A Field-Programmable Pin-Constrained Digital Microfluidic Biochip

Daniel Grissom, Philip Brisk
Department of Computer Science and Engineering
University of California, Riverside
{grissomd, philip}@cs.ucr.edu

ABSTRACT

As digital microfluidic biochips (DMFBs) have matured over the last decade, efforts have been made to 1.) reduce the cost, and 2.) produce general-purpose chips. While work done to generalize DMFBs typically depends on the flexibility of individually controlled electrodes, such devices have high wiring complexity, which requires costly multi-layer printed circuit boards (PCBs). In contrast, pin-constrained DMFBs reduce the wiring complexity, but reduce the flexibility of droplet coordination. We present a field-programmable pin-constrained DMFB that leverages the cost-savings of pin-constrained designs, but is general-purpose, rather than assay-specific. We show that with just a few more pins than the state-of-the-art pin-constrained designs, we can execute arbitrary assays almost as fast as the most recent general-purpose DMFB designs.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids; J.3 [**Life and Medical Sciences**]: Biology and Genetics, Health

General Terms

Algorithms, Design, Performance.

Keywords

Digital Microfluidic Biochip (DMFB), Laboratory-on-Chip (LoC), Pin-Constrained, Field-Programmable.

1. INTRODUCTION

This paper presents a design for a field-programmable, pin-constrained digital microfluidic biochip (DMFB). Just as a field-programmable gate array (FPGA) can be programmed by an end-user in the "field," a field programmable pin-constrained DMFB can be programmed to execute any *assay* (biochemical protocol) after it has been designed and manufactured. In contrast, prior pin-constrained DMFBs have been assay-specific [9][17].

Direct-addressing DMFBs provide independent control over each electrode; these devices are costly because the large number of control inputs and high wiring complexity increases the number of printed circuit board (PCB) layers. Pin-constrained DMFBs, in contrast, have fewer control inputs and low wiring complexity, but lack flexibility. This paper introduces *the first* pin-constrained DMFB with sufficient flexibility to enable field-programmability.

DAC '13, May 29 – June 07 2013, Austin, TX, USA.

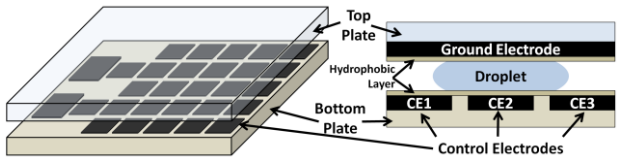

Figure 1. (a) Planar array of electrodes; (b) Cross-sectional view of electrode array.

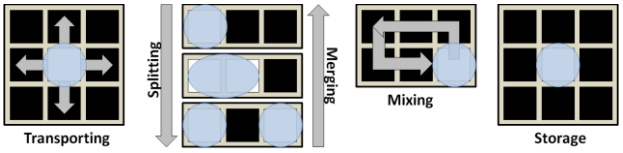

Figure 2. Basic microfluidic operations form the building blocks for assays to be executed on an array of electrodes.

1.1 DMFB Technology Overview

1.1.1 Background: Physical Droplet Manipulation

DMFBs execute assays by manipulating nanoliter-sized droplets of fluid. DMFBs are typically based on a phenomenon known as electrowetting [11]. An electrowetting-based DMFB, as seen in **Figure 1**, consists of a top and bottom plate coated with a hydrophobic layer. The bottom plate has an array of droplet-sized control electrodes, while the top plate has a single conducting electrode that spans the entire array of control electrodes. Each droplet is sandwiched between the bottom and top plates and will hold its place if its underlying electrode remains activated.

In **Figure 1(b)**, a droplet overlaps neighboring electrodes; if a neighboring electrode is activated, the droplet will begin to flow toward the newly activated electrodes. Thus, if CE3 is activated and CE2 is simultaneously deactivated, the entire droplet will move to cover CE3. As seen in **Figure 2**, with the proper sequence of electrode activations, several basic microfluidic operations can be performed. Sensor-based detection operations execute by moving a droplet to a detector (placed above an electrode) and storing the droplet there. Dispense and output operations are performed by I/O reservoirs on the perimeter.

If a droplet is not centered over or adjacent to any activated electrodes, it will *drift* across the DMFB in an undetermined and unpredictable manner.

1.1.2 Background: High-level Assay Synthesis

Figure 3 illustrates the process of synthesizing an assay onto a DMFB. A directed acyclic graph (DAG) represents the assay; each node represents a microfluidic operation (i.e., dispense, output, split, mix/merge, detect), while the edges represent dependencies and order of operations (e.g., in **Figure 3**, *M1* cannot be executed until *I1* and *I2* are complete).

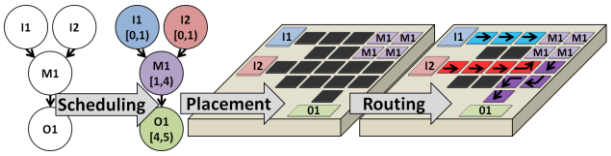

Figure 3. A microfluidic assay is represented in the form of a DAG; its operations are then *scheduled* and *placed* onto the DMFB array; droplets are then *routed* between operation locations.

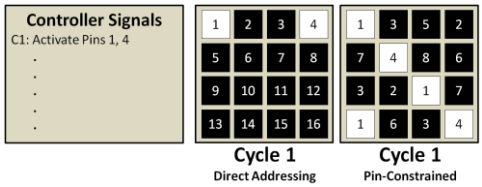

Figure 4. Activating a pin on a (a) direct-addressing DMFB activates (white) exactly 1 electrode per pin; (b) a pin on a pin-constrained DMFB activates 1+ electrodes per pin, depending on the pin layout.

The DAG is scheduled such that each operation has a specific start and stop time-step; a *time-step* is the basic scheduling unit for operations and usually lasts for 1s or 2s. Next, the placer selects a specific I/O port or group of cells, called a *module*, to perform each assay operation. Lastly, the router computes droplet pathways between all operations at the start of each time-step.

1.1.3 Background: Low-Level Pin Mapping

The output of the compiler is a list of pins to active each cycle; a *cycle* is the time it takes to move a droplet from one electrode to the next. In **Figure 4**, a "dry" controller (e.g., a PC) sends signals to activate pins during each cycle on the "wet" DMFB.

