Architecture Synthesis for Cost-Constrained Fault-Tolerant Flow-based Biochips

Morten Chabert Eskesen*, Paul Pop*, Seetal Potluri*
*Department of Applied Mathematics and Computer Science, Technical University of Denmark

***Abstract*—In this paper, we are interested in the synthesis of fault-tolerant architectures for flow-based microfluidic biochips, which use microvalves and channels to run biochemical applications. The growth rate of device integration in flow-based microfluidic biochips is scaling faster than Moore's law. This increase in fabrication complexity has led to an increase in defect rates during the manufacturing, thereby motivating the need to improve the yield, by designing these biochips such that they are fault tolerant. We propose an approach based on a Greedy Randomized Adaptive Search Procedure (GRASP) for the synthesis of fault-tolerant biochip architectures. Our approach optimizes the introduction of redundancy within a given unit cost budget, such that, the biochemical application can successfully complete its execution within its deadline, even in the presence of faults, and the yield is maximized. The proposed algorithm has been evaluated using several benchmarks and compared to the results of a Simulated Annealing metaheuristic.**

I. Introduction

Microfluidics-based biochips (also referred to as lab-on-a-chip) integrate different biochemical analysis functionalities on-chip, miniaturizing the macroscopic biochemical processes to a sub-millimeter scale. These microsystems offer several advantages over the conventional biochemical analyzers, e.g., reduced sample and reagent volumes, faster biochemical reactions, ultra-sensitive detection and higher system throughput, with several assays being integrated on the same chip [1], [2].

Biochips are used in many application areas like in vitro diagnostics (point-of-care), drug discovery (high-throughput screening, hit characterization), biotechnology (process monitoring and development) and ecology (agriculture, environment, homeland security) [2]. During the last decade, a significant amount of work has been carried out on the individual microfluidic components as well as the microfluidic platforms [2]. There are several types of biochip platforms, each having its own advantages and limitations [2].

In this paper, we focus on flow-based biochips, in which the microfluidic channel circuitry on the chip is equipped with integrated microvalves, that are used to manipulate more complex units such as mixers, micropumps, multiplexers etc., with several hundreds of such units being accommodated on a single chip [2]. Analogous to its microelectronics counterpart, this approach is called microfluidic Very Large Scale Integration (mVLSI) [3]. The soft lithography technology used for fabricating flow-based biochips, has advanced faster than Moore's law [4].

As these biochips grow more complex (commercial biochips are available, which use more than 25,000 valves and about a million features to run 9,216 polymerase chain reactions in parallel [5]), the manual methodologies used currently will not scale, and hence will become highly inadequate. Therefore, researchers have started to propose top-down design methodologies. A survey of the recent developments in mVLSI, including an overview of the proposed top-down automatic physical design methods, is presented in [6].

A roadblock in the deployment of biochips is their low reliability. Physical defects can be introduced during the fabrication process, which can reduce the manufacturing yield. Additionally, this may lead to failure of the biochemical application, which can be costly because of the need to redo lengthy experiments, using expensive reagents and often hard-to-obtain samples, and can be satefy-critical, e.g: in case of a cancer misdiagnosis. Researchers have started to propose fault models and test techniques for mVLSI biochips [7]. To increase the yield, and also to prevent the failures during the operation of biochip, we advocate the use of fault-tolerant biochip design. To address the case, when the consequences of failure are drastic, researchers have already considered introducing redundancy to provide fault-tolerance in mVLSI biochips. The manually designed "Mars Organic Analyzers" biomarker detector chip [8] is an example in this context.

In this paper, our assumption is that the faults are detected during testing, and that the biochemical application is recompiled offline to avoid the detected faults. We are interested to introduce redundancy during the design (synthesis) stage, such that the applications can still be recompiled and run successfully on a defective biochip. In the past, fault-tolerant design strategies have been proposed for the droplet-based biochips [9]; these biochips have a regular array structure, composed of electrodes which manipulate the droplets. In this context, redundant electrodes are introduced, so that, in case the electrodes in the original architecture become faulty, recompilation can be done to replace faulty electrodes with the redundant ones, to achieve fault-tolerance. However, these approaches are not suitable for mVLSI biochips.

In this paper, we propose a fault-tolerant design strategy, which takes the following inputs: (1) An initial architecture without fault-tolerance, modeled as netlist (i.e., the components in the architecture and their interconnections); (2) An application model represented using a sequencing graph, where each node is an operation and edges capture fluid dependencies; (3) Fault model; and (4) A set of constraints imposed by the designer, and produces a fault-tolerant architecture for the given biochip, such that the biochip unit cost is minimized. An architecture is fault-tolerant, if the application

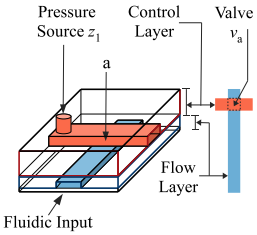


Figure 1. Structure of a valve fabricated using soft lithography

can successfully complete its execution, even in the presence of faults.

