A customizable, scalable control solution for digitally-based reconfigurable magnetic microfluidic systems

by

Veronica Cojocaru

B.A.Sc., Simon Fraser University, 2010

Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Applied Science

in the
School of Engineering Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY
Fall 2015

Approval

Name:	Veronica Cojocaru
Degree:	Master of Applied Science (Computer Engineering)
Title:	*A customizable, scalable control solution for digitally-based reconfigurable magnetic microfluidic systems*
Examining Committee:	**Dr. Ivan Bajic, P.Eng.** (chair) Associate Professor

Dr. Lesley Shannon, P.Eng.
Senior Supervisor
Associate Professor _____

Dr. Bonnie Gray, P.Eng.
Supervisor
Professor _____

Dr. Fabio Campi
External Examiner
Lecturer _____

Date Defended: 6 October 2015

Abstract

Recent trends in microfluidic technologies are leaning to more streamlined and integrated platforms that can perform a variety of tasks. In order to achieve this, a continuous-flow integrated microfluidics system needs to be made portable through the use of components that are digitally controllable. The proposed device will use magnetic-based microfluidics components, such as valves and mixers, which will require an electromagnetic based model of actuation.

The scope of this thesis is to design and optimize an FPGA-based control system comprised of a user interface, device libraries and circuitry to connect to the physical components. Particular focus is given to optimizing the actuation system for magnetic microvalves to ensure power efficiency, a trait that is paramount for a portable device such as the proposed microfluidics platform. Theoretical models and simulations are evaluated through experimentation to determine which best correlate with the physical system. This enables the selection of a set of parameters that result in a power-efficient actuation system. The simulations and evaluations are used to define a procedure for parameter selection.

The selection criteria for these parameters are evaluated for an example system and the resulting actuation system behaves as predicted in a physical demonstration. The actuation system is integrated with the user interface through a software framework designed to be modular, scalable and easy to upgrade.

Keywords: FPGA; microfluidics; microactuators; microvalves; magnetic actuation

Acknowledgements

I would like to thank my senior supervisor, Dr. Lesley Shannon and supervisor Dr. Bonnie Gray for their willingness to provide help and advice even through all their busy schedules. Thank you for involving me in this project, it may not have always gone as planned, but it was a great learning experience and I feel glad to have contributed to it.

I would also like to thank all my colleagues in the Reconfigurable Computing Lab and the Microinstrumentation lab for the amazing learning environment they helped create. I really appreciated all the group discussions (whether on topic or not), the paper review meetings with ice cream (thanks Eric) and the other social events.

In particular, thank you to Mona Rahbar, for providing all the mixer and valve samples for testing and for helping me figure out valves, mixers and other microfluidics stuff, Eric Matthews, for help on the computing side of things and Dan Hilbich for a bit of both.

Of course, thanks to my family and friends for always being there and to Cody and his family for being there also, especially for letting me use their (much cooler) house in the heat wave during which this thesis was written.

I would also like to thank Simon Fraser University for providing a wonderful place of learning and for helping me fund my degree through the TA opportunities provided.

Thank you to all of the above and to anyone else I may have forgotten to mention :)

Preface

The work presented in this thesis is part of a larger collaborative project (µROAMS project) combining the skills of engineers working in the microfluidic systems and reconfigurable computing fields. As of the time of this writing, the following researchers are directly contributing to this project, under the supervision of Dr. Lesley Shannon and/or Dr. Bonnie Gray:

Mona Rahbar (PhD candidate): is developing a flexible magnetic polymer technology in order to create and characterize microvalves and micro mixers. We use these valves and mixers in prototyping for the µROAMS system. All the valves and mixers used in the tests for this thesis have been designed and fabricated by her. She has published papers about her magnetic mixers and valves, however, some of the valve characterizations still need to be published in a future paper.

Hsiu-Yang Tseng (PhD candidate): is developing a micro qPCR sensor in order to detect specific kinds of DNA in a way that is suitable for a portable microfluidics device. He has published papers about his system.

Eric Matthews (PhD candidate): is developing a software toolchain in order to enable easier design and configuration of reconfigurable microfluidic platforms, in terms of system layout as well as runtime configuration choices. This work is not yet published in a paper.

Myself: I am responsible for the user control and system integration framework. To design the interface between the computer control and the physical system, in particular the magnetic valves and mixers designed by Mona, I had to get samples of these devices and ran tests to see how they fit into the system.

Glossary

CAD Computer Aided Design. 1–3, 17–19, 24, 25, 34, 35, 69, 71

FPGA Field Programmable Gate Array. 3–6, 17–20, 29, 34, 35, 66–70

GUI Graphical User Interface. 37

HDMI High-Definition Multimedia Interface. 33, 34

I/O Input/Output. 19, 20, 29, 37, 66–68

M-CP Magnetic Composite Polymer. 8, 22

MEMS Microelectromechanical Systems. 1, 9–11

PDMS Polydimethyl siloxane. 8, 9, 22, 31, 42, 56, 57

qPCR quantitative real-time Polymerase Chain Reaction. 18, 38

USB Universal Serial Bus. 33, 34

µROAMS Microfluidic Reconfigurable On-site Analyser for Multiplexed Samples. 2–4, 12, 17–20, 24, 27, 30, 61, 69

µTAS Micro Total Analysis System. 1, 69

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, lab-on-a-chip systems developed in the field of Micro Total Analysis System (μTAS) are becoming more and more streamlined, integrated to enable multi-tasking [3]. The end goal is to design a cost-efficient device that can be taken into the field and used to perform tests quickly and reliably. In order to achieve a cost-efficient device, the microfluidics system needs to be made out of easy to fabricate components, with an overall structure that is easily mass-produced, analogous to the fabrication techniques developed for silicon-based computer chips.

Traditionally, large scale tests need to be performed using conventional laboratory techniques, or otherwise using large dedicated equipment. Many of these tests are not automated: one or more people are required to be in the lab to keep track of the samples, take them and place them in the appropriate test receptacles and manually run them through the machines. These tests are integral to many fields such as environmental monitoring, chemical detection, and medical testing. Clearly, there is a need for evolving microfluidics systems and to develop a methodology for optimizing these devices in terms of design time, routability, power consumption, etc.

In order to accomplish this sort of optimization, a continuous-flow, reconfigurable microfluidics platform is being designed by a collaboration of Microelectromechanical Systems (MEMS) engineers and computer engineers. The idea is that with the microfluidics components knowledge as well as the system level integration and Computer Aided Design (CAD) knowledge, a cooperative co-design is achieved between the design of physical microfluidic components and the design of CAD tools for layout and control. This joint microflu-

idics project is a Microfluidic Reconfigurable On-site Analyser for Multiplexed Samples (µROAMS). This microfluidics platform includes microfluidics flow control components, modules for performing tests on the fluid samples and a display and user interface that is integrated with on-system CAD tools for controlling the operation of the device. A second set of CAD tools for system design are also in development: they will enable researchers to easily design microfluidics platforms for specific tasks.

Thus far, reconfigurable microfluidic devices have only used discrete flow for moving droplets of fluid across their chips, such as in [4]. Continuous flow allows for volume control and provides more device usage options. A continuous-flow reconfigurable microfluidics such as µROAMS is a big step forward in microfluidics analysis systems. The µROAMS project is discussed further in the background section. µROAMS is still in an early prototyping stage: individual components are fabricated but the interfacing between components is still in early development. The importance of the work in this thesis is that it is a first step to microcomponent control and integration to allow a prototype version of the µROAMS device to be built.

1.1 Motivation

Microfluidics components designed for the µROAMS project, such as valves and mixers, are made out of a flexible, magnetically-doped polymer. Magnetic microvalves are easy to fabricate, have a fast response time and can easily be scaled up to arrays of various sizes, thus an array of these valves is the ideal choice for a portable platform [5]. An active mixer using cilia made out of this material has also been fabricated [2].

This focus on magnetically actuated microdevices requires designing a control system in such a way that it can be used to electromagnetically actuate a variety of magnetic microdevices. It must be easily tweaked to incorporate changes and optimizations to the current microdevices as well as devices that will be designed in the future. Therefore, the control system design must allow for the choice of parameters to be determined specifically for a number of different microdevices in such a way that can be incorporated into a larger layout design CAD tool to assist the design of the microfluidics layer of the overall device.

The motivation behind this research is to answer the question: how to design a scalable actuating systems for microfluidic components and integrate it into a portable, software-

controlled system? Some of the specific problems this study is trying to address are: how to make the actuation solution be easily integrated with computer control, the scalability of the solution and the power consumption of the design (i.e. can the final device be portable). Ideally, portable should mean that the device can operate on battery power, in which case the power consumption should be less than 100 W (i.e. no more than a typical laptop). However, if it is not possible to get the power consumption to be low enough for the device to operate simply from a battery, having the device be able to operate from a bigger power supply such as being plugged into a car's power supply would also be acceptable.

A brief analysis is given to the suitability of using an Field Programmable Gate Array (FPGA) for the computing requirements of the system. Using an FPGA for this kind of a control system, instead of a general microcontroller, seems ideal because the FPGA has many user controlled I/O pins, which would allow for easy control of an array of microfluidic components. The fact that the FPGA can accelerate tasks in hardware is useful for the potentially computationally intense run-time routing CAD tool flow for computing the ideal routing solution. However, FPGAs are not ideal for their power consumption when compared to a microcontroller. Since the device is still in an early prototyping stage, the choice of using an FPGA is not finalized and the power optimizations discussed in this work relate to the microfluidic actuation system only.

Three main parts make up the work on this project: the physical component including the valves, mixers and sensors, the CAD tools (layout design and run-time routing tools) component and the integration component. I am responsible for investigating the integration component, including actuation and computer control, and setting up this system in such a way that make it easy to use, upgrade and scale.

1.2 Objectives

The objective of my work is to provide a framework for integrating the physical components of the µROAMS system that rely on electromagnetic actuation, such as the valves [1] and mixers [2] with the computing aspects, such as the user interface and the CAD tools used to program the system. This control solution is tailored to scalable electromagnetically based systems that can be reconfigured at run-time for different test flows.

Specifically, my work demonstrates the actuation of microvalves and magnetic cilia through a FPGA-based control system. In order to accomplish this, it is necessary to perform additional tests on the existing microvalves and cilia to determine the amount of magnetic field needed to actuate them. After the requirements are determined, I design a system that provides the necessary magnetic field while meeting timing constraints. This actuating system needs to be optimized for power consumption, because the final device is required to be portable. I consider all these requirements and trade-offs in my analysis and determine a matching solution. Then, I test the proposed control solution to ensure that the theoretical equations used in the control system implementation hold up when applied to the physical world. I evaluate the effectiveness of the control solution and outline potential improvements.

1.3 Contributions

For this thesis, I have designed the interface between the existing physical components and the digital-based control system. This enables the integration of the system so that it can be scaled and updated for future use. I design and build a FPGA-based control system for actuating magnetic components. In this thesis, I present the methodology I use to determine a reliable and efficient design for the given constraints. I explain the importance of each of the parameters I test for, then show how I determine my control solution based on the test results. µROAMS is meant to be a portable microfluidics solution, thus minimizing power consumption is paramount, however, reliability and scalability, especially in terms of being able to mass-produce the system, are also very important. Therefore, I evaluate my design choices mainly based on these three constraints.

The major contributions of this thesis are as follows:

- I identify and evaluate the different parameters of an electromagnetic actuating solution
- I specify and test a procedure for optimizing the electromagnetic actuating solution based on the parameter evaluation and system requirements
- I integrate the physical device actuation with the digital control system by determining the interface circuitry

- I build the code framework for the user interface and device libraries to allow the whole system to be easily used and upgraded

1.4 Thesis Organization

This thesis is organized into six chapters. The next chapter provides the background information about microfluidics, the microvalves used in this project, inductors and electromagnets, FPGAs and device libraries. Chapter 3 gives details about the design methodology used to design the physical actuation system whereas Chapter 4 provides the details of the overall software solution. Chapter 5 outlines the experiments conducted and gives the results from these experiments. Chapter 3 shows how the parameters are evaluated to obtain the solution, and finally Chapter 7 concludes the thesis.

Chapter 2

Background

There are two key contributions to this work, the microactuation circuitry for the microfluidic system and the digital system design. This chapter provides an overview of microfluidics and magnetic microvalve research, along with the particular design details for the microvalves used for this project. More details about magnets and electromagnets, including magnetic field and magnetic force calculations are also provided. Finally, an introduction to FPGAs and device libraries is given to provide context for the digital system design on the FPGA.

2.1 Microfluidics

Microfluidics is the study of how fluids behave when present in very small amounts, i.e. micro-litre amounts in channels and chambers on the micro-meter scale. The field of microfluidics is driven by the desire to innovate better ways of manipulating these aforementioned small amounts of fluid: a challenging task since, for micro-litre volumes of fluid, factors such as friction with the channel walls and capillary action play a much more important role than gravity does. Consequently, fluid control devices such as valves and mixers need to be designed differently because of the fluid and material constraints at these small dimensions. Friction is a much more daunting force to overcome at the micro-scale, so care must be used when designing any active microfluidics devices and actuating systems.

Typically, pressure driven flow is the mechanism used to direct fluid through a microfluidic system. Assuming no extra resistance or capillary action from the microchannel being

either hydrophobic or hydrophilic, the relationship between the pressure differential over a microchannel, the flow rate and fluidic resistance of that microchannel is as follows:

$$\Delta P = R \times Q, \tag{2.1}$$

where, ΔP is the change in pressure over the channel in question, R is the fluidic resistance, and Q is the flow rate. The fluidic resistance itself is calculated as follows:

$$R = \frac{128L}{{D_{eff}}^4}, \tag{2.2}$$

where D_{eff} is the effective hydraulic diameter. D_{eff} is calculated as:

$$D_{eff} = \frac{64D_h}{k}, \tag{2.3}$$

where D_h is simply the hydraulic diameter (four times the area divided by the perimeter), and k is a correction factor that depends on the geometry of the channel. For the 2:1 rectangular microchannels used as the system-level design parameter in this collaborative project, $k = 62.19$ [6].

2.1.1 Magnetic Microvalves

Magnetic microvalves are not a new concept in microfluidics [7]. The first miniaturized magnetic valve, perhaps the first microvalve, was presented in 1979 [8]. It was designed for a gas analysis system and fabricated in silicon using conventional silicon fabrication techniques [8]. New magnetic microvalve designs emerged in the 1990s and they continued to surge in popularity in the 2000s [9].

One of the big drawbacks of the earlier uses of magnetic components in microfluidics is that they achieved the magnetism by difficult to mass produce techniques [5]. One such method involved implanting small magnets into the desired structure, and the resulting devices are hard to integrate with the other components of the system [5]. To overcome this drawback, designers embedded the desired magnetic properties into the microfluidic materials themselves. Researchers have explored this method since the late 1990s, such as Lagorce et al., who embedded ferrite powder into a hard epoxy resin and magnetized it, creating disks that behaved like small magnets [10]. They patterned the epoxy into a

cantilever and actuated it using planar coils [10]. Other researchers applied the innovation to embed the desired microfluidic material with magnetic powder, including flexible materials, and studied the effects the embedding had on the material properties [11]. In particular, the valves designed to be bi-stable are complicated, as the valve in [12].

Recent advances in the use of rare earth magnetic powders to create magnetic nanocomposite polymers allow for the design of an easy to mass fabricate magnetic microvalve. By doping a flexible material, Polydimethyl siloxane (PDMS), that is frequently used in microfluidics, with rare earth magnetic powders, researchers create a magnetic material suitable for microfluidics [13]. Regular PDMS is a highly compliant material, with flexible properties that engineers easily use because it is patterned through conventional microfabrication techniques. The resulting Magnetic Composite Polymer (M-CP), retains most of the elasticity of the original PDMS sample, only becoming slightly more rigid [13]. Researchers fabricate diaphragms using this material and test them with the actuation force provided from a Helmholtz coil [14].

2.1.2 Magnetic Mixers

Microfluidic flow is almost always laminar rather than turbulent, because at such small channel dimensions, the ratio of inertial forces to the force of friction (Reynold's number) is small [15]. This property means that when two fluids meet in a channel, they are more likely to flow next to one another than mix with each other, thus very slow mixing by diffusion is what occurs in microfluidics, as in Figure 2.1.

In order to encourage the mixing process, fluids are manipulated, either passively or actively in order to increase the contact area between the two fluids. An example of a passive mixer is one in which ridges and extra walls are added to divert the fluids in such a way as to cause them to interact such as the coaxial jet mixer in [16]. Conversely, an active mixer is a device that performs an action to encourage the mixing, i.e. stirring, such as the magnetically actuated cilia developed in [17] or the magnetic stir rod in [18].

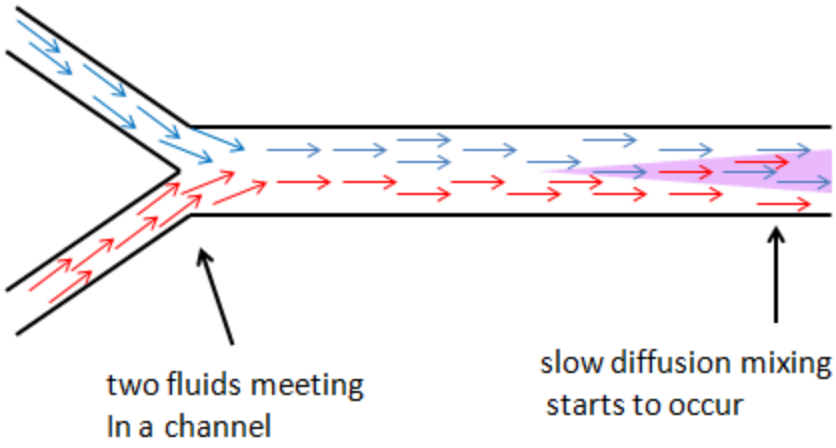

Figure 2.1: Diffusion mixing is the predominant form of mixing in microfluidics

2.1.3 Other Magnetic Microcomponents

Microvalves and mixers are not the only MEMS components developed to make use of magnetic actuation. Pumps, traps, transporters, separators and sensors exist that rely on magnetic forces for their operation [9].

Researchers have developed a variety of magnetic pump technologies such as magneto-hydrodynamic pumps that apply electric and magnetic fields to move conducting liquid, ferrofluidic pumps that manipulate magnetic fluid to generate movement in the channel, or a pump that uses a magnetic component to generate a stirring motion [9]. Hatch et al. present a ferrofluidic pump that provides an adjustable, continuous pumping flow rate as high as 45 µL/min [19]. Lemoff and Lee present a magnetohydrodynamic pump capable of continuous flow without containing any moving parts [20]. Pan et al. present a microvalve consisting of PDMS membranes, one of which has a small permanent magnet attached to it [21]. They apply an external magnetic field to generate a moving force in the membrane with the attached magnet, thus creating fluid flow [21].

Ferrofluidic materials have a wide range of applications in microfluidic systems: they can be used as pumps, valves, mixers, etc [22]. They can be coated onto non-magnetic components, thus allowing non-magnetic MEMS devices to be actuated magnetically [23].

Traps and transporters are very similar to pumps, but instead of operating on a continuous flow basis, they retain and move discrete droplets of fluid from one location (trap) to another [9]. One example of this is the magnetic particle displacement strategy presented

in [24]. Another way to move small amounts of chemical and biological reagents around a device is to move magnetic microparticles coated in the substance by applying magnetic fields [25]. Traps are used to manipulate the magnetic field to either separate or concentrate magnetic particles [26]. Separators use the magnetic properties of certain chemical or biological substances in order to manipulate them. They can be used to extract magnetic particles from a mixture as well as enable many chemical processes [27]. They can even be used to interact with magnetotatic bacteria [28].

Magnetic forces have the advantage of being able to induce large deflections in micromaterials, thus they are used for manipulating and arranging microstructures. Torsional microactuators with large out-of-plane displacements can be clamped and flexed by manipulating the magnetic field in embedded planar coils [29]. Electromagnetic MEMS mirror arrays are used for optical switching [30].

The types of magnetic microcomponents are numerous, and moreover, each type of components has many different implementations and variations. The methods for actuating these kinds of components are likewise varied, as is discussed in the next section. Some microcomponents are actuated with permanent magnets, some with electromagnetic coils, while others use actuation schemes that are custom designed and fabricated for their operation. In addition, some magnetic microcomponents are themselves an actuation system for other microcomponents, such as the micromagnetic system designed for manipulating microbeads presented in [31].

2.1.4 Magnetic Microactuating Systems

Researchers started developing magnetic microvalves for microfluidics more than 30 years ago, so they invented a variety of methods for actuating them. Actuating schemes involve the use of large and small electromagnetic coils as well as either stationary or movable permanent magnets [32, 1, 33, 34, 35, 36, 37, 38, 39, 12]. The method developed by Cao et al. manipulates magnetic microfluidic particles by applying an external magnetic field using a permanent magnet [36].

Actuating strategies using either permanent magnets or electromagnetic coils are not unique to microvalves, in fact, many magnetic microfluidic devices use one of these control schemes. In some cases, the magnetic field is generated through electromagnetic induction,

but not necessarily by winding conductors in the shape of a coil. MEMS traps are made by shaping conducting materials in a geometry that would induce the desired magnetic field when current is applies [26]. As discussed, ferrofluid is used in microfluidics for a variety of purposes such as pumps and valves. Melikhov et al. investigated a method for actuating the ferrofluid by energizing a specific set of conductors [40]. Similarly, researchers have also fabricated specific micromagnetic systems for particular applications [31]. Whereas these custom actuation schemes work well for the device they are intended to actuate, they do not necessarily adapt well to actuating other microdevices.

In some studies, magnetic valves are actuated by physically moving a permanent magnet [1, 32, 33]. This works well for the flow-control valves such as the ones designed by Rahbar et al. and the one designed by Cheng et al. because it allows for easy manual manipulation [1] [33]. In particular, such a method is ideal for the target drug delivery application of Cheng et al. because their proposed implant does not need any power source to control and it can be actuated by applying an external magnetic field through the skin [33]. Some actuation systems use large permanent magnets because they need to generate large magnetic fields, like the large 2.8 T field required to drive the ferrofluidic devices in [22]. Precision in the generated magnetic field is another good reason to rely on permanent magnets [23]. However, while this control scheme is fast and reliable, the disadvantage is that it is hard to automate. In some cases some external actuator, such as a motor needs to have magnets mounted on it and that is how the movement in the magnetic field is created [18].

Some actuating systems rely solely on electromagnetic coils such as the one employed by Hilbich et al. [14]. Jian et al. actuates their magnetic microvalve by moving magnetic fluid to block passageways using conveniently placed coils [35]. Lee et al. control magnetotatic bacteria through the use of electromagnetic coils [28]. The system studied by Shinozawa et al. is an early combination of a coil and a very small permanent magnet, although the magnet's purpose is simply to enhance the field of the coil [39]. Coils can be used to allow for flexibility in term of actuating individual devices, as wells as in being able to be built-in to the system when fabricating the microdevices, such as in [29]. The disadvantage of the coil-only solution is that current needs to be applied at all times when you require a magnetic field present, and if the valve itself does not have some sort of latching technology to keep it in one state after the field is removed, a high amount of power is used overall.

Bohm et al. present a bi-stable magnetic actuator that operates on the same principle as commercially available relays and linear motors: a spring-based iron armature sits inside of a large electromagnetic coil with a permanent magnet on top of it [34]. The magnet holds the armature in the open position, and to close the valve, a current runs through the coils to generate a magnetic field canceling the field of the magnet, thus causing the armature to spring down and close the valve [34]. The bi-stability is accomplished by the armature remaining in the closed position after being forced there by the spring even after current is no longer running through the coil [34]. To re-open the valve, a current in the opposite direction is applied to the coil and attracts the armature back in its place to be held by the magnet again [34]. Gray et al. similarly present a low-power actuator by combining a permanent and uniform external field with locally placed coils that selectively add or cancel the external field to achieve the local actuation [37]. Zhi et al. study the details of how to make small planar coils for use in this kinds of microactuators [38]. Østergaard et al. used a set of two electromagnets (for alignment and launch) and one permanent magnet (for compensation) to control the movement of their magnetic microparticles [25]. A combination of permanent magnets and coils is similarly used by Bernstein et al. to actuate their micro mirror array [30]. In this case, an array of specifically arranged micromagnets provide the required field gradient to allow the electromagnetic coils embedded in the micromirror array to position the mirrors [30].

Overall, actuation solutions that use both permanent magnets and electromagnets are far less common than simply using just permanent magnets or electromagnetic coils. This may be because it is harder to design such a system since it has more parameters to test for. Bi-stability is important for a low power application, such as the µROAMS collaborative project; so this thesis will present a similar hybrid magnet and coil actuating system that is tailored to work and be adjusted specifically for the magnetic microvalves in [1]. This system can be placed in arrays and forms one of the main goals of the thesis. This actuation system is a hybrid magnet and coil system in two senses of the word: it can use both a magnet and a coil to allow for easy control and high forces, but it can also be implemented with only the magnetic coil if the device that needs to be actuated has a low enough magnetic field requirement, as is the case for the magnetic mixer discussed in the next section.

2.2 Magnets, Inductors and Electromagnets

Magnets are pieces of magnetized metal that have a permanent magnetic field based on their magnetization, whereas inductors and electromagnets are made up of tightly wound wire that generates a magnetic field when current is run through them. Magnetic fields generated by steady currents are calculated using the Biot-Savart Law:

$$\mathbf{B}(\mathbf{r}) = \frac{\mu_0}{4\pi} \int \frac{\mathbf{I} d\ell \times \mathbf{r}}{\mathbf{r^2}} \tag{2.4}$$

In this equation, B is the vector representing the magnetic field, μ_0 is the permeability of free space, I is the current going through the wire and the integral is taken in terms of the current path, with r representing the vector from the origin of the coordinates to the point of interest. For magnetic fields generated by a magnetization in matter, the magnetization generates a surface current:

$$\mathbf{K_b} = \mathbf{M} \times \hat{\mathbf{n}}, \tag{2.5}$$

where K is the vector representing the surface current, M is the magnetization of the material and n is the normal unit vector to the surface. From this equation, the Biot-Savart law can be appropriated and used to calculate the overall magnetic field.

For example, these equations can be used to calculate that the magnetic field inside of an infinitely long cylinder as:

$$\mathbf{B} = \mu_0 \mathbf{M} \tag{2.6}$$

Using Biot-Savart law to calculate the magnetic field inside of an infinite solenoid results in an equations showing it to be finite:

$$\mathbf{B} = \mu_0 \mathbf{n} \mathbf{I} \hat{\mathbf{z}} \tag{2.7}$$

n is the number of turns and I the current through the wire, while z simply indicates that the magnetic field points in the direction of the z axis (height of the solenoid), thus it is apparent that this is comparable to the equation for the previously mentioned infinitely long cylinder if the magnetization of the cylinder is in the z direction.

