Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

A Case for Symbolic Limited Optimal Discrete Control: Energy Management in Reactive Data-flow Circuits ⋆

Mete Özbaltan * **Nicolas Berthier** *

* *Department of Computer Science, University of Liverpool, UK*

Abstract: We present a framework for achieving efficient dynamic management of configurable reactive data-flow circuits subject to global design objectives such as mutual exclusion on shared resources and minimization of energy consumption. We propose a new symbolic controller synthesis algorithm that targets the optimization of a cost function summed over a sliding window of a given number of reactions of the system. We then present a technique for constructing symbolic models of configurable data-flow circuits that lends itself to the automatic computation of dynamic configuration controllers. We use these models to experimentally evaluate our control algorithm, and make the case for symbolic optimal discrete controller synthesis on such designs.

Keywords: Symbolic Optimal Discrete Controller Synthesis, Digital Synchronous Circuits, Symbolic Controller Implementation

1. INTRODUCTION

High-level models are often required to reason on synchronous circuit designs, and apply scalable techniques to translate them into equivalent designs that meet various performance goals such as energy-efficiency [Zhang et al., 2006]. Among these models, the family of *data-flow process networks* [Lee and Parks, 1995] sees sub-circuits as actors that communicate tokens through communication channels (*FIFOs*) and react according to predefined firing rules. *Kahn Process Networks* (KPN) [Kahn, 1974] are a sub-class of such models where channels are considered unbounded, and a process is fired whenever there are enough tokens in its input channels. A consequence of this property is that writes can be considered *non-blocking*—*i.e.,* a process is never blocked when it writes to a channel—, whereas reads to empty channels are *blocking*. KPNs can be used to describe systems where the amount of data produced and consumed by a process is not statically determined.

We consider *reactive data-flow* circuit designs similar to KPNs, where each process offers a *discrete configuration means* that impacts its computation speeds and power consumption. Here, reactivity means that designs have to respond to a potentially infinite stream of new data. In practice, these designs are convenient when several implementations are available to perform a given computational task, each with various levels of resource consumption, quality of service, etc [An et al., 2016]. Process implementations may also make use of shared resources in such a way that *mutual exclusion constraints* must be enforced. Finding the optimal configuration for such a network at any given time is intrinsically a challenging task. We advance a framework for automatically computing and integrating a *dynamic configuration manager* into such designs. This manager is able to satisfy predefined *global design objectives*, such as reducing the overall energy consumption of the design over a period of time, while meeting the mutual exclusion constraints. We rely on the construction of an *abstract symbolic system model* of the design, and employ *discrete control* techniques to compute a piece of code (*e.g.,* a synchronous circuit) that implements *energy-efficient configuration management*. As we observe in the next Section on discrete optimal control, solutions that address the kind of problems above only focus on runs that reach a *given set* of target states. As such, they do not suit the needs imposed by the reactive designs we consider, for which one cannot identify a suitable set of target states. Therefore, we devise in Section 3 a *new algorithm for achieving symbolic optimal control, limited to a sliding window of a fixed number of discrete steps*. We then present in Section 4 our approach for constructing symbolic systems that model configurable reactive data-flow networks that permit the construction of energy-aware dynamic configuration managers. In Section 5, we use the models to: (i) experimentally evaluate our new algorithm; (ii) make the case for applying limited optimal controller synthesis on such designs; (iii) demonstrate the power of such symbolic techniques for producing results that can directly be implemented as control mechanisms in, *e.g.,* hardware circuits.

⋆ This work was supported by the EPSRC through grant EP/M027287/1.

2. CONTROL OF DISCRETE SYMBOLIC SYSTEMS

The symbolic systems we consider are built upon a set of *symbols* $\mathcal{S}$, each associated with a domain of definition $\mathbb{D} \in \mathcal{D}$ according to the mapping $\mathrm{Dom}\colon \mathcal{S} \to \mathcal{D}$. $\mathcal{D}$ comprises at least the Boolean domain $\mathbb{B} \stackrel{\text{def}}{=} \{\mathsf{tt}, \mathsf{ff}\}$, and numerical domains $\mathcal{D}_{\mathsf{num}} \subseteq \mathcal{D}$ such as the Integers ($\mathbb{Z}$) and Rationals ($\mathbb{Q}$), and $\mathcal{S}$ notably comprises all *constants* used to define domains in $\mathcal{D}$ (*e.g.,* $\{\mathsf{tt}, \mathsf{ff}, 0, -\frac{1}{2} \ldots\}$). Given a domain $\mathbb{D} \in \mathcal{D}$, the set $\mathcal{X}_{\mathbb{D}}$ of all $\mathbb{D}$-*valued symbolic expressions* comprises all formulae $\psi_{\mathbb{D}}$ that can be generated according to the following grammar:

$$\begin{aligned}
\psi_{\mathbb{D}} &::= s \mid \mathsf{if}\ \psi_{\mathbb{B}}\ \mathsf{then}\ \psi_{\mathbb{D}}\ \mathsf{else}\ \psi_{\mathbb{D}} && (\mathbb{D} \in \mathcal{D}, \mathrm{Dom}\,(s) = \mathbb{D}) \\
\psi_{\mathbb{B}} &::= \neg\psi_{\mathbb{B}} \mid \psi_{\mathbb{B}} \vee \psi_{\mathbb{B}} \mid \psi_{\mathbb{B}} \wedge \psi_{\mathbb{B}} \mid \psi_{\mathbb{B}} \Rightarrow \psi_{\mathbb{B}} \mid \psi_{\mathbb{D}} = \psi_{\mathbb{D}} \\
&\quad \mid \psi_{\mathcal{N}} \bowtie \psi_{\mathcal{N}} && (\mathbb{D} \in \mathcal{D}, \mathcal{N} \in \mathcal{D}_{\mathsf{num}}, \bowtie \in \{<, \leqslant\}) \\
\psi_{\mathcal{N}} &::= c\psi_{\mathcal{N}} \mid \psi_{\mathcal{N}} \between \psi_{\mathcal{N}} && (\mathcal{N} \in \mathcal{D}_{\mathsf{num}}, c \in \mathcal{N}, \between \in \{+, -\})
\end{aligned}$$