In a direct-addressing DMFB, each pin is electrically tied to a single electrode such that an $m \times n$ array of electrodes has $m \times n$ external pins driven by the dry controller. As each electrode is individually controllable, direct addressing allows for maximal flexibility in coordinating droplet-movement. Unfortunately, the complexity of routing $m \times n$ pins underneath a $m \times n$ electrode array is complicated and requires increasingly more PCB layers as the array increases in size, leading to expensive products [17].

Pin-constrained DMFBs connect each control pin to multiple electrodes (**Figure 4(b)**) to reduce the number of wires routed underneath the electrode array; this reduces the number of PCB layers, which, in turn, reduces cost. For a pin-constrained DMFB, activating a control pin will active multiple electrodes, as shown in **Figure 4(b)**. This complicates assay compilation, as independent control over individual electrodes no longer exists. Thus far, pin-assignment has been proposed to reduce the cost of assay-specific DMFBs [9][17], but generalized pin-constrained DMFBs that execute arbitrary assays have not yet been realized.

1.2 Contribution

The contribution of this paper is a pin assignment scheme that facilitates all basic microfluidic operations at pre-determined locations in a pin-constrained DMFB; the resulting DMFB is therefore field-programmable, rather than assay-specific. A high-level synthesis flow targeting this device establishes automatic compilation. Experiments demonstrate that field-programmability is achieved with a handful of additional control pins compared to state-of-the-art assay-specific pin-constrained DMFBs, and that the performance overhead incurred at the cost of field-programmability is marginal.

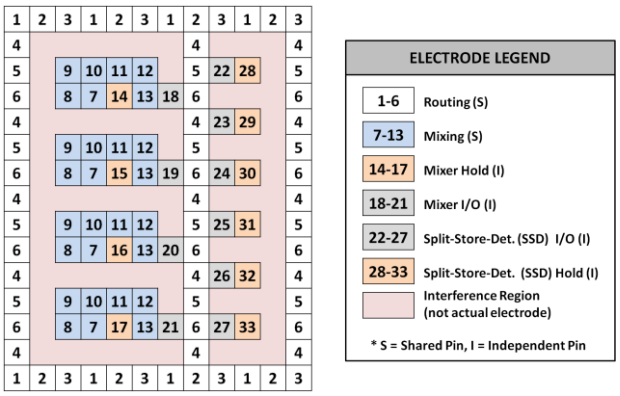

Figure 5. Pin diagram for a 12×15 field-programmable, pin-constrained DMFB which can accommodate 4 mix modules and 6 split/store/detect (SSD) modules. Routing and mixing pins are shared; the interference region does not contain actual electrodes. Holding and I/O electrodes are independently wired to single control pins for flexibility and programmability.

2. RELATED WORK

Griffith and Akella [1] and Grissom and Brisk [2][3] impose virtual topologies on top of direct-addressing DMFBs; a virtual topology is a mesh-like network of streets and rotaries that perform droplet transport, and dedicated reaction chambers that perform all other assay operations. This limits flexibility, but simplifies certain aspects of dynamic recompilation in response to operation variability and errors. The field-programmable pin-constrained DMFB employs a physical topology/architecture in order to facilitate general-purpose assay execution.

Xu and Chakrabarty [17] introduced a multi-functional pin-constrained DMFB that can execute a pre-defined set of assays; however, it is not field-programmable. Luo and Chakrabarty [9] introduced a pin-assignment algorithm that ensures that two droplets can move independently on a pin-constrained DMFB without conflicting; their algorithm reduced the number of pins required to realize the multi-functional architecture. To the best of our knowledge, their approach is not field-programmable when more than two droplets move concurrently.

Other works have since come and optimized various aspects of the pin-constrained problem. One such recent work is presented by Huang and Ho and combines the droplet routing and pin-count reduction problems together [6]. This is different from the typical approach which starts with droplet routes that have been computed on a direct-addressing array and then attempts to reduce pins. Their solution uses sequential global routing and incremental integer linear programming (ILP) stages to compute solutions. Zhao and Chakrabarty also offer an ILP and heuristic solution to the droplet routing and pin-count co-optimization problem [19].

Lin and Chang present work which focuses on the problem of droplet cross-contamination on pin-constrained devices [8]; they describe scalable algorithms which allow wash droplets to safely clean up contaminated areas on pin-constrained devices. Finally, S. Roy describes an orientation strategy to connect and wire external pins to electrodes on multi-chip devices executing identical assays in lockstep [12].

3. PIN-CONSTRAINED ASSIGNMENT

The field-programmable pin-constrained DMFB employs a pin assignment scheme that enables all of the basic assay operations (**Figure 2**) to execute in a conflict-free manner. **Figure 5** shows a

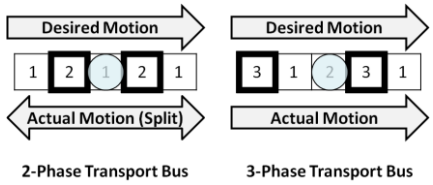

Figure 6. At least 3 repeatable pins are needed to move a droplet along a straight path without causing the droplet to split. Electrodes with bold borders indicate electrodes being activated next cycle.

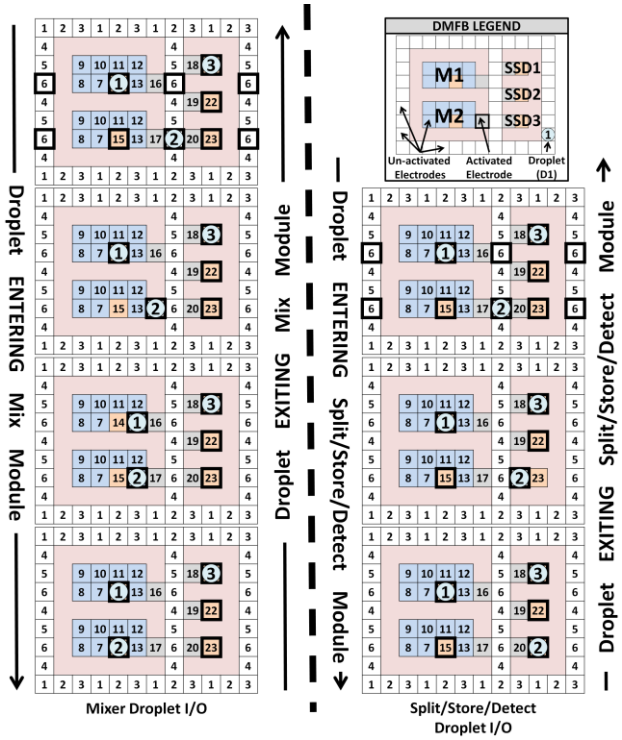

Figure 7. Pin-activation sequence showing how a single droplet (*D2*) can enter/exit (a) mix modules and (b) split/store/detect modules. Sequences are designed to allow a droplet to enter/exit any module without adversely affecting droplets (*D1*, *D3*) in other modules.

