical MDP scheme is illustrated in Figure 10. This scheme decomposes analysis into four MDPs that are arranged hierarchically. These four MDPs are labeled (from topmost to bottom-most in the hierarchy) MDP-I, MDP-II, MDP-III and MDP-IV, respectively. MDP-I decides whether to apply DPD or not based on the current PA model and transmission power. If DPD is to be applied according to the decision from MDP-I, then MDP-II determines whether to turn on the learning phase of the DPD. When enabled, this learning phase recomputes the filter coefficients based on the PA model and transmission power.

MDP-III and MDP-IV together optimize the DPD system configuration based on the current system state and transmission power. Specifically, MDP-III is responsible for determining the values of $P$ and $Q$, and MDP-IV is responsible for determining the filter order of each employed filtering branch.

As an example, the core MDP components of MDP-III are defined as follows.

- System state space: $\{P, Q\}$, where $P \in \{1, 2, 3, 4\}$ and $Q \in \{1, 2, 3, 4\}$.
- Environment state space: $\{\rho\}$, where $\rho \in \{17\text{dBm}, 19\text{dBm}, 21\text{dBm}\}$ represents the transmission power.
- Action space: $\{P_{\text{action}}, Q_{\text{action}}\}$. The actions on both $P$ and $Q$ belong to the set $\{\text{no change}, \text{reduce by } 1, \text{increase by } 1\}$.
- Reward function: $\text{Reward}(\text{state}, \text{action}) = C_1 \times \text{ACPR}(\text{state}) - C_2 \times \text{EVM}(\text{state}) - C_3 \times \text{Power}(\text{state}) - C_4 \times \text{cost}(\text{action})$, where ACPR, EVM and power are measurements with respect to the specified state; $C_1, C_2, C_3, C_4$ are constants that represent weighting factors for the different metrics considered; and cost represents the system cost incurred by taking the specified action.
- Constraints: The generated policy is subject to ACPR, EVM and power consumption constraints that are imposed on the system at run-time. Furthermore, the selected action should not lead to branch count configurations that violate the pre-specified $P, Q$ ranges.

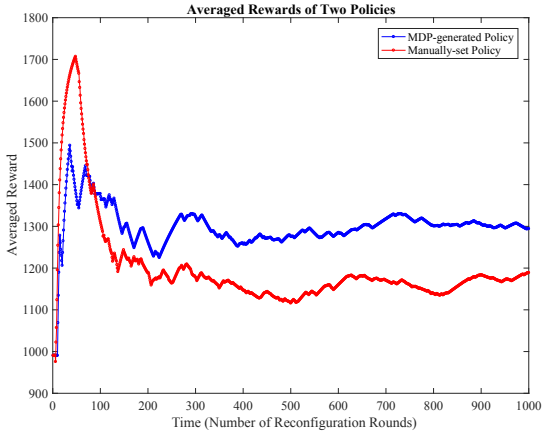


Figure 11: Comparison among MDP-generated policies and manually-set policies.

### 6.6 Simulation Results

We run simulations in MATLAB to measure ACPR and EVM related to each $P, Q$ combination in the system state space. The modulation employed to derive these measurements is QPSK. The results from these measurements are provided as input to the MDP models, and used to generate control policies.

In Figure 11, we compare the policy generated from the proposed hierarchical MDP approach with a manually-set policy that always selects the action that maximizes the current reward without considering stochastic characteristics. As we can see, the manually-set policy may lead to more reward than the MDP-generated policy in the short term (i.e., when time $< 100$ reconfiguration rounds in this case). However, the MDP-generated policy outperforms the manually-set policy over longer time periods.

## 7 Conclusions

In this work, we have developed the Hierarchical MDP framework for Compact System-level Modeling (HMCSM) and its application to design and implementation of adaptive embedded signal processing systems. HMCSM provides a structured design methodology that integrates model-based design for embedded signal processing in terms of dataflow methods; MDP formulation using compact, hierarchical models; optimal policy generation from these models at design time; and dynamic, system-level reconfiguration at run time. The framework enables systematic derivation of system-level reconfiguration policies that are based on application-specific functional requirements and operational constraints. The effectiveness of our proposed MDP-based system design framework was demonstrated through an adaptive wireless communication application: a digital predistortion (DPD) system, which benefits from an implementation that can dynamically generate optimal control policies. Additionally, the LIDE-V library has been updated to support power-efficient hardware implementation of

signal processing systems. These techniques for power-efficient implementation are relevant to hardware acceleration of the novel DPD system that has been developed in this work.

# References


[1] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, eds., *Handbook of Signal Processing Systems*. Springer, second ed., 2013.

[2] L. Li, A. Ghazi, J. Boutellier, L. Anttila, M. Valkama, and S. S. Bhattacharyya, "Evolutionary multiobjective optimization for digital predistortion architectures," in *Proceedings of the International Conference on Cognitive Radio Oriented Wireless Networks*, (Grenoble, France), pp. 498–510, May 2016.

[3] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1994.

[4] S. Chen, Y. Wang, and M. Pedram, "A semi-Markovian decision process based control method for offloading tasks from mobile devices to the cloud," in *Proceedings of the IEEE Global Telecommunications Conference*, pp. 2885–2890, 2013.

[5] A. Munir and A. Gordon-Ross, "An MDP-based application oriented optimal policy for wireless sensor networks," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pp. 183–192, 2009.

[6] C. Boutilier, R. Dearden, and M. Goldszmidt, "Exploiting structure in policy construction," in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1104–1111, 1995.