II. Fault Model and System Model

Biochips are fabricated using multilayer soft lithography [10], using a cheap, rubber-like elastomer (polydimethylsiloxane, PDMS) with good biocompatibility and optical transparency being used as the fabrication substrate. Physically, the biochip can have multiple layers, but the layers are logically divided into two types: *flow layer* and the *control layer*. The basic building block of a biochip is a micro-mechanical valve (also known as microvalve, as shown in Fig. 1), which restricts/permits the fluid flow, and hence used to manipulate the fluid in the flow layer (blue). As shown in Fig. 1, the microvalve is controlled using an external air pressure source, through a control pin z_1. Control pins provide access to the control layer (red). The flow layer is connected to a fluid reservoir, through a pump that generates the fluid flow. When the pressure source is not active, the fluid is permitted to flow freely (open valve). When the pressure source is activated, high pressure causes the elastic control layer to pinch the underlying flow layer (point a in Fig. 1), thereby blocking the flow at point a (closed valve). Next, we describe our fault model.

A. Fault Model

It is known that the growth rate in complexity of these mVLSI biochips is faster than Moore's law [4], thereby increasing the defective rates proportionately. Broadly speaking, the consequences of the defects can be described as either a block or a leak [7], [11]. We assume that the designers, will specify for a biochip $\mathcal{A}$, a fault model $\mathcal{Z} = (\mathcal{VF}, \mathcal{CF}, v_{max}, c_{max})$, where $\mathcal{VF}$ and $\mathcal{CF}$ are a finite set of valve and channel faults respectively, while v_{max} and c_{max} specify the maximum number of valve and channel faults, respectively, that may occur in the architecture $\mathcal{A}$ at the same time. We have a general fault model, where the sets $\mathcal{VF}$ and $\mathcal{CF}$ contain all the valves and all the channels, respectively. This means that the biochip $\mathcal{A}$ may experience any combination of v_{max} valve failures and c_{max} channel failures. However, we also allow the designers to specify a certain list of faults based on their experience, gained from fabricating biochip $\mathcal{A}$, as follows. A valve fault $VF(N, w, t) \in \mathcal{VF}$ may occur in a component $N \in \mathcal{N}$, where w is the particular valve affected inside component N, and t denotes the type of fault, which could be *stuck open* or *stuck closed*. Similarly, a channel fault $CF \in \mathcal{CF}$ may occur either in a component $N \in \mathcal{N}$ denoted with $CF(N, t)$ or in a channel D, denoted with $CF(D, t)$, where t is the type of fault, which can be a *block* or a *leak*. An example fault list specified by the designers, for the architecture in Fig. 3 (that uses the mixer in Fig. 2a), is given in Table II. We define a *fault scenario* as any combination of faults in the fault model $\mathcal{Z}$, considering the maximum number of faults v_{max} and c_{max}. Thus, a possible fault scenario in our example could be $\{VF_1, VF_3, CF_1, CF_3\}$. Considering all combinations of faults in $\mathcal{Z}$, $|\mathcal{Z}|$ fault scenarios are possible. We define an architecture as being fault-tolerant to $\mathcal{Z}$, if an application $\mathcal{G}$ is able to complete successfully within its deadline $d_{\mathcal{G}}$ under all ($|\mathcal{Z}|$, in our case) the possible fault scenarios in $\mathcal{Z}$. Next, we describe how to design components, such that they tolerate faults.

Table I
Component Library ($\mathcal{L}$): Flow Layer Model

Component	Phases($\mathcal{P}$)	C	H
Mixer	Ip1 / Ip2 / **Mix** / Op1 / Op2	0.5 s	30 × 30
FT-Mixer	Ip1 / Ip2 / **Mix** / Op1 / Op2	0.5 s	30 × 30
Filter	Ip / **Filter** / Op1 / Op2	20 s	120 × 30
FT-Filter	Ip / **Filter** / Op1 / Op2	20 s	120 × 60
Detector	Ip / **Detect** / Op	5 s	20 × 20
FT-Detector	Ip / **Detect** / Op	5 s	20 × 40
Separator	Ip1 / Ip2 / **Separate** / Op1 / Op2	140 s	70 × 20
FT-Separator	Ip1 / Ip2 / **Separate** / Op1 / Op2	140 s	70 × 40
Heater	Ip / **Heat** / Op	20° C/s	40 × 15
FT-Heater	Ip / **Heat** / Op	20° C/s	40 × 30
Storage	Ip or Op	-	90 × 30
FT-Storage	Ip or Op	-	90 × 40
Metering	Ip / **Met** / Op1 / Op2	-	30 × 15
Multiplexer	Ip or Op	-	30 × 10