However, real physical systems do not have infinitely long magnets or coils (i.e. solenoids), and the magnetic field is harder to calculate for cylinders of finite height. If the magnetic field along the z axis is solely of interest, this turns into a much more reasonable equation:

$$B = \frac{B_0}{2}\left\{\frac{z+l/2}{r^2+\left(\sqrt{(}z+l/2\right)^2)} - \frac{z-l/2}{\sqrt{(}r^2+(z-l/2)^2)}\right\}, \tag{2.8}$$

where B_0 represents the magnetic field at the very centre of the cylinder, l represents the length of the cylinder and r is the radius of the cylinder. This equation is in terms of the field at the centre of the cylinder and not the magnetization of a permanent magnet or number of turns of a coil, thus it can apply to both as it really is only about the behaviour of a magnetic field in a given geometry. For example, in a cylindrical magnet (assuming it has a typical magnetization along the height), the field is parallel to the height on the inside of the magnet, and it curves around on the outside from the north to the south pole, as seen in the Figure 2.2.

The force that a magnetic field exerts on an infinitesimally small magnetic dipole of moment m is:

$$\mathbf{F} = \nabla \mathbf{m} \cdot \mathbf{B} \tag{2.9}$$

For example, the magnetic force between two very large magnetized surfaces that are relatively close together is given by the formula:

$$F = \frac{B_1 B_2 A}{2} \tag{2.10}$$

This is only really accurate if the distance between the two surfaces is much smaller than the height and area of the surfaces. Appropriating such an equation to calculate the force between two cylindrical magnets is more complicated.

2.2.1 Ferromagnetism

Ferromagnetic materials, such as iron and rare earth magnets, are defined to be materials with a large magnetic susceptibility, i.e. their ability to become magnetized in response to an external applied magnetic field [5]. Electrons that are not paired in filled electron

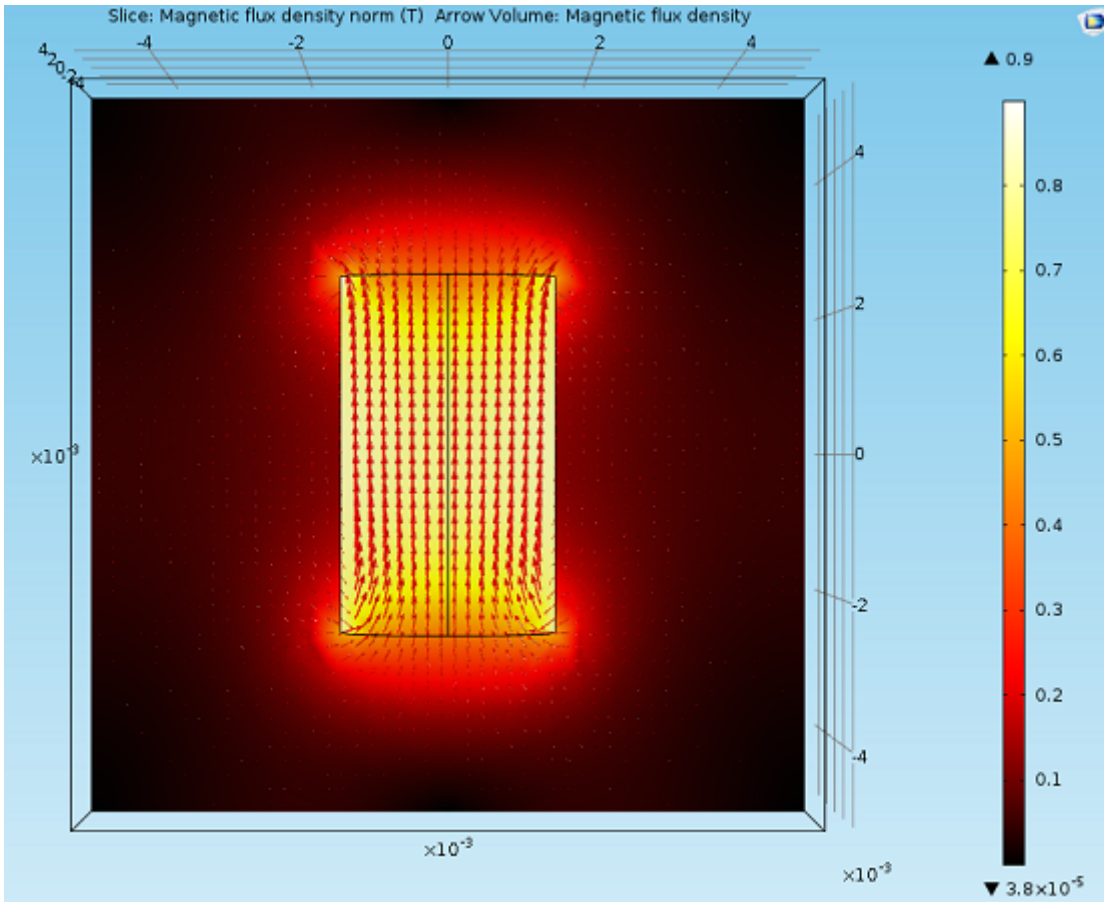

Figure 2.2: Magnetic field generated by a cylindrical magnet

shells (as are found in many metals) have a natural dipole moment. In magnetic materials, they can spontaneously align to an external magnetic field and thus, the material becomes magnetized, at least temporarily. If the external magnetic field is removed, ferromagnetic materials tend to stay magnetized at a certain percentage of their full magnetization potential. This ability to behave in accordance to the magnetic fields experienced in the past is called hysteresis.

Ferromagnetic materials are divided into magnetic domains. Each of these domains has their electron's magnetic dipole moment aligned in a particular direction. In this case, the overall field of the material is very weak or non-existent because the different orientations of the domains cancel each other out. A strong external field can align all the domains, thus causing the magnet to have a stronger overall magnetic field.

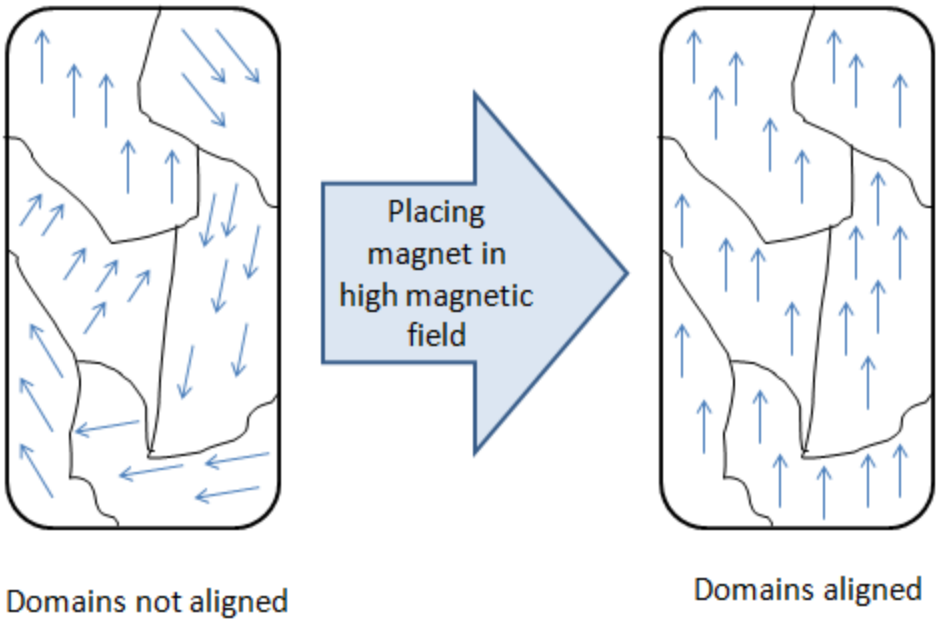

Figure 2.3: Domains being aligned by placing the material in a strong uniform magnetic field

2.2.2 Ferromagnetic Core Inductors

The type of core that an inductor or electromagnet is wound around has a big impact on the resulting magnetic field. Having a core made out of a ferromagnetic material, such as iron or iron alloy strengthens the observed magnetic field by a considerable amount. This happens because the core draws in the field while concentrating it on itself (by aligning the domains inside the ferromagnetic material) [41]. Different material choices for the core have different effects on the field because of the permeability of the ferromagnetic material (i.e. the ease with which it becomes magnetized under an external field) [41]. Cores made out of iron, nickel and steel alloys as well as soft ferromagnetic materials are the most effective, with the more complex alloys being the best, but a simple iron core is the cheapest [41].

2.2.3 Inductance and Time Constant

Inductance is a measure of how easily a current running through the inductor creates a voltage in the inductor. The time constant of an inductive circuit comprised of an inductive element of inductance L and a resistive element of resistance R is RL. It takes about five

time constants from the time the circuit is connected to power to the time the inductor is fully charged.

The inductance of a coil is determined by:

$$L = \frac{\mu_0 \mu_r A N^2}{l}, \tag{2.11}$$

where μ_r is the permeability of the material used as the core of the coil, N is the number of turns of the coil, A is the cross-sectional are and l is the length.

2.3 Proposed µROAMS Microfluidics Platform

The goal of the µROAMS project is to design a reconfigurable microfluidics platform that provides faster testing time by providing high-throughput, multiplexed sample analysis. The µROAMS project not only aims to create a microfluidic device for chemical and biological testing, but also to create the associated CAD tools for designing and using the system. The main difference between this device and other microfluidics devices is that it can be customized and reconfigured to perform a variety of tests. This way, the need for many different machines to perform specific tests is avoided, and space on the microfluidics layer of the system itself is saved by re-using the same multiplexed fluidic paths with a variety of sensors. In other words, this platform will perform microfluidics testing much in the way a FPGA performs computing in hardware.

The key components that will allow µROAMS to accomplish such a task are: a robust set of CAD tools and the easy-to-control magnetic microcomponents. There are two sets of CAD tools in development for this project: system design CAD tools and on-system CAD tools. The system design tools will assist researchers in choosing microfluidic components, placing these components on the microfluidics layer and placing the microchannels and interconnects between the components, as well as perform other tasks needed in creating a device, such as generating masks for fabrication processes. These tools will be used for prototyping and designing the physical µROAMS device, but they may also be potentially used for the design of other microfluidic devices, whether they are reconfigurable or not.

The proposed physical µROAMS device has more than simply magnetic microcomponents. In fact, the microfluidic flow-control components are only a single layer of the overall

device. This layer is comprised of the microfluidic channels and chambers used for routing the fluid across the chip and storing it. Valves are built into the intersections of the channels to allow the path of the fluid to be controlled. The valves are on another layer from the normal routing channels, as is discussed in the section on the magnetic microvalves used in this project. Yet another layer is comprised of the magnet/electromagnet actuation devices used to interact with the valve layer. An electronics control layer is used to provide the electrical signals that drive the electromagnets and other components and the user will be able to obtain results from a sensor read-out module (perhaps a display built in to the computer control board).

In addition to these layers, the µROAMS device will have modules that can be swapped in and out for different purposes. A sensor module containing various chemical and biological sensors (eg. temperature sensors, quantitative real-time Polymerase Chain Reaction (qPCR) sensors, etc.) can be swapped in as appropriate. A module can also be used for the fluidic inputs into the chip, which will depend on the kinds of tests being carried out. Currently, there is interest in the environmental testing potential of µROAMS. An industrial partner, Environmental Bio-detection Products, Inc., provided samples and advice for use in developing sensors.

As stated, µROAMS is still in an early prototyping stage: individual components are fabricated but the interfacing between components is still in early development. Currently, valves, mixers, some sensors and microfluidic channels and chambers are developed and ready to be integrated into the system and tested. The valves provide a very effective seal when closed to ensure samples are not lost through leakage when routed through many intersections.

The CAD tools and user interface are in development. The fluidic routing tool will be an important aspect of keeping the use of the system optimized at run-time. This, in conjunction with a user interface to control the microfluidics system, as well as device libraries that allow for the easy interface with the actuators controlling the physical components, make an easy-to-use and robust system. An FPGA is used for early prototyping of the computing, software, and interconnecting components.

For early prototyping, a syringe pump is used to provide the driving pressure in the system. A syringe pump operates by providing a steady input flow rate into the microchannels. As the amount of fluid it pumps into the system increases, if the fluid does not have

an outlet at the current pressure, the pressure builds up until a pathway gives into the pressure and open up. In order to protect the system from high pressure build-up that may be caused by blocked pathways, a safety release valve near the fluid pump is used.

2.3.1 FPGAs

An FPGA is an integrated circuit that can be reconfigured to perform various hardware tasks. An FPGA has a large array of logic cells connected by programmable interconnects, thus the desired circuit is created on the device by selecting appropriate interconnects between the logic elements. Current FPGAs not only have large structures of logic blocks, but also contain dedicated multipliers, digital signal processing blocks and different sizes of memory cells, distributed in various patterns across the chips. A hardware designer specifies the details of their design in a hardware description language, and manufacturer-provided tools then synthesize that and turn it into a bitstream with which the device is programmed. Some FPGAs also allow run-time partial reconfiguration, allowing the hardware instantiated in the FPGA to be partially (only some of the hardware instances on the device) changed while the system is still running.

FPGAs are a good fit for the µROAMS project because they allow flexibility in terms of how to structure the computing components of the device that a microcontroller does not provide. An FPGA, gives the flexibility to run the code, be it the user interface, control logic or run-time routing CAD tools either in software of hardware. When dealing with very time-critical computations or control logic, the option to run it in hardware is very useful for making a system that runs smoothly.

FPGA I/O

An FPGA typically has a large number of Input/Output (I/O) pins, which makes it ideal for the µROAMS project because it allows dedicated I/O pins to control each valve, even for a relatively large array of valves. For example, one of the newest FPGAs families designed by Xilinx, the Virtex 7 has between 300 and 1200 I/O pins depending on the actual device model [42].

The actual voltage and current that the I/Os can provide depends on the standards that they support. In the past, it was common for I/Os to be able to provide 5 V, 3.3 V

and 2.5 V, but the new devices, such as the aforementioned Virtex 7, contain 3.3 V high range (HR) I/O banks that support an absolute maximum voltage range between -0.5 and 3.6 V and the 1.8 V high performance (HP) I/O banks that support an absolute maximum voltage range between -0.5 and 2.0 V [43]. The I/Os can output a maximum of 16 mA in the HP drives and 24 mA in the HR drives [43]. This means that unless the various actuators for the microfluidic devices only need a very small amount of current, an external power source that powers the magnetic coils to actuate the valves and mixers needs to be used. This can be accomplished by a battery, or the final device may have to be plugged in to a larger power supply (such as a car battery if the device is used in the field), depending on the total power estimate that is discussed in a later chapter.

For the µROAMS project, a Virtex 7 FPGA would not be ideal because of their power consumption. Additionally, the Virtex 7 FPGA is a large, powerful device, with much more computing power than is needed for this project. A smaller, more power efficient FPGA, perhaps one of the lower end ones from Xilinx's 28 nm or 20 nm Kintex FPGAs would be more suitable [44].

2.3.2 The Magnetic Microvalves used in µROAMS

The valves being actuated in this project are based on flexible polymers doped with rare earth magnetic materials [1]. These valves are ideal because Rahbar et al. designed and characterized their behaviour as well as demonstrate their potential for mass production and their ease of control and strength of forces they can withstand [5] [1]. The valve itself employs a simple design being made up of 4 parts: a valve chamber, a magnetic flap, an inlet and an outlet as seen in Figure 2.4 [1].

The valve itself in this configuration is normally open because the flap on its own does not provide significant resistance to the incoming fluidic pressure. If a magnetic field is applied by a magnet placed underneath the flap, its polarization aligned to the magnetization of the flap, this creates a downwards force on the flap, allowing the flap to withstand incoming fluid pressure. The amount of fluidic pressure that the flap can withstand depends on the strength of the applied magnetic field, the height of the magnetic flap (if the flap is smaller, there are fewer magnetic particles within it to interact with the magnetic field) and the doping level of the flap material with magnetic particles [1]. However, the last

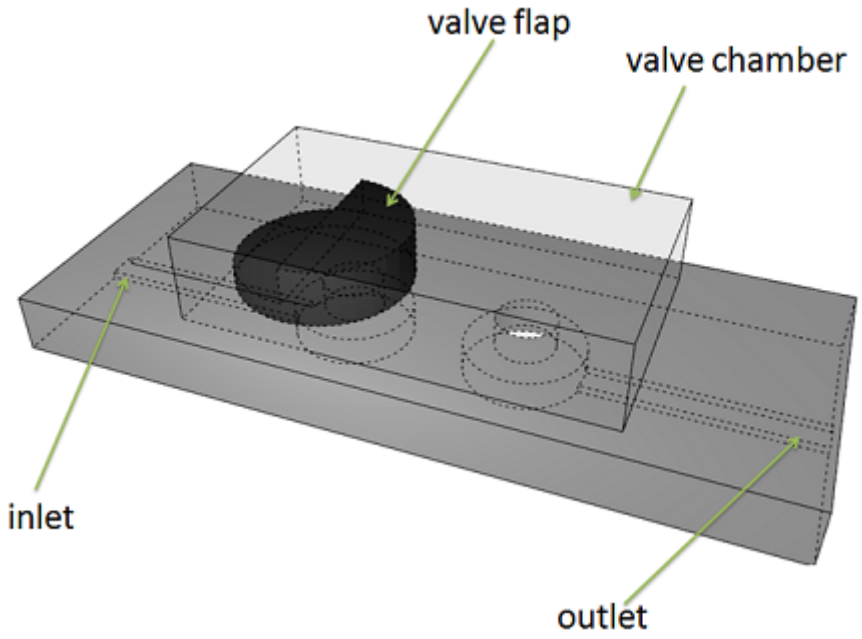

Figure 2.4: The magnetic microvalve used in µROAMS (figure adapted from design presented in [1])

parameter, the doping level of the flap, is less useful to vary since varying the height of the flap accomplishes the same result in an easier way and, in fact, the optimal doping level was found and used throughout [1].

As previously mentioned, the valve is actuated by applying a magnetic field aligned with the direction of fluid flow through the valve in order for the attraction between the magnet and the flap to provide the force necessary to withstand fluidic pressure. This field is applied by a cylindrical rare earth magnet magnetized along its z axis. The magnet needs to be aligned with the flap, thus it needs to be placed either underneath or above the flap. The magnet is placed underneath for ease of construction and actuation. The magnet is some distance away from the flap depending on the thickness of the substrate, therefore the magnetic field at the flap is less than the magnetic field at the surface of the magnet. When this thesis mentions a certain amount of magnetic field applied to the flap, it is referring to the amount of field as it would be measured at the flap, in the z direction. For example, if the valve is fabricated using the process designed by Rahbar et al. with a 2.3 mm tall flap, it can withstand up to 9.65 kPa of incoming fluid pressure without any leakage when an 80 mT field is applied to the flap directly beneath the flap [1]. Rahbar et al. demonstrated this using a 7 mm diameter rare earth magnet to generate the field [1].

In order to use the valve to direct fluid through the channels, it is fabricated into a single-input, multiple-output configuration, as shown in Figure 2.5 [1]. Any pathway can be selected, and the valves along this pathway can be placed in the correct state, since each flap is individually controlled.

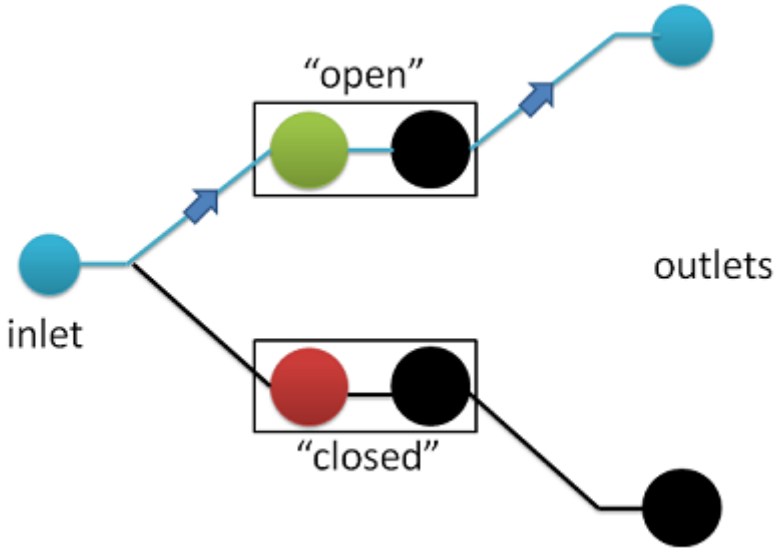

Figure 2.5: Routing the fluid using the valves (image adapted from figure presented in [1])

2.3.3 The Magnetic Mixers used in µROAMS

Mixing is an integral part of many chemical processes, thus a reconfigurable microfluidics platform that performs various chemical tests requires mixers. For that purpose, Rahbar et al. designed a cilia based magnetic mixer [2]. These cilia are fabricated using a new micromolding technique that achieves high aspect ratio while being easy to scale for large volume production [2]. Also, the fabrication process is highly flexible and easily allows for any size and shape of the mixing chamber as well as any number of cilia to be placed inside the chamber. That is because the chamber is made out of PDMS and patterned using conventional patterning techniques, and the cilia themselves are formed by shaping a mold using by inserting a microneedle then removing it, creating a vaccum that draws in the M-CP [2].

The cilia are about 2 mm tall with 0.13 mm diameter and unlike the magnetic microvalves, they do not require high magnetic fields to operate: when actuated with a field of 7 mT they are able to achieve 85% mixing in a chamber after 3.5 minutes [2].

The cilia are easily actuated using an electromagnet or a magnetic coil by placing the magnet or coil underneath the cilia as shown in Figure 2.6. If the coil is connected to an alternating current source (either a square or sinusoidal wave) the cilia move by bending back and forth in response to the changing magnetic field. They are magnetized in order to achieve this behaviour (an unmagnetized piece of metal is attracted to magnetic fields, it cannot be repelled by a magnetic field) [2].

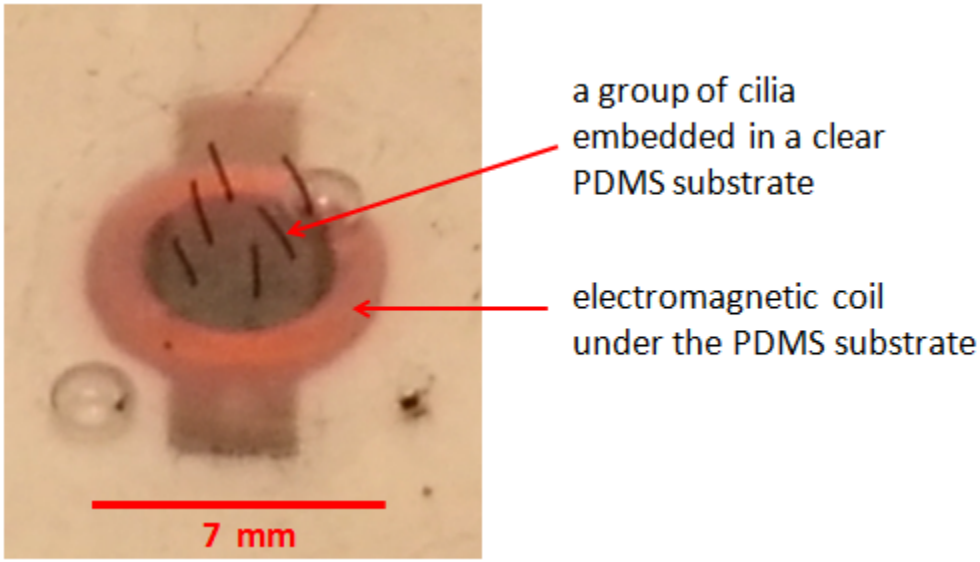

Figure 2.6: Magnetic cilia being actuated by a 7 mm electromagnetic coil (picture similar to design presented in [2])

The strongest factor influencing the mixing speed of the cilia mixer is the displacement of the cilia (how much they bent in either direction) which is in turn affected by the aspect ratio of the cilia as well as the applied magnetic field. Interestingly, the speed of the mixing, determined by the frequency of the alternating current has almost no effect as long as the cilia are in motion [2].

2.3.4 Device Drivers and the Overall System

In the field of computing, a device driver is the name for a piece of code or program that interfaces with actual hardware devices. Thus, device drivers act as bridges between the operating system of a computing to the hardware components. Device drivers also allow for the abstraction of the hardware functionality from the software that might be using it. This is important because it allows individuals who are not experts on the details of the hardware device to be able to use it as if it is a black box. The abstraction is also important for security reasons, to help prevent any rogue program from randomly accessing the hardware components.

For example, if someone writes software for an embedded system that incorporated a camera, and there are no device drivers for the camera, it means that they have to access the pins connecting to the camera directly and need to know specifically what each pin does and how to drive it in order to use the camera in the desired way. The pins are manipulated directly without the abstraction provided by an operating system controlling the camera through a device driver, thus it is possible for other programs to access the pins directly.

The goal of µROAMS is to provide a full microfluidics system complete with CAD tools for determining the optimum way to design the system as well as the optimum way to route fluid across the microfluidics layer while the system is operational. The latter will be operating at run-time on a processor incorporated in the device. Therefore, it is vital to have as many modular and easy-to-use components as possible. Utilizing device drivers for operating the physical components is important because the routing algorithm can easily make use of them and control the valves. The software aspect of the project must employ a general structure for the usability perspective of other researchers who may be interested in using these tools for their own system design. This procedure also enables easy expansion of the software to incorporate new components, which is important for future developments in the project. Since the system is at an early prototyping stage, it is not yet clear that the software component will need to run on a processor with an operating system. Thus, for now, device libraries are coded instead of device drivers to contain the functions for controlling the components. If an operating system is later found to be the appropriate choice for the final system, then the device libraries will be expanded into device drivers.

Researchers have been considering how to build a fully integrated and reconfigurable microfluidics device for over a decade [45]. However, taking the steps to physically build such a system is a more recent advance. In particular, recently, researchers such as McDaniel et al. are applying CAD algorithms, such as simulated annealing, that are commonly used in the design of computer chips, to the design of microfluidics chips [46]. Many such papers only focus on a very specific problem for designing the chip, such as the layer of routing interconnects [47]. So far, an actual structural software framework for controlling these kinds of chips has not yet been developed.

Chapter 3

Proposed Physical Actuation Solution

As stated, one of the objectives of this thesis is to create a magnetic actuation scheme suitable for operating a variety of magnetic microcomponents as described in the background chapter. Many of these components, such as the valves designed by Rahbar et al., the pumps designed by Hatch et al. and Pan et al. require large attraction forces that can only be supplied by using permanent magnets [1] [19] [21]. In order to actuate systems such as these, permanent magnets must be used, but in order for the system to be easily controlled by a software component, electromagnetic coils are more desirable. Thus, a hybrid solution is ideal. The proposed hybrid solution is highly flexible and easily adaptable to the device that needs to be actuated. If high magnetic fields are required, the system can use big magnets to provide these fields, however, if low magnetic fields suffice, the actuating system can work with simply the magnetic coils. For this project, the magnetic valves are actuated using a magnet and coil scheme, whereas the magnetic cilia are actuated with only the electromagnetic coils.

This chapter describes the methodology used to design the various components of the physical control system. It discussed the various design parameters and how they affect the overall system. In particular, the procedure for designing the microvalve actuation system is detailed.

As discussed in the Chapter 2, actuation techniques for controlling magnetic components use either permanent magnets because of the large steady fields they produce, or

electromagnetic coils because of the ability to vary the magnetic field. The main benefit of using an electromagnetic coil rather than a permanent magnet is that the magnetic field generated by the coil is easily adjusted by varying the current through the coil, and even allows the direction of the magnetic field to be reversed. The disadvantage of using the coils is, of course, that they cannot provide as powerful a magnetic field as a permanent magnet and they require continuous power to achieve this magnetic field.

