

DESIGN, IMPLEMENTATION AND OPTIMIZATION OF A HIGH-SPEED  
LASER-SCANNING MICROSCOPE FOR SCANNING ANGLE INTERFERENCE  
MICROSCOPY

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Marshall James Colville

May 2020

© 2020 Marshall James Colville

DESIGN, IMPLEMENTATION AND OPTIMIZATION OF A HIGH-SPEED  
LASER-SCANNING MICROSCOPE FOR SCANNING ANGLE INTERFERENCE  
MICROSCOPY

Marshall James Colville, Ph. D.

Cornell University 2020

Fluorescence imaging has become an invaluable tool for the investigation of subcellular structure and organization. However, there is a fundamental limit to the resolution that can be achieved using traditional imaging systems such as fluorescence microscopes. Scanning Angle Interference Microscopy (SAIM), a technique capable of nanometer-scale localization along the optical axis, overcomes this limit using surface generated interference effects. The spatial frequency of standing waves near a reflective surface is dependent on the polar angle of the incident light. SAIM exploits this phenomenon by carefully controlling the incident angle of the excitation light on a fluorescent sample. In order to achieve high precision localizations, the excitation light must be homogeneous within the sample plane. However, coherent light sources, such as lasers, typically form characteristic fringe and speckle artifacts on length scales relevant to subcellular imaging. To eliminate these artifacts and improve image quality we have designed and built a circle-scanning microscope specifically optimized for live-cell SAIM imaging. In our microscope the excitation laser is continuously rotated through the azimuth while maintaining a constant polar angle in each exposure. Furthermore, we employ high-speed shuttering triggered by a state-of-

the-art scientific camera to achieve precise synchronization between sample excitation and the camera's exposure window, minimizing photodamage to the sample. We demonstrate potential instrument-induced artifacts in SAIM imaging using a conventional acquisition scheme and how our purpose-built instrument addresses and overcomes each. Finally, we present a detailed protocol for the instrument construction, sample preparation and execution of high-speed live-cell SAIM experiments using the circle-scanning instrument and custom, open-source instrument control hardware and graphical interface.

## **BIOGRAPHICAL SKETCH**

Marshall Colville was born in Detroit, Michigan in 1982. At the age of 4 Marshall's family moved to Portland, Oregon. He graduated from Lincoln High School in 2000, where he played football, competed in alpine ski racing on both his school and club teams, and threw shot and discus on the track team. In 2002 Marshall enlisted in the United States Marine Corps Reserve. After completion of recruit training, combat training and military occupational specialty school he returned to his reserve unit in Portland. Two months later he was recalled to active duty and deployed in support of Operation Iraqi Freedom. In 2004 was again mobilized and ordered to Tampa, Florida where he provided administrative support for units deployed overseas. Marshall was mobilized for a third and final time in 2007 and served as administrative chief for the 8<sup>th</sup> Provisional Security Company, Combined Joint Task Force - Horn of Africa in Djibouti, Africa. While deployed he enrolled in distance learning courses at Portland Community College. Upon returning to Portland in 2009, he enrolled as a full-time student at Portland State University where he developed a passion for experimental physics and scientific instrumentation. After graduating with Bachelors' Degrees in physics and biochemistry in 2013, Marshall began his doctoral research at Cornell University in the laboratory of Prof. Matthew Paszek.

This work is dedicated to my family.

## ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Matthew Paszek, for his support throughout my time at Cornell. Working with Matt has been a privilege that I cannot adequately describe. The breadth of his knowledge in the fields of engineering, cell biology, glycobiology and imaging has been an invaluable resource. More importantly, he has always supported me taking risks on new ideas outside of his area of expertise. The faith and confidence he has placed in me is humbling and very much appreciated.

I would also like to thank Professor Warren Zipfel, who opened his lab, machine shop, electronics shop, and door to me as if I was one of his own students. Every circuit board, power supply, mechanical part or other piece of custom hardware that I have designed, soldered, machined or assembled at Cornell was because of the phenomenal resources, training, and support he has provided me over the last 6 years.

I would also like to thank Professor Barbara Baird and Dr. David Holowka, with whom I did my first research rotation. As a first-year student with very little biological research background, they brought me into their lab and introduced me to biological imaging, cell culture and many other fundamental skills.

I am grateful to my advisors from Portland State University, Prof. Drake Michell, Prof. Erik Sánchez, and Prof. Andres La Rosa, for inviting me into their labs as an undergrad and encouraging me down this path.

To the members of the Paszek Lab, thank you. I couldn't have accomplished this work without your support both in and out of the lab.

Finally, I would like to thank my family. They have watched me leave home many times on many new adventures and always stood by me, offering unconditional support. I will be forever grateful.

# TABLE OF CONTENTS

<b>BIOGRAPHICAL SKETCH.....</b>	<b>v</b>
<b>ACKNOWLEDGEMENTS.....</b>	<b>vii</b>
<b>TABLE OF CONTENTS.....</b>	<b>viii</b>
<b>LIST OF FIGURES.....</b>	<b>xi</b>
<b>CHAPTER 1.....</b>	<b>1</b>
Introduction .....	1
<b>CHAPTER 2.....</b>	<b>10</b>
Genetically encoded toolbox for glycocalyx engineering: tunable control of cell adhesion, survival, and cancer cell behaviors .....	10
2.1 Abstract.....	10
2.2 Introduction .....	12
2.3 Results and Discussion.....	14
2.4 Conclusion.....	30
2.5 Materials and Methods .....	33
2.6 Acknowledgements .....	40
2.7 Author Contributions.....	41
<b>CHAPTER 3.....</b>	<b>47</b>
High-speed device synchronization in optical microscopy with an open-source hardware control platform .....	47
3.1 Abstract.....	47
3.2 Introduction .....	49
3.3 Results .....	51
3.4 Discussion.....	71
3.5 Methods .....	76
3.6 Acknowledgements .....	82
3.7 Author Contributions.....	82
<b>CHAPTER 4.....</b>	<b>87</b>
Azimuthal beam scanning microscope design and implementation for axial	

localization with scanning angle interference microscopy.....	87
4.1 Abstract.....	87
4.2 Introduction .....	89
4.3 Materials.....	95
4.4 Methods .....	101
4.5 Notes.....	118
4.6 Author Contributions.....	124
<b>CHAPTER 5.....</b>	<b>127</b>
Conclusions .....	127
Related Work and Future Directions.....	130
<b>Appendix A.....</b>	<b>138</b>
Supporting Information for Chapter 2.....	138
<b>Appendix B.....</b>	<b>147</b>
Supporting Information for Chapter 3.....	147
Supplementary Note 1: Hardware description .....	148
Supplementary Note 2: Mechanical considerations in azimuthal scanning.....	151
Supplementary Note 3: Controller induced latency .....	155
Supplementary Note 4: Software development.....	157
Supplementary Note 5: Incidence angle calibration.....	158
Supplementary Note 6: SAIM analysis .....	159
Supplementary Note 7: Firmware architecture.....	161
<b>Appendix C.....</b>	<b>182</b>
Firmware source code for the instrument controller as used in these studies .....	182
File: firmware_0_0.h.....	183
File: firmware_0_0.c .....	189
File: SAIM_utilities.c.....	205
File: Simple_SAIM.c.....	217
File: AD5440_drvr.c.....	220
File: AD5547_dual_drvr.c.....	222
File: AD5583_dual_drvr.c.....	228
File: AD9833_dual_drvr.c.....	231

<b>Appendix D.....</b>	<b>235</b>
File: SAIMScannerV3.h.....	236
File: SSV3Manager.cpp .....	244
File: SSV3Device.cpp .....	248
<b>Appendix E.....</b>	<b>268</b>
SAIM analysis source code .....	268
File: saim_model_cpu.h .....	269
File: saim_model_cpu.cpp.....	272
<b>Appendix F .....</b>	<b>288</b>
Bootloader documentation for the instrument controller project .....	288

## LIST OF FIGURES

Figure 2.1 Vector for stable expression.....	16
Figure 2.2 Genetically encoded glycoproteins for glycocalyx editing.....	19
Figure 2.3 Glycoproteins of tunable length.....	21
Figure 2.4 Engineering the chemical and physical properties of the glycocalyx. ....	22
Figure 2.5 Expression of a bulky glycocalyx inhibits cellular adhesion.....	25
Figure 2.6 Expression of a bulky glycocalyx enhances cell survival in suspension. ...	27
Figure 2.7 Cellular division is associated with adhesion. ....	30
Fig. 3.1 Hardware control integration facilitates the cooperation of a variety of peripheral devices.....	52
Fig. 3.2 Circle scanning microscope design with hardware control.....	56
Fig. 3.3 Azimuthal beam scanning reduces incidence angle dependent laser artifacts in scanning angle interference microscopy (SAIM) imaging.....	59
Fig. 3.4 Hardware based synchronization minimizes latency-induced artifacts in SAIM. ....	65
Fig. 3.5 Implementation of circle scanning multiangle TIRF (MA-TIRF). ....	69
Fig. 4.1 Surface generated interference and localization in SAIM. ....	89
Fig. 4.2 Principles of circle-scanning excitation. ....	92
Fig. 4.3 Comparison of static and circle-scanned image quality in TIRFM and SAIM. .....	93
Fig. 4.4 Mapping the plasma membrane topography of adherent cells with SAIM. ...	94
Fig. 4.5 Sample preparation for SAIM imaging.....	101
Fig. 4.6 Circle-scanning system construction.....	105
Fig. 4.7 Arrangement of lenses within the laser scanning subsystem. ....	108
Fig. 4.8 Definition of laser orientation and alignment within the optical system. ....	109
Fig. 4.9 Scanning mirror power supplies and driver layout. ....	111

Fig. 4.10 Circle-scanning system calibration. ....	112
Fig. 4.11 Control flow diagram of a generalized experimental sequence. ....	115
Fig. 5.1 Confinement between rigid surfaces as a mechanical probe of the glycocalyx. .....	132
Fig. 5.2 Trifunctional click chemistry probes for expansion microscopy of non- traditional target molecules. ....	134
Supplementary Fig. A1.1 Immunoblot. ....	139
Supplementary Fig. A1.2 RTK Screen. ....	140
Supplementary Fig. A2.1 Expanded schematic diagrams of the controller analog subcircuits from Fig. 1. ....	167
Supplementary Fig. A2.2 Example timing waveforms from a sequence acquisition. ....	168
Supplementary Fig. A2.3 Galvanometer scanning mirror response to discretized (point scanning) and continuous waveforms in circle scanning experiments. ....	169
Supplementary Fig. A2.4 Laser speckle and fringing depend on incidence angle and lead to artifacts in SAIM reconstructions. ....	171
Supplementary Fig. A2.5 Excitation shuttering time comparison between software (left) and hardware (right) control. ....	173
Supplementary Fig. A2.6 Time lapse SAIM reconstructions of the focal adhesion protein Zyxin. ....	174
Supplementary Fig. A2.7 Circuit board layout. All the controller electronics are integrated into a single, compact PCB. ....	175
Supplementary Fig. A2.8 The complete controller assembly with relevant connectors. .....	176
Supplementary Fig. A2.9 Graphical interface for circle scanning applications. ....	177
Supplementary Fig. A2.10 Control flow block diagrams for polling and interrupt approaches to firmware design. ....	179
Supplementary Fig. A2.11 The firmware memory organization uses static storage for the experimental parameters and dynamic storage for experiment design. ....	180
Supplementary Fig. A2.12 Internal connections in the completed controller assembly. .....	181

# CHAPTER 1

## Introduction

Fluorescence imaging is a powerful tool in cell biology that can reveal sub cellular organization with molecular specificity. A wide variety of labeling techniques exist with molecular specificity including genetically encoded fluorescent proteins, antibodies, lectins and small peptide tags among others.<sup>1,2</sup> These strategies, when coupled with the biocompatibility of light microscopy can reveal molecular details in living cells. However, the resolving power of an imaging system is set by the maximum spatial frequency observable. Named the diffraction limit, this value was defined by Ernst Abbe as

$$d_{xy} = \frac{\lambda}{2NA} \quad (1)$$

$$d_z = \frac{2\lambda}{NA^2} \quad (2)$$

where  $\lambda$  is the wavelength of the radiation and  $NA$  is the numerical aperture of the system. In a biological context this gives a lateral resolution limit of  $\sim 200$  nm with currently available high  $NA$  objective lenses. In the axial direction resolution limit is significantly greater at  $\sim 500$  nm. While these values are small in comparison to the size of a typical human cell, most proteins are on the scale of 1 to 10s of nanometers. The diffraction limit is therefore a significant obstacle in the study of macromolecular organization of the densely crowded environments found in and around cells. As the fields of biology and biochemistry advance there is a need for advanced imaging techniques that can overcome the classical diffraction limit.

Recently, several super resolution and localization techniques have been developed that extract additional spatial information from the sample<sup>3,4</sup>. Single molecule localization microscopies (SMLM), such as STORM<sup>5</sup>, PALM<sup>6</sup>, and DNA-PAINT<sup>7</sup> among others, rely on the temporal separation of emission from individual fluorophores to create a sparse array in each camera exposure. Individual molecules are localized in each frame and super-resolved images are formed by combining the localizations from hundreds or thousands of images. Since initial reports of SMLM many techniques for obtaining 3-dimensional reconstruction have been developed including point spread function shaping<sup>8,9</sup>, and dual-objective<sup>10,11</sup> approaches.

Other approaches to super resolution microscopy use instrumentation to impart additional information on the sample and image processing to construct the super-resolved image. One such approach, super resolution structured illumination microscopy (SR-SIM), utilizes a diffraction grating or spatial light modulator placed in the excitation path located at a conjugate image plane to the sample. An image of the grating is superimposed on the sample, creating additional spatial frequencies in the fluorescence image. The grating is rotated through several orientation and a super-resolved image is computationally reconstructed from the set of individual images, achieving up to double the resolving power of widefield and confocal microscopy.<sup>12</sup> Since it's original conception, SR-SIM many derivations of SR-SIM have been developed including total internal reflection fluorescence (TIRF) SIM<sup>13</sup>, image scanning microscopy<sup>14</sup>, and two-photon multifocal SIM<sup>15</sup>. Structured illumination techniques generally use lower excitation intensities and require fewer images than SMLM, making it more suitable for live-cell imaging.<sup>16</sup>

Another form of excitation patterning is surface-generated interference techniques. These techniques generate a series of fringes along the excitation optical axis through the use of a mirror placed behind the sample. The excitation light undergoes a phase reversal upon reflection, giving rise to a difference in phase of

$$\varphi = \frac{4\pi nd \cos \theta_{ex}}{\lambda_{ex}} \quad (3)$$

where  $n$  is the refractive index of the media the,  $d$  is the distance from the surface,  $\theta_{ex}$  is the polar angle of incidence with respect to the surface normal, and  $\lambda_{ex}$  is the wavelength of the excitation. The counter-propagating waves interfere, creating a periodic series of maxima and minima in the axial excitation profile by superposition. This effect is the basis of fluorescence interference-contrast microscopy (FLIC) wherein the sample is prepared on a silicon wafer that has been micropatterned with a layer of thermal oxide to produce regions of varying separation between the oxide surface and the reflective layer.<sup>17</sup> In FLIC fluorescence emission from regions of the sample with different oxide thickness is fit to the optical model to determine the distance between the fluorophore and the oxide surface. Scanning angle interference microscopy (SAIM) uses the same principle, but rather than varying the thickness of the oxide spacer the excitation angle of incidence is changed across multiple images.<sup>18</sup> This has the effect of varying the spatial frequency of the interference fringes in a well-defined manner. For a fluorescent molecule in a given position, the observed emission profile as a function of angle has a unique solution to the optical model, allowing axial localization on nanometer scales.

For optimal SAIM data the excitation field should be pixel-wise consistent

across all images and free of aberrations. Additionally, the excitation light must be coherent over difference in the optical path length between the direct and reflected excitation at the sample. Lasers provide a convenient source of coherent, monochromatic light for SAIM. However, diffractive imperfections in the optics and internal reflections can cause additional, unwanted interference in the excitation field.

A common fluorescence excitation method, total Internal Reflection Fluorescence Microscopy (TIRFM), exploits the thin evanescent field formed when the excitation light strikes the glass-to-sample interface at a supercritical angle<sup>19</sup>. Laser interference is a common problem in TIRFM and a number of methods to eliminate the appearance of interference speckles and fringes have been developed. One method is to scan the excitation beam through its azimuth, commonly called circle-scanning.<sup>20-24</sup> By superimposing the sample fluorescence at different azimuthal angles in a single exposure of the camera an image of the sample is acquired with an even effective excitation field. Operating on this principle, SAIM could be improved by the integration of circle scanning.

While the following chapters detail the specifics of the instrumentation, development process and applications of the circle-scanning SAIM microscope, it is important to state the overall goal of this research. This project, from conception to execution, is dedicated to providing a set of tools that will enable research into cellular organization and morphology, opening new lines of inquiry. The collection of technology and refined methodology have been guided by the current and needs of collaborators. Many of the design decisions were made with flexibility and broad applicability in mind.

In Chapter 2 we present a genetic toolbox for precision manipulation of the cellular glycocalyx. We develop a library of edited and semisynthetic sialomucins and detail their effects on glycan content, the physical properties of the extracellular glycoprotein coating, and phenotypic alteration. We successfully replace the transmembrane and intracellular domains of two model sialomucins, mucin 1 (Muc1) and podocalyxin (Podxl) to nullify intracellular signaling functions. We then develop and construct a modular strategy for modifying the extracellular domain length of Podxl. By mutating a segment of the native Podxl extracellular domain to introduce two orthogonal restriction enzyme recognition sites we create a building block that can be directionally and repeatedly inserted between the signal peptide and membrane-proximal domains. We validate this modular construction approach by building constructs of 2/3 and 4/3 length and successfully express them in human epithelial cells. We then use SAIM to measure the glycocalyx thickening in normal epithelial cells overexpressing native and our semisynthetic mucins. Finally, we perform live-cell time course experiments to investigate expression kinetics, motility and survival of cells modified with our toolbox. For these experiments we employ a custom incubator fluorescence microscope, which enables long-term, high-volume imaging of cells in a multiwell plate format and specialized analysis software to process the large data volumes. With these tools we track and quantify the detachment of cells with a thickened glycocalyx from the substrate and find that cells overexpressing Muc1 survive over 3-fold longer in suspension than unmodified cultures. Finally, we track cells cycling through detachment and reattachment, and find that cell with a thickened glycocalyx are able to re-adhere to the substrate for division.

In Chapter 3 we introduce the custom SAIM microscope and hardware controller. The controller theory of operation is described, including the electronic design, firmware, and programming interface. We describe its integration with the peripheral devices required for high-speed circle-scanning acquisitions. We then demonstrate the benefits of the circle-scanning approach in SAIM by quantification of the reconstruction artifacts that occur with static-beam excitation. To illustrate the benefits of low level, interrupt driven hardware control we characterize timing latencies and variability introduced by USB communications in and their effect on data quality. We extend this to show how inaccurate device synchronization can lead to the appearance of artifactual topography in SAIM reconstructions. As a demonstration of the platform's versatility we perform multiangle TIRF imaging with our SAIM microscope.

In Chapter 4 we provide instructions for building and operating a circle-scanning SAIM microscope. A list of required equipment and components is given. Sample preparation, including the characterization, cleaning and chemical modification of the reflective substrates for cell culture is detailed. We give instructions for building the scanning optics and laser launch, with tips for proper alignment of the optical train, one of the crucial steps in achieving optimal circle scanning performance. System setup using the tool in the controller graphical interface is covered, providing a fast, reliable, and convenient method for accurate scan angle calibration. Finally, instructions for designing and executing SAIM experiments in the controller software, as well as typical experimental parameters, are given.

## REFERENCES

1. Snapp, E. Design and Use of Fluorescent Fusion Proteins in Cell Biology. *Curr Protoc Cell Biol* **CHAPTER**, Unit-21.4 (2005).
2. Giepmans, B. N. G., Adams, S. R., Ellisman, M. H. & Tsien, R. Y. The Fluorescent Toolbox for Assessing Protein Location and Function. *Science* **312**, 217–224 (2006).
3. Huang, B., Bates, M. & Zhuang, X. Super-Resolution Fluorescence Microscopy. *Annual Review of Biochemistry* **78**, 993–1016 (2009).
4. Sydor, A. M., Czymmek, K. J., Puchner, E. M. & Mennella, V. Super-Resolution Microscopy: From Single Molecules to Supramolecular Assemblies. *Trends in Cell Biology* **25**, 730–748 (2015).
5. van de Linde, S. *et al.* Direct stochastic optical reconstruction microscopy with standard fluorescent probes. *Nature Protocols* **6**, 991–1009 (2011).
6. Betzig, E. *et al.* Imaging Intracellular Fluorescent Proteins at Nanometer Resolution. *Science* **313**, 1642–1645 (2006).
7. Schnitzbauer, J., Strauss, M. T., Schlichthaerle, T., Schueder, F. & Jungmann, R. Super-resolution microscopy with DNA-PAINT. *Nature Protocols* **12**, 1198–1228 (2017).
8. Huang, B., Wang, W., Bates, M. & Zhuang, X. Three-dimensional Super-resolution Imaging by Stochastic Optical Reconstruction Microscopy. *Science* **319**, 810–813 (2008).
9. Jia, S., Vaughan, J. C. & Zhuang, X. Isotropic three-dimensional super-resolution imaging with a self-bending point spread function. *Nature Photonics* **8**, 302–306 (2014).
10. Xu, K., Babcock, H. P. & Zhuang, X. Dual-objective STORM reveals three-dimensional filament organization in the actin cytoskeleton. *Nature Methods* **9**, 185–188 (2012).
11. Shtengel, G. *et al.* Interferometric fluorescent super-resolution microscopy resolves 3D cellular ultrastructure. *Proceedings of the National Academy of Sciences* **106**, 3125–3130 (2009).
12. Gustafsson, M. G. L. Surpassing the lateral resolution limit by a factor of two using structured illumination microscopy. *Journal of Microscopy* **198**, 82–87 (2000).
13. Fiolka, R., Beck, M. & Stemmer, A. Structured illumination in total internal

- reflection fluorescence microscopy using a spatial light modulator. *Opt. Lett.*, *OL* **33**, 1629–1631 (2008).
14. Müller, C. B. & Enderlein, J. Image Scanning Microscopy. *Physical Review Letters* **104**, (2010).
  15. Ingaramo, M. *et al.* Two-photon excitation improves multifocal structured illumination microscopy in thick scattering tissue. *Proc. Natl. Acad. Sci. U.S.A.* **111**, 5254–5259 (2014).
  16. Heintzmann, R. & Huser, T. Super-Resolution Structured Illumination Microscopy. *Chemical Reviews* **117**, 13890–13908 (2017).
  17. Lambacher, A. & Fromherz, P. Fluorescence interference-contrast microscopy on oxidized silicon using a monomolecular dye layer. *Applied Physics A: Materials Science & Processing* **63**, 207–217 (1996).
  18. Paszek, M. J., DuFort, Christopher C, Rubashkin, Matthew G, Davidson, Michael W, Thorn, Kurt S, Liphardt, Jan T, Weaver, Valerie M., Scanning angle interference microscopy reveals cell dynamics at the nanoscale. *Nat Meth Nature Methods* **9**, 825–827 (2012).
  19. Axelrod, D. Cell-substrate contacts illuminated by total internal reflection fluorescence. *The Journal of Cell Biology* **89**, 141–145 (1981).
  20. Singh, A., Doris, E. A., Song, A. & Zipfel, W. R. An Optimized Azimuthal Scanning Platform for TIRF and HILO Imaging. *Biophysical Journal* **106**, 402a (2014).
  21. Mattheyses, A. L., Shaw, K. & Axelrod, D. Effective elimination of laser interference fringing in fluorescence microscopy by spinning azimuthal incidence angle. *Microscopy Research and Technique* **69**, 642–647 (2006).
  22. Zong, W. *et al.* Shadowless-illuminated variable-angle TIRF (siva-TIRF) microscopy for the observation of spatial-temporal dynamics in live cells. *Biomedical Optics Express* **5**, 1530 (2014).
  23. Johnson, D. S., Toledo-Crow, R., Mattheyses, A. L. & Simon, S. M. Polarization-Controlled TIRFM with Focal Drift and Spatial Field Intensity Correction. *Biophysical Journal* **106**, 1008–1019 (2014).
  24. Fiolka, R., Belyaev, Y., Ewers, H. & Stemmer, A. Even illumination in total internal reflection fluorescence microscopy using laser light. *Microscopy Research and Technique* **71**, 45–50 (2008).



## CHAPTER 2<sup>†</sup>

### Genetically encoded toolbox for glycocalyx engineering: tunable control of cell adhesion, survival, and cancer cell behaviors

#### 2.1 Abstract

The glycocalyx is a coating of protein and sugar on the surface of all living cells. Dramatic perturbations to the composition and structure of the glycocalyx are frequently observed in aggressive cancers. However, tools to experimentally mimic and model the cancer-specific glycocalyx remain limited. Here, we develop a genetically encoded toolkit to engineer the chemical and physical structure of the cellular glycocalyx. By manipulating the glycocalyx structure, we are able to switch the adhesive state of cells from strongly adherent to fully detached. Surprisingly, we find that a thick and dense glycocalyx with high *O*-glycan content promotes cell survival even in a suspended state, characteristic of circulating tumor cells during metastatic dissemination. Our data suggest that glycocalyx-mediated survival is largely independent of receptor tyrosine kinase and mitogen activated kinase signaling. While anchorage is still required for proliferation, we find that cells with a thick glycocalyx can dynamically attach to a matrix scaffold, undergo cellular division, and quickly disassociate again into a suspended state. Together, our

---

<sup>†</sup> **This chapter is reprinted with permission from:** Shurer, C. R.\*, Colville, M. J.\*, Gupta, V. K., Head, S. E., Kai, F., Lakins, J. N., & Paszek, M. J. Genetically Encoded Toolbox for Glycocalyx Engineering: Tunable Control of Cell Adhesion, Survival, and Cancer Cell Behaviors. *ACS Biomater. Sci. Eng.* (2017). doi:10.1021/acsbio.7b00037. \*These authors contributed equally.

technology provides a needed toolkit for engineering the glycocalyx in glycobiology and cancer research.

## 2.2 Introduction

The glycocalyx is a polymer meshwork comprising proteins and complex sugar chains called glycans that assemble on the surface of the eukaryotic cell.<sup>1</sup> By mediating receptor–ligand interactions,<sup>2</sup> cell-to-cell communication,<sup>3</sup> and cell-matrix adhesion,<sup>4</sup> glycans within the glycocalyx are now known to play an essential role in most biological processes, including development,<sup>5</sup> migration,<sup>6,7</sup> adhesion,<sup>8</sup> immune response,<sup>9–11</sup> and disease progression.<sup>12,13</sup> A unique feature of the glycocalyx is its dynamic nature. The composition, density, and structural organization of the glycocalyx changes profoundly with cell fate transitions, including differentiation<sup>14</sup> and transformation.<sup>15,16</sup> A key challenge that remains is to understand on a mechanistic level how changes to the glycocalyx regulate and refine complex cellular and multicellular programs.

While much attention has focused on the biochemical properties of glycans and their chemical interactions, the glycocalyx also functions as an important structural material on the cell surface.<sup>17–19</sup> Separating the cell membrane and membrane proteins from the local microenvironment, the glycocalyx is uniquely positioned to biophysically regulate cell-extracellular matrix (ECM) and cell–cell interactions.<sup>20</sup> For example, the physical properties of the glycocalyx are now known to dictate the kinetics and thermodynamics of integrin ligation with the ECM, thus controlling the spatial organization of integrin bonds and downstream signaling processes.<sup>8</sup> Moreover, all cell-surface receptors are at least partially embedded within the glycocalyx, and the structural properties of the glycocalyx impact receptor diffusion, activation, and signaling.<sup>21,22</sup> Consequently, glycocalyx remodeling in normal physiological processes

and disease states is expected to have broad biophysical consequences on cellular signaling and associated behaviors.

Aberrant glycosylation is a universal feature of cancer, but in most cases, the implications of a cancer-specific glycocalyx are poorly understood.<sup>23,24</sup> For example, circulating tumor cells (CTCs) express an abundance of large, heavily glycosylated proteins, such as the mucin Muc1, which serves as a biomarker for isolation and detection of CTCs in clinical practice.<sup>8,25,26</sup> Aberrant glycosylation is now recognized to contribute to nearly every step in cancer progression, including proliferation,<sup>27</sup> survival,<sup>7,8,28</sup> angiogenesis,<sup>29</sup> and metastatic dissemination.<sup>30,31</sup> Whether biophysical changes in the glycocalyx also contribute to some of the unique features of aggressive cancer cells and CTCs, including detachment from the ECM and survival in circulation, must still be determined.

A major roadblock to advancing cancer glycobiology has been the relative lack of tools for precision editing of the glycocalyx. One of the most successful approaches to date is based on membrane incorporation of fully synthetic polymers that mimic key features of glycoproteins.<sup>32,33</sup> Glycopolymers enable tunable control over the types and frequencies of glycoconjugates, membrane densities, and cell surface retention times.<sup>9,34</sup> The library of glycopolymers, referred to here as the synthetic toolkit, has been applied successfully to investigate cell adhesion processes,<sup>8</sup> immune cell activation,<sup>10</sup> and host–pathogen interactions.<sup>35</sup>

Relatively little work has been done in developing a genetic toolbox specifically designed for glycocalyx engineering. A genetically encoded toolbox could provide several unique capabilities that would enable new research in biophysical

glycoscience and cancer glycobiology. First, compared to the existing synthetic toolkit, genetic encoding would support prolonged surface expression of glycocalyx elements for long-term in vitro and in vivo experiments.<sup>36</sup> Second, a genetic approach would afford the use of native glycoproteins that are relevant to the particular biological system or disease model of interest. Third, DNA technology provides the flexibility to quickly generate a diverse library of glycoprotein mutants through modern synthetic biology approaches.<sup>37,38</sup>

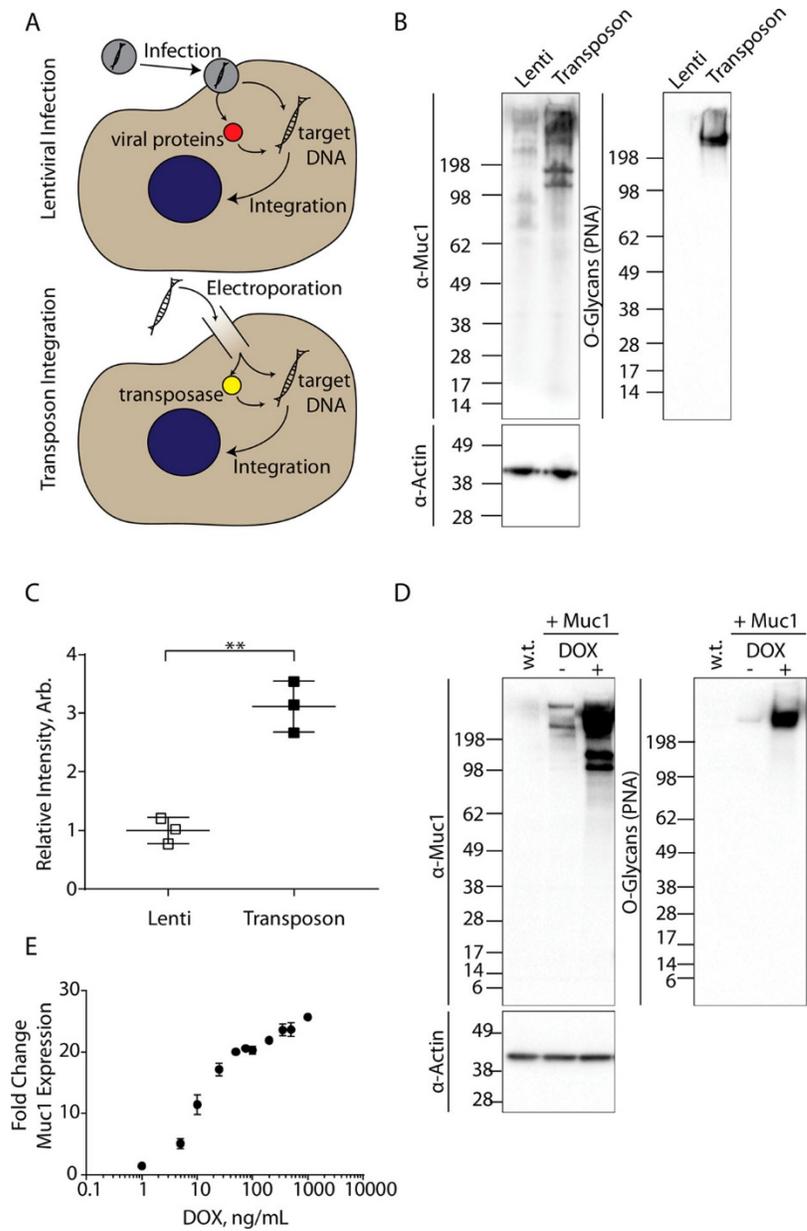
Here, we address this unmet need in glycobiology and develop a toolkit of genetically encoded glycoproteins and expression systems to engineer the structure and composition of the cellular glycocalyx. We apply our system to model the CTC glycocalyx and find that the glycocalyx itself could contribute to the unique adhesive properties and survival characteristics of CTCs.

## **2.3 Results and Discussion**

### **System for Stable Incorporation of Engineered Glycoproteins**

Our first goal was to develop and validate a strategy for stable expression of glycoproteins in mammalian cells for glycocalyx engineering. We envisioned that incorporation of our constructs and promoters in the cellular genome could (1) provide consistent and reliable levels of glycoprotein expression and glycan presentation, (2) support sorting and selection methods for high expression levels, and (3) enable temporal control over expression through the use of inducible promoters. Our choice for the promoter was the reverse tetracycline-controlled transactivator (rtTA) system, which can provide temporal control as well as tunable expression levels through

titration of doxycycline, the chemical inducer of expression.<sup>36</sup> As a test glycoprotein, we chose the mucin Muc1, a key structural element in the glycocalyx of many cancer cell types.<sup>39,40</sup> For stable integration of the inducible promoter, transgene, and selectable marker, we first tested the utility of standard lentiviral systems (Figure 2.1A). We found that Muc1 expression levels in epithelial cells were low, and the glycoprotein product was often of lower molecular weight than expected (Figure 2.1B). We suspected that the highly repetitive sequences in the Muc1 tandem repeats were recombined at some stage of viral packaging or cellular transduction, and we discontinued the further use of lentiviral systems.



**Figure 2.1 Vector for stable expression. (A)** Graphic illustration of the lentiviral and transposon stable incorporation systems. **(B)** Representative immunoblot (left) and lectin blot (right) comparison of stable Muc1 expression and PNA binding in lentiviral infection versus transposon integration,  $n = 3$ . **(C)** Mean integrated signal density from  $\alpha$ -Muc1 immunoblots in B normalized to lentiviral samples; error bars represent the SD,  $n = 3$ . **(D)** Immunoblot (left) and lectin blot (right) of Muc1 expression in w.t. MCF10A cells compared to stable expression lines uninduced and after 24 h induction with  $0.2 \mu\text{g mL}^{-1}$  doxycycline,  $n = 1$ . Cell lines were prepared with the transposon incorporation system. **(E)** Fold change in Muc1 analyzed by flow

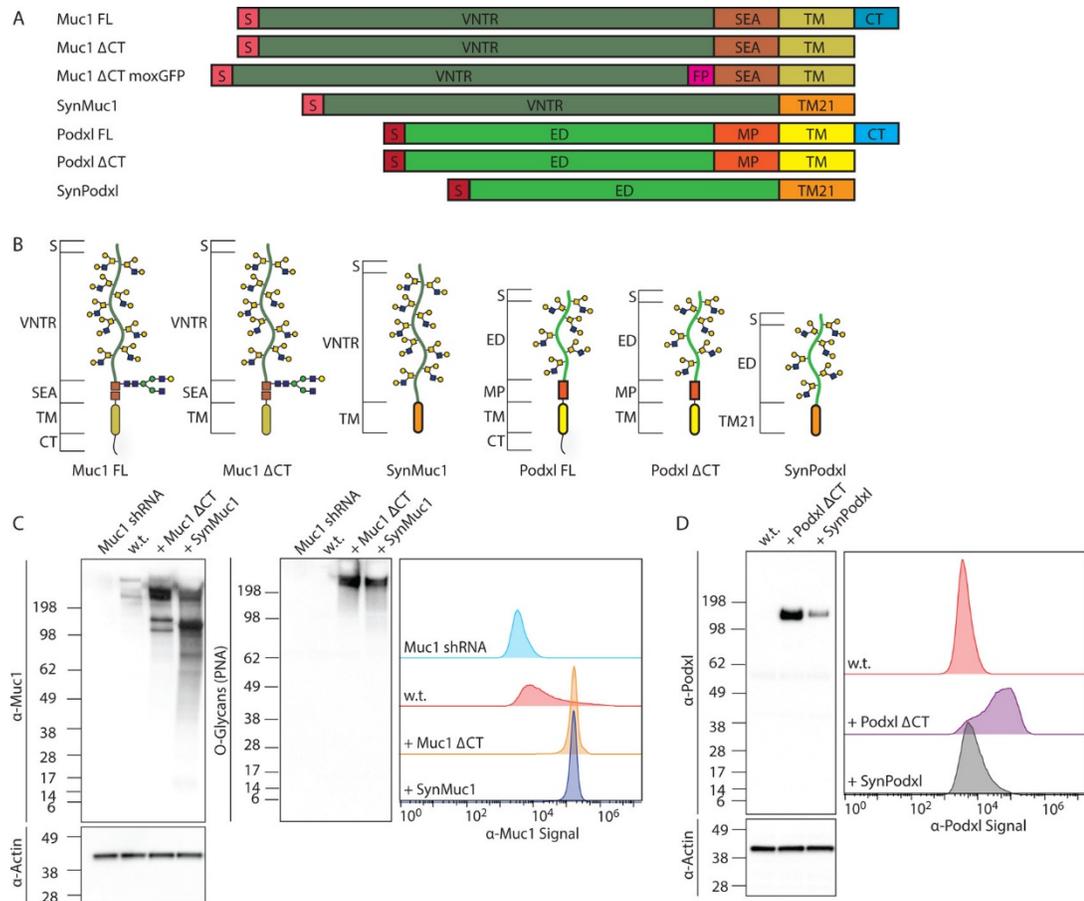
cytometry upon induction with various doxycycline concentrations, n = 3. \* p < 0.05; \*\* p < 0.01 (two-tailed t test).

We next tested the viability of a transposon-based system<sup>41-43</sup> for stable expression of large, repetitive glycoproteins like mucins (Figure 2.1A).<sup>41</sup> We found that Muc1 expression levels were dramatically improved with the transposon system compared to lentiviral transduction (Figure 2.1B,C). The mucins expressed in transposon-edited cells were heavily glycosylated and had a high molecular weight (Figure 2.1B). Finally, we confirmed that the mucin expression levels could be tuned through doxycycline induction (Figure 2.1D,E). On the basis of this performance, the inducible transposon system was applied for all subsequent editing of the glycocalyx.

