A Hierarchical Mathematical Model for Automatic Pipelining and Allocation using Elastic Systems

(Invited Paper)

Jordi Cortadella and Jordi Petit
Department of Computer Science
Universitat Politècnica de Catalunya, 08034 Barcelona, Catalunya

Abstract

The advent of FPGA-based accelerators has encouraged the use of high-level synthesis (HLS) for rapid prototyping and design space exploration. In this context, design optimization at behavioral level becomes a critical task for the delivery of high-quality solutions. Time elasticity opens a new avenue of optimizations that can be applied after HLS and before logic synthesis, proposing new sequential transformations that expand beyond classical retiming and enlarge the register-transfer level (RTL) exploration space. This paper proposes a mathematical model for RTL transformations that exploit elasticity to select the best implementation for each functional unit and add pipeline registers to increase performance. Two simple examples are used to validate the effectiveness and potential benefits of the model.

I. Introduction

High-level synthesis (HLS) [1] has raised the abstraction level at which designers can specify the behavior of a system. HLS produces a register-transfer level (RTL) description in which operations are scheduled in time and resources are assigned to operations [2].

The RTL description usually consists of a datapath and a control unit. The datapath contains functional units (e.g. arithmetic units) and registers that perform operations and store data, respectively. The control unit determines the execution steps of the algorithm and the dataflow across functional units.

The functional units are often obtained from a library of reusable building blocks (integer/floating-point adders/multipliers, shifters, etc), where different implementations are eligible for each block to trade-off area and performance. For example, a ripple-carry or carry-lookahead adder can be selected to perform additions. The library may also contain implementations with different timing characteristics, e.g. a combinational or sequential multiplier.

During HLS, performance analysis must resort to simple delay models to estimate the critical paths of the system. Such estimations may suggest chaining multiple units on the same cycle and reducing the number of execution steps or having multi-cycle operations to avoid excessively long cycle periods [3]. After synthesis, the common strategy for HLS tools is to deliver cycle-accurate static schedules that cannot be modified during RTL synthesis. This means that a combinational unit cannot be substituted by a sequential one or that two chained operations cannot be separated by a register.

Elastic circuits [4] have emerged as an alternative to design correct-by-construction systems with elastic timing. The basic idea behind this concept is that timing can be *elasticized* by using handshake signals

This work has been supported by funds from the Spanish Ministry for Economy and Competitiveness and the European Union (FEDER funds) under grant TIN2013-46181-C2-1-R, and the Generalitat de Catalunya (2014 SGR 1034).

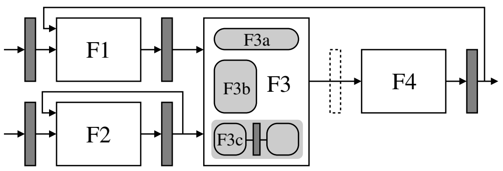

Fig. 1. Example to illustrate RTL optimizations.

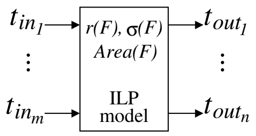

Fig. 2. Interface of the hierarchical ILP model.

that indicate the availability of data and resources. Thus, elastic modules can be connected with a modular plug-and-play interconnectivity paradigm. By using elastic units, timing can be modified without affecting functionality.

II. Overview

This paper proposes a Mixed-Integer Linear Programming (MILP) model that enables the exploration of RTL optimizations using elastic timing. To illustrate the set of optimizations that can be handled by the model, we use the simple example depicted in Fig. 1.

The figure shows an RTL block diagram with functional units (F1-F4) and registers (shadowed boxes). Retiming [5] is one of the transformations implicitly represented by the model. However, elasticity also allows the model to support the insertion of *bubbles* (registers with invalid data), as proposed in [6]. For example, by inserting a bubble between F3 and F4 (dotted box), the cycle period (measured as ns/cycle) could be reduced. However, this also reduces the throughput of the system, since the cycle F1-F3-F4-F1 would contain three registers: two with valid data (tokens) and one with invalid data (bubble). The throughput would be reduced to 2/3 tokens/cycle. Still, the important figure of merit is performance (tokens/ns). If each functional unit would take a 1 ns delay, then cycle period would be reduced from 2 ns (chain of F3-F4) to 1 ns, thus increasing performance from 1/2 to 2/3 tokens/ns.

