Pin-count reduction for continuous flow microfluidic biochips

Schneider, Alexander; Pop, Paul; Madsen, Jan

Published in:
Microsystem Technologies

Link to article, DOI:
10.1007/s00542-017-3401-1

Publication date:
2017

Document Version
Peer reviewed version

Noname manuscript No.
(will be inserted by the editor)

Pin-Count Reduction for Continuous Flow Microfluidic Biochips

Alexander Schneider · Paul Pop · Jan Madsen

Received: date / Accepted: date

Abstract Microfluidic biochips are replacing the conventional biochemical analyzers integrating the necessary functions on-chip. We are interested in flow-based biochips, where a continuous flow of liquid is manipulated using integrated microvalves, controlled from external pressure sources via off-chip control pins. Recent research has addressed the physical design of such biochips. However, such research has so far ignored the pin-count, which rises with the increase in the number of microvalves. Given a biochip architecture and a biochemical application, we propose an algorithm for reducing the number of control pins required to run the application. The proposed algorithm has been evaluated on several biochips, including the AquaFlux biochip from Microfluidic Innovations LLC.

1 Introduction

Microfluidics refers to a technology that miniaturizes biological and chemical processes to a sub millimeter scale. A biochip, also referred to as lab-on-a-chip, integrates different biochemical functionalities such as dispensers, mixers, separators, filters and detectors on a single chip, leading to higher portability, throughput and sensitivity, while reducing sample volume consumption. Microfluidic Very Large-Scale Integration (mVLSI) enables the development of microfluidic chips using hundreds of such functions, allowing multiple assays to be run in parallel, making them usable

A. Schneider
Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark
Tel.: +45 25 30 33
E-mail: alsch@dtu.dk

P. Pop
Tel.: +45 25 37 32
E-mail: paupo@dtu.dk

J. Madsen
Tel.: +45 25 37 51
E-mail: jama@dtu.dk

for tasks such as Protein Crystallography, Amino Acid Analysis or Chemical Synthesis [9]. The key for complex functionality on biochips is the use of on-chip valves, similar to transistors in semiconductor VLSI. Such valves are manufactured using multilayer soft lithography and are controlled by outside pressure sources [14]. Using these valves the flow of fluid within the chip can be restricted, allowing to decide if and in which order the functions on the chips are used.

Ongoing research enables the fabrication of increasingly complex biochips, with an integration density advancing faster than Moore's Law [6]. For example, a commercial LoC featuring 25,000 integrated microvalves that can run 9,216 polymerase chain reactions in parallel has been available since 2008 [16]. Many biochip enabled systems are available from companies such as Fluidigm, where even fully automated components can be obtained [4]. Through these advancements, current methodologies for chip design, flow, and control synthesis become inadequate [13]. In this paper we focus on part of the control synthesis problem, namely the control pin reduction problem. A control pin is a physical hole in the chip, which provides access to the control layer. By applying pressure (or vacuum, depending on the valve type) [19] to the control pin, a valve on the chip can be activated. However, a large number of control pins is infeasible, due to the resulting consumption of chip area and required bulky off-chip control. The "AssayMark" controller from Microfluidic Innovations LLC for example, is only capable of providing pressure to up to 36 individual control pins, using solenoid valves [11]. By allowing valve sharing, i.e. the sharing of the same control by multiple valves, it is thus possible to reduce the required amount of control pins to meet these restrictions. Doing so reduces the flexibility of the chip, since the shared valves will work in unison. Valves sharing a single control pin therefore have to be chosen carefully, in order to keep the chip operable [14].

Researchers have previously proposed approaches to the application mapping and scheduling [12]. Based on this schedule of operations, the control information (which valves to open and close at what time and for how long) can be extracted. Using this information, optimization schemes can be applied to minimize the chip's pin-count in the control layer. Recent research has proposed approaches to control pin minimization following this idea [14] [18]. However, the ordering in which scheduling and pin-count reduction is executed is crucial, since the execution of either process poses constraints onto the other. Combining valve controls and therefore having valves work in unison imposes scheduling constraints. Conversely, if the schedule has been fixed and operations are to be executed at the same time, valve control combinations are restricted to valves which are never in opposing states. Reducing the pin-count beforehand would therefore allow to trade off flexibility during scheduling for additional combinations.

Contribution: We propose a new pin-count reduction technique which is applied before the application is scheduled. Without considering the scheduling constraints, we can find additional combinations for controls. However, when later performing scheduling, this will potentially increase the application completion time. Our approach offers the possibility for a direct trade-off between the pin-count and the applications completion time. To improve our pin-count reduction approach, we also

use a new routing technique which prioritizes routes that maximize the potential for control combinations.

2 Biochip Architecture Model

Fig. 1 shows a functional view of a flow-based biochip containing multiple **Functional Fluid Units (FFU)** such as inputs and heaters, as well as switches connecting the channels leading to these components. The basic building block of these components is a microvalve, which can be used to manipulate the fluid flow.

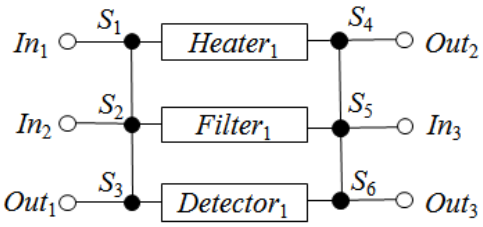


Fig. 1: Architecture model example

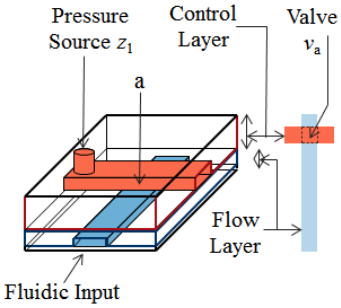
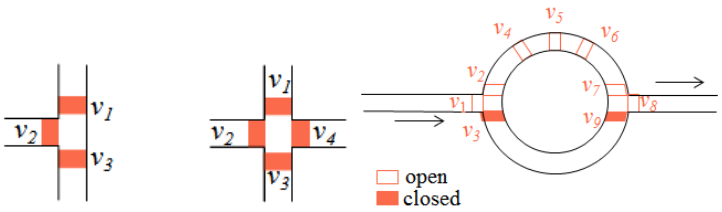