### **Genetically Encoded Toolkit for Editing the O-Linked Glycocalyx**

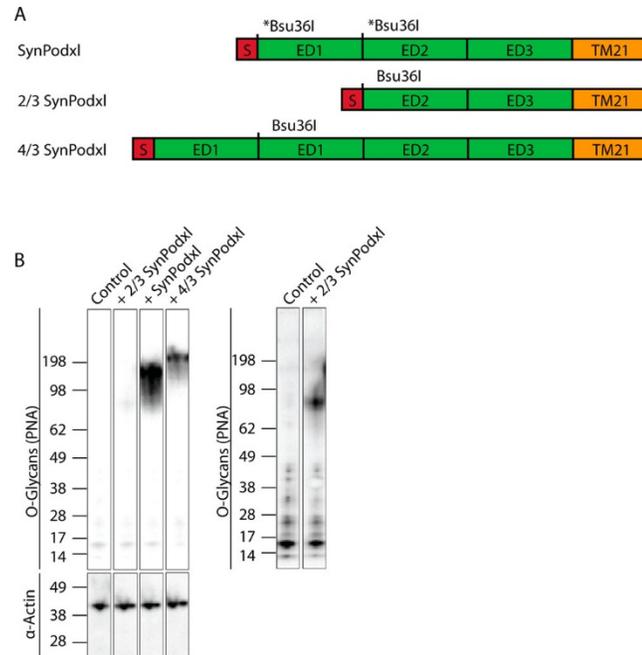
Our next goal was to design and fabricate a series of constructs for engineering the structure and O-glycan composition of the glycocalyx. Mucin glycoproteins are defined by their densely clustered O-glycans, which help to extend and rigidify the mucin polypeptide backbone.<sup>40,44</sup> As a consequence, mucins extend above most other glycoproteins on the cell surface and are ideal candidates for glycocalyx engineering. Our strategy was to create a library of mutant and semisynthetic mucins that would serve as structural elements and glycan scaffolds on the cell surface. Starting with two relevant mucins in the cancer cell glycocalyx, Muc1 and podocalyxin (Podxl, a structurally analogous glycoprotein<sup>45</sup>), we removed their cytoplasmic signaling domains ( $\Delta$ CT; Figure 2.2A,B). Our goal was to minimize the biochemical activity of the mucins so that the mutants would function primarily as structural elements in the

glycocalyx. In order to track phenotypic and morphological changes as a function of glycocalyx thickening, we also inserted a fluorescent protein between the O-glycan-rich tandem repeats and membrane proximal domains of Muc1 ( $\Delta$ CT moxGFP; Figure 2.2A). To effectively eliminate signaling motifs that could be present in native mucins, we next created a series of semisynthetic constructs. We fused the O-glycan rich domains from Muc1 and Podxl to a partially synthetic membrane proximal region, cytoplasmic domain, and transmembrane domain<sup>46</sup> (SynMuc1 and SynPodxl; Figure 2.2A,B). We designated these semisynthetic glycoproteins as SynMucins. All constructs were expressed efficiently through our transposon based system and modified the O-glycan content of the cell surface accordingly (Figure 2.2C,D).



**Figure 2.2 Genetically encoded glycoproteins for glycofocalyx editing. (A)** Schematic representation of the components and features of the native and engineered glycoproteins used in this study. S, signal sequence; VNTR, variable number tandem repeat; SEA, sea-urchin sperm protein, enterokinase, and agrin domain; TM, transmembrane domain; CT, cytoplasmic tail; FP, fluorescent protein; TM21, synthetic transmembrane domain, 21 amino acids; MP, membrane proximal domain; and ED, O-glycosylation rich ecto-domain region. **(B)** Cartoon illustration of the approximate relative size and features of various engineered glycoproteins. **(C)** Immunoblot (left) and lectin blot (center) showing the molecular weights and expression levels of Muc1 mutants in mammary epithelial cells (MECs) stably expressing the indicated gene, n = 3. Flow cytometry histograms (right) of the  $\alpha$ -Muc1 antibody binding in cells expressing each of the engineered mutants compared to knockdown (shRNA) and w.t. cells, > 50,000 cells measured per condition. **(D)** Immunoblot (left) showing the relative size and expression level of Podxl mutants in stable MEC cell lines, n = 3. Flow cytometry histograms (right) of  $\alpha$ -Podxl antibody binding in the same cell lines; >50,000 cells measured per condition.

We also developed a modular strategy for generating SynMucins of varying length. We introduced Bsu36I restriction sites that would serve as handles for the removal or addition of O-glycan-rich blocks in our SynPodxl construct. We validated this approach by generating synthetic mucins with O-linked glycosylation domains of  $4/3$  and  $2/3$  the length of native Podxl (Figure 2.3A). In principle, this approach could be adopted to generate constructs of increasingly large size. Lectin blots confirmed the expression of O-rich glycoproteins of varying size by our  $2/3$ ,  $1/1$ , and  $4/3$  length SynPodxl constructs (Figure 2.3B). Notably, lectin binding to the  $2/3$  mutant is decreased compared to  $1/1$  and  $4/3$  lengths indicating an expected decrease in O-glycosylation due to the decrease in glycosylation sites on the protein. Immunoblot analysis with an anti-Podxl antibody verified expression of the higher molecular weight  $4/3$  SynPodxl, while the  $2/3$  length glycoprotein was not detected by immunoblot, likely due to the inadvertent deletion of the antibody recognition motif in this construct (Supporting Figure 1). Together, our library of constructs and expression systems provide a toolkit for expression of structural glycoproteins of varying size and tunable density on the cell surface.

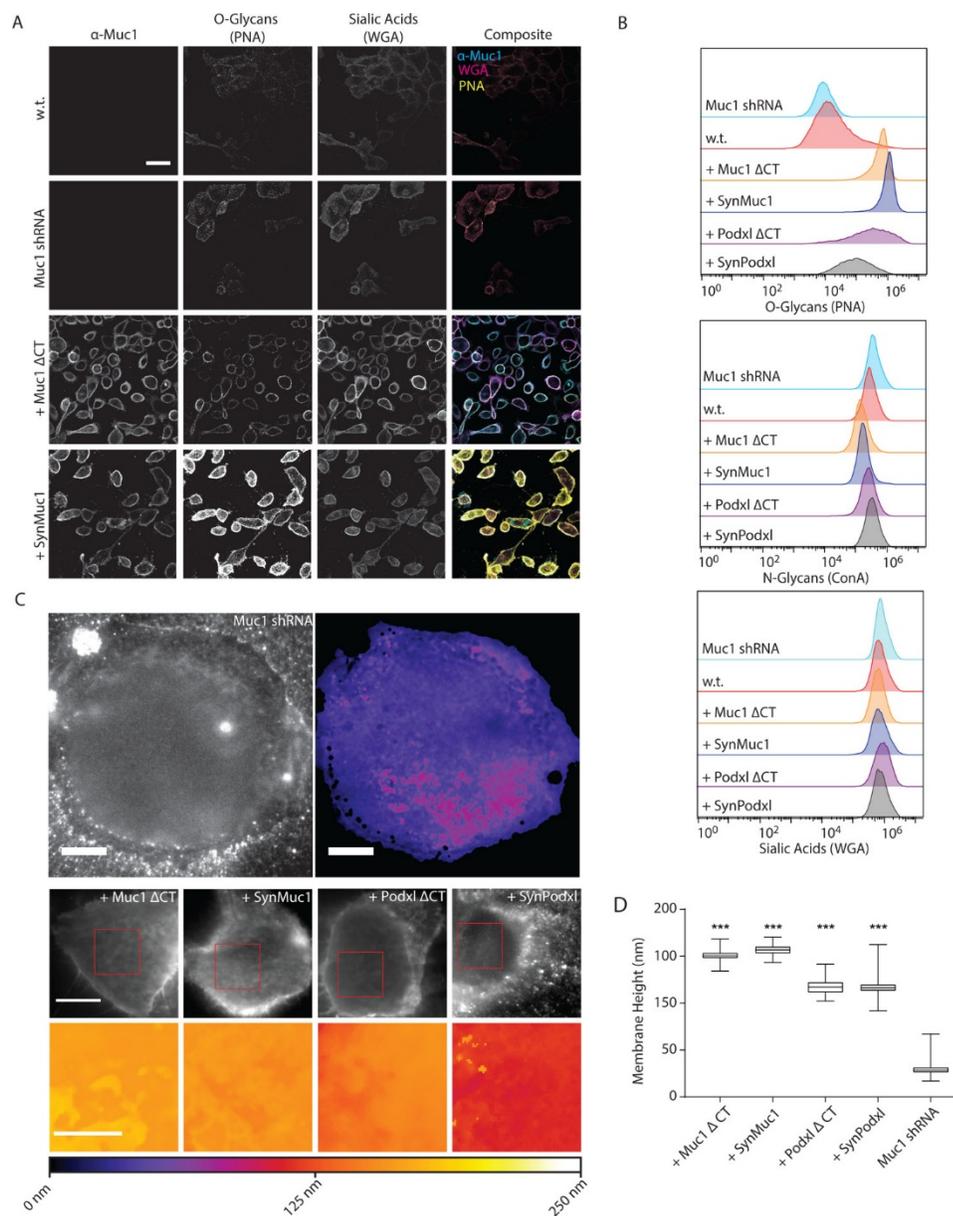


**Figure 2.3 Glycoproteins of tunable length. (A)** Schematic illustration of the restriction sites introduced via mutation (\*), reassembly fragments, and relative lengths of the engineered SynPodxl variants. **(B)** Lectin blot (left) of the relative O-glycosylation level of each mutant transiently expressed in HEK293T cells and an increased exposure time of the same lectin blot (right), n = 2.

### Altering the Chemical and Physical Environment at the Cell Surface

We next tested the ability of our toolkit to modify the chemical composition and physical structure of the cell surface. Our model cell line was the nontransformed mammary epithelial cell (MEC) line, MCF10A, which has low endogenous Muc1 and undetectable Podxl expression. We found that the MCF10A cell line has low overall levels of cell-surface O-glycans, making it an ideal system for directed assembly of an O-glycan rich glycocalyx (Figure 2.4A,B). We first demonstrated an ability of our mutant and semisynthetic mucins to alter the chemical environment of the cell surface (Figure 2.4A,B). We observed significantly increased levels of O-glycans on the surface of cells expressing our mutant and semisynthetic mucins compared to those of

control MCF10A and Muc1-knockdown cells (Muc1 shRNA). We did not detect changes in N-glycosylation or sialic acids by our mucin constructs as detected by lectins (Figure 2.4B).



**Figure 2.4 Engineering the chemical and physical properties of the**

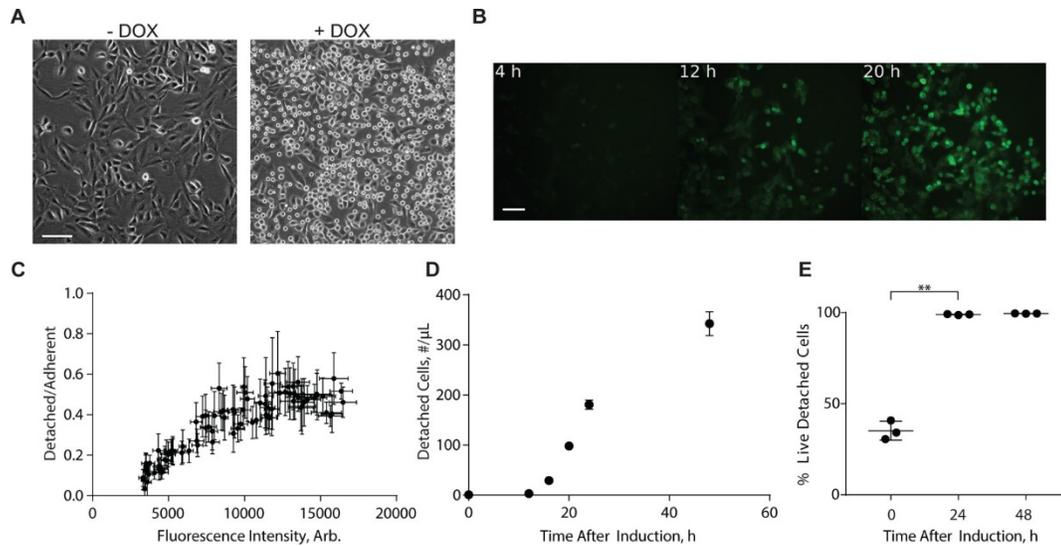
**glycocalyx.** (A) Confocal images of Muc1 (left), O-glycan (center left), and sialic acid (center right) showing cell surface localization in control and stably expressing MECs for each indicated construct (scale bar 25  $\mu\text{m}$ ),  $n = 3$ . (B) Representative flow cytometry histograms showing cell surface O and N-glycan and sialic acid levels of Muc1 and Podxl  $\Delta\text{CT}$  and SynMucin mutants;  $>20,000$  cells measured per condition and  $n = 3$ . (C) Representative fluorescence maximum intensity projections of SAIM image sequences (top left, second row, scale bar 10  $\mu\text{m}$ ) and membrane height reconstructions (top right, scale bar 10  $\mu\text{m}$ , bottom row, scale bar 5  $\mu\text{m}$ ) of the ROIs boxed in red. (D) Average membrane height from 2k pixels in each of 5 cells per condition. Boxes correspond to SD and whiskers to extrema. Asterisks indicate statistical comparison of each mucin-expressing cell line to shRNA Muc1 expressing cells. \*  $p < 0.05$ ; \*\*  $p < 0.01$ ; \*\*\*  $p < 0.001$  (one-way ANOVA).

We also tested the performance of our genetic tools in modifying the physical structure of the cellular glycocalyx. We used scanning angle interference microscopy (SAIM),<sup>47</sup> a fluorescence localization technique with 5–10 nm axial precision, to map changes in the plasma membrane topography and glycocalyx thickness following the expression of our mutant and synthetic constructs (Figure 2.4C,D). We made all measurements on cells that were adhered to silicon substrates adsorbed with fibronectin. Muc1 knockdown MECs had an average ventral membrane height of 30.3 nm (SD 7.83), while Muc1  $\Delta\text{CT}$  expression increased the separation between the ventral plasma membrane and the substrate by an average of 120 nm (150.3 nm, SD 15.9) in comparison. The shorter Podxl  $\Delta\text{CT}$  generated a more modest average change of 87 nm (117.3 nm, SD 13.1) compared to the knockdown cells, as expected based on the lower molecular weight of Podxl versus Muc1. Notably, the SynMuc1 and SynPodxl constructs significantly expanded the glycocalyx and performed comparably to the  $\Delta\text{CT}$  mutants of the same glycoproteins (156.6 nm, SD 6.12 and 119.6 nm, SD 12.9, respectively). Together, these results confirm the utility of our toolkit for editing

both the chemical composition and physical structure of the glycocalyx.

### **Thick and Dense Glycocalyx Can Trigger Cellular Detachment from the Matrix**

Using our toolkit, we generated cellular models to investigate the consequence of a thick, O-glycan rich glycocalyx on cellular behaviors. This glycocalyx is typical of many cancer cells, including highly aggressive cancers presenting with CTCs.<sup>8</sup> We first tested the implications of dense and thick glycocalyx on cellular adhesion to the ECM. Our analysis focused on our Muc1 mutant construct, which is the largest of our engineered glycoproteins. Notably, we found that a thick and dense glycocalyx could trigger complete cellular detachment from the ECM (Figure 2.5A,B and Supporting Movie 1). In live-cell time-course experiments that tracked morphological changes dynamically following doxycycline induction, we observed a transition from a classical cuboidal cell phenotype to a more rounded morphology, and finally to a completely detached state, where cells float as spheres above the ECM substrate (Figure 2.5B and Supporting Movie 1).



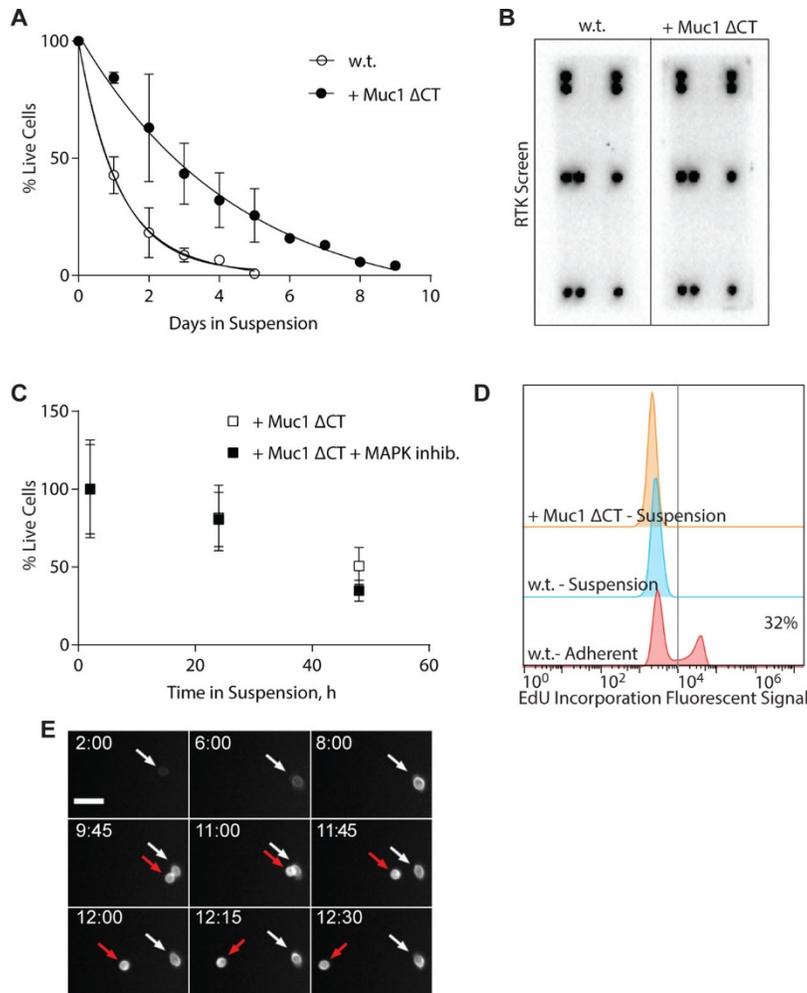
**Figure 2.5 Expression of a bulky glycolyx inhibits cellular adhesion. (A)** Representative phase-contrast images of MECs expressing Muc1  $\Delta$ CT, uninduced, and induced for 24 h (scale bar 100  $\mu$ m). **(B)** Representative epifluorescence images of MECs expressing Muc1  $\Delta$ CT moxGFP 4, 12, and 20 h postinduction (scale bar 100  $\mu$ m),  $n = 2$ . **(C)** Ratio of detached MECs to adherent as a function of relative Muc1 expression measured by Muc1  $\Delta$ CT moxGFP fluorescence intensity,  $n = 2$ . **(D)** Concentration of live, detached MECs expressing Muc1  $\Delta$ CT as a function of time postinduction, quantified by flow cytometry with live/dead cell stain,  $n = 3$ . **(E)** Percentage of live cells in total detached population at various time points after induction quantified by flow cytometry with live/dead cell stain,  $n = 3$ . All error bars are SD.

The fraction of detached cells correlated with Muc1 surface expression, strongly suggesting that detachment was a consequence of high mucin surface densities (Figure 2.5C). Indeed, the kinetics of cellular detachment during the first 24 h after doxycycline addition roughly matched the temporal increase in mucin protein expression levels (Figure 2.5D). We continued to see an increase in the detached cell fraction after 24 h, the time at which maximal protein expression was obtained (Figure 2.5D). Unexpectedly, we discovered that the majority of the detached cells were viable 24 and 48 h after induction (Figure 2.5E). On the basis of these observations, we hypothesized that the thick, O-glycan-rich glycolyx enhanced cell survival in poorly

adhesive conditions.

### **Glycocalyx Prolongs Cellular Survival in Suspension**

Given the surprising ability of cells with a prominent glycocalyx to survive in anchorage-free conditions, we investigated the survival and proliferation of these cells in suspension culture over longer timeframes. We envisioned these studies would provide a basic model of CTCs, which frequently have a thick O-glycan rich glycocalyx and survive, suspended in circulation during dissemination.<sup>8</sup> We observed that MECs engineered to have a thick and dense glycocalyx live approximately four times longer in suspension than control cells (Figure 2.6A). The viability of control and engineered cells in suspension was well modeled by an exponential decay with a single decay constant, the half-life. We found that the half-life for MECs with an engineered glycocalyx is 3.12 days (1.96 to 6.51, 95% CI) compared to that of control MECs, which have a half-life of 0.797 days (0.623 to 1.03, 95% CI). We next tested whether this survival phenotype could be attributed to increased receptor tyrosine kinase (RTK) signaling. In a screen of 28 RTKs and 11 signaling nodes, no apparent differences in phosphorylation were observed when comparing engineered and control MECs grown in suspension for 24 h (Figure 2.6B and Supporting Figure 2). Furthermore, inhibition of MAPK signaling with the inhibitor U-0126 did not significantly change the decay constant for survival in engineered MECs, suggesting the promotion of survival by the glycocalyx was independent of MAPK (Figure 2.6C).



**Figure 2.6 Expression of a bulky glycoalyx enhances cell survival in suspension. (A)** Percentage live cells as a function of time for MECs growing in suspension culture,  $n = 3$ . The solid curves represent a single exponential decay fit,  $t_{w.t.1/2} = 0.797$  d,  $t_{Muc11/2} = 3.12$  d. **(B)** Receptor tyrosine kinase signaling screen for MECs grown in suspension in full serum for 24 h,  $n = 2$ . Visible signal from positive controls. See Supporting Figure 2 for full details. **(C)** Percent live MECs expressing Muc1  $\Delta$ CT grown in suspension with 10  $\mu$ M MAPK inhibitor U-0126 or DMSO control,  $n = 2$ . **(D)** Representative flow cytometry histogram showing incorporation of EdU into MECs under adherent or in suspension culture conditions;  $> 100,000$  cells measured per condition,  $n = 2$ . Percentage indicates the fraction of cells with signal  $> 10^4$  (gray line) after background subtraction. **(E)** Epifluorescence images of an MEC undergoing division with increasing expression of Muc1  $\Delta$ CT moxGFP. Following separation, one of the new cells retains attachment to the substrate (white arrow), while the other detaches and drifts out of frame (red arrow) (scale bar

50  $\mu\text{m}$ ). All error bars are SD. Timestamp shown in hour:min.

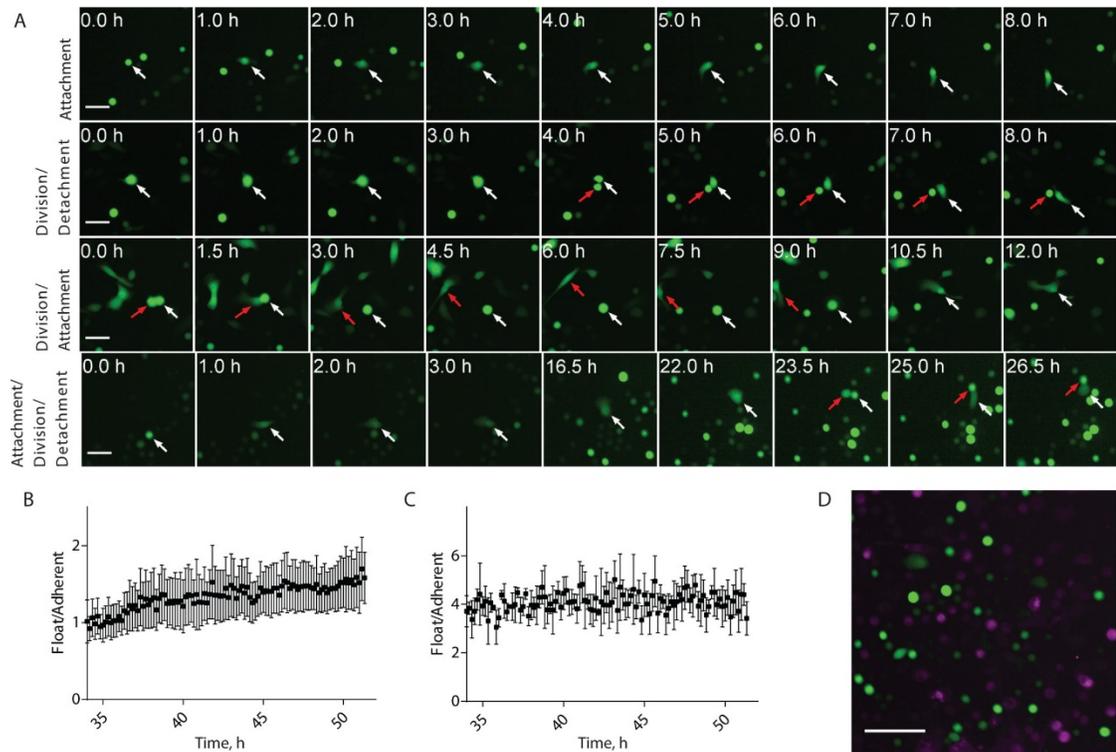
We next assayed whether cells in suspension were actively proliferating. Despite their prolonged survival in suspension, we observed no proliferation by our engineered cells while in this state indicating that these cells still require anchorage to undergo cell division (Figure 2.6D). Using time-lapse imaging and single-cell tracking, we determined that the engineered cells proliferate while in an anchored state, but frequently detach from the matrix following division (Figure 2.6E). Prior to division, cells expressing Muc1  $\Delta\text{CT}$  adopted a nearly spherical morphology but did not fully detach from the substrate. At the moment of division, however, one or both daughter cells would often lose attachment. Taken together, these results indicate that high levels of cell-surface mucins can trigger a switch from an adhesion-dependent phenotype to an adhesion-free phenotype with enhanced viability.

### **Cells with a Thick Glycocalyx Dynamically Attach to the Matrix for Proliferation**

We next considered two possible mechanisms for how cells with a thick glycocalyx might proliferate: (1) that upon division, the glycocalyx biomass is asymmetrically transferred to daughter cells, leading to a split population of adherent and detached cells or (2) that previously detached cells regain attachment to divide. To test, we generated two populations of Muc1  $\Delta\text{CT}$  expressing MECs, one with constitutive cytoplasmic EGFP and a second with cytoplasmic mCherry. Twenty-four hours following the induction of Muc1  $\Delta\text{CT}$  expression, the supernatant of each culture containing detached cells was exchanged, giving mixed cultures of mCherry and EGFP cells, allowing us to track the progress of both starting conditions in a

single experiment.

In these experiments, a variety of behaviors were observed, including reattachment of floating cells, detachment of daughter cells following division, and reattachment and spreading of daughter cells following division (Figure 2.7A and Supporting Movies 2 and 3). Another frequent occurrence was the temporary detachment of one or both daughter cells following division (Figure 2.7A, row 3). Most interestingly, floating cells also were observed to temporarily attach to the substrate, divide, then detach again (Figure 2.7A, row 4 and Supporting Movie 4). In the case of the initially attached condition, we observed a monotonically increasing ratio of floating to adherent cell populations over the 28 h following exchange, suggesting that the dynamically adherent subpopulation continuously generated suspended daughter cells (Figure 2.7B). On the other hand, the initially detached population reached an equilibrium within 10 h of exchange as demonstrated by the constant population ratio (Figure 2.7C). Similar results were observed for both starting conditions in the mCherry and EGFP cell lines leading to mixed populations in both states, indicating that the transfer was bidirectional for either starting condition (Figure 2.7D). These results show that cells switch between the substrate attached and free floating states with individual cells occupying both as a function of cell cycle.



**Figure 2.7 Cellular division is associated with adhesion.** (A) Live cell time-course images of a detached cell reattaching to the substrate (top), a division event wherein one of the new cells detaches from the substrate (top center, white arrow), a division event wherein a new cell briefly detaches then reattaches to the substrate (bottom center, white arrow), and a detached cell attaches to the substrate, divides, then one of the new cells detaches (bottom, white arrow) (scale bars 50  $\mu\text{m}$ ). (B) Ratio of detached, floating cells to adherent as a function of time in an initially attached population expressing Muc1  $\Delta\text{CT}$ . (C) The same measurement as B for an initially detached population. Error bars in B and C represent one SD,  $n = 3$  for both conditions. (D) Epifluorescence image of a mixed population of MECs expressing Muc1  $\Delta\text{CT}$  and either cytoplasmic EGFP (green, initially detached) or mCherry (magenta, initially attached) demonstrating the exchange of states occurs in both directions (scale bar 100  $\mu\text{m}$ ).

## 2.4 Conclusion

We developed a library of mutant and semisynthetic constructs for expressing heavily O-glycosylated proteins of varying size. Combined with a system for temporal control over expression levels, our systems constitute a genetically encoded toolbox

for engineering the cellular glycocalyx. We anticipate that our technology for glycocalyx editing will open new avenues of research previously inaccessible, including more precise modeling of the cancer-specific glycocalyx for in vitro and in vivo studies. Indeed, the temporal control and stable expression that our systems afford enable dynamic tuning of the glycocalyx in organoids, as well as syngeneic and xenogeneic mouse models. Furthermore, our system provides new “knobs” for tuning key features of the glycocalyx, including glycan content, glycoprotein size distribution, and glycoprotein surface density.

An important contribution by this work is the design and validation of a series of semisynthetic mucins. While cell-surface mucins have a dominant structural function in the cancer cell glycocalyx, they also have strong biochemical activities mediated by their cytoplasmic and membrane proximal domains.<sup>48</sup> Our semisynthetic mucins, SynMuc1 and SynPodxl, should enable more reliable physical editing of the glycocalyx, while minimizing the direct biochemical activity of the expressed glycoproteins. We show that our approach for manufacturing synthetic glycoproteins is effective in modulating the thickness and biochemical properties of the glycocalyx. Future studies could add control over the flexural rigidity of individual structural elements. For example, the degree of glycosylation can control the persistence length of mucin biomimetic polymers,<sup>44</sup> and an analogous strategy to engineer the number of serine and threonine glycosylation sites in genetically encoded mucins could be conducted. As another example, membrane-proximal regions of native Muc1 have been shown to regulate Muc1 trafficking and glycosylation, including extent of glycosylation and sialylation.<sup>49–51</sup> The dependence of glycosylation on trafficking may

explain why we observe more under-glycosylated mucin with SynMuc1, which lacks native membrane proximal and transmembrane domains (Figure 2.2C). This may also explain the surprising result that, while we increase the O-glycan content of the glycocalyx upon expression of our various constructs, we do not observe a significant change in sialic acid content of the glycocalyx (Figure 2.4B). Additional rounds of protein engineering could also mechanically optimize the hinge point between the O-rich glycodomains and our synthetic membrane proximal region to reinforce the glycocalyx under compressive forces. Thus, we envision that future efforts could build on our current library to provide greater control over the physical structure and deformability of the glycocalyx.

Our toolkit overcomes some of the key technical hurdles that have historically limited the generation of cell lines with a cancer-specific glycocalyx. Foremost, we provide a strategy for generating cells with a thick and dense glycocalyx, similar to those of aggressive cancer cells. On the basis of our observation that cells with a thick and dense glycocalyx frequently detach from the ECM substrate, we envision that routine cell culture could quickly select against cells with a prominent glycocalyx. Indeed, many of the existing cancer cell lines have been generated through repeated subculturing, which would likely select against a thick and less compliant glycocalyx that causes cells to detach from the culture vessel. Our inducible system solves this problem by enabling temporal control over glycocalyx biosynthesis in real-time at the start of experimentation.

An unexpected observation in our work is the strong induction of survival by the glycocalyx for cells in a suspended state. Previous work has shown that a thick

glycocalyx can support cell survival of anchored cells by promoting integrin assembly, signaling, and cross-talk with growth factor receptor pathways. Now we show that the glycocalyx can also support survival of suspended cells. Importantly, our RTK screen and MAPK inhibitor data suggest that this survival mechanism is independent of growth factor receptor signaling. The precise mechanism of action for the glycocalyx in this adhesion-independent survival pathway must be resolved in future studies.

In summary, our work describes a set of genetic tools for glycocalyx engineering and rational construction of a cancer-specific glycocalyx in cell lines of interest. Our toolkit provides a necessary technology for addressing unresolved questions in cancer glycobiology, including why CTCs and aggressive cancer cells frequently overexpress large glycoproteins.

## **2.5 Materials and Methods**

### **Antibodies and Reagents**

The following antibodies were used: FITC-human CD227 (Muc1) (559774, BD Biosciences), human CD227 (555925, BD Biosciences) (Muc1), Alexa-488-human podocalyxin (222328, R&D Systems), actin (sc1615, Santa Cruz), goat anti-mouse IgG-HRP (sc-2005, Santa Cruz), and mouse anti-goat IgG-HRP (sc-2354, Santa Cruz). Lectins used were: biotinylated peanut agglutinin (PNA; B-1075, Vector Laboratories), CF568 Arachis hypogaea Lectin PNA (29061, Biotium), CF640R Arachis hypogaea lectin PNA (29063, Biotium), CF633 wheat germ agglutinin (WGA; 29024, Biotium), and FITC concanavalin A (ConA; FL-1001, Vector Laboratories). Biotinylated lectins were detected using ExtrAvidin-Peroxidase (E2886,

Sigma). MAPK inhibitor was U-0126 (70970, Cayman Chemical). To induce transactivator cell lines, doxycycline was used (sc-204734, Santa Cruz).

### **Cloning and Constructs**

Full details on cloning and assembly of lentiviral, piggybac, and transient expression vectors with SynMucins are given in the Supporting Information.

### **Cell Lines and Culture**

MCF10A and HEK293T cells were obtained from ATCC. MCF10A cells were cultured in DMEM/F12 media supplemented with 5% horse serum, 20 ng/mL EGF, 10 µg/mL insulin, 500 ng/mL hydrocortisone, and 100 ng/mL cholera toxin. HEK293T cells were cultured in DMEM high glucose supplemented with 10% fetal bovine serum. Cells were maintained at 37 °C, 5% CO<sub>2</sub>, and 90% RH. MCF10A shRNA Muc1 cells were prepared by lentiviral transformation using Muc1 shRNA pLKO.1. MCF10A rtTA stable line was prepared by lentiviral transformation using pLV rtTA-NeoR plasmid as previously described.<sup>8,47</sup> Cells were further modified with the transposon system to create stable lines expressing Muc1 ΔCT; SynMuc1; Muc1 ΔCT moxGFP; Podxl ΔCT; and 2/3, 1/1, and 4/3 SynPodxl. These stable lines were generated using Nucleofection Kit V (Lonza) and HyPBBase, an expression plasmid for a hyperactive version of the PiggyBac transposase<sup>52</sup> (kindly provided by Dr. Alan Bradley, Wellcome Trust Sanger Institute, UK). MCF10A rtTA Muc1 ΔCT and MCF10A rtTA SynMuc1 were sorted for expression using the FITC-Human CD227 antibody. MCF10A rtTA Muc1 ΔCT EGFP and MCF10A rtTA Muc1 ΔCT mCherry

cells were generated by further lentiviral modification of MCF10A rtTA Muc1  $\Delta$ CT cells with pLenti EGFP (Addgene plasmid #17446<sup>53</sup>) and pCDH mCherry, respectively. Additionally, a Muc1  $\Delta$ CT stable line was created by lentiviral transformation of the MCF10A rtTA cell line using a pLV tetOn Muc1  $\Delta$ CT plasmid. HEK293T cells were transiently transfected using a calcium phosphate transfection protocol with pMAX GFP, pcDNA3.1(+) 2/3 SynPodxl, pcDNA3.1(+) 1/1 SynPodxl, and pcDNA3.1(+) 4/3 SynPodxl plasmids.

### **Immunoblot Analysis**

Cells were plated at 20,000 cells/cm<sup>2</sup> and grown for 24 h. Cells were then induced with 0.2  $\mu$ g/mL doxycycline for 24 h before lysis with Tris-Triton lysis buffer (Abcam). Any detached cells were included in lysates. Lysates were separated on Nupage 4–12% Bis-Tris gels and transferred to PVDF membranes. Membranes were blocked with 3% BSA TBST solutions for 2 h at room temperature or overnight at 4 °C. Primary antibodies were diluted 1:1000 and lectins were diluted to 1  $\mu$ g/mL in 3% BSA TBST and incubated 4 h at room temperature or overnight at 4 °C. Secondary antibodies or ExtrAvidin were diluted 1:2000 in 3% BSA TBST and incubated for 2 h at room temperature. Blots were developed in Clarity ECL (BioRad) substrate, imaged on a ChemiDoc (BioRad) documentation system, and quantified in Fiji.<sup>54</sup>

### **Flow Cytometry**

Cells were plated at 20,000 cells/cm<sup>2</sup> and grown for 24 h. Cells were then induced with 0.2  $\mu$ g/mL doxycycline for 24 h. Adherent cells were nonezymatically

detached by incubating with 1 mM EGTA in PBS at 37 °C for 20 min and added to the population of floating cells, if present. Antibodies were diluted 1:200, and lectins were diluted to 1 µg/mL in 0.5% BSA PBS and incubated with cells at 4 °C for 30 min. The BD Accuri C6 flow cytometer was used for analysis. For doxycycline-titration flow cytometry, Muc1 ΔCT moxGFP expressing cells were plated and induced similarly with varying concentrations of doxycycline (0, 1, 5, 10, 25, 50, 75, 100, 200, 350, 500, and 1000 ng/mL). Cells were nonenzymatically detached as described above and fixed with 4% paraformaldehyde, and the GFP signal was analyzed on the BD Accuri C6 flow cytometer. Median GFP signal is reported.

### **Confocal Microscopy**

Cells were plated at 5,000 cells/cm<sup>2</sup> and grown for 24 h and subsequently induced with 0.2 µg/mL of doxycycline for 24 h before being fixed with 4% paraformaldehyde. Samples were blocked with 5% normal goat serum PBS for 1 h at room temperature. Antibodies were diluted 1:200 in 5% normal goat serum PBS and incubated overnight at 4 °C. Lectins were diluted to 1 µg/mL in 5% normal goat serum PBS and incubated for 2 h at room temperature. Samples were imaged on a Zeiss LSM inverted 880 confocal microscope using a 40× water immersion objective.

### **Cell Detachment Time Course**

Cells were plated at 20,000 cells/cm<sup>2</sup> and grown for 24 h. All media were changed to remove any detached cells at this time. Cells were induced by spiking 0.2 µg/mL doxycycline into the media at the designated time points. Detached cells were

collected, stained with Sytox-Green (Thermo), and analyzed with a BD Accuri C6 flow cytometer.

### **Suspension Cultures**

Cells were grown and induced with 0.2 µg/mL doxycycline for 24 h using an adherent cell culture technique. Cells were then detached with 0.05% trypsin EDTA (Thermo) and resuspended at 375,000 cells/mL. Cells were grown in suspension using a cell-culture magnetic stirrer at 100 rpm, 37 °C, 5% CO<sub>2</sub>, and 90% RH. Every 24 h, a small sample was drawn, and cells were counted with a hemocytometer using Trypan Blue (Thermo) to differentiate live and dead cells. For inhibitor studies, 10 µM MAPK inhibitor or DMSO (control) was added when cells were transferred to suspension.