B. Fault-Tolerant Components

The components of a biochip are built using microvalves and channels, e.g., mixers (see an example mixer in Fig. 2a), switches (which control the fluid flow at channel intersections), micropumps etc. We use a dual-layer component modeling framework, consisting of a *flow layer model* and a *control layer model*. The flow layer model $(\mathcal{P}, C, H)$ of each component $N \in \mathcal{N}$ is characterized by a set of operational phases $\mathcal{P}$, execution time C and geometrical dimensions H. Table I shows a flow layer model library, obtained from [12]. In this table, columns $\mathcal{H}$ and $\mathcal{C}$ give the geometrical dimensions (length×width, scaled with a unit length being equal to 150 μm [12]) and the execution times of the component respectively. The execution times do not include the time required to fetch the input fluids or to remove the output fluids from the component. The control layer model captures the valve actuation details $\mathcal{C}$ required for the on-chip execution of all operational phases $\mathcal{P}$ of a component. The components in the library $\mathcal{L}$ are built from microvalves and channels; some components such as heaters also include actuators, and components such as detectors consist of sensors. Fig. 2(a) shows the example of a pneumatic mixer, implemented using nine microfluidic valves, v_1 to v_9. The valve-set $\{v_1, v_2, v_3\}$ acts as the input switch, $\{v_7, v_8, v_9\}$ as the output switch and $\{v_4, v_5, v_6\}$ as the on-chip pump used

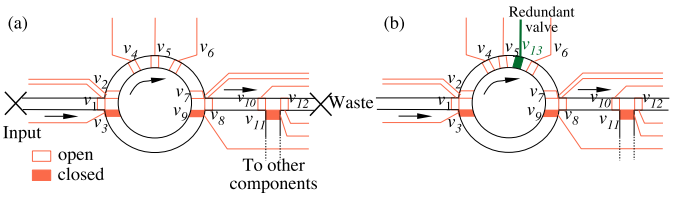


Figure 2. Rotary mixer: (a) regular design (b) fault-tolerant design

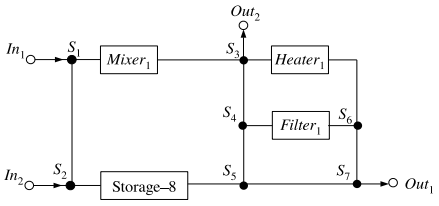


Figure 3. Biochip architecture example

Table II
EXAMPLE FAULT MODEL FOR ARCHITECTURE IN FIG. 3

Name	**Vertex** ($N \in \mathcal{N}$)/ **Valve affected** (w)	**Type** (t)
VF_1	$Mixer_1$ / v_5	Open
VF_2	S_6 / v_3	Open
VF_3	S_5 / v_2	Open
VF_4	S_3 / v_3	Open
Name	**Component** ($M \in \mathcal{N}, \notin \mathcal{S}$) **/ Connection** $D_{i,j} \in \mathcal{D}$	**Type** (t)
CF_1	$Heater_1$	Block
CF_2	$Filter_1$	Block
CF_3	$S_2 \rightarrow$ Storage-8	Block
CF_4	$S_1 \rightarrow Mixer_1$	Block

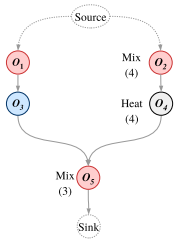


Figure 4. Example application graph $\mathcal{G}$

to perform the mixing. To mix, input and output valves (v_1 and v_8) are closed, while valves in the valve-set $\{v_2, v_3, v_7, v_9\}$ are opened and the mixing operation is initiated, by opening and closing valves in the valve-set$\{v_4, v_5, v_6\}$ in a sequence which generates a pumping action [12]. We assume that the designers will include in library $\mathcal{L}$, the fault-tolerant versions of some components (prefixed with "FT–" in Table I), which are able to tolerate certain types of faults. A possible fault-tolerant design of the mixer in Fig. 2(a) is presented in Fig. 2(b) (see [11] for details). The critical function of the mixer in Fig. 2(a), is its pumping action, achieved by the valve set $\{v_4, v_5, v_6\}$. The mixer in Fig. 2(b) can tolerate one stuck-open valve failure in the pump $\{v_4, v_5, v_6\}$ by adding a redundant valve v_{13} to the pump. As explained earlier, the failure is determined during testing, and the application is recompiled to avoid the failing valve and use the redundant valve as a replacement. Depending on its design, a fault tolerant component will tolerate a given set of faults, and may use an increased chip area and execution time, due to the added redundancy, as shown earlier in Table I.