An important RTL optimization is the binding (selection) of units to operations in case the library contains multiple implementations for the same operation. Figure 1 shows three possible implementations for F3. Two of them (F3a and F3b) are combinational units with different area/performance parameters. The third one (F3c) is a pipelined unit represented as a netlist of two combinational blocks and one register.

Again, the modularity enabled by elastic systems is what allows the selection of units with different timing characteristics (e.g., combinational or pipelined). In general, any arbitrary netlist of combinational blocks and registers can be accepted as an implementation for an operation, as long as the interface is elastic. The MILP model allows to *select* the best implementation to optimize the desired figure of merit (area or performance) under a set of constraints (also in area or performance).

The modularity of elastic systems can also be exported to the MILP model in such a way that the top model can be built by hierarchically composing the MILP models of the components. Each MILP model provides a set of interface variables that can be *connected* to other MILP models according to the structure of the system. Figure 2 depicts the interface variables of each model: arrival times at the inputs/outputs of

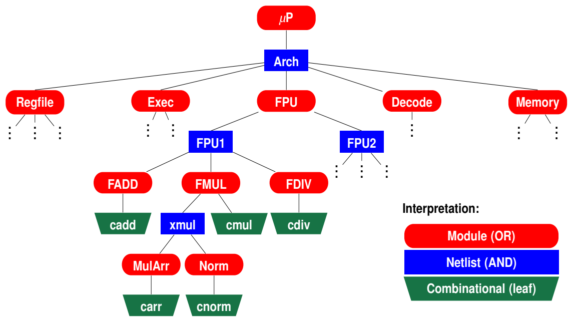

Fig. 3. Example of architectural hierarchy.

the component (t_{in}, t_{out}), area ($Area(F)$), register retiming ($r(F)$) and token retiming ($\sigma(F)$). It is important to realize that $r(F)$ and $\sigma(F)$ are used to model retiming hiearchically, i.e., moving registers and tokens across the full unit F without modifying the internal structure of the unit. Fluid retiming is used to model the throughput of the system.

The next sections describe the details of the MILP model and illustrate how RTL optimizations can be exploited by using two simple examples.

III. Hierarchical specification

We can consider a system as a composition of *modules* in which each module has an *interface* and an internal implementation. The interface specifies the set of input/output *ports* and their bitwidth. A composition of modules is defined by connecting ports via channels.

Every module can have various architectures (implementations). For example, the module *multiplier* may have two architectures: a combinational multiplier (*cmul*) and a sequential one (*smul*). All architectures have the same interface.

Architectures are defined recursively. An architecture can be either

- a *combinational* block, possibly with registers at its input/output ports. Combinational architectures are the leaf components of the hierarchy, i.e., they do not contain other modules, or
- a *netlist* of modules connected by *channels* (see Fig. 1). Each channel can hold registers and each module can recursively have various architectures (implementations). There is no limit in the depth of the hierarchy as long as it is finite, i.e., no cyclic definitions exist.

The combinational (leaf) blocks of the hierarchy have attributes that describe the area of the block and the pin-to-pin delay of the combinational logic. A cost-per-bit attribute also defines the cost of registers. The total cost of each register finally depends on the bitwidth of the channel where it is located.

Figure 3 depicts an architectural hierarchy that describes a microprocessor. Modules are represented by ovals, netlists by rectangles and combinational blocks by trapezoids. The root module (μP) represents the microprocessor and only has one architecture (Arch). The architecture includes several modules (Regfile, Exec, ...) and their interconnects. Each module may have different architectures. E.g., there are two architectures for the FPU unit: FPU1 and FPU2. FPU1 is a netlist with three components (FADD, FMUL and FDIV). FMUL has two architectures, the former being a netlist of two modules and the latter being a combinational architecture.

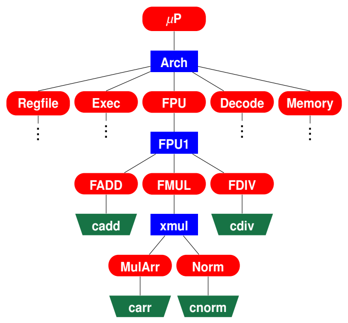

Fig. 4. A subtree representing a possible implementation (OR nodes only have one child).