Fig. 2: Micromechanical valve

Fig. 2 shows a micromechanical valve which can be manipulated by an external force to either restrict or permit the fluid flow. To do so, the chip is logically divided into two layers, the flow-layer (colored in blue in Fig. 2) which contains the fluid and the control-layer (colored in red) which can manipulate the flow-layer. Such valves are fabricated using a layer of PDMS for each of the two logical layers. The PDMS is poured onto the control- and flow-mold respectively and then baked. Afterwards the layers are peeled off, aligned and bonded together by further baking, see [2] [8] for more details on the fabrication process. The pressure source (z_1 in Fig. 2) is connected

to the control channel via a control pin. A control pin is a physical hole, typically significantly larger in diameter than the control channel it is connected to, located at the edges of a biochip. When pressure is applied, the elastic control layer will "pinch" the flow layer (at point *a* in Fig. 2), blocking the flow of fluid (closed valve). If no pressure is applied, the fluid can flow freely through the flow layer (open valve). Hence, this is called a "normally-open valve". Normally-closed valves can be fabricated as well and information about their functionality and fabrication is available in [5]; our work can be used with any microvalve technology. To create functional components, multiple valves are needed. Fig. 3a shows two variations of switches, requiring three and four valves respectively to control the path of the fluid entering from any side. Mixers as shown in Fig. 3b require nine valves to be operational. v_2 and v_7 as well as v_3 and v_9 are used to close of one half of the mixer to allow the other half to be filled. Valves v_1 and v_8 close off the mixer during the mixing process, which is indicated by valves v_4, v_5 and v_6, acting as a pneumatic pump. Other components such as filters, heaters or detectors only require two valves to close off the component during its execution, similar to valve 1 and 8 in the mixer [1].

As mentioned, each valve is in one of two physical states. The *open* state (denoted with "0") allows for fluid to be transport through a channel or component and the *closed* state (denoted with "1") prevents fluid from leaking into other channels or components. Additionally, we also distinguish a third, logical state we call *don't care* (denoted with "X"). While valves in the *open* or *closed* state affect the fluid transport by allowing or restricting the flow respectively, valves in the *don't care* state have no effect on the fluid transport at all, making the physical state they are in irrelevant. Consider the mixer in Fig. 3b, to fill the bottom half, valves 1, 3, 8 and 9 have to be open to allow fluid transport, while valves 2 and 7 have to be closed to prevent leakage into the top half. The state of valves 4, 5, and 6 however is irrelevant, since the channel they are located in cannot be reached by any fluid when valves 2 and 7 are closed, placing them in the *don't care* state.



(a) Switches (b) Mixer

Fig. 3: Microfluidic components

2.1 Architecture Model

In this paper we use the system-level architecture model based on a topology graph, as proposed in [12]. Fig. 1 shows an example of a biochip model, containing three inputs and outputs, one filter, one heater and one detector. We distinguish between two kinds of vertices in this model: Switches, which create intersections between multiple channels (e.g. S_1) and FFUs which can perform a certain action (e.g. In_1, $Heater_1$, $Filter_1$). Edges represent channels through which fluid can be transported and are considered bi-directional throughout this paper. Fluid can be transported through these components by applying pressure to an input from an outside source. A functional route must therefore always start at an input where pressure is applied and end in an output where the pressure is released from the chip. For all following examples, we assume that an input is capable of providing pressure, as well as being used as a source of reagents.

To express routes on an architecture, we use the following definitions. A **Flow Path Segment (FPS)** is a set of vertices, forming a directed route from one FFU to another. To connect the two FFUs, a FPS can contain any number of switches, but no additional FFUs, e.g. (In_1, S_1, S_2, $Filter_1$). FPSs are mutually exclusive, meaning they cannot be used to transport fluids at the same time, if they share at least one vertex. A **Flow Path (FP)** is a set of FPSs which form a route usable for fluid transport within the given architecture. The first FPS of such a set has to start with an input, since

$FPS_1 = (In_1, S_1, Heater_1)$
$FPS_2 = (In_1, S_1, S_2, Filter_1)$
$FPS_3 = (In_1, S_1, S_2, S_3, Detector_1)$
$FPS_4 = (In_2, S_2, S_1, Heater_1)$
$FPS_5 = (In_2, S_2, Filter_1)$
$FPS_6 = (In_2, S_2, S_3, Detector_1)$
$FPS_7 = (In_3, S_5, S_4, Heater_1)$
$FPS_8 = (In_3, S_5, Filter_1)$
$FPS_9 = (In_3, S_5, S_6, Detector_1)$
$FPS_{10} = (Heater_1, S_1, S_2, S_3, Out_1)$
$FPS_{11} = (Heater_1, S_1, S_2, Filter_1)$
$FPS_{12} = (Heater_1, S_1, S_2, S_3, Detector_1)$
$FPS_{13} = (Heater_1, S_4, Out_2)$
$FPS_{14} = (Heater_1, S_4, S_5, Filter_1)$
$FPS_{15} = (Heater_1, S_4, S_5, S_6, Out_3)$
$FPS_{16} = (Heater_1, S_4, S_5, S_6, Detector_1)$
$FPS_{17} = (Detector_1, S_3, Out_1)$
$FPS_{18} = (Detector_1, S_3, S_2, Filter_1)$
$FPS_{19} = (Detector_1, S_3, S_2, S_1, Heater_1)$
$FPS_{20} = (Detector_1, S_6, Out_3)$
$FPS_{21} = (Detector_1, S_6, S_5, S_4, Out_2)$
$FPS_{22} = (Filter_1, S_2, S_3, Out_1)$
$FPS_{23} = (Filter_1, S_2, S_3, Detector_1)$
$FPS_{24} = (Filter_1, S_2, S_1, Heater_1)$
$FPS_{25} = (Filter_1, S_5, S_4, Out_2)$
$FPS_{26} = (Filter_1, S_5, S_6, Out_3)$
$FPS_{27} = (Filter_1, S_5, S_6, Detector_1)$

Table 1: All available FPSs for the architecture in Fig. 1

they act as a pressure source. All following FPSs have to start where the previous FPS ended. A FPS ending in an output finishes the FP which then forms a continuous route from an input to an output.