12×15 field-programmable, pin-constrained DMFB. The topology/architecture reserves specific DMFB regions for assay operations and others for routing. The topology contains a vertical column of mixing modules on the left (blue/orange electrodes, 7-17) and a vertical column of different modules on the right (orange electrodes, 28-33) that perform splitting, storage, and detection (which requires an external detector affixed above the module); we call these modules *SSD modules*.

White electrodes labeled 1-6 surround the two columns of modules and define the droplet routing regions. I/O reservoirs can be placed anywhere along the perimeter of the chip. The gray electrodes labeled 18-27 indicate pins that allow droplets to enter and exit each module. An *interference region* (pink) surrounds each module to isolate droplets within the module from droplets in the routing region or adjacent modules. These regions are not functional, and do not contain electrodes.

The layout is designed for operation concurrency and scalability. Since routing times are so much shorter than operation times [16], we devote more pins to modules (as opposed to the routing region) to allow more operations to be executed simultaneously at any time-step. The architecture can also be lengthened or shortened in the vertical dimension to produce a DMFB with any desired number of modules.

3.1 DMFB Operations and Synchronization

The following sub-sections briefly describe how the basic microfluidic operations are performed on our field-programmable, pin-constrained DMFB.

3.1.1 Droplet Transport

Figure 6 shows that at least 3 pins are required to successfully transport a droplet along a straight path; this is called a 3-phase transport bus [13]. In **Figure 5**, pins 1-3 control two horizontal transport buses; pins 4-6 construct three vertical transport buses. This pin-constrained virtual architecture facilitates droplet transfer between horizontal and vertical transport busses, and routable paths exist between all modules and I/O reservoirs on the chip's perimeter. Chips of arbitrary vertical height can be instantiated without remapping the transport electrodes. The mix and SSD module hold electrodes remain active during routing to ensure that droplets within the modules do not drift.

Droplets are routed one at a time because the 3-phase transport busses do not provide a sufficient number of unique pins to sufficiently hold droplets in the routing area while another droplet enters/exits a module. Additional cells could be added to the bus to increase routing parallelism; however, given that routing times (milliseconds) are much smaller than operations (seconds), they are typically considered negligible and are often ignored [16]. See supplemental **Section S2** for more details on sequential routing.

3.1.2 Droplet Dispensing and Outputting

I/O reservoirs are placed on the DMFB perimeter. Each I/O reservoir has an individually controlled electrode that leads a droplet to the edge of the array; these are left off the diagram in **Figure 5** because they are common to all DMFB designs.

3.1.3 Merging/Mixing

Figure 7(a) illustrates a droplet (*D2*) entering and exiting a mixing module (*M2*) without conflicting with droplets in other modules (*D1*, *D3*). At the top, *D2* has reached the routing electrode adjacent to the mixing module (*M2*) it will enter; *D1* is stored in mixing module *M1* and *D3* is stored in SSD module *SSD1*. All SSD module electrodes are activated (pins 21-23) to hold all stored droplets in place during mixing module I/O. Activating pin 17 (*M2*'s I/O cell) moves droplet *D2* to a position adjacent to *M2*. Activating pin 13 draws *D2* into *M2*, while transporting *D1* to an adjacent cell within *M1*. Next, all droplet hold cells (pins 14 and 15) move *D1* and *D2* to identical positions within *M1* and *M2* respectively. **Figure 7(a)** also shows that the electrode sequence is simply reversed to facilitate a droplet leaving a mixing module.

Before mixing, two droplets must first merge (i.e., collide into each other). A nearly-identical electrode sequence as the one seen in **Figure 7(a)** handles this case. For the interested reader, we demonstrate this in **Figure S1** in the supplementary section. Once *M1* and *M2* each contain a merged droplet, they can perform mixing operations concurrently by activating cells 7-13 in sequence, followed by 14 and 15 together. This permits both droplets to complete one clockwise cycle in the mixing modules. Mixing pauses by holding droplets on their hold cells whenever another mixing module I/O operation occurs.

3.1.4 Storage, Detection, and Splitting

SSD modules perform storage and detection (if equipped with an external detector). Both operations require a droplet to enter an

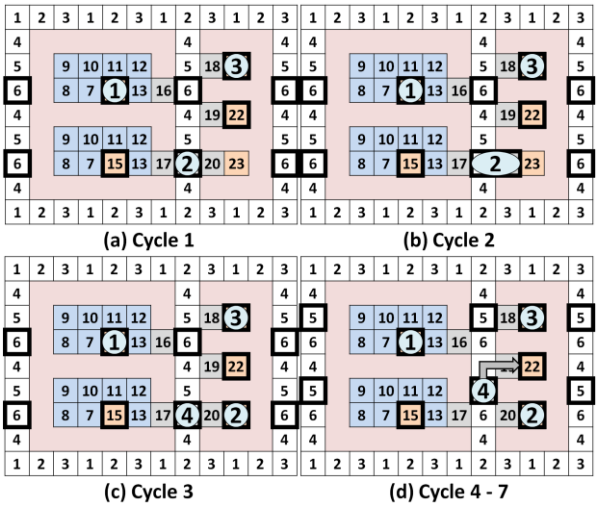

Figure 8. Pin-activation sequence for splitting a droplet (*D2*) and storing in split/store/detect (SSD) modules. Sequences are designed to allow a droplet to split and store without adversely affecting droplets (*D1*, *D3*) in other modules. NOTE: Legend same as Figure 7.

SSD module and remain in place. **Figure 7(b)** illustrates the process by which a droplet enters/exits an SSD module (*SSD3*) without affecting other droplets in other modules. All SSD hold electrodes are kept on, except for *SSD3*'s, which allows droplet *D2* to enter. *SSD3*'s I/O electrode is activated, followed by its hold electrode, to complete the entrance. This sequence is reversed to facilitate droplets exiting SSD modules.