IV. The MILP model

In this section we describe the elements of the MILP model that, given an architectural hierarchy, selects its best implementation in order to optimize area or performance, subject to performance or area constraints.

The underlying theory of the model is based on the one presented in [6]. Two main contributions are incorporated to that model:

- It can handle hierarchy without the need to flatten the original specification.
- It allows to have several architectures for each module and select the best one according to the cost function and constraints.

On one hand, the MILP model will perform an exploration that is represented by an *AND-OR* tree. Modules correspond to OR nodes that represent all possible choices for their implementation. In a particular implementation, each module shall select exactly one architecture. Netlists are AND nodes, i.e., all child modules of the netlist should be present in the implementation. Finally, combinational nodes are at the leaves of the tree. The exploration returns a possible implementation, that is, a subtree of the hierarchy in which the OR nodes preserve exactly one child and the AND nodes preserve all children. Fig. 4 depicts a possible implementation obtained from the architectural hierarchy in Fig. 3.

On the other hand, the MILP model will also have to decide the location of the registers according to the rules of retiming and bubble insertion.

The MILP model is constructed hierarchically in a way that each node in the hierarchy defines a set of local variables and constraints. The top model is finally obtained by joining all the constraints and by relating the variables associated to the interfaces of the nodes according to the connectivity provided by the netlists.

In the following, we detail in turn the variables and constraints for each component in the hierarchy, for the combinational blocks (leaf nodes), for the netlist blocks (AND nodes), and for the selection blocks (OR nodes). We finally present the possible optimization functions.

Note that some constraints introduced in this section use conditions; these may be easily linearized using standard piecewise linear functions tricks. Also, unless said otherwise, all variables are real.

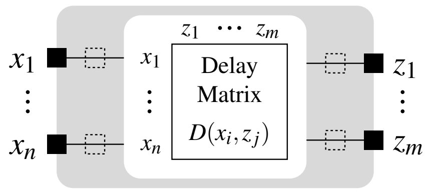

Fig. 5. Model for a combinational architecture.

A. Global variables

The model has three global variables that determine the area and performance parameters of the system. These variables can be defined as constants (as a constraint) or used in the cost function for optimization:

$$\begin{aligned} A_{\max}: &\quad \text{The maximum area of the system} \\ T_{\max}: &\quad \text{The maximum cycle period} \\ \theta: &\quad \text{The minimum throughput} \end{aligned}$$

B. Interface variables at each component

For all nodes N in an architectural hierarchy, the MILP defines the following *interface variables*:

- Variable $Area(N)$ is the area of component N.
- For all ports p of N, the variables $t(p)$, $r(p)$ and $\sigma(p)$. The variable $t(p)$ represents the arrival time at p, whereas the integer variable $r(p)$ and the real variable $\sigma(p)$ represent the retiming value and the fluid retiming of tokens (used to calculate throughput) at p, respectively..

In order to ensure that the arrival times do not exceed the cycle period, the following constraint is defined for every port p: $t(p) \leq T_{\max}$.

When a module can be implemented by several architectures, a binary selection variable $sel(A)$ is defined for each architecture A.

C. MILP model for combinational architectures

Let A be a combinational architecture node associated to one of the leaves of the hierarchy (see Fig. 5). A has a set of input ports $\{x_1, \ldots, x_n\}$ and output ports $\{z_1, \ldots, z_m\}$. Each port p can potentially hold a register between the environment and the combinational logic (dotted slots in the figure). The combinational logic is also characterized by all the port-to-port delays represented by a delay matrix $D(x_i, z_j)$.

Following the theory in [6], each combinational block A has three variables to model the presence of registers and tokens in the register holders:

- $R(p)$ denotes the number of registers at the holder of port p after retiming.
- $r(A)$ is the retiming value of the block.
- $\sigma(A)$ is the fluid retiming value of tokens.

The fluid retiming of tokens determines the average number of tokens present in a register during runtime. This number is required to compute the throughput.