where s denotes any symbol in $\mathcal{S}$, and $\mathcal{D}_{\mathsf{fin}}$ is the set of all finite domains in $\mathcal{D}$. Note that rule $\psi_{\mathbb{D}}$ is polymorphic (*i.e.,*

generic) in the domain $\mathbb{D}$, and restricts conditional constructs to cases where the two rightmost expressions are of the same type. Also, this grammar features the following constructs that are available as syntactic sugar to ease readability: $\neg$, $\vee$, $\wedge$ and $\Rightarrow$ are usual logical connectives that can be expressed using conditional constructs, and $\varphi - \psi$ is equivalent to $\varphi + -1\psi$. Informally, the rules $\psi_{\mathcal{N}}$ and $\psi_{\mathbb{B}}$ respectively produce *guarded linear arithmetic expressions* and *propositional predicates with equality and linear (in)equalities*.

Non-constant symbols are *variables*, and a *valuation* $\mu \in \mathrm{Val}(V)$ for each variable in $V \subseteq \mathcal{S}$ maps every variable x from V to $\mathrm{Dom}(x)$. We compose valuations for disjoint sets of variables with $\uplus$. The *support* of e, denoted $\mathrm{Support}(e)$, is the set of variables that appear anywhere in its constructs. We shall index sets of expressions with the variables that make up the smallest super-set of their support when such a precision appears relevant for our exposition: *e.g.,* $\mathcal{X}_{\mathbb{D},V} \stackrel{\text{def}}{=} \{e \in \mathcal{X}_{\mathbb{D}} \mid \mathrm{Support}(e) \subseteq V\}$; when domains appear irrelevant, however, we abbreviate the set of all symbolic expressions as $\mathcal{X} \stackrel{\text{def}}{=} \bigcup_{\mathbb{D}\in\mathcal{D}} \mathcal{X}_{\mathbb{D}}$. Given $e \in \mathcal{X}_{\mathbb{D},V}$ and a valuation $\mu \in \mathrm{Val}(V)$, the *interpretation of* e *w.r.t.* μ, noted $[\![e]\!]\mu$, belongs to $\mathbb{D}$ and can be computed according to the usual semantics of the operators. $\mathcal{P} \stackrel{\text{def}}{=} \mathcal{X}_{\mathbb{B}}$ is the set of all *propositional predicates* with (in-)equalities; $\mathcal{P}$ is closed under elimination of variables defined on finite domains. Further, for every domain $\mathbb{D} \in \mathcal{D}$, $\mathcal{X}_{\mathbb{D}}$ is also closed under substitution of any symbolic expression that belongs to $\mathcal{X}_{\mathbb{D}'}$ for any variable v *s.t* $\mathrm{Dom}(v) = \mathbb{D}'$. We generalize to multiple variables and denote with $e\big[\mu\big]$ such a substitution in $e \in \mathcal{X}$ according to a mapping μ from a set of variables to symbolic expressions of the same domain of definition. We use the variable assignment denotation $x := e_x$ to incrementally construct such a mapping in Section 4 [1] .

Symbolic systems are made of disjoint sets of *state* and *input* variables, respectively denoted X and I. The values associated to state variables are fixed initially, and evolve according to a discrete step (*lock-step*) semantics very similar to that of synchronous circuits: *discrete evolutions* are defined using a mapping T with one *assignment* $x := e_x$ per state variable $x \in X$, where e_x is a $\mathrm{Dom}(x)$-valued symbolic expression that determines the value to be memorized by variable x based on the current valuations for state and input variables, at each *tick* of an *implicit basic clock*. Such a system induces a state machine whose *state-space* consists of all possible valuations for every variable in X, and whose transitions are encoded by the discrete evolutions T. If the domain of definition of every variable in X is finite, then this constitutes a *finite-state machine* (FSM).

2.1 Symbolic Control

Solving *symbolic control problems* on such systems can be seen as solving a game where, at each tick, one player (the environment) gives a value for a fixed portion of the input variables, *then* the other player (the *controller*) assigns values to every other input variable, *and then* the game evolves into a subsequent state according to the discrete evolutions and the inputs given by the players. The input variables assigned by the first (resp. second) player are said *non-controllable* (resp. *controllable*); we denote these sets U and C, respectively. *Control objectives* expressed as logic formulas that involve state and/or input variables are assigned to the controller, and the solution of the control problem consists in a *strategy* that this player can follow to win the game by fulfilling all its objectives. Systems where such a choice for the first player always exists are said *deadlock-free*, and algorithms solving control problems should preserve (or enforce) this property.

The control objectives that we use in this work are twofold: First, achieving a *safety control objective* consists in enforcing a *safety property*. Such properties can be expressed using some temporal logic like LTL [Clarke et al., 1986]. In our case however, we use the same symbolic constructs as for the system to build *stateful observers* that represent the temporal aspects of the properties we need (*e.g.,* sequence, iteration), and can therefore restrict the safety objective formulas to propositional logic. Second, satisfying *optimal control objectives* consists in minimizing a *cost function*, possibly summed over a sliding window of a given number of ticks. In systems as described above, when costs are associated with the *transitions* of the underlying FSM instead of its states, the cost function is a total mapping from state and input valuations into some numerical domain: in symbolic terms, it can therefore be expressed as an expression in $\mathcal{X}_{\mathcal{N},X\cup I}$, with $\mathcal{N} \in \mathcal{D}_{\mathsf{num}}$. When both a safety and an optimal control objective are to be enforced, the combined strategy can be obtained by first computing a strategy that ensures the safety objective, and then refining it to satisfy the optimal control objective.