Figure 8(a)-(c) illustrates droplet splitting. The initial position of droplet *D2*, which will be split, is on a vertical transport bus next to an SSD module's I/O cell (a). The cell on the transport bus is activated throughout the split. The I/O cell is then activated, which stretches *D2* to cover both cells (b). Next, the SSD module's hold cell is activated, and the I/O cell is deactivated; this splits *D2* into two separate droplets: *D2*, on the hold cell, and *D4*, in the transport bus. If storage is required for *D4*, then it must be routed to an available SSD module, as shown in **Figure 8(d)**.

4. FIELD-PROGRAMMABLE SYNTHESIS

This section describes the synthesis flow (**Figure 3**) that maps an assay to the field-programmable pin-constrained DMFB.

4.1 Scheduling

List scheduling [3][16] is a fast, greedy, single-path scheduling algorithm. List scheduling targeting the field-programmable, pin-constrained DMFB differs from prior implementations in several respects. The most important difference is that prior list schedulers use one generic module type for all assay operations, rather than distinguishing between mixing and SSD modules.

As shown in **Figure 8(d)**, split modules may require two SSD modules if both droplets that are produced must be stored. As shown in **Figure 9**, the split node is converted into an instantaneous split followed by two storage operations.

The scheduler reserves one SSD module to address routing deadlocks, as explained later in **Section 4.3**. Thus, in **Figure 5**, only 5 of the 6 SSD modules are available for storage and detection. Prior list schedulers may transport droplets between modules for storage for a variety of reasons [3][10]. Since only SSD modules perform storage and each stores at most one droplet, there is no motivation to transport droplets between SSD modules during storage; thus, a stored droplet remains in a single SSD

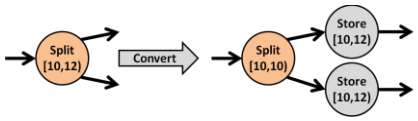

Figure 9. Split operations are converted to a split and two stores for synthesis.

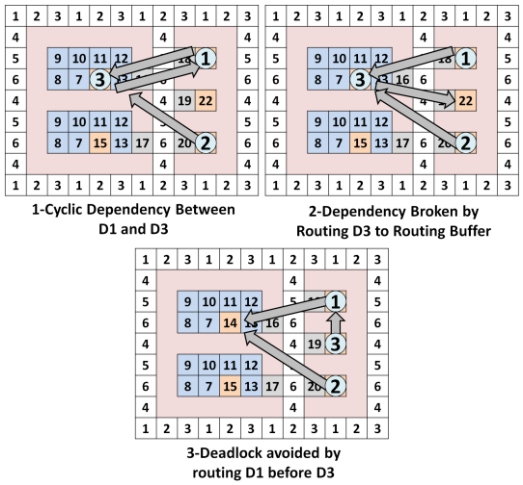

Figure 10. Cyclic routing dependencies can be broken by first routing a droplet in the cycle to the routing buffer module (one of the SSD modules). Arrows indicate that the droplet at the tail end is about to travel to the module at the head end. NOTE: Legend same as Figure 7.

module for the entirety of its storage lifetime. This may reduce the number of droplets that must be routed in certain cases.

4.2 Placement/Binding

Similar to Grissom and Brisk [2][3], we reduce placement to a binding problem, which is solved using the left-edge algorithm [7]. One minor difference between our binder and others is that we do not bind split operations since they yield two immediate storage nodes (**Figure 9**). Instead, we simply bind the two storage children directly. In the interest of space, we direct the reader to other references for a more-complete description and psuedocode of the left-edge binder [2][3].

4.3 Routing

4.3.1 Route Computation

A routing *sub-problem* refers to the set of droplets that must be routed just before each time-step begins in the schedule. Before each time-step (operation) begins, droplets are routed sequentially, one at-a-time. Given the topology in **Figure 5**, three different types of routes must be computed: input reservoir to module, module to module, and module to output reservoir.

To route a droplet from an input reservoir to a module, it suffices to compute the shortest distance from the input reservoir to the electrode adjacent to the target module's I/O electrode. The main question is to determine whether the clockwise or counter-clockwise path is shorter. Once the droplet arrives, the appropriate module input sequence is applied, as discussed in **Section 3.1**.

Module-to-module routing uses the vertical column in the center of the DMFB. The router applies the output sequence to extract the droplet from the source module, routes the droplet north or south as appropriate, and applies the input sequence to deliver the droplet to its target module.

Table 1. Experimental results comparing the direct-addressing DMFB (DA) [3] with our field-programmable, pin-constrained DMFB (FP).

Direct-Addressing DMFB (DA) vs. Field-Programmable Pin-Constrained DMFB (FP)												
Benchmarks	Array Dim.		# Electrodes Used		# Pins		Routing Time (s)		Operations Time (s)		Total Time (s)	
	DA	FP	DA	FP	DA	FP	DA	FP	DA	FP	DA	FP
PCR	15x19	12x21	285	153	285	43	0.7	2.1	11	11	11.7	13.1
In-Vitro 1	15x19	12x21	285	153	285	43	0.7	2.6	14	14	14.7	16.6
In-Vitro 2	15x19	12x21	285	153	285	43	1.2	3.8	18	18	19.2	21.8
In-Vitro 3	15x19	12x21	285	153	285	43	1.9	6.2	22	18	23.9	24.2
In-Vitro 4	15x19	12x21	285	153	285	43	1.8	8.8	24	19	25.8	27.8
In-Vitro 5	15x19	12x21	285	153	285	43	2.9	11.6	32	25	34.9	36.6
Protein Split 1	15x19	12x21	285	153	285	43	1.8	2.9	71	71	72.8	73.9
Protein Split 2	15x19	12x21	285	153	285	43	6.2	6.1	106	106	112.2	112.1
Protein Split 3	15x19	12x21	285	153	285	43	13.9	13.5	176	176	189.9	189.5
Protein Split 4	15x19	12x21	285	153	285	43	32.9	29.3	316	316	348.9	345.3
Protein Split 5	15x19	12x25	285	177	285	49	63.6	61.4	670	596	733.6	657.4
Protein Split 6	15x25	12x29	375	203	375	55	161.2	127.4	1156	1156	1317.2	1283.4
Protein Split 7	15x25	12x31	375	239	375	63	290.3	260.6	2353	2276	2643.3	2536.6
Avg. Normalized Improvement: (> 1 is improvement)			**1.82**		**6.53**		**0.68**		**1.07**		**0.98**	

Table 2. We present results from Xu's [17] and Luo's [9] pin-constrained designs for chips which can run PCR, In-Vitro 1, Protein Split 3 and a multi-functional chip which can run all three.