For the propagation of combinational delays across the register holders, the following two variables are defined for every port p:

- ${}^\bullet t(p)$ denotes the arrival time before the holder.
- $t^\bullet(p)$ denotes the arrival time after the holder.

The arrival time of the output of a register is the clock-to-Q delay (denoted by D_{reg}). For simplicity, all registers are assumed to have the same delay.

The following equations define the constraints for the arrival times at each port p:

$$t^{\bullet}(p) = \begin{cases} {}^{\bullet}t(p) & \text{if } R(p) = 0, \\ D_{\text{reg}} & \text{if } R(p) > 0. \end{cases}$$

The propagation delays across the combinational logic are modeled by the following constraint applied to each output port z:

$${}^{\bullet}t(z) = \max_{x \in X} \left(t^{\bullet}(x) + D(x,z) \right) \leq T_{\max},$$

where X is the set of inputs of A, and $D(x,z)$ is the delay from x to z.

The following constraints define the number of registers at the holders of each input port x and output port z; notice that the inequality allows the insertion of bubbles [6]:

$$R(x) \geq R_0(x) - r(x) + r(A),$$
$$R(z) \geq R_0(z) + r(z) - r(A),$$

where $R_0(p)$ is the initial number of registers at the holders of the port before retiming. Similarly, the constraints for the fluid retiming of tokens to guarantee a minimum throughput θ are

$$\theta \cdot R(x) \leq R_0(x) - \sigma(x) + \sigma(A), \tag{1}$$
$$\theta \cdot R(z) \leq R_0(z) + \sigma(z) - \sigma(A). \tag{2}$$

Finally, the interface variables obbey the following constraints:

For every input port x, $t(x) = {}^{\bullet}t(x)$.

For every output port z, $t(z) = t^{\bullet}(z)$.

$$Area(A) = CombArea(A) + \sum_{p \in Ports(A)} A_{\text{reg}}(p) \cdot R(p).$$

where *CombArea* represents the area of the combinational circuit and $A_{\text{reg}}(p)$ is the area of a register located at port p.

D. MILP model for netlist architectures

Let N be a netlist node in the hierarchy. Let $Mods(N)$ and $Chans(N)$ denote the set of modules and channels of the netlist, respectively. Each channel $c = (p \to q) \in Chans(N)$ is a link between a source port p and a destination port q.

As in combinational architectures, we define a selection binary variable $sel(N)$ to indicate whether N is selected or not. We also define variables $R(c)$ that denote the number of registers for all $c \in Chans(N)$.

The following constraints (over all channels $c = p \to q$) take care of the register and token retiming variables:

$$R(c) \geq R_0(c) - r(p) + r(q),$$
$$\theta \cdot R(c) \leq R_0(c) - \sigma(p) + \sigma(q) \tag{3}$$

where $R_0(c)$ is the initial number of registers present at channel c before applying retiming.

The following constraints (also over all channels $c = p \to q$) propagate times accross channels under the presence of lack of registers:

$$t(q) = \begin{cases} t(p) & \text{if } R(c) = 0, \\ D_{\text{reg}} & \text{if } R(c) > 0. \end{cases}$$

The area is defined as:

$$Area(N) = \sum_{M \in Mods(N)} Area(M) + \sum_{c \in Chans(M)} A_{\text{reg}}(c) \cdot R(c).$$

with $A_{\text{reg}}(c)$ being the area of a register present at channel c.

E. MILP model for selection modules

A selection module M consists of a set of architectures among which exactly one of them must be selected. Let $Archs(M)$ denote the set of possible architectures of M.

Given an architecture $a \in Archs(M)$ and one of its ports p, we denote by $t_a(p)$, $r_a(p)$ and $\sigma_a(p)$ the variables representing the arrival time and retiming values of p in a. Likewise, we denote by $sel(a)$ and $Area(a)$ the variables representing the selection and area of a.