Strategies as Efficient Sequential Code Usually, strategies that are computed by algorithms dedicated to operate on symbolic systems eventually take the form of a *predicate* over state and input variables: *i.e.,* they belong to $\mathcal{P}_{X\cup I}$. Then, given a valuation for state and non-controllable inputs, a constraint solver needs to be used to find a suitable valuation for controllable inputs that satisfies the predicate. The existence of such a solution is guaranteed by the control algorithm; when this solution is always unique, moreover, the strategy is *deterministic*.

To avoid relying on a constraint solver, a *triangularization* procedure [Hietter et al., 2008] can be used to translate the strategy into a mapping from valuations for state and non-controllable input variables into valuations for controllable input variables, which is basically a combinatorial circuit. This translation operates via successive variable substitutions and partial evaluations of the predicate strategy. Triangularizing non-deterministic strategies requires a total order on solutions, *e.g.,* with a total order on both the controllable input variables and their respective domains of definition: this can be achieved by ordering (prioritizing) the controllable input variables, and assigning them with "default" or "preferred" values. The triangularized strategy can directly be combined with the original system to form a controlled system that, when fed with values for non-controllable inputs, keeps track of the system state and outputs appropriate values for controllable variables to enforce the desired control objectives. This controlled system does not rely on any constraint solver, and can therefore easily be implemented as an efficient piece of sequential code or synchronous circuit.

2.2 Tooling and Related Works on Discrete Optimal Control

Few works have addressed the problems of enforcing safety control and optimization objectives on the kind of symbolic systems we consider; they mostly derive from the seminal work

[1] Throughout the paper, and unlike $:=$, $s \stackrel{\Delta}{=} e$ denotes the classical formal definition of a left-hand side symbol s with a right-hand side expression e: s can be seen as a placeholder for expression e everywhere in the paper. Alternatively, we use $\stackrel{\text{def}}{=}$ when defining structures and algorithms.

of Ramadge and Wonham [1989]. Marchand et al. [2000] and Miremadi et al. [2012] implemented tools that are suitable for enforcing safety objectives. Berthier and Marchand [2014, 2015] extended these algorithms to operate on systems where state variables may take their values in infinite numerical domains like Integers or Rationals, and implemented their solutions as part of the ReaX tool [2]. In turn Marchand and Le Borgne [1998], and Dumitrescu et al. [2010], implemented algorithmic solutions for optimization objectives; these works focus on finite-state systems equipped with cost functions relating to states only, and essentially consist in symbolic adaptations of Bellman's algorithm for the computation of optimal strategies using dynamic programming [Bellman, 1957]: as such, they operate based on a given set of *target states* in the underlying FSM, to be reached using a path that incurs an optimal cost.

The algorithm we present in the next Section alleviates the need for specifying target states to better accommodate the modeling of reactive systems for which the solutions above are inadequate. Instead, we take an alternative approach by operating on *every path of a given length at once*.

3. AN ALGORITHM FOR SYMBOLIC LIMITED OPTIMAL CONTROLLER SYNTHESIS

We now present our new algorithm for refining a base strategy σ towards fulfilling a given optimization objective. We consider the objective that seeks the minimization of a cost function $\zeta \in \mathcal{X}_{\mathcal{N},X\cup I}$ summed over $k \in \mathbb{N}^+$ ticks, with $\mathcal{N} \in \mathcal{D}_{\mathsf{num}}$. Observe that ζ actually associates a cost on *every transition* of the model: given a state $q \in \mathrm{Val}(X)$ and a valuation $\iota \in \mathrm{Val}(I)$ for inputs, $[\![\zeta]\!] q \uplus \iota$ gives the cost incurred by the current tick as a quantity in $\mathcal{N}$. In turn, the base strategy σ is given as a predicate on state and input variables, *i.e.,* $\sigma \in \mathcal{P}_{X\cup I}$.

3.1 Computing Best Expected Outcomes

Our algorithm starts by computing the *best outcome* η_k as a symbolic expression on the numerical domain $\mathcal{N}^* \stackrel{\text{def}}{=} \mathcal{N} \cup \{\infty\}$, *i.e.,* $\mathcal{N}$ extended with ∞ so that ∞ is the supremum element for $\mathcal{N}^*$, and $\infty < \infty$ does not hold [3]. $\eta_k \in \mathcal{X}_{\mathcal{N}^*,X\cup I}$ gives the best outcome that any strategy refined from σ can achieve on a time window of k ticks, given any valuation for the state and input variables. η_k can be recursively computed as

$$\eta_1 \stackrel{\text{def}}{=} \mathsf{if}\ \sigma\ \mathsf{then}\ \zeta\ \mathsf{else}\ \infty \tag{1}$$

$$\eta_{i+1} \stackrel{\text{def}}{=} \mathsf{if}\ \sigma\ \mathsf{then}\ \big(\mathrm{Max}_U \circ \mathrm{Min}_C(\eta_i)\big)\big[T\big] + \zeta\ \mathsf{else}\ \infty \tag{2}$$

where

$$\mathrm{Min}_V(s), s \in \mathcal{S} \stackrel{\text{def}}{=} s \tag{3}$$

$$\mathrm{Min}_V(ce), c \in \mathcal{N} \stackrel{\text{def}}{=} \begin{cases} c\,\mathrm{Min}_V(e) & \text{if } c \geqslant 0 \\ c\,\mathrm{Max}_V(e) & \text{if } c < 0 \end{cases} \tag{4}$$

$$\mathrm{Min}_V(e_1 + e_2) \stackrel{\text{def}}{=} \mathrm{Min}_V(e_1) + \mathrm{Min}_V(e_2) \tag{5}$$

$$\mathrm{Min}_V(\mathsf{if}\ p\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) \stackrel{\text{def}}{=} \mathsf{if}\ \exists_V p \wedge \exists_V \neg p\ \mathsf{then}\ \min\big(\mathrm{Min}_V(e_1), \mathrm{Min}_V(e_2)\big)\ \mathsf{else\ if}\ \exists_V p\ \mathsf{then}\ \mathrm{Min}_V(e_1)\ \mathsf{else}\ \mathrm{Min}_V(e_2) \tag{6}$$

with $\min\big(e_1, e_2\big) \stackrel{\text{def}}{=} \mathsf{if}\ e_1 < e_2\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ (and similarly for Max_V, max). $\exists_V p$ denotes the existential elimination of every *finite* variable in V from a predicate p. Eqs (3)-(6) describe a solution to the *symbolic optimization* problem that consists in finding, given a set of variables V and a numerical expression $e \in \mathcal{X}_{\mathcal{N},W}$, a numerical expression e' *without any variable from* V (*i.e.,* $e' \in \mathcal{X}_{\mathcal{N},W\setminus V}$) such that: (i) e evaluated according to any possible valuation for all variables in V is always greater or equal than e'; and (ii) there exists a valuation for V such that e equals e'; *i.e.,* $\forall \omega \in \mathrm{Val}(\mathrm{Support}(e) \setminus V)$,