### **RTK Screen**

Cells were grown in suspension or adherent populations and induced for 24 h. Cells were then lysed per the previous discussion with the addition of sodium fluoride and sodium orthovanadate to the lysis buffer. The PathScan RTK Signaling Antibody Array Kit (7982, Cell Signaling) protocol was used to complete the assay using 0.5 mg/mL of each lysate.

### **EdU Proliferation Assay**

Click-iT Plus EdU Alexa Fluor 594 Flow Cytometry Assay Kit (Thermo) was used. Cells were grown in suspension for 22 h as described above before adding 10 µM EdU to the culture for 2 h. Adherent cells were plated at 20,000 cells/cm<sup>2</sup> and

grown for 46 h before adding 10  $\mu$ M EdU to the culture. The procedure was completed per the assay kit manual, and samples were analyzed on a BD Accuri C6 flow cytometer.

### **Scanning Angle Interference Microscopy**

Silicon wafers with 1900 nm thermal oxide were purchased from Addison Engineering and diced into 1 cm<sup>2</sup> chips, cleaned in 5 M NaOH, rinsed extensively with deionized water, then oxygen plasma cleaned for 60 s before coating with 50  $\mu$ g/mL human plasma fibronectin (Millipore) for 2 h at 37 °C. Cells were seeded onto the prepared substrates at 5,000 cells/cm<sup>2</sup> and cultured for 24 h before the addition of doxycycline, cultured for an additional 24 h, then fixed in 4% paraformaldehyde at 37 °C for 15 min. Following fixation samples were rinsed extensively in PBS, then stained with 1  $\mu$ g/mL DiI-DS (ThermoFisher) for 1 h at RT. Samples were imaged with a custom SAIM microscope based on a Nikon Ti-E body with ultrastage and focus lock (FocalPoint) with a 560 nm excitation laser (MFB). Experiments were automated with a homemade microscope controller and galvanometer scanning mirror (Cambridge Technology), and images were acquired over a series of 62 angles from 0 to 36.7° with a 60 $\times$  NA 1.2 water immersion objective (Nikon) on a sCMOS camera (Zyla 4.2, Andor) controlled with  $\mu$ Manager software (Open Imaging).<sup>55</sup> Image sequences were fit pixel-wise in a least-squares sense according to the SAIM optical model<sup>47</sup> with a custom software application (source code available upon request), and the resulting reconstructions were postprocessed using Fiji. For each condition, 2–4 independent samples were imaged. Because of variability in label density and the

effects of dye aggregates in many cells, only a small portion of the membrane provided an adequate reconstruction. Cells with globally low residuals and continuous  $40 \times 50$  (2k total) pixel regions in the reconstructions were selected from the pooled samples after reconstruction. A rectangular box of  $40 \times 50$  pixels was drawn directly under the cell body, as the region on the periphery is prone to artifacts in reconstruction owing to the membrane being oriented roughly parallel to the optical axis or the close proximity between the ventral and apical membranes. The pixels heights within these boxes were used for quantification of membrane height.

### **In-incubator Microscope**

Live cell time-course imaging was carried out on a custom-built microscope designed to operate in a standard cell culture incubator. The microscope consists of a mechanical XY stage, with separate focus drive, and filter wheel controlled by a central hub (ASI Tiger) and operated through  $\mu$ Manager control software.<sup>55</sup> Excitation by LED sources (ThorLabs) was controlled by an Arduino board through  $\mu$ Manager. A Nikon 10 $\times$  NA 0.50 CFI S Fluor objective was used to image onto a CMOS camera (PointGrey). Full details on microscope design are available upon request.

### **Live-Cell Time-Course Imaging**

Glass bottomed cell culture plates (Cellvis) were oxygen plasma cleaned for 60 s then coated with 50  $\mu$ g/mL human plasma fibronectin for 2 h at 37  $^{\circ}$ C. Cells were seeded at 5,000 cells/cm<sup>2</sup> and cultured for 24 h prior to the addition of doxycycline. Following induction, the sample was placed on the in-incubator microscope and

allowed to equilibrate for 4 h. Images were then acquired for 20 h at multiple positions in 2 different wells for induction time-course series. For adhesion/detachment studies, cells were initially imaged for 24 h following induction, then the plate removed and supernatant swapped between each of 3 wells of EGFP and mCherry cell lines. The plate was then replaced and imaging continued for an additional 33 h. Images were analyzed with custom analysis software written by the authors and available upon request. Briefly, in each frame circular cells and all cells are identified, then a mean intensity calculated for all cells in the frame. The number of adherent cells is then inferred as the difference between circular and total cells.

## **Statistics**

Statistical significance was determined by Student's t test, two-tailed, or ordinary one-way ANOVA as appropriate. Statistical analysis was performed using Prism (GraphPad)

## **2.6 Acknowledgements**

We thank V. Weaver for the lentiviral and transposon plasmids, as well as helpful discussions. This investigation was supported by the National Institute of General Medical Sciences Ruth L. Kirschstein National Research Service Award 2T32GM008267 (to C.R.S., M.J.C.), Knight Family Foundation Graduate Research Fellowship in Nanoscience and Technology (to C.R.S.), Samuel C. Fleming Family Graduate Fellowship (C.R.S.), National Science Foundation Graduate Research Fellowship DGE-1650441 (M.J.C.), Canadian Institutes of Health Research (F.K.),

National Institute of Health New Innovator DP2 GM229133 (M.J.P.), and National Cancer Institute U54 CA210184 (M.J.P.) and R33-CA193043 (M.J.P.). Confocal imaging was supported through the Cornell University Biotechnology Resource Center (BRC) Imaging Facility (NIH S10OD018516).

## **2.7 Author Contributions<sup>‡</sup>**

C.R.S., M.J.C. and M.J.P. conceived the project and designed experiments. C.R.S. designed and created the transposon system incorporation vectors for Muc1 and its derivatives. M.J.C. designed and created the transposon system incorporation vectors for Podxl. F.K. and J.N.L. designed and created the transposon based stable incorporation system. M.J.C. conceived and generated the modular length Podxl cDNA constructs. C.R.S. performed flow cytometry, immunoblot, and lectin blot experiments. C.R.S. and S.E.H. performed cell viability experiments. M.J.C. performed SAIM experiments. C.R.S. and M.J.C. performed confocal microscopy experiment. M.J.C. and V.K.G. designed and built the in-incubator epifluorescence microscopes. C.R.S. performed live-cell experiments with the in-incubator microscopes. M.J.C. wrote the image analysis software for SAIM and in-incubator microscope experiments. C.R.S. and M.J.C. analyzed imaging data, performed statistical analysis, and generated figures.

---

<sup>‡</sup> Not included in the original publication.

## REFERENCES

1. Hudak, J. E. & Bertozzi, C. R. Glycotherapy: New Advances Inspire a Reemergence of Glycans in Medicine. *Chem. Biol.* **21**, 16–37 (2014).
2. Kaszuba, K. *et al.* N-Glycosylation as determinant of epidermal growth factor receptor conformation in membranes. *Proc. Natl. Acad. Sci.* **112**, 4334–4339 (2015).
3. Wesseling, J., van der Valk, S. W. & Hilkens, J. A mechanism for inhibition of E-cadherin-mediated cell-cell adhesion by the membrane-associated mucin episialin/MUC1. *Mol. Biol. Cell* **7**, 565–577 (1996).
4. Wesseling, J., van der Valk, S. W., Vos, H. L., Sonnenberg, A. & Hilkens, J. Episialin (MUC1) overexpression inhibits integrin-mediated cell adhesion to extracellular matrix components. *J. Cell Biol.* **129**, 255–265 (1995).
5. Haltiwanger, R. S. & Lowe, J. B. Role of glycosylation in development. *Annu. Rev. Biochem.* **73**, 491–537 (2004).
6. Janik, M. E., Lityńska, A. & Vereecken, P. Cell migration—the role of integrin glycosylation. *Biochim. Biophys. Acta* **1800**, 545–555 (2010).
7. Radhakrishnan, P. *et al.* Immature truncated O-glycophenotype of cancer directly induces oncogenic features. *Proc. Natl. Acad. Sci. U. S. A.* **111**, E4066–4075 (2014).
8. Paszek, M. J. *et al.* The cancer glycocalyx mechanically primes integrin-mediated growth and survival. *NATURE* **511**, 319+ (2014).
9. Xiao, H., Woods, E. C., Vukojcic, P. & Bertozzi, C. R. Precision glycocalyx editing as a strategy for cancer immunotherapy. *Proc. Natl. Acad. Sci. U. S. A.* **113**, 10304–10309 (2016).
10. Hudak, J. E., Canham, S. M. & Bertozzi, C. R. Glycocalyx engineering reveals a Siglec-based mechanism for NK cell immunoevasion. *Nat. Chem. Biol.* **10**, 69–75 (2014).
11. Marth, J. D. & Grewal, P. K. Mammalian glycosylation in immunity. *Nat. Rev. Immunol.* **8**, 874–887 (2008).
12. Freeze, H. H. Understanding human glycosylation disorders: biochemistry leads the charge. *J. Biol. Chem.* **288**, 6936–6945 (2013).
13. Ohtsubo, K. & Marth, J. D. Glycosylation in cellular mechanisms of health and disease. *Cell* **126**, 855–867 (2006).

14. Satomaa, T. *et al.* The N-glycome of human embryonic stem cells. *BMC Cell Biol.* **10**, 42 (2009).
15. Buck, C. A., Glick, M. C. & Warren, L. Glycopeptides from the surface of control and virus-transformed cells. *Science* **172**, 169–171 (1971).
16. Hakomori, S. Tumor-associated carbohydrate antigens defining tumor malignancy: basis for development of anti-cancer vaccines. *Adv. Exp. Med. Biol.* **491**, 369–402 (2001).
17. Kesimer, M. *et al.* Molecular organization of the mucins and glycocalyx underlying mucus transport over mucosal surfaces of the airways. *Mucosal Immunol.* **6**, 379–392 (2013).
18. Cruz-Chu, E. R., Malafeev, A., Pajarskas, T., Pivkin, I. V. & Koumoutsakos, P. Structure and Response to Flow of the Glycocalyx Layer. *Biophys. J.* **106**, 232–243 (2014).
19. Thi, M. M., Tarbell, J. M., Weinbaum, S. & Spray, D. C. The role of the glycocalyx in reorganization of the actin cytoskeleton under fluid shear stress: a ‘bumper-car’ model. *Proc. Natl. Acad. Sci. U. S. A.* **101**, 16483–16488 (2004).
20. Adhesion-related glycocalyx study: quantitative approach with imaging-spectrum in the energy filtering transmission electron microscope (EFTEM). - PubMed - NCBI. Available at: <https://www.ncbi.nlm.nih.gov.proxy.library.cornell.edu/pubmed/9657389?dopt=Abstract>. (Accessed: 15th August 2019)
21. Dennis, J. W., Nabi, I. R. & Demetriou, M. Metabolism, cell surface organization, and disease. *Cell* **139**, 1229–1241 (2009).
22. Lajoie, P. *et al.* Plasma membrane domain organization regulates EGFR signaling in tumor cells. *J. Cell Biol.* **179**, 341–356 (2007).
23. Ju, T. *et al.* Tn and sialyl-Tn antigens, aberrant O-glycomics as human disease markers. *PROTEOMICS – Clin. Appl.* **7**, 618–631 (2013).
24. Pinho, S. S. & Reis, C. A. Glycosylation in cancer: mechanisms and clinical implications. *Nat. Rev. Cancer* **15**, 540–555 (2015).
25. Horm, T. M. & Schroeder, J. A. MUC1 and metastatic cancer. *Cell Adhes. Migr.* **7**, 187–198 (2013).
26. Cheng, J.-P. *et al.* MUC1-positive circulating tumor cells and MUC1 protein predict chemotherapeutic efficacy in the treatment of metastatic breast cancer. *Chin. J. Cancer* **30**, 54–61 (2011).

27. Lau, K. S. *et al.* Complex N-glycan number and degree of branching cooperate to regulate cell proliferation and differentiation. *Cell* **129**, 123–134 (2007).
28. Raina, D., Kharbanda, S. & Kufe, D. The MUC1 Oncoprotein Activates the Anti-apoptotic Phosphoinositide 3-Kinase/Akt and Bcl-xL Pathways in Rat 3Y1 Fibroblasts. *J. Biol. Chem.* **279**, 20607–20612 (2004).
29. Croci, D. O. *et al.* Glycosylation-dependent lectin-receptor interactions preserve angiogenesis in anti-VEGF refractory tumors. *Cell* **156**, 744–758 (2014).
30. Snyder, K. A. *et al.* Podocalyxin enhances breast tumor growth and metastasis and is a target for monoclonal antibody therapy. *Breast Cancer Res. BCR* **17**, 46 (2015).
31. Häuselmann, I. & Borsig, L. Altered tumor-cell glycosylation promotes metastasis. *Front. Oncol.* **4**, 28 (2014).
32. Godula, K. *et al.* Control of the Molecular Orientation of Membrane-Anchored Biomimetic Glycopolymers. *J. Am. Chem. Soc.* **131**, 10263–10268 (2009).
33. Godula, K., Rabuka, D., Nam, K. T. & Bertozzi, C. R. Synthesis and Microcontact Printing of Dual End-Functionalized Mucin-like Glycopolymers for Microarray Applications. *Angew. Chem. Int. Ed.* **48**, 4973–4976 (2009).
34. Woods, E. C., Yee, N. A., Shen, J. & Bertozzi, C. R. Glycocalyx Engineering with a Recycling Glycopolymer that Increases Cell Survival In Vivo. *Angew. Chem. Int. Ed Engl.* **54**, 15782–15788 (2015).
35. Huang, M. L. *et al.* Determination of receptor specificities for whole influenza viruses using multivalent glycan arrays. *Chem. Commun. Camb. Engl.* **51**, 5326–5329 (2015).
36. Gossen, M., Bender, G., Muller, G., al, et & Freundlieb, S. Transcriptional activation by tetracyclines in mammalian cells. *Science* **268**, 1766 (1995).
37. Ma, S., Tang, N. & Tian, J. DNA synthesis, assembly and applications in synthetic biology. *Curr. Opin. Chem. Biol.* **16**, 260–267 (2012).
38. Hughes, R. A., Miklos, A. E. & Ellington, A. D. Gene synthesis: methods and applications. *Methods Enzymol.* **498**, 277–309 (2011).
39. Hollingsworth, M. A. & Swanson, B. J. Mucins in cancer: protection and control of the cell surface. *Nat. Rev. Cancer* **4**, 45–60 (2004).
40. Hatrup, C. L. & Gendler, S. J. Structure and function of the cell surface (tethered) mucins. *Annu. Rev. Physiol.* **70**, 431–457 (2008).

41. Wilson, M. H., Coates, C. J. & George, A. L. PiggyBac transposon-mediated gene transfer in human cells. *Mol. Ther. J. Am. Soc. Gene Ther.* **15**, 139–145 (2007).
42. Li, X. *et al.* piggyBac transposase tools for genome engineering. *Proc. Natl. Acad. Sci. U. S. A.* **110**, E2279–2287 (2013).
43. Woodard, L. E. & Wilson, M. H. piggyBac-ing models and new therapeutic strategies. *Trends Biotechnol.* **33**, 525–533 (2015).
44. Kramer, J. R., Onoa, B., Bustamante, C. & Bertozzi, C. R. Chemically tunable mucin chimeras assembled on living cells. *Proc. Natl. Acad. Sci. U. S. A.* **112**, 12574–12579 (2015).
45. Nielsen, J. S. & McNagny, K. M. The Role of Podocalyxin in Health and Disease. *J. Am. Soc. Nephrol.* **20**, 1669–1676 (2009).
46. Mercanti, V. *et al.* Transmembrane domains control exclusion of membrane proteins from clathrin-coated pits. *J. Cell Sci.* **123**, 3329–3335 (2010).
47. Paszek, M. J., DuFort, Christopher C, Rubashkin, Matthew G, Davidson, Michael W, Thorn, Kurt S, Liphardt, Jan T, Weaver, Valerie M,. Scanning angle interference microscopy reveals cell dynamics at the nanoscale. *Nat Meth Nat. Methods* **9**, 825–827 (2012).
48. Carraway, K. L., Ramsauer, V. P., Haq, B. & Carothers Carraway, C. A. Cell signaling through membrane mucins. *BioEssays News Rev. Mol. Cell. Dev. Biol.* **25**, 66–71 (2003).
49. Engelmann, K. *et al.* Transmembrane and secreted MUC1 probes show trafficking-dependent changes in O-glycan core profiles. *Glycobiology* **15**, 1111–1124 (2005).
50. Litvinov, S. V. & Hilkens, J. The epithelial sialomucin, episialin, is sialylated during recycling. *J. Biol. Chem.* **268**, 21364–21371 (1993).
51. Parry, S. *et al.* Identification of MUC1 proteolytic cleavage sites in vivo. *Biochem. Biophys. Res. Commun.* **283**, 715–720 (2001).
52. Yusa, K., Zhou, L., Li, M. A., Bradley, A. & Craig, N. L. A hyperactive piggyBac transposase for mammalian applications. *Proc. Natl. Acad. Sci. U. S. A.* **108**, 1531–1536 (2011).
53. Campeau, E. *et al.* A versatile viral system for expression and depletion of proteins in mammalian cells. *PloS One* **4**, e6529 (2009).
54. Schindelin, J. *et al.* Fiji: an open-source platform for biological-image analysis.

*Nat. Methods* **9**, 676–682 (2012).

55. Edelstein, A. D. *et al.* Advanced methods of microscope control using  $\mu$ Manager software. *J. Biol. Methods* **1**, 10 (2014).

# CHAPTER 3<sup>§</sup>

## High-speed device synchronization in optical microscopy with an open-source hardware control platform

### 3.1 Abstract

Azimuthal beam scanning eliminates the uneven excitation field arising from laser interference in through-objective total internal reflection fluorescence (TIRF) microscopy. The same principle can be applied to scanning angle interference microscopy (SAIM), where precision control of the scanned laser beam presents unique technical challenges for the builders of custom azimuthal scanning microscopes. Accurate synchronization between the instrument computer, beam scanning system and excitation source is required to collect high quality data and minimize sample damage in SAIM acquisitions. Drawing inspiration from open-source prototyping systems, like the Arduino microcontroller boards, we developed a new instrument control platform to be affordable, easily programmed, and broadly useful, but with integrated, precision analog circuitry and optimized firmware routines tailored to advanced microscopy. We show how the integration of waveform generation, multiplexed analog outputs, and native hardware triggers into a single central hub provides a versatile platform for performing fast circle-scanning

---

<sup>§</sup> **This chapter is reprinted with permission from:** Colville, M. J., Park, S., Zipfel, W. R., & Paszek, M. J. High-speed device synchronization in optical microscopy with an open-source hardware control platform. *Scientific Reports* (2019). doi:10.1038/s41598-019-48455-z

acquisitions, including azimuthal scanning SAIM and multiangle TIRF. We also demonstrate how the low communication latency of our hardware platform can reduce image intensity and reconstruction artifacts arising from synchronization errors produced by software control. Our complete platform, including hardware design, firmware, API, and software, is available online for community-based development and collaboration.

### **3.2 Introduction**

Modern imaging applications typically require the cooperative action of several independent devices, such as stages, filters, excitation modulators, and many others. Control of peripheral devices directly from the instrument computer requires independent connections, communication protocols and manufacturer-provided drivers and application programming interfaces (APIs). Integration of several peripherals in this way is technically challenging, and while software packages exist from both commercial and open-source<sup>1</sup> suppliers, they each have a limited set of supported devices. Moreover, the latency introduced by serial communications, the operating system (OS), application overhead and suboptimal API implementations can have a significant effect on synchronization accuracy. Because of these limitations, the time taken to update the experimental parameters across multiple peripherals can be much greater than the individual devices' response times, limiting the effective acquisition speed and increasing the risk of synchronization errors that could cause artifacts in the collected data.

To overcome these challenges, hardware controllers centralize device control within a single unit, typically using digital triggers or analog voltages to step peripheral devices through a series of predefined states. Hardware controllers can decrease communications overhead and implementation complexity by eliminating serial communications during an experiment. For devices that support digital triggering or analog control, the update time is reduced to the controller's response time, eliminating the additional delay of serial data transfer and decoding. A

controller with sufficient speed and memory can have the entire acquisition sequence pre-programmed, allowing autonomous operation from the instrument computer, eliminating communication overhead and processing time delays at runtime.

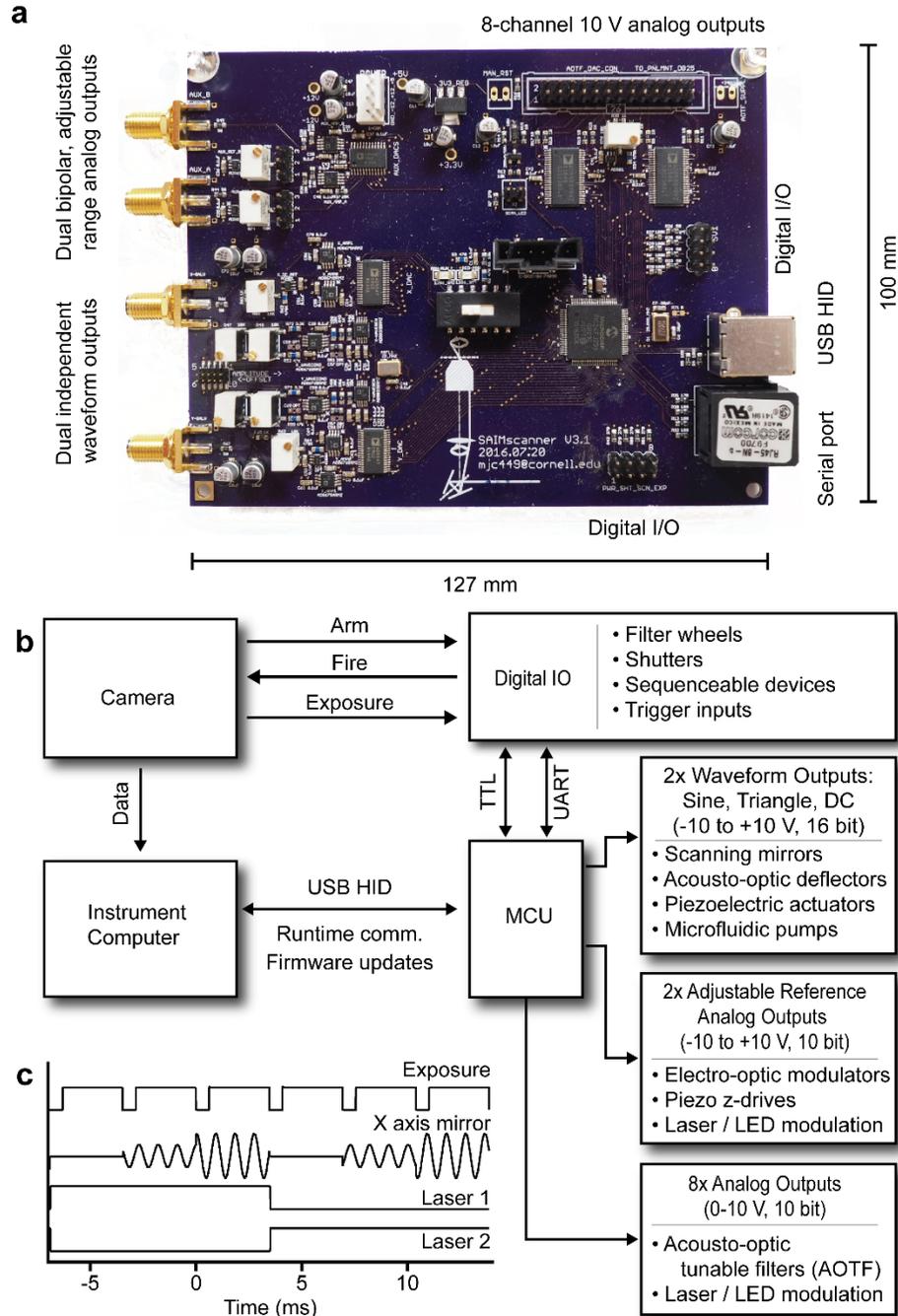
Universal microcontroller breakout boards, such as the Arduino devices<sup>2</sup>, have provided individuals and communities with a flexible platform to prototype and create interactive hobbyist-grade electronic devices. However, these controller platforms lack the appropriate I/O and optimizations to control many of the high precision, scientific-grade peripherals in state-of-the-art microscopes. General-purpose PC bus and USB data acquisition (DAQ) cards are popular alternatives that offer more powerful general-purpose I/O and system integration. However, they suffer the drawbacks of implementation complexity at hardware and software levels<sup>3,4</sup>. While the underlying controller or DAQ may be general purpose, it is implemented and developed in a single, highly specialized application. Because of this the completed systems generally lack portability, limiting transferability and community support.

Here we describe an open-source instrument-control platform that is designed with flexibility, ease of adoption, performance, and cost in mind. Throughout the development of our device, we have committed to three major design principles: 1) the device should be hardware independent and support a wide range of peripherals, 2) the device should not limit data acquisition rate and 3) the device should be an open-source community resource. While the instrument controller is general purpose by design, specific I/O and routines have been included to provide a low-cost, high-performance solution for rapid peripheral device synchronization in advanced microscopes, including laser-scanning systems. We demonstrate the advantages of

hardware control in a custom azimuthal beam scanning microscope by characterization of the effects of excitation quality and timing accuracy in a pair of complementary axial localization techniques: scanning angle interference microscopy (SAIM)<sup>5,6</sup> and multiangle total internal reflection fluorescence microscopy (MA-TIRF)<sup>7</sup>.

### **3.3 Results**

#### **Controller design**



**Fig. 3.1 Hardware control integration facilitates the cooperation of a variety of peripheral devices. (a)** Completed PCB assembly with key I/O features and dimensions. **(b)** Conceptual system topology wherein the microcontroller (MCU) acts as an intermediary between the instrument computer and multiple peripherals. **(c)** Timing waveforms from a maximum framerate acquisition sequence. The camera's exposure acts as the timing signal to synchronize peripheral device

updates in a typical circle-scanning experiment. The experiment was defined as a series of three repeating scan radii (mirror waveform amplitude) for each of two excitation wavelengths (Lasers 1 and 2).

To effectively provide a centralized hub we integrated both analog and digital functions on a single printed circuit board (PCB) (Fig. 3.1a,b, Supplementary Fig. 1), simplifying connector wiring and providing protection against unwanted electrical interference. Our design further subdivided the analog circuitry into waveform generation (2x) by independent direct digital synthesizers (DDS), high-speed bipolar analog outputs (2x) with adjustable references, and a multiplexed 8-channel analog output (Fig. 3.1b, Supplementary Note 1), allowing us to optimize the design of each subcircuit for its intended task. We incorporated full-speed USB connectivity, 8 digital I/O lines and a serial communication port in the hardware design. Given the high component and trace density required while considering cost and availability, we opted for a 4-layer PCB design with minimum 6 mil trace width and separation. This design could be affordably sourced, unpopulated, from many custom PCB manufacturers (i.e. OSH Park, 3 boards \$197.90). Following the same reasoning we selected components for our design that were actively being produced and available from multiple suppliers (i.e. Digikey, complete component set \$270.90 per board) in package sizes that could be placed and soldered by hand without the need for specialized equipment. We then populated the PCBs using basic surface mount and through-hole soldering tools such as a hot air rework station, soldering iron and tweezers, commonly available in most electronics labs. The completed, populated PCBs could be built by hand in around 6 hours apiece (~\$200 and 2 additional hours

of labor were required to assemble the enclosure and internal connectors). The PCB layout incorporated a 0.1 inch 6-pin header for connection to an in-circuit serial programmer and debugger (ICD-U64, Custom Computer Services Inc., \$89), which facilitated rapid development of updates to the controller firmware.

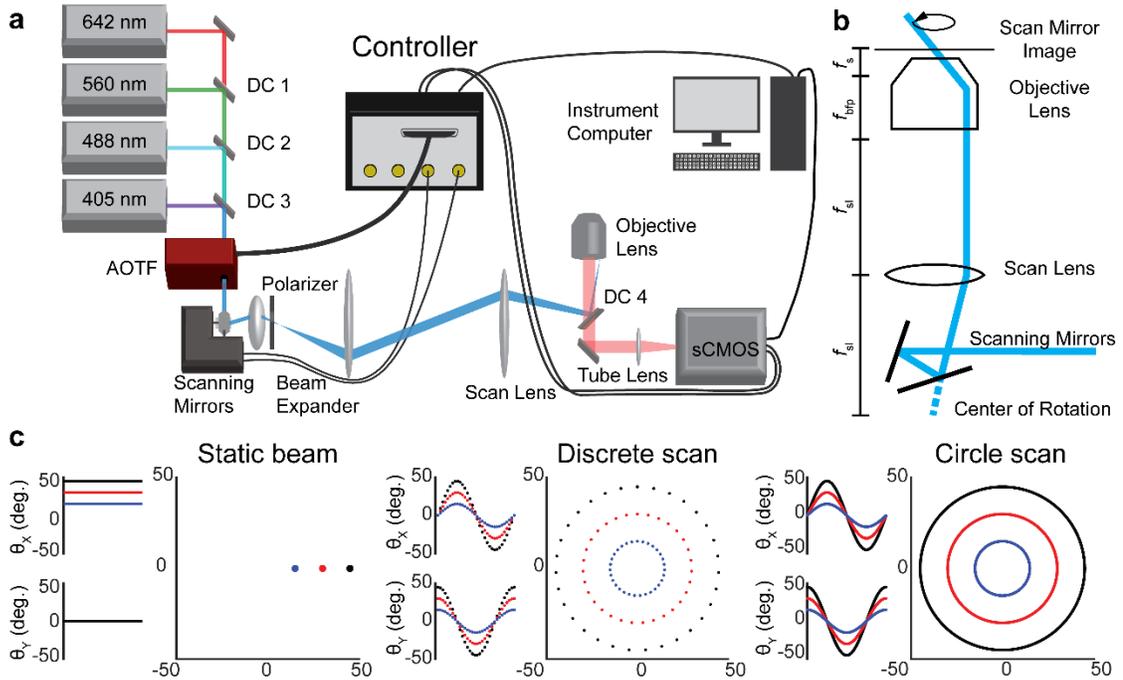
For communications between the computer and instrument controller we utilized the microcontroller's native full-speed universal serial bus (USB) human interface device (HID) protocol. This protocol was supported on all major computer operating systems (OS) and eliminated the need for OS-specific USB drivers, making the controller "plug and play". A limitation of the full-speed USB HID protocol was the 64 kB/s bandwidth and 1 ms frame length. Even with higher frequency transfers in faster USB protocols (the high-speed USB standard operated on 0.125 ms microframes) there was no mechanism for synchronizing USB transfers with events coming from the peripheral devices such as the camera. The round-trip time for the controller to recognize a synchronization signal from a peripheral, notify the instrument computer via USB, wait for the computer to process the signal and respond to the controller with the appropriate action to take was too long for accurate synchronization. We addressed this issue by selecting a 16-bit PIC® microcontroller with 256 kB program memory and 96 kB RAM in our design, allowing us to load complex sequences of experimental parameters in the controller memory. In this way peripheral updates and simple processing tasks were done without the need for communication between the controller and the instrument computer, circumventing the synchronization limitations of the USB protocol.

The controller firmware was written to prioritize minimum latency for the most

critical tasks such as shuttering. We built our hardware controller around a microcontroller to take advantage of its low cost, field reprogrammability, native interrupts, and chip-level integration of relevant functions, such as those for serial communications, counters, and parallel ports. One limitation of the 16-bit microcontroller used in our design was the relatively slow processor speed and low computational throughput. To address this, we offloaded calculations for acquisition sequences to the instrument computer. Prior to data acquisition the experimental parameters were uploaded to the controller, which ran as a finite-state machine during the experiment. The controller stepped through the experimental sequence using the microcontroller's native interrupts triggered by the camera exposure, device triggers or internal timers (Fig. 3.1c). Furthermore, we prioritized the allocation of parallel ports and internal serial communication hardware to minimize processor overhead wherever possible.

We found that for a typical experimental step, including waveform amplitude change, excitation shuttering and modulating 2 lasers, the total update time was on the order of 50  $\mu$ s. For reference, the vertical shift time of our electron-multiplying CCD camera (EMCCD; Andor iXon 888 Ultra) was approximately 600  $\mu$ s and the readout time of our scientific CMOS (sCMOS; Andor Zyla 4.2) was approximately 3.9 ms (Supplementary Fig. 2). Thus, the controller could synchronize several peripheral devices far faster than the dead time between exposures on a state-of-the-art microscope and was capable of handling complex, experimental sequences without limiting speed.

## Azimuthal beam scanning



**Fig. 3.2 Circle scanning microscope design with hardware control. (a)**

Schematic illustration of the key components of the circle scanning microscope. DC1-DC3: Laser combining dichroic mirrors. AOTF: Acousto-optic tunable filter. Polarizer:  $m = 1$  vortex half-wave retarder. DC4: Quad bandpass dichroic. Additional components omitted for clarity. (b) Conceptual diagram of the circle scanning optics.  $f_{sl}$ : Scan lens focal length.  $f_{bfp}$ : Objective lens rear focal length.  $f_s$ : Objective lens front focal length. (c) Scan mirror command signals and laser pattern on the objective rear focal plane. Static beam: the laser parked at a single location on the objective back focal plane by applying a constant voltage to the galvanometer-mounted mirrors. Discrete scan: the mirrors drive the beam through 32 points approximating a circle in the objective back focal plane. Circle scan: the mirrors are driven with sinusoidal voltages scanning the beam in a continuous circle.

Speckles and fringes caused by imperfections in the optical train, such as dust particles or internal reflections, are inherent to laser illumination<sup>8</sup>. The uneven excitation profile caused by these coherence artifacts is particularly noticeable in TIRF excitation and other widefield imaging techniques. While a variety of technologies

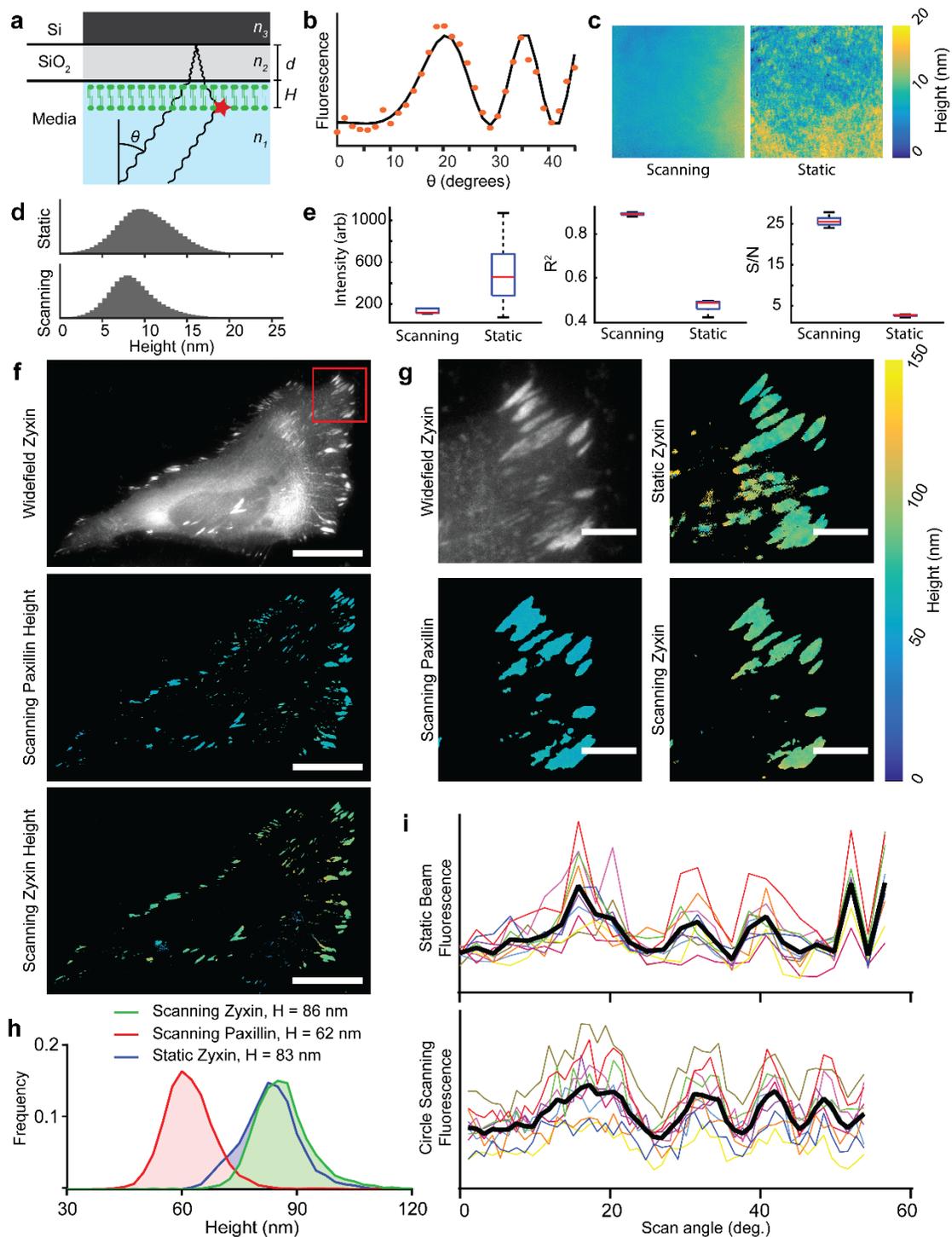
have been developed to reduce or eliminate these artifacts<sup>9-11</sup>, azimuthal beam scanning, or “circle scanning,” has become an attractive solution to reduce the effects of interference fringes and speckle artifacts in fluorescence microscopy<sup>12-14</sup>. Therefore, we incorporated several laser-scanning routines into our controller and demonstrated how a super-resolution azimuthal beam scanning microscope could be built around our open-source control platform (Fig. 3.2a).

In our microscope, the controller formed a central hub, synchronizing the cooperative actions of the laser-scanning mirrors, acousto-optic tunable filter (AOTF), sCMOS camera and other devices (mechanical shutter, EMCCD etc., Fig. 3.2a). The scanning mirrors’ angular deflections were controlled by the dual waveform outputs of our controller. The scanning mirrors’ apparent center of rotation was positioned in the objective’s conjugate image plane so that the excitation beam location remained static in the sample plane regardless of the azimuthal or axial angle (Fig. 3.2b). This configuration of the optical system produced a homogeneous excitation field by scanning many azimuthal angles within a single camera frame, with the effective profile being the average across the azimuthal angles in each exposure. The sCMOS fire-all and EMCCD exposure outputs were both connected to one of the controller’s digital I/O ports, which was configured as an external interrupt for acquisition synchronization. Using the cameras’ exposure signal allowed precise shuttering by the AOTF, only exciting the sample during the sensor active time. Additionally, peripheral updates, such as changing the scan radius or excitation wavelength, were begun immediately at the end of an exposure, ensuring the system had enough settling time before the beginning of the next exposure without adding additional delays

between frames (Supplementary Fig. 2, Supplementary Note 2).