[7] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 742–760, June 1999.

[8] A. Jonsson and A. Barto, "Causal graph based decomposition of factored MDPs," *Journal of Machine Learning Research*, vol. 7, pp. 2259–2301, 2006.

[9] A. Sapio, M. Wolf, and S. S. Bhattacharyya, "Compact modeling and management of reconfiguration in digital channelizer implementation," in *Proceedings of the IEEE Global Conference on Signal and Information Processing*, (Washington, D.C.), pp. 595–599, December 2016.

[10] L. Li, A. Sapio, J. Wu, Y. Liu, K. Lee, M. Wolf, and S. S. Bhattacharyya, "Design and implementation of adaptive signal processing systems using Markov decision processes," in *Proceedings of the International Conference on Application Specific Systems, Architectures, and Processors*, (Seattle, Washington), pp. 170–175, July 2017.

[11] O. Sigaud and O. Buffet, eds., *Markov Decision Processes in Artificial Intelligence*. Wiley, 2010.

[12] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, pp. 773–799, May 1995.

[13] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, pp. 397–408, February 1996.

[14] B. D. Theelen, M. C. W. Geilen, T. Basten, J. P. M. Voeten, S. V. Gheorghita, and S. Stuijk, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *Proceedings of the International Conference on Formal Methods and Models for Codesign*, July 2006.

[15] E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow," *Proceedings of the IEEE*, vol. 75, pp. 1235–1245, September 1987.

[16] B. Bhattacharya and S. S. Bhattacharyya, "Parameterized dataflow modeling for DSP systems," *IEEE Transactions on Signal Processing*, vol. 49, pp. 2408–2421, October 2001.

[17] B. Bhattacharya and S. S. Bhattacharyya, "Quasi-static scheduling of reconfigurable dataflow graphs for DSP systems," in *Proceedings of the International Workshop on Rapid System Prototyping*, (Paris, France), pp. 84–89, June 2000.

[18] V. Bebelis, P. Fradet, A. Girault, and B. Lavigueur, "BPDF: A statically analyzable dataflow model with integer and Boolean parameters," in *Proceedings of the International Workshop on Embedded Software*, pp. 1–10, 2013.

[19] K. Desnos, M. Pelcat, J. Nezan, S. S. Bhattacharyya, and S. Aridhi, "PiMM: Parameterized and interfaced dataflow meta-model for MPSoCs runtime reconfiguration," in *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pp. 41–48, 2013.

[20] C. Shen, W. Plishker, H. Wu, and S. S. Bhattacharyya, "A lightweight dataflow approach for design and implementation of SDR systems," in *Proceedings of the Wireless Innovation Conference and Product Exposition*, (Washington DC, USA), pp. 640–645, November 2010.

[21] C. Shen, L. Wang, I. Cho, S. Kim, S. Won, W. Plishker, and S. S. Bhattacharyya, "The DSPCAD lightweight dataflow environment: Introduction to LIDE version 0.1," Tech. Rep. UMIACS-TR-2011-17, Institute for Advanced Computer Studies, University of Maryland at College Park, 2011. http://hdl.handle.net/1903/12147.

[22] S. Lin, Y. Liu, K. Lee, L. Li, W. Plishker, and S. S. Bhattacharyya, "The DSPCAD framework for modeling and synthesis of signal processing systems," in *Handbook of Hardware/Software Codesign* (S. Ha and J. Teich, eds.), pp. 1–35, Springer, 2017.

[23] P. L. Fackler, "MDPSOLVE a MATLAB toolbox for solving Markov decision problems with dynamic programming — user's guide," tech. rep., North Carolina State University, January 2011.

[24] S. J. Darak, A. P. Vinod, R. Mahesh, and E. M.-K. Lai, "A reconfigurable filter bank for uniform and non-uniform channelization in multi-standard wireless communication receivers," in *Proceedings of the International Conference on Telecommunications*, pp. 951–956, 2010.

[25] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[26] A. Ghazi *et al.*, "Low power implementation of digital predistortion filter on a heterogeneous application specific multiprocessor," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, (Florence, Italy), pp. 8391–8395, 2014.

[27] L. Anttila, P. Händel, and M. Valkama, "Joint mitigation of power amplifier and I/Q modulator impairments in broadband direct-conversion transmitters," *IEEE Transactions on Microwave Theory and Techniques*, vol. 58, no. 4, pp. 730–739, 2010.

[28] T. Fanni, L. Li, T. Viitanen, C. Sau, R. Xie, F. Palumbo, L. Raffo, H. Huttunen, J. Takala, and S. S. Bhattacharyya, "Hardware design methodology using lightweight dataflow and its integration with low power techniques," *Journal of Systems Architecture*, vol. 78, pp. 15–29, August 2017.

[29] C. Çiflikli and A. Yapící, "Genetic algorithm optimization of a hybrid analog/digital predistorter for RF power amplifiers," *Analog Integrated Circuits and Signal Processing*, vol. 52, no. 1, pp. 25–30, 2007.

[30] L. Ding *et al.*, "Compensation of frequency-dependent gain/phase imbalance in predistortion linearization systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 1, pp. 390–397, 2008.

[31] M. Nizamuddin, *Predistortion for Nonlinear Power Amplifiers with Memory*. PhD thesis, Virginia Polytechnic Institute and State University, December 2002.

[32] J. A. Sills and R. Sperlich, "Adaptive power amplifier linearization by digital pre-distortion using genetic algorithms," in *Proceedings of the Radio and Wireless Conference*, pp. 229–232, 2002.