C. System model

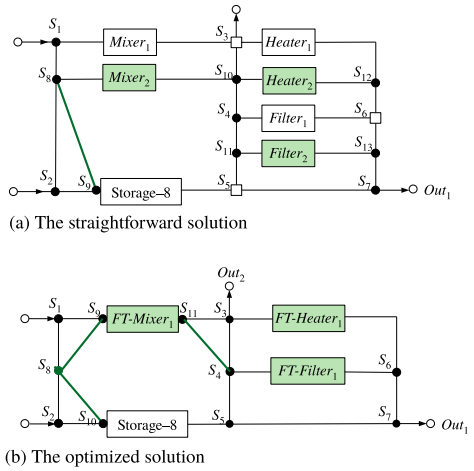
We model the biochip architecture as a topology graph $\mathcal{A}(\mathcal{N}, \mathcal{D})$, where each element in the vertex set $\mathcal{N}$ represents a component and each element in the edge-set $\mathcal{D}$ represents a fluidic channel. An example biochip architecture shown in Fig. 3, which has two inputs (In_1 and In_2), two outputs (Out_1 and Out_2), one mixer, one heater, one filter, seven switches (S_1 to S_7) and eight storage reservoirs (the component 'Storage-8). We model a biochemical application using a sequencing graph $\mathcal{G} = (\mathcal{O}, \mathcal{E})$, where each vertex $O_i \in \mathcal{O}$ represents an operation that can be bound to a component N_j using a binding function $\mathcal{B} : \mathcal{O} \rightarrow \mathcal{N}$, and the edge set $\mathcal{E}$ models the dependency constraints in the assay. Each vertex has an associated weight $C_i^{N_j}$, which denotes the execution time required for the operation O_i to be completed on component N_j. Similarly, $e_{i,j} \in \mathcal{E}$ is an edge from O_i to O_j, which indicates that the output of O_i is the input of O_j. Applications have a deadline $d_{\mathcal{G}}$, by which they have to complete their execution. Next, we describe our problem formulation.

III. PROBLEM FORMULATION

Given a netlist $\mathcal{A}$, a component library $\mathcal{L}$, an application graph $\mathcal{G}$, with a deadline $d_{\mathcal{G}}$ and a fault model $\mathcal{Z}$, the problem is to determine a fault tolerant netlist, $\mathcal{A}^+$, such that the biochip is fault tolerant and its unit cost is minimized.

Let us consider the architecture $\mathcal{A}$ in Fig. 3, which does not have any redundancy for fault-tolerance and the application graph $\mathcal{G}$ in Fig. 4. We are interested in synthesizing an architecture $\mathcal{A}^+$ such that when application $\mathcal{G}$ is run on it, it is tolerant to the faults given by $\mathcal{Z} = (\mathcal{VF}, \mathcal{CF}, 2, 2)$, with $\mathcal{VF}$ and $\mathcal{CF}$ given by Table II and such that the biochip unit cost is minimized. A straightforward solution (SFS) to the problem is to add redundancy to the original netlist $\mathcal{A}$, for *each* fault listed in the fault model $\mathcal{Z}$, such that the locally added redundancy helps in tolerating the faults. For our example, such a straightforward solution (SFS) is depicted in Fig. 5a. Thus, fault-tolerant switches, S_3, S_5 and S_6, have been added to compensate for the failing valves in $\mathcal{Z}$. Similarly, a redundant channel has been added to compensate for the blocked channel $S_2 \rightarrow$ Storage-8. No channel has been added to make the $S_1 \rightarrow Mixer_1$ channel redundant, as it is not needed by virtue of $Mixer_2$. Such an approach leads to a costly fault-tolerant architecture $\mathcal{A}^+$, because not all faults in $\mathcal{Z}$ will occur simultaneously (maximum $v_{max} = 2$ valve faults and $c_{max} = 2$ channel faults may occur according to the model).

We define the unit cost $Cost_{\mathcal{A}^+}$ as the sum of the total number of valves and channels in $\mathcal{A}^+$, which represents a measure of the complexity of the biochip (in principle, our method is capable of using any other cost models, provided by the designer). Thus, for Fig. 5a, we have $Cost_{\mathcal{A}^+} = 129$.