The following constraint guarantees that exactly one architecture is selected:

$$\sum_{a \in Archs(M)} sel(a) = 1.$$

Bearing in mind that only one of the above selection variables is non-zero, the remaining interface variables for the module can be represented as follows:

$$\begin{aligned}
Area(M) &= \sum_{a \in Archs(M)} sel(a) \cdot Area(a), \\
t(p) &= \sum_{a \in Archs(M)} sel(a) \cdot t_a(p), \\
r(p) &= \sum_{a \in Archs(M)} sel(a) \cdot r_a(p), \\
\sigma(p) &= \sum_{a \in Archs(M)} sel(a) \cdot \sigma_a(p).
\end{aligned}$$

The terms at the RHS of the equations are non-linear. They can be easily linearized by using some known tricks for binary variables.

F. Objective functions

The objective function can be defined according to the constraints of the design and the parameter that is to be optimized. The parameters that can be used are throughput (θ), cycle period ($T_{\max}$) and area ($A_{\max}$). Two options are possible to keep the problem linear:

1) Minimize $A_{\max}$ for a given θ and $T_{\max}$.
2) Minimize $T_{\max}$ for a given $A_{\max}$ and θ.

Notice that the minimization of θ is not possible due to the non-linearity of constraints (1-3) and the presence of θ in the LHS of the inequalities.

An interesting problem is the maximization of performance ($\theta/T_{\max}$) for a given area constraint. This problem must be solved iteratively. This problem must be solved iteratively by defining constants values of θ and minimizing cycle period. One strategy is to solve multiple MILP models by applying a binary search on the value of θ. The strategy is similar to the one presented in [6].

V. Two examples

In this section we present the application of the previous model to two examples that illustrate the capability of design exploration using the proposed mathematical model. The MILP models have been solved using Gurobi [7].

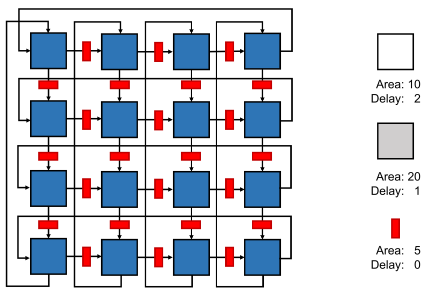

Fig. 6. Sample 4×4 systolic toroidal grid.

A. Systolic grid

Consider the 4×4 systolic toroidal grid shown in Figure 6. Two architectures can implement the combinational modules in the grid: a small but slow architecture (area=10, delay=2, white) and a fast but large architetcure (area=20, delay=1, gray). Let us assume that each register has area=5 and zero delay. The goal is to study which are the Pareto-optimal design points of performance ($\theta/T_{\max}$) with respect to area.

Table I shows the Pareto-optimal choices found by the exploration. Figure 7 depicts the actual Pareto solutions.

It is important to notice that some of the designs include *bubbles* (registers with invalid data.). The bubbles are represented by green rectangles. Bubbles appear when $\theta < 1$ and contribute to reduce the cycle period at the expense of reducing throughput. Even though the introduction of fast modules helps to reduce the cycle period, it has no impact until the particular configuration C is obtained. Solution C guarantees that two fast modules are present in each row and each column without registers between them.

The next Pareto-optimal point achieves the minimum $T_{\max}$. However, this configuration requires the insertion of bubbles to avoid chains of two modules.

The modularity of the MILP model determined by the hierarchical structure of the system introduces some algebraic redundancies, which are usually eliminated by the presolver (e.g., by variable substitution). In this example, the initial MILP model has 1766 rows and 1334 columns, with 578 integer variables, 66 binary variables and 4778 non-zeros. After presolving, the matrix is reduced to 857 rows and 809 columns, with 432 integer and 16 binary variables with 2752 non-zeros. Two minutes of CPU were required to solve the most difficult instances.

B. Fork/join pipeline

We show now another example for architectural exploration. Figure 8 depicts a cyclic pipeline with a join/fork structure. Each stage of the pipeline can hold multiple functional units of the same type separated by communication FIFOs.

Each pipeline stage can have multiple functional units of the same type. The basic unit (F) at stage S has the area and delay shown in the picture. Hence, the throughput achievable by each unit is $\theta(F) = 1/delay(F)$. If a pipeline stage S holds k units of F then we assume that $Area(S) = k \cdot Area(F)$ and $\theta(S) = k \cdot \theta(F)$.