Table 1 shows the resulting FPSs determined by paring all available FFUs shown in Fig. 1 that are directly connected to each other. Note that this list is already stripped of such FPSs, that cannot be used to create FPs in the architecture, e.g. (Detector_1, S_6, S_5, Filter_1) since it is impossible to connect an input to Detector_1 and an output to Filter_1, without overlapping paths.

Previous research [12] has considered a simplified routing model, which assumes that fluids can be moved along all given FPSs, regardless of the presence of inputs or outputs. This only works under the assumption of implicit inputs and outputs, which would be located in front of and behind FFUs, allowing FPSs to be functional routes on their own, e.g. FPS_{12} would be a valid FP to transport fluid from Heater_1 to Detector_1, even though no input or output is connected. We refer to these additional interfaces as implicit, since they are not modelled in the architecture as opposed to explicit ports, which are part of the model. This results in invalid schedules, unless all additionally assumed inputs and outputs are added to the architecture, which may be infeasible due to the increase in valve- and control counts as well as the chip size. In this paper we consider a realistic routing model, where every route starts at an input and ends at an output explicitly stated by the architecture model.

2.2 Application Model

To model a biochemical application, we use a directed, acyclic and polar sequencing graph model [12]. A node in this graph is an operation that runs on a FFU. The edges denote dependencies between operations that require fluid transport. Each operation O_i has an execution time C_i when running on a FFU_j. We also model the fluid transport times depending on the length of the channel. An example of such an application graph can be seen in Fig. 4. For the examples in this paper, we will assume, for simplicity, that an operation takes 4 time units and the transport through any given channel takes 1 time unit. Application models are not architecture specific. Therefore, before an application can be run on a biochip, the application has to be mapped

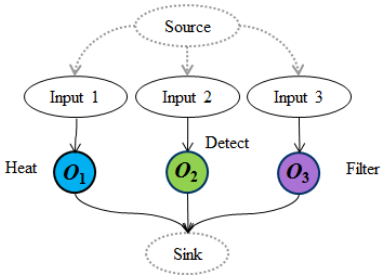


Fig. 4: Application model example

to the chip. This includes binding operations to the functional units (where the operation should be executed), the routing of fluids (which channels should be used for transport) and scheduling (at which times the fluids are transported and the operations are executed). Typically that also includes the assignment of dispensing operations to input reservoirs, however in Fig. 4 we have already done this assignment to simplify the examples in which this application is used.

3 Problem Formulation

Let us illustrate our problem using the In-Vitro Diagnostics application (IVD) from Fig. 6 that has to be mapped to the architecture from Fig. 5. We indicate with O_i the i-*th* operation in the IVD. Let us assume that the binding of operations is as follows: O_1, O_2 and O_5 are bound to $Mixer_1$; O_6, O_9 and O_{10} are bound to $Mixer_2$; O_3, O_4 and O_7 are bound to $Detector_1$ and O_8, O_{11} and O_{12} are bound to $Detector_2$. Furthermore, let us assume that we schedule this application such that the restulting schedule is the one shown in Fig. 7a leading to the valve actuation sequence shown in Table 2. The schedule shows the start times and the duration of operations as references on a timeline, and the four rows correspond to the FFUs in the architecture from Fig. 1. We denote with M_i the fluid transport operations. The time units are listed at the bottom, indicating the elapsed time since the start of the application. Note that for space reasons we only show partial schedules in Fig. 7 and a partical valve actuation table in Table 2. The rows of the table represent the valves and each column represents a time step in which certain valves need to be actuated. As mentioned, related work [14] performs control pin minimization after scheduling, thus using the data from Table 2 as a starting point. There, $Valve_1$ and $Valve_3$ for example, can be in the same state throughout the depicted time steps (at time step 18, $Valve_3$ is in a *don't care* state and can be opened to suit $Valve_1$) and their controls can therefore be combined. The *don't care* state is present whenever the part of the chip where the valve is located is unused, e.g. when transporting fluid from In_3 to $Mixer_1$ in Fig. 5, the part of the chip containing $Detector_2$ is unused. This means the detector's valves are in the *don't care* state, since they neither have to restrict nor permit fluid flow and their physical states are therefore irrelevant.

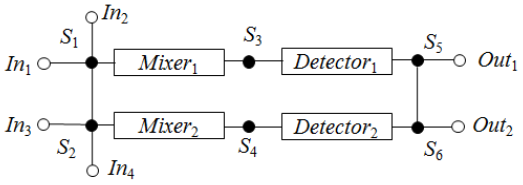


Fig. 5: IVD Architecture

Note that in the IVD schedule in Fig. 7a, the detection operations O_3 and O_4 are waiting for the mixing operations O_1 and O_2 to finish respectively. The fluid transport

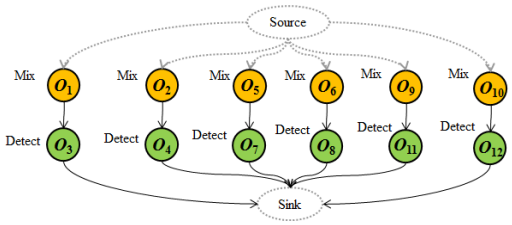
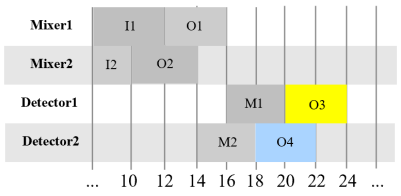
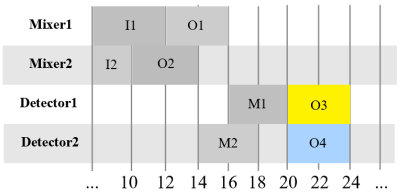


Fig. 6: Typical IVD application that mixes various samples, reagents and buffers and analyses the results.