(a) The straightforward solution

(b) The optimized solution

Figure 5. Fault-tolerant architecture solutions

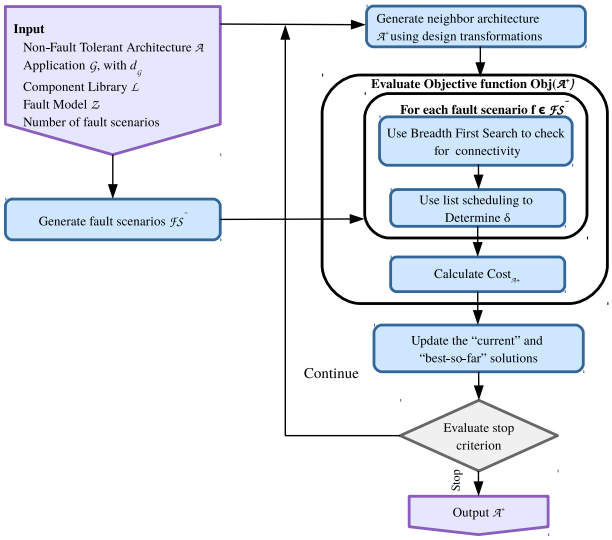


Figure 6. Fault Tolerant Architecture Synthesis (FTAS)

The focus of this paper is to propose a method to optimize the introduction of redundancy such that the cost of the fault tolerant architecture $\mathcal{A}^+$ is minimized. Such an optimized solution, which is also fault-tolerant to all the possible fault scenarios in $\mathcal{Z}$, is depicted in Fig. 5b. In this solution, the fault tolerant variants of filter, heater and mixer have replaced the original components, and thus can tolerate some of the faults in the fault model. In addition, two redundant channels have been added to compensate for the two blocked channels. Routing is still possible even though there are valves failing in some switches. The cost of the architecture in Fig. 5b is 96, which is significantly cheaper compared to SFS. The assumption is that the fault-tolerant architecture synthesis is part of a methodology, which has the following steps: (1) An architecture $\mathcal{A}$ is created by the designer either manually or using synthesis tool flows [12], [13]; (2) A fault-tolerant netlist $\mathcal{A}^+$ is synthesized for $\mathcal{A}$; (3) Considering an application graph $\mathcal{G}$ with a deadline $d_{\mathcal{G}}$ and a fault model $\mathcal{Z}$; (4) Physical synthesis of the fault-tolerant netlist $\mathcal{A}^+$ thus generated in step 2 using a physical synthesis tool like [12]; (5) Fabricating the biochip using the physical layout obtained in the step 3; and finally (6) Testing and diagnosing each of the biochips to determine if they have permanent faults using techniques such as the one proposed in [7]. The locations of faults are determined during this step. We compile the biochemical application $\mathcal{G}$ on $\mathcal{A}^+$, avoiding the use of faulty components or channels, using a technique such as [14].

The focus of this paper is on the second step of the methodology i.e., *the fault-tolerant architecture synthesis problem*. Note that if in Step 5 (Testing), we detect faults which are not in the fault model $\mathcal{Z}$ given by the designer, or if the netlist $\mathcal{A}^+$ turns out not to tolerate a detected fault present in $\mathcal{Z}$, we will discard the chip. This will reduce the yield of the fabrication process, but will guarantee that all the biochips can be successfully used even if they contain permanent faults. The next section explains the details of the optimization strategy used to synthesize the fault-tolerant netlist $\mathcal{A}^+$.

IV. Optimization Strategy

The problem presented in the previous section is NP-hard [15], because we have to use binding and scheduling of $\mathcal{G}$ on $\mathcal{A}^+$, to check if $\mathcal{A}^+$ can execute $\mathcal{G}$ such that it completes within its deadline $d_{\mathcal{G}}$, in every fault scenario in $\mathcal{Z}$. Our optimization strategy is depicted in Fig. 6 and it is based on Greedy Random Adaptive Search Procedure (GRASP) [16]. GRASP is an iterative metaheuristic, which consists of a construction phase and a local search phase. In the construction phase, the GRASP algorithm builds an initial solution based on a list of candidate design transformations. Subsequently, the local search phase explores the neighborhood of this initial solution to find the local optimum. Each solution visited is evaluated using an objective function, which has to be minimized. GRASP runs for a number of iterations and returns the best solution amongst them. We use the following objective function to evaluate a visited architecture $\mathcal{A}^+$

$$Obj(\mathcal{A}^+) = W_{ft} \times \sum_{f \in \mathcal{FS}^-} \neg ft + W_s \times \sum_{f \in \mathcal{FS}^-} max(0, \delta - d_{\mathcal{G}}) + Cost(\mathcal{A}^+) \quad (1)$$

The objective function has three terms. The third term, $Cost(\mathcal{A}^+)$, captures the unit cost of the architecture, which has to be minimized, and which is defined in Section III. The first two terms check if $\mathcal{A}^+$ is fault-tolerant.