(i) $\forall \nu \in \mathrm{Val}(V), [\![e]\!]\omega \uplus \nu \geqslant [\![e']\!]\omega$; and
(ii) $\exists \nu \in \mathrm{Val}(V), [\![e]\!]\omega \uplus \nu = [\![e']\!]\omega$.

Eq. (3) is straightforward since V must not contain infinite variables and $s \in \mathcal{X}_{\mathcal{N}}$: therefore $s \notin V$. Eqs (4)-(6) operate recursively on the structure of the expression e. Elimination in conditional constructs essentially involves building a new expression that separately handles three sub-cases based on whether there exist valuations for V that maintain satisfaction of the condition p or not—the fourth case, that does not appear in Eq. (6), equates to p unsatisfiable.

Going back to Eqs (1)-(2) for the computation of η_k, the base case for $k = 1$ consists in associating every transition that does not satisfy σ with the supremum cost ∞. In turn, the computation of η_{i+1} given η_i in Eq. (2) can be broken down as follows:

(Min_C) solve the symbolic optimization problem of finding the minimum for every controllable variable (C): this actually represents the choice of values for controllable input variables that best fulfill the objective;

(Max_U) solve the dual symbolic optimization problem for every non-controllable variable (U), representing the worst possible move of the environment against the desired objective. This leads to a numerical expression that only involves state variables since the combined optimization problems above eliminate all input variables;

($\cdot\big[X \mapsto T\big]$) substitute every state variable with its corresponding evolution expression (note that this may re-introduce input variables);

($\cdot + \zeta$) add ζ to account for the cost of the additional transition. The result of this operation builds a new cost function that associates any transition in the underlying state-machine with the best expected outcome that can be achieved using any choice for controllable variables against the worst choices for non-controllable variables on a *subsequent* path of length i;

($\mathsf{if}\ \sigma\ \mathsf{then}\ \cdot\ \mathsf{else}\ \infty$) lastly, associate any choice for inputs that does not satisfy the strategy σ with the supremum cost.

The result $\eta_k \in \mathcal{X}_{\mathcal{N}^*,X\cup I}$ represents the best expected outcome towards the optimization objective among *every path of length* k that originates from *any state*. This is to be contrasted with the existing algorithms mentioned in Section 2.2, that compute the best *cost* for reaching any *target state*.

3.2 Computing the Refined Strategy

Given η_k, a strategy that fulfills the objective consists in choosing values for variables in C that minimize η_k *at the current tick*, given valuations for state and non-controllable input variables. We compute the predicate that encodes this strategy as

$$\sigma' \stackrel{\text{def}}{=} \big(\nexists_{C'}(\eta_k\big[C \mapsto C'\big] < \eta_k)\big) \wedge \sigma \tag{7}$$

where C' are primed versions of all variables in C. The innermost parenthesized expression in Eq. (7) denotes the condition upon which choices for variables in C' are expected to produce a strictly better outcome according to η_k, than choices for variables in C. The resulting strategy σ' thus consists in keeping only choices for C such that no strictly better choice for variables in C' exists, and ensuring that these choices are indeed compatible with the strategy to be refined σ.

[2] Available at https://reatk.gforge.inria.fr/.
[3] In the case of a maximization, $\mathcal{N}$ is extended with $-\infty$ instead.

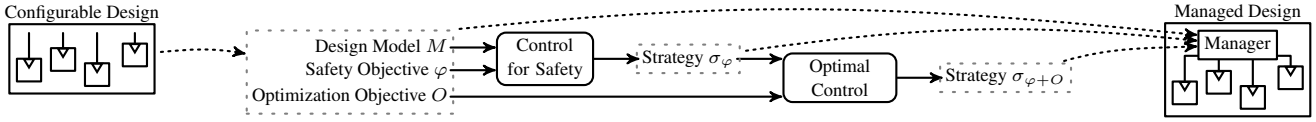

Fig. 1. Overview of our suggested work-flows for computing energy-aware configuration managers.

4. ENERGY-AWARE RECONFIGURATION

We consider *controllable designs* made of *processes* that communicate through *channels*. Each process consumes and processes data from one or more input channels whenever possible—*i.e.,* a process does not wait unless one of its input channels is empty—, and produces results into at least one output channel. *Upon consumption* of new pieces of data, each process reads a *configuration signal* that influences the speed (*e.g.,* the number of clock cycles in a hardware circuit) and power consumption its takes it to finish processing the data and produce an output. Lastly, the design may feature a set of shared resources that processes may make use of depending on their configuration (*e.g.,* a specialized signal processing unit in an FPGA): choices for process configurations must therefore obey a set of *mutual exclusion constraints*. The design receives pieces of data from its environment through one or more input FIFO channels.

Given such a design, our goal is to automatically construct an *energy-aware configuration manager* whose role is to *dynamically* select configurations for each process according to various *global design objectives*.

4.1 Overview of the Approach for Computing Managers

We describe in Fig. 1 the work-flows offered by our approach. A symbolic system model M is first constructed from the configurable design. M is made of the synchronous parallel composition of: (i) a model for each channel, defined using state and non-controllable variables, and appropriate encoding of discrete evolutions; (ii) a similar model for each process, that additionally involves controllable variables that offer levers for the sought-after manager to select configurations. We associate the symbolic system model with a series of definitions based on process and channel models, that permit the expression of control objectives (see below).