In an azimuthal scanning microscope, the excitation laser must be steered about the optical axis such that its position describes a circle in the objective lens' back focal plane. This can be accomplished by a variety of beam steering methods including crossed acousto-optic deflectors<sup>15</sup>, deformable micromirror devices<sup>7</sup>, and motorized mirrors<sup>13,16</sup>. We noted that circular scans could be approximated with a discrete set of points (discrete scan, Fig. 3.2c and Supplementary Fig. 3, Supplementary Note 2) lying along a circle. However, for mechanical beam steering mirrors with finite maximum accelerations, we found that our galvanometer-driven mirrors were unable to follow the command signal perfectly. When we used a 32-point circle approximation, the mirror momentum resulted in oscillations about each point, decreasing the precision with which the axial incidence angle at the sample could be controlled (Supplementary Fig. 3).

As an alternative strategy, we used the integrated waveform generators of our control platform to drive the mirrors with a pair of sine waves with a  $\pi/2$  phase offset (circle scan, Fig. 3.2c and Supplementary Fig. 3). With this approach the mirrors remained settled on the circle and in constant motion, which eliminated the noise associated with starting and stopping at each point in a discrete scan (Supplementary Fig. 3).



**Fig. 3.3 Azimuthal beam scanning reduces incidence angle dependent laser artifacts in scanning angle interference microscopy (SAIM) imaging.** (a) Schematic illustration of SAIM. Samples were prepared on reflective silicon substrates with a defined oxide layer of thickness  $d$  and excited with a laser at incidence angle  $\theta$  to generate an axial

interference pattern with known spatial frequency. The measured fluorescence intensity at distance from the substrate  $H$  varies with  $\theta$  for a supported lipid bilayer (SLB) labeled with DiI. **(b)** Example single-pixel raw data (orange dots) and best-fit curve. **(c)** Reconstructed SLB height maps with circle scanning or static beam excitation schemes (full image  $147.9 \times 147.9 \mu\text{m}$ ). **(d)** Pooled DiI height distributions in SLBs acquired with static beam and circle scanning excitation schemes ( $n = 5$  regions,  $4.2 \times 10^6$  pixels per region). **(e)** Quantification of the average fit parameters in circle scanning and static beam SAIM imaging of the 5 lipid bilayer regions in **d**. Red lines indicate mean, boxes the central quartiles, whiskers min and max. **(f)** Widefield and circle scanning SAIM height reconstructions of a live HeLa cell expressing the focal adhesion components mEmerald-Zyxin and mCherry-Paxillin, scale bars  $25 \mu\text{m}$ . **(g)** Enlarged view of the region boxed in **f**. The colored images represent pixel-by-pixel SAIM height reconstructions of mEmerald-Zyxin or mCherry-Paxillin as noted, scale bars  $5 \mu\text{m}$ . A static beam reconstruction of zyxin height is included for comparison. **(h)** Distribution of pixelwise height fits from adhesion complexes in **g**. Heights are the average of all non-zero fits. **(i)** Plots of fluorescence intensity as a function of scan angle for the static and circle scanning Zyxin images in **g**. Colored lines indicate the same 10 representative pixels from each image, black lines the average of all 10 pixels.

Utilizing the integrated waveform generators in circle scanning mode, we investigated the improvements that could be gained through circle scanning in SAIM experiments to minimize laser-induced interference artifacts. SAIM is a surface-generated interference localization microscopy technique that measures the pixelwise average axial position of fluorescent probes with better than 10 nm precision. SAIM samples were prepared on a reflective silicon wafer with a defined layer of thermal oxide to act as a transparent spacer (Fig. 3.3a). When illuminated with a coherent light source, the direct and reflected excitation created an axial interference pattern with a characteristic spatial frequency that depended on the axial angle of incidence<sup>17</sup>. A series of images at known axial incidence angles were acquired and fit pixelwise to

an optical model that predicted the observed fluorescence intensity fluctuations as a function of axial position (height) above the SiO<sub>2</sub>/water interface (Fig. 3.3b). Previous SAIM implementations utilized a static beam excitation scheme<sup>5</sup>, the simplest method to implement on a commercial microscope.

We hypothesized that circle scanning SAIM would remove the pixelwise, frame-to-frame variations in the excitation intensity arising from speckle and fringe artifacts in a static beam system that degrade the accuracy of axial position reconstructions. To illustrate the advantages of circle scanning over static beam excitation in SAIM experiments, we used our microscope to image supported lipid bilayers (SLBs) labeled with DiI using either a static beam or circle scanning excitation scheme. The bilayer topography from static beam reconstructions showed local topographical variations of several nanometers that were not present in circle scanning reconstructions (Fig. 3.3c). When multiple, independent regions of the samples were imaged, we found that the same topographical patterns emerged, indicating that the excitation artifacts caused erroneous height estimations in the analysis (Supplementary Fig. 4). Figure 3.3d shows the height distributions from the combined results of 5 bilayer regions with average heights of  $10.2 \pm 3.46$  nm and  $8.72 \pm 3.10$  nm for the static beam and circle scanning experiments, respectively. To verify that the samples were equivalent in terms of labeling density and bilayer continuity we examined the fluorescence intensity maps (Supplementary Fig. 4). Based on a visual inspection, the height reconstructions in both cases were independent of variations in label density or bilayer defects. The average of 5 height reconstructions showed the topographical variations seen in the static beam experiments were consistent across

multiple regions unlike the circle scanning experiments, indicating that they were systematic and did not reflect the sample topography (Supplementary Fig. 4). This led us to conclude that the topography observed in the static beam experiments was a consequence of laser excitation artifacts that are mitigated by circle scanning. The improvement in the quality of the raw data was reflected in the more consistent average intensity across image sets and higher average  $R^2$  value from the reconstructions (Fig. 3.3e). Finally, circle scanning resulted in a greater than 5-fold increase in the average signal to noise ratio (S/N) across the experiments, a direct consequence of eliminating the incidence angle dependent laser artifacts.

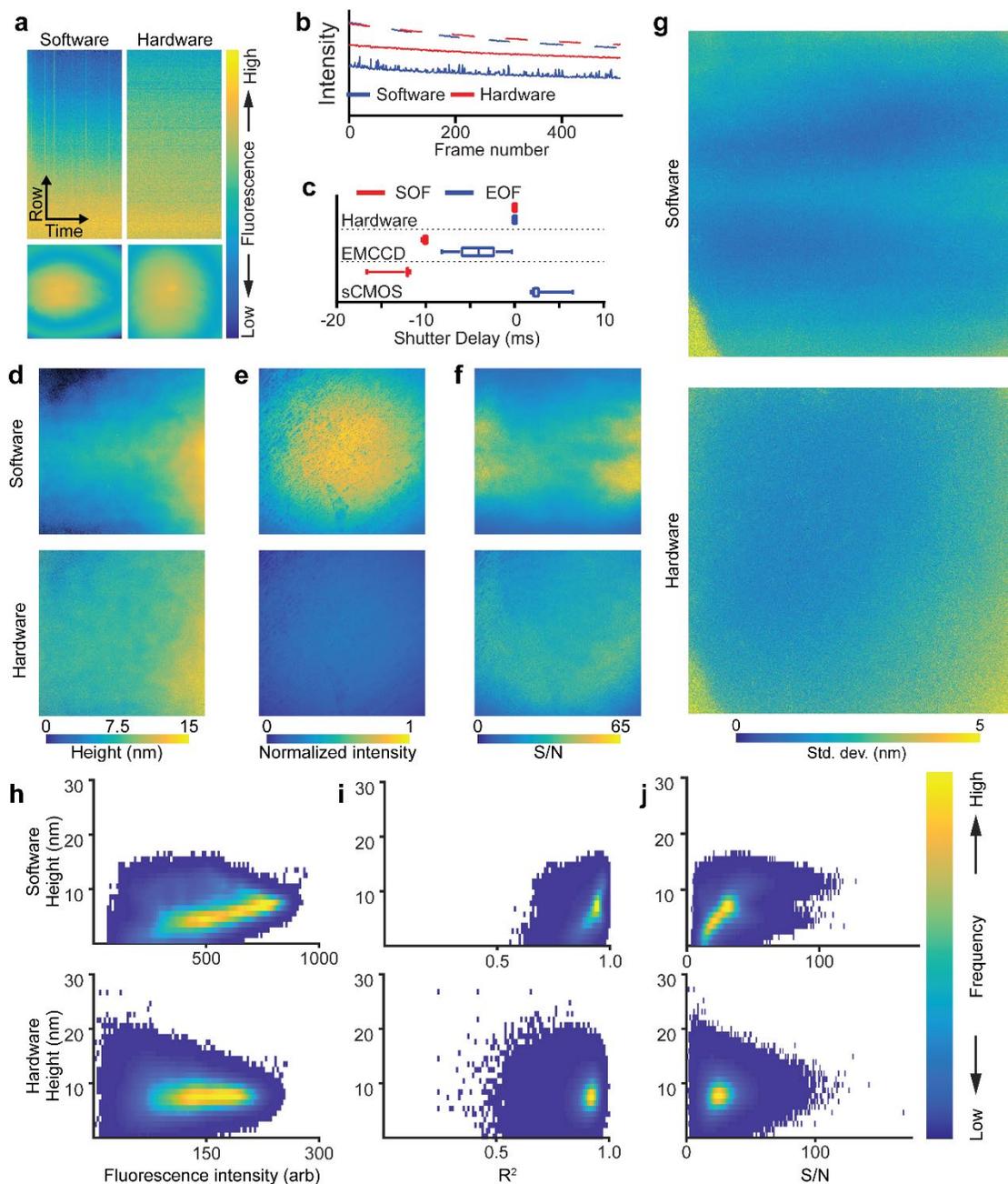
When imaging biological samples, such as live cells, high background fluorescence, local variations of the sample's optical properties caused by intracellular structures and dynamic processes conspire to disrupt the ideal axial interference pattern. While these properties cannot be eliminated as they are intrinsic to the sample, optimizing instrument performance could possibly mitigate their effects. To demonstrate the advantages of circle scanning in a biologically relevant context, we imaged live HeLa cells expressing the focal adhesion proteins mEmerald-zyxin and mCherry-paxillin (Fig. 3.3f). We noted that paxillin and zyxin could serve as axial measuring sticks as their height distributions are well known from several sources<sup>18-20</sup>. The widefield image in Fig. 3.3f highlights the background fluorescence in a typical live-cell sample. Because SAIM illuminated throughout the sample volume, the fluorescence from diffuse pools of free probes within the sample contributed to the observed intensity at each incidence angle. For bright, discrete structures, such as focal adhesions on the cell periphery, the intensity fluctuations of the background were

typically much smaller than the signal magnitude and robust localization was possible. With dim structures or in regions with no structure, fluctuations in the background, such as those arising from laser excitation artifacts, could be erroneously identified as real structures in the reconstruction. In the magnified view of the cell's leading edge (Fig. 3.3g), the widefield image showed several adhesion complexes on the periphery of the cell and a typical inhomogeneous background away from the cell's edge. The laser artifacts in the static beam case resulted in the artifactual appearance of structures that were not visible in the widefield image. The circle scanning reconstructions were free of these errors and better matched the focal adhesions from the widefield image. Both circle scanning and static beam reconstructions of zyxin showed similar distributions in height of  $86.4 \pm 8.17$  nm and  $83.1 \pm 7.69$  nm, respectively, as expected (Fig. 3.3h). Inspecting the pixelwise raw data demonstrated the improved S/N and reduction in outlier data points at high incidence angles where laser illumination artifacts are particularly problematic (Fig. 3.3i).

### **Accuracy of peripheral device synchronization in hardware versus software control**

Another potential source of error in SAIM and other techniques that make frame-to-frame intensity comparisons challenging is variation in the excitation dose. Excitation shuttering minimizes photobleaching and phototoxicity in living samples, reducing the total light dose to the minimum required for the experiment. However, the excitation shutter must be accurately synchronized with the camera exposure. Variations in shutter/camera synchronization between frames are reflected in the

observed fluorescence intensity, leading to localization artifacts. This effect is particularly problematic when using sCMOS cameras with rolling shutters where the beginning of exposure and readout are done by pixel rows<sup>21</sup>. Shuttering partway through the exposure/readout cycle results in the over exposure of some rows, while exposure throughout the exposure/readout cycle can lead to spatial distortion of fast-moving objects. sCMOS cameras can be run in a simulated global shutter mode wherein the excitation exposure is begun once all pixel rows are exposing and ended when the first row begins to readout (i.e. “fire-all” mode).



**Fig. 3.4 Hardware based synchronization minimizes latency-induced artifacts in SAIM.** (a) Top: fluorescence kymographs of the top half of the center pixel column of an sCMOS camera imaging a homogeneous dye monolayer. Bottom: Full sensor single frame images from the same acquisition sequence. (b) Plot of average row intensities (2048 pixels) from the series in a. Dashed lines represent the center pixel row, solid lines the top pixel row. (c) Distribution of excitation shutter delays at the start of frame (SOF) and end of frame (EOF) relative to the camera's exposure signal over 50 frames. Lines

represent the average of 50 frames, boxes the center quartiles and whiskers the minimum and maximum values. **(d-f)** Lipid bilayer height, fluorescence intensity and signal-to-noise ratio in SAIM reconstructions from image sets acquired with software or hardware synchronization. **(g)** Standard deviation from SAIM reconstructions of lipid bilayer height. Each of 6 equivalent regions of the sample were imaged first with hardware synchronization and then immediately after with software synchronization. The horizontal bands in the software image follow the center-out readout pattern of the sCMOS camera. **(h-j)** Lipid bilayer height distributions in SAIM reconstructions vs. fluorescence intensity, coefficient of determination and signal-to-noise ratio with the excitation under software (top row) or hardware (bottom row) control. Images in **a** and **d-g** reflect the full sensor area of 2048 x 2048 pixels, corresponding to a 147.9 x 147.9  $\mu\text{m}$  field of view. Data in **a** and **b** were normalized to the average intensity in the first exposure for comparison.

We considered that the roundtrip communication travel time from camera to computer to AOTF and the variable latency inherent in software control schemes could limit the precision of shutter synchronization. We collected 500 frames of a homogeneous fluorescent monolayer with either software control or hardware control to illustrate the effects of synchronization errors with a sCMOS camera (Fig. 3.4a). From the pixel column kymograph, with software control the upper pixel rows were relatively underexposed compared to the center pixel row. Under hardware control this effect was reduced, and a more even fluorescence intensity pattern was observed. Timing variability had the greatest effect on the last rows to be exposed or readout (those furthest from the center of the sensor in a center-out readout configuration, Fig. 3.4b). The center row (dashed lines in Fig. 3.4b) showed little difference between software and hardware control schemes while the top row (solid lines) had a lower average value and greatly increased noise under software control when compared to hardware control (Fig. 3.4a,b). To quantify the synchronization errors, we placed a

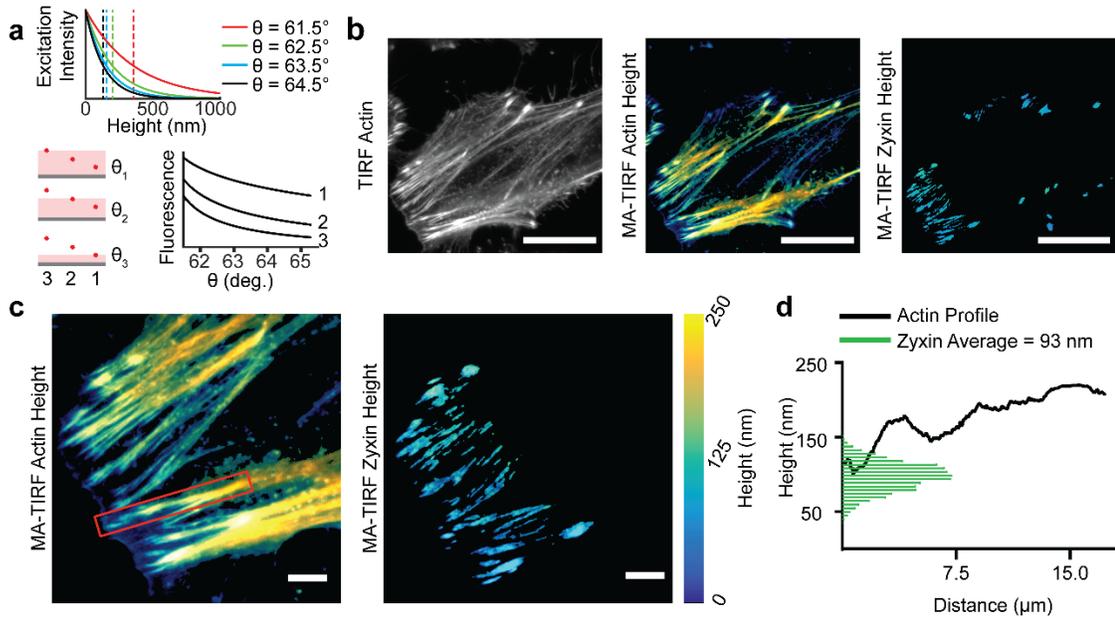
fast photomultiplier tube directly in the excitation path of our microscope (Fig. 3.4c, Supplementary Fig. 5). With software synchronization, we measured an average -  $12.34 \pm 0.9943$  ms and  $2.714 \pm 1.236$  ms delay between the beginning or end of exposure, respectively, and shutter operation. On the other hand, when the camera's fire-all signal was used to trigger the excitation shutter through our hardware controller, we achieved a 3 order of magnitude improvement in synchronization latency ( $12.73 \pm 12.80$   $\mu$ s and  $10.59 \pm 1.552$   $\mu$ s start and end of exposure, respectively, Fig. 3.4c, Supplementary Fig. 5). Although the hardware controller does introduce an additional delay in signal transmission, we found this to be on the order of 25% of the total time difference between the trigger signal and the effective shuttering time (Supplementary Note 3). With careful calibration the average delay under software control could be reduced. However, the variability due to communications and software overhead were intrinsic and could not be easily accounted for. It is important to note that a suboptimal or incorrect implementation of hardware control could introduce the same or even greater errors in synchronization than software control. Similarly, highly optimized software implementations could approach the minimum latency introduced by the device communication protocol but cannot overcome this fundamental limit.

Considering these results, we hypothesized that the increased precision of hardware synchronization would result in a higher S/N and better localization precision in SAIM. To make a direct comparison between software and hardware synchronization we imaged SLBs first with software synchronization and then, without moving the sample, with hardware synchronization at the same excitation

intensity and over the same range of angles, such that an identical observation of the sample was made. The height reconstructions of SLBs (Fig. 3.4d) acquired with software synchronization demonstrated a strong dependence of fitted height on distance from the center of the sensor which was not reflected in the hardware synchronization acquisitions. The fitted fluorescence intensity profile (Fig. 3.4e) was closely matched with both synchronization schemes. The hardware acquisition sequences had a lower average intensity, which was expected given the shorter overall excitation time (Fig. 3.4c). The S/N from the reconstructions with software synchronization showed a high S/N band in the region around the center of the detector that decays toward the top and bottom of the image compared to the hardware synchronization acquisition where the S/N closely followed the excitation intensity profile, as expected (Fig. 3.4f). Using the standard deviation in height over several independent SLB regions as a metric for localization precision, we found that the sensor readout pattern was reflected in the localization precision with software synchronization and not with hardware synchronization (Fig. 3.4g). Owing to the large sensor size in the sCMOS camera, the excitation profile was not perfectly flat across the field of view, which could make interpretation of the reconstructions difficult. To understand the effects of synchronization errors, we examined the height distributions as a function of intensity, coefficient of determination ( $R^2$ , a measure of goodness of fit) and S/N (Fig. 3.4h-j). In the case of software synchronization, the reconstructed heights showed a strong dependence on the fluorescence intensity with higher intensity corresponding to larger height values. A similar trend was observed for both  $R^2$  and S/N. With hardware synchronization, the reported height was

independent of intensity and similar distributions were seen for both  $R^2$  and S/N.

## MA-TIRF



**Fig. 3.5 Implementation of circle scanning multiangle TIRF (MA-TIRF).**

**(a)** Concept of MA-TIRF. Top: The evanescent excitation field intensity (solid lines) as a function of height above the glass-liquid interface and penetration depth (dashed lines) for various angles of incidence. Lower left: Schematic representation of fluorescent probes (red dots) at 300, 200 and 100 nm (3, 2 and 1, respectively) from the glass-liquid interface. Lower right: Theoretical intensity profiles as a function of angle of incidence for the 3 fluorescent probes. **(b)** Widefield and height reconstructions of phalloidin-AF560 and mEmerald-Zyxin in a fixed HeLa cell. Scale bars 25  $\mu\text{m}$ . **(c)** Enlarged view of the leading edge of the cell in **(b)**. Scale bars 5  $\mu\text{m}$ . **(d)** Reconstructed actin height profile (solid black line, red box in **(c)**) and zyxin height distribution (green bars, non-zero fits from **(c)**).

To demonstrate some of the versatility of our hardware control platform, we performed MA-TIRF imaging experiments utilizing the same hardware and sequencing functions used in SAIM experiments. Like SAIM, MA-TIRF achieves axial localization of fluorescent probes by reconstruction from multiple independent

images. When the excitation incidence angle at the coverslip to sample interface is beyond the critical angle,  $\theta_c$ , total internal reflection occurs, and the excitation potential is confined to a thin evanescent field within a few hundred nanometers of the interface. The exponential decay of the evanescent field depends on the angle of incidence and is characterized by the penetration depth, which decreases with angle of incidence<sup>22</sup>. In an MA-TIRF experiment the excitation incidence is swept through a range of angles greater than the critical angle and individual images are acquired at each step (Fig. 3.5a).

As a test subject, we imaged actin stress fibers, which are involved in adhesion-based motility and are required for maintenance of focal adhesion complexes. The ubiquity of actin fibers throughout cells complicates methods such as SAIM that lack optical sectioning. The thin evanescent excitation field in MA-TIRF rendered the ventral stress fibers clearly visible. We also imaged zyxin, which is known to interact with actin in the upper layers of focal adhesions where the mechanical coupling between the cytoskeleton and adhesion machinery occurs. Simultaneous imaging of actin and zyxin highlighted MA-TIRF's ability to resolve stress fiber profiles (Fig. 3.5b,c). Compared to SAIM, MA-TIRF returned a similar average zyxin height in adhesion complexes (93.3 nm) but a broader standard deviation (19.9 nm vs. 8.17 nm for circle scanning SAIM). Using MA-TIRF we were able to visualize the height profile along actin stress fibers (Fig. 3.5d) originating in adhesion zones as determined by zyxin enrichment. We found that in the zyxin-rich regions the stress fiber height matches the average zyxin height but increased with distance from the zyxin end. The ability to perform both SAIM and MA-TIRF

experiments on the same microscope platform demonstrated the ability of our hardware controller to adapt existing infrastructure for new applications.

### **3.4 Discussion**

High-speed synchronization of multiple devices has become a necessity in modern microscope systems and can be a complicated barrier for researchers who need to build or customize microscopes for specific imaging tasks. The two common approaches to peripheral integration are through software packages or dedicated hardware controllers. Software is typically the simplest path to system integration, but generally limits the available hardware pool to a list of supported devices. In some cases, such as with the open-source Micro-Manager<sup>1</sup> microscopy software, unsupported devices can be added by writing new device adapters. Doing so requires familiarity with the specific programming languages used, the device and software APIs and in some cases the operating system architecture. Furthermore, inefficient or improper implementations can limit performance and affect data quality. As we have shown in Fig. 3.4, even in the case of a stable and widely used software package, limitations imposed by the communication protocol's speed and intricacies can limit the synchronization precision of multiple devices. With the rapid progress of image processing technology, online and real-time analysis are being applied in many fields such as imaging flow cytometry<sup>23</sup>, presenting even greater challenges in device synchronization. While microcontrollers may lack the computational resources for high-throughput data processing, they can provide precise device synchronization by limiting the need for high-latency, non-deterministic communication between multiple

peripherals. For example, a single USB transaction between the host computer and a microcontroller can be reliably used to trigger multiple peripheral updates within a few microseconds of the controller receiving the transmission.

Hardware device synchronization can overcome many of the limitations imposed by software. We demonstrated this for the example case of camera and AOTF timing wherein the excitation is shuttered in concert with the camera's actual exposure. A naïve solution would be to avoid shuttering the excitation at the cost of increased excitation dose, photobleaching and phototoxicity. The greater the interframe interval the more pronounced each of these effects becomes, making it infeasible for time-course experiments with significant delays between frames. Under software control and with shuttering we observed timing differences of 10 or more milliseconds between the exposure and shuttering. If synchronization differences are consistent they can be accounted for in software by adding a delay between the slow and fast devices. In the case we have presented, latency between camera and AOTF events had variabilities on the order of milliseconds, and a simple delay would not suffice. Under hardware control we were able to achieve precise and consistent synchronization regardless of the framerate

We have used lipid bilayers to demonstrate that small variances in the timing between excitation and exposure can decrease the accuracy and precision of SAIM experiments. SLBs formed by fusion of small unilamellar vesicles make an ideal test sample with known properties (e.g. thickness, index of refraction, structure), are easily labeled with a variety of dyes and have extremely low background fluorescence. The SLB experiments illustrate the importance of frame-to-frame consistency in

maximizing the precision of SAIM localizations. Inconsistent illumination caused by interference at the sample (laser fringes and speckles) has the same effect as inconsistent timing on SAIM experiments. Although the causes are different, the result is that on a pixelwise basis, the instrument is causing additional intensity variations that obscure the predicted intensity variations in the raw data. In this study we chose not to use a set of background or correction images in any experiment. We found that the combined circle scanning and hardware synchronization approach eliminated the need for preprocessing the data with correction images.

Resolution of fast biological processes places additional constraints on the imaging system, and the effective time resolution is limited to the elapsed time to acquire the complete image series for a range of scan angles. Ultimately, this limit is set by the camera framerate plus any delay associated with changing the excitation angle or wavelength. In a typical circle scanning SAIM experiment we acquire between 16 and 32 images over the incidence angle series. The exact number of images is determined empirically at the time of the experiment based on balancing acquisition time with data quality. Our hardware controller allows acquisition at the camera framerate, that is there is no additional delay added and all devices are updated during readout. Under these conditions the effective time resolution is the number of frames per acquisition multiplied by the exposure period. For instruments with slower peripheral devices (i.e. motorized TIRF prisms), or under software control, an adequate delay between frames must be added while the system reaches the new stable point for the next camera frame. Motion of the sample between the first and last frames of a SAIM or MA-TIRF sequence will introduce additional errors in the

reconstruction because a pixel in the first frame is not mapped to the same sample location in subsequent frames in the angle series. It follows that the time between the first exposure and last exposure in a set of incidence angles should be less than the time it takes the sample to move by one pixel. This condition can be difficult to satisfy but hardware control maximizes the achievable acquisition rate, providing a more accurate multi-frame snapshot in SAIM and MA-TIRF experiments (Supplementary Fig. 6, Supplementary Videos 2,3).

The concept of a general-purpose scientific instrument controller is not unique to this project and several examples exist, some as commercial products. One open source instrumentation project like what we have reported is the Pulse Pal<sup>24</sup> signal generator (Sanworks, \$745 at the time of writing), which features 4 independent +/- 10 V waveform outputs and 2 trigger inputs. While Pulse Pal could be used for driving the scanning system, additional hardware would be required to integrate multichannel excitation control and shuttering. Another open-source example is the FlyPi, a multifunctional microscopy project (project costs range between ~\$100 and ~\$250 depending on the number of additional modules the user incorporates)<sup>25</sup>. The FlyPi utilizes a Raspberry Pi computer for data acquisition and experiment control and an external Arduino microcontroller to control the microscope peripherals such as stage, temperature, illumination and others. While similar in approach, the FlyPi is a complete microscope solution designed for accessibility whereas our instrument controller is designed to offer high-precision, high speed peripheral integration in advanced system development. Outside of dedicated microscopy systems, there are many examples, such as the general purpose instrument STEMLab<sup>TM</sup> hardware

platform (Red Pitaya, ~\$290 at the time of writing) which features 2 radio frequency (RF) inputs, 2 RF outputs, 16 digital I/O signals and 4 additional low speed analog outputs as well as significant computational resources due to its integrated FPGA and dual core CPU. The hardware is limited, though, to +/- 1 V for the RF outputs and 0-1.8 V for the analog outputs, requiring additional, custom amplification stages to interface with many devices. Furthermore, while the software is open source, the hardware is not. Many non-open source solutions are available such as the Triggerscope 16 (Advanced Research Consulting, \$1499 at the time of writing) which features 16 0-10 V analog outputs, 16 TTL outputs and sequencing capabilities of up to 60 steps. Other popular options are the DAQ solutions offered by National Instruments which range in cost from ~\$200 to more than \$6000, with cards providing 4 analog outputs, 48 digital I/Os and USB connectivity beginning at ~\$1100. While these boards arguably offer greater flexibility and standardization than our controller, they do not have the same level of microscopy-specific optimization or integration of design, requiring additional effort in development. For comparison, a researcher wishing to build a copy of our controller as reported here could do so for ~\$525 (we have included a bill of materials in the project repository including part numbers and suppliers) and 8 hours of labor with access to basic surface mount soldering and test equipment found in most electronics labs.

There are many forms of hardware control that have been used in microscopy from plugin extension PCBs for universal microcontroller platforms to commercial field programmable gate array boards to custom embedded electronics platforms covering a broad range of cost, complexity and flexibility. We have sought to create a

broadly useful platform that is economical and easy to integrate. The open-source model used by Micro-Manager has greatly contributed to the success of the project and we seek to emulate their example by making all design and source code freely available<sup>26,27</sup>. We have made all levels of the project from hardware to driver API and user interface software open-source. This will allow us to capitalize on the skills and feedback of users and create a community of contributors, distributing the cost of further development based on the framework presented here. Furthermore, while we have demonstrated the utility of our controller in a single, circle scanning instrument the control platform we have developed has a broad range of applications including laser-scanning confocal or two photon microscopies, imaging cytometry, microfluidic systems and spectroscopy to name a few. Through continuing, community-driven development of the project we envision our hardware as a valuable resource for general scientific device control.

### **3.5 Methods**

#### **Preparation of supported lipid bilayers**

A chloroform solution of 1-palmitoyl-2-oleoyl-glycero-3-phosphocholine with 0.1 mole percent 1,2-dipalmitoyl-sn-glycero-3-phosphoethanolamine-N-(cap biotinyl) (Avanti Polar Lipids) was dried first under a stream of nitrogen and then under vacuum to remove the solvent. The resulting film was dissolved to 1 mg/mL in PBS pH 7.4 at 37 °C with extensive vortexing and freeze-thawed 7x by cycling between liquid nitrogen and 37 °C baths. The lipid suspension was then extruded 13x through double stacked 100 nm polycarbonate membranes (Whatman) to yield small

unilamellar vesicles (SUVs). SUVs prepared in this way were stored at 4 °C and used within 2 weeks. Silicon wafers with ~1900 nm thermal oxide (Addison Engineering) were diced into 1 cm<sup>2</sup> chips, and the oxide layer thickness for each chip was measured with a FilMetrics F50-EXR. The chips were then cleaned in piranha solution (30 volume % hydrogen peroxide in sulfuric acid), rinsed extensively and stored in deionized water until use. SLBs were formed by incubating the cleaned silicon chips in the SUV solution for 1 hour, and then rinsed with PBS, incubated with 1.75 μM DiIC<sub>18</sub> for 30 minutes, and rinsed again with PBS. The samples were then inverted into a 35 mm glass-bottom imaging dish (MatTek) containing PBS and immediately imaged.

### **Preparation of HeLa cell samples**

The plasmids mEmerald-Zyxin-6 (Addgene plasmid # 54319; <http://n2t.net/addgene:54319>; RRID: Addgene\_54319) and mCherry-Paxillin-22 (Addgene plasmid # 55114; <http://n2t.net/addgene:55114>; RRID: Addgene\_55114) were a gift from Michael Davidson. HeLa cells (ATCC CCL-2) were cultured in Dulbecco's Modified Eagle Medium (ThermoFisher) supplemented with 10% fetal bovine serum (ThermoFisher) and 100 U/mL penicillin-streptomycin (ThermoFisher) at 37 °C and 5% CO<sub>2</sub>.

For SAIM experiments, silicon chips were prepared as for SLBs except the chips were sonicated for 15 mins in acetone, 15 minutes in 1 M NaOH and UV sterilized in place of the piranha cleaning. Cells were seeded onto the chips at 2x10<sup>4</sup> cm<sup>-2</sup> in full culture medium and incubated overnight. Cells were transfected with 1:1

mEmerald-Zyxin-6 to mCherry-Paxillin-22 using Lipofectamine 3000 (ThermoFisher) per the manufacturer's protocol and allowed to grow for 24 hours. Samples were then rinsed 3x in PBS, inverted into a 35 mm glass-bottom imaging dish containing HEPES-Tyrode's (119 mM NaCl, 5 mM KCl, 25 mM HEPES buffer, 2 mM CaCl<sub>2</sub>, 2 mM MgCl<sub>2</sub>, 6 g/L glucose, pH 7.4) and imaged at 37 °C.

For MA-TIRF experiments, cells were seeded at  $2 \times 10^4$  cm<sup>-2</sup> directly into 35 mm glass-bottom imaging dishes and incubated overnight to allow attachment. Samples were then transfected with mEmerald-Zyxin-6 as before and incubated an additional 24 hours. Following incubation, the samples were rinsed with PBS and fixed in 4% formaldehyde in PBS for 10 minutes, rinsed 3 additional times and permeabilized with 0.1% v/v Triton X-100 in PBS for 10 minutes. The solution was then aspirated, and the sample was incubated for 1 hour at room temperature with Alexa Fluor™ 568 Phalloidin (ThermoFisher) diluted to 1 U/mL in PBS with 1% bovine serum albumin (ThermoFisher). Samples were then rinsed 3 times in PBS and imaged immediately.

## **Controller Design**

Full details on the controller design are included in Supplementary Note 1 and 3. Briefly, the analog subcircuits were designed and simulated using CircuitLab<sup>28</sup>. The controller PCB layout was made using EAGLE (Autodesk). Printed circuit boards were ordered unpopulated from OSH Park. The components were purchased from Digi-Key and then hand assembled. Firmware was written in the C programming language, compiled and debugged with C-Aware IDE (Custom

Computer Service Inc.). Host-side driver and GUI software was written in C++ with Visual Studio 2017 (Microsoft) using the Qt<sup>29</sup> framework, HIDAPI<sup>30</sup>, boost C++<sup>31</sup>, and OpenCV libraries<sup>32</sup>.

## Microscope Design

The circle scanning microscope was built on a Nikon Ti-E inverted fluorescence microscope. The 405 nm (Power Technology Inc.), 488 nm (Coherent), and 560 nm (MPB Communications Inc.) lasers were combined with the 642 nm (MPB Communications Inc.) to make a colinear beam by a series of dichroic mirrors (Chroma). The combined output beam was attenuated and shuttered by an AOTF (AA Opto-Electronic) before being directed onto the galvanometer scanning mirrors (Cambridge Technology). The image of the laser on the scanning mirrors was magnified and relayed to the sample by 2  $4f$  lens systems, the beam expanding telescope and the scan lens / objective lens combination. The beam expander is formed by  $f = 30$  mm and  $f = 300$  mm achromatic lenses with an  $m = 1$  zero-order vortex half-wave plate positioned between them and positioned  $2f$  from the 300 mm achromatic scan lens (Thorlabs). SAIM experiments were performed with a 60x N.A. 1.27 water immersion objective, and MA-TIRF with a 60x N.A. 1.49 oil immersion objective lens (Nikon). Fluorescence emission was collected with a quad band filter cube and single band filters (TRF89901-EMv2, Chroma) mounted in a motorized filter wheel (Sutter). For both SAIM and MA-TIRF experiments, images were acquired with a Zyla 4.2 sCMOS (Andor) camera using the microscope's 1.5x magnifier for a total magnification of 90x. In all experiments the open source software Micro-

Manager<sup>1</sup> was used for camera and filter wheel control and image acquisition. For timing data with software control the AOTF driver hardware was disconnected from the hardware controller and connected to the instrument computer via USB. The AOTF was added to the micromanager configuration utilizing the AA AOTF device adapter written by Erwin Peterman and supplied with the software. A single AA AOTF device was added to the hardware configuration to avoid potential delays associated with using multiple channels. Details on excitation incidence angle calibration are given in Supplementary Note 4.

### **Timing Data acquisition**

Timing data acquired with the EMCCD camera was done with frame transfer and electron multiplying gain enabled. The sCMOS camera auxiliary output was configured for “fire all” mode. In both cases the camera fire output was connected to the hardware controller through a BNC “T” connector. The other side of the “T” was connected to a digital oscilloscope (MSO2014B, Tektronix) and single shot acquisitions were acquired triggered by the rising edge of the camera fire signal. A fast photomultiplier tube (Hamamatsu) was placed directly in the excitation path immediately after the AOTF and the amplifier output connected to the oscilloscope. Automatic shuttering was enabled in either Micro-Manager (software control) or via the hardware controller (hardware control) and single exposures were initiated with the “Snap” function.

### **SAIM**

For static beam SAIM experiments, a series of 31 images were acquired at evenly spaced incidence angles from -45.8 to 43.9 degrees. For circle scanning experiments, 32 images were acquired at evenly spaced intervals from 0 to 30 degrees in the case of live cells or 0 to 45 degrees in the case of SLBs. Angle ranges are given with respect to the wafer normal in the imaging media (Fig. 3.3a). The raw images were fit pixelwise without further preprocessing using a custom analysis program written by the authors (Supplementary Note 5) to obtain the reconstructed height topography of the samples. Images were postprocessed with Fiji<sup>33</sup> and statistical analysis was performed in MATLAB (MathWorks) and Prism (GraphPad).

### **MA-TIRF**

For MA-TIRF experiments, a series of 20 images were acquired from 61.5 degrees to 65.3 at intervals of 0.2 degrees with respect to the excitation angle of incidence in the coverslip at the glass to water interface. The galvanometer command amplitude corresponding to the critical angle was experimentally confirmed using the fluorescence intensity of a dye monolayer<sup>34</sup>. MA-TIRF reconstructions were done with MATLAB using the ADMM<sup>35</sup>. The resulting height images were postprocessed in Fiji and statistical analysis was performed in Prism.

### **Code availability**

All source code written by the authors including controller firmware, driver, graphical interface and SAIM analysis code as described here is available in the project repository at <https://github.com/mjc449/SAIMscannerV3.git> (commit

80819d26b2d570e266a7220d5379a96686e4b835). Additional source code and third-party libraries are available from the links referenced in the controller design section. Analysis code for the MA-TRIF experiments is available at <https://github.com/zcshinee/Pol-TIRF.git> (commit 5279d16bb0b9bcbdd1fe98836f08d99dfbb50d628).