The first term checks if, in every fault scenario $f \in \mathcal{FS}$, where $\mathcal{FS}$ is the set of all possible fault scenarios in $\mathcal{Z}$, the architecture is still *connected*, considering the faults in f. An architecture is connected (denoted with Boolean variable ft) if there exists a route between any two components that have to be connected, i.e., have to exchange fluids. If the components become disconnected because of the faults in f (denoted with $\neg ft$), we cannot successfully run the application. We use a Breadth First Search algorithm to check for connectivity. Next, to successfully run our application $\mathcal{G}$ on $\mathcal{A}^+$, we need

to complete its execution within its deadline $d_{\mathcal{G}}$, in every fault scenario $f \in \mathcal{FS}$. This is checked in the second term of the objective function. We use binding and scheduling to determine the completion time $d_{\mathcal{G}}$ of the application $\mathcal{G}$ on $\mathcal{A}^+$ with the faults in f, which is then compared to the deadline $d_{\mathcal{G}}$ (see Section IV-B).

Note that the first two terms are constraints, i.e., if the architecture is fault tolerant, they are zero, and thus we minimize $Cost(\mathcal{A}^+)$. However, if the architecture is not fault tolerant, we strongly penalize the objective function with the penalty weights W_{ft} and W_s, so that the optimization will focus on searching for a fault-tolerant solution instead of minimizing $Cost(\mathcal{A}^+)$.

An essential component of the algorithm is the generation of a new solution, starting from the current one, which is called a *design transformation* or *move*. Normally, we would have to use all the fault scenarios in $\mathcal{FS}$ when doing the tests for connectivity and scheduling. However, this is computationally infeasible for large biochips that have complex fault models. Therefore, our heuristic in this paper uses only a subset $\mathcal{FS}^- \subset \mathcal{FS}$ of fault scenarios. A discussion of how to generate $\mathcal{FS}^-$ is presented in Section IV-A. By virtue of this, we are not guaranteed to create an architecture, which is fault-tolerant to all fault scenarios. However, our argument is that, by using a subset $\mathcal{FS}^-$ during synthesis, we can produce an architecture that can tolerate most of the fault scenarios. If after testing, we determine that a fault, which cannot be tolerated is present, we will discard the chip, which may reduce the yield. We evaluate the impact of the subset $\mathcal{FS}^-$ on the yield in section V.

In this paper, we use the following *design transformations*: (1) *Add redundant component*: In order to make an architecture fault-tolerant, another redundant component of that type is added to the architecture; (2) *Convert regular component to the fault-tolerant version*: By virtue of the component library $\mathcal{L}$ given in Table I, a component can be changed to its fault-tolerant version, if one exists; (3) *Add redundant channel*: A redundant channel is added, to compensate for a failing channel between the two components; (4) *Remove redundant component*: A redundant component is removed from the architecture; (5) *Convert fault-tolerant component to the regular version*: This move is the reverse of move 2; and (6) *Remove redundant channel*: A redundant channel is removed from the architecture.

A. Generation of Fault Scenarios

The fault scenarios, $\mathcal{FS}^-$, are generated from the fault model $\mathcal{Z} = (\mathcal{VF}, \mathcal{CF}, v_{max}, c_{max})$. A fault scenario $f \in \mathcal{FS}^-$ is a set of faults, containing a subset of valve faults from $\mathcal{VF}$ and a subset of channel faults from $\mathcal{CF}$, in the fault model. The fault scenarios are generated such that they are unique, i.e. $f \in \mathcal{FS}^-$ will occur once and only once. The generation of fault scenarios is divided into two phases: (1) The first phase consists of generating all the possible combinations from the set of valve faults, $\mathcal{VF}$, and the set of channel faults, $\mathcal{CF}$. Recall that v_{max} is the maximum number of valve faults happening in the architecture, and likewise c_{max} is the maximum number of channel faults. Therefore, all combinations of the set of valve faults needs to be generated, where the cardinality of each subset is $0 \leq k \leq v_{max}$; and Similarly, for the subsets of channel faults, the cardinality of each subset is $0 \leq j \leq c_{max}$. (2) In the second phase, we iterate through the list created in the first phase, and eliminate each fault scenario that is already contained by another fault scenario in the list. Then, we iteratively pick a given *Number of fault scenarios* (see the input box in Figure 6), such that the number of components and channels in the biochip $\mathcal{A}$ affected by the fault scenarios is maximized and the overlap with the already picked fault scenarios, in terms of affected components and channels, is minimized.