The actuating system presented in this thesis is a hybrid solution using both permanent magnets and electromagnetic coils in order to achieve better power efficiency as well as easier integration into a digital system. Using this implementation, a bi-stable system is achieved: a latching solution despite the fact that the valves themselves do not have latching capabilities on their own.

3.1 Microvalve Control System Latching Design

The biggest stipulation of the µROAMS project is that the final device should be portable, therefore, each subsystem must be designed to be as power-efficient as possible. A latching mechanism must be employed in order to achieve the power-efficiency goal for the microvalve control. Specifically, power should only be needed to switch the valve between its "open" and "closed" states, not continuously to keep the valve in either state. The microvalves used for this project, as described in the background section, do not themselves employ any latching mechanism [1], so this must be accomplished in the actuating system itself. Permanent magnets provide a steady field and do not require power to maintain this field, so it comes to reason that they should be used to maintain steady states. Of course, functionality must be implemented to change between the states, which can be accomplished through the use of appropriate electromagnetic coils.

In particular, the permanent magnet and electromagnetic coil are structured as follows: the permanent magnets apply the field needed to keep the valves in their closed state. In order to switch to an "open" state, current is run through an electromagnetic coil located below the magnet as shown in Figure 3.1, and it attracts the magnet towards it, thus lowering the magnetic field that the valve flap experiences and allowing the valve to be in an "open" state.

The latching is done through a small piece of magnetic material located at the coil. This keeps the magnet in place in the OPEN position even when the current is no longer applied to the electromagnetic coil. The latching piece is chosen to be a cylindrical piece of the same magnetic polymer used in the valve flaps for the simplicity of simulation and to allow the use of the magnetic field formulas discussed.

Of course, the primary concern of selecting the latching piece is to prevent the magnet from being attracted back to the valve flap. This is dependent on how thick the valve flap is and how far away the magnet is attracted to in the OPEN state. In particular, the distance between the latching area and the valve flap is the most important of these parameters as the magnetic field drops significantly over distance. Another important factor in determining the size of the latching piece is that the stronger the attraction between this piece and the magnet, the more resilient the system is to latching failure caused by ambient magnetic fields or simply from an acceleration force caused by sudden movement.

The downside of a strong attraction between the latching piece and the magnet is that it requires the coil to assert a stronger force in order to detach the magnet and force it back to the CLOSED position attracted to the valve flap. The most important aspect of this control system is the power consumption, thus the optimal way to select the latching piece is to optimize the system for the other parameters, determine the current needed to attract the magnet given those parameters, and select the latching piece such that the same amount of current (in the other direction, of course) can be used reliably to detach the magnet from the latching piece. If this does not actually fulfill a design requirement, then the parameters of the system need to be re-evaluated. With this approach, the strongest source of latching possible is achieved without actually increasing the power consumption due to the latching mechanism. A later section presents the details of this, and calculations on how to determine the strength of the latching.

In Figure 3.1, the valve on the right is in an OPEN state while the valve on the left is in a CLOSED state with the magnet applying the magnetic field required to keep it closed. The magnet itself is situated inside of a cylindrical channel that is just wide enough to allow easy up and down motion without friction being a big problem. This magnet channel keeps the magnet in place and helps reduce the amount of cross-talk since there is less freedom of motion.

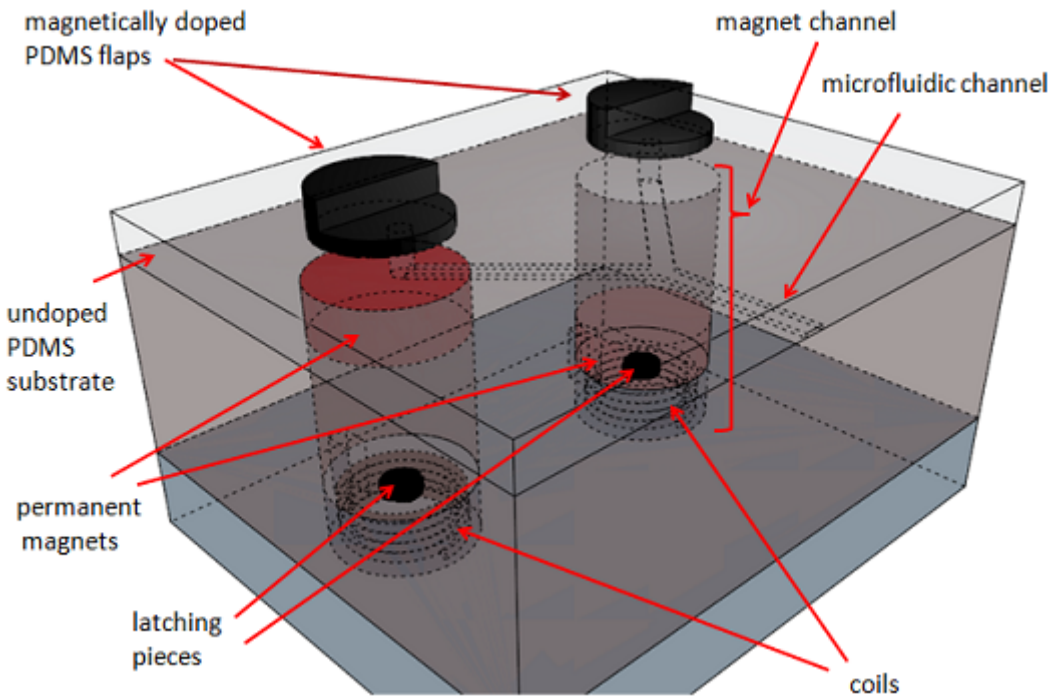

Figure 3.1: The valve control mechanism

This control mechanism is easy to integrate with FPGA control since the electromagnetic coils are easily controlled through digital means. Although, in the current system, the power required to drive the coils is more than a single FPGA I/O pin can provide, but this is solved by using the FPGA I/Os to control the source of the current to the coils. The solution can be easily scaled to a larger array of valves since each valve has its own actuating system, as long as the valves are far enough apart that cross-talk between the magnets is not a problem.

This actuation scheme can be used to drive many of the devices described in the background chapter, although adjustments would have to be made to be for each of them. If a device is used such that the magnet has nothing to latch to at the top of the magnet channel, an additional latching piece can be added there in order to achieve bi-stability. For certain other cases, such as ferrofluids requiring very high magnetic field or traps requiring very small electromagnetic coils, this actuating solution would not be ideal [22] [30]. This thesis will focus on how to optimize the system parameters to achieve the desired effect in terms of control and power usage. In particular, optimizing the control solution for the valves de-

signed by Rahbar et al. is given particular focus, since these are the magnetic microdevices that are readily available for testing [1].

3.2 Design Considerations

In order to create an appropriate control system for µROAMS, the relevant overall parameters of the final design must be considered. Specifically, parameters to remember are: the power consumption, the size, the susceptibility to outside magnetic fields, the magnetic field profile, as well as the heat dissipated by the actuator and valve system.

Preceding sections already described the project's need for a low power solution. The overall size of the design is important because a large array of channels and valves must fit on a small device, thus the valve and actuator need to be miniaturized as much as possible. The magnetic profile of the actuator and valve system as well as the system's susceptibility to outside magnetic fields are important because it determines how they interact with other magnetic components of the system (or other valve and actuator pairs). Potential cross-talk between the magnets of adjacent valves is something that must be examined to determine what the safe distance between them is. Also, the system uses strong permanent magnets, thus the magnetic field of the whole system must be monitored for the potential effects it may have on other magnetic components such as the magnetic cilia, or on the electronic components. Lastly, the electromagnetic coils are likely to generate heat when turned on, especially if they need to be on for a long duration or if they need to be switched frequently. The impact of this heating on the system must be investigated, and potential cooling solutions need to be determined.

In order to design the control solution, the parameters that impact the system must be identified and then tested to determine the exact extent of their impact.

3.3 Parameters to Test and Optimize

In order to achieve design optimization, the relevant parameters and combinations thereof must be tested to observe how they affect the power consumption of the system. The only power source in the coil actuation system is the battery/power supply that is connected to the coil, therefore the power that this supply needs to provide should be minimized. Power

is calculated as the product of voltage and current, so the power used up by an actuator and valve pair can be measured as the power supplied by its battery/power supply by reading the input current and voltage. Thus to optimize power, the parameters that affect the needed current and voltage need to be determined and examined.

Current is directly proportional to the needed magnetic field, so the stronger the magnetic field needs to be, the more current has to be applied. Of course, the amount of magnetic field generated by the current is dependent on the electromagnetic coil used. The voltage needed to supply this current is also dependent on the internal resistance of the electromagnetic coil used. Thus, the power consumption, assuming the power source is directly connected to the coil, is dependent on the coil used and magnetic field required.

The parameters to test and optimize for this control scheme are:

-the magnets (type and dimensions)

-the electromagnetic coils (size, type of wire, number of turns, type of core)

-the height of the magnet channel

-the distance between the valves

-the amount of current

-the thickness of the valve flap

-the height of the PDMS substrate

-the fluidic pressure to run the system at

-fluidic pressure difference between the open and closed state

These parameters are all interdependent. The magnet must be selected such that it provides enough magnetic field to keep the valve closed, but the amount of field required is itself dependent on the fluidic pressure in the channels as well as the thickness of the valve flap. The strength of the magnet used also affects the minimum distance between the valves since stronger magnets have a larger area of influence. The height of the magnet channel is dependent on the strength of the magnet and how much the field at the flap must be decreased before it is considered to be in an OPEN state. In turn, the height of the magnet channel affects the magnetic field needed from the electromagnetic coils to attract the magnet across that distance.

Chapter 4

Proposed Software Solution

While the previous chapter presents the design considerations for the valve control system, this chapter presents the overall design considerations for interfacing with physical components of the system such as the existing mixer mentioned in the background section, or general sensors. In particular, this chapter presents the proposed software solution including the device libraries and user interface.

4.1 Interfacing with Other Components

When designing a user interface for a component, the starting point is listing all the user parameters for that component, i.e. all the variables that may need to be changed regarding the operation of this component. For example, for the existing magnetic cilia mixer developed by Rahbar et al., the user parameters to be as follows [2]:

-the speed of the cilia motion

-the deflection of the cilia

-the amount of time to mix

These parameters are controlled by the frequency of the oscillating magnetic field, the strength of the magnetic field, and the time the magnetic field is applied for, respectively.

These user parameters are specifically for the magnetic cilia mixer. A much more complex mixer, for example, has a built-in sensor that determines the progress of the mixing and feeds this information back to the control, or a mixer that needs to be mixing for a pre-determined amount of time with little tolerance. Suppose a mixer is built into a chamber with many inputs and outputs that need to be controlled separately. Conversely, there

can exist a much more basic mixer that has no other parameters that whether the mixer is on or off.

In order to be efficient and easily integrate different kinds of mixers, the basic control for a mixer should be defined: MIXER ON and MIXER OFF. All mixers have at least those basic controls. On top of those basic control, specific control options depend on the type of mixer being used.

On top of this, it may be necessary to keep track of and consider the physical properties of the specific mixer being used. For example, suppose that the magnetic cilia mixers for this device are distributed on the device in a few different configurations: some of them have three cilia while others have six. In this case, it is useful to know, for the specific mixer currently in use, how many cilia are present and how much time it takes to fully mix the sample. Other potential parameters are the length of the cilia and the dimensions of the chamber. The library should have functions for calculating useful data, such as the estimated time to fully mix the solution based on the specific mixer used and the fluids that need mixing.

4.1.1 Device Libraries and Device Class

As discussed in the background chapter, device drivers should be used to interact with hardware devices in order to abstract the details of the interface implementation to the user of the device when an operating system is used. Since the current prototype is not using an operating system, device libraries will be used to accomplish the task of making the software modular, scalable and easy to use.

In a computing system, there are different classes of device drivers depending on the kind of interface the device uses. For example, there are audio class devices like speakers and headphones, input type devices like a mouse and keyboard, etc. This allows for using a set procedure to connect to that interface rather than having each device driver define it itself.

Obviously, such a classification works well for computer device drivers because the kinds of interfaces available on a computer are standardized and either specifically designed for certain tasks, like High-Definition Multimedia Interface (HDMI), or very broadly used but bound by a tight protocol, like Universal Serial Bus (USB). There are no standardized inter-

faces like the aforementioned HDMI or communication protocols like USB for microfluidic components, thus choosing to group them by the kind of interface they use does not seem reasonable. Types of interfaces can be defined based on the number and kinds of pins they are connected to on the FPGA, but this does not seem in any way efficient as the interface needs of each microfluidic device may be quite different leading to many different classes.

A better procedure for defining classes for microfluidic components is to group them by their function in the system. As previously discussed, mixers can be grouped into a class that describes the basic interface with a mixer-type microfluidic object. Similar device classes can be made for valves, pumps, heaters sensors, etc. This way, the overall procedure for using the device will be described by the device class rather than the specific implementation. For specific device libraries, the requirements on the output pins can be specified as part of the library.

This way of classification and specification allows for the inclusion of components that have not been fabricated yet and make placeholders for them in the control code as well as the CAD tools. The current versions of components are incorporated into this system, but it allows for variations or new versions as the technology develops. Thus, a component repository is created, with each device file specifying details about how to interface with the component which is required for the user interface and routing tools. The device files also specify details that are useful for other phases of the synthesis flow for the project. They contain details about the specifics of operation of the devices to let the high level synthesis tools select the appropriate components for their overall specifications, but also size and placement requirements that will be used by the place and route tools to lay out the microfluidics layer and maybe even by automatic mask generator tools.

4.2 FPGA Control Scheme

In order to connect everything together and to test and use the overall system, a software solution for the user interface needs to allow input to be given to the device and output to be processed. Not many components are yet available for using and testing, thus the current implemented interface is quite simple and allows for the system integration to be tested. It can easily be expanded in the future since basic device libraries for the different classes of components have been written, making it easy to interface with them.

The end goal for the interface is to create a scalable and re-usable framework that can be easily incorporated into future versions of the project. This will be accomplished by being integrated into the suite of CAD tools being developed by others for this project. It will make use of the aforementioned microfluidic component repository to interface with the specific components.

The user interface runs in software on a processor on the FPGA. For the current implementation, it runs on a Microblaze on a Virtex 5 FPGA. This FPGA is used because, as previously described, the system is in a very early prototyping stage and the full computing and power requirements are not yet known. The Virtex 5 FPGA is convenient to use because it was readily available in the lab and its capabilities are known. For the actual device, or even a more advanced prototype, a more power efficient FPGA, such as one from Xilinx's Kintex Ultrascale series or, if we do not require the high number of outputs and hardware computing capabilities of an FPGA, then a microcontroller can be used [48].

4.2.1 Defining the Microfluidic Component Device Classes

This section elaborates on how the device classes for the microfluidic device libraries are defined. The device classes identified thus far are as follows:

- valve
- mixer
- pump
- heater/cooler
- sensor

The dimension information each of these devices for the placer is also stored for every device, this information does not need to be mentioned for each device individually. The following table shows the basic parameters and methods for each of the defined classes.

If the code is run on a FPGA with a C++ compatible processor, the device classes would be written as objects, with the associated device library functions written as methods. This way, specific kinds of devices are subclasses of the basic kind of device and their specific methods can overwrite the basic methods. But as it stands, they are written as structures with simple functions to regulate their operations.

Class	Parameters	Methods
valve	-valve open or closed	-open valve -close valve
mixer	-mixing on or off	-turn on mixer -turn off mixer
pump	-pumping on or off -pumping strength -pumping speed	-turn on pump -turn off pump -adjust parameter
heater/cooler	-heater on or off -desired temperature -current temperature	-turn on heater -turn off heater -heat to desired temp -adjust parameter -check current temperature
sensor	-what data to read -sensor data	-check sensor data -adjust parameters

Figure 4.1: The parameters and methods for the basic class typed

For example, here is the structure for a basic mixer and the magnetic cilia mixer used in this project:

```
struct MixerClass{
struct Device *thisDevice; // pointer to device class structure for this mixer

int typeOfMixer;        //specifies what kind of mixer this is
void* specificMixer;    //pointer to specific device structure

int GPIO_num;           //the GPIO number that drives this device
int GPIO_mask;          //the GPIO mask for connecting to this device

int MIXING;             // keeps track if the particular mixer is on or off
};

void MixerClass_ON(struct MixerClass *mixer_inst); // starts the mixing process
void MixerClass_OFF(struct MixerClass *mixer_inst); // stops the mixing process
struct MixerClass* MixerClass_newMixer(struct Device *thisDevice, int
    typeOfMixer, void* specificMixer, int GPIO_num, int GPIO_mask, int MIXING);
int MixerClass_SELF_TEST(int device_num);

struct MixerClass_MagneticCilia{
struct MixerClass *mixer;  // pointer to mixer class structure for this device
```

```
int num_cilia;              // additional info about the mixer
float cilia_height;         // specifies height of cilia used in this mixer
int mixing_strength;        // keeps track of the desired mixing strength
int mixing_speed;           // keeps track of the desired mixing speed
};

void MixerClass_MagneticCilia_setParams(struct MixerClass_MagneticCilia *
    cilia_inst, int mixing_speed, int mixing_strength);
void MixerClass_MagneticCilia_MIX(int parameters[NUMBER_PARAMETERS]);
void MixerClass_MagneticMixer_SELF_TEST(struct MixerClass_MagneticCilia *mixer);
```

As previously discussed, functions are needed to turn on and off a generic mixer, whereas the magnetic cilia mixer has more parameters that can be varied such as the strength and speed of the mixing, thus requiring a function to set those parameters. Self test functions are used to verify that a certain device is operating properly. Arrays of the devices of each device class are kept in system memory. The device itself is specified as a map detailing how the devices are connected together.

4.2.2 The User Interface

This section describes how the user interface is implemented. The current implementation is a strictly text-based interface without a visual Graphical User Interface (GUI). The final computing device that the control system will eventually run on for the prototype has not been chosen yet and the display capabilities of this potential device are not yet known, therefore a GUI implementation is too early at this time.

The initial functionality of the user interface is built and run on a Microblaze soft processor on the Virtex 5 ML505 evaluation platform board to test the interface [49]. The processor runs at 100 MHz and the code base takes up 95852 MB of space. It only requires a few microseconds to toggle the I/O pin that controls the output devices, but in the case of the valve, the output needs to be kept on for at least one or two second to generate the magnetic field and attract or repel the magnet, so in addition to this switching time, a clock needs to be set to determine how long the output needs to be on for, then an interrupt will need to be serviced when then time is up and then the output needs to be turned off. This adds additional computational requirements on the Microblaze processor, and additional

Micorblazes may be needed to handle all this in the case that too many valves need to be controlled quickly. Three main modes are provided for detailed control options.

The first mode is a system level mode in which the user specifies the top level test they want to run. For example, they can use a command such as "runQPC" to initiate the whole qPCR process for the desired inputs. Most of the functionality of this mode has not yet been implemented because a full prototype of the device has not yet been built. Instead, a basic skeleton of the functionality of this mode has been laid out.

The second mode is the command mode. In this mode, the user runs commands or scripts in order to achieve the overall desired sequence. This mode is like a programming/scripting mode that a more experienced user can operate to specify their own specific tests. Examples and stock scripts to use in this mode can be provided and more will be added in the future as functionality increases.

The third mode is a testing mode, made specifically for the system designers to test the functionality of the system. In this mode the user specifies a type of device and it shows all of those devices in the system. Then the user can individually actuate each device in order to test the system. The code for the user interface is in the appendix.

Chapter 5

Parameter Characterization

This chapter describes the testing carried out as well as the results obtained. It begins by describing how the different parameters of the microvalve control system are analyzed and tested. Then it describes how these results are used to determine a procedure to optimize the system for power efficiency. Finally, it details the test set-up used to test the control system as well as the set of test conducted on the control system and presents the overall results.

5.1 Evaluating Microvalve Control Parameters

This section goes through the valve control parameters outlined in Chapter 3 and describes how each of them are tested to assess their impact on the control system.

5.1.1 Procedure for Measuring Magnetic Field

The procedure used for measuring the magnetic field of the magnets, electromagnetic coils and valve flaps is described in this section. It is necessary to actually measure the magnetic field of these components rather than just calculate or simulate it since the field generated by the magnets is slightly irregular, and since the coils are obtained without knowing details of their make-up (the kind of wire and the number of turns of the coil). Measurements are also required in order to confirm the accuracy of calculations, approximations and simulations.

The magnetic field measurements are taken with an F.W. Bell model 5180 Gauss/Tesla Meter [50] by placing the tip of the Tesla meter perpendicular to the direction of the

magnetic field. The tip is placed directly on top and gradually moved further away in order to measure the drop of the magnetic field over distance away from the magnet/coil. As stated, the magnetic field is slightly irregular (or the Tesla meter tip itself is slightly irregular) and the highest magnetic field is not exactly at the centre of the magnet, as expected. To compensate, the measurements are taken by locating this point of highest field and keeping the Tesla meter in place over that spot as the magnet is moved further away. The tip of the Tesla meter is kept in place using a piece of 3D printed plastic as seen in the Figure 5.1. The end of the tip hangs over the plastic in order to make sure that the measurements are taken with nothing but air between the tip and the magnetic field source. Even though the magnetic susceptibility of non-magnetic materials such as plastic and paper is not that much different that that of air [51], the best approach is to have as little interference as possible. The Tesla meter tip is held in place, therefore the jig has to be shifted around until the point of highest field is found. Once the relative configuration between the magnet and tip is found, the aligner is fixed in place to the magnet holder, and add spacers to distance the magnet from the Tesla meter tip.

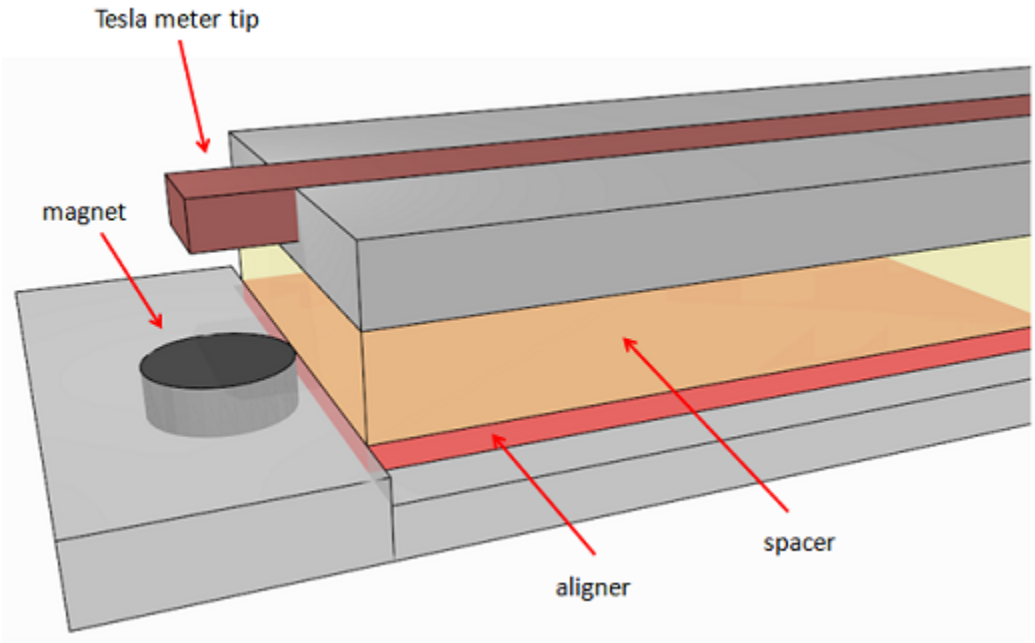

Figure 5.1: Setup used to measure the magnetic field

This manual jig is chosen over something more automatic such as using a linear stage to move the magnet away from the tip because setting up such a system is difficult since the

Tesla meter tip is quite fragile and must not be twisted. This meant that the tip must be held in place horizontally to prevent twisting and breaking at the base, which is established by keeping the whole Tesla meter tip encased in the plastic jig.

5.1.2 Magnetic Field of the Magnets

Although cylindrical magnets come in many shapes and sizes, for this project magnets that are of comparable sizes to the valves themselves are considered. Three magnets in particular are tested thoroughly and considered for the control jig: a 6.5 mm diameter by 2.5 mm height magnet that is the same magnet used to keep the valves closed in [1], a 6.5 mm diameter by 1.5 mm height magnet that, being slightly more than half the size of the previous magnet, is considered in case a lower-field solution is desirable. And a third, much smaller magnet of 3.2 mm diameter by 0.8 mm height is mostly considered to determine what magnitudes of fields can be obtained if the whole valve system is miniaturized. A picture of the magnets and a graph of the results are shown in Figure 5.2 and Figure 5.3. The magnet dimensions are confirmed by measuring them with a TRESNA micrometer [52], the results are presented in the Table 5.1.