Xu's Pin-Constrained Results vs. Luo's Pin-Constrained Results						
Benchmark	Array Dim.	# Electrodes Used	# Pins		Total Time (s)	
			Xu	Luo	Xu	Luo
PCR	15x15	62	14	22	20	30
In-Vitro 1	15x15	59	25	21	73	90
Protein Split 3	15x15	54	26	20	150	170
Multi-Function	15x15	81	37	27	150	170

Table 3. We demonstrate the three benchmark assays from Xu [17] and Luo [9] on our field-programmable, pin-constrained DMFB design of various sizes.

Total Assay Times for Increasing Field-Prog., Pin-Constrained Array Size						
Array Dim.	#Module (Mix/SSD)	# Electrodes Used	# Pins	Total Time(s)		
				PCR	In-Vitro 1	Protein Split 3
12x9	2/3	62	23	18.59	19.00	-
12x12	3/4	89	27	15.89	17.26	-
12x15	4/6	111	33	12.88	16.56	-
12x18	5/7	133	39	13.00	16.60	189.65
12x21	6/9	153	43	13.08	16.60	189.53

To route a droplet from a module to an output reservoir, the output sequence is applied to extract the droplet from the source module; then the droplet is routed either clockwise or counter-clockwise along the shortest path to the output reservoir.

4.3.2 Droplet Dependencies and Deadlock

Special care must be taken to prevent droplet dependencies from turning into deadlock. Routing deadlock occurs when one or more droplets are waiting for resources to become available that will never become free. This can occur when a droplet dependency cycle occurs, as seen in **Figure 10**. *D1* is in *SSD1* and waiting for droplet *D3* to leave *M1*, while droplet *D3* is in *M1* and waiting for droplet *D1* to leave *SSD1*. To break the cycle, we pick *D3* to first route itself to an empty SSD module (*SSD2*), as shown in step 2 of **Figure 10**. The scheduler always keeps one SSD module unallocated, as it cannot predict routing dependencies a-priori.

As seen in step 3 of **Figure 10**, although the cyclic dependency is broken between droplets *D1* and *D3*, deadlock can still occur if the sequential droplet routing order is chosen poorly. Now, droplet *D3* travels to *SSD1*, while droplets *D1* and *D2* travel from *SSD1* and *SSD3*, respectively, to *M1*. Droplet *D2* can be routed at any time, because no droplets will travel to *SSD3* (its source) and no droplets remain at *M1* (its destination) that must first move. If the router tries to route droplet *D3* before routing *D1*, then deadlock will occur because *SSD1* is not yet free to receive new droplets. If there is no such dependency check, droplet contamination will occur. We describe a general algorithm to eliminate droplet dependencies and provide the details and pseudocode in supplemental **Section S3**.

5. EXPERIMENTAL RESULTS

We implemented our field-programmable, pin-constrained DMFB in C++; we compare with Grissom and Brisk's fast online synthesis framework [3], which is publicly available online [5]. All tests were run using a 2.8GHz Intel Core i7 CPU and 4GB RAM on a 64-bit version of Windows 7.

5.1 Comparison to General DMFB

We first compare our implementation to the most recent generally programmable direct-addressing DMFB design [3]. We run a set of 13 assays based on the PCR [15], in-vitro diagnostics [14][15] and protein-split benchmarks [4]. **Table 1** shows the number of seconds spent both routing and executing assay operations; the total time is the sum of the two. Results are also given for the number of usable electrodes (i.e., tied to a control pin) and number of external control pins for the DMFB size used. For Protein Split 5-7, the array dimensions had to be increased to execute the assay for one or both of the DMFBs.

Our DMFB has longer routing times for the first 7 benchmarks because of sequential routing; however, it actually has shorter routing times for Protein Split 2-7 because additional routes are not generated between storage nodes, as described in **Section 4.1**. Modules in the direct-addressing DMFB [3] can store up to two droplets at any time. To utilize as many resources as possible, droplets stored alone in separate modules will consolidate in order to free up more modules to do useful work; routing these droplets adds to the routing time, and therefore the total time as well.

The bottom row of **Table 1** shows the average improvement of our field-programmable DMFB compared to the direct-addressing DMFB. We calculated this metric by computing the improvement of FP over DA (baseline) for each benchmark and then averaging these values over the entire set of benchmarks. Any value over 1 means FP is an improvement. Notice that, although FP's average routing time is 32% slower, its average operation time is 7% faster. On average, the field-programmable pin-constrained DMFB only suffers an average 2% slowdown in total execution time, while reducing the pin count by 6-7×.

5.2 Comparison To Pin-Constrained DMFBs

Table 2 presents results for two prior pin-constrained assay-specific and multi-functional DMFB architectures [9][17]; assay-specific architectures were generated for PCR, In-vitro 1, and Protein Split 3 assays, while the multi-functional chip can perform all three assays. Many differences exist between these designs, most notably that they are assay specific while ours is field-programmable, and that they use linear array mixing modules, which have longer latencies than the 4×2 mixers used here. Thus, the schedules are different, and it is unclear if their reported results include droplet routing times, as their primary objective was to reduce the cost of their pin-constrained DMFBs by reducing the pin-count. **Table 2** is reproduced from Ref. [9].

Table 3 reports the performance and pin-count for PCR, In-Vitro 1, and Protein Split 3 for field programmable pin-constrained DMFBs of varying sizes. For PCR and In-Vitro 1, execution times decrease as the DMFB size (and thus the number of available modules) increase, saturating at 12×15. For larger DMFBs, performance degrades slightly due to longer routing times.

The Protein Split 3 assay requires 6 droplets to be stored at several instances during the assay; thus, the 12×18 array with 7 SSD modules (6 available to the scheduler) is the smallest compatible device. The total execution time remains steady, regardless of resources (we also tested on a 12×81 DMFB with abundant resources) at 189s. In this case, the total execution time is not limited by resource availability, but by the 7s droplet dispense times [15]. It is unclear what droplet dispense times were assumed in prior work [9][17]; reducing the dispense times to 2s instead of 7s reduces the assay execution time to approximately 100s.