B. Binding and Scheduling

An architecture is fault-tolerant if it can run the application within its deadline. To determine the finishing time δ of the application, on a given architecture that is affected by a fault scenario, we use *List Scheduling (LS)* [12]. Mapping the application onto the architecture involves binding of operations onto the allocated components, scheduling the operations and performing the required fluidic routing. *LS* is a heuristic approach to solve the problem of application mapping in a computationally efficient manner. Together with the operation binding and scheduling, the heuristic approach also considers the fluidic routing and channel contention. The input to LS is the application $\mathcal{G}$, the library $\mathcal{L}$, and the netlist $\mathcal{A}^+$, which is not yet placed and routed, i.e. the physical placement of components is not known yet. We only know the exact routing latencies if we know the routes. However, the routing latencies should not be completely ignored when determining a schedule using LS. Therefore, we assume that the designer gives an average latency, which we use when determining a schedule. This may mean that the application is actually not schedulable as we could be using routing latencies, which are smaller than those resulted after the physical synthesis. However, it is still a reasonable estimation of the application completion time. In case the application is actually not schedulable, this will be known after the testing, when we have the physical architecture. If the application turns out not to be schedulable, the chip is discarded.

V. Experimental Evaluation

Our FTAS optimization strategy has been implemented in Python 3.4 and all experiments run on an Intel Xeon X5550 processor running at 2.66 GHz, with 24GB of RAM. In the first set of experiments, we were interested to determine the ability of our FTAS approach to obtain fault-tolerant architectures at a minimum cost. We have used a synthetic test case `SB1`, and two real-life test cases, `IVD` (In-Vitro-Diagnostics) [12] and `PCR` (the mixing stage of the Polymerase Chain Reaction) [12]. For each test case, we have used the tools from [12] to generate a netlist $\mathcal{A}$ (which is not fault-tolerant). Features of $\mathcal{A}$ are presented in Table III, where $|\mathcal{N}|$ is the number of components, $|\mathcal{D}|$ is the number of channels, and $Cost(\mathcal{A})$ is the cost of $\mathcal{A}$. For each test case, we have

Table III
EXPERIMENTAL RESULTS

Name	$\mathcal{A}$			$\lvert\mathcal{FS}\rvert$	$\lvert\mathcal{FS}^-\rvert$	$\mathcal{A}^+_{SFS}$			$\mathcal{A}^+_{GRASP}$		
	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{D}\rvert$	$Cost(\mathcal{A})$			$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{D}\rvert$	$Cost(\mathcal{A}^+_{SFS})$	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{D}\rvert$	$Cost(\mathcal{A}^+_{GRASP})$
SB1	15	17	84	121	100	20	27	133	15	20	102
PCR	14	16	88	77	50	18	25	135	14	17	92
IVD	52	78	274	841	100	57	92	379	52	78	279

specified a fault model $\mathcal{Z}$, based on the kinds of faults that may occur in the architecture $\mathcal{A}$. The total number of fault scenarios possible $|\mathcal{FS}|$ considering $\mathcal{Z}$, is listed in the table. For e.g., for IVD, 841 fault scenarios are possible. If we attempt to synthesize a fault tolerant architecture $\mathcal{A}^+$, without optimizing the introduction of redundancy using the straightforward solution (SFS) presented in Section IV, we obtain the results from columns $7-9$ labeled as $\mathcal{A}^+_{SFS}$. All the architectures are fault-tolerant, and we list their $Cost(\mathcal{A}^+_{SFS})$ and number of components $|\mathcal{N}|$ and channels $|\mathcal{D}|$. Using our FTAS optimization heuristic, we obtain the results from columns $10-12$ in Table III (labeled $\mathcal{A}^+_{FTAS}$). As we can see from the results, our optimization is able to significantly reduce the architecture cost $Cost(\mathcal{A}^+_{FTAS})$, compared to SFS, while at the same time synthesizing fault-tolerant implementations. As a terminating condition for FTAS, we have used a time limit of 1 hour. To evaluate the quality of GRASP implementation, we have also implemented a Simulated Annealing (SA) based metaheuristic. We have started SA from three initial solutions: SFS, GRASP and the architecture $\mathcal{A}$ without fault-tolerance. We have used a very aggressive cooling schedule for SA, which resulted in a runtime of more than 24 hours. SA was not able to improve the results of our GRASP-based FTAS implementation.

