8th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS
Suceava, Romania, May 25 – 27, 2006

AN EXTRINSIC EVOLVABLE HARDWARE APPROACH TO LOGIC SYNTHESIS OPTIMIZATION

Codrin PRUTEANU [1], Dan GÂLEA[2], Cristian-Győző HABA[3]
"Gh. Asachi" Technical University
Computer Engineering Department
[1] codrin.pruteanu@gmail.com, [2] dangalea@wtci.ro

"Gh. Asachi" Technical University
Electrical Engineering Department
[3] cghaba@yahoo.com

Abstract. *In this paper we present and implement an Evolutionary Algorithm which determines an optimal decomposition of a specified finite state machine given as input. The main idea is to define and find a cost function that allow us to obtain the best implementation of states assignment for the resulting submachines in order to optimize the speed or area of the initial circuit.*
Keywords: *evolvable hardware, evolutionary algorithms, logic synthesis, optimization, finite automata.*

1. Introduction

Traditional circuit design methodologies rely on rules that have been developed over many decades. Due to the increasingly design process the modern circuits design trends are following one of these two main directions: the first is to employ more designers with greater expertise which is expensive, and the second to simplify circuit design by imposing higher levels of abstraction to the design space. The hardware description languages are such an example. The drawback is that this could reduce the potential circuit behavior. Many EDA modern EDA tools use intelligent techniques in the optimization algorithms, but the circuits design along with some optimization decisions during the synthesis process are still in the domain of the human designer. Only in recent approaches there are implemented some evolutionary techniques to higher up the VLSI circuit design flow in order to generate creative designs that can rival or improve the human implementations. The recent approach defined by the new field of evolutionary techniques allow us to use the evolution mechanism to automatically design the circuits for us. This field has been classified as evolutionary electronics or hardware evolution. We will name it Evolvable hardware (EH). Evolvable hardware (EH) refers to hardware that can change its architecture and behavior dynamically and autonomously by interacting with its environment. At present, almost all EH use an evolutionary algorithm (EA) as their main adaptive mechanism. The EH respects the Darwinian evolutionary theories. There are different views about EH. One view regards EH as an application of evolutionary techniques to circuit synthesis. Another view regards EH as hardware that is capable of online adaptation through reconfiguring its architecture dynamically and autonomously. EH is different from the hardware implementation of EA's, in which the hardware architecture does not change and is used to implement EA functions, such as selection, recombination and mutation. The main advantage of hardware implementation of EA's is to speed up the execution of EA functions. The increase of speed does not imply faster EA execution because it does not speed up the fitness function evaluation which is often the most time-consuming part of an EA application. The EA can be classified into genetic algorithms (GA), genetic programming (GP), evolutionary programming (EP), and evolutionary strategies (ES). Another classification of the EH can be described by it's implementation: extrinsic and intrinsic (de Garis [1]). Extrinsic EH simulates evolution by software and downloads the best configuration to hardware in each generation only once. Intrinsic EH simulates the evolution process directly in hardware, every chromosome

being used to dynamically reconfigure the hardware. This results in a number of reconfigurations equal to the population size in each generation. One of the advantages of the extrinsic EH is that one can obtain symbolic representation of circuits which can be implemented in a variety of digital platforms. Recent EH approaches focuses on evolutionary design of complex circuits in order to develop dynamically evolved hardware. According to the level of chromosome representation, the design implementation can be classified into direct and indirect approaches. In the direct approach the EH encodes the circuit's architecture bits as chromosomes, which specify the connectivity and functions of different hardware components at low level of the circuit. On the other hand, the indirect approach, does not evolve architecture bits directly. It uses a higher level representation, such as trees or grammars, as chromosomes, which are then used to generate circuits. A grammar is represented by a production diagram which can be a directed graph. The evolutionary algorithm most commonly used to evolve hardware is the genetic algorithm, where each trial circuit design is encoded as a bit string. The algorithm explicitly separates the genetic information that is recombined and mutated (the genotype) from the current circuit that is evaluated (the fenotype). The fitness function needs to evaluate each step of the evolutionary algorithm and must be calculated for each member of the evolving population. The canonical genetic algorithm has three operators: selection, crossover and mutation. There is also another operator called elitism, which can be added to the list of the well know operators. This will be used for us in finding the best solution for the current population.

2. Previous work

The partitioning technique developed by Hartmanis and Stearns is until now the most systematic study of the subject [2]. One of its drawbacks is that not all state machines have well behaved closed partitions. An FSM with n states requires a minimum of s state variables for the assignment, where $s = [\log_2 n]$.

The existence of closed partitions on the states of a FSM gives an insight into the structure of the circuit. If the state assignment code is based on closed partitions then the circuit is decomposed into smaller units operating in parallel, serial or as a composite structure. In fact the structure can be predicted by simple inspection of the lattice diagram of the partitions.

The reduced dependency between the state variables after decomposition normally results in simpler logic equations. It should be pointed out that not all state machines have closed partitions though in theory they can be made to do so by state splitting.

State assignment is a mapping from the set of states (symbolic names) of a Finite State Machine (FSM) to the set of binary codes. FSMs are commonly described by a State Transition Graph (STG) or State Transition Table (STT). The desirable feature of the state assignment is to assign binary codes to the states, in the STG, in such a way that can optimize some objective function. Although not necessarily, the objective function is to minimize the area of the combinational circuit required to realize the FSM. For this purpose, once binary codes have been assigned to states by some state assignment algorithm, next-state and output equations are defined and optimized with logic minimization tools. Therefore, state assignment must be performed in such a way that favors the simplification of the next-state and the output logic. The problem of finding an optimal state assignment is NP-hard.