### **3.6 Acknowledgements**

We thank Susan Daniel and Rohit Singh (Cornell University) for discussions and assistance preparing SLB samples. This research was funded by the National Institute of General Medical Sciences Ruth L. Kirschstein National Research Service Award 2T32GM008267 (M.J.C.), National Science Foundation Graduate Research Fellowship DGE-1650441 (M.J.C.), National Cancer Institute U54 CS210184 (M.J.P. and W.R.Z.) and R33-CA193043 (M.J.P. and W.R.Z.), National Science Foundation Award 1752226 (M.J.P.) and the Kavli Institute at Cornell for Nanoscale Science (M.J.C., W.R.Z. and M.J.P.). This work was performed in part at the Cornell NanoScale Science & Technology Facility (CNF), a member of the National Nanotechnology Coordinated Infrastructure (NNCI), which is supported by the National Science Foundation (Grant NNCI-1542081).

### **3.7 Author Contributions\*\***

M.J.C., W.R.Z. and M.J.P. conceived the project and devised the studies. M.J.C. designed the controller electronics and firmware based on previous work by

---

\*\* Modified from the original publication to include a more detailed attribution of credit.

W.R.Z. M.J.C. designed and integrated DDS waveform generation, 8-channel excitation control, serial I/O, dual auxiliary analog outputs and expanded digital I/O from the original hardware design. M.J.C. performed circuit simulations, designed and built the controller hardware. M.J.C. designed and constructed the optical system, including component layout and fabrication of custom mechanical and electronic components. M.J.C. wrote the driver, interface and analysis software. M.J.C. and S.P. generated SLB samples, performed SLB experiments and collected instrument performance data. M.J.C. generated samples for both live-cell SAIM and fixed-cell MA-TIRF experiments and collected data. M.J.C. analyzed imaging data, performed statistical analysis and prepared figures. M.J.C. wrote the manuscript with input from all authors.

## REFERENCES

1. Edelstein, A., Amodaj, N., Hoover, K., Vale, R. & Stuurman, N. Computer Control of Microscopes Using  $\mu$ Manager. in *Current Protocols in Molecular Biology* (eds. Ausubel, F. M. et al.) (John Wiley & Sons, Inc., 2010).
2. OpenLabTools. Available at: <http://openlabtools.eng.cam.ac.uk/Instruments/Microscope/>. (Accessed: 18th December 2018)
3. Planchon, T. A. *et al.* Rapid three-dimensional isotropic imaging of living cells using Bessel beam plane illumination. *Nature Methods* **8**, 417–423 (2011).
4. York, A. G. *et al.* Instant super-resolution imaging in live cells and embryos via analog image processing. *Nature Methods* **10**, 1122–1126 (2013).
5. Paszek, M. J. *et al.* Scanning angle interference microscopy reveals cell dynamics at the nanoscale. *Nature Methods* **9**, 825–827 (2012).
6. Carbone, C. B., Vale, R. D. & Stuurman, N. An acquisition and analysis pipeline for scanning angle interference microscopy. *Nature Methods* **13**, 897–898 (2016).
7. Zong, W. *et al.* Shadowless-illuminated variable-angle TIRF (siva-TIRF) microscopy for the observation of spatial-temporal dynamics in live cells. *Biomedical Optics Express* **5**, 1530 (2014).
8. Goodman, J. W. *Speckle Phenomena in Optics: Theory and Applications*. (Roberts and Company Publishers, 2007).
9. Ma, H., Fu, R., Xu, J. & Liu, Y. A simple and cost-effective setup for super-resolution localization microscopy. *Scientific Reports* **7**, 1542 (2017).
10. Saloma, C., Kawata, S. & Minami, S. Large-amplitude current modulation of semiconductor laser and multichannel optical imaging. *Optics Communications* **93**, 4–10 (1992).
11. Dingel, B. & Kawata, S. Speckle-free image in a laser-diode microscope by using the optical feedback effect. *Opt. Lett., OL* **18**, 549–551 (1993).
12. Mattheyses, A. L., Shaw, K. & Axelrod, D. Effective elimination of laser interference fringing in fluorescence microscopy by spinning azimuthal incidence angle. *Microscopy Research and Technique* **69**, 642–647 (2006).
13. Johnson, D. S., Toledo-Crow, R., Mattheyses, A. L. & Simon, S. M. Polarization-Controlled TIRFM with Focal Drift and Spatial Field Intensity Correction. *Biophysical Journal* **106**, 1008–1019 (2014).

14. Fiolka, R., Belyaev, Y., Ewers, H. & Stemmer, A. Even illumination in total internal reflection fluorescence microscopy using laser light. *Microscopy Research and Technique* **71**, 45–50 (2008).
15. van 't Hoff, M., de Sars, V. & Oheim, M. A programmable light engine for quantitative single molecule TIRF and HILO imaging. *Optics Express* **16**, 18495 (2008).
16. Fu, Y. *et al.* Axial superresolution via multiangle TIRF microscopy with sequential imaging and photobleaching. *Proceedings of the National Academy of Sciences* **113**, 4368–4373 (2016).
17. Lambacher, A. & Fromherz, P. Fluorescence interference-contrast microscopy on oxidized silicon using a monomolecular dye layer. *Applied Physics A: Materials Science & Processing* **63**, 207–217 (1996).
18. Kanchanawong, P. *et al.* Nanoscale architecture of integrin-based cell adhesions. *Nature* **468**, 580–584 (2010).
19. Liu, J. *et al.* Talin determines the nanoscale architecture of focal adhesions. *Proceedings of the National Academy of Sciences* **112**, E4864–E4873 (2015).
20. Paszek, M. J. *et al.* Scanning angle interference microscopy reveals cell dynamics at the nanoscale. *Nature Methods* **9**, 825–827 (2012).
21. Liang, C., Chang, L. & Chen, H. H. Analysis and Compensation of Rolling Shutter Effect. *IEEE Transactions on Image Processing* **17**, 1323–1330 (2008).
22. Martin-Fernandez, M. L., Tynan, C. J. & Webb, S. E. D. A ‘pocket guide’ to total internal reflection fluorescence: A ‘POCKET GUIDE’ TO TOTAL INTERNAL REFLECTION FLUORESCENCE. *Journal of Microscopy* **252**, 16–22 (2013).
23. Heo, Y. J., Lee, D., Kang, J., Lee, K. & Chung, W. K. Real-time Image Processing for Microscopy-based Label-free Imaging Flow Cytometry in a Microfluidic Chip. *Scientific Reports* **7**, (2017).
24. Sanders, J. I. & Kepecs, A. A low-cost programmable pulse generator for physiology and behavior. *Front. Neuroeng.* **7**, (2014).
25. Maia Chagas, A., Prieto-Godino, L. L., Arrenberg, A. B. & Baden, T. The €100 lab: A 3D-printable open-source platform for fluorescence microscopy, optogenetics, and accurate temperature control during behaviour of zebrafish, *Drosophila*, and *Caenorhabditis elegans*. *PLOS Biology* **15**, e2002702 (2017).
26. mjc449. SAIMscannerV3. (2018). Available at:

- <https://github.com/mjc449/SAIMscannerV3>. (Accessed: 14th December 2018)
27. mjc449/SSv3\_1 · CircuitHub. Available at: [https://circuitlab.com/projects/mjc449/SSv3\\_1/revisions/16275](https://circuitlab.com/projects/mjc449/SSv3_1/revisions/16275). (Accessed: 14th December 2018)
  28. Online circuit simulator & schematic editor. *CircuitLab* Available at: <https://www.circuitlab.com/>. (Accessed: 16th December 2018)
  29. Qt Wiki. Available at: <https://wiki.qt.io/Main>. (Accessed: 16th December 2018)
  30. Ott, A. *A Simple library for communicating with USB and Bluetooth HID devices on Linux, Mac, and Windows.: signal11/hidapi*. (2018).
  31. Boost C++ Libraries. Available at: <https://www.boost.org/>. (Accessed: 16th December 2018)
  32. OpenCV library. Available at: <https://opencv.org/>. (Accessed: 16th December 2018)
  33. Schindelin, J. *et al.* Fiji: an open-source platform for biological-image analysis. *Nature Methods* **9**, 676–682 (2012).
  34. Fu, Y. *et al.* Axial superresolution via multiangle TIRF microscopy with sequential imaging and photobleaching. *Proceedings of the National Academy of Sciences* **113**, 4368–4373 (2016).
  35. Zheng, C. *et al.* Three-dimensional super-resolved live cell imaging through polarized multi-angle TIRF. *Optics Letters* **43**, 1423 (2018).

## CHAPTER 4<sup>††</sup>

### **Azimuthal beam scanning microscope design and implementation for axial localization with scanning angle interference microscopy**

#### **4.1 Abstract**

Azimuthal beam scanning, commonly called circle-scanning, is an effective way of eliminating coherence artifacts with laser illumination. With a static excitation spot, imperfections in the optical system, such as diffractive contaminants and internal reflections, conspire to produce an uneven excitation field. These artifacts become more pronounced in TIRF microscopy, where the excitation is confined to a narrow, evanescent field within a few hundred nanometers of the coverslip. The patterns that arise from these imperfections depend on the path of the excitation beam through the microscope optical train. By rapidly rotating the beam through its azimuth the uneven illumination is canceled by signal averaging over the camera exposure time. Scanning angle interference microscopy (SAIM) is a surfaced-generated axial localization technique with nanometer-scale precision that requires similar instrumentation to TIRF microscopy. For robust SAIM localization laser excitation with a homogeneous profile over a range of polar angles is required. We have applied the circle-scanning principle to SAIM, constructing an optimized instrument configuration and custom,

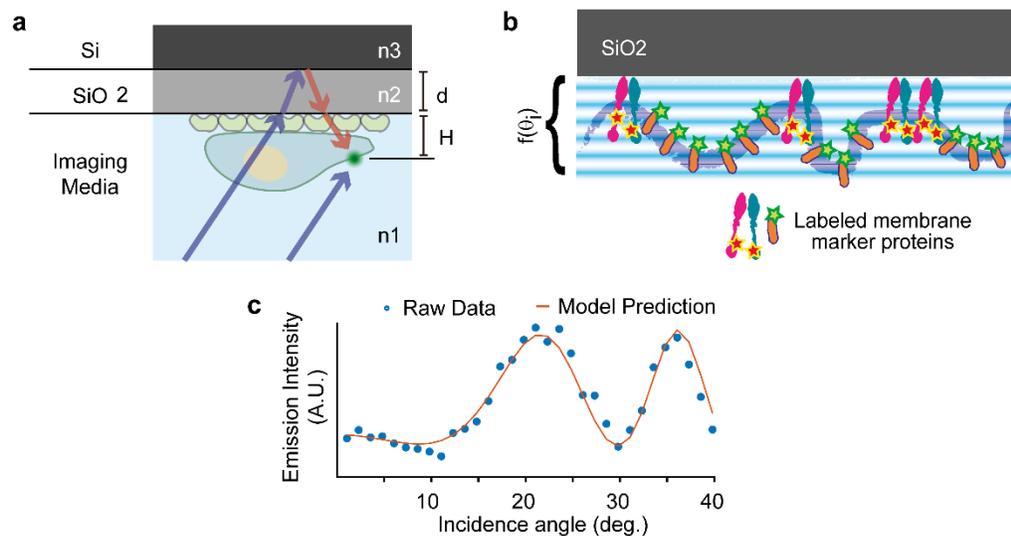
---

<sup>††</sup> **This chapter is an authors' invited manuscript for publication as:** Colville, M., Park, S., Singh, A., Paszek, M., & Zipfel, W. Azimuthal beam scanning microscope design and implementation for axial localization with scanning angle interference microscopy. in *Biosensors and Biodetection. Methods in Molecular Biology*

open-source hardware, enabling high precision localization and significantly higher temporal resolution than previous implementations. In this chapter we detail the design and construction of the SAIM instrument, including the optical configuration, required peripheral devices, and system calibration.

## 4.2 Introduction

Fluorescence microscopy is a powerful tool for use in structural and functional studies of biological specimens. The characteristic excitation and emission spectra of different fluorescent probes can be easily separated in microscope optics, allowing for visualization of multiple, independent components within a sample. This ability to perform multiplexed spatial investigation has yielded significant insight into subcellular structural organization. However, widefield fluorescence microscopy lacks axial resolution. In a widefield microscope the entire sample volume is excited and fluorescence is collected from throughout the observable depth of the sample.

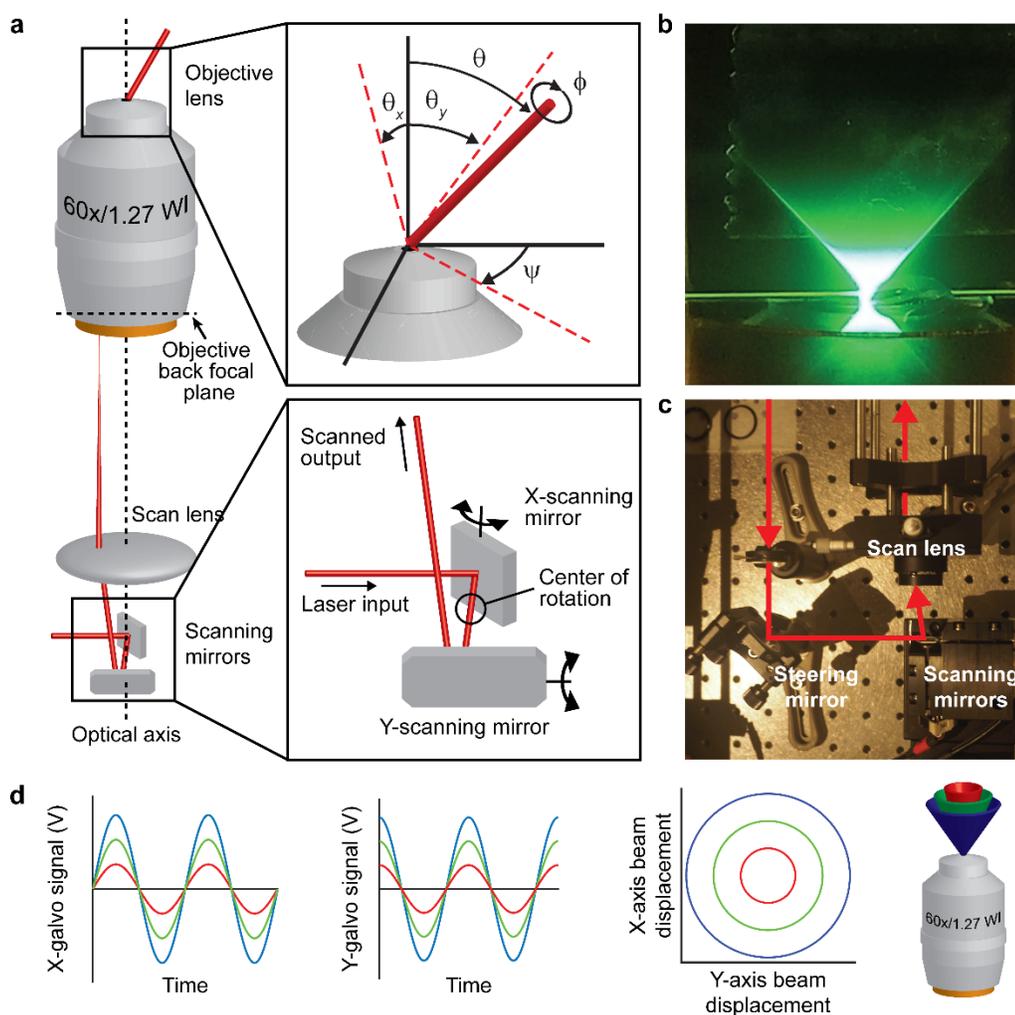


**Fig. 4.1 Surface generated interference and localization in SAIM.** (a) Samples are prepared on a silicon substrate with a defined layer of thermal oxide. (b) Standing wave excitation pattern that changes as a function of the angle of incidence (anti-nodes spaced on the order of  $\lambda/2$ ). (c) Example raw data from a single pixel in a SAIM experiment with Fibronectin-Cy3 (blue dots) and fitted intensity profile with the optical model (red line). For imaging Fibronectin-Cy3 on silicon substrate with  $1.917 \mu\text{m}$  of silicon oxide, raw data was obtained from  $1.25^\circ$  to  $40^\circ$  in  $1.25^\circ$  increments.

Scanning Angle Interference Microscopy (SAIM), utilizes a patterned excitation field to impart axial information on the sample<sup>1</sup>. Placing a reflective surface behind the sample generates standing waves of excitation by interference of the incoming excitation light with the back-reflected wave. The spatial frequency of the interference fringes is well-defined and can be controlled by altering the angle of incidence of the excitation light on the reflective surface (Fig. 4.1a). As the incidence angle is changed, the relative intensity of fluorescently labelled structures within the sample varies as the standing wave excitation pattern changes. These periodic fluctuations are described by a mathematical model and depend on the fluorophore to reflective surface distance and the thickness of the SiO<sub>2</sub> layer (Fig. 4.1b)<sup>2</sup>. In a SAIM experiment images are collected over a series of excitation incidence angles. Intensity data from individual pixels over the image series is then computationally fit to the model to determine the pixel-wise height of the sample (Fig. 4.1c). From this data, the topographical maps of the features of interest are reconstructed.

A key requirement for artifact-free, high precision SAIM is an even, consistent excitation profile over the range of incidence angles. Lasers are commonly used in modern fluorescence microscopes to provide a high-intensity source of collimated excitation. Although the SAIM method can be employed with a standard lamp excitation source and a series of annular masks, the relatively slow switching time of the masks and short coherence length of the light source are less than ideal<sup>3</sup>. Taking this into account, SAIM benefits from the use of laser illumination. However, the relatively long coherence length of lasers gives rise to additional interference effects including speckles, fringes, and rings caused by imperfections and internal reflections

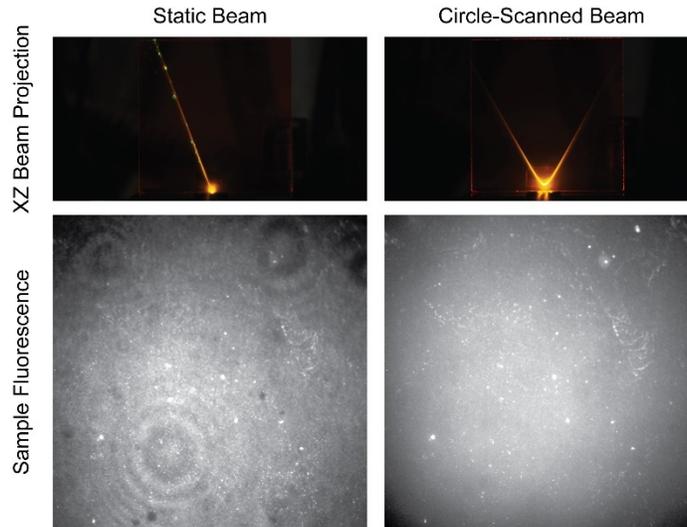
in the optical train<sup>4</sup>. Total Internal Reflection Fluorescence (TIRF) microscopes commonly employ lasers as excitation sources and suffer from the same effects in the quality of the illumination field. With the popularity of TIRF microscopy a variety of techniques have been developed to eliminate laser-induced excitation artifacts<sup>5-7</sup>. A commonly applied approach in localization-based super-resolution TIRF imaging is to use a “spinning beam” design. The laser is rotated continuously through the azimuth while maintaining a constant angle with respect to the optical axis, called circle-scanning<sup>8,9</sup>.



**Fig. 4.2 Principles of circle-scanning excitation.** (a) Top: The excitation beam in the sample plane is described by 3 coordinates, the angles  $\theta_x$  and  $\theta_y$ , and polarization,  $\phi$ . The  $\theta_x$  and  $\theta_y$  coordinates can be mapped to a polar angle,  $\theta$ , and azimuthal angle  $\psi$ . Bottom: A pair of galvanometer-mounted scanning mirrors are used to set the polar and azimuthal angles of the scanned excitation beam. (b) A circle-scanned beam projected onto a sheet of fluorescent plastic. (c) Image of the optical configuration of the scanning mirror and scan lens system. (d) Left: Per-axis command signals used to generate a series of polar scan angles with constant azimuthal velocity. Right: Beam position in the objective back aperture for a series of polar scan angles and 3D schematic of the excitation outputs.

In a circle scanning microscope the excitation light is focused on the rear focal plane of the objective by a scan lens so that the beam exiting the front pupil is collimated (Fig. 4.2a, left). The beam direction is controlled with a pair of close-coupled galvanometer-mounted scanning mirrors, one each for the X- and Y-axis (Fig. 4.2a). The scanning mirrors are positioned in the optical train to form an image of the apparent center of rotation (typically the midpoint between the mirrors) on the sample, preventing translation of the beam and providing a constant excitation profile throughout the scanning mirror's range of motion. In the sample plane, the beam can be described by its angular deflection in each axis,  $\theta_x$  and  $\theta_y$ , which are proportional to the scanning mirror positions. The deflections can be easily transformed to a spherical coordinate system to obtain a polar angle,  $\theta$ , and azimuthal angle,  $\psi$ , of deflection. The polar angle corresponds to the excitation angle on incidence. When  $\theta$  is kept constant and  $\psi$  is continuously varied, the beam forms a cone above the objective (Fig. 4.2b). Implementation of a circle-scanning system is straightforward, requiring only a few optical components (Fig. 4.2c). In order to maintain a constant velocity in  $\psi$ , the scanning mirrors are driven with sine waves (Fig. 4.2d). The frequency of the

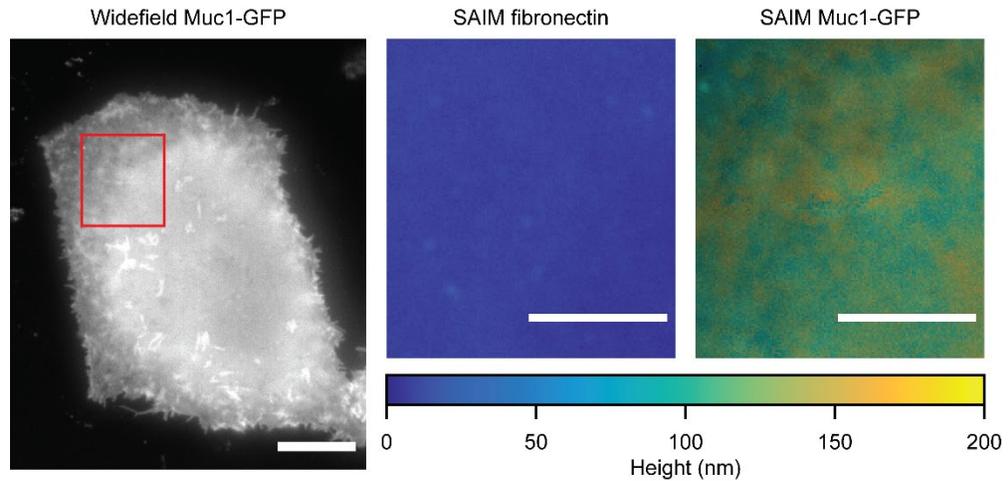
waveforms is matched with one axis offset by  $90^\circ$  in time, creating a circular scan pattern on the objective back focal plane.



**Fig. 4.3 Comparison of static and circle-scanned image quality in TIRFM and SAIM.** Top: Projection of the excitation laser from the objective lens onto a vertical screen. Bottom: Images of a fluorescent monolayer on a silicon wafer chip.

Applying the circle-scanning instrument design to SAIM significantly improves the quality of the excitation field, and in turn the reconstruction accuracy<sup>10</sup>. Figure 4.3 illustrates these effects on a fluorescent test sample. The sample was prepared by binding a homogeneous dye layer to a silicon wafer chip, typically used in SAIM. On the left a static beam, where the laser deflection was held at constant  $\theta$  and  $\psi$ , was used to excite the sample. Diffraction rings and dark spots caused by imperfections in the excitation optics are clearly visible. When the beam is scanned in  $\psi$  at a constant  $\theta$ , as in the image on the right, the rings disappear and the illumination is even over the field of view. With circle-scanning, we have been able to reliably

reconstruct height maps of previously difficult features such as the membrane-bound glycoprotein Muc1 (Fig. 4.4)<sup>11</sup>.



**Fig. 4.4 Mapping the plasma membrane topography of adherent cells with SAIM.** Left: Widefield view of a human epithelial cell ectopically expressing the fluorescent glycoprotein Muc1-GFP. Scale bar: 10  $\mu\text{m}$ . Center: Height reconstruction of the substrate bound matrix protein fibronectin. Right: Height reconstruction of GFP in the same region. Center and right images correspond to the boxed region in the widefield image. Scale bars: 5  $\mu\text{m}$ .

We have recently shown the benefits of both circle-scanning and hardware-based synchronization of peripheral devices in SAIM<sup>10</sup>. When working with live biological samples it is important to obtain the entire set of images in as short a time as possible. Accurate reconstruction of the sample topography assumes that features do not move between the acquisition of the first and last images. Furthermore, the high-intensity excitation light can be toxic to live samples and cause unwanted photobleaching. In this chapter we detail the preparation of live-cell SAIM samples, the construction of a high-speed, circle-scanning microscope optimized for SAIM, and

a basic procedure for performing SAIM experiments.

### 4.3 Materials

#### 4.3.1. Silicon Substrates

1. 4" silicon wafers with 19000 Å thermal oxide (Addison Engineering Inc., San Jose, CA).
2. Silicon wafer dicing tape.
3. Silicon wafer dicing saw (DAD3240; DISCO, Tokyo, Japan).
4. Film thickness measurement system (F50-EXR; Filmetrics, San Diego, CA).
5. Piranha solution: 25% (v/v) hydrogen peroxide solution (30% w/w) in concentrated sulfuric acid (*see* Note 1).
6. Wafer cleaning solution: 50% (v/v) concentrated hydrochloric acid in methanol.
7. HPLC grade acetone.
8. Sodium hydroxide solution: 1 M in deionized water.
9. Plasma cleaner (PDC-32G, Harrick Plasma, Ithaca, NY).
10. MPTS solution: 4% (v/v) (3-mercaptopropyl)trimethoxysilane in absolute ethanol.
11. GMBS solution: 4 mM 4-maleimidobutyric acid *N*-hydroxysuccinimide ester in absolute ethanol.
12. Fibronectin solution: 50 mg/mL human plasma fibronectin dissolved in PBS (*see* Note 2).

#### 4.3.2. Cell culture

1. MCF 10A human epithelial cells (ATCC# CRL-10317; American Type Culture Collection, Manassas, VA).
2. Cell culture medium: 5% (v/v) donor horse serum, 1% (v/v) penicillin-streptomycin solution, 10  $\mu\text{g}/\text{mL}$  insulin, 500 ng/mL hydrocortisone, 100 ng/mL cholera toxin and 10 ng/mL epidermal growth factor dissolved in 1:1 DMEM:F12.
3. Imaging media: 119 mM NaCl, 5mM KCL, 25 mM HEPES buffer, 2mM CaCl<sub>2</sub>, 2mM MgCl<sub>2</sub>, 6 g/L glucose, 1 g/L bovine serum albumin dissolved in deionized water at pH 7.4.
4. Glass bottom imaging dishes (coverslip thickness: No. 1.5, well diameter: 14 mm).

#### 4.3.3. Circle-Scanning Optical Components (Note 3)

1. Vibration isolation optical table.
2. Single-band laser cleanup filters (diameter: 1", mounted; ZET405/20x, ZET488/10x, ZET561/10x, ZET635/20x; Chroma Tech. Corp., Bellows Falls, VT).
3. Long-pass laser dichroic mirrors (diameter: 1", mounted; ZT405rdc, ZT488rdc and ZT561rdc; Chroma).
4. Broadband beam steering mirrors (diameter: 1", quantity: 7; BB1-E02; Thorlabs Inc., Newton, NJ).
5. Kinematic mirror mounts (quantity: 10; KS1; Thorlabs).
6. Iris diaphragms (quantity: 2; ID15; Thorlabs).

7. Air-spaced achromatic doublet scan lens ( $f$ : 30 mm, diameter: 1"; ACA254-030-A, Thorlabs).
8. Mounting tube for scan lens (length: 0.45"; SM1L03; Thorlabs).
9.  $M = 1$  zero-order vortex half-wave retarder (WPV10L-532; Thorlabs).
10. XY translation mount (ST1XY-S; Thorlabs).
11. Broadband elliptical mirrors (diameter: 2", quantity: 2; BBE2-E02; Thorlabs).
12. Right-angle kinematic mirror mounts for 2" elliptical mirrors (KCB2EC; Thorlabs).
13. Achromatic doublet relay lenses ( $f$ : 300 mm, diameter: 2", quantity: 2; AC508-300-A; Thorlabs).
14. Quick release cage system lens mounts (LCP90F; Thorlabs).
15. Optical cage system components for periscope construction (Thorlabs).
16. Optical posts and bases for mirrors (Thorlabs).
17. Optical power meter (PM100D and S122C; Thorlabs).
18. Neutral density filters for laser attenuation during alignment.
19. Laser collimation test device (shearing interferometer SI035; Thorlabs).
20. Beam alignment aids (1" optic alignment plate LMR1AP, iris diaphragm SM2D25D in removable 60 mm cage plate QRC2A and 60 mm cage alignment plate LCPA1; Thorlabs).

#### 4.3.4. Electronics and Peripheral Devices

1. 405 nm laser diode module (IQ1C100(LD2070); Power Tech. Inc., Little Rock, AR).
2. 488 nm diode-pumped solid-state laser module (Sapphire 488 LP; Coherent Inc., Santa Clara, CA).
3. 560 nm fiber laser module (VFL-560; MPB Communications Inc., Montreal, Quebec).
4. 642 nm fiber laser module (VFL-642; MPB).
5. Mechanical beam shutter (SHB1T, Thorlabs).
6. Visible spectrum acousto-optic tunable filter (AOTF) and multichannel RF driver ( $\lambda = 400\text{-}650$  nm; AOTFnC-400.650-TN and MPDS8CD6; AA Opto-Electronic, Orsay, France).
7. Mounted XY galvanometer scanning mirror set with driver (6215H galvanometers and 673 series dual servo driver; Cambridge Technology, Bedford, MA).
8. Scanning mirror driver heatsink.
9. DC power supplies (output: 28 Vdc, 5.4 A, quantity: 2).
10. Power supply protection diodes (Axial can,  $I_o$ : 10 A min).
11. Enclosure for scanning mirror driver and power supplies.
12. Electrical hook-up wire, connectors, power switch, indicator light and ventilation fan.
13. Coaxial SMA cables for controller/servo driver connection.
14. Coaxial BNC cable for controller/shutter connection.

15. DB25 male to female cable for controller/AOTF driver connection.
16. USB 2.0 type A-male to B-male cable for controller/computer connection.
17. Instrument controller (details and complete list of materials available from <https://github.com/mjc449/SAIMscannerV3>).
18. PICkit 4 in-circuit debugger (PG164140; Microchip Technology Inc., Chandler, AZ).
19. USB webcam for scanning system calibration.

#### 4.3.5. Imaging System

1. Inverted fluorescence microscope (e.g. Eclipse Ti2; Nikon Instruments Inc., Melville, NY).
2. Objective lens (CFI Plan Apo IR 60x WI, N.A. = 1.27; Nikon).
3. Quad bandpass fluorescence filter set with single-band emission filters (TRF89902-EM, mounted; Chroma).
4. Motorized emission filter wheel (LB10-B and LB10-NWE; Sutter Instrument, Novato, CA).
5. Scientific camera (Zyla 4.2 sCMOS; Andor, Concord, MA).
6. Computer workstation for data acquisition and instrument control (*see* Note 4).

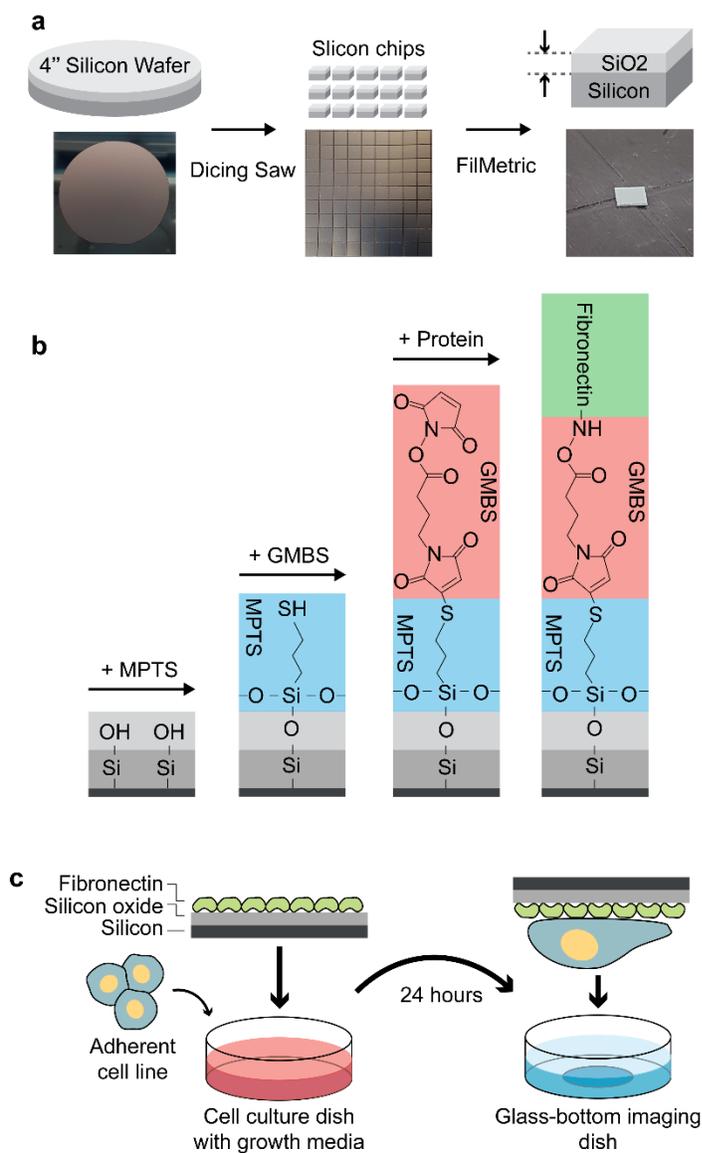
#### 4.3.6. Software

1. Instrument controller graphical interface (SSv3 Control Panel; available from <https://github.com/mjc449/SAIMscannerV3>).

2. MPLAB Integrated Programming Environment (available from <https://microchip.com/mplab/mplab-integrated-programming-environment>; Microchip).
3. Micro-Manager<sup>12</sup> (available from <https://micro-manager.org>).
4. FIJI<sup>13</sup> (available from <https://fiji.sc>).
5. SAIM analysis plugin for ImageJ<sup>14</sup> (available from <https://imagej.net/Saim>).

## 4.4 Methods

### 4.4.1. SAIM Sample Preparation



**Fig. 4.5 Sample preparation for SAIM imaging.** (a) Silicon wafers with a layer of thermal oxide are diced into 5 mm chips using a dicing saw. The thickness of the silicon dioxide layer is measured for each chip. (b) Chips are chemically functionalized to promote cellular adhesion with the extracellular matrix protein fibronectin. (c) Functionalized chips are placed in a culture dish and adherent cells are culture on the reflective surface of the chips. After 24 hours the chips are inverted

into a glass-bottom imaging dish for SAIM data collection.

The surface generated interference pattern required for axial localization is created by preparing samples on a highly reflective substrate. Polished silicon wafers are an inexpensive, easy to work with material for preparing SAIM samples. Wafers are commercially available with a range of thermal oxide thickness, and previous work has shown oxide layers of  $\sim 2 \mu\text{m}$  to be an ideal thickness for SAIM imaging<sup>1</sup>. The silicon dioxide surface of the wafers is functionalized with an appropriate matrix protein, such as fibronectin or collagen, to enhance cellular attachment.

The following protocol is generally applicable to all adherent cell lines. The choice of matrix protein will depend on the cell line under study. For poorly adherent cell lines the silanization and protein binding steps can be replaced with a 30 minute incubation in poly-L-lysine solution, however morphology and motility may be altered.

1. Affix dicing tape to the reflective surface of the wafer.
2. Using a wafer dicing saw, cut the wafer into 5 mm x 5mm chips (*see* Note 5).
3. Remove the chips from the dicing tape with tweezers. Measure the oxide layer thickness with a thin film measurement system (*see* Note 6).
4. Thoroughly clean the chips by incubation for 30 minutes in hot piranha solution. If piranha solution is not available clean the chips by successive 30-minute incubations in wafer cleaning solution, acetone, and 1 M sodium hydroxide solution in an

ultrasonic bath. Rinse the chips thoroughly with deionized water after each cleaning step. Store the chips submerged in deionized water until use.

5. Dry the chips thoroughly under a stream of pure nitrogen. Place the chips in a clean dry dish with the reflective side facing up and plasma clean for 60 s.
6. Remove from the plasma cleaner and add enough MPTS solution to completely cover the chips in the dish. Cover the dish and incubate for 30 minutes at room temperature.
7. Rinse the chips 3 times with absolute ethanol then add enough GMBS solution to completely cover the chips. Cover the dish and incubate for 30 minutes at room temperature.
8. Rinse the chips once with absolute ethanol then 3 times with PBS. Rinses should be performed in rapid succession to preserve GMBS reactivity. Immediately cover the surface of the chips with fibronectin solution. Incubate at room temperature for 2 hours or 4 °C overnight.
9. Rinse the functionalized chips thoroughly with PBS. If the chips will not be used immediately, they should be stored in sterile PBS containing 0.05% (w/v) sodium azide.
10. Rinse the chips with sterile PBS to remove storage buffer and transfer, reflective side up, to a clean cell culture dish. Seed cells onto the chips at a density of  $10,000 \text{ cm}^{-2}$ . Cover with cell

culture media and incubate at 37 °C and 5% CO<sub>2</sub> for 24 hours  
(*see* Note 7).

11. Prepare glass bottom imaging dishes by adding 1.5 mL of imaging media and placing in the cell culture incubator for 1 hour to equilibrate.
12. Using tweezers, remove a single chip from the cell culture dish, invert, and place in the imaging dish with the reflective side facing downward. Position the chip in the center of the glass-bottom well (*see* Note 8).