Then, a *safety objective* φ is built, in the form of a conjunction of propositional formulas that involve the state variables of process models. Each one of these formulas expresses a mutual exclusion constraint between process configurations that make use of a shared resource. At this stage, a symbolic safety control algorithm can be used to compute a strategy σ_φ. This strategy is guaranteed to select values for the process configuration inputs (*i.e.,* controllable input variables of M) that ensure that the safety objective φ is satisfied. σ_φ and M can be used in combination to form a manager that outputs configuration choices for each process; a design whose processes are configured according to the outputs of this manager cannot violate any of the aforementioned mutual exclusion constraints. Alternatively, the strategy σ_φ can be improved by using a symbolic optimal control algorithm, such as the one that has been presented in Section 3, that ensures an additional optimization objective O. In our case, the cost function that we use to define O basically consists of an estimation of the energy consumption for all processes. The resulting refined strategy $\sigma_{\varphi+O}$ can be used in the same way as σ_φ to dynamically select configurations for each process. We further detail the construction of the symbolic system model M and associated definition of control objectives to obtain energy-aware designs in the remainder of this Section.

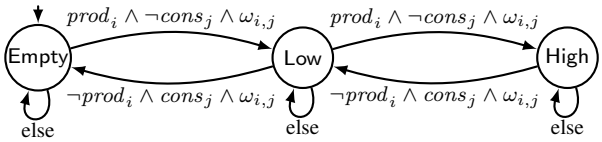

Fig. 2. Representation of 3-state channel model encoded as $q_{i,j}$.

4.2 Abstract Channel Model

Several options are available when modeling the kind of channels featured in the configurable models we consider. The choice for one option or the other notably depends on whether the manager that we seek to obtain needs to precisely track the level of occupation of the FIFOs or not, or, if FIFOs were bounded for instance, if it actually needs to ensure the absence of overflows. In our use-case the configuration manager can only alter the speed of processes by selecting appropriate configurations, and preventing FIFO overflows would therefore require more insight (such as computation rates) about the processes at hand. We can however suppose that a manager that is able to distinguish *almost empty* FIFOs will be able to leverage this additional knowledge of *future process activities* to achieve a more energy-efficient *planning for process configurations*. In order to permit such a distinction while limiting the growth of the state-space, we *abstract* the state of a channel from process i to process j using a state variable $q_{i,j} \in X$ that takes its values in $\{\mathsf{Empty}, \mathsf{Low}, \mathsf{High}\} \in \mathcal{D}_{\mathsf{fin}}$. We further introduce *non-determinism* in each channel model $q_{i,j}$ with the help of a Boolean *oracle* input $\omega_{i,j} \in U$. This oracle is a non-controllable input used to non-deterministically transition between the abstract states of the channel.

We illustrate in Fig. 2 the abstract behavior of a FIFO using a Mealy-machine partitioned according to the domain of $q_{i,j}$; this automaton is strictly equivalent to the assignment that we use to define the discrete evolutions of every $q_{i,j}$ in the symbolic model M. $prod_i$ and $cons_j$ denote predicates that hold whenever production and consumption of elements occurs in the channel: we define the associated symbolic expressions in the next Section as part of the model of processes. Observe that, for instance, this automaton clearly shows that the shortest path from High to Empty involves two ticks where $cons_j$ holds and $prod_i$ does not.

4.3 Abstract Process Model

We now turn to the symbolic model of a process p, defined as:

$$
\begin{aligned}
cons_p &\triangleq \bigwedge_{i \in Q_p} (q_{i,p} \neq \mathsf{Empty}) \wedge \mathsf{if}\ p_p = \mathsf{Idle}\ \mathsf{then}\ \mathsf{tt}\ \mathsf{else}\ e_p \\
prod_p &\triangleq p_p \neq \mathsf{Idle} \wedge e_p \\
p'_p &\triangleq \mathsf{if}\ cons_p\ \mathsf{then}\ \mathsf{Act}\text{-}cfg_p\ \mathsf{else} \\
&\qquad \mathsf{if}\ \neg cons_p \wedge prod_p\ \mathsf{then}\ \mathsf{Idle}\ \mathsf{else}\ p_p \\
p_p &:= p'_p
\end{aligned}
\tag{8}
$$

We further represent the variables and definitions above in context in Fig. 3 where we illustrate a simple configurable design. Input r denotes the arrival of a new piece of data into the network. A process p becomes or remains active whenever a data token is available in all its input channels (a set that we note here with Q_p): this is represented in the model using a predicate $cons_p$. The model of a process p also features a predicate $prod_p$, which indicates that the active process p has terminated its computations and produced a new piece of data into (each one of) its output channel(s). Since the model does not track any internal operational process state, and in particular none of its progress towards terminating its computations, an input variable e_p is used to drive process terminations. Upon termination of its current computations (this event manifests in the system model as a tick where e_p holds), p either becomes idle if any one of its input channel is empty, or consumes a new element from all of them and remains active. Eq (8) defining expression p'_p represents the value held by state variable p_p *starting from the next tick*; the new abstract state of p's model that this expression represents is either Idle, or some active state that is tagged with the configuration that was selected by the controller as controllable variable cfg_p when p last consumed data. The state variable $p_p \in X$ takes its values in the domain $\mathrm{Modes}_p \stackrel{\text{def}}{=} \{\mathsf{Idle}, \mathsf{Act}\text{-}1, \ldots, \mathsf{Act}\text{-}|\mathrm{Configs}_p|\} \in \mathcal{D}_{\mathsf{fin}}$ where $|\mathrm{Configs}_p| \in \mathbb{N}^+$ is the number of available configuration options (*aka modes*) for p; in turn, $cfg_p \in C$ is defined in the domain $\{1, \ldots, |\mathrm{Configs}_p|\} \in \mathcal{D}_{\mathsf{fin}}$.