In general, the field-programmable pin-constrained DMFBs require more pins than the assay-specific or multi-functional pin-constrained DMFBs reported in **Table 2**; this is to be expected because our device is optimized for field-programmability, while their devices are optimized for reduced pin-count.

Luo and Chakrabarty's pin assignment scheme [9] theoretically provides some flexibility, as two droplets are guaranteed to be able to move without interfering with one another; however, they did not provide details on how synthesis was performed, so it is difficult to provide a direct comparison.

In contrast, the different versions of our field-programmable pin-constrained DMFB reported in **Table 3** are the same generic 2-column architecture, but with a different number of resources. As seen in **Table 1**, if we pick dimensions of reasonable size (12×21), we can run all three assays in **Table 2**, as well as others, due to the field-programmable nature of our design.

6. CONCLUSION

This paper has introduced the first field-programmable pin-constrained DMFB that can execute arbitrary assays; prior pin-constrained DMFBs have all been assay-specific or multi-functional, but not field-programmable. To program the device, we describe modifications to a synthesis flow for DMFBs, which address architectural issues that are specific to our design.

Compared to field-programmable, direct-addressing DMFBs, the field-programmable, pin-constrained DMFB offered comparable or improved performance, while reducing the pin-count by 6-7x. Compared to assay-specific, pin-constrained DMFBs, the field-programmable device offered better performance and a comparable pin-count for the PCR and In-vitro 1 benchmarks, but degraded performance and a 2x higher pin-count for Protein Split 3. Compared to the multi-functional, pin-constrained DMFB, the field-programmable pin-constrained DMFB required 1.44x more pins for a comparably sized array (12x18 vs. 15x15). Thus, field-programmable does come at a price in terms of pin-count and, sometimes, assay execution time, compared to state-of-the-art assay-specific and multi-functional pin-constrained DMFBs; however, the flexibility provided is unmatched by prior DMFBs and offers a significant advancement in terms of programmability.

7. ACKNOWLEDGEMENTS

This work was supported in part by NSF Grant CNS-1035603. Daniel Grissom was supported by an NSF Graduate Research Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the NSF.

8. REFERENCES

[1] E. Griffith and S. Akella. Performance characterization of a reconfigurable planar-array digital microfluidic system. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst, 25(2), Feb. 2006.

[2] D. Grissom and P. Brisk. A high-performance online assay interpreter for digital microfluidic biochips. In Proc. of GLSVLSI, pages 103-106, Salt Lake City, UT, USA, May 3-4, 2012.

[3] D. Grissom and P. Brisk. Fast online synthesis of generally programmable digital microfluidic biochips. In Proc. of CODES+ISSS, pages 413-422, Tampere, Finland, Oct. 7-12, 2012.

[4] D. Grissom and P. Brisk. Path scheduling on digital microfluidic biochips. In Proc. of DAC, pages 26-35, San Francisco, CA, USA, Jun. 3-7, 2012.

[5] D. Grissom, et al. A digital microfluidic biochip synthesis framework. In Proc. of VLSI-SoC, Santa Cruz, CA, Oct. 7-10,2012.

[6] T-W. Huang and T-Y. Ho. A two-stage integer linear programming-based droplet routing algorithm for pin-constrained digital microfluidic biochips. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst, 30(2), Feb. 2011.

[7] F. J. Kurdahi and A. C. Parker, "REAL: a program for REgister ALlocation. In Proc. of DAC, pages 210-215, Miami, FL, USA, 1987.

[8] C. C-Y. Lin and Y-W. Chang. Cross-contamination aware design methodology for pin-constrained digital microfluidic biochips. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst, 30(6), Jun. 2006.

[9] Y. Luo and K. Chakrabarty. Design of pin-constrained general-purpose digital microfluidic biochips. In Proc. of DAC, pages 18-25, San Francisco, CA, USA, Jun. 3-7, 2012.

[10] K. O'Neal, D. Grissom, and P. Brisk. Force-directed list scheduling for digital microfluidic biochips. In Proc. of VLSI-SoC, Santa Cruz, CA, USA, Oct. 7-10, 2012.

[11] M. G. Pollack, A.D. Shenderov, and R. B. Fair. Electrowetting-based actuation of droplets for integrated microfluidics. Lab on a Chip, 2:96-101, 2002.

[12] S. Roy, D. Mitra, B. B. Bhattacharya, K. Chakrabarty. Congestion-aware layout design for high-throughput digital microfluidic biochips. ACM Journal on Emerging Technologies in Computing Systems, 8(3): article #17, Aug. 2012.

[13] V. Srinivasan, V. Pamula, and R. Fair. An integrated digital microfluidic lab-on-a-chip for clinical diagnostics on human physiological fluids. Lab on a Chip, 4:310-315, 2004.

[14] F. Su and K. Chakrabarty. Architectural-level synthesis of digital microfluidics-based biochips. In Proc. of ICCAD, pages 223-228, San Jose, CA, USA, Nov. 7-11, 2004.

[15] F. Su and K. Chakrabarty. "Benchmarks" for digital microfluidic biochip design and synthesis. Duke University, Department of Electrical and Computer Engineering, 2006. http://www.ee.duke.edu/~fs/Benchmark.pdf

[16] F. Su and K. Chakrabarty. High-level synthesis of digital microfluidic biochips. ACM Journal on Emerging Technologies in Computing Systems, 3(4): article #16, Jan., 2008.

[17] T. Xu and K. Chakrabarty. Broadcast electrode-addressing for pin-constrained multi-functional digital microfluidic biochips. In Proc. of DAC, pages 173-178, Anaheim, CA, USA, Jun. 8-13, 2008.

[18] P-H. Yuh, C-L. Yang, and Y-W. Chang. BioRoute: a network-flow-based routing algorithm for the synthesis of digital microfluidic biochips. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., 27(11):1928-1941, Nov. 2008.

[19] Y. Zhao and K. Chakrabarty. Simultaneous optimization of droplet routing and control-pin mapping to electrodes in digital microfluidic biochips. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst, 31(2), Feb. 2012.