M_2 and the following detector operation O_4 are started as soon as possible, in order to minimize the application completion time. This however means, that the equivalent operations for Detector_1 (M_1, O_1) are not executed at the same time as for Detector_2. This leads to the valves used by these four operations to be in opposing states in time step 18 as highlighted with the corresponding color coding in Table 2 and Fig. 7a, meaning their controls can't be combined.



(a) Partial IVD schedule before the pin-count reduction, all operations are executed as early as possible.



(b) Partial IVD schedule after pin-count reduction with scheduling constraints in place, leading to a deferred execution of O_4

Fig. 7: Impact of pin-count reduction on IVD schedule

Valve No. / TS	8	10	12	14	16	18	20	22
1	0	0	0	0	0	0	X	X
2	1	1	0	0	1	1	X	X
3	0	0	0	0	0	X	X	X
4	1	0	0	1	1	X	X	X
5	0	0	1	1	0	0	X	X
6	0	1	1	0	0	X	X	X
7	0	0	X	X	0	0	1	1
8	0	X	X	0	0	1	1	X
...	...	...	...	...	...	...	...	...
8A	0	X	X	0	0	X	1	1

Table 2: Partial actuation Table for IVD. 0: Open. 1: Closed. X: Don't Care

As we can see from this example, the scheduling step has introduced constraints that restrict the valve sharing options. Let us assume that we have decided to combine the valves used by Detector_1 and Detector_2 before scheduling the operations. This means that the two detectors can only operate simultaneously, which is a scheduling constraint. We can enforce this constraint during scheduling, for example by postponing the execution of operation O_4 until operation O_3 can be executed as well, the resulting schedule is shown in Fig. 7b. The resulting change in the control logic for Valve_8 is stated as Valve_{8A} in Table 2, which is now compatible with Valve_7. However, the introduced change in the schedule can affect the application execution time, as Detector_2 is now occupied by O_4 until time step 24 instead of time step 22, potentially postponing other operations by 2 time steps as well.

By determining the valve sharing before the scheduling step, we can reduce the number of control pins required. For the complete IVD application, we can reduce the pin-count from 20, as determined by the technique from [14] to 14. The so introduced scheduling constraints do however delay the operations, increase the application completion time and possibly exceed the application deadline, which has to be satisfied for certain biochemical protocols. For this example the reduction of required control pins to 14 increases the application execution time from 81 (as determined before the reduction) to 107 time steps. In addition, we have to make sure that the application can still be executed successfully as control pin combinations can force valves to be in states which are incompatible with certain routes, which is explained in detail in Sect. 4.2.

The problem we address in this paper is defined as follows:

Input: Topology-graph architecture model, application model including application deadline and available number of control pins.

Determine: The sharing of control pins by valves in the architecture, the binding, routing and scheduling of operations in the application, such that the number of control pins required does not exceed the given limit and the application deadline is satisfied.

4 Proposed Method

Researchers have proposed a design flow for mVLSI biochips [17], which is shown in Fig. 8. To solve the problem outlined in the previous section we propose the following steps, which are also shown in Fig. 8 as a detailed view of the “Application Mapping” and “Control Synthesis”.

In step 1, we bind and route the operations given by the application onto to the given architecture model, using a novel routing technique presented in Sect. 4.1. Step 2 performs a preliminary control synthesis which determines the valve states for every created route. Contrary to the complete control synthesis [14], this version does not yet contain any information about timing. The determined valve states are then used as input along with the application deadline for our proposed pin-count reduction algorithm from Sect. 4.2. The algorithm produces a grouping of valves that can share the same control pin, which introduces scheduling constraints for the next step. Since scheduling the whole application is time consuming, step 4 uses a

prediction function presented in Sect. 4.2 to determine the impact of the constraints introduced on the schedule. The application is scheduled in step 5. Step 6 concludes this process with control synthesis, which in contrast to the one performed in step 2, does also incorporate the schedule. To schedule the application we have implemented a List-Scheduling algorithm [10] and extended it to take scheduling constraints into account. After the control synthesis is performed, we know how many control pins are required and how they have to be connected to valves. The "Physical Design" task for the control layer is responsible to determine the routing of the control channels in the architecture [7]. Note that our approach is iterative (the back arrow from step 4 to step 3) and allows the designer to perform a trade-off between the number of control pins and the application completion time. If the end-user already has a control box with a given number of outputs, then this number can be given as an input, and our algorithm in step 3 will stop when this is satisfied. Our algorithm also stops before the application deadline (if given) would be exceeded.

4.1 Routing

Given the architecture and application models (see Sects. 2.1 and 2.2 respectively), the routing algorithm (step 1 in Fig. 8) determines a route for every operation. A route is constructed from a FP. We use the term "route" to refer to those FPs which have been chosen to be used by an operation, distinguishing from other available, FPs not used as routes. Binding, meaning which FFU in the target architecture is used to execute an operation (e.g. heating) is also determined during routing.