The problem to be solved is: given a constraint on the total area, what is the maximum achievable throughput? The problem can be formulated as an exploration in the same terms proposed in the previous

TABLE I
PARETO-OPTIMAL POINTS FOR THE 4×4 GRID.

Sol	Area	$T_{\max}$	θ	$\theta/T_{\max}$
A	280	4	1	1/4
B	320	2	3/4	3/8
C	360	2	1	1/2
D	480	1	3/4	3/4

TABLE II
PARETO-OPTIMAL POINTS FOR THE EXAMPLE OF FIG. 8.

A	B	C	D	E	Area	θ
1	1	1	1	1	75	1/40
2	2	2	1	1	95	1/30
3	3	3	2	2	185	3/40
4	4	4	3	2	250	1/10

section. The "trick" here is to model the choice of k for each stage as a module with $k_{\max}$ architectures, each one representing a different value for k.

Table II shows the Pareto-optimal points of throughput with respect to area for this example. Every row represents a different design and indicates the number of units associated to each pipeline stage.

VI. CONCLUSION

Elastic systems open new opportunities for the application of RTL optimizations after high-level synthesis. This paper has proposed a mathematical model for a specific set of transformations. The scalability of the model and the extension to other optimizations are some aspects that must be explored in the future.

REFERENCES

[1] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Design and Test of Computers*, vol. 26, no. 4, pp. 18–25, 2009.

[2] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An introduction to high-level synthesis," *IEEE Design and Test of Computers*, vol. 26, no. 4, pp. 8–17, 2009.

[3] D. C. Zaretsky, G. Mittal, R. P. Dick, and P. Banerjee, "Balanced scheduling and operation chaining in high-level synthesis for FPGA designs," in *8th Int. Symp. on Quality Electronic Design (ISQED'07)*, Mar. 2007, pp. 595–601.

[4] J. Carmona, J. Cortadella, M. Kishinevsky, and A. Taubin, "Elastic circuits," *IEEE Transactions on Computer-Aided Design*, vol. 28, no. 10, pp. 1437–1455, 2009.

[5] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.

[6] D. Bufistov, J. Cortadella, M. Kishinevsky, and S. Sapatnekar, "A general model for performance optimization of sequential systems," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, 2007, pp. 362–369.

[7] Gurobi Optimization, Inc., "Gurobi Optimizer Reference Manual," `http://www.gurobi.com`, 2016.

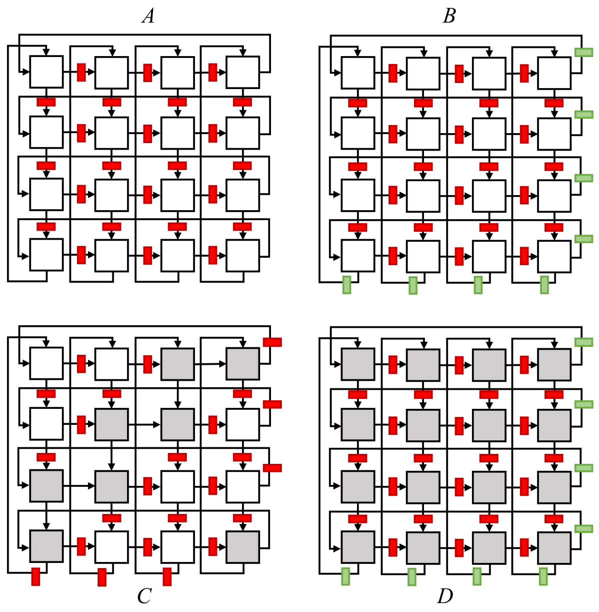

Fig. 7. Pareto solutions for the 4×4 grid.

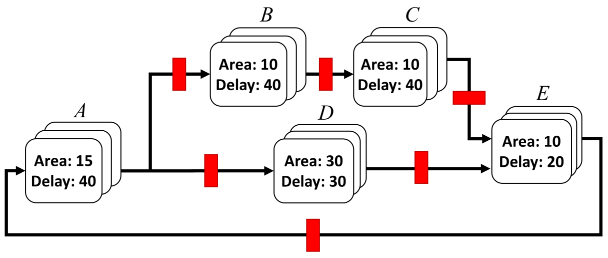

Fig. 8. Sample fork/join pipeline.