Wort-case Energy Estimation In our modeling approach, the designer associates each configuration c for a process p with a worst-case execution-time $wcet_{p,c} \in \mathcal{N}$ and peak-power consumption $pp_{p,c} \in \mathcal{N}$, using a numerical domain $\mathcal{N} \in \mathcal{D}_{\mathsf{num}}$. An over-approximation of the energy consumption incurred by any configuration choice for a process p that consumes some data during a tick (and thus starts a new computation cycle) is therefore

$$we_p \stackrel{\triangle}{=} \sum_{c \in \mathrm{Modes}_p} \mathsf{if}\ cons_p \wedge p'_p = c\ \mathsf{then}\ wcet_{p,c} \times pp_{p,c}\ \mathsf{else}\ 0,$$

and the global energy consumption $we \in \mathcal{X}_{\mathcal{N}, X \cup I}$ is the sum over all processes P: $we \stackrel{\triangle}{=} \sum_{p \in P} we_p$. Observe that $(wcet_{p,c} \times pp_{p,c})$ terms are constants that can be evaluated during the construction of the symbolic model. Also, we involves expressions for p'_p as defined in Eq (8), and it thus associates energy costs to *transitions* of the system.

4.4 Control Objectives

Our control objectives fall into two categories: *safety objective* φ and *optimization objective* O. In our use-case, the former category embodies mutual exclusion constraints between process configurations that share resources. For instance, a safety property given as predicate $\varphi \stackrel{\triangle}{=} \neg(p_p = \mathsf{Act}\text{-}i \wedge p_q = \mathsf{Act}\text{-}j)$ indicates that processes p and q cannot be active in their respective ith and jth modes at the same time. At last, one can make use of the worst-case energy estimation expression we defined above to specify an optimization objective O to be enforced using the symbolic optimal control algorithm we presented in Section 3.

We have defined at this point a full system model and associated control objectives. We can thus employ a work-flow as sketched in Fig. 1 to obtain a control strategy σ, and eventually obtain

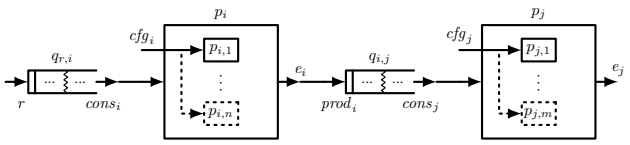

Fig. 3. Graphical representation of a 2-process 2-channel configurable design; the $p_{i,1}, \ldots, p_{i,n}$'s represent the distinct modes of process i—similarly for j. r, cfg_i, cfg_j, e_i, and e_j are inputs of the system model, whereas $cons_i$, $prod_i$ and $cons_j$ denote expressions (predicates).

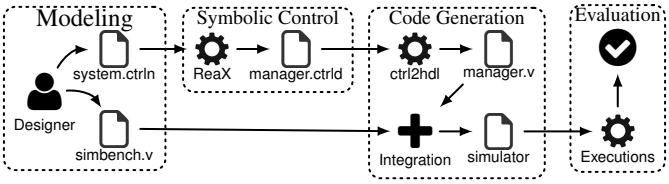

Fig. 4. Synthesis & simulation tool-chain for assessing global design objectives.

an implementable configuration manager by means of the triangularization process described at the end of Section 2.

5. EXPERIMENTAL EVALUATIONS

Let us now turn to the experimental evaluations of our contributions. Apart from carrying out some performance evaluations of our implementation in ReaX of the new optimal control algorithm of Section 3, we also want to empirically assess that it is actually able to enforce optimization goals by using the models constructed above. Indeed, such actual performance evaluations are necessary in the case of symbolic implementations, where one usually observes significant gaps between the practical performances and theoretical complexity results. We also want to experimentally assess whether configuration managers as produced using our approach effectively improve energy consumption whatever the sequences of inputs the resulting designs are subject to. This essentially means that the energy-efficiency of a manager needs to be evaluated on as many (realistic) scenarios as possible. At last, we want to observe the impact of the size of the sliding window used to specify the limited optimal control objectives.

5.1 Constructing Simulators of Managed Designs

We represent in Fig. 4 an overview of the series of modeling steps and tools that we used to carry out our evaluations. The "Modeling" box on the left-hand side depicts the designer's manual tasks of both constructing the symbolic system model and control objectives (in system.ctrln), as well as a simulation bench in the form of a Verilog file simbench.v. The latter will be used to carry out multiple stochastic simulations of the resulting managed design, encoded as a sequential circuit (we give more details on the simulation bench in Section 5.2 below).

In the "Symbolic Control" phase, we use ReaX to perform safety and limited optimal control on the system, triangularize the resulting strategy, and combine it with the original system to construct the manager. The latter is a controlled system (in file manager.ctrld), that can directly be translated into a sequential circuit using ReaX's companion tool ctrl2hdl: this translator turns deterministic symbolic systems into equivalent code in a hardware description language such as Verilog: this step produces manager.v. The integration of this manager with

```
module fifo (input clk, input c, input p, output ω);
   reg [31:0] fifo;
   wire empty = 0, low_u = 10;
   assign ω = (p  & ~c & (fifo==empty   | fifo==low_u)) |
              (~p &  c & (fifo==empty+1 | fifo==low_u+1));
   initial fifo = 0;
   always @(posedge clk) begin
      if (p & ~c) fifo <= fifo + 1;
      if (~p & c) fifo <= fifo - 1;
   end
endmodule
```

Listing 1. Verilog module for FIFO's simulated behaviors.

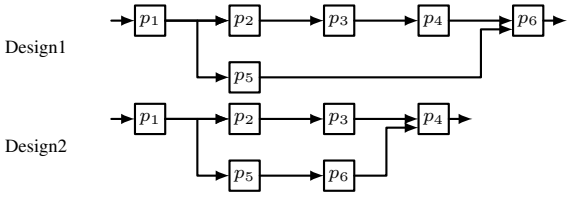

Fig. 5. Schematic representation of configurable designs.

Table 1. Specification values for 6-process example.