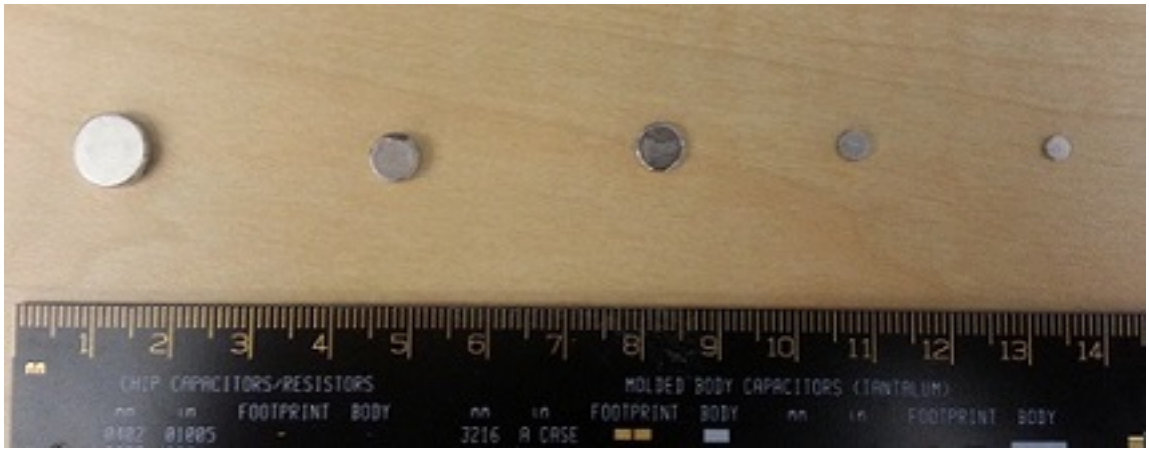

Figure 5.2: Magnets considered for the control system

The magnetic field of the magnets is measured over a large distance because how far away the field drops to negligible amounts impacts how far the magnets have to be moved in order to have a sufficiently low field. As expected, either the 6.5 x 2.5 mm magnet or the 6.5x 1.5 mm magnet are candidates for the set-up using the current valves. As detailed in [1], it takes 80 mT of magnetic field at the flap to hold the 2.3 mm flap against 10.3 kPa

Table 5.1: Dimensions of the magnets tested

Magnet	Diameter (mm)	Height (mm)
1	3.22	0.83
2	4.37	1.53
3	6.45	1.52
4	6.47	2.55
5	9.54	2.55

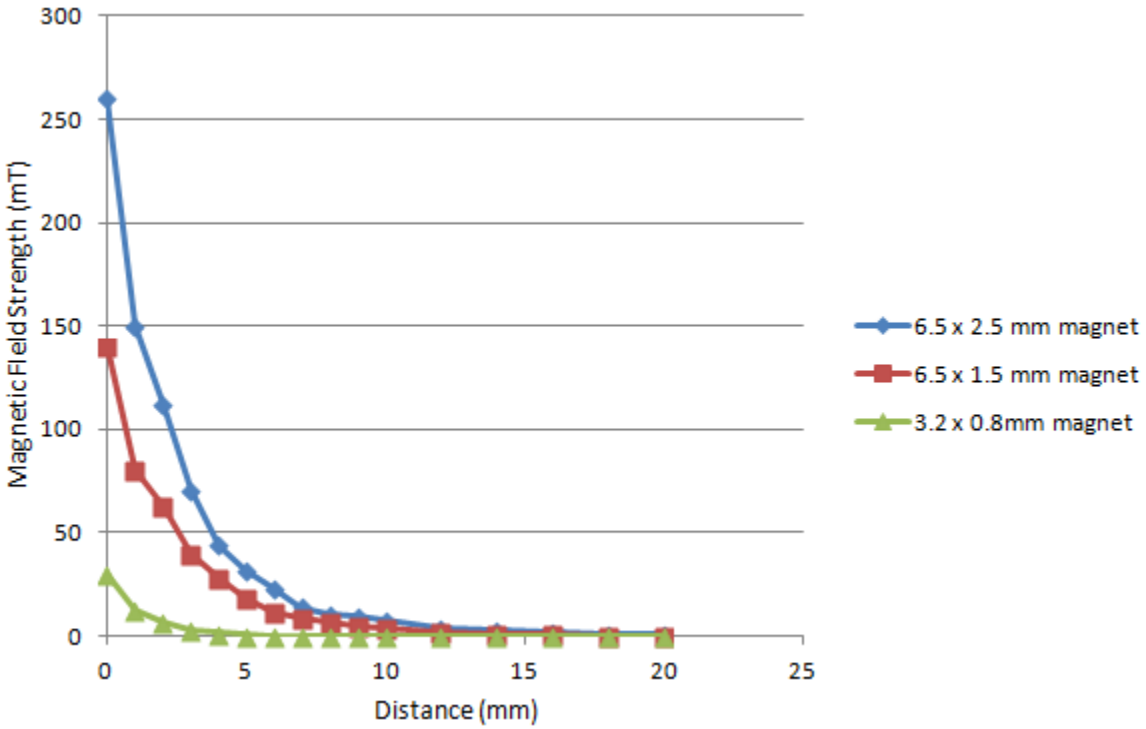

Figure 5.3: The magnetic field of the magnets

of fluidic pressure. The PDMS substrate is 2 mm – 2.5 mm thick typically, thus either of the first two magnets provide the required field (assuming that the field going through the PDMS does not affect it much compared to the field measured in air). In fact, depending on the amount of pressure the flaps have to withstand, a smaller magnet than that may be used. The small 3.5 mm magnet however has too weak a field at 2.5 mm away, less than 10 mT, which is much too small for the current valves.

In addition to measuring the magnetic field of those three magnets over a large distance, the magnetic field of some additional magnets is characterized over the smaller distance of 0 to 5 mm away in order to test their magnetic fields in the most interesting region as

shown in Figure 5.4. It is observed is that even though both sets of tests include the three sizes of magnets from the original extended data, that the results of the magnetic field are different. Thus, even when the same size of magnets are used, the magnetic field can vary between individual magnets. The results in Figure 5.4 are obtained by selecting the strongest magnet of that particular size from the available samples. This means that care must be taken when selecting the magnets to ensure that the magnetic field of the actual magnets used match the expected magnetic field for that magnet size. This is especially important in a homogeneous system using the same magnets throughout.

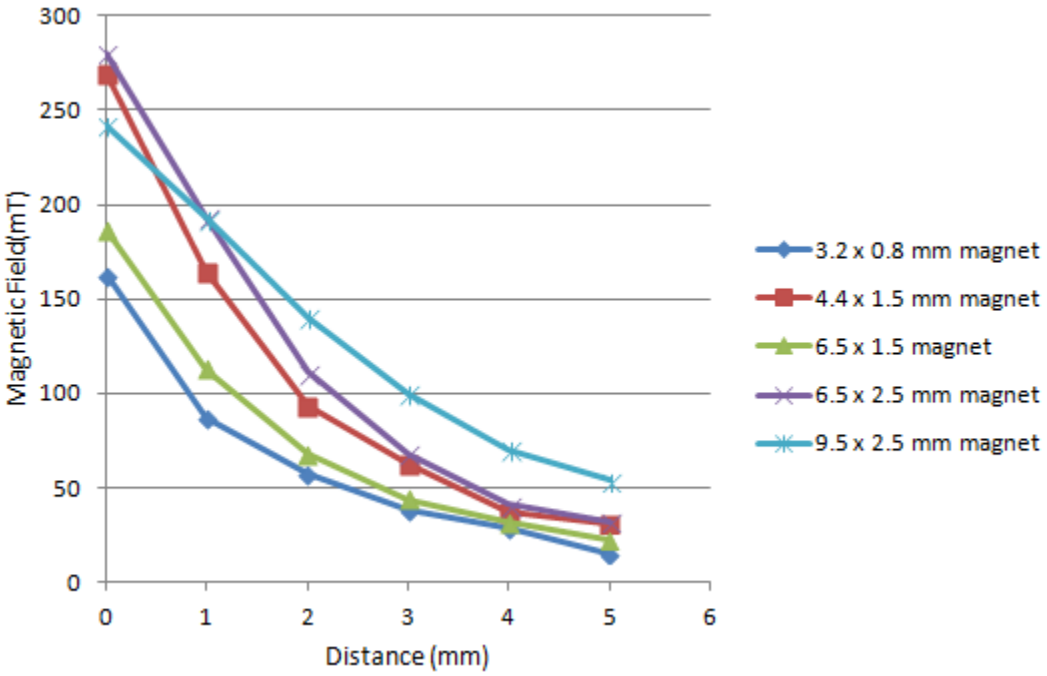

Figure 5.4: Magnetic field of a larger sampling of magnets over a smaller distance range

From the graph, the 6.5 x 2.5 mm and the 6.5 x 1.5 mm magnets are the appropriate choices for use in the control system. The other smaller magnets, even though they have a much higher magnetic field in these measurements, still have a much too small area that this field is distributed over so the force between them and the valve flap is not strong enough. Further characterization of this is in the valve thickness section.

COMSOL simulation of Magnets

In order to better understand the magnetic field associated with different geometries of cylindrical magnets, COMSOL simulations are performed on cylindrical magnets of varying heights and diameters. Cylindrical magnets of 4 different heights and 3 different diameters are simulated for a total of 12 magnet sizes suitable for the project. The results for magnets of 6.5 mm diameter are in Figure 5.5, the results for magnets of 4.5 mm diameter are in Figure 5.6 and the results for magnets of 2.5 mm diameter are in Figure 5.7.

The simulations are completed by defining a geometry to correspond with the desired shape and size of the magnet and specifying the magnetization of that object. The magnetization is set to be the same for all the tests set to the average calculated value for the test magnets), since the effects that the geometry has on the magnetic field is the only property of interest. Mesh independence is verified by changing the size of the mesh to ensure consistent results. The field of the magnet is graphed on the z-axis (height) with the edge of the magnet itself at the origin.

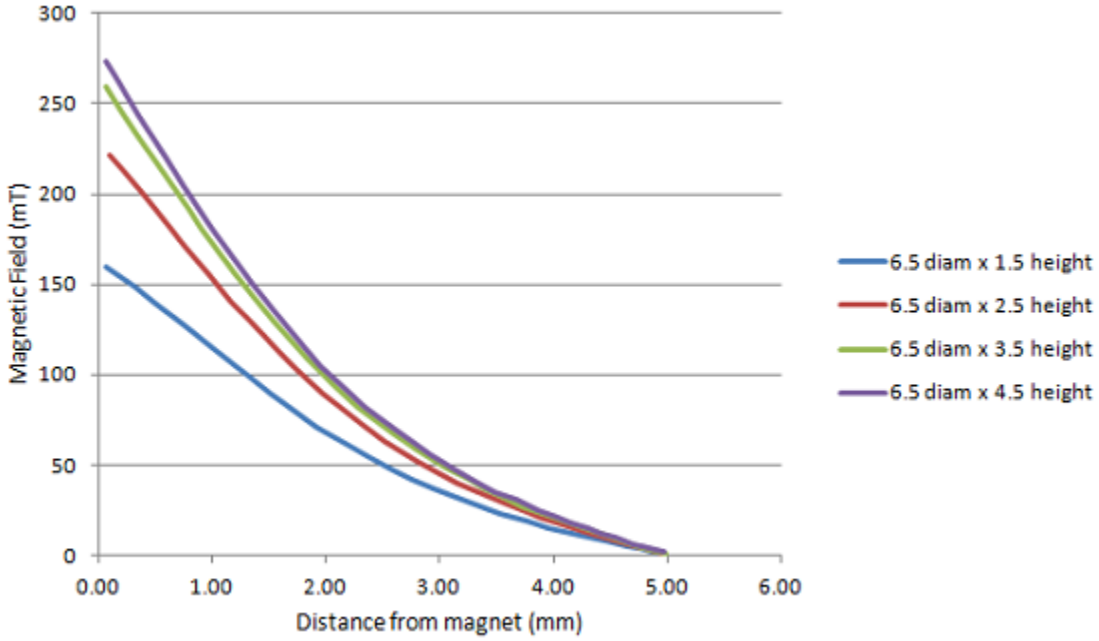

Figure 5.5: Simulated magnetic field of cylindrical magnets

These numbers are quite close to the measured results in the previous section for the magnets of corresponding sizes. The differences are between 10% and 25% when comparing

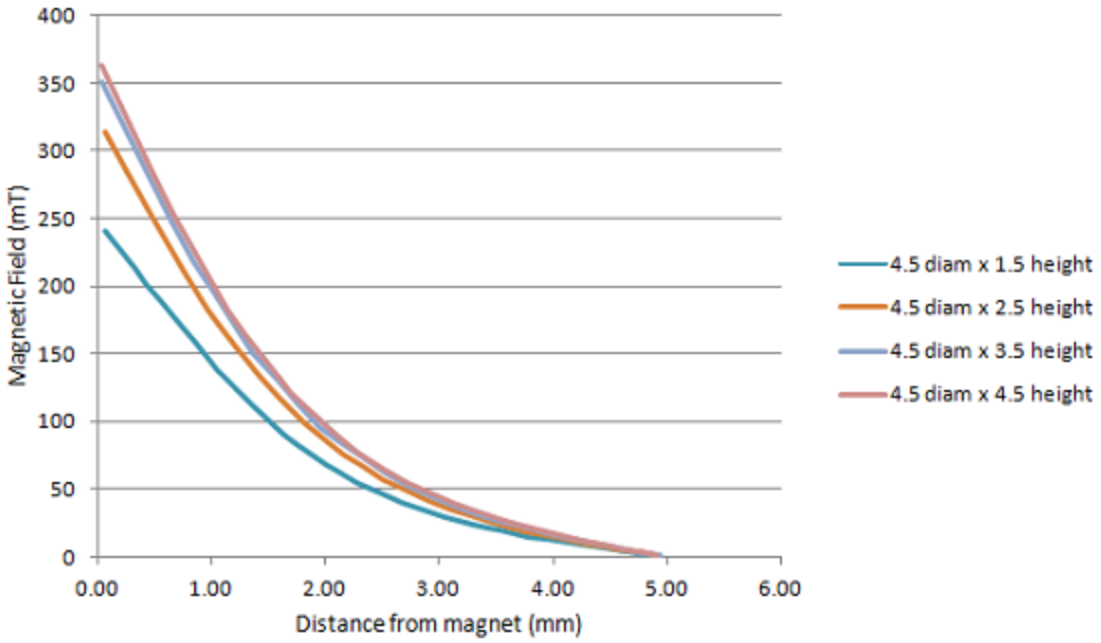

Figure 5.6: Simulated magnetic field of cylindrical magnets

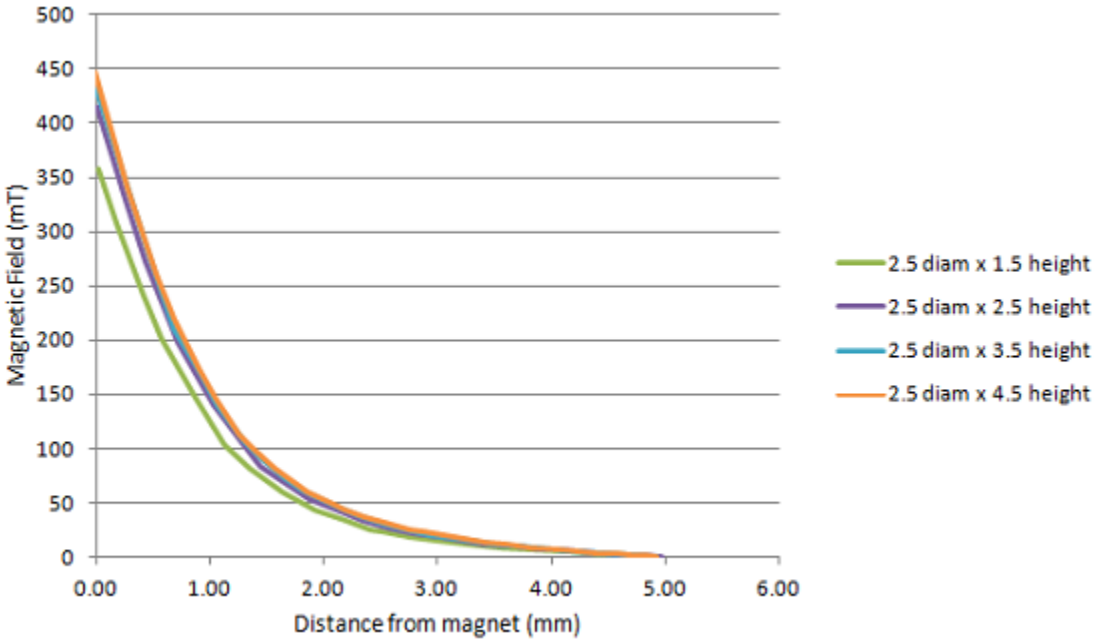

Figure 5.7: Simulated magnetic field of cylindrical magnets

similarly sized magnets. This means that the COMSOL simulation can safely be used as a good approximation for the magnetic field.

According to these simulations, the magnetic field increases as the height of the magnet increases, the rate of increase dropping off at larger heights, which is expected. However, the simulations imply that the magnetic field is higher for the smaller diameters of magnets. They all have the same magnetization, i.e. the same density of magnetic dipole movement, so this should not be the case.

5.1.3 Magnetic Field of the Electromagnetic Coils

Using the same measuring procedure described at the start of this chapter, three electromagnetic coils are tested to determine their field over distance. The three coils are 4 mm, 7 mm and 10 mm in diameter. The exact dimensions of the coils are measured with the same micrometer as the magnets, and the results are shown in Table 5.2.

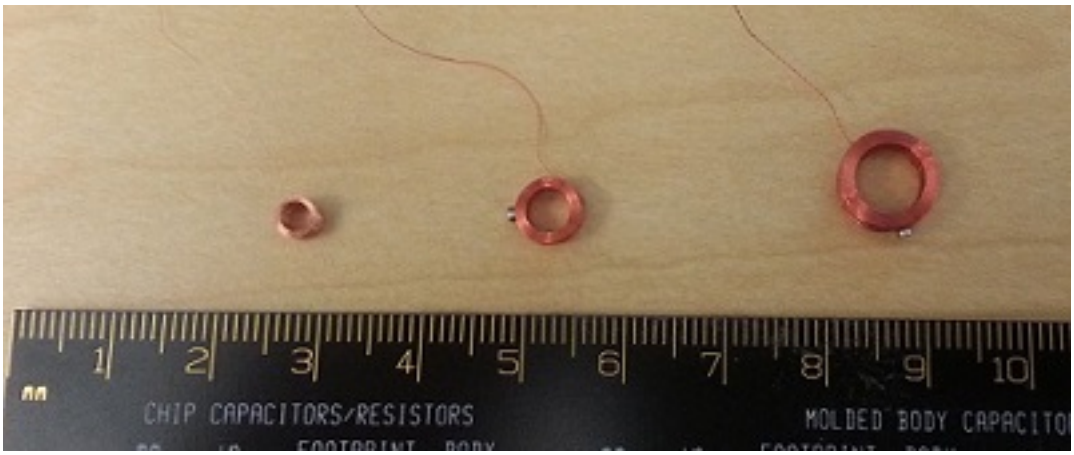

Figure 5.8: Electromagnetic coils considered for the control system

The electromagnetic coils also have another parameter: current through the coil. Running current through the coil over an extended period of time, or in many smaller bursts, causes the coil to heat up and thus lower its resistance. If power is applied to the coil by connecting it directly to a voltage source, lowering the resistance of the coil results in less current running through it, it produces less magnetic field. The exact effects of this are

Table 5.2: Dimensions of the coils tested

Electromagnetic Coil	Diameter (mm)	Height (mm)
1	4.12	2.55
2	6.63	2.87
3	10.21	3.65

detailed in a later section. This effect requires that the coils are cooled between taking measurements. Multiple measurements are taken to ensure accuracy, and the field is only measured using 75 mA and 150 mA of current. The field is expected to be directly proportional to the current, thus by confirming if the field at 150 mA is roughly twice that at 75 mA, predictions can be made for any current magnitude by multiplying by the appropriate number. This is the case as shown in the Figure 5.9.

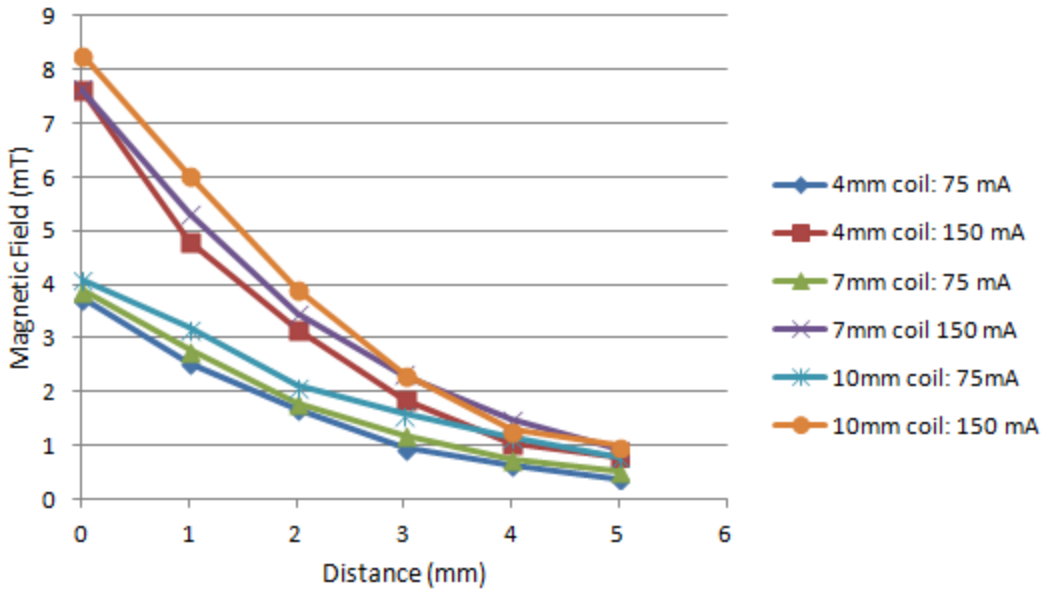

Figure 5.9: Magnetic field of the electromagnetic coils and field generated by the flap

All three coils had similar strength of magnetic fields, which leaves only the dimensions of coil to affect the choice. "Magnetic field" as used in this thesis is, in actuality, magnetic flux density, thus the total force created by it is dependent on the size of the area that the flux density is distributed over. Other considerations might be how much current the coil can withstand before getting too hot or damaged. As seen in the picture, the 4 mm coil has much thinner wire than the other coils, thus it cannot withstand as high a current as the other coils.

Power Efficiency of Coils

The power efficiency of a coil can be estimated by dividing the total magnetic flux generated by it (magnetic field times the inner area of the coil) by the power required to generate this field (current times the square of the resistance of the coil). This is possible since the

magnetic inside the coil is consistent. This parameter should be independent of the actual current used, since the magnetic field is directly proportional to current as seen in Table 5.3.

As predicted, the power efficiency is the same regardless of what current it is measured for. As seen, the largest coil is by far the most power efficient, and this is the one to use whenever the space available allows it.

Calculated Versus Actual Results

Using Equation 2.8, B_0, the magnetic field at the centre of the coil, is determined by letting $z = l/2$ and $B =$ magnetic field touching the coil and calculating B_0. From there, Equation 2.8 is further used to calculate the magnetic field anywhere on the z axis. Table 5.4 shows the resulting B_0 values and Figure 5.10 shows the magnetic field calculated using equation 2.8 compared to the measured magnetic field above for the 6.5 x 2.5 mm magnet and the 7 mm coil. As Table 5.4 shows, the B_0 value for the magnets is close to 1 T, which is consistent with the fact that the $\mu_0 M_0$ value for magnetized iron is 1 T [51].

From Figure 5.10, it is evident that the measured magnetic field and the calculated magnetic field vary only slightly, with the calculated field tending to be slightly larger than the measured field. This means that this equation can be used to calculate the magnetic field of coils and magnets, given knowledge of either the measured magnetic field at the edge of the magnet, or by knowing the magnetization of the material. Thus, the magnetic field for any cylindrical magnet in any dimensions and material can be approximated, provided a known magnetization. As well, the figure shows that the COMSOL simulated field also agrees with the calculated and measured results with only approximately 20% difference. This is useful later when selecting an appropriate magnet for the control system.

Table 5.3: Power efficiency of the coils

Coil	Inner Diameter	Resistance	Power Efficiency at 75 mA	At 150 mA
4 mm	3	77	0.060	0.061
7 mm	4.5	60	0.230	0.225
10 mm	6.5	53	0.646	0.653

Table 5.4: B_0 values for coils and magnets

Number	Dimension	$B_0(mT)$
Magnet 1	3.2x0.8 mm	729
Magnet 2	4.4x1.5 mm	959
Magnet 3	6.5x1.5 mm	892
Magnet 4	6.5x2.5 mm	918
Magnet 5	9.5x2.5 mm	1039
Coil 1	4 mm at 150 mA	19
Coil 2	7 mm at 150 mA	20
Coil 3	10 mm at 150 mA	28

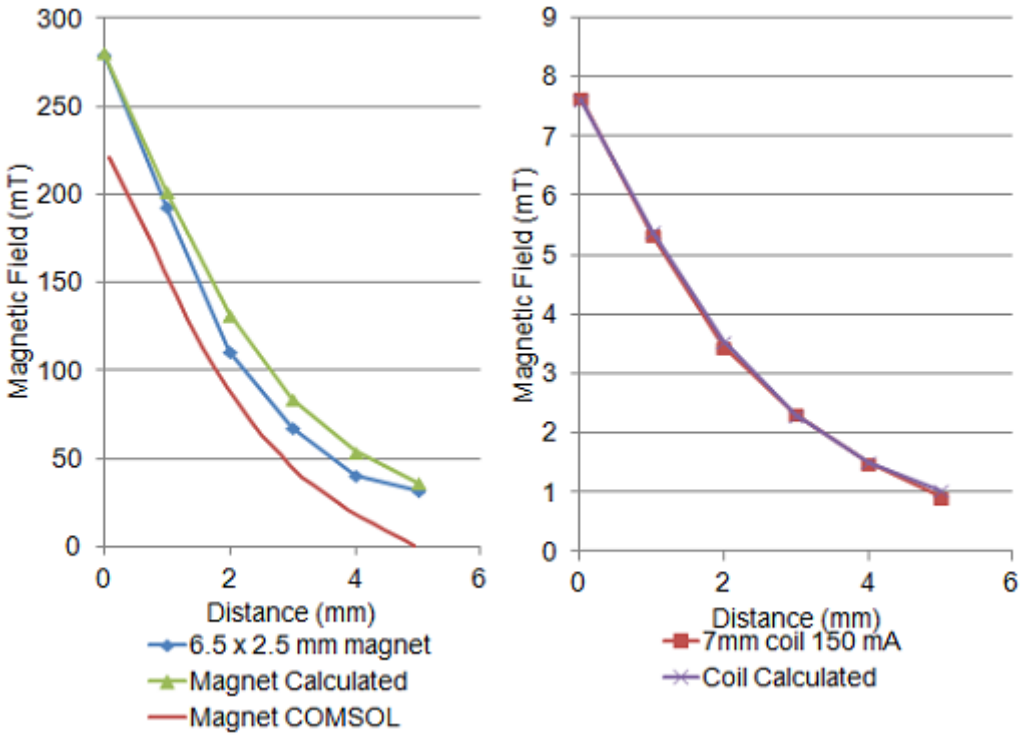

Figure 5.10: Calculated magnetic field compared to measured field

Coils with Ferromagnetic Core

Equation 2.7 says that the magnetic field inside of an (air core) coil is proportional to the number of turns of the coil as well as the amount of current flowing through it. Another way to improve the strength of the field produced by an electromagnetic coil is to wind the coil around a ferromagnetic core. This is tested using the 4 mm air core coil by obtaining an iron core from an inductor of the same size and placing it inside the coil. The results are shown in Figure 5.11.

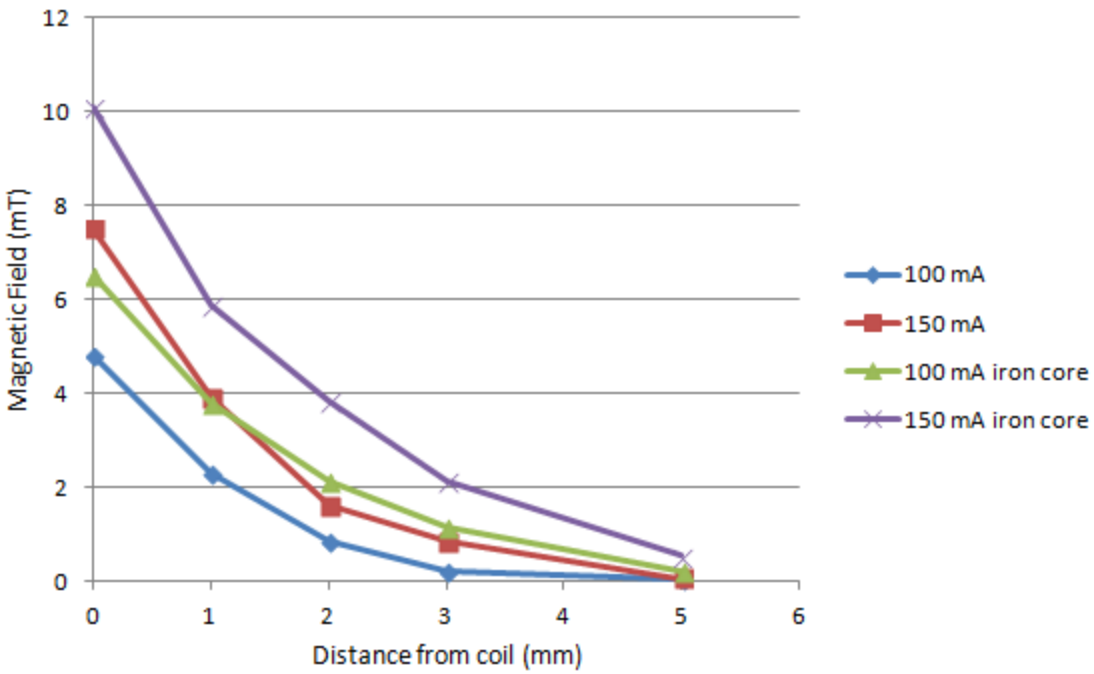

Figure 5.11: Increase in magnetic field due to iron core for 4 mm coil

The graph shows that this set-up allows the 4 mm coil to produce 1.4 times as much magnetic field without requiring any additional current. The result can be improved by using a better material for the core.