#### **4.4.2. Microscope and Imaging System Setup (see Note 9)**

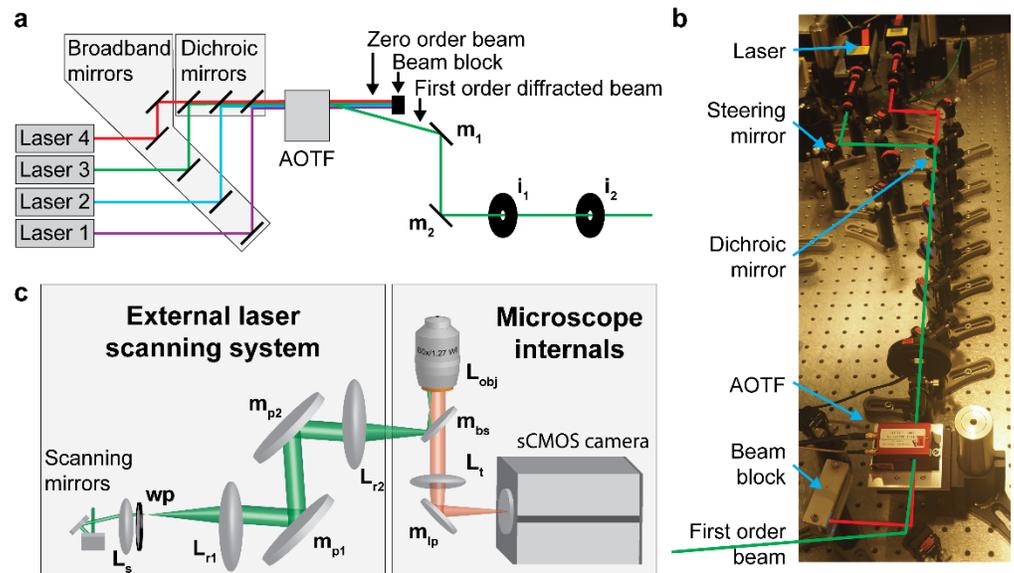
The location of the microscope body on the optical table will dictate the placement of the scanning optics, laser launch and other peripheral devices.

Therefore, it is important to leave sufficient space behind and around the microscope for the other components. As the largest and heaviest component of the system, the microscope body forms the base from which to build the scanning optics and laser launch (*see* Note 10).

1. Prepare the microscope body by removing the epi-fluorescence illuminator and optics if equipped. Bolt the microscope body securely to the table in its final position. Mount the objective lens and insert the quad-bandpass filter cube into the microscope body.

2. Load the emission filters into the filter wheel. Attach the filter wheel and camera to the microscope per the manufacturer's instructions.
3. Connect the camera, filter wheel, and instrument controller to the computer. In the Micro-Manager software add the camera and filter wheel to a new hardware profile (*see* Note 11).
4. Connect the camera's TTL auxiliary output to the instrument controller's trigger input. Configure the camera output to "fire all" in Micro-Manager.

#### 4.4.3. Laser Launch Construction



**Fig. 4.6 Circle-scanning system construction.** (a) Four individual free-space lasers are combined to form a single, colinear beam in the laser launch. The AOTF selects the excitation wavelength by modulating the intensity of the first order diffracted beam. The individual lasers are made colinear by alignment on irises  $i_1$  and  $i_2$ . Steering mirrors  $m_1$  and  $m_2$  are used to align the excitation beam on the scanning system

entrance pupil. **(b)** Image of the laser launch with individual components labeled. Note that both transmitted beams from the AOTF are reflected by mirror m1 before the zero-order beam is intercepted by the beam block, unlike in **a**. **(c)** Schematic diagram of the laser scanning and imaging systems with key components in their relative positions. Ls, scan lens; wp, vortex retarder; Lr1, Lr2, relay lenses; mp1, mp2, elliptical periscope mirrors; Lobj, objective lens; mbs fluorescence beam splitting mirror; Lt, tube lens; mlp, light path selection mirror.

The laser launch combines the 4 monochromatic laser beams into a single polychromatic beam using dichroic mirrors (Fig. 4.6a). The polychromatic beam is directed through the AOTF, which is responsible for wavelength selection, intensity control and high-speed shuttering. A mechanical shutter is included as a safety interlock, and iris diaphragms are used to align the lasers, ensuring a colinear beam and proper alignment on the scanning optics entrance pupil. It is important that the alignment irises are placed between the AOTF and scanning mirrors to account for the small but present wavelength-dependent variation in the first order diffracted beam direction.

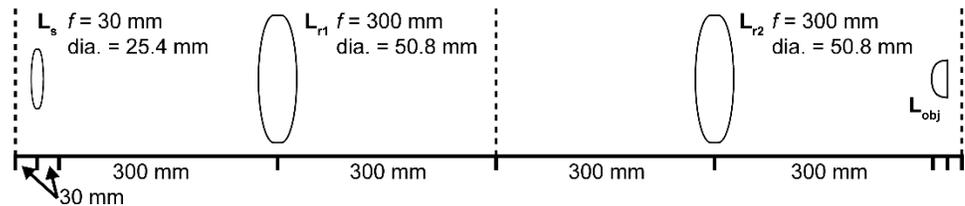
1. Attach the laser modules to the optical table securely using optical posts. The 405 and 488 nm laser heads are cooled by forced air and can be mounted with standard mechanical components. The 560 and 642 nm fiber laser modules can be located on a convenient equipment shelf, with their second harmonic generators (laser heads) are bolted to aluminum heat sinks with thermal conductive paste and mounted to the optical table via posts. The emission windows of all laser heads should lie in a

single plane at the same height as the AOTF and scanning mirrors.

2. Mount the dichroic long-pass mirrors in kinematic mirror mounts and attach to the optical table with optical posts. Position the mirrors in a straight line and orient at  $45^\circ$  to the AOTF (Fig. 4.6a,b). The 642 nm laser does not require a dichroic mirror, a broadband mirror should be used in its place.
3. Mount the broadband mirrors in kinematic mirror mounts. Position the mirrors such that the beam path forms a  $90^\circ$  angle between the laser emission window, broadband mirror and dichroic mirror (Fig. 4.6a,b). Secure the mirrors to the table.
4. Position the mechanical shutter immediately after the final dichroic mirror and directly in the beam path.
5. Attach the AOTF to an optical post using an appropriate heat sink and mount in the beam path (*see* Note 12). Position the beam block to intercept the zero-order transmitted beam from the AOTF and secure both the AOTF and beam block to the optical table.
6. Mount the broadband beam steering mirrors  $m_1$  and  $m_2$  in Fig. 6a to the optical table. Mirror  $m_1$  reflects the first order diffracted beam from the AOTF onto  $m_2$ , which reflects the beam into the entrance pupil of the scanning optics.

7. Connect the AOTF to the multichannel RF driver with the supplied cables. Connect the RF driver to the instrument controller with a DB25 cable. Connect the mechanical shutter TTL input to the instrument controller output with a BNC cable and configure the shutter controller for external triggering.

#### 4.4.4. Circle-Scanning Optics Construction



**Fig. 4.7 Arrangement of lenses within the laser scanning subsystem.** Each lens pair forms a  $4f$  system.  $L_s$ , scan lens;  $L_{r1}$  and  $L_{r2}$ , relay lenses;  $L_{obj}$ , objective lens.

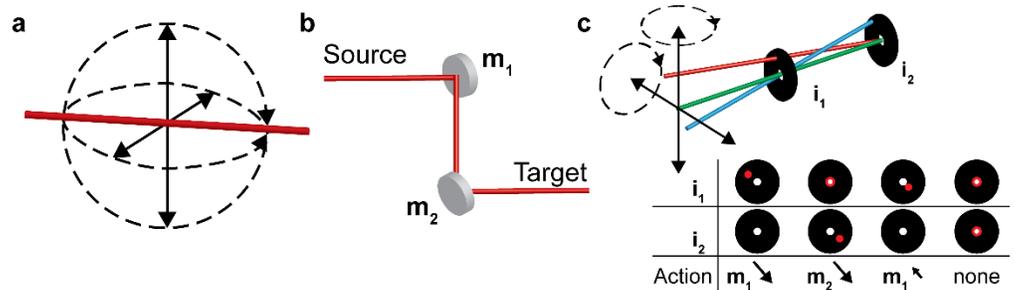
The circle scanning subsystem controls the beam deflection (polar and azimuthal angles) in the sample plane by relaying an image of the scanning mirrors to the sample plane of the microscope (Fig. 6c). It is critical that the image of the excitation beam center of rotation is relayed onto the sample. Misalignment of the optical elements can lead to distortion of the excitation profile in the sample plane, compromising data quality.

1. Assemble the laser scanning optics in the cage system according to Fig. 6c. The scan lens  $L_s$  is glued into its mounting tube and attached to the fixed side of the XY translation mount. The

vortex retarder is mounted in the translation mount and oriented to produce s-polarized excitation in the sample plane.

2. Starting with the second relay lens,  $L_{r2}$ , adjust the position of each lens in the cage system according to Fig. 7 (see Note 13).
3. Mount the scanning mirrors on an appropriate optical post. Position the scanning mirrors such that their exit pupil is aligned with the optical axis of the cage system. At this point the Y-axis scanning mirror should be centered in the scan lens with its rotational axis orthogonal to the optical axis of the cage system.
4. Ensure that the final beam steering mirror from the laser launch,  $m_2$  in Fig. 6a, is centered in the scanning mirror entrance pupil.

#### 4.4.5. Laser Alignment



**Fig. 4.8 Definition of laser orientation and alignment within the optical system.** (a) A beam is described by its position (solid lines) and angular direction (dashed arcs). (b) Beam alignment is set with two mirrors. (c) Top: Schematic diagram of two-point beam alignment using a pair of iris diaphragms,  $i_1$  and  $i_2$ . Bottom: Beam position and corrective actions to align a laser to a pair of irises. The beam position on each of the irises is denoted by the red spot.

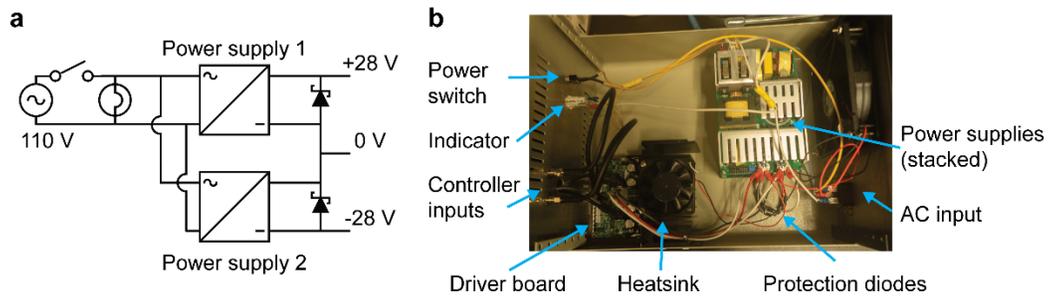
The laser beam within the optical system can be described by its offset and

direction with respect to the optical axis of the system, giving 4 coordinates (Fig. 4.8a). Two mirrors are employed, giving 4 degrees of freedom in orienting the beam (Fig. 4.8b). Similarly, two reference points formed by a pair of irises, are used to check beam alignment (Fig. 4.8c). Proper beam alignment with respect to the optical axis is critical in maintaining high-quality circle-scanned excitation. Small errors in beam position within the optical train can degrade the surface-generated interference pattern in SAIM experiments. Alignment of the lasers can be affected by multiple sources including changes in ambient temperature and mechanical perturbations. The beam alignment of each laser within the laser launch is verified against the permanently mounted irises, shown in Fig. 4.6a, at the beginning of each imaging session. Individual lasers are adjusted with the first broadband mirror and dichroic mirror that the laser encounters without altering mirrors in the polychromatic beam path ( $m_1$  and  $m_2$  in Fig. 4.6a). The same procedure can be employed to align the beam in the scanning mirror entrance pupil and entrance to the microscope. For the scanning mirrors an additional set of post-mounted irises are temporarily inserted immediately after the polychromatic beam steering mirrors  $m_1$  and  $m_2$  in Fig. 4.6a. To align the beam on microscope excitation port the removeable cage system iris and alignment plate should be placed between the second periscope mirror and the microscope excitation port. Alignment begins with the laser launch, followed by the scanning mirrors' entrance pupil and finally the microscope excitation port. The following procedure depicted schematically in Fig. 4.8c, applies in all cases.

1. Close both iris diaphragms until the opening is approximately the diameter of the beam.

2. Adjust the position of the beam on iris 1 using mirror 2 until the beam is centered in the iris.
3. Using mirror 1 while observing the beam position on iris 2, move the beam in a straight line directly away from the center of the iris.
4. Re-center the beam in iris 1 with mirror 2 and check its position on iris 2.
5. Repeat steps 3 and 4 until the beam is centered in both irises.

#### 4.4.6. Scanning Mirror Electronics

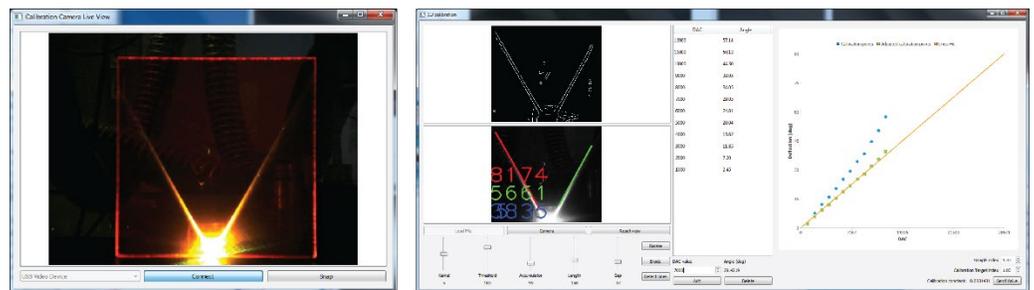


**Fig. 4.9 Scanning mirror power supplies and driver layout.** (a) Power supply circuit diagram. (b) Image of the power supply and driver layout within the enclosure.

The scanning mirrors' driver must be adequately cooled to dissipate heat generated during operation. Furthermore, the driver requires a high-quality bipolar power supply for proper operation. A circuit diagram for the series connection of two 28 V DC supplies is given in Fig. 4.9a. The diodes protect the power supplies during startup and shutdown from harmful current inversions but are reverse-biased in normal operation. Diodes with a small forward voltage drop and fast switching are selected. When arranging the components within the enclosure ensure that the driver heatsink is

between the driver board and power supplies (Fig. 4.9b). A thin layer of thermal conductive paste is applied to the driver mating surface before mounting. Once all components are mounted in the enclosure make the electrical connections according to the circuit diagram and the manufacturer’s instructions. Ensure that all components have a low-resistance ground connection to the enclosure and AC input ground conductor.

#### 4.4.7. Circle-Scanning Calibration



**Fig. 4.10 Circle-scanning system calibration.** Left: An inexpensive webcam attached to the instrument computer is used to capture images of the scanned excitation beam on a fluorescent target. Right: The calibration image is run through an edge detection algorithm to find the edges of beam. A series of images at different scan angles is used to determine the calibration value.

Before collecting data, the circle scanning system must be properly configured through the controller software. It is essential that the scanned beam is centered on and aligned to the optical axis. The waveform generators within the controller are fully programmable through the controller software for easy tuning of the scanned beam profile to correct ellipticity and alignment.

The scanned beam polar angle is proportional to the output waveform amplitude of the instrument controller, however, the relationship between output

amplitude and scan angle must be determined empirically. The waveform output is described in digital to analog converter (DAC) units and the conversion factor (calibration constant) to scan angle is determined by linear regression. A user friendly, image-based calibration tool is included to simplify the calibration procedure (Fig. 4.10). The following alignment, tuning and calibration procedure should be performed at the beginning of each imaging session.

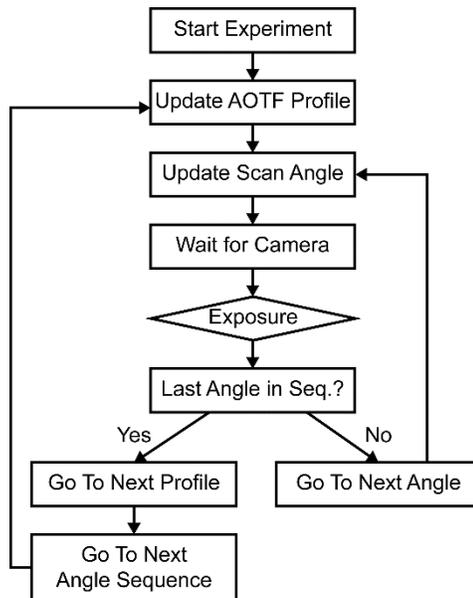
1. Prepare a dilute solution of carboxyfluorescein (405 and 488 nm lasers) or rhodamine B (560 nm laser) in water and add approximately 2 mL to a glass bottom imaging dish. The solution should be dilute enough to visualize the entire beam within the sample (~100  $\mu\text{M}$  is usually sufficient, although the exact concentration does not matter). Mount the dish on the microscope stage.
2. Start the controller software and connect to the device. Enable the excitation laser by clicking “On” under “Laser Controls”. Increase the power manually until the beam is clearly visible.
3. Under “Scanning Controls” click “Scan” and gradually increase the radius until a clearly visible cone is formed by the excitation laser in the dish. Check the box next to “Advanced” to enable the scan center, phase and Y-scale controls. Using these controls, center the excitation cone on the optical axis. The phase and Y-scale controls can be used to correct ellipticity in the scan profile.

4. Once the scanning system has been properly tuned in the software click the “Calibrate” button to open the calibration tool.  
Remove the alignment sample from the microscope and place the calibration target onto the stage (*see* Note 14). In the lower right of the calibration window enter the refractive indices of the calibration medium and sample medium (*see* Note 15).  
Connect a USB webcam to the computer.
5. In the calibration window click the “Camera” button. Select the USB webcam from the drop-down menu and click “Connect”.  
A live feed from the camera is displayed (Fig. 4.10, left). Set the scan radius to 1000 DAC units in the main window.  
Position the webcam to obtain a clear view of the beam on the calibration target. Click “Snap” in the camera window to send a still image to the calibration window.
6. In the calibration window, click and drag on the greyscale image to zoom in on the portion of the image containing the scanned beam. Below the image adjust the kernel size, accumulator, minimum length and maximum gap values until a clear view of the beam outline appears in the upper image. Click “Detect Lines.” The red and green lines overlaid on the lower window should align with the edges of the beam pattern. If the beam edges are not properly highlighted adjust the detection parameters and repeat the detection until they are. Enter the

current DAC value in the field below the data table and click “Add” to add the calibration point to the list. The new point will be added to the graph on the right side of the window. The graph displays the raw data (blue dots), refractive index corrected data (green squares) and best fit line for the current list of points.

7. Repeat steps 5 and 6 in increments of 1000 DAC units until the beam is no longer clearly visible on the calibration target. Once enough points have been collected click “Send Value” in the lower right of the calibration window to update the calibration value in the main software.

#### 4.4.8. Performing Circle-Scanning SAIM Experiments



**Fig. 4.11 Control flow diagram of a generalized experimental sequence.** Sequences of scan angles and excitation profiles are stored in the

instrument controller memory. The sequences and profiles are connected in a linked list of experimental steps.

Circle scanning SAIM experiments are designed in the controller software as combination of excitation profiles and scan angle sequences. Excitation profiles are laser intensity settings, controlled by the AOTF, describing the relative power of each channel. The experiments are built in steps, with each step containing a laser profile and angle sequence (Fig. 4.11). For each step the AOTF channel settings are updated before the first exposure of the camera and the scan angle is set to the first value in the sequence (*see* Note 16). After each exposure the scan angle is set to the next sequence value while the AOTF channel settings remain unchanged. When exposure of the last angle in the sequence is finished the experiment proceeds at the next step in the list, updating the AOTF channels are setting the scan angle to the first value in the new sequence. Profiles and sequences can be reused indefinitely within an experiment, and the experiment can be set to continue from any step when the end is reached.

1. In the controller software main window click “Fire On” to enable the AOTF shutter. Start the live view in Micro-Manager. Adjust the laser intensities in the controller software to the desired level. Click “Add” under “Laser Profiles” in the controller software to add the current AOTF setting as a new profile. To update a profile select the profile, adjust the power and click “Update”. The old levels will be overwritten with the current settings.

2. To set the scan angles click “New” under “Scan Angle Sequences” in the controller software. A dialog box will open. Set the first and last angles and total number of angles for the sequence (*see* Note 17). Click “OK” to close the window and save the sequence.
3. In the “Experiment Design” section of the controller software click “Add” and a dialog window will open. Select the step, sequence, and profile number then click “Add” to close the window and add the step to the experiment.
4. Steps can be reordered in the experiment list using the “Up” and “Down” buttons. The “Delete” Button will remove the currently selected step. Selecting a step in the list and clicking the “Loop” button will cause the experiment to continue from the selected step when the end of the list is reached.
5. Once the experiment has been defined click “Start”. At this point the controller is placed in autonomous operation. At the end of each of the camera’s exposures the controller will go to the next scan angle and, if applicable, update the AOTF profile.
6. For data collection in simple experiments with a single excitation profile the multi-dimensional acquisition feature in Micro-Manager provides an easy and convenient interface. For more complex experiments including multiple color channels,

multiple stage positions or delays between exposure sequences the scripting interface in Micro-Manager should be used.

7. For processing images are organized in stacks by excitation wavelength. Each stack contains a single sequence of excitation angles. For details on SAIM data processing in Fiji see <sup>14</sup>. Alternatively, a highly optimized command line analysis program is supplied as C/C++ source code in the instrument controller project repository. Compiling the analysis program requires additional open-source and proprietary libraries and should be done on the analysis workstation. For more details see <sup>10</sup>.

#### 4.5 Notes

1. Piranha solution should be prepared fresh before use. Use extreme caution when mixing and handling. Always keep piranha solution in a chemical fume hood and wear proper personal protective equipment. Check local regulations concerning the preparation, use and disposal of piranha solution. If proper equipment for preparation and disposal of piranha solution is not available wafer cleaning solution can be used as a less-effective substitute.
2. To create a fluorescent surface on the SAIM substrate the matrix protein can be dye labeled. We routinely label fibronectin using the following procedure:

- i. Dissolve Cy3-*N*-hydroxysuccinimide ester in anhydrous DMOS, make 25 ng aliquots and lyophilize.
- ii. Add 1 mL of a 1 mg/mL fibronectin solution in PBS to a lyophilized aliquot of the dye.
- iii. Incubate at room temperature for 1 hour with mixing.
- iv. Remove the free dye by dialysis against 1 L of PBS overnight at 4 °C.

3. Specific parts for mechanical components (e.g. optical posts, post bases and optical cage components) should be selected based on materials on hand and the mechanical dimensions of the microscope, lasers and other components.
4. When selecting an instrument control and acquisition computer it is important to match the storage capacity and speed with the camera's data generation rate. Scientific cameras often produce data faster than the storage speed of a single hard disk or solid-state drive over a SATA connection. With a sufficiently large free memory pool (RAM), the incoming images can be spooled in a buffer, allowing the camera to run at full speed until the system memory is full. Alternatively, a properly configured redundant array of inexpensive disks (RAID) can achieve write speeds that exceed the camera's data generation rate. However, hardware RAID controllers are expensive and can be difficult to configure and maintain. PCIe-connected solid-state drives are relatively inexpensive, readily available, and fast enough to handle the

full-framerate of scientific cameras, making them our preferred storage method.

5. If a dicing saw is not available, the wafer can be sectioned into chips with a diamond-tipped scribe. In this case the wafer should be placed reflective side down on a clean tissue. Draw the scribe across the back of the wafer several times using a straight-edge. Press down on the scribed line to snap the wafer. Clean silicon particles from the chip with a blast of clean compressed air before proceeding.
6. The silicon oxide film thickness can vary significantly over the surface of the wafer. It is important to measure the thickness on each chip before proceeding with functionalization and imaging.
7. For fluorescent protein fusions, the culture should be transfected 24 hours prior to seeding on the silicon chips. Follow the transfection protocol provided by the reagent supplier.
8. If vibrations or buoyancy are an issue the silicon chips can be held with a small, non-reactive metal weight. We have found that a stainless steel  $\frac{1}{4}$ "-20 nut works well for this purpose. The weight should be sterilized with 70% ethanol and equilibrated to the sample temperature.
9. Many of the peripheral devices have external power supplies and driver modules. These should be placed on a convenient equipment shelf that is detached from the optical table to prevent the transmission of vibrations to the optical system. Vibration producing equipment should be isolated from the optical system wherever possible.

10. It is advisable to build a “mock-up” of the completed system using the major components (lasers, AOTF, scanning mirrors, periscope mechanical components and microscope) on the optical table before assembly. Use a tape measure or ruler and masking tape on the table top to indicate the approximate placement of the lenses and mirrors. This will ensure that there is adequate space for placement of all the components and serve as a guide during construction.
11. Instructions for configuring and running Micro-Manager are available at <https://micro-manager.org> or the Micro-Manager mailing list.
12. The AOTF orientation relative to the beam path is important for proper operation. During initial system construction it is useful to align the lasers using 2 iris diaphragms temporarily mounted long the path of the polychromatic beam. Once aligned, the AOTF should be positioned and tuned according to the manufacturer’s instructions using an optical power meter. After this the irises can be relocated and used as  $i_1$  and  $i_2$  in Fig. 4.3a.
13. Coarse positioning of the lenses within the scanning system can be done with a ruler, however final adjustment should be made with respect to the excitation laser collimation. This is particularly true of the distance between the second relay lens and the objective lens. The objective lens is constructed of many individual optical elements and its rear focal plane must be determined empirically. We recommend the following procedure for fine-tuning the lens’ positions within the

system after coarse assembly (use proper precautions to prevent dangerous eye exposure to the laser beams). Before starting, ensure that the scanning mirrors are in a neutral, centered position and the lasers are aligned to the optical axis of the cage system.

- v. Remove the scan lens and first relay lens. Place a drop of water and glass coverslip on the microscope sample holder. Enable one of the excitation lasers so that a spot is visible on the ceiling above the microscope. Adjust the second relay lens ( $L_{r2}$  in Fig. 4.4) to minimize the projected spot size. Secure the relay lens in the cage system.
- vi. Remove the objective lens from the microscope and insert the first relay lens ( $L_{r1}$  in Fig. 4.4) into the cage system. Place the shearing interferometer into the beam path on top of the microscope stage. Adjust the position of  $L_{r1}$  until a collimated beam is achieved.
- vii. Attach the scan lens to the XY translator and place the shearing interferometer immediately after  $L_{r1}$ . Adjust the position of the XY translator/scan lens assembly until a collimated beam is achieved.
- viii. Mount the objective lens to the microscope body and prepare a dilute fluorescent solution in a glass-bottom dish. We typically use carboxyfluorescein dissolved in deionized water with the 488 nm laser for this step. The solution

should completely fill the dish to give the best view of the beam. Adjust  $L_{r2}$  to form a collimated beam in the solution.

ix. Repeat steps ii and iii to finalize the axial positioning of the scanning system lens stack.

14. A simple calibration target can be constructed from a rectangular piece of fluorescent plastic, such as that in Fig. 4.7. The plastic is attached to a 90° mounting bracket. The plastic is positioned vertically in the sample holder and the beam projected along its surface. Alternatively, a piece of plain white paper can be suspended directly above the objective, however positioning the paper is more difficult than the rigid plastic sheet.

15. Entering accurate values for the target and sample refractive indices is critical to accurate calibration of the system. The “Sample index” value should be that of the actual sample to be imaged during experiments. The “Calibration Target Index” should be set to that of air if the beam is being projected onto the surface of the fluorescent plastic sheet. If the beam is propagating inside the plastic then the refractive index of the material should be entered in this field. The refractive index values can be changed at any time during calibration and the new values will be applied to all existing data points.

16. The emission filter wheel is the slowest peripheral device in the system. For this reason, the experimental sequence prioritizes scan angle changes over excitation profile changes. In some cases it may be

necessary to change excitation profiles with each exposure. In this case sequences should be limited to a single angle and a step should be added to the experiment profile for each exposure in the acquisition.

17. The range of angles to scan should encompass at least 1 full cycle in the intensity oscillation profile for accurate reconstruction of the sample topography. The number of angles should balance acquisition speed, photodamage to the sample, and data size. For samples prepared on substrates with 19000 Å thermal oxide we typically use minimum and maximum scan angles of 1.25° and 40°, respectively. If the data will be processed using the Fiji plugin,<sup>14</sup> the specific number of angles is less important. The analysis software included in the instrument controller project repository has been optimized for sequence lengths that are multiples of 16 (e.g. 16, 32, 64), with 32 being the preferred number. While sequences of any length can be analyzed, deviation from this pattern will increase analysis time.

#### **4.6 Author Contributions**

M.J.C., A.S., W.R.Z. and M.J.P. conceived the circle scanning approach to SAIM. M.J.C. designed and built the circle scanning microscope and controller electronics. M.J.C. wrote the driver, firmware and graphical interface software. M.J.C. and S.P. developed the sample preparation protocols. S.P. prepared samples and collected representative data for the figures. M.J.C., S.P. and W.R.Z. created figures. M.J.C. wrote the manuscript with input from all authors.

## REFERENCES

1. Paszek, M. J., DuFort, Christopher C, Rubashkin, Matthew G, Davidson, Michael W, Thorn, Kurt S, Liphardt, Jan T, Weaver, Valerie M,. Scanning angle interference microscopy reveals cell dynamics at the nanoscale. *Nat Meth Nature Methods* **9**, 825–827 (2012).
2. Lambacher, A. & Fromherz, P. Fluorescence interference-contrast microscopy on oxidized silicon using a monomolecular dye layer. *Applied Physics A: Materials Science & Processing* **63**, 207–217 (1996).
3. Ajo-Franklin, C. M., Ganesan, P. V. & Boxer, S. G. Variable Incidence Angle Fluorescence Interference Contrast Microscopy for Z-Imaging Single Objects. *Biophysical Journal* **89**, 2759–2769 (2005).
4. Goodman, J. W. *Speckle Phenomena in Optics: Theory and Applications*. (Roberts and Company Publishers, 2007).
5. Dingel, B. & Kawata, S. Speckle-free image in a laser-diode microscope by using the optical feedback effect. *Opt. Lett., OL* **18**, 549–551 (1993).
6. Mattheyses, A. L., Shaw, K. & Axelrod, D. Effective elimination of laser interference fringing in fluorescence microscopy by spinning azimuthal incidence angle. *Microscopy Research and Technique* **69**, 642–647 (2006).
7. Fiolka, R., Belyaev, Y., Ewers, H. & Stemmer, A. Even illumination in total internal reflection fluorescence microscopy using laser light. *Microscopy Research and Technique* **71**, 45–50 (2008).
8. Fu, Y. *et al.* Axial superresolution via multiangle TIRF microscopy with sequential imaging and photobleaching. *Proceedings of the National Academy of Sciences* **113**, 4368–4373 (2016).
9. Chen, Y. *et al.* Multi-color live-cell super-resolution volume imaging with multi-angle interference microscopy. *Nature Communications* **9**, 4818 (2018).
10. Colville, M. J., Park, S., Zipfel, W. R. & Paszek, M. J. High-speed device synchronization in optical microscopy with an open-source hardware control platform: Supplementary Information and Figures. *bioRxiv* (2019). doi:10.1101/533349
11. Shurer, C. R. *et al.* Genetically Encoded Toolbox for Glycocalyx Engineering: Tunable Control of Cell Adhesion, Survival, and Cancer Cell Behaviors. *ACS Biomaterials Science & Engineering* acsbiomaterials.7b00037 (2017). doi:10.1021/acsbiomaterials.7b00037

12. Edelstein, A., Amodaj, N., Hoover, K., Vale, R. & Stuurman, N. Computer Control of Microscopes Using  $\mu$ Manager. in *Current Protocols in Molecular Biology* (eds. Ausubel, F. M. et al.) (John Wiley & Sons, Inc., 2010).
13. Schindelin, J. *et al.* Fiji: an open-source platform for biological-image analysis. *Nature Methods* **9**, 676–682 (2012).
14. Carbone, C. B., Vale, R. D. & Stuurman, N. An acquisition and analysis pipeline for scanning angle interference microscopy. *Nature Methods* **13**, 897–898 (2016).

# CHAPTER 5

## Conclusions

Scanning angle interference microscopy is a powerful tool for investigating the nanometer scale spatial organization of subcellular structures in biological samples. The high precision axial localization afforded by SAIM has been used to visualize the stratified architecture of both integrin-based cell-matrix adhesions<sup>1</sup> and cadherin-based cell-cell adhesions<sup>2</sup>, as well as to demonstrate the kinetic segregation of receptors on the extracellular leaflet of the plasma membrane in areas of close contact with a rigid surface<sup>3,4</sup>. In this work we have developed and validated an optimized SAIM microscope for live cell imaging that increases temporal resolution while simultaneously mitigating the effects of instrument induced reconstruction artifacts.

In chapter 2 we introduced a library of variable length sialomucins for systematic manipulation of the cellular glycocalyx thickness. These designer glycoproteins were based on Muc1 and Podxl, both of which have been shown to have roles in intracellular signaling. To limit the intracellular biochemical activity of our designer mucins we created a parallel library consisting of fusion proteins of the extracellular mucin domains to a synthetic transmembrane stalk. Using SAIM, we then verified that the semisynthetic mucins were able to recapitulate the glycocalyx thickening effects of the original mucins. Additionally, we found that the use of a transposon-based integration system led to significantly higher levels of mucin expression than the lentiviral system used in previous reports, and as a consequence caused the cells to adopt a spherical morphology and eventually led to complete

detachment of cells from the substrate and their survival in suspension. While the SAIM studies in chapter 2 provided a measure of the glycocalyx thickening effects of various mucins, they were conducted on fixed samples and as such could not capture the dynamic, mechanical process of cellular detachment. Furthermore, we found that mucin expressing cells were unable to proliferate under suspension culture conditions, but observed detached cells reattaching to the substrate and undergoing division. Presumably, the glycocalyx of a detached cell is fully relaxed, forming a thick, repulsive barrier between cell-surface adhesion receptors and the substrate. In future experiments live-cell SAIM could be used to observe the mechanical compression of the glycocalyx during reattachment.

Previous reports of FLIC and SAIM provided a theoretical framework<sup>5</sup>, methodological description<sup>6</sup>, and acquisition and analytical tools<sup>7</sup> in the form of plugins for the popular open source software packages Micro-Manager<sup>8</sup> and ImageJ<sup>9</sup> based on the adaptation of commercially available microscope systems. In chapter 3 we introduced a SAIM-specific microscope that addressed the limitations of previous implementations with respect to data collection rate and systematic, instrument-induced excitation artifacts. To achieve an even excitation field across the range of angles used in SAIM we adapted the azimuthal beam scanning method popular in TIRFM. We then compared the performance of azimuthal beam scanning to conventional, static beam SAIM using supported lipid bilayers. Azimuthal beam scanning resulted in a greater than 5-fold increase in the observed signal to noise ratio, leading to a narrower distribution of fitted bilayer heights and more accurate reconstructions. Furthermore, static beam SAIM resulted in reconstructions that

showed consistent topographical artifacts across multiple samples. These artifacts were pixel-dependent, indicating that they were instrument induced and not a property of the samples. We also demonstrated the effects of timing uncertainty and characterized the artifacts arising from poor synchronization between the excitation shutter and camera exposure. We observed a variable latency between shuttering events and the camera exposure signal on the order of 10 ms when using a software-based synchronization scheme. We then implemented a hardware control scheme that reduced latency and variability to the order of 10  $\mu$ s. In comparing reconstructions with the two control schemes we found a correlation between the fitted height and both observed intensity and signal to noise ratio under software control. Under hardware control the fitted height was independent of these parameters. Through a systematic investigation of the instrument components we were able to optimize our microscope for fast, high-precision SAIM imaging.

Instrument design and construction can be a difficult and expensive process. Many of the optimizations we have performed on our microscope have been through incremental improvements over the course of multiple rounds of prototyping. In chapter 4 we provided a detailed parts list and instructions for building an azimuthal beam scanning SAIM microscope and conducting high-speed SAIM experiments. A key development in our SAIM specific microscope was the hardware controller which performs synchronization between the individual components. We integrated waveform generation, multiple analog outputs, digital I/O and USB HID connectivity into a single hub that can control a broad range of peripheral devices. The design, firmware and control software were published as an open source project. A limitation

of the current controller is the SAIM focused firmware and software. At the time of this writing we are in the process of developing a new firmware framework that will provide a flexible interrupt assignment scheme with the goal of allowing the user to define the relationship between an interrupt source and the action taken by the controller through software without having to rewrite the controller firmware.

Coincident with this area of development we have switched to using a USB bootloader for firmware updates to eliminate the need for an external hardware programmer.

Continuing work on this project will include full integration into the Micro-Manger microscope control software, providing a single interface for the entire instrument.

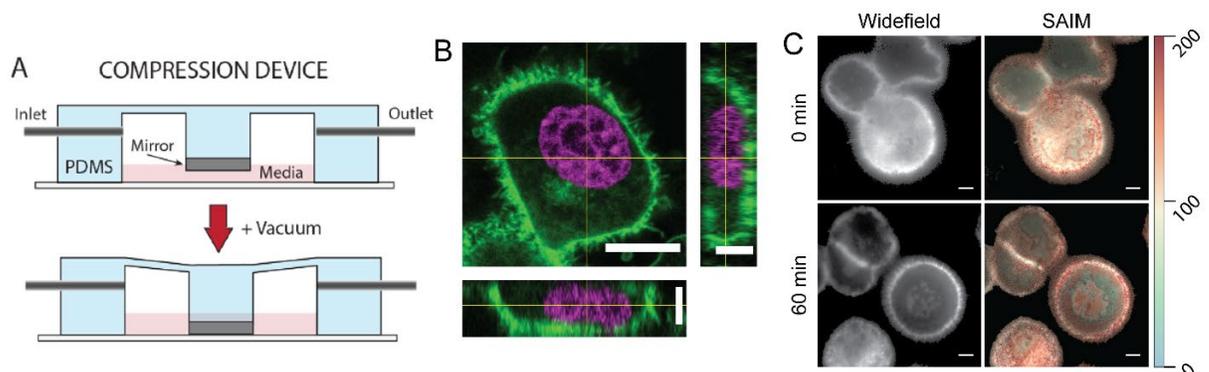
### **Related Work and Future Directions**

One of the motivations for our work in the preceding chapters has been to better understand the mechanical and physical properties of the glycocalyx. Recently, we have shown that overexpression of bulky glycocalyx components, such as Muc1, can drive the formation of highly curved plasma membrane structures, such as blebs and microvilli, on the apical surface of cells<sup>10</sup>. A key finding of this study was that the transition from a bleb to tube morphology corresponded with the polymer surface density at which mucins form a polymer brush, giving rise to an entropic penalty as the long, flexible extracellular domains are packed together and restricted to an extended conformation. Other work in our lab has taken a computational approach to predicting the glycocalyx and membrane deformation in response to mechanical compression<sup>11</sup>. Additionally, we have continued to expand and characterize our designer mucin library to include a wider range of backbone lengths and sequence-

specific mucins with varying numbers of potential glycosylation sites<sup>12</sup>. Future studies coupling the imaging technologies presented in this work with the tools developed in our lab for precision glycocalyx editing will seek to understand the relationship between the biochemical composition of the glycocalyx and its physical and mechanical properties.