Process p	$c \in \text{Modes}_p$	$wcet_{p,c}$ (clock-cycles)	$pp_{p,c}$
p_1	Act-1 / Act-2	100 / 150	55 / 35
p_2	Act-1 / Act-2	40 / 50	70 / 27
p_3	Act-1 / Act-2	40 / 50	43 / 80
p_4	Act-1 / Act-2	40 / 50	43 / 80
p_5	Act-1 / Act-2 / Act-3	180 / 210 / 195	8 / 7 / 7
p_6	Act-1 / Act-2	80 / 90	70 / 60

the simulation bench produces a hardware circuit that can be efficiently executed using compilers and simulators for synchronous circuits.

5.2 Simulation Bench

Several Verilog modules make up the basis for constructing a simulation bench for our designs. A module `process` is instantiated for each process in the design, and simulates computations of dynamically parameterizable length. It accepts a parameter `et`, and basically starts counting upon receiving a *run* signal `r`. It emits `e` when its counter reaches the value of its parameter `et`. The value of `et` for a process will be randomly drawn upon every one of its data consumptions. In turn, module `fifo` (shown in Listing 1) accurately tracks the amount of simulated payload in a FIFO channel, and reports when this occupation level crosses some fixed boundaries using a dedicated output signal ω, which is fed as input to the manager (see module `main` below) to drive the corresponding abstract 3-state behavior illustrated in the Mealy-machine of Fig. 2. In this particular instance, every concrete simulated FIFO that has between 1 and 10 elements is considered in the abstract state Low; the value of 10 is here chosen arbitrarily.

An additional module `main` assembles processes and FIFOs along with the manager produced by ReaX and ctrl2hdl. This module reflects the behavior of the modeled system, and includes the produced manager to form a controlled circuit that integrates dynamic reconfiguration capabilities.

At last, a module `simbench` feeds the simulated design by periodically raising `r`. This module is also responsible for uniformly drawing values for actual execution times and power consumptions of each process, depending on its current configuration and in accord with the specification used to

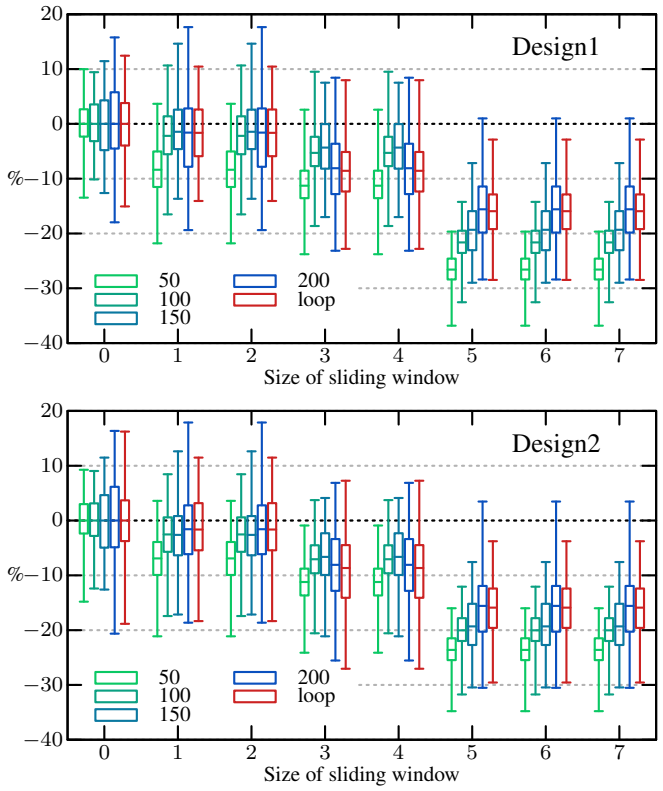

Fig. 6. Gain in simulated energy per completely processed data for each design illustrated in Fig. 5.

define the system model (in system.ctrln in Fig. 4). `simbench` also reports simulated energy consumption data every 1 000 clock cycles. This way of partitioning the resulting stochastic simulation trace into chunks of 1 000 clock cycles allows us to gather statistics on the simulated energy consumption for the design under multiple configurations and loads (*i.e.,* state of the channels) at once.

5.3 Simulation Results

Let us now present the simulation results we obtain for two designs with 6 processes shown in Fig. 5, whose specification values for worst-case execution times and power peaks for each process configuration are given in Table 1; observe that process p_5 offers 3 possible configurations. For both designs, additional mutual exclusion constraints must ensure that process p_3 must not be in mode Act-1 while p_5 is in mode Act-1 (and conversely); similarly for mode Act-1 of p_4 and Act-2 of p_5:

$$\varphi = \neg\,(p_3 = \text{Act-1} \wedge p_5 = \text{Act-1}) \wedge \\ \neg\,(p_4 = \text{Act-1} \wedge p_5 = \text{Act-2})\,.$$

We represent in Fig. 6 the gains in energy consumption per processed data that we observe *w.r.t.* the size of the sliding window used for the limited optimal control algorithm. For each size of sliding window (on the horizontal axis), we give the minimum, average, and maximum gain in percentage, as well as the low and high quartiles (for each case, aggregated over 100 simulations). A window of size 0 indicates that the manager is produced without any enforcement of optimal control objective (yet, mutual exclusion constraints are enforced). All gains (on the vertical axis) are given in percentage of the average energy consumption obtained for the non-optimized design (represented with the dashed horizontal line). For each example design, we

Table 2. Synthesis time and memory footprint of ReaX for example design Design1 *w.r.t.* size of sliding window selected for the limited optimal control algorithm (k); we also report the size of the resulting manager circuit (in number of logic elements—LE).

k	Time (s)	Memory (KB)	Manager size (LE)
0	0.13	50424	660
1	7.49	192940	661
2	37.40	523300	663
3	89.86	1572140	839
4	133.20	2848112	1035
5	164.42	2531132	1161
6	198.01	3030064	1190
7	210.84	3054040	1192

show 5 series of results that each represent the average period (50, 100, 150, or 200, in clock cycles) after which a new token in fed into the circuit (drawn uniformly within a $\pm 25\%$ range upon each token firing); "loop" represents a minimal workload with exactly one token in the circuit at any given time. Upon consumption of a token by a process p_i with configuration c, its actual simulated execution time is drawn uniformly within 60% and 100% of its respective specification value $wcet_{i,c}$. Likewise, its instantaneous power consumption is uniformly drawn at each clock cycle, within 60% and 100% of $pp_{i,c}$. As expected, the overall energy consumption per processed data decreases with the size of the sliding window (and the simulated workload). These results empirically confirm that the size of the sliding window considered for the optimization can greatly impact the ability of the limited optimal control algorithm to actually achieve a noticeable reduction in energy consumption.