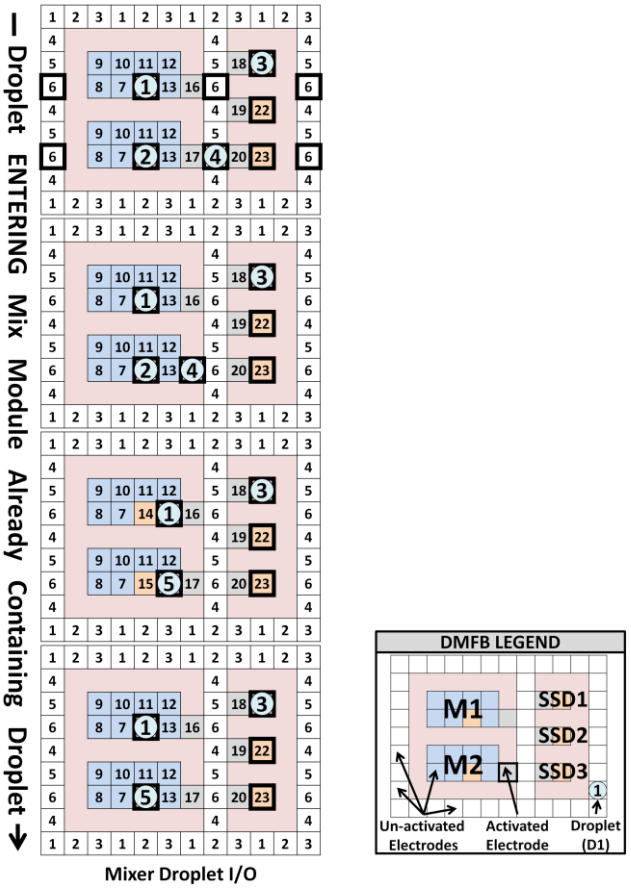

Figure S1. Shows the electrode/pin activation sequence (from top to bottom) that causes *D4* to merge with *D2* (in *M2*) to become *D5* (twice the volume) and re-sync with any other droplets in mix modules (i.e., *D1* in *M1*).

S1. MERGING TWO DROPLETS

Figure S1 shows how a droplet (*D4*) merges with an existing droplet (*D2*) in *M2* to become *D5*. Once merged, the new droplet (*D5*, with twice the volume) is synced with *D1* back to the mixers' hold locations (the bottom image in **Figure S1**). If *D1* has already been merged, then mixing can begin; if not, the general process in **Figure S1** must be repeated to merge a new droplet with *D1* before the merged droplets in *M1* and *M2* can be mixed.

S2. SEQUENTIAL ROUTING

In this section, we provide more details to explain why sequential routing was chosen. As mentioned in **Section 3**, operation times are on the order of seconds, while droplet transport times are on the order of milliseconds. A typical time-step is 1 or 2 seconds [16], while a typical droplet actuation is at most 10ms (100Hz) [18]. With this in mind, we chose the 3-phase bus approach because, although it restricts droplet routing parallelism, it simplifies the general-purpose nature of the device.

Consider **Figure S2** which shows our field-programmable, pin-constrained design with two different numbers of modules and module sizes. Notice that, despite the central vertical bus ending with pin 4 or pin 5, a clean transition can be made between buses because all of the pins adjacent to the intersection are guaranteed to be unique [9]. Thus, the same algorithms can be used to map assays to field-programmable, pin-constrained arrays of various

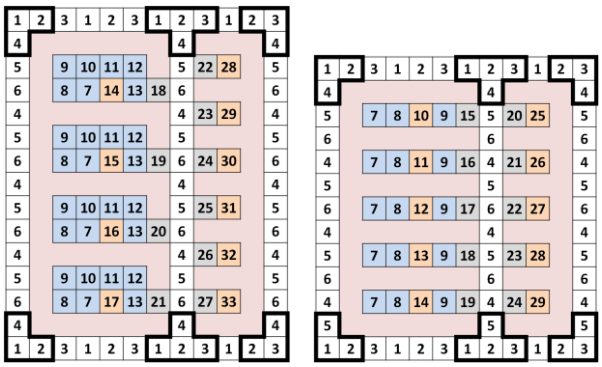

Figure S2. Shows that the number or size of modules can be changed and the 3-phase bus can be repeated, regardless of array size, without causing pin-conflict at the vertical-horizontal bus intersections (bold borders).

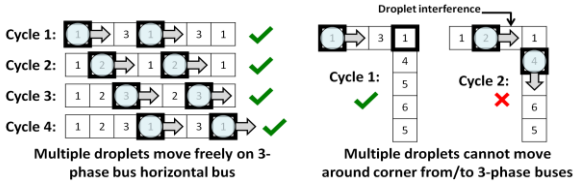

Figure S3. Shows that moving two droplets concurrently is (a) feasible when moving in a straight path, but (b) not always possible when moving around a bend because droplet interference can occur.

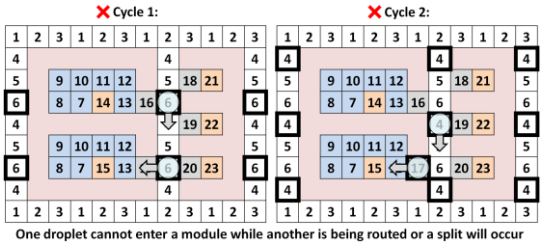

Figure S4. Shows that multiple droplets moving through the vertical bus will result in an unintentional split when one tries to enter a module.

sizes, given that they keep the same general form. This is important because it would allow an end-user to design an assay and then go purchase the cheapest compatible pin-constrained DMFB; here, compatibility means that there are sufficient resources available (meaning mixing and SSD modules) and that the SSD modules have appropriate detectors.

As seen in **Figure S3(a)**, it is always possible to move multiple droplets along a straight path on the 3-phase bus because there is sufficient space between repeating pin numbers. However, **Figure S3(b)** shows that droplet interference can occur when moving around a corner. In cycle two, if the next two pins are activated (pin 3 and pin 5), the droplets will most likely merge. It would be possible to hold pin 2 in cycle 3 such that the top droplet would stall and avoid the droplet interference in cycle 3. However, consider that droplets will only be making this transition when traveling to or from an I/O reservoir.