3. Problem Formulation

A FSM can be classified as deterministic or non-deterministic, and completely or incompletely specified. A completely specified state machine M can be described by a tuple $\boldsymbol{M} = (I, S, O, \delta, \lambda)$, where $\boldsymbol{I}$ is a set of primary inputs, $\boldsymbol{S}$ is a set o state symbols, $\boldsymbol{O}$ is a set of primary outputs, $\boldsymbol{\delta} : \boldsymbol{I} \times \boldsymbol{S} \rightarrow \boldsymbol{S}$ is the next state function, and $\boldsymbol{\lambda} : \boldsymbol{I} \times \boldsymbol{S} \rightarrow \boldsymbol{O}$ is the output function for Mealy machines or $\boldsymbol{\lambda} : \boldsymbol{S} \rightarrow \boldsymbol{O}$ is the output function for Moore

machines. Moore machines can be considered a special case of Mealy machines. Therefore the set of theories and methods used for Mealy machines will be also applicable to Moore machines. For this reason in this paper we will focus on Mealy FSMs. In the case of incompletely specified FSMs, $\boldsymbol{I}$ represents a finite set of inputs, $\boldsymbol{S}$ is a finite non-empty set of internal states, $\boldsymbol{O}$ is a finite set of outputs, $\boldsymbol{\delta : I \times S \rightarrow 2^S}$ is the next-state function, and $\boldsymbol{\lambda : I \times S \rightarrow 2^O}$ is the output function.

Definition: A **partition** Π on a set of states S is a collection of disjoint subsets of S, called blocks, whose set union is S. A partition Π on the set of states S of a machine M is said to be a **closed partition** if and only if for any two states s and t which are in the same block of Π, and for any input $i \in I$, the next states $\delta(s,i)$ and $\delta(t,i)$ are in a common block of Π. A partition is a **general partition** if it is not closed.

The original machine is called the **prototype machine** and the individual machines that make up the overall realization are called the **submachines**. The machine obtained as a result of the decomposition is called the **decomposed machine** and is implemented as a network of interconnected submachines. The general structure of such a network is shown in Fig.1.

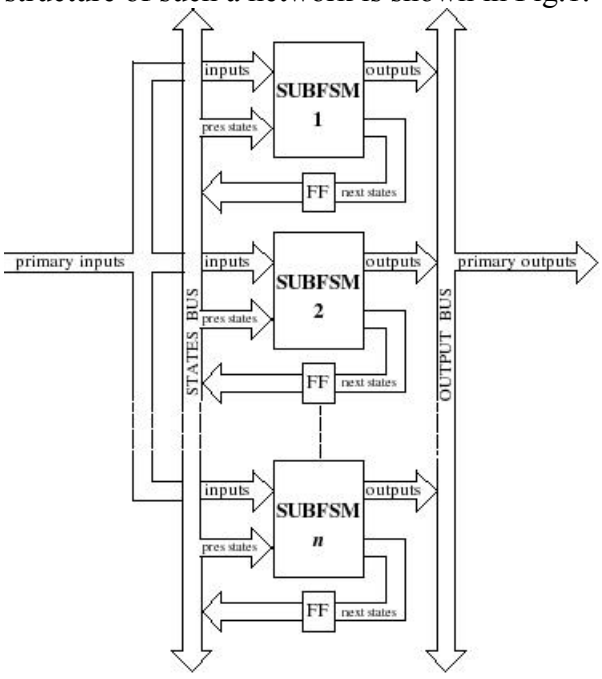

Figure1. General Decomposition topology.

Zero partition $\Pi(0)$ denote a partition with |S| blocks such that each block contains exactly one state. It can be shown that a machine M can be decomposed into a set of n interacting machines that perform the same function as M if and only if there exists a set of nontrivial partitions:
$\Pi_1, \Pi_2, \ldots, \Pi_n$ such that $\Pi_1 . \Pi_2 \ldots \Pi_n = \Pi(0)$

Definition: A legal decomposition of a machine M exists if for a given set of partitions $\Pi_1, \Pi_2, \ldots, \Pi_n$, their product is equal to $\Pi(0)$.

4. Implementation

The Evolutionary Algorithm used in this paper to produce the evolved circuit designs is a simple form of (1+λ)-ES evolutionary strategy. The algorithm is as follows:

Step1. Randomly initialize a population of gentypes subject to constraints imposed by the feed-forward nature of the circuits
Step2. Evaluate fitness of genotypes
Step3. Copy fittest genotype into new population
Step4. Fill remaining places in population by mutated versions of fittest genotype
Step5. Replace old population by new and return to step 2 unless stopping criterion reached

5. Results and Conclusions

It has been shown in this paper that one can explore a much larger space of possible designs by employing an evolutionary algorithm together with a process of assembling and testing the designs. The paper has examined some novel approaches regarding the role of evolutionary algorithms as a novel methodology design. An automated approach is essential.

References

[1] De Garis H, (1994) *An artificial Brain: ATR's CAM-Brain Project Aims to Build/Evolve an Artificial Brain with a Million Neural Net Modules Inside a Trillion Cell Cellular Automata Machine*, New Generation Computing J. 12, no. 2, pp. 215-221

[2] R. E. Stearns and J. Hartmanis (1961) *"On the state assignment problem for sequential machines II",* IRE Trans. Elect. Comput., vol. EC-10, pp.593-603.