Whereas this is promising for power consumption concerns, it is unfortunately not something that can be employed in the latching mechanism described in Section 3 because testing determined that the magnet becomes too attached to the iron core of the coil and no amount of (reasonable) current through the coil allows the magnet to be repelled from this position. However, power consumption can be improved without weakening the magnetic field by using a ferromagnetic core inductor for driving the magnetic cilia.

Inductance of Coil and Switching Speed

The inductance of the electromagnetic coils is an important property because it determines the coil's capability for storing power in the magnetic field. If the coil is directly connected to the power source, with no other components directly in the circuit, the inductance, L together with the internal resistance of the coil, R determines the time constant, T:

$$T = L/R \tag{5.1}$$

The time constant determines how fast the coil fully generates a magnetic field once power is applied to it. After an amount of time equal to five time constants, the coil generates a magnetic field at full power for the amount of current applied. For this application, the time constant is relevant to knowing how fast magnetic field in the coil can be switched on and off.

The inductance of the 4 mm coil and the 7 mm coil is calculated by placing them in an LR circuit and observing the time constant. The inductance of the 4 mm and 7 mm coils are 14 mH and 22 mH, respectively. Their corresponding time constants, assuming that the coils are directly connected to the power source and there is no other component in the circuit are 0.2 ms and 0.3 ms for the 4 mm coil and 7 mm coil, respectively. The time constants are so low, thus there is no need to worry about switching time with these coils.

5.1.4 Valve Flap Thickness Evaluation

To determine how the magnetic flap of the valve interacts with other magnetic components, its magnetic field is measured, using the aforementioned process used to measure the field of the magnets and coils, for a variety of flaps which are measured with the micrometer to accurately determine their heights. The results are shown in Figure 5.12.

This data compares well with data previously obtained by Rahbar et al. for M-CP pieces with different geometries.Whereas the flaps characterized in this thesis are cylindrical and asymmetrical to enable them to be used as flaps, Rahbar et al. performed characterizations on square and uniform pieces of the same material. The trends in the overall data are the same for both sets of measurements with the individual values showing a percent difference due to the difference in geometry and sample.

Measuring these flaps is especially important since they are not uniformly cylindrical, and their fields are hard to approximate with equations. From these measurements, simplified equations can be used to approximate the force between the valve flap and the magnet at different distances. How well the magnetic field equations hold up for the valve flaps can also be examined since they are not exactly cylindrical. Table 5.5 shows the calculated B_0

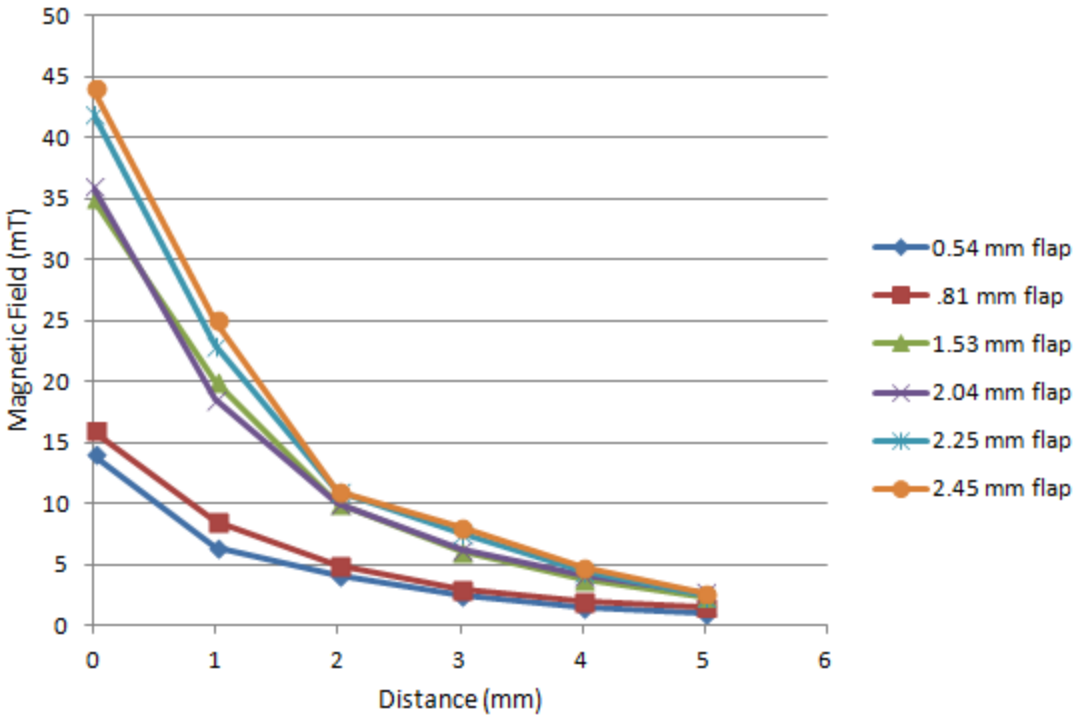

Figure 5.12: Magnetic field for the valve flaps

values for the flaps and Figure 5.13 shows the how the measured magnetic field compares to the calculated magnetic field for two of the flaps.

The magnetic field calculated from the formula is quite different (larger) than the measured magnetic field for values after the one used to calculate B_0 for. This is expected since the formula assumes that the flaps are cylindrical, while, in fact they are more like half cylinders. The formula overestimates the magnetic field value by increasingly large amounts, until it stabilizes at 2 times the actual measured value. This makes sense since

Table 5.5: B_0 values for valve flaps

Number	Dimension	$B_0(mT)$
1	0.54 mm	184
2	0.81 mm	142
3	1.53 mm	175
4	2.04 mm	143
5	2.25 mm	155
6	2.45 mm	153

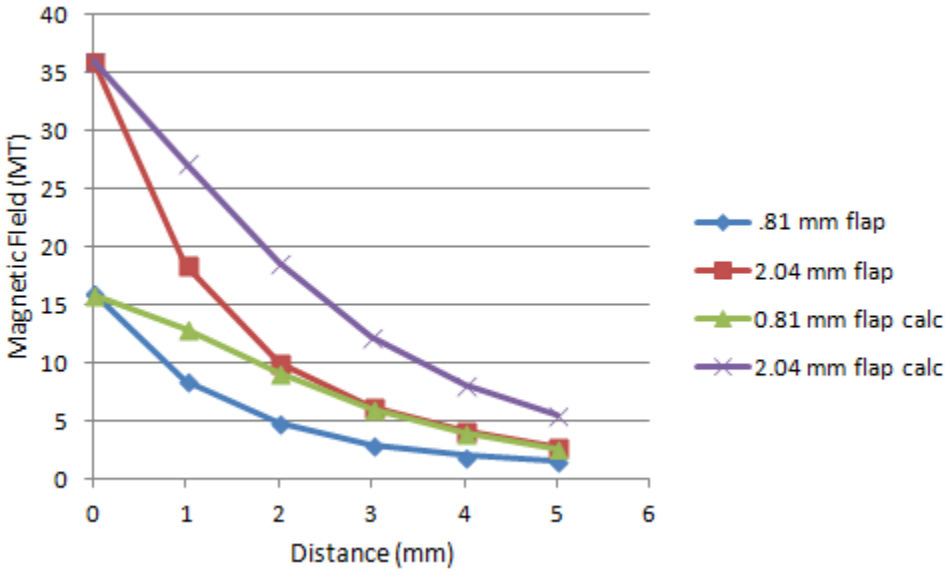

Figure 5.13: Calculated magnetic field vs measured magnetic field for flaps

the formula assumes that the flaps are perfectly cylindrical, whereas the bulk of them actually fill half that space. This is something that must be kept in mind for future formulas.

Results in terms of how the thickness of the valve flap is related to the pressure the valve can withstand and magnetic field used are presented in [1] and and do not need to be further characterized for this thesis. Instead, the values presented there are used as the basis for calculations.

5.1.5 Height of the Magnet Channel

Whereas the strength of the magnet determines how much pressure the valves can withstand in their CLOSED state, the height of the magnet channel determines how much the field is lowered when in the OPEN state. In other words, the height of the magnet channel dictates the pressure differential between OPEN and CLOSED. If the magnet channel is shorter, the magnet and valve flap are not separated as much when in the OPEN state, thus the force between them is stronger than when the channel is taller as shown in Figure 5.14 The closer the two states are together, the bigger chance that mistakes happen if the pressure is not exact, because the device would be operating close to the limit at all times. This also restricts the size of the channels, if uniform pressure is used, since pressure drops normally

over the length of a channel. As well, there are restrictions on the viscosity of fluids, since more viscous fluids require more pressure to operate.

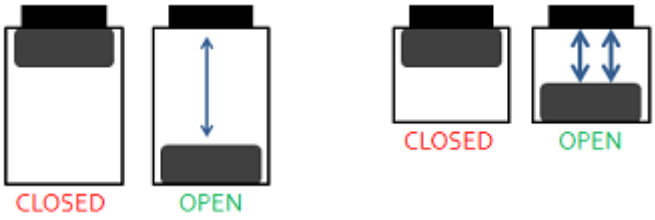

Figure 5.14: OPEN and CLOSED states for shorter vs taller magnet channel

Conversely, the higher the height of the magnet channel, the more magnetic field the electromagnetic coil is required to generate in order to attract the magnet. In this case, a performance versus power trade-off needs to be made. How much a valve is OPEN or CLOSED is quantified by the amount of fluidic pressure required to push past the valve. The valve is considered open for fluid of pressure higher than this leakage pressure and closed for fluid below this pressure. In the case of the OPEN and CLOSED states, the leakage pressure for both of these positions is measured to determine how much pressure the valve can withstand while closed and how much pressure is needed to operate the system at when the valve is open. Rahbar et al. characterizes fluidic leakage pressure for different valve thicknesses at the same applied magnetic field [1].

Based on that information and the magnetic field strength of the valve flap and magnets over distance, the expected leakage pressure can be calculated using a few approximations. Considering the force of the fluid pushing up on the valve flap versus the force of the magnetic field pulling the flap down, and assuming that no other forces are present (i.e. that the valve flap on its own does not offer much of a resistance to the fluid) then if the fluidic force is stronger than the magnetic force, the valve leaks. The fluidic force is equal to the fluidic pressure times the area of the channel. The force is approximated using an equation from [53] for the magnetic force between two cylindrical magnets:

$$F_z = -\frac{\pi \mu_0 M^2 R^4}{4} \left\{ \frac{1}{x^2} - \frac{1}{x+t_1} - \frac{1}{x+t_2} + \frac{1}{x+t_1+t_2} \right\} \tag{5.2}$$

where F_z is the force in the z direction (parallel to the height of the magnets), M is the magnetization of the magnets, R is the radius of the magnets, x is the distance between the

magnets and t_1 and t_2 are the heights of the two magnets. Using the fact that $B_0 = \mu_0 M$ and approximating with two different magnetizations and radii, gives:

$$F_z = -\frac{\pi B_{01} B_{02} {R_1}^2 {R_2}^2}{4\mu_0} \left\{ \frac{1}{x^2} - \frac{1}{x+t_1} - \frac{1}{x+t_2} + \frac{1}{x+t_1+t_2} \right\} \tag{5.3}$$

Using the equations to calculate the force between the magnet and flap at 3.5 mm distance (that is the distance between the 9.5 mm diameter magnet and flap to achieve 90 mT of field at the flap), gives a value of 2.198 and 2.197 N for the 2.25 and 2.45 mm flaps, respectively. Given that the diameter of the channel at the flap is 0.75 mm then for a 3.5 kPa pressure against the flap, this amounts to a total of 0.006 N of force that the fluid is exerting on the flap. This is inconsistent with the equation calculations, either because the equation results are inaccurate due to the small distances or because the force between the magnet and the flap is spread out over a larger area because the magnet is pushing up on the whole substrate rather than just against the flap. Likely, both are contributing is the case, and Equation 4.2 should only be used when comparing forces to determine which is more attracted rather than comparing directly with the pressure of the fluid on the flap. Values from this equation may still be used to compare between two magnet and valve pairs, if not actually use the resulting values. Table 5.6 shows the calculated force values for three magnets against five flap thicknesses at 3.5 mm separation distance and one at 5.5 mm separation distance. The relativity of some the results in the table are confirmed experimentally. For example, it makes sense that a larger magnet would produce more force as well as a larger flap, but putting the 9.5 mm magnet farther away resulted in less force than all the other results which is confirmed experimentally.

Perhaps a better approximation would be to use the fact (from these equations and from 2.10) that the force should be proportional to the product of the two magnetic fields.

Table 5.6: Force calculated values for magnets and flaps

Flap thickness	9.5 x 2.5 mm	6.5 x 2.5 mm	6.5 x 1.5 mm	9.5 mm, 5.5 mm away
0.0005	2.17871	1.05157	1.05148	0.88048
0.0008	2.17899	1.05171	1.05157	0.88059
0.0015	2.17951	1.05196	1.05173	0.88079
0.00204	2.17979	1.0521	1.05181	0.88091
0.00225	2.17988	1.05214	1.05185	0.88095
0.00245	2.17997	1.05218	1.05187	0.88099

The results from this section as well as the results from [1] show that the leakage pressure increases linearly with increased magnetic field from the flap being thicker as well as with the strength of the magnetic field increasing.

COMSOL is also used to simulate the force between the magnet and valve flap. This is done by adding another magnetic component to the magnet simulation the desired distance away. COMSOL could not provide reliable data if that other component is also magnetized, so it had to be approximated by using a non magnetized component instead. For the simulated 6.5 mm diameter and 2.5 mm height magnet, with a simulated iron flap of 6 mm diameter and 2 mm height, the force between the magnet and flap is 0.197 N when they are 2 mm away, but it drops to merely $1.85 * 10^{-3}$ N at 5 mm away and further drops to $5.25 * 10^{-4}$ at 6 mm away. This shows that moving the magnet 5 mm or 6 mm away from the flap should be enough to weaken the force experienced by the flap enough to open the valve.

As well, Rahbar et al. characterized how a changing magnetic field affects the leakage pressure of the valve. From this data, it is clear that after moving the magnet away a few millimeters from the valve flap, that the leakage pressure drops to the point of the valve being considered OPEN. This data supports the approximation of 5 mm distance for the valve to be considered OPEN.

5.1.6 Other Parameters

This section finishes discussing the parameters identified in Section 3.3. The remaining parameters that have not yet been discussed how to manage and test are:

1. the distance between the valves
2. the height of the PDMS substrate
3. the fluidic pressure to run the system at

The minimum distance between the vales is directly related to the magnets that are used. The stronger the magnet, the farther apart they need to be in order to avoid interfering with each other. Where this gets interesting is that if the distance between the valves need to be large, this increases the total size of the overall microfluidics layer, thus increasing the maximum possible path that needs to be driven, thereby increasing the pressure needed

to drive the microfluidics at. The horizontal distance that the magnets interfere with each other (measured centre of magnet to centre of magnet), on their own or in the control jig (this is tested using valves that are positioned close to the edge of the glass slide so the distance between them is easily adjusted) is presented in Table 5.7.

Table 5.7: Horizontal minimum distance between magnets without interference

Magnet	Diameter (mm)	Distance in Space (mm)	Distance in Jig (mm)
1	3.22	16	11
2	4.37	22	16
3	6.45	28	19
4	6.47	32	22
5	9.54	43	30

The height of the PDMS substrate is one of the parameter that can be eliminated, since it effects the field that the magnet exerts at the position of the flap (the thicker the substrate, the farther away the magnet is, the weaker the field is), but this can easily be managed by choosing a different magnet.

The fluidic pressure required is dependent on what kinds of fluids need to be moved across the system as well as how large the microfluidics layer is. This is, in actuality, a requirement and not a parameter.

Chapter 6

Selecting and Evaluating Control Solution

This chapter describes how to use the parameter characterization from the previous chapter and derive the ideal parameters of the control solution.

6.1 Optimizing for Power Consumption

This section presents a procedure for selecting the appropriate parameters to optimize the control solution for power consumption, while keeping with some stated requirements. A basic procedure is presented, then some more considerations that can be included are described.

6.1.1 Determining Control System Parameter to Optimize for Power

Beginning with the assumptions: firstly, assume that the size of the final microfluidics layer is known and that the worst-case path-length is equal to twice the length of the layer. While it is true that the choice of magnet determines how close the valves can be together and this influences the final size of the system, assume that the valves are already placed far enough away or can be placed far enough away such that there is no interference without having to make the system larger. Next, assume that the viscosity of the most viscous and the viscosity of the typical fluid (average) used is known. Thirdly, assume all valves are the same throughout and that the system is driven by constant but adjustable flow rate

like from a syringe pump. Fourth, assume for the fluidic resistance calculations that any additional resistance added by valves in their open state is negligible even for the path that has to transverse the largest number of valves. For this procedure also assume that the height of the PDMS substrate is fixed.

With those assumptions in mind, here are the steps to choose a system that is optimized for power consumption while keeping the system reliable. This procedure is summarized in Figure 6.1.

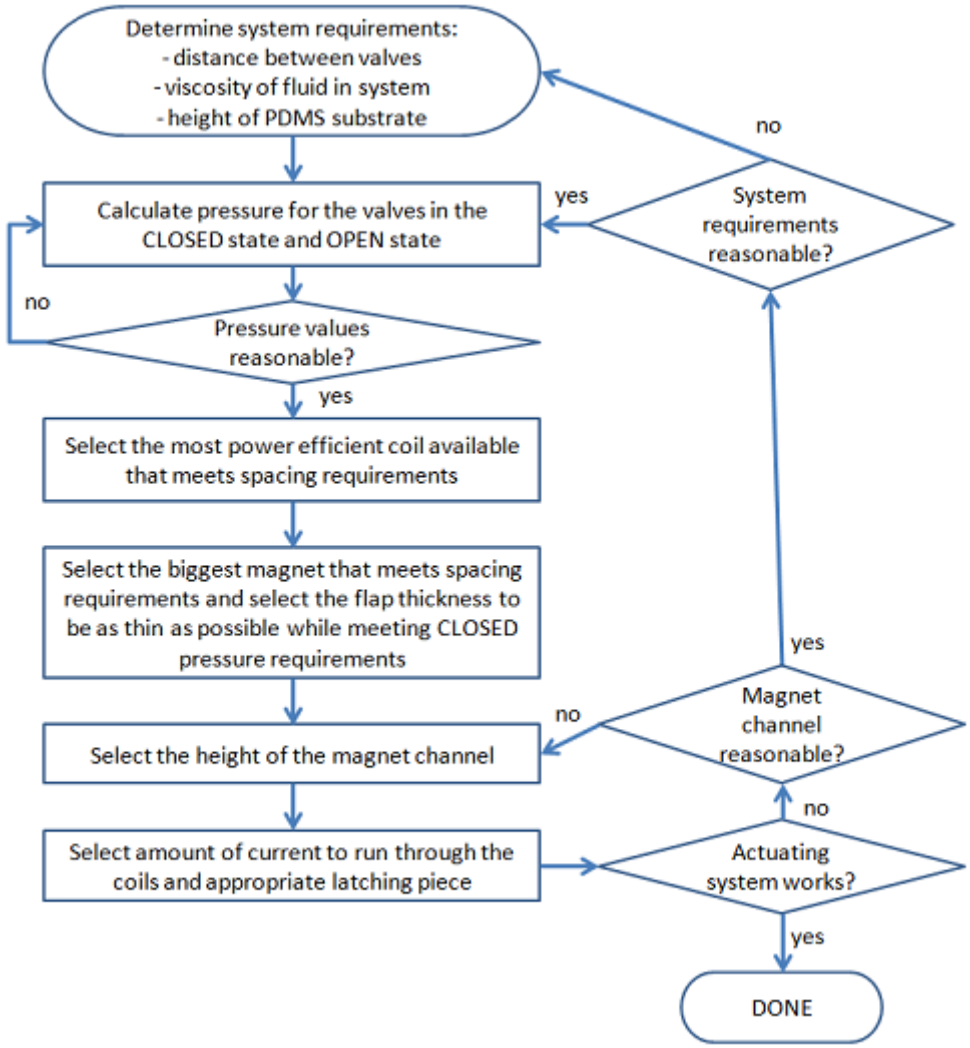

Figure 6.1: Parameter selection for power optimization

1. Determine the pressure the valves need to withstand in their CLOSED state in the system by:

(a) Calculate the pressure drop for the typical fluid and the most viscous fluid over the worst case path assuming the highest required flow rate for that fluid.

(b) Take either 10 x the pressure drop for the typical fluid or 5 x the pressure drop for the most viscous fluid, whichever is larger to be (i.e. the pressure loss over this path should be no more than 10% of the maximum pressure for a typical fluid of 20% of the maximum pressure for the most viscous fluid). This is the pressure the valves need to withstand in the system. If at any point in the procedure this pressure is too small for reliability or too big for power consumption, come back here and choose a new pressure.

2. Depending on how far away the magnet is moved and on the particulars of the valves, even in their OPEN state they withstand some amount of pressure. Determine the pressure the valves need to allow in their OPEN state by taking it to be half of the pressure for the CLOSED state. If at any point in the procedure it is determined that this is not sufficient for reliability, come back to this step and choose a lower pressure, or if it is too much in terms of power consumption, choose a higher pressure.

3. Given the pressure requirement on the valves in their OPEN and CLOSED state, the parameters: magnet, coil, height of the magnet channel, valve flap thickness and current needed, are chosen as follows (assuming the choices of magnets and coils are discrete):

   (a) Start by choosing the most power efficient coil allowed by space constraints

   (b) Choose a magnet and flap thickness combination that withstands the required pressure given the substrate thickness provided by the manufacturing process:

      i. Choose a reasonably sized magnet based on the size constraints

      ii. Choose a flap thickness such that the force between the flap and the magnet is enough to withstand the required fluid pressure

      iii. In this step it is best to use as powerful a magnet as possible and as weak a flap as possible

   (c) Choose the height of the magnet channel by determining the desired difference between the OPEN and CLOSED states.

(d) Choose the current to run through the coil to be strong enough to attract the magnet over the height of the magnet channel. Specifically, the magnetic field that the coil exerts at the location of the magnet should be greater that the magnetic field that the flap exerts at the location of the magnet. That way, the coil attracts the magnet away from the flap.

(e) Choose latching component such that the magnet can be repulsed from the coil using the same amount of current that it is attracted with (in the opposite direction)

A note about the assumption and approximations: while some of these numbers may seem quite arbitrary, and in some cases they are and in order to tune this algorithm they do need to be further investigated and quantified, the approximations are determined through discussion and deliberation with the other researchers working on µROAMS.

6.2 Evaluating the Solution

This section discusses the evaluation of the solution using the previously described procedure as well as the power versus area trade-off and the reliability versus power trade-off and how to improve this.

6.2.1 Demonstrating Parameter Choice With a Case Study

Suppose that for this experiment a 25 cm x 25 cm microfluidics layer is used and that the fluid tested in the system is twice as dense as plain water, which has a viscosity of 1 x 10^{-3} Pa/s and that the flow rate ends to be 200 µL/min with the given channel size of 100 x 200 microns. Also suppose that the use of a valve series with 3.5 mm substrate thickness and the choices are limited to 0.5 mm, 1.5 mm and 2.25 mm flap thickness.

From the pressure drop calculation using the equations from the microfluidics background section, the expected pressure drop calculated over the worst-case path is 964 Pa. In this case, for reliability, the procedure says that at least ten times this value should be taken to be the CLOSED pressure. This means, ideally, the CLOSED pressure is 9.6 kPa, and the OPEN pressure is at most half that which is 4.8 kPa, or as close as possible to

this given the capabilities of this technology. These numbers are pretty close to the valve capabilities as characterized by Rahbar et al.

Suppose that the spacing requires the use of the 6.5 x 2.5 mm magnet. The magnet is 3.5 mm away from the valve flap, so it exerts 60 mT of magnetic field at the flap. The thickest flap, of course, performs the best. Sacrificing reliability for power, allows for the use of either one of the other flaps as long as the leakage pressure in the CLOSED state is greater than the pressure of the longest path. This means that the magnet channel has to be long enough to drop the magnetic field from the original 60 mT to low enough to significantly drop the leakage pressure to an OPEN state. According to Rahbar et al., as discussed in the previous section, it is sufficient for the magnet to be 5 or 6 mm away from the flap.

From the graphs and equations, at the 3.5 mm distance away from the flap, the 0.5 mm flap has a 2.5 mT field, the 1.5 mm flap has a 5 mT field and the 2.25 mm flap has a 7 mT field. As stated, these measurements are reliable since they resemble the values measured by Rahbar at al. for their flap configuration.This means that the coil needs to exert at least this much field at 2 mm away in order to attract the magnet away from the flap, which gives 85 mA for the 0.5 mm flap, 190 mA for the 1.5 mm flap and 280 mT for the 2.25 mm flap. This is tested and works reliably since these magnetic field requirements are an overestimate because the area of the coil is larger than the area of the flap.

To test the power requirement in a more robust system, the same test is performed with the magnet channel being 5 mm. In this case, the coil required 170 mA for the 0.5 mm flap, 280 mA for the 1.5 mm flap and 440 mA for the 2.25 mm flap. For this reliable implementation, the power consumption is calculated to be 10.2 W. However, since the coil only needs to be on for at most a second to actuate the valve, the total energy consumption is at most 10.2 J.

6.2.2 Power vs Reliability

One way to reduce the power consumption is to reduce the size of the magnet channel. With this being lowered, the coil does not require as much current to attract the magnet, thus lowering the power consumption. Three kinds of problems may arise from this if the channel is too short. There is less current passing through the coil, therefore less of a repulsive as

well as an attractive force is present, which means that the latching component has to be smaller in order to repel the magnet from itself. However, with a small latching component and being closer to the valve flap, the magnet is less likely to stay attached to the latching component, and may, whether on its own or because of some external force, latch back onto the valve flap.