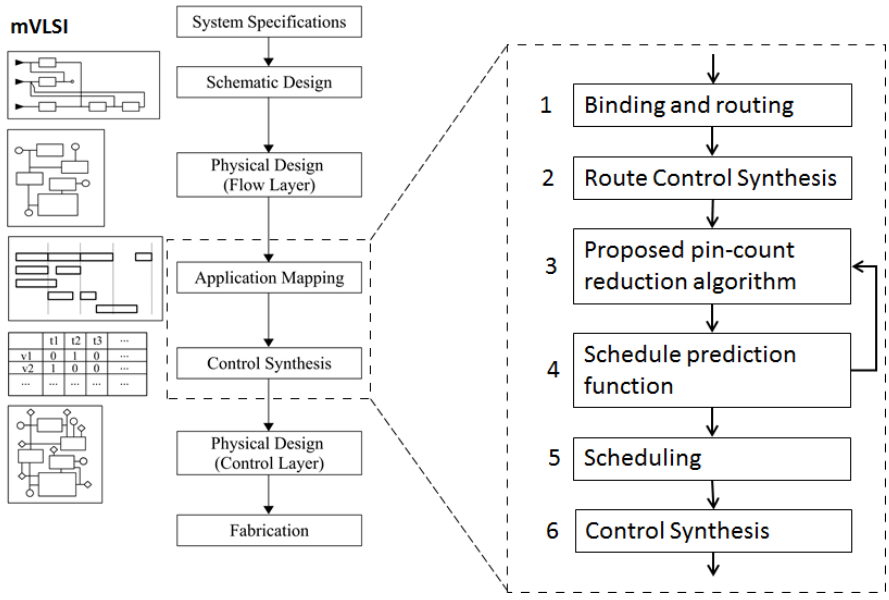
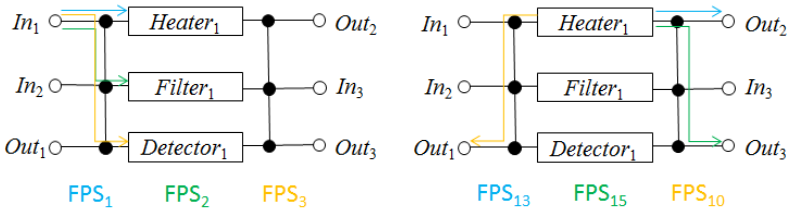


Fig. 8: mVSLI design cycles and our proposed method



(a) Available FPSs from In_1 (b) Available FPSs from $Heater_1$ to outputs

Fig. 9: Microfluidic components

We construct a route by linking together multiple FPSs which are determined first, as explained in Sect. 2.1. From this list of FPSs (which are stated in Table 1 for our example) we can determine FPs to suit the operations of the application. We divide this process into three steps. (1) A connection between the FFU where the fluid is currently located and the target FFU has to be found. For example, operation O_1 from Fig. 4 requires fluid from In_1 to be transported to a heater. O_1 can execute on any heater, and our routing will search for a heater FFU to bin operation O_1 to. We determine a path between these FFUs using one, or a set of connected FPSs. This is done using a Breadth-First Search algorithm, with all FPSs beginning at In_1 as starting points. Additional FPSs are linked to the established FPSs until a connection between the FFUs has been found. Fig. 9a shows all FPSs starting at In_1. Since FPS_1 directly connects In_1 to $Heater_1$, it is chosen as the route for operation O_1 while the other options are discarded. Two additional steps are necessary to form a valid FP. (2) The path determined in (1) has to be connected to an output to be considered a valid FP. To the heating operation O_1 we bound $Heater_1$ and therefore need to connect to an output from there. This is done using the same Breadth-First Search algorithm as in (1), only with $Heater_1$ as the starting point and any output as target. However, during this search conflicts with the previously determined path from In_1 to $Heater_1$ have to be omitted. Considering all available FPSs that lead to outputs as shown in Fig. 9b FPS_{10} cannot be used to connect $Heater_1$ with Out_1, since this FPS overlaps with FPS_1 which is already in use. For operation O_1, we therefore use FPS_{13} to connect $Heater_1$ to Out_2 without causing any overlap. (3) The path determined in (1) has to be connected to an input to be considered a valid FP. For operation O_1 this is already the case and no further action is required to complete the route. For operations that do not start at an input, an input has to be found in the same fashion as an output in (2). Depending on the architecture, it is possible that multiple paths connecting the source and target FFU exist or multiple components of the same type to which an operation can be bound (e.g. Heater, Detector) are available, meaning we will end up with multiple options for all three routing steps, e.g. in Fig. 9b we could have also used FPS_{15} to connect $Heater_1$ with Out_3. However, our strategy is to choose the route which blocks the smallest number of FFUs and switches on the chip, which is not necessarily the shortest route. A component is considered blocked if it is actively used (i.e. routed through), or if it cannot be used by another route, since an adjacent component is used. Take FPS_4 in Table 1 for example. Even though it does

not use Filter_1, the valve leading towards it has to be closed and the filter is considered blocked, since it is not possible to form a route that uses Filter_1, while FPS_4 is active. Choosing routes with the lowest number of blocked components therefore allows a higher flexibility while scheduling, since more components are still available to be used by another route.

4.2 Reducing the Pin-Count

Using the previously defined routes as input we can determine controls that can be combined to lower the chip's pin-count. As mentioned previously, combining controls without having a schedule in place provides additional possibilities for combinations, but also has the potential to increase the application execution time. This is due to scheduling constraints that are introduced when combining controls. The reason for the increase in execution time is the same for all constraints: FPs that were previously able to be executed in parallel, now have to be executed in sequence to avoid unintentional mixing. The amount of additionally required execution time however can vary, depending on the length of the FPs and the number of FPs that have to be executed in sequence because of the introduced constraint. Cases of such constraints will be shown using examples from Table 4.
Consider Fig. 1 as an example architecture. To operate the chip, 32 valves are required. A partial example of where those valves are located in our architecture is shown in Fig. 10.

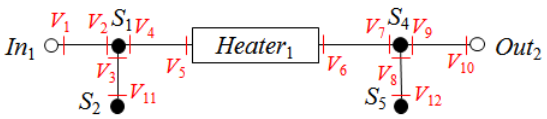


Fig. 10: Valves

The first set of combinations performed differs from other combinations, as they will never introduce any constraints onto the schedule. Those combinations can be made, regardless of the architecture, for all valves which are located in a channel between two switches, an input and a switch, or an output and a switch. Consider In_1 and S_1 in Fig. 10. They are connected by a channel which contains Valve_1 and Valve_2. Only two possibilities exist in which this channel can be used: Either the channel is routed through, in which case both valves have to be opened, or it is not routed through, in which case both valves can be closed. Therefore, the valves have no need to be actuated individually. The same principle applies to the valves which close off any given FFU, along with other valves that are located in this channel (e.g. valves 4 to 7 in Fig. 10). Therefore, these valves' controls can be combined and share a control pin, meaning that these valves will always work in unison, in what we call a **Valve Group (VG)**. Fig. 11 shows the result of applying this to the presented architecture, where the previously mentioned Valve_1 and Valve_2 now form VG_1, valves 4 to 7 form VG_2

Table 3: Valve Group Configuration

Route	Open VGs	Closed VGs	Don't Care VGs
1	1, 2, 3	4, 5	6, 7, 8, 9, 10, 11, 12, 13
2	6, 9, 12, 13	4, 7, 10, 11	1, 2, 3, 5, 8
3	7, 8, 9, 11	4, 5, 6, 10, 12	1, 2, 3, 13

and so on. Further combination of VGs means that all valves of both groups share a single control pin and are therefore activated in unison.