We further report in Table 2 the run-time performances of ReaX for Design1 (Design2 exhibits similar results). Manager sizes are between 600 and 1 192 logic elements, which are relatively negligible compared to the several hundred thousands to millions of logic elements that currently available FPGAs offer.

Observe that in Fig. 6, a plateau is reached for sliding windows of size greater than 5. For the two designs, this actually corresponds to the iteration i after which the expected outcome as computed in Eq. (2) only ever increases by some fixed amount ξ; *i.e.,* in these cases, $\exists \xi \in \mathcal{X}_{\mathcal{N},X\cup I}, \forall j \geqslant i, \mu_{j+1} = \mu_j + \xi$. Note that our implementation of the algorithm does not attempt to detect these cases to stop the computation of expected outcomes before reaching the full length of the sliding window.

6. CONCLUSIONS & FUTURE WORKS

We have presented a new algorithm for achieving limited optimal control on symbolic system models. This algorithm alleviates the need for specifying target states by operating on a finite sliding window of parameterizable length. It also accepts cost functions directly defined on transitions, which is a novelty among symbolic approaches for optimal control. In order to carry out some empirical evaluations of this new control algorithm, we have also advanced a framework for producing an energy-aware dynamic reconfiguration manager for reactive data-flow circuits. Our technique permits the systematic construction of abstract symbolic models of such designs, as well as associated global control objectives. Through the construction of an efficient simulator using a hardware description language, we have also demonstrated that the symbolic model and the resulting strategy can be translated into a piece of circuit that encodes an efficient configuration manager. By construction, this manager ensures any set of mutual exclusion constraints on the design, and is able to reduce overall energy consumption.

We plan to develop a tool and design guidelines for using our approach on the automated construction of the abstract symbolic models. As stated in Section 2, the strategies are usually computed under the assumption that the environment of the model behaves as an adversary: in a sense the strategy is pessimistic. A natural extension of our work is to take some stochastic models of the environment (*e.g.,* inferred from simulation traces) into account to compute strategies that achieve better energy efficiency *on average*. Last, our symbolic optimization algorithm could be extended in the two following directions: (i) the tool ReaX that we use for computing strategies is already able to enforce safety properties on infinite-state systems. An extension of our limited optimal control algorithm towards handling such systems appears a natural extension of our work; (ii) in some cases, more than one design metrics are relevant for specifying optimal control objectives. Regarding our circuit use-case for instance, one can also identify the dice temperature as a factor that would be interesting to take into account. Handling such new quantitative objectives would require extending our optimal control algorithm towards multi-criteria objectives.

REFERENCES

An, X., Rutten, É., Diguet, J.P., and Gamatié, A. (2016). Model-Based Design of Correct Controllers for Dynamically Reconfigurable Architectures. *ACM Trans. Embed. Comput. Syst.*, 15(3), 51:1–51:27.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.

Berthier, N. and Marchand, H. (2014). Discrete Controller Synthesis for Infinite State Systems with ReaX. In *12th Int. Workshop on Discrete Event Systems*, WODES '14, 46–53. IFAC.

Berthier, N. and Marchand, H. (2015). Deadlock-Free Discrete Controller Synthesis for Infinite State Systems. In *54th IEEE Conference on Decision and Control*, CDC '15, 1000–1007. IEEE.

Clarke, E.M., Emerson, E.A., and Sistla, A.P. (1986). Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.*, 8(2), 244–263.

Dumitrescu, E., Girault, A., Marchand, H., and Rutten, É. (2010). Multicriteria optimal discrete controller synthesis for fault-tolerant tasks. In *10th Int. Workshop on Discrete Event Systems*, WODES '10, 356–363. IFAC, Amsterdam, The Netherlands.

Hietter, Y., Roussel, J., and Lesage, J. (2008). Algebraic Synthesis of Transition Conditions of a State Model. In *9th International Workshop on Discrete Event Systems*, WODES '08, 187–192. IEEE.

Kahn, G. (1974). The semantics of a simple language for parallel programming. *Information processing*, 74, 471–475.

Lee, E.A. and Parks, T.M. (1995). Dataflow process networks. *Proceedings of the IEEE*, 83(5), 773–801.

Marchand, H., Bournai, P., Le Borgne, M., and Le Guernic, P. (2000). Synthesis of discrete-event controllers based on the signal environment. *Discrete Event Dynamic System: Theory and Applications*, 10(4), 325–346.

Marchand, H. and Le Borgne, M. (1998). On the optimal control of polynomial dynamical systems over Z/pZ. In *4th International Workshop on Discrete Event Systems*, 385–390. IEEE, Piscataway, NJ, USA.

Miremadi, S., Lennartson, B., and Åkesson, K. (2012). A bdd-based approach for modeling plant and supervisor by extended finite automata. *IEEE Trans. Contr. Sys. Techn.*, 20, 1421–1435.

Ramadge, P. and Wonham, W. (1989). The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1), 81–98.

Zhang, Y., Roivainen, J., and Mammela, A. (2006). Clock-Gating in FPGAs: A Novel and Comparative Evaluation. In *Proceedings of the 9th EUROMICRO Conference on Digital System Design*, DSD '06, 584–590. IEEE Computer Society, Washington, DC, USA.