The second reason a shorter magnetic channel could be a problem is by reducing the overall reliability through insufficient distance between the OPEN and CLOSED states of the valve. In this case, the valve that is supposed to be OPEN might not allow enough of a flow rate through it and the pressure continues to build up, eventually reaching the leakage pressure of the valve that is supposed to be in a CLOSED state. This problem is mitigated by using a lower flow rate so the valves are more likely to be able to keep up with it. Another mitigating factor is how long the valves need to stay in such a state. In testing with two valves, even if the leakage pressure of one of the valves is only slightly lower than the other, that valve always leaks first and depending on the difference the other valve may or may not leak after an extended amount of time. If a 30 mT difference is provided between the OPEN valve and the CLOSED valve, by having the 6.5 mm x 2.5 mm magnet adjacent to the CLOSED valve and 2 mm away from the OPEN valve and if the system is in this configuration for several minutes at 250 µL/min flow rate, the CLOSED valve did not leak. Further testing needs to be done to investigate this for different kinds of valves and different flow rates. Perhaps once the specific microvalves are chosen for the system based on the other parameter, this can be tested more in depth and the system optimized for power consumption by limiting the reliability to the required amount.

The third reason that power impacts reliability is through the heat caused by the coils operating at high current. Unlike the other two, in this case, high power usage negatively impacts reliability because the higher the current, the more the coils heat up and can potentially cause damage, or, if they are connected directly to a voltage source, they draw less current since their resistance decreases. To address this, cooling systems must be employed to regulate the temperature of the coils. Of course, the small coils heat up the most, thus greatly lowering their internal resistance as seen in Figure 6.2. The larger the coil, the more constant its current is over a long time of being operational.

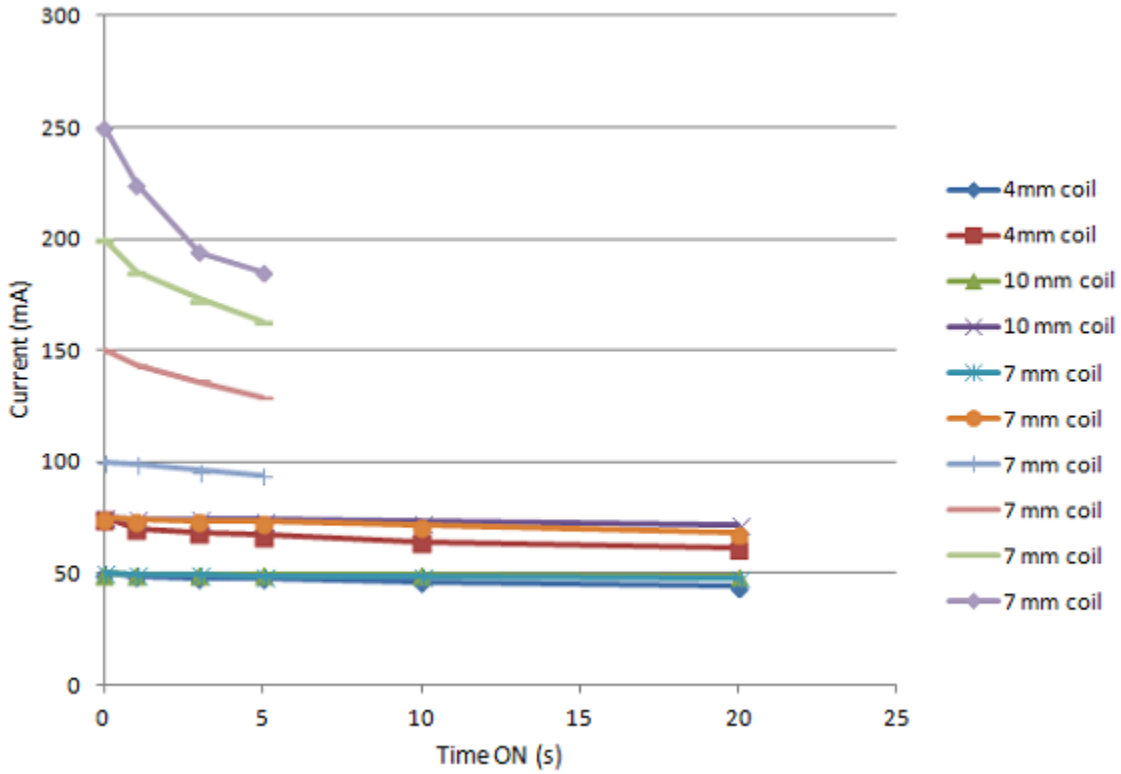

Figure 6.2: Current over time for different coils with constant voltage

6.2.3 Power Versus Area Trade-off

Potential ways to improve the power performance versus area usage of the control system are discussed in this section. The first thing to consider is the large area overhead due to the size and strength of the magnets. However, using smaller magnets leads to requiring a thicker valve flap to achieve the same leakage pressure, which means that attracting the magnet away from the flap is more difficult, leading to higher power consumption.

One way to lower the area requirement is to use valves that do not require a permanent magnet to actuate them. This can be accomplished by using a weaker valve employing technology that helps keep it closed for a desirable amount of pressure. Thus, this valve can be considered naturally closed and if one of the other pathways is open (as long as the leakage pressure of the OPEN pathway is considerable less than the CLOSED leakage pressure of this weaker valve), the fluid flows through the OPEN pathway. If none of the other pathways are open, the pressure builds up until this weaker valve gives out and fluid flows through it (assuming the leakage pressure of this valve is less than the leakage

pressure of the other valves in their CLOSED state). It can also be thought of as a naturally-open valve and can act as a default pathway or as a safety outlet to prevent breakage in case the pressure of the system builds up too much. Additional fabrication methods and configurations for improving the CLOSED state seal are discussed by Rahbar et al. in.

Of course, this procedure works best if each node in the microfluidics circuit has a low "fan-out". To be precise, for this it is better if each path leads to only a small number of valves, since each de-multiplexer only has one of these valves as in the right of Figure 6.3. Otherwise it is hard to control which of these valves the fluid goes through and more power is needed since power needs to be applied to keep them closed.

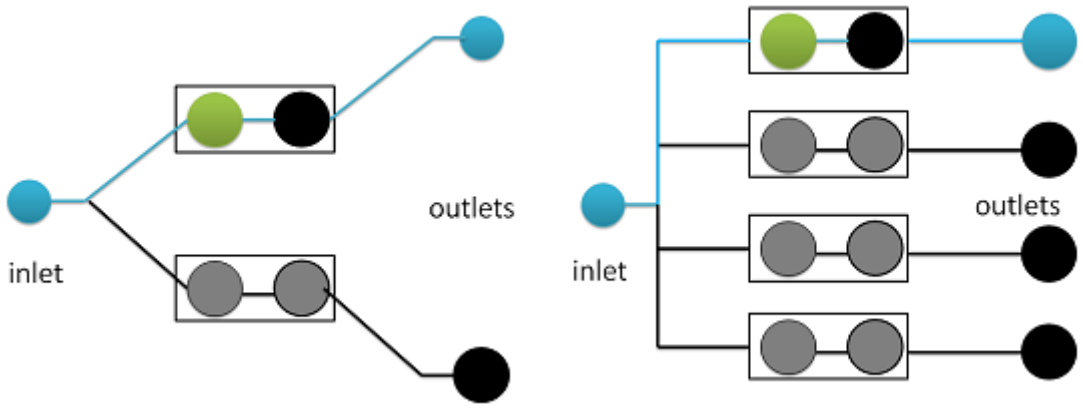

Figure 6.3: A coil-only valve in a low fan-out node vs a high fan-out node

In the image on the left, one channel splits into two channels. If all the de-multiplexing is done in such a way in the system, the power consumption can be halved by having one of the valves be a weak valve. In the image on the right, one channel is splitting into 4 channels. In this case, only one quarter of the power can be saved by having one of the valves be a weak valve. However, if only one to two splitting is used, in order to accomplish the same number of output channels to the input channel, the same amount of power is needed, and there would be an excess of extra area usage with the weak valves.

Alternatively, to minimize the area of the microfluidics layer at the cost of power consumption, valves that are controlled strictly by coils can be used. Their downside is that constant power must be supplied to the coils to operate them. The ideal way to do this is to have naturally closed valves by employing a technology such as magnetizing the whole substrate underneath to help keep them closed. In this case, they can be opened by having a coil situated above the valve flap, just over the chamber. If the valve chamber can be

made small enough, this minimizes the distance between the coil and the valve flap. The problem being that at the slow flow rates experienced in microfluidic systems, the valves, and consequently the coils, may need to be on for upwards of a few minutes. Even just assuming they need to be on for one minute at the same amount of current used in the control system with the magnet, this would mean an overall power increase of 60 times! This can be reduced by using less current as well as having a weak valve as previously discussed that does not need power to be opened. As long as this valve is on the pathway that is used more frequently the power consumption is reduced by the ratio of the amount of time this valve needs to be opened multiplied by the power consumption.

6.3 Control System Assembly

This section describes how the overall control system using the FPGA is designed and built. The device libraries are coded based on their specific requirements and they can be accessed through the control system.

6.3.1 Valve to FPGA Hardware Interface

The valves are connected to the FPGA using the header I/O pins as shown in Figure 6.4. This is the ideal way since the header pins provide the most homogeneous output pins which can be used to control the various devices.

As previously stated, the power for the electromagnetic coils comes directly from a constant voltage source power supply. In order to allow for easy switching of the power and bi-directional functionality, the coil is connected to a L293D dual H-Bridge [54]. This particular H-Bridge limits the output current to 600 mA, so if more than that is required in order to power the coils, a more powerful one needs to be used. The H-bridge requires two outputs from the FPGA in order to function: one output to run the current in the forward direction and the other to run the current in the reverse direction. If both the outputs are ON or OFF at the same time, then no current flows. This provides safety when switching between forward and reverse modes. Thus, the H-bridge provides the bi-directional current required to generate attracting and repelling magnetic fields through the coils.

The most the output pins can supply is 3.3 V, and the H-bridge itself requires 2.3 V minimum on its forward and reverse inputs, the FPGA outputs can be direclty connected

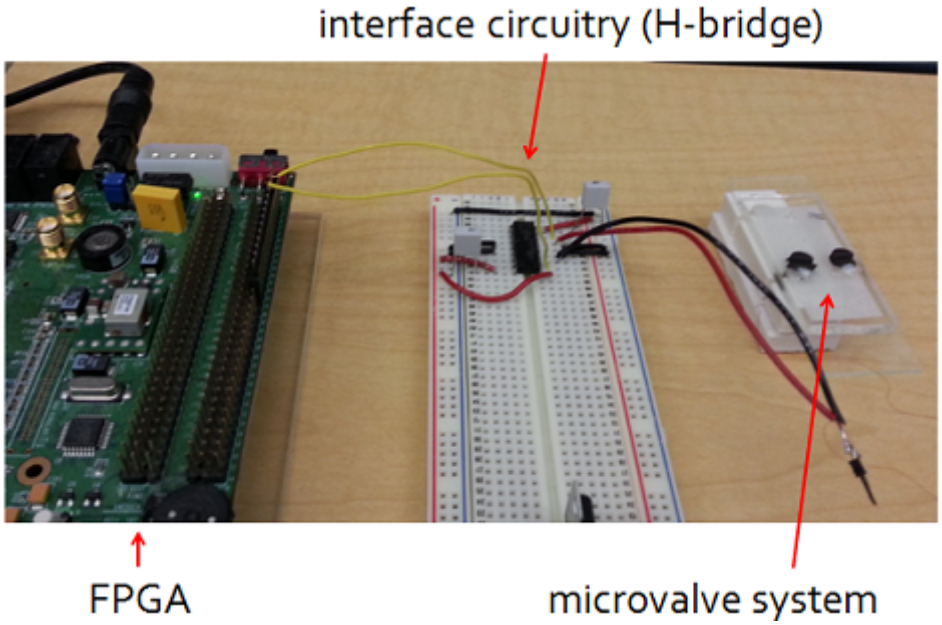

Figure 6.4: FPGA connected to the actuating system

to the H-bridge. If an FPGA that cannot meet this voltage requirement is used, the actual FPGA pins must be connected to MOSFETS used as switches between an input coming from the power supply and the H-bridge inputs. The MOSFETS have almost no base current, therefore, this does not cause loading on the FPGA pins and is unlikely to damage the device. Using an H-bridge and MOSFET does not add a significant overhead to the power consumption of the system.

6.4 Verifying Software and Hardware Control Solution

This section describes how the control system is tested. The control system is tested with a single valve connected to the header pins as described. Two outputs are used to specify the forward and reverse direction of the current. The control system is connected to the 10 mm coil and 6.5 mm magnet configuration used in the solution evaluation section.

6.5 Scaling the Control Solution

This section describes how the control system is scaled to any number of valves. As previously stated, each valve requires two I/Os to operate. That means that twice the number of I/Os as valves are required in the array in order to control all of the valves. As previously

discussed FPGAs with up to 1200 I/Os can be purchased for use in the system, so lack of pins to control the valves with should not be a problem. However, if a device with less I/Os is used to control a large array, some de-multiplexing scheme needs to be constructed by which the I/Os themselves specify the sates of all the valves in the system and circuitry is used to convert this to the actual valve control. Each valve has three states (forward, reverse and off), thus for n valves, $3n$ states are needed to specify all the states of the valves. Differentiating between the two off sates of the H-bridge (both inputs high and both inputs low) requires $4n$ states to specify the position of all the valves. This is encoded in binary using output pins, so $2\sqrt{n}$ pins are required to fully specify the configuration of the valves.

The power consumption of the system can likewise be scaled smartly. As seen in the solution evaluation section, 10.2 W is used to power a single valve. So, it would be easy to say that the total power of the valves in the system is simply $10.2n$ Watts. But this assumes that all the valves in the system need to be powered on at once. The valves do not take a long time to switch states (under a second) and the microfluidics application does not need to be quickly reconfigured since processes are quite slow, therefore the power consumption can be reduced all the way down to the power it takes to operate a single valve by only allowing one coil to be switched on at a time. If it takes less than one second to configure a valve and using a system with 100 valves, then it would take about a minute and a half to reconfigure every valve in the system. If this is too slow for some application, only a suitable number of coils are switched on at once to allow for faster configuration.

Chapter 7

Conclusion and Future Work

Over the last few years, researchers have made numerous advancements in the field of µTAS that lead to more and more integrated and multi-purpose microfluidics systems. The µROAMS project is one such microfluidics platform that brings the reconfigurability of FPGA style logic into the microfluidics domain, complete with a set of CAD tools to help users design, reprogram and use their systems.

Magnetic microvalves have existed in microfluidics research for quite some time, however, the difficulty of actuating them reliably has prevented them from being used in many applications.

To reiterate, the major contributions of this thesis are as follows:

- I identify and evaluate the different parameters of an electromagnetic actuating solution
- I specify and test a procedure for optimizing the electromagnetic actuating solution based on the parameter evaluation and system requirements
- I integrate the physical device actuation with the digital control system by determining the interface circuitry
- I build the code framework for the user interface and device libraries to allow the whole system to be easily used and upgraded

7.1 Conclusion

This thesis focuses on designing a control system for a reconfigurable microfluidics platform. In particular, this control system is designed by focusing on optimizing the magnetic microvalve actuating system for power consumption and incorporating it into the FPGA system. This means that different parameters relevant to the operation of the control system are tested and optimized for performance and power. After that, then interface between the actual physical device and the user interface running on the FPGA is created.

To reiterate, the parameters tested are as follows:

-the magnets (type and dimensions)

-the electromagnetic coils (size, type of wire, number of turns, type of core)

-the height of the magnet channel

-the distance between the valves

-the amount of current

-the thickness of the valve flap

-the height of the PDMS substrate

-the fluidic pressure to run the chip at

-fluidic pressure difference between the open and closed state

Equations are formulated to predict the interactions of these parameters and their results are compared to the measured results. A procedure is specified for determining the values of these parameters to optimize for power consumption. The last three parameters of the aforementioned list should be predetermined requirements based on the specifics of the chip and technology used, thus are not real parameters. The coils should be selected based on size requirement and power efficiency, the magnet based on size requirement, and then the thickness of the valve flap based on the required pressure it needs to withstand. The height of the magnet channel is selected to allow enough of a difference between the CLOSED and OPEN states for the system to perform reliably and, finally, the amount of current is selected to allow the coil to attract the magnet over the distance of the magnet channel.

7.1.1 Control System Performance

The procedure for selecting the parameters is tested in the actuating system for a certain set of parameters and the results imply the equations and approximations work quite well

to predict the final result. The chosen values work as expected and changes can easily be made to improve certain aspects, such as trading off power consumption for reliability.

7.1.2 Possible Improvements

Although the procedure for choosing the parameters of the actuating system to works quite well for optimizing for power given the system and device requirements, improvements can still be made. In particular, better equations can be devised for parts of the procedure that do not have reliable equations to rely on. As well, different kinds of valves and maybe different kinds of actuation systems can be used in order to lower the area overhead of using the magnets in the control system.

7.2 Future Work

This work has laid out a foundation for determining a power-efficient actuating scheme for magnetic components and sketched out a framework for the user interface and device libraries to integrate such components in the system, so there is much potential for expanding this work.

A test platform needs to be fabricated and in-system tests need to be performed on the solution for driving the valves. Reliability tests on how a large array of valves behave under such circumstances as long continuous operation, large number of switches from open to closed and vice versus are also needed.

In particular, the material for the magnet channel needs to be selected and tested for friction and tightness to prevent possible problems such jamming of the magnet or flipping the magnet. Future work should also include testing how the difference in the amount of pressure the valves withstand in the CLOSED and OPEN states affects the reliability of routing and how this scales with size of array as well as the configuration of the connections in the array.

The framework for the user interface needs to be integrated with the rest of the run-time CAD tools and the control parameters need to be incorporated into this CAD toolflow. The library of components will be expanded to include other kinds of components and variations thereof. The user parameters need to be determined in terms of the choices a researcher who is using these tools to design their own microfluidics chip as well as for an end user

using the chip. Support for hybrid and compound components will be investigated as well as the ability to create them from existing components. For example, a heating chamber can contains a mixer and a sensor. Additional support will be added to allow the user to easily program the chip both in terms of reconfigurability and running of tests. Perhaps the current command mode of the user interface can be expanded into a scripting interface with an easy to use microfluidics scripting language.

Bibliography

[1] M. Rahbar, L. Shannon, and B. L. Gray, "Arrayable microfluidic valves based on rare earth permanently magnetic polymer for use in microfluidic flow switching," *MicroTAS*, 2014.

[2] ——, "Microfluidic active mixers employing ultra-high aspect-ratio rare-earth magnetic nano-composite polymer artificial cilia," *Journal of Micromechanics and Microengineering*, vol. 24, no. 2, p. 025003, 2014.

[3] C. T. Culbertson, T. G. Mickleburgh, S. A. Stewart-James, K. A. Sellens, and M. Pressnall, "Micro total analysis systems: Fundamental advances and biological applications," *Analytical chemistry*, vol. 86, no. 1, pp. 95–118, 2013.

[4] V. Srinivasan, V. K. Pamula, and R. B. Fair, "An integrated digital microfluidic lab-on-a-chip for clinical diagnostics on human physiological fluids," *Lab on a Chip*, vol. 4, no. 4, pp. 310–315, 2004.

[5] B. L. Gray, "A review of magnetic composite polymers applied to microfluidic devices," *Journal of The Electrochemical Society*, vol. 161, no. 2, pp. B3173–B3183, 2014.

[6] F. M. White, "Fluid mechanics, wcb," *ed: McGraw-Hill, Boston*, 1999.

[7] K. W. Oh and C. H. Ahn, "A review of microvalves," *Journal of micromechanics and microengineering*, vol. 16, no. 5, p. R13, 2006.

[8] S. C. Terry, J. H. Jerman, and J. B. Angell, "A gas chromatographic air analyzer fabricated on a silicon wafer," *Electron Devices, IEEE Transactions on*, vol. 26, no. 12, pp. 1880–1886, 1979.

[9] N. Pamme, "Magnetism and microfluidics," *Lab on a Chip*, vol. 6, no. 1, pp. 24–38, 2006.

[10] L. K. Lagorce, O. Brand, and M. G. Allen, "Magnetic microactuators based on polymer magnets," *Microelectromechanical Systems, Journal of*, vol. 8, no. 1, pp. 2–9, 1999.

[11] W. Wang, Z. Yao, J. C. Chen, and J. Fang, "Composite elastic magnet films with hard magnetic feature," *Journal of Micromechanics and microengineering*, vol. 14, no. 10, p. 1321, 2004.

[12] B. Yang, B. Wang, and W. K. Schomburg, "Structure design and fabrication of a bistable microvalve," in *Mechanic Automation and Control Engineering (MACE), 2011 Second International Conference on*. IEEE, 2011, pp. 1635–1638.

[13] A. Khosla, J. L. Korčok, B. L. Gray, D. B. Leznoff, J. Herchenroeder, D. Miller, and Z. Chen, "Fabrication of uv-micro-patternable permanent micro magnets for lab on a chip and mems," in *SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring*. International Society for Optics and Photonics, 2010, pp. 76 461L–76 461L.

[14] D. D. Hilbich, A. Khosla, B. L. Gray, and L. Shannon, "Bidirectional magnetic microactuators for utas," in *SPIE MOEMS-MEMS*. International Society for Optics and Photonics, 2011, pp. 79 290H–79 290H.

[15] T. M. Squires and S. R. Quake, "Microfluidics: Fluid physics at the nanoliter scale," *Reviews of modern physics*, vol. 77, no. 3, p. 977, 2005.

[16] L. Scampavia, G. Blankenstein, J. Ruzicka, and G. Christian, "A coaxial jet mixer for rapid kinetic analysis in flow injection and flow injection cytometry," *Analytical chemistry*, vol. 67, no. 17, pp. 2743–2749, 1995.

[17] J. V. Timonen, C. Johans, K. Kontturi, A. Walther, O. Ikkala, and R. H. Ras, "A facile template-free approach to magnetodriven, multifunctional artificial cilia," *ACS applied materials & interfaces*, vol. 2, no. 8, pp. 2226–2230, 2010.

[18] K. S. Ryu, K. Shaikh, E. Goluch, Z. Fan, and C. Liu, "Micro magnetic stir-bar mixer integrated with parylene microfluidic channels," *Lab on a Chip*, vol. 4, no. 6, pp. 608–613, 2004.

[19] A. Hatch, A. E. Kamholz, G. Holman, P. Yager, and K. F. Böhringer, "A ferrofluidic magnetic micropump," *Microelectromechanical Systems, Journal of*, vol. 10, no. 2, pp. 215–221, 2001.

[20] A. V. Lemoff and A. P. Lee, "An ac magnetohydrodynamic micropump," *Sensors and Actuators B: Chemical*, vol. 63, no. 3, pp. 178–185, 2000.

[21] T. Pan, S. J. McDonald, E. M. Kai, and B. Ziaie, "A magnetically driven pdms micropump with ball check-valves," *Journal of Micromechanics and Microengineering*, vol. 15, no. 5, p. 1021, 2005.

[22] H. Hartshorne, C. J. Backhouse, and W. E. Lee, "Ferrofluid-based microchip pump and valve," *Sensors and Actuators B: Chemical*, vol. 99, no. 2, pp. 592–600, 2004.

[23] R. Olaru, C. Petrescu, and R. Hertanu, "A novel double-action actuator based on ferrofluid and permanent magnets," *Journal of Intelligent Material Systems and Structures*, p. 1045389X12449916, 2012.

[24] J. Joung, J. Shen, and P. Grodzinski, "Micropumps based on alternating high-gradient magnetic fields," *Magnetics, IEEE Transactions on*, vol. 36, no. 4, pp. 2012–2014, 2000.

[25] S. Østergaard, G. Blankenstein, H. Dirac, and O. Leistiko, "A novel approach to the automation of clinical chemistry by controlled manipulation of magnetic particles," *Journal of Magnetism and Magnetic Materials*, vol. 194, no. 1, pp. 156–162, 1999.

[26] J. R. Basore and L. A. Baker, "Applications of microelectromagnetic traps," *Analytical and bioanalytical chemistry*, vol. 403, no. 8, pp. 2077–2088, 2012.

[27] I. Safarik, M. Safarikova, and S. Forsythe, "The application of magnetic separations in applied microbiology," *Journal of Applied Bacteriology*, vol. 78, pp. 575–575, 1995.

[28] H. Lee, A. M. Purdon, V. Chu, and R. M. Westervelt, "Controlled assembly of magnetic nanoparticles from magnetotactic bacteria using microelectromagnets arrays," *Nano Letters*, vol. 4, no. 5, pp. 995–998, 2004.

[29] J. W. Judy and R. S. Muller, "Magnetically actuated, addressable microstructures," *Microelectromechanical Systems, Journal of*, vol. 6, no. 3, pp. 249–256, 1997.

[30] J. J. Bernstein, W. P. Taylor, J. D. Brazzle, C. J. Corcoran, G. Kirkos, J. E. Odhner, A. Pareek, M. Waelti, and M. Zai, "Electromagnetically actuated mirror arrays for use in 3-d optical switching applications," *Microelectromechanical Systems, Journal of*, vol. 13, no. 3, pp. 526–535, 2004.

[31] T. Deng, G. M. Whitesides, M. Radhakrishnan, G. Zabow, and M. Prentiss, "Manipulation of magnetic microbeads in suspension using micromagnetic systems fabricated with soft lithography," *Applied physics letters*, vol. 78, no. 12, pp. 1775–1777, 2001.

[32] J. Casals-Terré, M. Duch, J. Plaza, J. Esteve, R. Perez-Castillejos, E. Vallés, and E. Gómez, "Design and characterization of a magnetic digital flow regulator," *Sensors and Actuators A: Physical*, vol. 162, no. 1, pp. 107–115, 2010.

[33] C.-H. Cheng, C. Chao, Y.-N. Cheung, L. Xiao, M. Yang, and W. Leung, "A transcutaneous controlled magnetic microvalve based on iron-powder filled pdms for implantable drug delivery systems," in *Nano/Micro Engineered and Molecular Systems, 2008. NEMS 2008. 3rd IEEE International Conference on.* IEEE, 2008, pp. 1160–1163.

[34] S. Böhm, G. Burger, M. Korthorst, and F. Roseboom, "A micromachined silicon valve driven by a miniature bi-stable electro-magnetic actuator," *Sensors and Actuators A: Physical*, vol. 80, no. 1, pp. 77–83, 2000.

[35] W. Jian and L. Tonggang, "Analytical modeling and optimization for a microvalve actuated by magnetic fluid," in *Digital Manufacturing and Automation (ICDMA), 2013 Fourth International Conference on.* IEEE, 2013, pp. 250–252.

[36] Q. Cao, X. Han, and L. Li, "Configurations and control of magnetic fields for manipulating magnetic particles in microfluidic applications: magnet systems and manipulation mechanisms," *Lab on a Chip*, vol. 14, no. 15, pp. 2762–2777, 2014.

[37] G. D. Gray and P. A. Kohl, "Magnetically bistable actuator: Part 1. ultra-low switching energy and modeling," *Sensors and Actuators A: Physical*, vol. 119, no. 2, pp. 489–501, 2005.

[38] C. Zhi, T. Shinshi, M. Saito, and K. Kato, "Planar-type micro-electromagnetic actuators using patterned thin film permanent magnets and mesh type coils," *Sensors and Actuators A: Physical*, vol. 220, pp. 365–372, 2014.