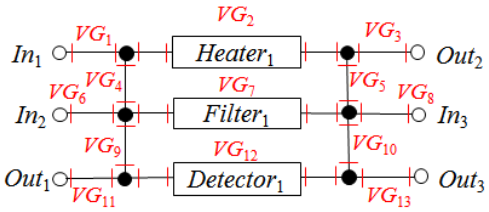


Fig. 11: Valve Groups

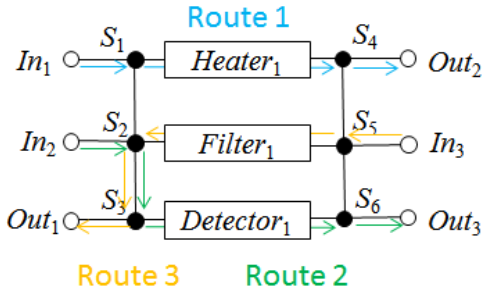


Fig. 12: Routes

Additional combinations will introduce scheduling constraints, which may affect the application execution time negatively. To ensure that the application deadline will not be exceeded, it is therefore necessary to determine combinations that introduce constraints with minimal effect on the schedule. To clarify the effects of such constraints we map the application from Fig. 4 to our example architecture in Fig. 1. The three resulting routes can be seen in Fig. 12.From these routes we can extract the valve states for each route, which is illustrated in Table 3. Each route has sets of VGs in the previously explained states *open*, *closed* and *don't care*.
Given this data, the application graph, the desired number of control pins and (if relevant) the maximum application execution time, Algorithm 1 can determine the

Example	Combined VGs	Result	Reason
1	VG_1 and VG_4	Combination not valid	Route 1 is invalidated, since VG_1 has to be open to allow fluid transport while VG_4 has to be closed to prevent leakage
2	VG_1 and VG_2	Combination valid and no increase in schedule length	VG_1 and VG_2 are *open* for Route 1 and in a *don't care* state for Routes 2 and 3. Therefore no matter which routes are executed in parallel, VG_1 and VG_2 can always be open
3	VG_6 and VG_{13}	Combination valid and no increase in schedule length	Even though this combination does prohibit routes 2 and 3 from being executed in parallel, no increase in schedule length will occur since these routes were not able to run in parallel in the first place, due to their partially overlapping FPs
4	VG_1 and VG_{12}	Combination valid and possible increase in schedule length	VG_1 has to be open for Route 1, while VG_{12} has to be closed for Route 3, prohibiting parallel execution of these routes
5	VG_1 and VG_{10}	Combination valid and possible increase in schedule length	VG_1 has to be open for Route 1, while VG_{10} has to be closed for routes 2 and 3, prohibiting parallel execution of these routes

Example	Constraint	Result	Exec. time
6	Routes 1 and 2 restricted from being executed at the same time	Routes 1 and 3 are executed in parallel Route 2 is executed as soon as Route 1 finishes	18
7	Routes 1 and 3 restricted from being executed at the same time	Routes 1 and 2 are executed in parallel Route 3 is executed as soon as Route 1 finishes	16

Table 4: Valve Group combination examples: Examples 1-5 show the outcome of combining certain VGs regarding the architecture and routes from Fig. 12. Examples 6 and 7 show how different constraints can have varying effects on the schedule length. For these examples, 10, 8 and 6 time units are assumed for routes 1, 2 and 3 respectively.

potential effect on the schedule for each combination of VGs and hence choose the appropriate combinations that have the minimum impact on the application execution time.

Algorithm 1 starts with the assumption that all VGs can be combined, hence all permutations of combinations are iterated through in line 1. However, several combinations can be discarded right away as they would invalidate at least one of the given routes, meaning the application could no longer be correctly executed. This is the case whenever some of the valves in questions are in the *open* state, while another part is in the *closed* state for a single route, as shown in Example 1 in Table 4 and checked in line 2 in the algorithm. If this is not the case, the algorithm continues in line 4, determining whether this combination introduces a scheduling constraint. This is similar to how valve groups are determined in the related work in [14]. However, the related work only determines whether the combination is valid for an already given schedule. We determine if the combination is valid for **any** possible schedule, meaning no scheduling constraint is introduced. This is the case if all potentially combined valves can be in the same state for all routes, i.e. all valves are either in the *open* or *don't care*