Towards this end, we have begun efforts to integrate the SAIM microscope with micromechanical manipulators to probe glycocalyx compressibility. Using the designer glycoprotein libraries, we can vary both the glycocalyx thickness (glycoprotein backbone length) and density (glycan grafting density) in a systematic manner. Coupling these perturbations with the nanometer scale imaging capabilities afforded by SAIM and micromanipulation will allow future studies to build a more coherent model for the physical properties that govern glycocalyx mechanics. In particular, we have developed a one-dimensional confinement system by adapting a previously reported design to include a reflective silicon substrate for SAIM<sup>13</sup> (Fig. 5.1a). In this system cells are cultured in a standard glass bottom imaging dish. Rigid microspheres of a defined diameter are added to the sample media and an elastic plunger fitted with a reflective surface is gently lowered onto the sample by drawing a vacuum within the dish until it comes to rest on the spheres. The gap between the glass coverslip and reflective substrate, or confinement height, is then set by the diameter of the spheres. In preliminary experiments we have validated that cells remain viable for at least 60 minutes of continuous confinement at heights between 5 and 12  $\mu\text{m}$  with human epithelial cells expressing full length Muc1 dCT (Fig. 5.1b). We also performed time-lapse SAIM experiments over 90 minutes. Interestingly, on

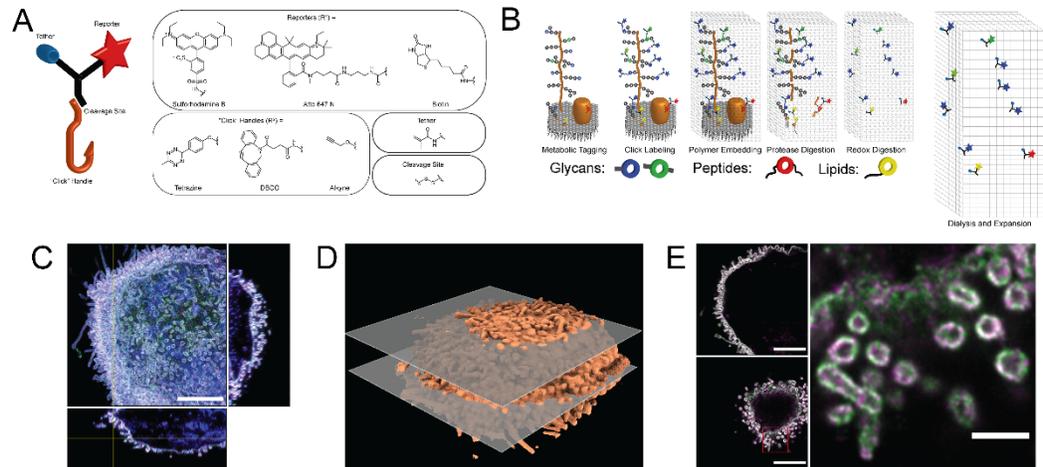
initial application of confinement there was no obvious change in the Muc1 lateral distribution on the ventral membrane. Over the course of these experiments, however, we observed the progressive formation of large, Muc1 dense domains. As expected, the membrane-substrate separation was greater in the Muc1 enriched regions than outside. These results aligned with those previously reported for kinetic segregation of CD45 in T cell signaling<sup>3</sup>. In future studies we will determine whether the formation of mucin rich domains is a condensation phenomenon or the coalescence of multiple, smaller possibly preexisting domains in response to increased mechanical strain on the Muc1 brush.



**Fig. 5.1 Confinement between rigid surfaces as a mechanical probe of the glycocalyx.** (A) Schematic depiction of the one-dimensional confinement device. (B) Confocal midplane and orthogonal slices of a MCF10A human epithelial cell expressing Muc1  $\Delta$ CT moxGFP (green) and confined to a height of 6  $\mu$ m. DNA was labeled with Hoescht stain (magenta) to visualize nuclear deformation. Scale bars 10  $\mu$ m in x/y and 5  $\mu$ m in x/z and y/z. (C) Widefield fluorescence and SAIM height reconstruction images of 4T1 mouse epithelial cells expressing Muc1  $\Delta$ CT moxGFP under 3  $\mu$ m confinement. Times are since the beginning on confinement. Scale bars 10  $\mu$ m.

While SAIM provides high-precision axial localization the lateral resolution is

still subject to the classical diffraction limit. An alternative technique, called Expansion Microscopy (ExM) circumvents this resolution limit by chemically linking the sample to a swellable polymer gel followed by mechanical magnification<sup>14</sup>. To date several ExM linking strategies have been developed, however none are optimal for direct analysis of glycans. In collaboration with members of the Alabi research group we have designed and synthesized a panel of trifunctional ExM reagents that use orthogonal “click” chemistry for molecular detection of unnatural glycans on the cell surface (Fig. 5.2a). These linker reagents are based on an oligoTEA trifunctional scaffold supporting the click handle, a methacrylamide moiety and a fluorophore. The linker backbone was resistant to the protease digestion and detergent extraction step required in ExM, resulting in strong fluorescence retention throughout the sample preparation protocol (Fig. 5.2b). In preliminary experiments we metabolically labeled Muc1 moxGFP expressing cells with the unnatural sugar azidomannose followed by click labeling with our trifunctional linker to selectively detect cell-surface sialic acid residues. We then performed the standard protein retention ExM protocol and visualized the expanded cells by confocal microscopy<sup>15</sup> (Fig. 5.2c,d). Surprisingly, on the highly curved surfaces of microvilli we observed inhomogeneous and anti-correlated spatial distributions of Muc1 and sialic acid (Fig. 5.2e). While these were preliminary results and additional experiments are needed to confirm our findings, we have corroborating evidence in the form of TIRFM images that suggest a similar phenomenon on the ventral membrane.



**Fig. 5.2 Trifunctional click chemistry probes for expansion microscopy of non-traditional target molecules.** (A) Cartoon depiction and chemical structures of the key components of the ExM click chemistry probe. (B) Schematic representation of the sample preparation ExM workflow. (C) Maximum intensity projection and orthogonal views of a human epithelial cell expressing Muc1-moxGFP (blue). Metabolically incorporated ManNAz was labeled with the trifunctional rhodamine linker molecule (green) and Atto 647N anti-GFP nanobody prior to gelation and expansion (magenta). Scale bar 10  $\mu\text{m}$ . (D) Surface rendering of the cell in A. (E) Single x/y slices corresponding to the gray planes from D (left) and zoomed view of the membrane projections boxed in red (right). Scale bars 5  $\mu\text{m}$  and 1  $\mu\text{m}$ , respectively. The GFP channel was removed from the images to aid in visualizing the cell surface structures.

Fluorescence imaging is a powerful tool for the study of biomolecule spatial organization, and in this work, we have sought to enhance and expand upon a few state of the art techniques with an emphasis on their applications in glycocalyx biophysics. Several of our results suggest that the heavily O-glycosylated extracellular domain of mucins plays an important role in their spatial distribution beyond simple steric interactions. For example, in preliminary experiments we have observed the formation of Muc1 rich domains, like those seen under mechanical confinement, by lowering the sample temperature. Future studies should address the

formation and composition of these domains to determine whether they are an example of thermodynamic phase separation and if they are in fact analogous to those seen under confinement. Furthermore, the role of crosslinking agents in the glycocalyx is largely unexplored. For example, multivalent galectins could play a role in crosslinking Muc1 glycan side chains along the protein contour, extending the backbone conformation and promoting dense packing. Another important consideration is the existence of nanoscale Muc1 clusters under physiological conditions and the transition boundaries to resolvable, micrometer size domains. Future studies should benefit greatly from the enhanced imaging tools that we have developed in addressing these questions.

## REFERENCES

1. Xia, S., Yim, E. K. F. & Kanchanawong, P. Molecular Organization of Integrin-Based Adhesion Complexes in Mouse Embryonic Stem Cells. *ACS Biomater. Sci. Eng.* **5**, 3828–3842 (2019).
2. Bertocchi, C. *et al.* Nanoscale architecture of cadherin-based cell adhesions. *Nature Cell Biology* **19**, 28–37 (2016).
3. Carbone, C. B. *et al.* In vitro reconstitution of T cell receptor-mediated segregation of the CD45 phosphatase. *Proceedings of the National Academy of Sciences* **114**, E9338–E9345 (2017).
4. Paszek, M. J. *et al.* The cancer glycocalyx mechanically primes integrin-mediated growth and survival. *Nature* **511**, 319–325 (2014).
5. Lambacher, A. & Fromherz, P. Fluorescence interference-contrast microscopy on oxidized silicon using a monomolecular dye layer. *Applied Physics A: Materials Science & Processing* **63**, 207–217 (1996).
6. Paszek, M. J., DuFort, Christopher C, Rubashkin, Matthew G, Davidson, Michael W, Thorn, Kurt S, Liphardt, Jan T, Weaver, Valerie M,. Scanning angle interference microscopy reveals cell dynamics at the nanoscale. *Nat Meth Nature Methods* **9**, 825–827 (2012).
7. Carbone, C. B., Vale, R. D. & Stuurman, N. An acquisition and analysis pipeline for scanning angle interference microscopy. *Nature Methods* **13**, 897–898 (2016).
8. Edelstein, A., Amodaj, N., Hoover, K., Vale, R. & Stuurman, N. Computer Control of Microscopes Using  $\mu$ Manager. in *Current Protocols in Molecular Biology* (eds. Ausubel, F. M. *et al.*) (John Wiley & Sons, Inc., 2010).
9. Schindelin, J. *et al.* Fiji: an open-source platform for biological-image analysis. *Nature Methods* **9**, 676–682 (2012).
10. Shurer, C. R. *et al.* Physical Principles of Membrane Shape Regulation by the Glycocalyx. *Cell* **177**, 1757-1770.e21 (2019).
11. Gandhi, J. G., Koch, D. L. & Paszek, M. J. Equilibrium Modeling of the Mechanics and Structure of the Cancer Glycocalyx. *Biophysical Journal* **116**, 694–708 (2019).
12. Pan, H., Colville, M. J., Supekar, N. T., Azadi, P. & Paszek, M. J. Sequence-Specific Mucins for Glycocalyx Engineering. *ACS Synth. Biol.* **8**, 2315–2326

(2019).

13. Liu, Y.-J. *et al.* Confinement and Low Adhesion Induce Fast Amoeboid Migration of Slow Mesenchymal Cells. *Cell* **160**, 659–672 (2015).
14. Chen, F., Tillberg, P. W. & Boyden, E. S. Expansion microscopy. *Science* **347**, 543–548 (2015).
15. Tillberg, P. W. *et al.* Protein-retention expansion microscopy of cells and tissues labeled using standard fluorescent proteins and antibodies. *Nat Biotech* **34**, 987–992 (2016).

# Appendix A<sup>‡‡</sup>

## Supporting Information for Chapter 2

### TITLE

A Genetically Encoded Toolbox for Glycocalyx Engineering: Tunable Control of Cell Adhesion, Survival, and Cancer Cell Behaviors

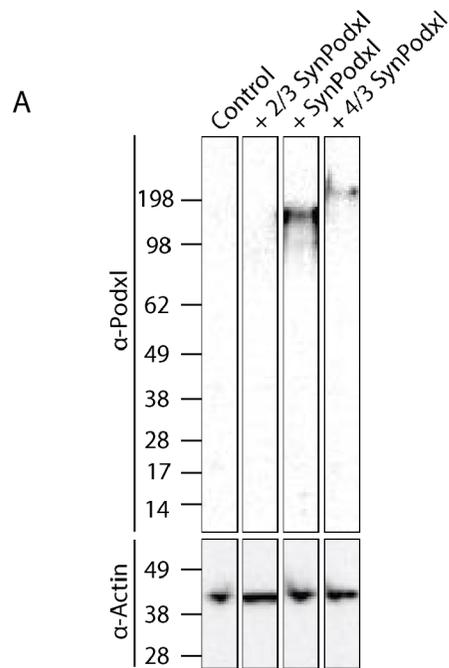
### AUTHORS

Carolyn R. Shurer\*, Marshall J. Colville\*, Vivek K. Gupta, Shelby E. Head, FuiBoon Kai, Jonathon N. Lakins, Matthew J. Paszek

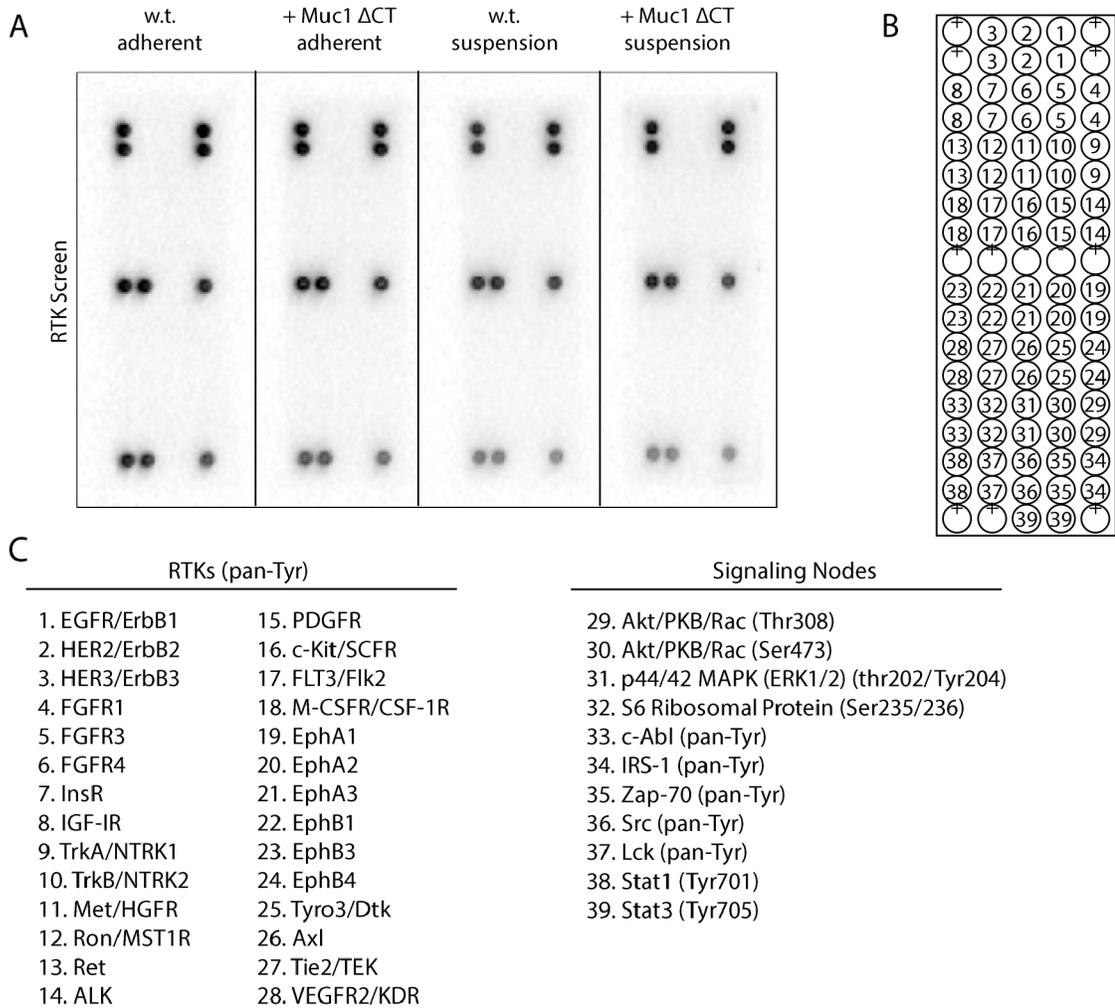
\*Contributed equally

---

**‡‡ Additional Supporting movies are available online at:  
<https://doi.org/10.1021/acsbio.7b00037>**



**Supplementary Fig. A1.1 Immunoblot.** A, Immunoblot showing the relative molecular weight of each of the Podxl mutants expressed transiently in HEK293T cells, n=2. Not that the 2/3 length Podxl is not detected by the anti-Podxl antibody due to deletion of the recognition domain during cloning.



**Supplementary Fig. A1.2 RTK Screen.** (A) Full array image of PathScan RTK antibody slide. Induced cells were grown for 24 hours in either adherent or suspension conditions before being lysed and analyzed per the kit protocol. (B) Schematic of RTK array slide. (C) Key for schematic of RTK array slide.

## SUPPORTING MOVIE LEGENDS

**Supporting Movie 1 - Time-course of Muc1  $\Delta$ CT expression.** MECs were allowed to adhere and spread on fibronectin coated glass substrates before inducing expression of the fluorescent Muc1  $\Delta$ CT moxGFP variant. As fluorescence intensity increases cells begin to adopt a rounded phenotype and eventually detach from the substrate. Elapsed time is relative to the time of doxycycline addition (scale bar 100  $\mu$ m).

**Supporting Movie 2 - Attachment followed by detachment.** An MEC expressing cytoplasmic EGFP and Muc1  $\Delta$ CT drifts into the center of the frame, attaches to the substrate and spreads at 37 hours. After a period of attachment the cell rounds up and detaches from the substrate at 58 hours. Elapsed time is relative to the addition of doxycycline. Elapsed time is relative to the addition of doxycycline (scale bar 100  $\mu$ m).

**Supporting Movie 3 – Division leads to detachment.** A MEC expressing cytoplasmic EGFP and Muc1  $\Delta$ CT divides at 37.5 hours following the addition of doxycycline. The two daughter cells detach from the substrate following division and float freely in the culture media. Elapsed time is relative to the addition of doxycycline (scale bar 100  $\mu$ m).

**Supporting Movie 4 – Attachment is required for division.** A detached MEC expressing cytoplasmic EGFP and Muc1  $\Delta$ CT attaches to the substrate and spreads at 32.5 hours. At approximately 40 hours the cell divides, and both daughter cells detach from the substrate and float freely in the culture media. Elapsed time is relative to the addition of doxycycline (scale bar 100  $\mu$ m).

## **SUPPORTING TEXT**

### **Cloning and Constructs**

Muc1  $\Delta$ CT was inserted into a second-generation lentiviral vector with a doxycycline inducible promoter and puromycin resistance cassette as previously described<sup>1</sup>. A lentiviral vector for stable, shRNA-mediated knockdown of Muc1 was constructed by inserting the shRNA sequence 5'- CCG GGA CAC AGT TCA ATC AGT ATA ACT CGA GTT ATA CTG ATT GAA CTG TGT CTT TTT G -3' into the pLKO.1 lentiviral backbone. The inducible PiggyBac transposon vector pPB tetOn was created first by PCR amplification and insertion of 5' and 3' PiggyBac ITRs from pPB 5'PTK3' (kindly provided by Dr. Alan Bradley, Wellcome Trust Sanger Institute, UK) into a minimal backbone for bacterial propagation and selection consisting of ampicillin resistance gene and ColE1 origin of replication PCR amplified from pBluescriptII KS+ (Stratagene). Between ITRs a puromycin resistance expression cassette consisting of SV40 promoter puromycin acetyltransferase gene and SV40 polyadenylation sequence was then inserted back to back with a Tet regulatable expression cassette consisting of hybrid hepatamerized Tet operator minimal CMV promoter, chimeric synthetic intron between human  $\beta$ -globulin, and immunoglobulin heavy chain introns, multiple cloning site (MCS) and polyadenylation sequence from bovine growth hormone. To clone Muc1  $\Delta$ CT into transposon vector, Muc1 N-terminus (fragment A) was amplified with PCR using a forward primer 5'- ACT ACT ACT AGT ACC ATG ACA CCG GGC ACC CAG T -3' and a reverse primer 5'- ACT ACT AGA TCT GGT CGT CGT CAT CCT TGT AAT CAG CA -3' using MMTV-Muc1 construct as the template (kindly provided by Sandra Gendler). The

central sequence of Muc1 harboring tandem repeats (fragment B) was excised from the same construct using BglII and ApaLI. The C-terminus of the Muc1  $\Delta$ CT (fragment C) was amplified using a forward primer 5'- ACT ACT GTG CAC AAC GGC ACC TCT GCC AGG GCT -3' and a reverse primer that introduced a stop codon immediately following the Muc1 transmembrane domain 5'- ACT ACT GAA TTC CTA GCA CTG ACA GAC AGC CAA GGC AAT -3'. The fragments A, B, and C were digested with SpeI/BglII, BglII/ApaLI, and ApaLI/EcoRI, respectively. These three fragments were subsequently cloned into SpeI/EcoRI-digested transposon vector. pPB tetOn SynMuc1 was prepared by inserting the DNA sequence 5'- TCA GGC ATA CTT TAT TGG CGA AAC CCA ACG GAA AGT GAT AGC ATC GTT TTG GCA ATT ATC GTC CCC AGT CTG CTC CTC TTG CTC TGC CTG GCT TTG TTG TGG TAC ATG CGC CGA CGA AGT ATG TAG GG -3' into the Bsu36I and EcoRI sites of pPB tetOn Muc1  $\Delta$ CT. pPB tetOn Muc1  $\Delta$ CT moxGFP was prepared by PCR of the moxGFP gene (Addgene plasmid #68070<sup>2</sup>) using the forward primer 5'- GGC ACC ATG GCA TGG TGT CCA AGG GCG AGG AGC TGT -3' and the reverse primer 5'- GGC ACC ATG GGC CTT GTA CAG CTC GTC CAT GCC GTG A -3' and non-directional insertion into the NcoI site of pPB tetOn Muc1  $\Delta$ CT.

For creation of pPB tetOn Podxl, the human Podxl cDNA (clone ID 9051823, Dharmacon) was amplified via PCR with forward and reverse primers 5'- GGC AGG ATC CAC GAC ACG ATG CGC TGC GCG CT -3' and 5'- GGC AGA ATT CCT AGA GGT GTG TGT CTT CCT CCT C -3'. The PCR product was subsequently inserted into the BamHI and EcoRI sites of pPB tetOn. pPB tetOn Podxl  $\Delta$ CT was generated through amplification of the Podxl cDNA with 5'- GGC AGG ATC CAT

GCG CTG CGC GCT GGC GCT CTC G -3' and 5'- CCG TCC TGA GGT TAG  
AGG CGC TGG TGG CAG CAG CCA TAG AGG -3' and insertion into the pPB  
tetOn vector as before. SynPodxl was created by amplification of the Oglycosylation  
rich ecto-domain from full-length human Podxl via PCR amplification with 5' - GGC  
AGG ATC CAT GCG CTG CGC GCT GGC GCT CTC G -3' and 5'- CCG TCC  
TGA GGT TAG CAT GCT GAA GCG GTC CTC GGC C -3' forward and reverse  
primers to add 5' BamHI and 3' Bsu36I restriction sites. Following restriction digest,  
the ectodomain was ligated with the synthetic membrane proximal domain and 21-  
amino acid transmembrane domain into the pPB tetOn vector. Podxl  $\Delta$ CT and  
SynPodxl were subcloned into the BamHI and EcoRI sites of pcDNA3.1(+) vector for  
transient expression and further cloning.

Modular SynPodxl constructs were generated by introducing Bsu36I restriction  
sites between residues T40D41 and A179E180 (relative locations at the end of the  
signal peptide and 1/3 through the O-glycosylation rich region) by site-directed  
mutagenesis with the following primers: T40D41 forward 5'- CCT AAG GAC TCA  
TCT AAC AAA ACA GC -3', reverse 5'- CGT AGT AGT CTG GGT TGC -3';  
A179E180 forward 5'- CCT AAG GAA CAT CTG ACG ACC CCT -3', reverse 5'-  
TGC CTT AGT GGA TGT GAG -3'. One mutant was generated for each new  
restriction site, and the resulting plasmids as well as pcDNA3.1(+) SynPodxl were  
digested with Bsu36I and BamHI to yield 2 fragments from each mutant and the  
vector with transmembrane sequence from SynPodxl. 2/3 SynPodxl was constructed  
by ligation of the 129 bp fragment from T40D41 and 865 bp fragment from 179E180  
into the vector. 4/3 SynPodxl was constructed in the same manner from the 537 bp

fragment from A179E180 and 1.3 kbp fragment from T40D41.

## REFERENCES

1. Paszek, M. J. *et al.* The cancer glycocalyx mechanically primes integrin-mediated growth and survival. *Nature* **511**, 319–325 (2014).
2. Costantini, L. M. *et al.* A palette of fluorescent proteins optimized for diverse cellular environments. *Nat. Commun.* **6**, 7670 (2015).

## **Appendix B<sup>§§</sup>**

### **Supporting Information for Chapter 3**

#### **TITLE**

High-speed device synchronization in optical microscopy with an open-source hardware control platform

#### **AUTHORS**

Marshall J. Colville, Sangwoo Park, Warren R. Zipfel, Matthew J. Paszek

---

**§§ Additional Supporting movies are available online at:  
<https://doi.org/10.1021/acsbiomaterials.7b00037>**

## **Supplementary Note 1: Hardware description**

The controller electronics are divided into analog, digital I/O, in-circuit serial programming (ICSP) and microcontroller (MCU) subcircuits. Supplementary Fig. 1 provides a conceptual schematic of the analog subcircuit designs. Supplementary Fig. 7 shows the populated PCB assembly highlighting the various subcircuits in the physical layout. The complete schematics and design files are available at <https://github.com/mjc449/SAIMscannerV3.git>. Supplementary video 1 shows a brief demonstration of high-speed beam steering and excitation synchronization.

### **Waveform generators**

The analog waveform outputs consist of a pair of matched subcircuits and are designed to provide accurate synchronization while still being independently controllable. To accomplish this each has independent DC references and direct digital synthesis (DDS) components. The output frequency and phase of each DDS is set by the MCU with ranges of  $\sim 0.1$  to  $12.5 \times 10^6$  Hz and 0 to  $2\pi$ , respectively. Because the DDS units share a 25 MHz master clock the relative frequency and phase settings are matched between the units, preventing drift. The DDS units have independent serial data lines to the MCU so that each can be programmed individually while maintaining common frame synchronization and serial clock signals. In this way the phase, frequency and waveform type for each unit can be updated simultaneously, ensuring consistent phase values and timing. The waveform output of the DDS units has a positive bias and relatively small amplitude of  $\sim 650$  mV which is insufficient to directly control most peripheral devices. To precondition the DDS output the signal amplitude is increased and a bias is applied by a precision amplifier for each channel.

The amplification and bias are controlled by trimming potentiometers on the circuit board. The waveform output amplitude from the controller is proportional to the conditioned waveform reference with 16-bit resolution over the reference voltage range. The tunable amplifier circuit therefore provides a means to set a constant DC bias to the waveform reference signals as well as limit the maximum and minimum output values without decreasing the output resolution. For applications where a high-resolution DC signal is required the waveform generators can be disabled by setting the DDS units in reset (midscale output). Each of the two waveform outputs are controlled by independent dual-channel digital to analog converters (DACs). For each output's DAC one reference channel is the conditioned DDS. The second DAC channel uses a precision, trimmable 10 V reference and operates in a bipolar mode to provide a full-scale range of -10 to 10 V DC. At the output stage the two channels of each DAC are summed in a current to voltage output amplifier. The dual-channel DACs share a 16-bit parallel port on the MCU and common output register load signal. The DACs can be programmed independently or simultaneously by channels, allowing both independent and concurrent updates. Values are written to the outputs simultaneously by the common load signal, eliminating delays and ensuring output synchronization.

### **Bipolar analog outputs**

In addition to the precision, 16-bit waveform analog outputs we have included a pair of 10-bit analog outputs with user adjustable references. Each channel has an independent 10 V reference and operates in bipolar mode. The references are connected to the DAC's reference inputs through a user-selectable jumper and

trimming potentiometer. The jumper can be selected to bypass the center tap on the potentiometer to set the DAC output to a full-scale range of -10 to 10 V.

Alternatively, the jumper can be set to connect the potentiometer center tap to the DAC channel's reference input. In this configuration the potentiometer forms a voltage divider, setting the reference value ( $V_{ref}$ ) from  $\sim 0.1$  to 10 V. The DAC output range in this mode is limited to  $-V_{ref}$  to  $+V_{ref}$ , providing the full, 10-bit resolution across a wide spectrum of applications. The DAC is connected to one of the MUC parallel ports to maximize update speeds.

### **8-channel 0-10 V analog outputs**

The 8-channel voltage outputs are generated by pair of 4-channel DACs with a common 10 V reference. The DACs share a parallel data port from the MCU and data is written to each channel individually. However, both DACs are connected to a common output register load signal. Channels can be set individually, updating as soon as the new data is written, or simultaneously where the data for any number of the channels is loaded into the DACs then all outputs are set simultaneously. At the board level there is also a 10 V digital output and external power supply passthrough on the same pin header as the analog outputs. We use the 10 V digital line as a global shutter for our AOTF.

### **Digital I/O**

We define 2 types of digital I/O for the purposes of this discussion: general purpose I/O (GPIO) and triggers. GPIO lines are those which the MCU reads based on polling the state of the pin within the thread of execution. Triggers are pins that are enabled as hardware interrupts. Interrupt events (change of pin state) trigger the MCU

to halt its current thread of execution and jump to the interrupt service routine (ISR) associated with the interrupt source. When the ISR completes, the MCU continues the main thread of execution from where it left off. Our controller has 3 trigger pins that can be set programmatically to function as either triggers or GPIO and 5 additional GPIO pins. Of these pins, 4 GPIO pins and 2 triggers are +5.5 V tolerant and can operate on 3.3 V or 5 V logic without the need for logic-level conversion. There are an additional 4 digital I/O lines on an RJ45 connector. These can be set programmatically to GPIO, triggers, or utilize the MCU serial communications hardware for linking other devices, such as a second controller or other custom peripherals using SPI or UART.

### **Connectors**

We have avoided the use of obscure or proprietary connectors in the design of the controller (Supplementary Fig. 7). The waveform and bipolar analog outputs have SMA type connectors which can easily be adapted to BNC, SMB or many other coaxial types. The 8-channel analog outputs, digital I/O and ICSP have 0.1" pin headers at the board level for maximum versatility. For example, in our completed controller the digital I/O are broken out into 2 DB-9 connectors, 4 +5.5 V tolerant GPIO lines on one and the triggers and remaining GPIO line on another. The 8-channel analog output, 10 V digital pin and external power supply pins have been connected to a DB-25 in a pin arrangement matching our AOTF analog control connector.

### **Supplementary Note 2: Mechanical considerations in azimuthal scanning**

Galvanometer scanning mirrors are economical and easy to integrate into optical systems such as the microscope demonstrated in this study. Commercial solutions that come packaged with appropriate driver circuitry are available from several suppliers and can even be salvaged from retired confocal and other laser scanning systems. While acousto-optic deflectors (AODs) have much faster response times, the challenge of integration of two orthogonal AODs for 2D beam scanning and higher price tag make them less attractive to most builders of circle scanning microscopes. One of the significant drawbacks of galvanometer scanning mirrors is that they are mechanical devices and as such have much slower responses to changes in position than AODs. Furthermore, the momentum of the mirrors requires that the driver electronics can quickly supply large currents proportional to the step size of the command voltage when making a large step between points such as in a discrete scan at a TIRF radius.

Discrete scans ideally should match the number and frequency of scan points to some factor of the small-step response time ( $T_s$ ), the 99% settling time, of the galvanometers. In the case of those used in our microscope this is 120  $\mu$ s. To maintain the mirrors in constant motion, the time between steps along the circle must be at least  $T_s/2$ , or 60  $\mu$ s in our case, however faster sampling will improve the quality of the waveform scanned. Ideally the update period would be  $\ll T_s$ .

Another consideration in constructing a circle scanning microscope is the circular frequency. If the camera frame rate is on the order of the time to complete a single azimuthal scan the circular frequency must be matched to the frame rate or the benefits of circle scanning in terms of eliminating interference in the sample plane are decreased. When they are not matched for some portion of the exposure only a part of

the azimuthal scan is completed and any variation in the excitation profile will be reflected in the fluorescence image. Matching the camera framerate to the scanning frequency is technically challenging. A simple solution to this problem is to maintain a scan frequency that is much greater than the minimum framerate such that the laser beam completes multiple scans in any given exposure. In this way the partial scan at the end of exposure has a negligible effect on the excitation profile.

To keep the mirrors settled on the desired waveform, the time between changes in the command signal must be held constant or fluctuations in the update rate will be reflected in the motion of the mirrors. Considering the case of a USB HID interface with the controller or driver circuitry this is difficult to achieve, as interrupt exchanges occur at a rate of 1 kHz. If the analog voltage, as in the case of our controller, has 16-bit precision this equates to a maximum of 16 points per transfer (64 bytes per packet / (2 bytes per axis \* 2 axes) at 1 kHz, or 62.5  $\mu$ s per scan location. This also assumes that there are no missed transfers and that the latency in processing time of each report is less than the time between the report receipt and the previous change in the command signal. The constant timing constraint is further complicated by other tasks the controller is required to perform, and a high-priority timer interrupt is the most reliable solution to this problem. While interrupts take priority over the current computation, care must be taken in programming to ensure that the controller spends the minimum amount of time in the interrupt service routine (ISR) or other processes may incur an unacceptable delay while the ISR completes. Furthermore, a race condition exists between the priority level of scanning mirror updates and USB packet processing, as neither is more important than the other. With these constraints in place

it is infeasible to construct a system wherein the controller relies on the instrument computer for continuous transfer of the scan coordinates.

To circumvent the problems associated with USB transfers the entire set of discrete scan coordinates can be preloaded into the controller before the experiment begins. In this case the controller's memory should ideally store all coordinates for the experiment, and at minimum the integer number of circles to be scanned at the reliable transfer rate. Take, for instance, a real-world scenario in which the controller can reliably receive and respond to 1 in 4 in reports, or 16 kB/s and each circle contains 32 points. In this case 2 successful transfers are required per scan radius, limiting the system to a minimum of 8 ms per exposure neglecting processing time for the new radius, system update time and settling time on the new scan waveform.

A more reliable mode of operation is to have all waveforms preprogrammed in the controller before the experiment begins. For an acquisition of 32 points per radius (as in the discrete scans presented in the main text, which is still insufficient to keep the mirrors settled on the scan waveform as demonstrated in Supplementary Fig. 3) with 32 individual radii 4 kB of memory is occupied by the discrete scan points alone. In a microcontroller with limited memory the number of discrete scan points and number of scan radii quickly becomes the limiting factor as far as the experimental complexity that can be achieved.

In our controller the memory requirement is reduced to 4 bytes per scan radius when using the integrated waveform generators. The integrated direct digital synthesizers (DDS) run on a dedicated 25 MHz clock. Regardless of the system state the waveforms have a constant frequency and phase, the result of which is a constant

circular scan of the laser independent of processor load. This also simplifies the process of settling the mirrors on the new scan radius during an update. By referencing the output DAC to the waveform input any changes are immediately reflected in the command signal sent to the mirrors. The result is a system in which the mirrors are settled on the new scan radius with a delay of  $T_s$ , which is the shortest possible for a mechanical mirror system.

### **Supplementary Note 3: Controller induced latency**

The hardware controller introduces latency in the signal transmission between the camera and the excitation source (in our case the AOTF). This delay can be divided into 2 sources, the electronics and the firmware. In our system the AOTF global blank is an analog input of 0-10 V with 10 V being fully “open”. Even if we were to bypass the controller, additional electronics would be required to step up the 3.3 V camera logic to 10 V. Similarly, the microcontroller operates on 3.3 V logic, so we implement global blanking through a simple logic level converter made up of a MOSFET and Zener diode voltage regulator on the PCB, eliminating the need for external electronics. This was a conscientious choice to optimize performance. The logic level converter simply follows the state of a single I/O pin and has a total turn-on/off time of less than 75 ns, which we consider to be negligible. The firmware, on the other hand, must enter the ISR associated with the camera’s input before switching the global blank pin, which accounts for most of the delay introduced by the hardware controller. In Supplementary Fig. 2a the green trace tracks the controller global blanking output at the end of exposure and yellow trace is the camera fire signal.

Although not explicitly stated in the text, the actual delay introduced by the controller is approximately  $2.5 \mu\text{s}$ , which can be seen at the bottom of the figure in the line 1 to 4 transition time. This data was collected directly from the controller inputs and outputs, whereas the timing data in the main text reflects the actual shuttering time of the excitation laser. Connecting the camera directly to our AOTF would only improve the latency by approximately 20% at the start of exposure and 25% at the end of exposure. The remaining latency is due to the AOTF driver hardware and cannot be avoided.

With the camera connected directly to the blanking input it becomes more complicated to implement a user-selectable override. In fact, we typically leave the cameras idle or powered off when performing alignment and calibration operations at the beginning of each imaging session. We do this to avoid damaging the detectors but need to have the excitation enabled to observe the laser beam which would not be possible with the camera blanking scheme alone. Another consideration is the case of multiple independent excitation sources that are not controlled through a central device such as an AOTF or LED combiner. In such a system the camera alone could not discriminate between the excitation source required for a given frame and all the sources would need to be either on or off. With the controller described in our work the multichannel output can be used to individually modulate each source independently for a given frame. While this would carry an additional latency of up to  $50 \mu\text{s}$ , this is still much faster than what is achievable through software using USB communications.

#### **Supplementary Note 4: Software development**

Software control is a popular option for many researchers because of its ease of implementation. For many peripheral devices all that is necessary is a driver installation and/or configuring the control software. Hardware control can be significantly more difficult to accomplish. Our controller is designed to be both a development platform and a general solution for instrument control. The CAD files and bill of materials are freely available in our project repository and the populated board can be purchased fully assembled from several sources such as [circuithub.com](https://www.circuithub.com), where we have posted the necessary project files<sup>1</sup>.

The controller firmware is available through our project repository and inexpensive programming hardware can be purchased from Microchip to flash the latest firmware into the MCU (PICkit 4, ~\$48). We have also created a graphic interface that simplifies creating and executing circle scanning experiments as well as general system control (Supplementary Fig. 8). The software handles all aspects of communication and control of the hardware, including calibration, alignment and synchronization.

The features and applications of the controller presented in this work are only a subset of its capabilities and we envision the hardware platform being useful in many other applications. The success of many open-source projects is the collaborative input and development efforts of users and contributors. Through open-source development our project can benefit from the expertise and feedback of users in fields other than our own, expanding on the capabilities and functionality of the controller.

The host software is divided into 2 layers, an application programming

interface (API) which we have written in C/C++ using the HIDAPI library and compiles to a driver (.dll), and the graphical user interface written in C/C++ and Qt. Each of these layers builds on top of the other in the sense that the controller with firmware does not require the API or software, the API requires the firmware but can be integrated into user code without the rest of the software, and the software requires a controller with the firmware and driver. Finally, the driver and software have provisions for direct, packet level communication with the controller for firmware development and prototyping without needing to modify the GUI or driver code. We have adopted this model to provide a variety of entry points for working with our instrument controller. For instance a user wanting to develop a raster scan function in the firmware does not need to be familiar with HID protocol specifics or programming in Qt, while someone wanting to integrate the controller into a custom GUI in another programming language can access the controller functions as human-readable function calls in the API without needing to know the organization of data in the packets sent back and forth on the USB connection.

### **Supplementary Note 5: Incidence angle calibration**

Both SAIM and MA-TIRF rely on accurate knowledge of the angle of incidence of the excitation beam. Calibration of the beam deflection as a function of command voltage is critical in both techniques. In the past we calibrated the system with a protractor placed on the microscope stage, measuring the beam deflection at several voltages. With the development of the graphical interface software for our controller, we built in an image-based calibration tool for circle scanning. A piece of

fluorescent acrylic is placed on the microscope stage directly above the objective and the laser is scanned in the plastic. An inexpensive webcam connected to the instrument computer captures images of the laser at various waveform output amplitudes directly through the controller software. The software then finds the edges of the scanned laser in the image and calculates the effective calibration value based on the index of refraction of the acrylic and the imaging medium. For SAIM experiments the imaging medium was assumed to have an index of refraction of 1.34. For MA-TIRF the calibration was performed with a sample index of refraction of 1.52 corresponding to the index of refraction of the glass coverslip.