[39] Y. Shinozawa, T. Abe, and T. Kondo, "A proportional microvalve using a bi-stable magnetic actuator," in *Micro Electro Mechanical Systems, 1997. MEMS'97, Proceedings, IEEE., Tenth Annual International Workshop on.* IEEE, 1997, pp. 233–237.

[40] Y. Melikhov, S. Lee, D. C. Jiles, D. Schmidt, M. D. Porter, and R. Shinar, "Microelectromagnetic ferrofluid-based actuator," *Journal of applied physics*, vol. 93, no. 10, pp. 8438–8440, 2003.

[41] A. Goldman, *Handbook of modern ferromagnetic materials*. Springer Science & Business Media, 2012, vol. 505.

[42] Xilinx, "Virtex 7 device page," June 2015. [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html

[43] ——, "Virtex-7 t and xt fpgas data sheet: Dc and ac switching characteristics," March 2015. [Online]. Available: www.xilinx.com

[44] K. Subramaniyam, "Proven power reduction with xilinx ultrascale fpgas," *Xilinx White Paper*, Oct 2015. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp466-proven-ultrascale-power-leaders.pdf

[45] J. S. McCaskill and P. Wagler, "From reconfigurability to evolution in construction systems: spanning the electronic, microfluidic and biomolecular domains," in *Field-programmable logic and applications: the roadmap to reconfigurable computing*. Springer, 2000, pp. 286–299.

[46] J. McDaniel, B. Parker, and P. Brisk, "Simulated annealing-based placement for microfluidic large scale integration (mlsi) chips," in *Very Large Scale Integration (VLSI-SoC), 2014 22nd International Conference on*. IEEE, 2014, pp. 1–6.

[47] K. Hu, T. A. Dinh, T.-Y. Ho, and K. Chakrabarty, "Control-layer optimization for flow-based mvlsi microfluidic biochips," in *Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2014 International Conference on*. IEEE, 2014, pp. 1–10.

[48] Xilinx, "Ultrascale fpga product tables and product selection guide," 2015. [Online]. Available: http://www.xilinx.com/support/documentation/selection-guides/ultrascale-fpga-product-selection-guide.pdf

[49] ——, "Ml505/ml507/ml507 evaluation platform user guide," May 2011. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf

[50] F. Bell, "Hall effect gaussmeters rev g." [Online]. Available: http://fwbell.com/downloads/manuals/5170-5180-Manual-Rev-G.pdf

[51] B. D. Cullity and C. D. Graham, *Introduction to magnetic materials*. John Wiley & Sons, 2011.

[52] TRESNA, "Tube digital micrometer." [Online]. Available: http://www.tresnainstrument.com/product/m05.html

[53] D. Vokoun, M. Beleggia, L. Heller, and P. Šittner, "Magnetostatic interactions and forces between cylindrical permanent magnets," *Journal of Magnetism and Magnetic Materials*, vol. 321, no. 22, pp. 3758–3763, 2009.

[54] T. Instruments, “L293, l293d quadruple half-h drivers,” Nov 2004. [Online]. Available: http://www.ti.com/lit/ds/symlink/l293.pdf

Appendix A

Code

A.1 User Interface

```
/*-------------------------------------------------------------------------
 * interface..h
 *
 * Contains the definitions, structures and functions needed
 * to run the user interface for the uROAMS project.
 * Contains the structure definitions for a device and connection
 * as well as for a COMMAND mode command and SYSTEM mode test.
 *
 * Author: Veronica Cojocaru
 *
 *
 * Revision History:
 *    Dec 9, 2014 - Initial version
 *      August 5, 2015 - Latest revision
 -------------------------------------------------------------------------*/

#ifndef INTERFACE_H
#define INTERFACE_H


/*------------------------------ INCLUDES ---------------------------------*/
#include <stdio.h>
#include "xparameters.h"    /* generated system parameters                  */
#include "xbasic_types.h"   /* Xilinx basic types for device drivers        */
#include "xgpio.h"         /* layer 0/1 GPIO device driver                 */


/*------------------------------ DEFINES ----------------------------------*/
//if using a Xilinx system that has the UART bug of receiving NULL characters
    after every relavant character use this define
#define UART_BUG

//defines for the mode the user interface is operating in
#define SYSTEM_MODE     10
#define COMMAND_MODE 11
#define TEST_MODE    12
```

```
#define TRUE         1
#define FALSE        0

#define INPUT_LENGTH 20    //max length of an input string

#define NUMBER_PARAMETERS  5  //max number of parameter for driver functions

#define DELAY_COUNT  100         //about .1 sec delay

#define NUM_GPIOS 1        //number of GPIOs needed in the system
XGpio Gpio[NUM_GPIOS];

//setting the reading length of an input string depending on whether it is used
    in a system with the UART bug
#ifdef UART_BUG
#define UART_INPUT_LENGTH  INPUT_LENGTH*2
#endif

#ifndef UART_BUG
#define UART_INPUT_LENGTH  INPUT_LENGTH
#endif

//defines the total number of device classes available in the system
#define NUM_DEVICE_CLASS   7
//define device type IDs
#define VALVE_CLASS        30
#define MIXER_CLASS        40
#define PUMP_CLASS         50
#define HEATER_COOLER_CLASS   60
#define SENSOR_CLASS    70
#define INPUT_CLASS        80
#define OUTPUT_CLASS    90

//defines the specific device types
#define MAGNETIC_VALVE     31
#define MAGNETIC_MIXER     41

#define CONNECTION_INPUT   7
#define CONNECTION_OUTPUT  8
#define CONNECTION_I_O     9

#define printf xil_printf   // a smaller footprint printf

//defines a function pointer for dynamically calling functions
typedef void (*funct_p)(void);
typedef void (*funct_p2)(int parameters[NUMBER_PARAMETERS]);


/*------------------------------ FUNCTIONS ---------------------------------*/
void runSystemMode( void );      //perform actions of SYSTEM mode
void runCommandMode( void );  //perform actions of COMMAND mode
void runTestMode( void );     //perform actions of TEST mode

//adds a new command to the list of COMMANd mode commands
void add_command(char* CommandName, funct_p2 swFunct);

//adds a new system test to the list of SYSTEM mode tests
```

```
void add_system_test(char* SysTestName, funct_p swFunct, char* Description);

//system test functions
void qPCR_test( void );
void change_to_system( void );
void change_to_command( void );
void change_to_test( void );


/*------------------------- LINKED LIST STRUCTURES -------------------------*/
/* Linked list structure for the list of available SYSTEM MODE TESTS */
struct SystemTest {
   char* SysTestName;            /* the name of the test */
   funct_p swFunct;           /* the associated function */
   char* Description;            /* describes what the test does*/
   struct SystemTest* next;      /* points to next structure in list */
   struct SystemTest* prev;      /* points to previous structure in list */
};

/* Linked list structure for the list of available COMMAND MODE commands */
struct Command {
   char* CommandName;            /* the name of the test */
   funct_p2 swFunct;          /* the associated function */
   struct Command* next;         /* points to next structure in list */
   struct Command* prev;         /* points to previous structure in list */
};


/*-------------------- CONNECTION CLASS STRUCTURE --------------------------*/
struct Connection
{
   int DeviceTypeID;             //the class type this connection leads to
   int ConnectionDirection;         //specifies if the connection is an input/
       output or bidirectional
   void* ConnectedDevice;           //pointer to the structure of the connected
       device
   struct Connection* nextConnection;  //points to the next connection of this
       device
};

struct Connection* newConnection(int DeviceTypeID, void* ConnectedDevice, struct
     Connection* nextCommection);


/*---------------------- DEVICE CLASS STRUCTURE ---------------------------*/
struct Device {
int classID;      //specifies the class of device this object belongs to
int deviceID;     //specifies which instance of the class this is

void* device;     //pointer to device structure

float width;      // specifies the dimensions of the object for the placer in mm
float length;
float height;

float x_loc;      //specifies the location of the object on the chip
float y_loc;
```

```
int num_connections; //specifies the number of connections from this device to
    other devices
struct Connection* connectionList; //pointer to linked list of connections for
    this device
};

struct Device* newDevice(int classID, int deviceID, void* device, float width,
    float length, float height, float x_loc, float y_loc, int num_connections,
    struct Connection* connectionList);


/*------------------- EXTERNAL GLOBAL VARIABLE DEFINITIONS -----------------*/
extern struct SystemTest SystemTestList;
extern struct Command CommandList;
extern struct Device DummyDevice;
extern struct Connection DummyConnection;


#endif //(INTERFACE_H)



/*-----------------------------------------------------------------------------
 * interface.c
 *
 * Contains the main loop for the user interface for uROAMS
 * Contains the functions for the main modes of the user interface
 * as well as the system level tests
 *
 * Three modes are available for the user interface main loop
 * - SYSTEM mode displays a list of the available system tests
 * and prompts the user for input
 * - COMMAND mode prompts the user for a command
 * - TEST mode prints a list of the devices in the syste and
 * prompts the user for input
 *
 * Modes can be changed by typing the appropriate input in the prompt
 *
 * Author: Veronica Cojocaru
 *
 *
 * Revision History:
 *    Dec 9, 2014 - Initial version
 *      August 5, 2015 - Latest revision
 ----------------------------------------------------------------------------*/


/*------------------------------- INCLUDES ---------------------------------*/
#include <string.h>
#include <stdlib.h>
#include "populate_chip.h"
#include "interface.h"

/*--------------------------- GLOBAL VARIABLES -----------------------------*/
int mode;                       //keeps track of the mode of the program
char input[UART_INPUT_LENGTH];      //for user to type in input
char input_trunc[UART_INPUT_LENGTH];//for truncating the input
```

```
//head of system test linked list
struct SystemTest SystemTestList = { "COMMAND_MODE", change_to_command, "Go into
     COMMAND mode", NULL, NULL };

//head of commands linked list
struct Command CommandList = { "SYSTEM_MODE", change_to_system, NULL, NULL };

struct Device DummyDevice = {0, 0, NULL, 0, 0, 0, 0, 0, 0, NULL};
struct Connection DummyConnection = {0, 0, NULL, NULL};

//function variable for running commands
void (*f)();
void (*f2)(int array[NUMBER_PARAMETERS]);


/*------------------------------ FUNCTIONS ---------------------------------*/
//main unser interface loop
int main()
{
   //start off in system mode
   mode = SYSTEM_MODE;

   //populate initial linked list of system tests and commands
   //in the future this should be done from a file
   populate_tests_commands();
   initialize_chip();

   printf("\r\n");
   printf("\r\n");
   printf("         ___   ___    _    __  __  ___ \r\n");
   printf("  _  _  | _ \\ /   \\  /_\\  |  \\/  |/ __|\r\n");
   printf(" | || | |   / |(_)| / _ \\ | |\\/| |\\__ \\\r\n");
   printf("  \\_  | |_|_\\ \\___//_/ \\_\\|_|  |_||___/\r\n");
   printf("    | |                                 \r\n");
   printf("    |_|                                 \r\n");
   printf("\r\n");

   //user interface main loop
   while(1)
   {
      //check to see what mode you are in
      //in system mode it displays the available system level tests and asks the
           user to choose from them
      if(mode == SYSTEM_MODE)
      {
         runSystemMode();
      }

      //in command mode the user can enter commands to execute
      else if(mode == COMMAND_MODE)
      {
         runCommandMode();
      }
      //in test mode it displays a list of devices available in the system
      else if(mode == TEST_MODE)
      {
         runTestMode();
      }
```

```
      else
      {
         printf("Execution shouldn't have reached here.  ERROR!  Exiting Program
             . \n");
         return -1;
      }
   }
   return 0;
}

//performs system mode actions
//prints out list of available system tests and prompts for input
void runSystemMode( void )
{
   //variable definitions
   int i, valid, length;
   struct SystemTest *AvailableTest;

   printf("The available system test are: \r\n");
         printf("TEST");
         for(i=1; i<INPUT_LENGTH; i++)
            printf(" ");
         printf("DESCRIPTION \r\n");
         //print out the list of available tests
         AvailableTest = &SystemTestList;
         while(AvailableTest != NULL)
         {
            printf("%s", AvailableTest->SysTestName);
            length = strlen(AvailableTest->SysTestName);
            length = INPUT_LENGTH - length +3;
            for(i=0; i< length; i++)
               printf(" ");
            printf("%s", AvailableTest->Description);
            printf("\r\n");
            AvailableTest = AvailableTest->next;
         }
         //input prompt
         printf("\r\n");
         printf("Please enter the name of the test to perform: ");
         scanf("%s", (char*) &input);

//if using a system that has the UART bug, need to get rid of the NULL
    characters
#ifdef UART_BUG
         for(i=1;i<INPUT_LENGTH;i++)
            {
            input[i] = input[2*i];
            }
#endif
         printf("%s \r\n", &input);
      //check the linked list of commands and see if the input matches
      AvailableTest = &SystemTestList;
      valid = FALSE;
      while(AvailableTest != NULL)
      {
         if(strcmp(AvailableTest->SysTestName, &input)==0)
         {
            valid = TRUE;
```

```
            break;
         }
         AvailableTest = AvailableTest->next;
      }
      //if a match was found run the function associated with the specified
          system test
      if(valid == TRUE)
      {
         f = AvailableTest->swFunct;
         f();
      }
      else
      {
         printf("INVALID TEST \r\n");
      }
}


//performs command mode actions
//prompts user for command
void runCommandMode(void)
{
   //variable declarations
   int i, j, valid;
   struct Command *UserCommand;
   char input_ints[NUMBER_PARAMETERS][10];
   char *ptr, *str;
   int num_params;
   int parameters[NUMBER_PARAMETERS];

   //input prompt
         printf("Input> ");
         scanf("%s", (char*) &input);

//if using a system that has the UART bug, need to get rid of the NULL
    characters
#ifdef UART_BUG
         for(i=1;i<INPUT_LENGTH;i++)
            {
            input[i] = input[2*i];
            }
#endif
         printf("%s \r\n", &input);
      //truncate the input to only contain the command and not the parameters
      for(i=0; i<INPUT_LENGTH; i++)
      {
         if( input[i] == ':' )
         {
            input_trunc[i] = '\0';
            break;
         }
         input_trunc[i] = input[i];
      }

      //extract the parameter values and put them in parameter array
      for(i = 0; i<NUMBER_PARAMETERS; i++)
      {
         for(j = 0; j<10; j++)
```

```
            {
               input_ints[i][j] = '\0';
            }
         }

         num_params = 0;
         for(i=0; i<INPUT_LENGTH; i++)
         {
            if( input[i] == ':' )
            {
               num_params++;
               j = 0;
               continue;
            }
            if(num_params > 0)
            {
               input_ints[num_params-1][j] = input[i];
               j++;
            }
         }

         for(i = 0; i<NUMBER_PARAMETERS; i++)
         {
            str = input_ints[i];
            parameters[i] = strtol(str, ptr, 10);
         }

         //check the linked list of commands and see if the input matches
         UserCommand = &CommandList;
         valid = FALSE;
         while( UserCommand != NULL)
         {
            if(strcmp(UserCommand->CommandName, &input_trunc)==0)
            {
               valid = TRUE;
               break;
            }
            UserCommand = UserCommand->next;
         }
         //if a match was found run the function associated with the user command
         if(valid == TRUE)
         {
            f2 = UserCommand->swFunct;
            f2(parameters);
         }
         else
         {
            printf("INVALID COMMAND \r\n");
         }
}

//perform TEST mode actions
//prints out a list of devices in system and prompts for input
void runTestMode()
{
   int i, j, num_con;
   struct ValveClass *valve, *v2;
   struct MixerClass *mixer, *m2;
```

```
   struct Connection *con;
   struct Device *dev, *dev_con;

   printf("\n\r");

   //display the valves in the system
#ifdef NUM_VALVES
   printf("VALVES in the system are: \r\n");
   printf("VALVE                    CONNECTIONS \r\n");
   for(i=0; i<NUM_VALVES; i++)
   {
      printf("Magnetic Valve %d         ", i);
      valve = valve_array + i;
      num_con = (valve->thisDevice)->num_connections;
      con = (valve->thisDevice)->connectionList;
      for (j=0; j<num_con; j++)
      {
         if(con->ConnectionDirection == CONNECTION_INPUT)
            printf("INPUT from ");
         else if(con->ConnectionDirection == CONNECTION_OUTPUT)
            printf("OUTPUT to ");
         else
            printf("I/O: ");
         if(con->DeviceTypeID == MAGNETIC_VALVE)
         {
            v2 = con->ConnectedDevice;
            dev_con = v2->thisDevice;
            printf("Magnetic Valve %d", dev_con->deviceID);
         }
         else if (con->DeviceTypeID == MAGNETIC_MIXER)
         {
            m2 = con->ConnectedDevice;
            dev_con = m2->thisDevice;
            printf("Magnetic Cilia %d", dev_con->deviceID);
         }
         else if (con->DeviceTypeID == INPUT_CLASS)
         {
            dev_con = con->ConnectedDevice;
            printf("Chip Input %d", dev_con->deviceID);
         }
         else if (con->DeviceTypeID == OUTPUT_CLASS)
         {
            dev_con = con->ConnectedDevice;
            printf("Chip Output %d", dev_con->deviceID);
         }
         printf(";   ");
         con = con->nextConnection;
      }
      printf("\n\r");
   }
   printf("\n\r");
#endif

   //display the mixers in the system
#ifdef NUM_MIXERS
   printf("MIXERS in the system are: \r\n");
   printf("MIXER                    CONNECTIONS \r\n");
   for(i=0; i<NUM_MIXERS; i++)
```

```
   {
      printf("Magnetic Cilia %d         ", i);
      mixer = mixer_array + i;
      num_con = (mixer->thisDevice)->num_connections;
      con = (mixer->thisDevice)->connectionList;
      for (j=0; j<num_con; j++)
      {
         if(con->ConnectionDirection == CONNECTION_INPUT)
            printf("INPUT from ");
         else if(con->ConnectionDirection == CONNECTION_OUTPUT)
            printf("OUTPUT to ");
         else
            printf("I/O: ");
         if(con->DeviceTypeID == MAGNETIC_VALVE)
         {
            v2 = con->ConnectedDevice;
            dev_con = v2->thisDevice;
            printf("Magnetic Valve %d", dev_con->deviceID);
         }
         else if (con->DeviceTypeID == MAGNETIC_MIXER)
         {
            m2 = con->ConnectedDevice;
            dev_con = m2->thisDevice;
            printf("Magnetic Mixer %d", dev_con->deviceID);
         }
         else if (con->DeviceTypeID == INPUT_CLASS)
         {
            dev_con = con->ConnectedDevice;
            printf("Chip Input %d", dev_con->deviceID);
         }
         else if (con->DeviceTypeID == OUTPUT_CLASS)
         {
            dev_con = con->ConnectedDevice;
            printf("Chip Output %d", dev_con->deviceID);
         }
         printf(";   ");
         con = con->nextConnection;
      }
      printf("\n\r");
   }
   printf("\n\r");
#endif

   //display the inputs in the system
#ifdef NUM_INPUTS
   printf("CHIP INPUTS in the system are: \r\n");
   printf("CHIP INPUTS              CONNECTIONS \r\n");
   for(i=0; i<NUM_INPUTS; i++)
   {
      printf("Chip Input %d             ", i);
      dev = input_array +i;
      num_con = dev->num_connections;
      con = dev->connectionList;
      for (j=0; j<num_con; j++)
      {
         if(con->ConnectionDirection == CONNECTION_INPUT)
            printf("INPUT from ");
         else if(con->ConnectionDirection == CONNECTION_OUTPUT)
```

```
            printf("OUTPUT to ");
         else
            printf("I/O: ");
         if(con->DeviceTypeID == MAGNETIC_VALVE)
         {
            v2 = con->ConnectedDevice;
            dev_con = v2->thisDevice;
            printf("Magnetic Valve %d", dev_con->deviceID);
         }
         else if (con->DeviceTypeID == MAGNETIC_MIXER)
         {
            m2 = con->ConnectedDevice;
            dev_con = m2->thisDevice;
            printf("Magnetic Mixer %d", dev_con->deviceID);
         }
         else if (con->DeviceTypeID == INPUT_CLASS)
         {
            dev_con = con->ConnectedDevice;
            printf("Chip Input %d", dev_con->deviceID);
         }
         else if (con->DeviceTypeID == OUTPUT_CLASS)
         {
            dev_con = con->ConnectedDevice;
            printf("Chip Output %d", dev_con->deviceID);
         }
         printf(";   ");
         con = con->nextConnection;
      }
      printf("\n\r");
   }
   printf("\n\r");

#endif

   //display the outputs in the system
#ifdef NUM_OUTPUTS
   printf("CHIP OUTPUTS in the system are: \r\n");
   printf("CHIP OUTPUT              CONNECTIONS \r\n");
   for(i=0; i<NUM_OUTPUTS; i++)
   {
      printf("Chip Output %d            ", i);
      dev = output_array +i;
      num_con = dev->num_connections;
      con = dev->connectionList;
      for (j=0; j<num_con; j++)
      {
         if(con->ConnectionDirection == CONNECTION_INPUT)
            printf("INPUT from ");
         else if(con->ConnectionDirection == CONNECTION_OUTPUT)
            printf("OUTPUT to ");
         else
            printf("I/O: ");
         if(con->DeviceTypeID == MAGNETIC_VALVE)
         {
            v2 = con->ConnectedDevice;
            dev_con = v2->thisDevice;
            printf("Magnetic Valve %d", dev_con->deviceID);
         }
```

```
         else if (con->DeviceTypeID == MAGNETIC_MIXER)
         {
            m2 = con->ConnectedDevice;
            dev_con = m2->thisDevice;
            printf("Magnetic Mixer %d", dev_con->deviceID);
         }
         else if (con->DeviceTypeID == INPUT_CLASS)
         {
            dev_con = con->ConnectedDevice;
            printf("Chip Input %d", dev_con->deviceID);
         }
         else if (con->DeviceTypeID == OUTPUT_CLASS)
         {
            dev_con = con->ConnectedDevice;
            printf("Chip Output %d", dev_con->deviceID);
         }
         printf(";   ");
         con = con->nextConnection;
      }
      printf("\n\r");
   }
   printf("\n\r");
#endif

   //display the pumps in the system
#ifdef NUM_PUMPS

#endif

   //display the sensors in the system
#ifdef NUM_SNESORS

#endif

   //display heaters and coolers in the system
#ifdef NUM_HEATERS_COOLERS

#endif
   //input prompt
   printf("\r\n");
   printf("Please enter the name of the component class to test: ");
   scanf("%s", (char*) &input);

   //if using a system that has the UART bug, need to get rid of the NULL
       characters
#ifdef UART_BUG
   for(i=1;i<INPUT_LENGTH;i++)
   {
      input[i] = input[2*i];
   }
#endif

   if(strcmp("Q", &input)==0)
   {
      change_to_system();
   }
   if(strcmp("VALVE", &input)==0)
   {
```

```
      test_valve_in_system ();
   }
   if(strcmp("MIXER", &input)==0)
   {
      test_mixers_in_system ();
   }


   return;
}

//function for performing the system level qPCR test
//currently just a dumy
void qPCR_test( void )
{
   printf("Ran dummy qPCR test \r\n");
   return;
}


//changes to SYSTEM mode
void change_to_system( void )
{
   mode = SYSTEM_MODE;
   printf("Entering system mode... \r\n");
   return;
}

//changes to COMMAND mode
void change_to_command( void )
{
   mode = COMMAND_MODE;
   printf("Entering command mode... \r\n");
   return;
}

//changes to TEST mode
void change_to_test( void )
{
   mode = TEST_MODE;
   printf("Entering test mode... \r\n");
   return;
}

//adds a new test to the linked list of system test
void add_system_test(char* SysTestName, funct_p swFunct, char* Description)
{
   struct SystemTest *newTest, *p;

   newTest = malloc( sizeof(SystemTestList) );
   newTest->SysTestName = SysTestName;
   newTest->swFunct = swFunct;
   newTest->Description = Description;
   newTest->next = NULL;

   //find the end of the linked list
   p = &SystemTestList;
   while(p->next != NULL)
```

```
   {
      p = p->next;
   }

   p->next = newTest;
   newTest->prev = p;

   return;
}

//adds new command to the linked list of COMMAND mode commands
void add_command(char* CommandName, funct_p2 swFunct)
{
   struct Command *newCommand, *p;

   newCommand = malloc( sizeof(CommandList) );
   newCommand->CommandName = CommandName;
   newCommand->swFunct = swFunct;
   newCommand->next = NULL;

   //find the end of the linked list
   p = &CommandList;
   while(p->next != NULL)
   {
      p = p->next;
   }

   p->next = newCommand;
   newCommand->prev = p;

   return;
}

//creates a new connection structure and adds it to the linkes list
struct Connection* newConnection(int DeviceTypeID, void* ConnectedDevice, struct
     Connection* nextConnection)
{
   struct Connection* newConnection;

   newConnection = malloc( sizeof(DummyConnection) );
   newConnection->DeviceTypeID = DeviceTypeID;
   newConnection->ConnectedDevice = ConnectedDevice;
   newConnection->nextConnection = nextConnection;

   return newConnection;
}

//creates a new device structure and fills it with specified parameters
struct Device* newDevice(int classID, int deviceID, void* device, float width,
    float length, float height, float x_loc, float y_loc, int num_connections,
    struct Connection* connectionList)
{
   struct Device* newDevice;

   newDevice = malloc( sizeof(DummyDevice) );
   newDevice->classID = classID;
   newDevice->deviceID = deviceID;
   newDevice->device = device;
```

```
   newDevice->width = width;
   newDevice->length = length;
   newDevice->height = height;
   newDevice->x_loc = x_loc;
   newDevice->y_loc = y_loc;
   newDevice->num_connections = num_connections;
   newDevice->connectionList = connectionList;


   return newDevice;
}
```