Algorithm 1 Pin-count reduction

```
 1: for each possible combination of VGs comb do
 2:     if valves in comb have to be in opposite states for at least one route then
 3:         Combination not possible without invalidating such routes
 4:     else if all valves in comb are in the same state at any given time then
 5:         Combination valid and no increase in schedule length
 6:     else
 7:         for all possible pairings of given routes do
 8:             if combining the valves in comb can increase the schedule length then
 9:                 Use prediction function to determine by how much
10:             end if
11:             Store the prediction for this pair of routes, or 0 if no prediction was necessary
12:         end for
13:     end if
14: end for
15: Sort all combs according to the predicted increase in execution time
16: Combine VGs according to the sorted list until the application deadline is exceeded
```

state, or in the *closed* or *don't care* state, for all routes, as it is the case in Example 2. If a combination of valves does introduce a scheduling constraint, the algorithm continues on to determine the impact on the application execution time for this constraint in lines 7-12. As mentioned previously, a scheduling constraint prohibits two or more routes to be executed in parallel, because parallel execution would cause unintentional mixing of fluids, leading to a potential increase in application execution time as these routes have to be executed in sequence. Hence we iterate through all pairs of routes in line 7 in order to determine the effect on them. First, some routes were never able to be executed in parallel anyway, since they partially overlap, as demonstrated in Example 3. In this case, which is checked in line 8, the scheduling constraint does not have any effect on this pair of routes. In other cases such as in Example 4, the scheduling constraint does impact the parallelism of the application as routes 1 and 3 can no longer be executed at the same time. When such a case arises, the algorithm determines how much this scheduling constraint affects the application execution time in line 9. Exact results can however only be determined by scheduling the application and comparing the execution times. Since this is too time consuming, we propose a cost function to predict the effect on the schedule. This prediction is based on how many parallel executions of routes are prohibited by a scheduling constraint. E.g., Example 5 prohibits route 1 from being executed at the same time as either route 2 or 3. Example 4 on the other hand only prohibits parallel execution of routes 1 and 3, leaving the possibility to execute routes 1 and 2 at the same time, leading to a prediction of a smaller increase in execution time than for Example 5. Additionally the execution time of each route (how long it takes to transport the fluid along the FP) is taken into account. The cost function predicts that constraints affecting routes with long execution times have a larger impact on the schedule. This can be seen in Examples 6 and 7 where we assume that routes 1-3 have an execution time of 10, 8 and 6 time steps respectively. Both examples prohibit one pair of routes from parallel execution, yet Example 7 results in a shorter execution time, since the longest routes are not affected by constraints.

Once the impact on the application execution time has been predicted for all combinations of valves, these combinations are sorted from smallest to largest impact in line 15 and then applied to the architecture until a stopping criteria is reached. The end user can specify the stopping criteria: Stop the pin-count reduction when a given application deadline is reached, or stop when a given target number of control pins is reached. By varying the stopping criteria, an end-user can trade-off the number of control pins and the application completion times, as discussed in the experimental results.

5 Experimental Results

We were interested to evaluate the proposed Pin-Count Minimization (PCM) strategy. The evaluation has been performed for 4 biochips with the corresponding biochemical application. We have compared our PCM approach with the Control Synthesis Optimization (CSO) from [14], which performs pin-count reduction after the scheduling step, using a Graph-Coloring algorithm. We have used the following test cases for the evaluation:
Test Case 1 (TC1): We use an architecture capable of IVD (In-Vitro Diagnostics), which has various real life applications [3].
TC2: We have used the processing part of the MOA (Mars Organic Analyser) [15], and for TC3 we have modified it to an alternate design for the same application.
TC4: We have created a Multi-Purpose (MP) architecture to test the effects of our pin-count reduction technique. The corresponding architecture- and application model for our MP-Architecture can be seen in Figs. 13 and 14. The number of FFUs and valves in the architectures as well as the number of operations in the applications are given in columns 2-4 in Table 5. The architecture and application models are available as supplemental materials.

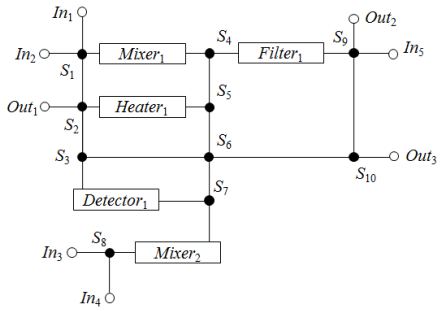


Fig. 13: Multi-purpose architecture

For all experiments we assume that it takes 1 time step for fluid to pass through any given channel (e.g. from one switch to another) and 4 time steps for an operation (e.g. mixing) to finish. Table 5 shows a direct comparison between the CSO presented

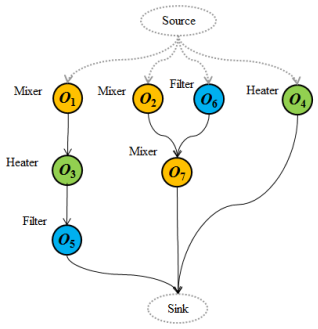


Fig. 14: Multi-purpose architecture application

in [14] and our PCM. The table contains the number of valves and FFUs in the architectures, the number of controls required and the corresponding execution times for both methods, for each of the tested architectures. As expected, the pin-count reduction comes at the expense of an increase in application completion time. However, for most cases, the minimum number of pins is obtained without a significant increase in completion time. The advantage of our approach is that it allows for a direct, stepwise trade-off between the number of control pins and the application completion time.

Hence, in our second experiment, we were interested to evaluate the ability of our proposed PCM approach to support such a trade-off. Thus using TC4 (TC1-3 are available in the supplemental material) we show in Fig. 15 how PCM enables the trade-off between the number of required control pins and the application completion time. The figure shows the number of control pins, starting at 31, which is the initial number of VGs created, on the Y-axis and the completion time on the X-axis. As we can see from the figure, the end-user can choose which trade-off is most appropriate, considering the constraints of their control solution in terms of number of control pins available versus the application completion time, which may be constrained by a deadline for certain applications. Such a trade-off is non-trivial, since the impact of a certain valve-sharing solution on the schedule is not easy to determine. For example, there are combinations which do not affect the schedule length, for reasons as shown in Examples 2 and 3 in Table 4. This is however only true for this particular order in which the combinations have been made. E.g., the combinations that reduced the control count from 24 to 16 do not generally have no effect on the schedule length. Instead, the scheduling constraints introduced by the combinations before, create a scenario similar to Example 3 for the following combinations. Especially for applications with few operations, such large jumps can occur frequently after multiple constraints have already been introduced. This is due to the fact that most of these few operations are already executed in sequence after a small number of constraints is applied, providing more options for combinations such as Example 3 in Table 4, which constrain the schedule no further. Additional combinations bring the control pin count to 14, which is the minimum number of control pins required to run the application as determined by the pin-count reduction algorithm. We have also validated

Test Case			PCM		CSO [14]	
Name	FFUs	Valves	Pin count	Completion time	Pin count	Completion time
TC1	8	46	14 (30%)	107 (32%)	20	81
TC2	13	84	17 (14%)	119 (0%)	21	119
TC3	15	74	14 (30%)	96 (13%)	20	85
TC4	13	66	14 (44%)	70 (56%)	25	45

Table 5: Comparison between PCM and CSO. The percentage values in PCM indicate the reduction of the pin count and the increase in completion time compared to CSO respectively.

our method on the AquaFlux biochip controlled by a 36-controls box from Microfluidic Innovations, LLC. Our method was able to determine the same pin-count as the one currently used by the manually designed AquaFlux biochip.