The number of fault scenarios $|\mathcal{FS}^-|$ generated in FTAS, out of the total number $|\mathcal{FS}|$ possible, is also presented in the table. The question we had in our second set of experiments was : *if a reduced number of fault scenarios are used to evaluate the fault tolerance of $\mathcal{A}^+$, will that reduce the yield significantly?* Recall from Section III that, if after testing a biochip, we find that it has faults which are not tolerable (because they are not in $\mathcal{Z}$ or because our FTAS heuristic was unable to produce a solution $\mathcal{A}^+$ that covers all fault scenarios $\mathcal{FS}$), we need to discard the biochip, which reduces the yield. Thus, for the test case $SB1$, using FTAS, we have generated several fault-tolerant architectures, as shown in Table IV. It is interesting to note that: (1) $\mathcal{A}^+_{121}$, which uses all 121 possible fault scenarios (which is infeasible for complex architectures) and (2) $\mathcal{A}^+_{25}$, which uses a subset $\mathcal{FS}^-$ containing 25 fault scenarios. $\mathcal{A}^+_{121}$, by its design, is able to tolerate any fault combination in $\mathcal{FS}$. However, although during synthesis, $\mathcal{A}^+_{25}$ uses only $\approx 20\%$ of the fault scenarios in $\mathcal{FS}$, it is able to tolerate 86.78% of the fault scenarios in $\mathcal{FS}$. This shows that our approach is able to produce fault-tolerant solutions in most cases, even if it uses only a fraction of the possible fault scenarios to evaluate an architecture $\mathcal{A}^+$, during the design space exploration.

Table IV
YIELD EVALUATION ON SB1 BENCHMARK

$\lvert\mathcal{FS}\rvert$	A_{GRASP}		$\lvert FT\rvert$	$FT\% = \frac{\lvert\mathcal{FS}\rvert}{\lvert FT\rvert} * 100$
	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{D}\rvert$		
25	16	20	105	$\frac{105}{121} \approx 86.8$
50	15	19	117	$\frac{117}{121} \approx 96.7$
85	16	21	121	$\frac{121}{121} = 100$
121	15	19	121	$\frac{121}{121} = 100$

VI. CONCLUSIONS

We have proposed an algorithm for *fault tolerant synthesis* of flow-based biochips. Evaluation has shown that our FTAS approach is able to successfully synthesize low-cost fault-tolerant architectures. Since evaluating the architecture under all possible fault scenarios is computationally expensive, only a fraction of them are generated and evaluated during synthesis. The yield evaluation proved that evaluation for even a small fraction of the possible fault scenarios, can lead to fault-tolerance on a large fraction of the possible fault scenarios.

REFERENCES

[1] J. Melin and S. Quake, "Microfluidic large-scale integration: The evolution of design rules for biological automation," *Annual Reviews in Biophysics and Biomolecular Structure*, vol. 36, pp. 213–231, 2007.

[2] D. Mark et al, "Microfluidic lab-on-a-chip platforms: requirements, characteristics and applications," *CSR*, vol. 39, pp. 1153–1182, 2010.

[3] I. E. Araci et al, "mVLSI with integrated micromechanical valves," in *Lab on Chip*, vol. 12, pp. 1463–1475, 2012.

[4] J. W. Hong et al, "Integrated nanoliter systems," *Nature Biotechnology*, vol. 21, pp. 1179–1183, 2003.

[5] J. M. Perkel, "Microfluidics - bringing new things to life science," *Science*, November 2008.

[6] P. Pop et al, "Continuous-flow biochips: Technology, physical design methods and testing," in *IEEE Design and Test of Computers*, 2015.

[7] K. Hu et al, "Testing of flow-based microfluidic biochips: Fault modeling, test generation, and experimental demonstration," in *IEEE TCAD, Vol. 33, No. 10, October 2014*, pp. 1463–1475, 2014.

[8] W. H. Grover et al, *"Monolithic Membrane Valves and Pumps" chapter in Lab-on-Chip Technology*. Caister Academic Press, 2009.

[9] F. Su et al, "Yield enhancement of reconfigurable microfluidics-based biochips using interstitial redundancy," *ACM JETC*, pp. 104–128, 2006.

[10] T. Thorsen et al, "Microfluidic large-scale integration," *Science*, vol. 298, no. 5593, pp. 580–584, 2002.

[11] I. Araci et al, "Microfluidic very large-scale integration for biochips: Technology, testing and fault-tolerant design," in *Test Symposium (ETS), 2015 20th IEEE European*, pp. 1–8, May 2015.

[12] W. H. Minhass, *System-Level Modeling and Synthesis Techniques for Flow-Based mVLSI Biochips*. PhD thesis, DTU Compute, 2012.

[13] W. H. Minhass et al, "Architectural synthesis of flow-based microfluidic large-scale integration biochips," in *CASES*, pp. 181–190, 2012.

[14] M. H. Minhass et al, "System-level modeling and synthesis of flow-based microfluidic biochips," in *CASES*, 2011.

[15] L. Wang et al, *Electronic Design Automation: Synthesis, Verification, and Test*. Systems on Silicon, Elsevier Science, 1st ed., 2009.

[16] T. F. Gonzalez, *Handbook of Approximation Algorithms and Metaheuristics*. Chapman & Hall/CRC, 2007.