A.2 Test Chip Specifications

```
/*-----------------------------------------------------------------------------
 * populate_chip.h
 *
 * Defines the contents of a test chip
 * Specifies the arrays of devices presetn in the chip
 * Defines functions for initializing the chip
 *
 * Author: Veronica Cojocaru
 *
 *
 * Revision History:
 *    April 12, 2015 - Initial version
 *      August 5, 2015 - Latest revision
 -----------------------------------------------------------------------------*/

#ifndef POPULATE_CHIP_H
#define POPULATE_CHIP_H


/*------------------------------- INCLUDES ---------------------------------*/
#include "xbasic_types.h"   /* Xilinx basic types for device drivers        */
#include "xgpio.h"         /* layer 0/1 GPIO device driver                 */
#include "xparameters.h"    /* generated system parameters                 */
#include "interface.h"
#include "mixers.h"
#include "valves.h"

/*------------------------------- DEFINES ----------------------------------*/
//defines for all the devices in the test chip
#define NUM_VALVES   4
#define NUM_MIXERS   2
#define NUM_INPUTS   1
#define NUM_OUTPUTS  1
//#define NUM_PUMPS  0
//#define NUM_HEATER_COOLER   0
//#define NUM_SENSORS   0

#define OUTPUTS_MASK 0x3FF    //mask for all the GPIOs connected to devices in
    the chip


/*------------------- EXTERNAL GLOBAL VARIABLE DEFINITIONS -----------------*/
```

```
//array of valves in chip
extern struct ValveClass valve_array[NUM_VALVES];
extern struct ValveClass_MagneticValve magnetic_valve_array[NUM_VALVES];
extern struct Device device_valve_array[NUM_VALVES];

//array of mixers in chip
extern struct MixerClass mixer_array[NUM_MIXERS];
extern struct MixerClass_MagneticCilia magnetic_mixer_array[NUM_MIXERS];
extern struct Device device_mixer_array[NUM_MIXERS];

//araay of inputs in chip
extern struct Device input_array[NUM_INPUTS];

//array of outputs in chip
extern struct Device output_array[NUM_OUTPUTS];

//array for how the connectivity structure of the chip
extern struct Connection TestChip[14];

/*------------------------------ FUNCTIONS ---------------------------------*/
//adds the needed commands for the user interface modes to the list of commands
void populate_tests_commands( void );

//initializations required for the deivices to function (such as GPIO)
void initialize_chip( void );

//runs through the self test for all the valves in the system
void test_valve_in_system ( void );

//runs through the self test for all the valves in the system
void test_mixers_in_system ( void );


#endif //(POPULATE_CHIP_H)



/*-----------------------------------------------------------------------------
 * populate_chip.c
 *
 * Specifies the contents of a test chip in terms of that devices
 * are presnet and how they are conected
 * Contains the functions for initializing the chip and interface
 * with appropriate device intialization and user interface
 * functions and commands
 *
 * Author: Veronica Cojocaru
 *
 *
 * Revision History:
 *    April 12, 2015 - Initial version
 *      August 5, 2015 - Latest revision
 -----------------------------------------------------------------------------*/

/*------------------------------ INCLUDES ----------------------------------*/
#include "populate_chip.h"
```

```
/*--------------------------- GLOBAL VARIABLES -----------------------------*/
//array that specifies all the connections between devices in the chip
struct Connection TestChip[14] =
{
      {MAGNETIC_VALVE, CONNECTION_OUTPUT, (valve_array), (TestChip + 2)},
      {INPUT_CLASS, CONNECTION_INPUT, (input_array), (TestChip + 4)},
      {MAGNETIC_VALVE, CONNECTION_OUTPUT, (valve_array + 1), NULL},
      {INPUT_CLASS, CONNECTION_INPUT, (input_array), (TestChip + 6)},
      {MAGNETIC_MIXER, CONNECTION_OUTPUT, (mixer_array), NULL},
      {MAGNETIC_VALVE, CONNECTION_INPUT, (valve_array), NULL},
      {MAGNETIC_VALVE, CONNECTION_OUTPUT, (valve_array + 2), (TestChip + 8)},
      {MAGNETIC_VALVE, CONNECTION_INPUT, (valve_array + 1), (TestChip + 10)},
      {MAGNETIC_VALVE, CONNECTION_OUTPUT, (valve_array + 3), NULL},
      {MAGNETIC_VALVE, CONNECTION_INPUT, (valve_array + 1), (TestChip + 12)},
      {MAGNETIC_MIXER, CONNECTION_OUTPUT, (mixer_array + 1), NULL},
      {MAGNETIC_VALVE, CONNECTION_INPUT, (valve_array + 3), NULL},
      {OUTPUT_CLASS, CONNECTION_OUTPUT, (output_array), NULL},
      {MAGNETIC_VALVE, CONNECTION_INPUT, (valve_array + 2), NULL},
};

//array of valves in chip
struct ValveClass valve_array[NUM_VALVES] =
{
      {(device_valve_array), MAGNETIC_VALVE, (magnetic_valve_array), 0, 0x1, 0x2
          , 0},
      {(device_valve_array + 1), MAGNETIC_VALVE, (magnetic_valve_array + 1), 0,
          0x4, 0x8, 0},
      {(device_valve_array + 2), MAGNETIC_VALVE, (magnetic_valve_array + 2), 0,
          0x10, 0x20, 0},
      {(device_valve_array + 3), MAGNETIC_VALVE, (magnetic_valve_array + 3), 0,
          0x40, 0x80, 0},
};

//array of magnetic microvalves in the chip
struct ValveClass_MagneticValve magnetic_valve_array[NUM_VALVES] =
{
      {(valve_array), 0, 0},
      {(valve_array + 1), 0, 0},
      {(valve_array + 2), 0, 0},
      {(valve_array + 3), 0, 0},
};

//device array for the valves int the chip
struct Device device_valve_array[NUM_VALVES] =
{
      {VALVE_CLASS, 0, (valve_array), 6, 6, 2, 0, 0, 2, (TestChip + 1)},
      {VALVE_CLASS, 1, (valve_array + 1), 6, 6, 2, 0, 0, 3, (TestChip + 3)},
      {VALVE_CLASS, 2, (valve_array + 2), 6, 6, 2, 0, 0, 2, (TestChip + 9)},
      {VALVE_CLASS, 3, (valve_array + 3), 6, 6, 2, 0, 0, 2, (TestChip + 7)},
};


//array of mixers in chip
struct MixerClass mixer_array[NUM_MIXERS] =
{
      {(device_mixer_array), MAGNETIC_MIXER, (magnetic_mixer_array), 0, 0x100,
          0},
```

```
      {(device_mixer_array + 1), MAGNETIC_MIXER, (magnetic_mixer_array + 1), 0,
          0x200, 0},
};

//array for the magnetic cilia mixers in the chip
struct MixerClass_MagneticCilia magnetic_mixer_array[NUM_MIXERS] =
{
      {(mixer_array), 6, 2, 0, 0},
      {(mixer_array + 1), 6, 2, 0, 0},
};

//deice array for the mixers in the chip
struct Device device_mixer_array[NUM_MIXERS] =
{
      {MIXER_CLASS, 0, (mixer_array), 5, 5, 2, 0, 0, 1, (TestChip + 5)},
      {MIXER_CLASS, 1, (mixer_array + 1), 5, 5, 2, 0, 0, 1, (TestChip + 11)},
};

//araay of inputs in chip
struct Device input_array[NUM_INPUTS] =
{
      {INPUT_CLASS, 0, NULL, 2, 2, 2, 0, 0, 2, (TestChip)},
};

//array of outputs in chip
struct Device output_array[NUM_OUTPUTS] =
{
      {OUTPUT_CLASS, 0, NULL, 2, 2, 2, 0, 0, 1, (TestChip + 13)},
};



/*------------------------------ FUNCTIONS ---------------------------------*/
//adds to the list of available commands for the user interface
void populate_tests_commands( void )
{
   printf("Initializing tests and commands ...");

   //adds commands for SYSTEM mode
   add_system_test("TEST_MODE", change_to_test, "Go into TEST mode");
   add_system_test("qPCR", qPCR_test, "Dummy qPCR test");

   //adds commands for TEST mode
   add_command("TEST_MODE", change_to_test);
   add_command("MIX", MixerClass_MagneticCilia_MIX);
   add_command("OPEN_VALVE", ValveClass_MagneticValves_OPEN);
   add_command("CLOSE_VALVE", ValveClass_MagneticValves_CLOSE);

   printf("DONE!\r\n");
   return;
}

//performs initializations required to use the devices
void initialize_chip( void )
{
   XStatus Status;
   XGpio *GpioLoc;
```

```
   printf("Initializing the devices in the system...");

   //initialize GPIOs
   GpioLoc = Gpio;
   Status = XGpio_Initialize(GpioLoc,
                             XPAR_XPS_GPIO_0_DEVICE_ID);
    if (Status != XST_SUCCESS)
    {
        printf("GPIO initialization error\n\r");
    }

   XGpio_SetDataDirection(GpioLoc, 1, ~OUTPUTS_MASK);

   XGpio_DiscreteWrite(GpioLoc, 1, 0x1);

   printf("DONE!\r\n");
   return;
}

//runs through self test for al the valves in the system
void test_valve_in_system ( void )
{
   int i;
   printf("Testing all the valves in the system\r\n");

   for(i=0; i<NUM_VALVES; i++)
   {
      ValveClass_SELF_TEST(i);
   }

   printf("DONE!\r\n");
   return;
}

//runs through self test for all the mixers in the system
void test_mixers_in_system ( void )
{
   int i;
   printf("Testing all the mixers in the system\r\n");

   for(i=0; i<NUM_MIXERS; i++)
   {
      MixerClass_SELF_TEST(i);
   }

   printf("DONE!\r\n");
   return;
}
```

A.3 Valve Library

```
/*-----------------------------------------------------------------------------
 * valves.h
 *
 * Defines the driver structures and functions needed to interface
 * with a valve type device
```

```
 * Currently implemenets interfacing for
 *    - generic valve
 *    - magnetic valve
 *
 * Author: Veronica Cojocaru
 *
 *
 * Revision History:
 *    April 12, 2015 - Initial version
 *      August 5, 2015 - Latest revision
 ------------------------------------------------------------------------*/

#ifndef VALVES_H
#define VALVES_H


/*------------------------------ INCLUDES ----------------------------------*/
#include "interface.h"


/*------------------------ VALVE CLASS STRUCTURE --------------------------*/
struct ValveClass{
struct Device *thisDevice; // pointer to device class structure for this valve

int typeOfVave;            //specifies what kind of valve this is
void* specificValve;    //pointer to specific valve structure

int GPIO_num;           //the GPIO number that drives this device
int GPIO_mask_OPEN;        //the GPIO mask for opening this valve
int GPIO_mask_CLOSE;    //the GPIO mask for closing this valve

int IS_OPEN;            // keeps track if the particular valve is open or closed
};


/*------------------------ FUNCTIONS FOR VALVE CLASS ----------------------*/

//newValve function creates a new instance of the class structure with the
    specified parameters
struct ValveClass* ValveClass_newValve(struct Device *thisDevice, int
    typeOfValve, void* specificValve, int GPIO_num, int GPIO_mask_OPEN, int
    GPIO_mask_CLOSE, int IS_OPEN);
void ValveClass_OPEN(struct ValveClass *valve_inst);  // opens the valve
void ValveClass_CLOSE(struct ValveClass *valve_inst); // closes the valve

int ValveClass_SELF_TEST(int valve_num);  //tests the operation of the device


/*---------------------- MAGNETIC VALVE CLASS STRUCTURE -------------------*/
struct ValveClass_MagneticValve{
struct ValveClass *valve;  // pointer to valve class structure for this device

float open_pressure;    //specifies the min pressure the valve will allow when
    open
float closed_pressure;     //specifies the max pressure the valve will withstand
     when closed
};
```

```
/*-------------------- FUNCTIONS FOR MAGNETIC VALVE CLASS ------------------*/
void ValveClass_MagneticValves_OPEN(int parameters[NUMBER_PARAMETERS]); // opens
     the valve
void ValveClass_MagneticValves_CLOSE(int parameters[NUMBER_PARAMETERS]); //
    closes the valve

void ValveClass_MagneticValves_SELF_TEST(struct ValveClass_MagneticValve *valve)
    ;   //tests the operation of the device


/*------------------- EXTERNAL GLOBAL VARIABLE DEFINITIONS -----------------*/
extern struct ValveClass DummyValve;


#endif //(VALVES_H)
```

```
/*-----------------------------------------------------------------------------
 * valves.c
 *
 * Contains the functions needed to interface with a valve type device.
 * Currently implemenets interfacing for
 *    - generic valve
 *    - magnetic valve
 *
 * Author: Veronica Cojocaru
 *
 *
 * Revision History:
 *    April 12, 2015 - Initial version
 *      August 5, 2015 - Latest revision
 ----------------------------------------------------------------------------*/

/*------------------------------- INCLUDES ---------------------------------*/
#include "valves.h"
#include "xbasic_types.h"   /* Xilinx basic types for device drivers        */
#include "xgpio.h"         /* layer 0/1 GPIO device driver                 */
#include "populate_chip.h" /* Contains information about the dumy chip     */
#include <stdlib.h>


/*--------------------------- GLOBAL VARIABLES -----------------------------*/
struct ValveClass DummyValve = {NULL, 0, 0, 0};


/*------------------------------ FUNCTIONS ---------------------------------*/

//opens the valve by sending an open pulse to the specified GPIO
void ValveClass_OPEN(struct ValveClass *valve_inst)
{
   int Delay, Delay2;
   int num;
   XGpio *GpioLocal;

   num = valve_inst->GPIO_num;
```

```
   GpioLocal = Gpio;
   GpioLocal += num;

   printf("Opening valve %d \r\n", (valve_inst->thisDevice)->deviceID);

   //turns on the GPIO corresponding to the open command for this valve
   XGpio_DiscreteWrite(GpioLocal, 1, valve_inst->GPIO_mask_OPEN);

   //holds it on for the duration of the pulse
   for (Delay2 = 0; Delay2 < 10; Delay2++)
   {
   for (Delay = 0; Delay < DELAY_COUNT; Delay++)
      printf(".");
   }

   //turn off the GPIO to end the pulse
   XGpio_DiscreteWrite(GpioLocal, 1, ~valve_inst->GPIO_mask_OPEN);
   printf("DONE! \r\n");

   //indicates the valve is now open
   valve_inst->IS_OPEN = TRUE;

    return;
}

//closes the valve by sending an open pulse to the specified GPIO
void ValveClass_CLOSE(struct ValveClass *valve_inst)
{
   int Delay, Delay2;
   int num;
   XGpio *GpioLocal;

   num = valve_inst->GPIO_num;
   GpioLocal = Gpio;
   GpioLocal += num;

   printf("Closing valve %d \r\n", (valve_inst->thisDevice)->deviceID);

   //turns on the GPIO corresponding to the close command for this valve
   XGpio_DiscreteWrite(GpioLocal, 1, valve_inst->GPIO_mask_CLOSE);

   //holds it on for the duration of the pulse
   for (Delay2 = 0; Delay2 < 10; Delay2++)
   {
   for (Delay = 0; Delay < DELAY_COUNT; Delay++)
      printf(".");
   }

   //turn off the GPIO to end the pulse
   XGpio_DiscreteWrite(GpioLocal, 1, ~valve_inst->GPIO_mask_CLOSE);
   printf("DONE! \r\n");

   //indicates the valve is now open
   valve_inst->IS_OPEN = FALSE;

   return;
}
```

```
//creates and initializes a new instance of the valve class
struct ValveClass* ValveClass_newValve(struct Device *thisDevice, int
    typeOfValve, void* specificValve, int GPIO_num, int GPIO_mask_OPEN, int
    GPIO_mask_CLOSE, int IS_OPEN)
{
   struct ValveClass* newValve;

   newValve = malloc( sizeof(DummyValve) );
   if(thisDevice >=0)
      newValve->thisDevice = thisDevice;
   else
      newValve->thisDevice = 0;
   newValve->typeOfVave = typeOfValve;
   newValve->specificValve = specificValve;
   if(GPIO_num >=0)
      newValve->GPIO_num = GPIO_num;
   else
      newValve->GPIO_num = 0;
   if(GPIO_mask_OPEN>=0)
      newValve->GPIO_mask_OPEN = GPIO_mask_OPEN;
   else
      newValve->GPIO_mask_OPEN = 0;
   if(GPIO_mask_CLOSE>=0)
      newValve->GPIO_mask_CLOSE = GPIO_mask_CLOSE;
   else
      newValve->GPIO_mask_CLOSE = 0;
   if((IS_OPEN==TRUE)||(IS_OPEN==FALSE))
      newValve->IS_OPEN = IS_OPEN;
   else
      newValve->IS_OPEN = FALSE;

   return newValve;
}

//tests the functionality of a generic valve by calling the appropriate test
    function
int ValveClass_SELF_TEST(int valve_num)
{
   struct ValveClass *valve;

   valve = valve_array + valve_num;

   //check what kind of valve and call approprite test function
   if(valve->typeOfVave == MAGNETIC_VALVE)
   {
      ValveClass_MagneticValves_SELF_TEST(valve->specificValve);
   }
   else
   {
      printf("ERROR: Device not recognised as a valid valve type.  Exiting self
          test! \n\r");
      return -1;
   }

   return 0;
}

//function to open a magnetic valve
```

```
//uses the paramater array implementation since it is meant to be called from
    the main loop
void ValveClass_MagneticValves_OPEN(int parameters[NUMBER_PARAMETERS])
{
   int device_id;
   struct ValveClass *this_valve;

   //first param is device ID
   device_id = parameters[0];

   if((device_id<0)||(device_id>=NUM_VALVES))
   {
      printf("ERROR: Valve id doesn't correspond to an existing valve. \r\n No
          action can be performed, exiting function!\r\n");
      return;
   }

   this_valve = valve_array + device_id;

   ValveClass_OPEN(this_valve);

   return;
}

//function to close a magnetic valve
//uses the paramater array implementation since it is meant to be called from
    the main loop
void ValveClass_MagneticValves_CLOSE(int parameters[NUMBER_PARAMETERS])
{
   int device_id;
   struct ValveClass *this_valve;

   //first param is device ID
   device_id = parameters[0];

   if((device_id<0)||(device_id>=NUM_VALVES))
   {
      printf("ERROR: Valve id doesn't correspond to an existing valve. \r\n No
          action can be performed, exiting function!\r\n");
      return;
   }

   this_valve = valve_array + device_id;

   ValveClass_CLOSE(this_valve);

   return;
}

//test function for the magnetic valve
//opens valve and keeps it open for 5 seconds, the closes valve
void ValveClass_MagneticValves_SELF_TEST(struct ValveClass_MagneticValve *valve)
{
   int Delay, Delay2;
   struct ValveClass *valve2;

   valve2 = valve->valve;
```

```
   printf("Putting valve in open state for 5 seconds. \n\r");
   ValveClass_OPEN(valve2);

   //5 second delay
   for (Delay2 = 0; Delay2< 50; Delay2++)
         {
            for (Delay = 0; Delay < DELAY_COUNT; Delay++)
            {
               printf(".");
            }
         }
   printf("\n\r");
   printf("Putting valve in closed state. \n\r");
   ValveClass_CLOSE(valve2);

   return;
}
```

A.4 Mixer Library

```
/*-------------------------------------------------------------------------
 * mixers.h
 *
 * Defines the driver structures and functions needed to interface
 * with a mixer type device
 * Currently implemenets interfacing for
 *    - generic mixer
 *    - magnetic cilia mixer
 *
 * Author: Veronica Cojocaru
 *
 *
 * Revision History:
 *    April 12, 2015 - Initial version
 *      August 5, 2015 - Latest revision
 -------------------------------------------------------------------------*/

#ifndef MIXERS_H
#define MIXERS_H


/*------------------------------ INCLUDES ---------------------------------*/
#include "interface.h"


/*------------------------ MIXER CLASS STRUCTURE --------------------------*/

struct MixerClass{
struct Device *thisDevice; // pointer to device class structure for this mixer

int typeOfMixer;        //specifies what kind of mixer this is
void* specificMixer;    //pointer to specific device structure

int GPIO_num;           //the GPIO number that drives this device
int GPIO_mask;          //the GPIO mask for connecting to this device
```

```
int MIXING; // keeps track if the particular mixer is on or off
};


/*------------------------ FUNCTIONS FOR MIXER CLASS -----------------------*/

void MixerClass_ON(struct MixerClass *mixer_inst);    // starts the mixing
    process

void MixerClass_OFF(struct MixerClass *mixer_inst); // stops the mixing process

//newMixer function creates a new instance of the class structure with the
    specified parameters
struct MixerClass* MixerClass_newMixer(struct Device *thisDevice, int
    typeOfMixer, void* specificMixer, int GPIO_num, int GPIO_mask, int MIXING);

int MixerClass_SELF_TEST(int device_num);       //tests the operation of the
    device


/*---------------------- MAGNETIC MIXER CLASS STRUCTURE --------------------*/

struct MixerClass_MagneticCilia{
struct MixerClass *mixer;  // pointer to mixer class structure for this device
int num_cilia;             // additional info about the mixer
float cilia_height;        // specifies height of cilia used in this mixer
int mixing_strength;       // keeps track of the desired mixing strength
int mixing_speed;          // keeps track of the desired mixing speed
};


/*-------------------- FUNCTIONS FOR MAGNETIC MIXER CLASS ------------------*/

//setParams fills in the appropriate parameter values in the structure
void MixerClass_MagneticCilia_setParams(struct MixerClass_MagneticCilia *
    cilia_inst, int mixing_speed, int mixing_strength);

//MIX function operates the mixer according to the following inputs:
//parameters[0] = deviceID of the mixer to use
//parameters[1] = mixing time in terms of how many times to oscillate the cilia
//parameters[2] = mixing speed in terms of the time between cilia oscillations
//parameters[3] = mixing strength of the cilia (currently not implememnted)
void MixerClass_MagneticCilia_MIX(int parameters[NUMBER_PARAMETERS]);

void MixerClass_MagneticMixer_SELF_TEST(struct MixerClass_MagneticCilia *mixer);
     //tests the operation of the device

/*------------------- EXTERNAL GLOBAL VARIABLE DEFINITIONS -----------------*/
extern struct MixerClass DummyMixer;

#endif //(MIXERS_H)



/*-----------------------------------------------------------------------------
 * mixers.c
 *
 * Contains the functions needed to interface with a mixer type device.
```

```
 * Currently implemenets interfacing for
 *    - generic mixer
 *    - magnetic cilia mixer
 *
 * Author: Veronica Cojocaru
 *
 *
 * Revision History:
 *    April 12, 2015 - Initial version
 *      August 5, 2015 - Latest revision
 ----------------------------------------------------------------------------*/


/*------------------------------ INCLUDES ----------------------------------*/
#include "mixers.h"
#include "populate_chip.h" /* Contains information about the dumy chip     */
#include "xbasic_types.h"   /* Xilinx basic types for device drivers        */
#include "xgpio.h"         /* layer 0/1 GPIO device driver                 */
#include <stdlib.h>


/*--------------------------- GLOBAL VARIABLES -----------------------------*/
struct MixerClass DummyMixer = {NULL, 0, 0, 0};


/*------------------------------ FUNCTIONS ---------------------------------*/

//starts the mixer by writing to the corresponding GPIO
void MixerClass_ON(struct MixerClass *mixer_inst)
{
   int num;
   XGpio *GpioLocal;

   //get GPIO for this specific mixer
   num = mixer_inst->GPIO_num;
   GpioLocal = Gpio;
   GpioLocal += num;

   //turn on mixer
   XGpio_DiscreteWrite(GpioLocal, 1, mixer_inst->GPIO_mask);
   mixer_inst->MIXING = TRUE;

    return;
}

//stops the mixer by writing to the corresponding GPIO
void MixerClass_OFF(struct MixerClass *mixer_inst)
{
   int num;
   XGpio *GpioLocal;

   //get GPIO for this specific mixer
   num = mixer_inst->GPIO_num;
   GpioLocal = Gpio;
   GpioLocal += num;

   //turn off mixer
   XGpio_DiscreteWrite(GpioLocal, 1, ~(mixer_inst->GPIO_mask));
```

```
   mixer_inst->MIXING = FALSE;

   return;
}

//creates and initializes a new instance of the mixer class
struct MixerClass* MixerClass_newMixer(struct Device *thisDevice, int
    typeOfMixer, void* specificMixer, int GPIO_num, int GPIO_mask, int MIXING)
{
   struct MixerClass* newMixer;

   newMixer = malloc( sizeof(DummyMixer) );
   if(thisDevice>=0)
      newMixer->thisDevice = thisDevice;
   else
      newMixer->thisDevice = 0;
   newMixer->typeOfMixer = typeOfMixer;
   newMixer->specificMixer = specificMixer;
   if(GPIO_num>=0)
      newMixer->GPIO_num = GPIO_num;
   else
      newMixer->GPIO_num = 0;
   if(GPIO_mask >=0)
      newMixer->GPIO_mask = GPIO_mask;
   else
      newMixer->GPIO_mask = 0;
   if((MIXING == TRUE)||(MIXING == FALSE))
      newMixer->MIXING = MIXING;
   else
      newMixer->MIXING = FALSE;

   return newMixer;
}

//tests the functionality of a generic mixer by calling the appropriate test
    function
int MixerClass_SELF_TEST(int device_num)
{
   struct MixerClass *mixer;

   mixer = mixer_array + device_num;
   printf("Self-test for valve %d \n\r", device_num);

   //call test function for corresponding mixer type
   if(mixer->typeOfMixer==MAGNETIC_MIXER)
   {
      MixerClass_MagneticMixer_SELF_TEST(mixer->specificMixer);
   }
   else
   {
      printf("ERROR: Device not recognised as a valid mixer type.  Exiting self
          test! \n\r");
      return -1;
   }
   return 0;
}
```

```
void MixerClass_MagneticCilia_setParams(struct MixerClass_MagneticCilia *
    cilia_inst, int mixing_speed, int mixing_strength)
{
   cilia_inst->mixing_speed = mixing_speed;
   cilia_inst->mixing_strength = mixing_strength;

   return;
}

//start mixer operation for the Magnetic Cilia Mixer
//mixes at the specified speed for the specified time
//uses the paramater array implementation since it is meant to be called from
    the main loop
void MixerClass_MagneticCilia_MIX(int parameters[NUMBER_PARAMETERS])
{
   struct MixerClass *this_mixer;
   int device_id;
   int mixing_times;
   int mixing_speed;
   int mixing_strength;
   int Delay, Delay2, times;

   //first param is device ID
   device_id = parameters[0];

   if((device_id<0)||(device_id>=NUM_MIXERS))
   {
      printf("ERROR: Mixer id doesn't correspond to an existing mixer. \r\n No
          mixing can be performed, exiting function!\r\n");
      return;
   }

   this_mixer = mixer_array + device_id;

   //second param is mixing time
   mixing_times = parameters[1];
   if(mixing_times <= 0)
      mixing_times = 1;

   //third param is mixing speed
   mixing_speed = parameters[2];
   if(mixing_speed <= 0)
      mixing_speed = 1;

   //fourth param is mixing strength
   mixing_strength = parameters[3];

   MixerClass_MagneticCilia_setParams(this_mixer->thisDevice, mixing_speed ,
       mixing_strength);

   //indicated mixer has started mixing
   printf("Mixing, mixing, mixing, ... \r\n");
   this_mixer->MIXING = TRUE;

   //mixes at the specified speed for the specified amount of times
   for(times = 0; times<mixing_times; times++)
   {
      MixerClass_ON(this_mixer);
```

```
      printf("Tick ");
      for (Delay2 = 0; Delay2< mixing_speed; Delay2++)
      {
         for (Delay = 0; Delay < DELAY_COUNT; Delay++)
         {
            printf(".");
         }
      }
      printf(" \r\n");
      MixerClass_OFF(this_mixer);
      printf("Tock ");
      for (Delay2 = 0; Delay2 < mixing_speed; Delay2++)
      {
         for (Delay = 0; Delay < DELAY_COUNT; Delay++)
         {
            printf(".");
         }
      }
      printf(" \r\n");
   }

   this_mixer->MIXING = FALSE;


   return;
}

//test function for the magnetic cilia mixer
//turns in mixer to mix at 10MHz for 10 seconds
void MixerClass_MagneticMixer_SELF_TEST(struct MixerClass_MagneticCilia *mixer)
{
   int parame[4] = {0, 100, 1, 0 };

   parame[0] = mixer->mixer->thisDevice->deviceID;

   printf("Mixing at 10Hz for 10 seconds. \n\r");
   MixerClass_MagneticCilia_MIX(parame);
   printf("Done mixing.\n\r");

   return;
}
```