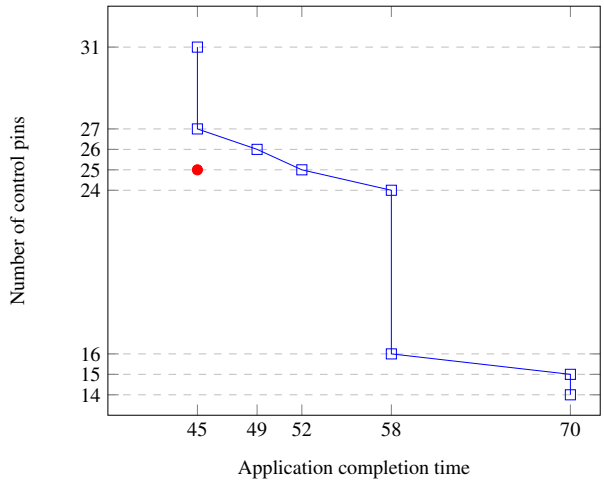


Fig. 15: Trade-off for TC4 using PCM (blue, squares and line). CSO result reference for TC4 (red dot)

6 Conclusions

With new advances in biochip fabrication, the number of valves integrated on a single chip is rising fast. The external hardware required to operate a biochip is however not as scalable, resulting in further increase in size and cost of this hardware, which is already magnitudes larger and more expensive than a biochip. In this paper we have proposed a new technique to reduce the required pin-count for flow-based biochips, effectively reducing the complexity of external hardware required. Contrary to previous work, this method is able to trade off execution time to reduce the pin-count even

further. Experimental results have shown that our algorithm is capable of reducing the pin-count significantly, while keeping the increase in schedule length acceptable. To produce realistic results, the current state-of-the-art routing model has been extended.

References

1. H.-P. Chou, M. A. Unger, and S. R. Quake. A microfabricated rotary pump. *Biomedical Microdevices*, 3(4):323–330, 2001.
2. O. J. S. David C. Duffy, J. Cooper McDonald and G. Whitesides. Rapid prototyping of microfluidic systems in poly(dimethylsiloxane). *Analytical Chemistry*, 70(23):4974–4984, 1998.
3. C. K. Dixit. Biochips based in vitro diagnostics: Market trends and research. *Journal of Biochips & Tissue Chips*, 3(2), 2013.
4. Fluidigm c1 system. `https://www.fluidigm.com/products/c1-system`. Accessed: 2017-02-27.
5. W. H. Grover, A. M. Skelley, C. N. Liu, E. T. Lagally, and R. A. Mathies. Monolithic membrane valves and diaphragm pumps for practical large-scale integration into glass microfluidic devices. *Sensors and Actuators B: Chemical*, 89(3):315 – 323, 2003.
6. J. W. Hong and S. R. Quake. Integrated nanoliter systems. *Nature Biotechnology*, 21:1179–1183, 2003.
7. M. Hrslev-Petersen. Computer-aided design for the physical synthesis of biochip control logic. 2016. Master's Thesis, Technical University of Denmark.
8. T. T. A. S. Marc A. Unger, Hou-Pu Chou and S. Quake. Monolithic microfabricated valves and pumps by multilayer soft lithography. *Science*, 288(7):113–116, 2000.
9. J. Melin and S. R. Quake. Microfluidic large-scale integration: The evolution of design rules for biological automation. *Annual Review of Biophysics and Biomolecular Structure*, 36(1):213–231, 2007. PMID: 17269901.
10. G. D. Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1st edition, 1994.
11. Microfluidic innovations llc assaymark. `http://www.microfluidicinnovations.com/Datasheets/AssayMark_DS1301.pdf`. Accessed: 2016-09-01.
12. W. Minhass, P. Pop, and J. Madsen. *System-Level Modeling and Synthesis Techniques for Flow-Based Microfluidic Very Large Scale Integration Biochips*. PhD thesis, Technical University of Denmark, 2012.
13. W. H. Minhass, P. Pop, and J. Madsen. System-level modeling and synthesis of flow-based microfluidic biochips. In *Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '11, pages 225–234, New York, NY, USA, 2011. ACM.
14. W. H. Minhass, P. Pop, J. Madsen, and T.-Y. Ho. Control synthesis for the flow-based microfluidic large-scale integration biochips. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pages 205–212, Jan 2013.
15. M. F. Mora, F. Greer, A. M. Stockton, S. Bryant, and P. A. Willis. Toward total automation of microfluidics for extraterrestial in situ analysis. *Analytical Chemistry*, 83(22):8636–8641, 2011. PMID: 21972965.
16. J. Perkel. Life science technologies: microfluidicsbringing new things to life science. *Science*, 5903(322):975–977, 2008.
17. P. Pop, W. H. Minhass, and J. Madsen. *Microfluidic very large scale integration (VLSI): modeling, simulation, testing, compilation and physical synthesis*. Springer, Cham, 2016.
18. M. L. Raagaard and P. Pop. Pin count-aware biochemical application compilation for mvlsi biochips. In *Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP), 2015 Symposium on*, pages 1–6, April 2015.
19. T. Thorsen, S. J. Maerkl, and S. R. Quake. Microfluidic large-scale integration. *Science*, 298(5593):580–584, 2002.