Most of the opportunities to parallelize routing occur when routing between modules. In light of this, consider **Figure S4**, which shows multiple droplets in the central vertical routing bus.

```
1   Given sequence graph G = (V, E)
2   int timeStep = 0;
3   Repeat {
4     graph d = ∅; // Dependencies
5       for (∀v ∈ V: v.startTime = timeStep)
6         for (∀p ∈ v.parents)
7           d.add(p.location, v.location);
8       end for
9
10      list cc = ∅; // Connected Components
11      list scc = ∅; // Strongly Connected Components
12      cc = findAllConnectedComponents(d);
13      scc = findStronglyConnectedComponents(cc);
14      resolveDependencies(scc);
15      reverseTopologicalSort(cc); // scc ⊆ cc
16
17      for (∀c ∈ cc)
18        for (∀o ∈ c: o.startTime = timeStep)
19          for(∀op ∈ o.parents)
20            routeSrcToDest(op.location, o.location);
21
22      timeStep++;
23  } until (time ≥ max (v.startTime, ∀v ∈ V)
```

Figure S5. Psuedocode for route computation.

For the lower droplet to enter the lower mixing module (*M2*), the DMFB must activate pin 17, while simultaneously deactivating pin 6. This is possible, but notice that the top droplet requires pin 4 to be activated to continue downward on its path. Activating this pin will cause two adjacent electrodes to be activated near the lower droplet, which will result in a split. Moving the top droplet up, down or keeping it stationary will require pins 5, 4, or 6 to be activated in cycle 2, respectively, which will each cause the bottom droplet to split. If pins 4-6 are not activated, then the top droplet will drift and the assay will not execute correctly.

Rather than deal with these complications, we chose to route droplets one-at-a-time instead, as the impact on total assay execution time is minimal.

S3. ROUTING ALGORITHM

This section elaborates on the routing process discussed in **Section 4.3**; **Figure S5** presents pseudocode. The router receives a scheduled and placed DAG $G = (V, E)$, where vertices represent operations and edges represent droplets that must be transferred between operations. Each vertex has a $location$, which indicates the module or I/O reservoir where the corresponding operation will take place.

Each vertex in V is scheduled to begin at a certain time-step, as computed by the scheduler. A time-step typically lasts one or two seconds and represents the time when operations are processed by their respective modules or I/O reservoirs. When a new time-step begins, then a new operation may start. This requires droplets to be routed to the module that will execute the operation. Thus, we start at time-step 0 (*Line 2*) and repeat the routing process for each time-step until the last scheduled operation begins (*Lines 3-23*); each iteration handles one routing sub-problem (time-step).

First, a graph of dependencies (d) is created based on the location of each node that is relevant to the current time-step (*Lines 4-8*). An edge *(D_x, D_y)* in the dependency graph means that droplet D_x will be routed to droplet D_y's current location, so D_y must be routed first. As seen in *Line 7*, dependencies are added to the graph based on the *location* field because droplets are being routed from the parents' location to the newly-executing node's location.

The next step is to decompose d into its connected components (*Line 12*), which can be computed using a simple recursive multi-directional, depth-first search [S1]. Connected components are processed on-by-one. To simplify further discussion, we will assume that d is composed of a single connected component.

Routing is simple if d is acyclic. Since the algorithm routes droplets one-at-a-time, edge *(D_x, D_y)* indicates that D_y must be routed before D_x; otherwise, D_x would merge inadvertently with D_y upon completing its route. A legal routing solution for the sub-problem can be achieved by routing the droplets one-by-one in reverse topological order [S2]. *Lines 10-20* in **Figure S5** solve the more complicated cyclic case, which is described next; in the simple acyclic case, *Lines 11*, *13*, and *14* are unnecessary.

If d is cyclic, routing becomes more complicated, as a cycle means that no droplet can complete its route without inadvertently merging with a droplet waiting at its destination. This problem is solved by temporarily allocating DMFB resources for storage.

The first step is to compute *strongly connected components (SCCs)* (*Line 13*) from the connected components using Gabow's path-based, depth-first search [S3]. One minor modification is that we only need to identify the SCCs that contain more than one node, as single-node SCCs do not have cyclic droplet dependencies.

Once the SCCs that represent cycles are identified, the cycles must be resolved (*Line 14*). As demonstrated in **Figure 10**, the router randomly selects a droplet D_y from the SCC and routes it to an empty SSD module for temporary storage, which breaks the dependency cycle. The dependency graph d is then modified to account for the relocated droplet's new location: each edge of the form *(D_x, D_y)* is removed from d as D_x is now free to move to its destination, since D_y has moved out of the way.

The scheduler always leaves at least one SSD module free so that there is room to break one cycle in the SCC. If the SCC contains multiple intersecting cycles, then any other free SSD or mixing module could be used for temporary storage. This process repeats until d becomes acyclic. Once d becomes acyclic, a legal routing solution can be found, as previously discussed.

One optimization that can reduce the extra storage requirement (not shown in **Figure S5**) is to break SCCs one-by-one. Droplets corresponding to vertices with no predecessors in d are routed immediately, and the corresponding vertex is removed from d. Then, an SCC is chosen that satisfies the following property: for every vertex D_x belonging to the SCC and each outgoing edge *(D_x, D_y)*, D_y also belongs to the SCC. Breaking all of the cycles in this particular SCC will ensure that at least one vertex in the updated graph d will have no successors.

The advantage of the second approach is that it reduces the need for temporary storage resources. As an example, suppose that d has two SCCs, scc_1 and scc_2, and that each requires one additional storage resource to resolve. Under the first approach, two storage resources must be allocated in order to convert d to an acyclic graph before the droplets can be routed. Under the second scheme, all of the droplets in scc_1 will be routed before all of the droplets in scc_2, or vice-versa. Therefore, both SCCs can use the same storage resource, so just one available module suffices. In general, if d contains k SCCs, and scc_i requires m_i storage modules, then the first scheme requires $M_1 = m_1 + m_2 + \ldots m_k$ modules for storage, whereas, the second requires $M_2 = max\{m_1, m_2, \ldots, m_k\}$ modules.

That being said, we did not encounter a single droplet dependency cycle in any of the 25 benchmarks seen in **Table 1** and **Table 3**; the largest assay, Protein Split 7, contains 2556 nodes. Although a droplet dependency problem can still occur in theory, it seems unnecessary, from a practical standpoint, to devote a large number of resources to resolving droplet dependency cycles, even for large assays.

S4. SUPPLEMENTAL REFERENCES

[S1] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. Communications of the ACM, 16(6):372-378, Jun 1973.

[S2] A. Kahn. Topological sorting of large networks. Communications of the ACM, 5(11):558-562, Nov 1962.

[S3] H. Gabow. Path-based depth-first search for strong and biconnected components. Information Processing Letters, 74(3-4):107-114, May 2000.