The range of angles used in MA-TIRF is beyond those measurable using the laser emission from the objective. Before MA-TIRF experiments, we verified the calibration by measuring the fluorescence of an adsorbed monolayer of IgG-Alexa Fluor 568 (ThermoFisher). As the incidence angle approaches the critical angle the excitation intensity at the glass to water interface increases rapidly, and past the critical angle the intensity falls off<sup>2</sup>. The fluorescent monolayer was imaged from sub- to super-critical angles in steps of 0.1 degrees. The critical angle was determined as the angle at which the maximum fluorescence emission was observed. The experimentally determined critical angle was then used to verify the image-based calibration.

### **Supplementary Note 6: SAIM analysis**

The SAIM optical model is based on the fluorescence interference-contrast microscopy model and has been reported many times in the literature<sup>3-5</sup>. We use the

simplified mathematical representation of Carbone et. Al. in a highly optimized fitting program written in C++ with the Math Kernel Library (Intel). The program uses an unbounded trust-region solver, and as such the choice of initial parameter values can lead to erroneous outputs corresponding to the nearest local minimum of the nonlinear least squares problem rather than the true value. For the samples used in this study we set the initial value of height to the average values known from the literature<sup>6</sup>. We have found that the solution is relatively insensitive to the initial values of intensity (A) and background (B). However, a good approximation of intensity is 80% of the difference between the highest value and lowest value in the image series for a given pixel and background is simply the lowest value. Initially we implemented the analysis routine in a multithreaded MATLAB application, however after multiple rounds of optimization analysis of a 2048 by 2048 pixel SLB image set would take between 4 and 6 hours on a 6-core Intel Core i7 5820k processor running at 3.3 GHz. With careful optimization and the low-level memory access provided by C++ we can analyze the same dataset in < 30s. The analysis code supplied in the software repository is a simple program for analyzing a single dataset, however we have written it in such a way that the processing functions are contained in a single class for portability.

In this work we define signal to noise (S/N) as the root mean square background subtracted prediction divided by the root mean square residuals, or

$$S/N = \frac{(f(\theta_i, \beta) - B)_{rms}}{(I_i - f(\theta_i, \beta))_{rms}}$$

Where  $f(\theta_i, \beta)$  is the optical model,  $\theta_i$  is the vector of incidence angles,  $\beta = (A, B, H)$

is the parameter vector,  $B$  is the background, and  $I_i$  is the observed intensity in frame  $i$ . Additionally we have found the combined evaluation of  $R^2$  and S/N for a given pixel a better metric of the quality of the fit than  $R^2$  alone.

### **Supplementary Note 7: Firmware architecture**

The controller firmware used in this study was developed for running circle-scanning acquisition sequences, and as such has been optimized for high speed, sequence-based applications. The controller's functions in these applications can be generally divided into acquisition and non-acquisition tasks. A few examples of non-acquisition tasks are changes in the waveform frequency, phase offset, opening and closing a mechanical shutter or manually adjusting a laser intensity when setting up an experiment. Acquisition tasks are those that the controller should execute autonomously while collecting data such as changing the scan angle or switching the excitation pattern between frames.

The firmware architecture is of a mixed design that uses a combination of interrupt-triggered synchronization functions and polling for USB communication. The controller, once initialized, enters an infinite loop with only 2 functions. The first checks for a USB packet from the computer, the second resets the watchdog timer. When a packet is received it is processed based on the first byte, which is the command code, in a switch case statement. For every packet received the controller sends a response that can be checked by the computer-side software. Simple commands are echoed, while other commands that request information about the controller settings, memory and state, will have a more complex response. For the

reasons explained in the main text polling for USB exchanges is not appropriate for timing or high-speed operations such as excitation synchronization or inter-exposure output changes. In these cases, the controller uses a combination of hardware (I/O pin) and timer interrupts to transfer execution from the main control loop to an appropriate interrupt service routine (ISR).

Although they add complexity to the firmware design, interrupts are an invaluable tool in achieving the level of precision control required in high-speed acquisitions. For a schematic representation of the control flow see Supplementary Fig. 10. To summarize, in a polling-based approach the controller continually executes a single, serial thread. At one or more points within this thread it evaluates the system state, including changes on input pins, elapsed time since an event occurred, or the result of incrementing some loop counter. Each of these must be checked in series which, for many checks, can add up to a significant amount of elapsed time to complete. Furthermore, at some point in the loop the controller must perform any action required as a result of these checks before the loop can complete and the checks be performed again. One of the most significant drawbacks to this approach is the indeterminacy in time to execute the control loop. If no changes have occurred the loop may complete in just a few processor cycles, while if multiple actions are required the loop may take many hundreds or thousands of processor cycles, depending on the complexity of the functions. Interrupts, on the other hand are called by hardware independent of the event loop, essentially constantly being monitored. While there is overhead associated with entering and exiting an ISR this is generally small and predictable with good programming practices. Furthermore,

interrupts in the microcontroller we have used can be assigned priority levels and nested, providing a simple mechanism for a high priority event to interrupt a low priority event which has itself interrupted the main event loop.

As an example of the interrupt system, the camera exposure ISR is enabled or disabled via a USB message from the computer. Once enabled, a state change on the trigger line will cause the ISR to execute regardless of the rest of the system. The camera exposure ISR can be disabled at any time by sending the appropriate command to the controller. The camera exposure ISR is complex, as it not only controls the excitation blanking, but also handles progressing through acquisition sequences, including changing scan radius and excitation intensities of multiple lasers. Each of these subfunctions can be directly manipulated over the USB communications, however. For instance, a routine adjustment to the excitation when first examining a sample could proceed as:

1. Disable the camera exposure ISR by issuing the “Fire off” command (0x51)
2. Set the global blank high by issuing the “Enable excitation” command (0x40)
3. Increase the excitation intensity of laser 1 to 25% with the “Set excitation channel” command (0x42 0x01 0x00 0xFF) where the parameters 0x01 correspond to channel 1 and 0x00 0xFF the new intensity value
4. Reenable the camera exposure ISR with the “Fire on” command (0x50)

Note that it is not necessary to disable the ISR before changing the excitation intensity. Changes to the scan radius, waveform frequency, phase, output etc. can all

be made in a similar fashion.

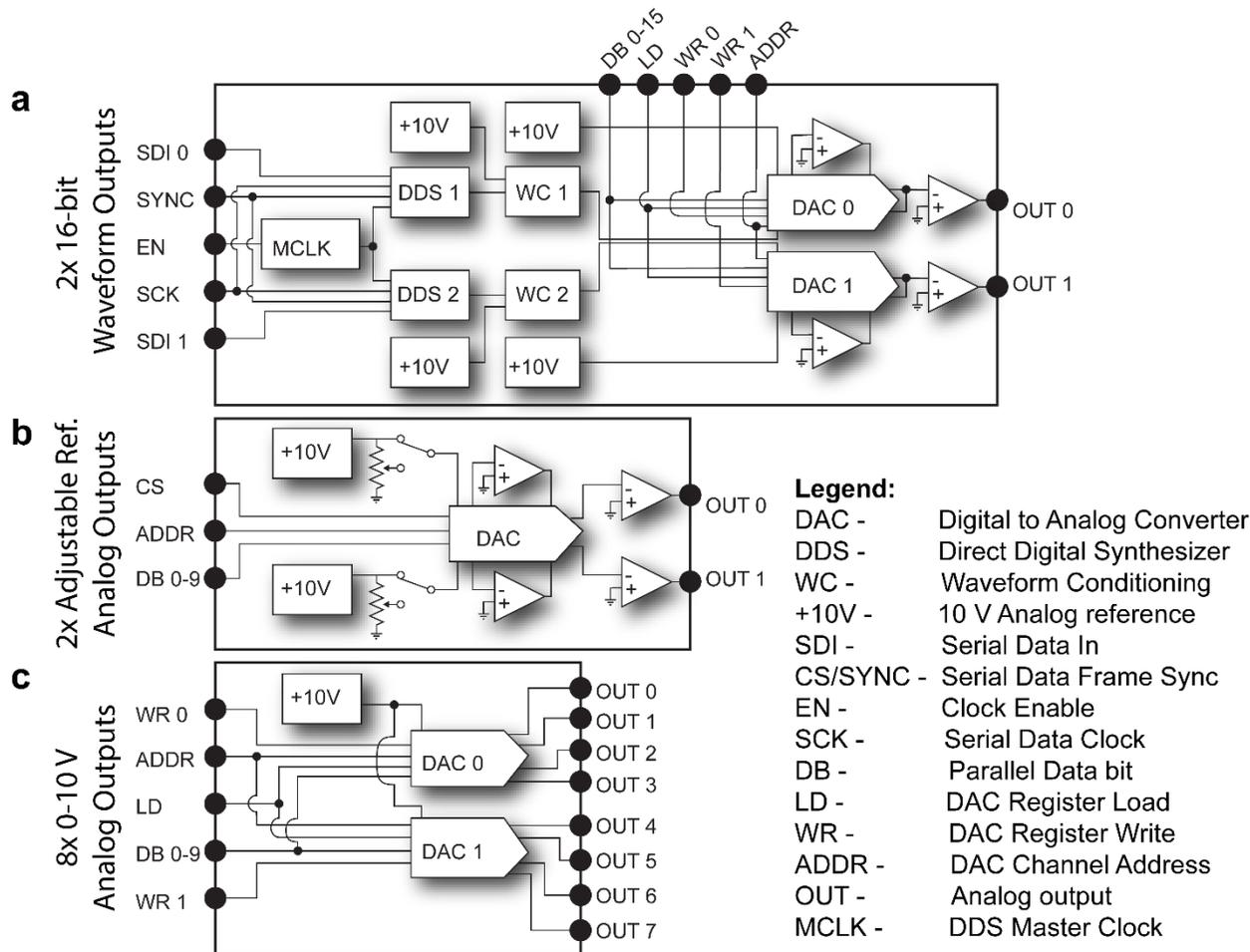
Acquisition sequences, called experiments in the firmware code, are organized in the controller memory as linked lists (Supplementary Fig. 11). Each node in the list contains pointers to a set of waveform amplitudes (named a sequence in the firmware) and an excitation setting (named a profile profile). The firmware has space statically allocated for up to 32 sequences of 128 angles each and up to 32 excitation profiles. Excitation profiles and angle sequences are sent to the controller by the instrument computer prior to building the experiment. Experiments are programmed by adding nodes to the experiment list, and up to 32 separate experiments can be stored in the controller. Nodes are dynamically allocated at the time of creation and destroyed when popped from the list or when the experiment is deleted. The length of an experiment is then limited to the free memory in the controller.

At acquisition time the instrument computer issues the command to start the experiment. Once received, the controller sets the waveform outputs to the initial values in the first scan sequence, the excitation levels to those in the first profile, sets the global blanking output low and enables the hardware trigger interrupt. At the first rising edge from the camera's exposure signal on the trigger the global blank is raised to the high level and the interrupt is changed to detect the falling edge. When the exposure ends and the trigger detects a falling edge the global blank is set low. The controller fetches the next x- and y-waveform amplitudes from memory and writes these values to the corresponding DACs. A timer is started that blocks inputs at the trigger until a user-defined system settling time has elapsed, preventing exposure of a new frame before the system has settled. We typically set this to the galvanometer

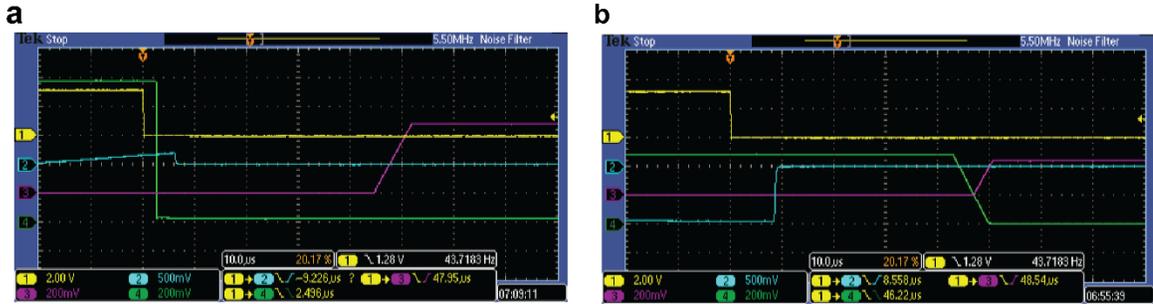
mirror settling time. If the end of a scan angle sequence has been reached the controller fetches the next excitation profile from memory and makes the appropriate updates to the output values. Finally, the trigger interrupt is reset to detect rising edges and the controller waits for the beginning of the next exposure.

## REFERENCES

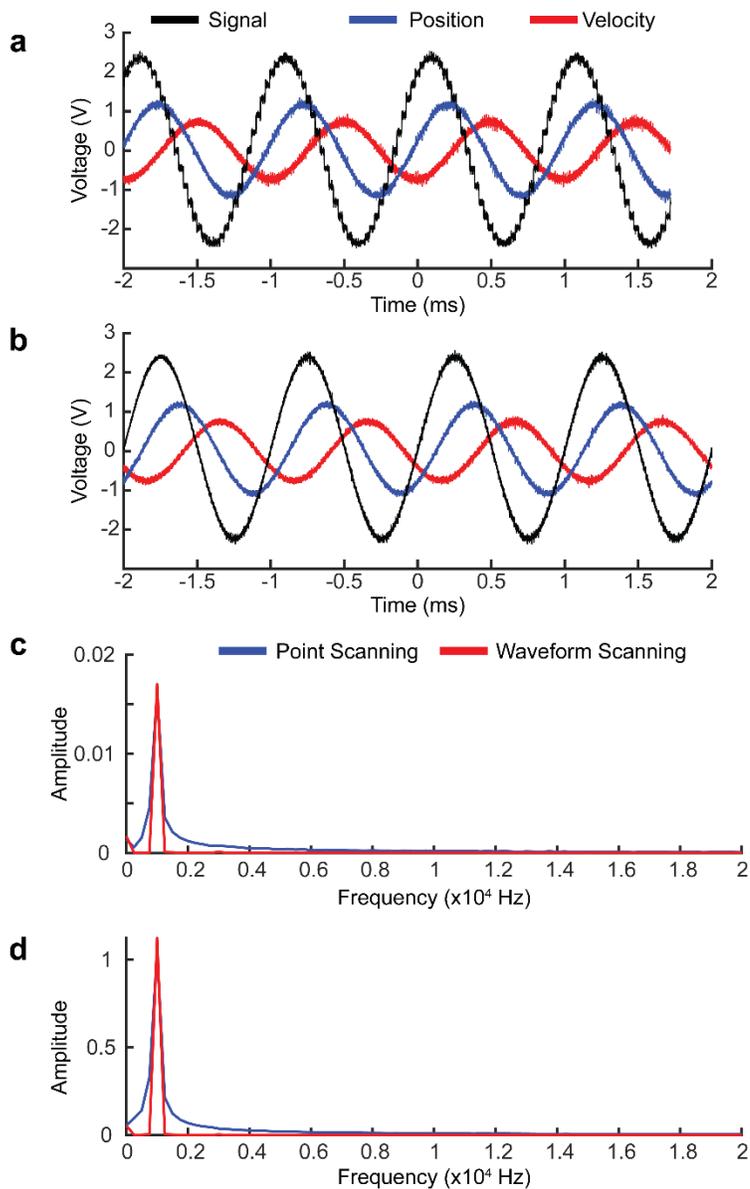
1. mjc449/SSv3\_1 · CircuitHub. Available at: [https://circuitHub.com/projects/mjc449/SSv3\\_1/visions/16275](https://circuitHub.com/projects/mjc449/SSv3_1/visions/16275). (Accessed: 14th December 2018)
2. Fu, Y. *et al.* Axial superresolution via multiangle TIRF microscopy with sequential imaging and photobleaching. *Proceedings of the National Academy of Sciences* **113**, 4368–4373 (2016).
3. Paszek, M. J. *et al.* Scanning angle interference microscopy reveals cell dynamics at the nanoscale. *Nature Methods* **9**, 825–827 (2012).
4. Liu, J. *et al.* Talin determines the nanoscale architecture of focal adhesions. *Proceedings of the National Academy of Sciences* **112**, E4864–E4873 (2015).
5. Lambacher, A. & Fromherz, P. Fluorescence interference-contrast microscopy on oxidized silicon using a monomolecular dye layer. *Applied Physics A: Materials Science & Processing* **63**, 207–217 (1996).
6. Kanchanawong, P. *et al.* Nanoscale architecture of integrin-based cell adhesions. *Nature* **468**, 580–584 (2010).



**Supplementary Fig. A2.1 Expanded schematic diagrams of the controller analog subcircuits from Fig. 1. (a)** The dual waveform generators share a master clock, sync and serial clock lines enabling simultaneous updates. The output amplitude and offset are individually adjusted by matched dual-channel DACs with a common load signal. The AC and DC outputs for each channel are summed in the final amplifier stage. **(b)** Auxiliary DAC outputs have individual, high precision 10 V references with selectable voltage dividers to tune the full-scale output range from  $\pm 0$  to  $\pm 10$  V. **(c)** Multichannel precision 0 to 10 V outputs are generated by a pair of matched 4 channel DACs sharing a data bus and load line. Each of the 8 outputs can be updated individually or all 8 simultaneously.

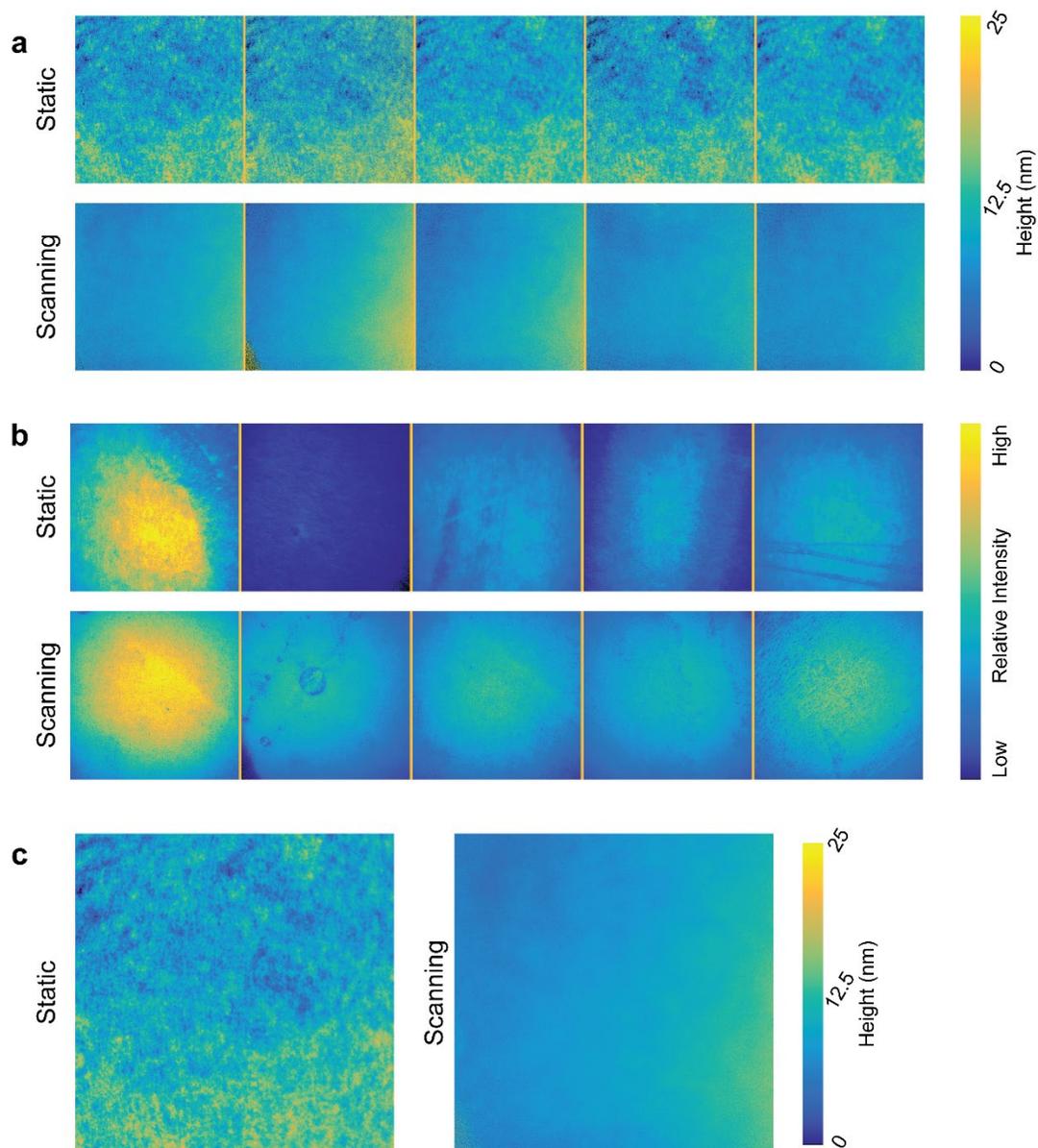


**Supplementary Fig. A2.2 Example timing waveforms from a sequence acquisition.** (a) The camera's exposure signal (Ch. 1, yellow) marks the end of the frame, triggering the controller to close the excitation shutter (Ch. 4, green). Immediately after shuttering the excitation the scanning mirror command waveform amplitude (Ch. 2, blue) is updated. Finally, the AOTF laser 0 channel (Ch. 3, pink) intensity is changed. The total update time is defined as the delay between the exposure signal trigger and the excitation amplitude settling on the new value. (b) Same as a with Ch4 (green) showing the AOTF laser 7 signal. All 8 channels of the AOTF are settled at the same time following the end of frame and the controller is ready to begin the next exposure in 50  $\mu$ s.



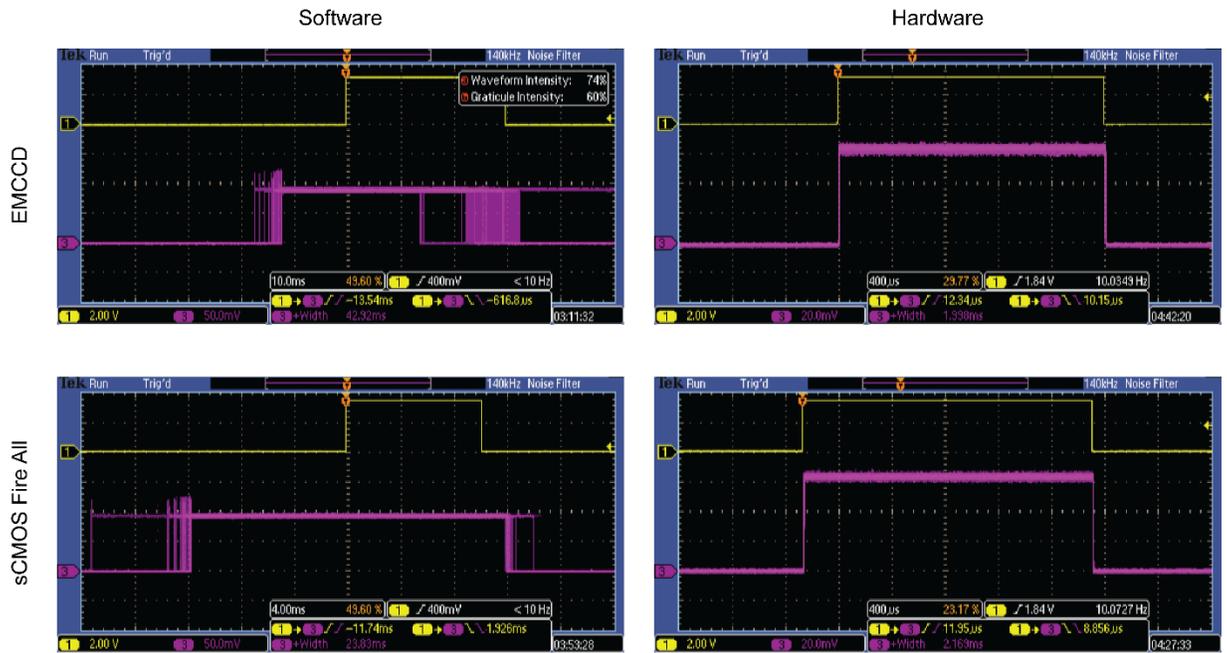
**Supplementary Fig. A2.3 Galvanometer scanning mirror response to discretized (point scanning) and continuous waveforms in circle scanning experiments.** (a) The scanning mirror is driven with a 32 point discretized sine wave at  $\sim 1$  kHz (black trace) while its position (blue trace) and velocity (red trace) are monitored. (b) Same as a when the command signal is replaced with a 1 kHz sine wave generated by the integrated waveform generators in the controller. Command signal amplitude in a and b is typical of a TIRF experiment. (c) Frequency spectra of the mirror position in a small angle scan of  $1^\circ$ . (d) Same as c with a large angle scan corresponding to a typical TIRF experiment.

Increased noise in the position trace from **a** compared with **b** manifests as a broadening of the frequency spectrum in **c** and **d**.

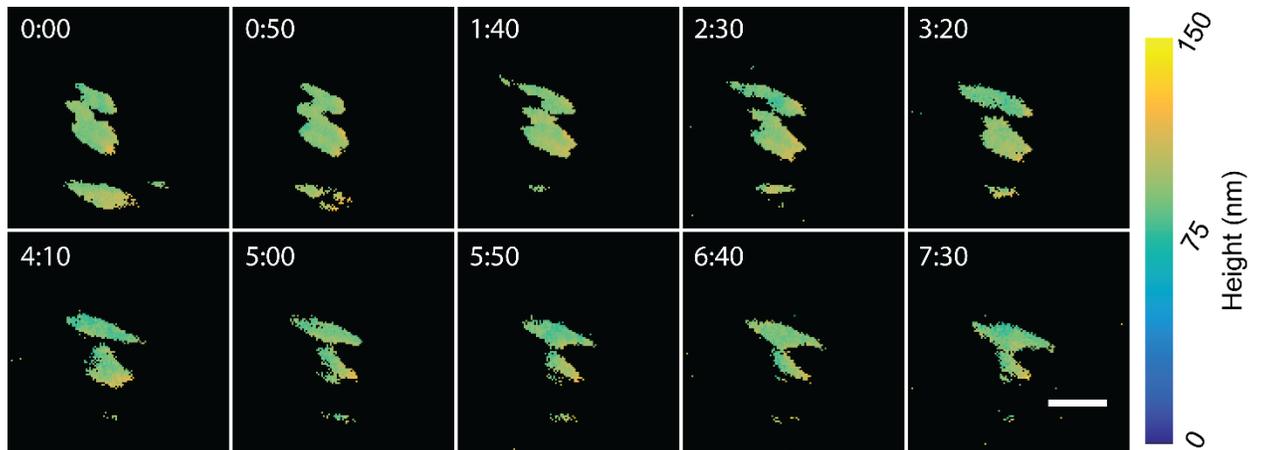


**Supplementary Fig. A2.4 Laser speckle and fringing depend on incidence angle and lead to artifacts in SAIM reconstructions.** (a) SAIM height reconstructions of 5 independent supported lipid bilayer regions acquired with static beam (top) or circle scanning (bottom) excitation schemes. (b) Relative intensity images (fit parameter  $A$  of the model function) of each bilayer region in a. Details in intensity images correspond to features in the bilayer resulting from labeling variations and bilayer defects, and are similar to a widefield image. (c) Pixelwise

average height of the 5 regions in **a**. Features in the average height image are the result of pixelwise incidence angle dependent excitation variations from laser speckle and fringes. All images are the full sensor area of 2048 x 2048 pixels, corresponding to a 147.9 x 147.9  $\mu\text{m}$  field of view.



**Supplementary Fig. A2.5 Excitation shuttering time comparison between software (left) and hardware (right) control.** Ch1 (yellow) camera exposure signal; Ch3 (pink) excitation intensity collected by a photomultiplier tube placed directly in the excitation path. Traces from 100 individual exposures are overlaid to highlight the variability in excitation timing.



**Supplementary Fig. A2.6 Time lapse SAIM reconstructions of the focal adhesion protein Zyxin.** A live HeLa cell expressing mEmerald-Zyxin was imaged at 5 second intervals using the circle scanning excitation scheme. Scale bar 2.5  $\mu\text{m}$ ; time stamps are min:sec.

# SSv3.1 Device Overview

Dual 10 bit high-speed bipolar analog outputs with 0-10 V adjustable references.

Dual waveform outputs with high-speed independent per-axis amplitude and DC bias control.

- +/- 10 V range
- sine, triangle, and square wave
- 16 bit amplitude and offset resolution
- 0.1 - ~1.1 kHz waveform frequency (can be unlocked to >0.5 MHz)
- $2\pi$  fully adjustable phase range

8-channel 10 bit 0-10 V analog outputs.

- Header designed for direct connection to AA Opto-Electronics MDSnC AOTF driver
- Easily reconfigurable for other applications
- Reset and +24 V supply headers
- 10 V digital global blank with status LED header

4 reconfigurable DIO pins.

- 3 interruptible inputs for hardware synchronization
- 2 +5.5 V tolerant interrupts

USB HID interface for broad compatibility

Reconfigurable serial communications port (UART, SPI)

ICSP port for firmware updates

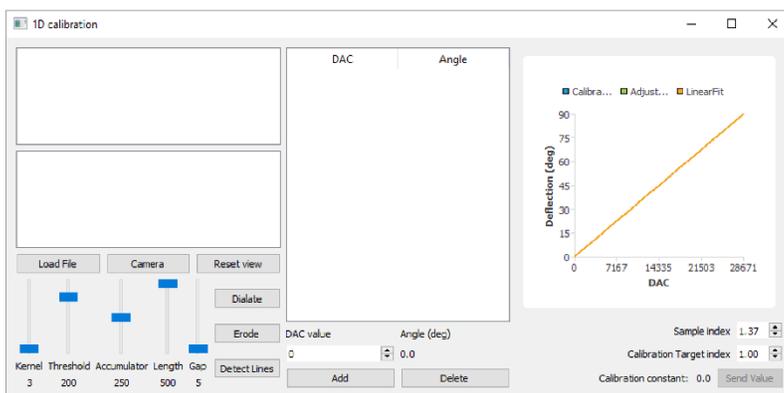
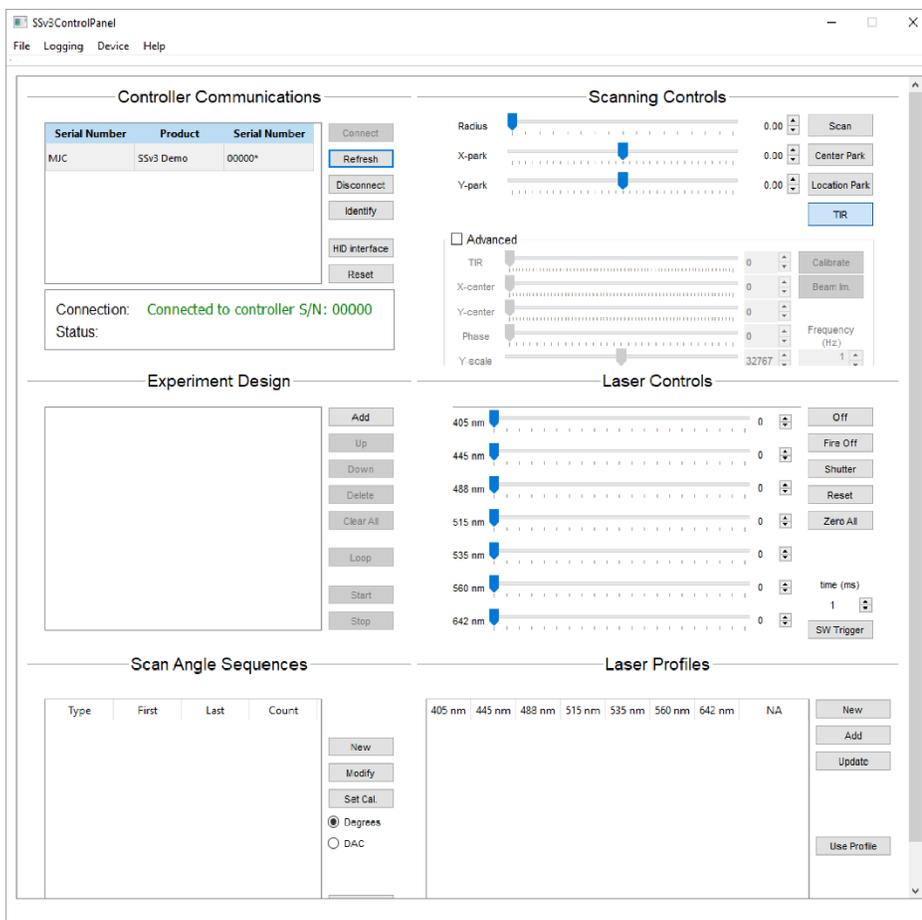
4 GPIO pins

- +5.5 V tolerant inputs

**Supplementary Fig. A2.7 Circuit board layout. All the controller electronics are integrated into a single, compact PCB.** The package sizes were chosen to strike a balance between small form factor and serviceability. Hand-building or repairing the electronics is possible with a basic, manual surface-mount device rework station.



**Supplementary Fig. A2.8 The complete controller assembly with relevant connectors.**



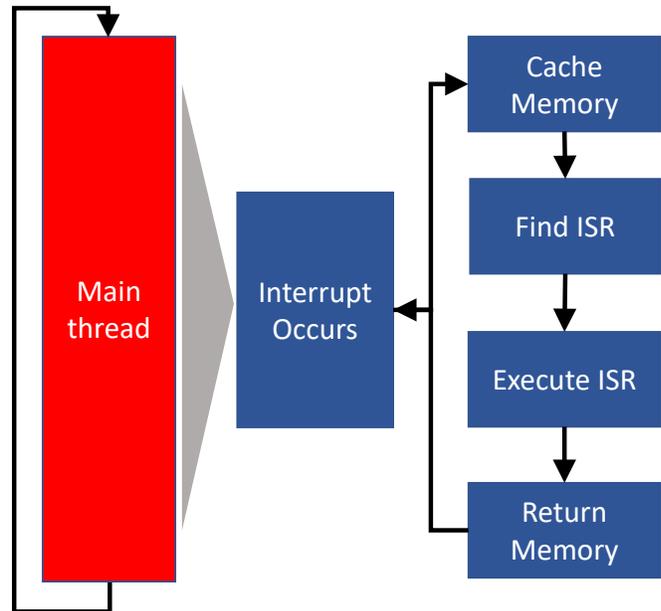
**Supplementary Fig. A2.9 Graphical interface for circle scanning applications.** Top: the main user interface can be used to create series of scan angles and sets of excitation intensities which can be used to build complex experimental sequences. The controller can also be run manually through manipulation of the various scan and excitation controls. Bottom: Integrated program and user interface for calibration

of the excitation laser beam angle at the sample. Using a webcam pictures of the scanned beam on a vertical surface (i.e. a piece of fluorescent plastic) are taken at various scan amplitudes. The software locates the edges of the scan projection in each image and calculates the calibration constant.

## Polling Approach

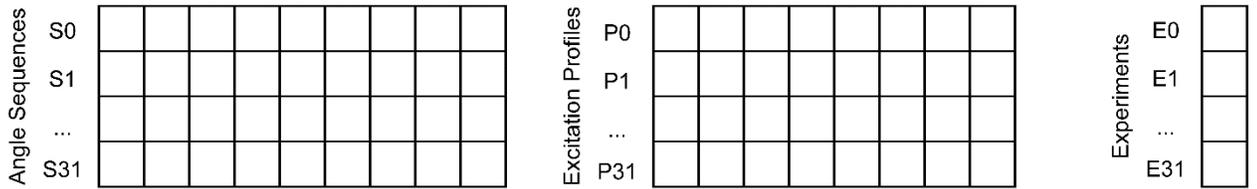


## Interrupt Approach

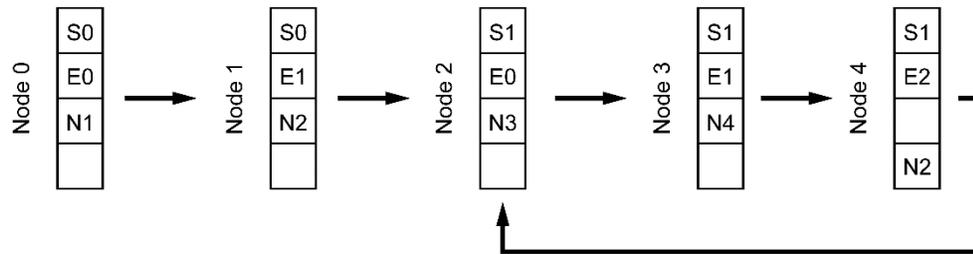


**Supplementary Fig. A2.10 Control flow block diagrams for polling and interrupt approaches to firmware design.** Left: In a polling-based approach the execution runs in a single, sequential thread. Within the thread the state of the system, including hardware I/O and program variables, is periodically sampled at pre-determined checkpoints. Right: In an interrupt-based approach the system performs low-priority tasks in the main thread of execution. When an interrupt is triggered by hardware the main thread is halted and execution is transferred to the interrupt service routine (ISR). Control is returned after the ISR completes to the main thread at the point the interrupt occurred.

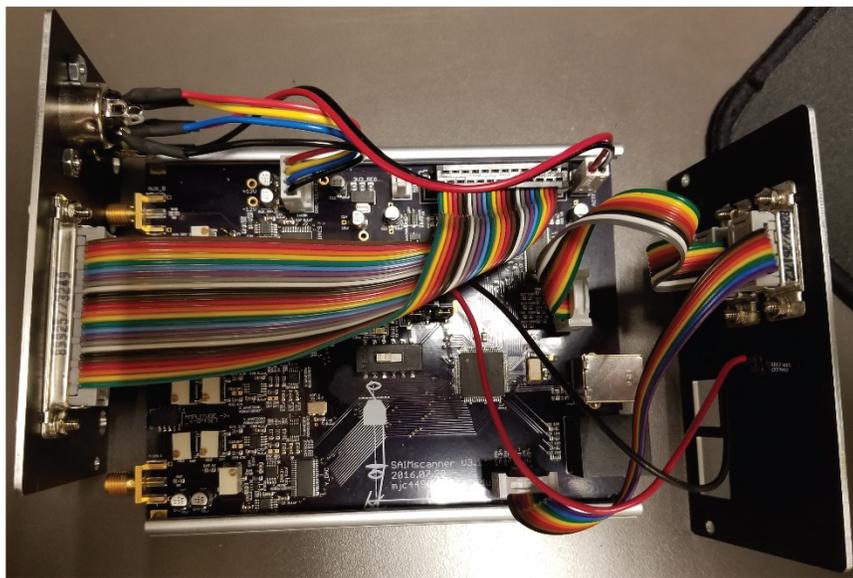
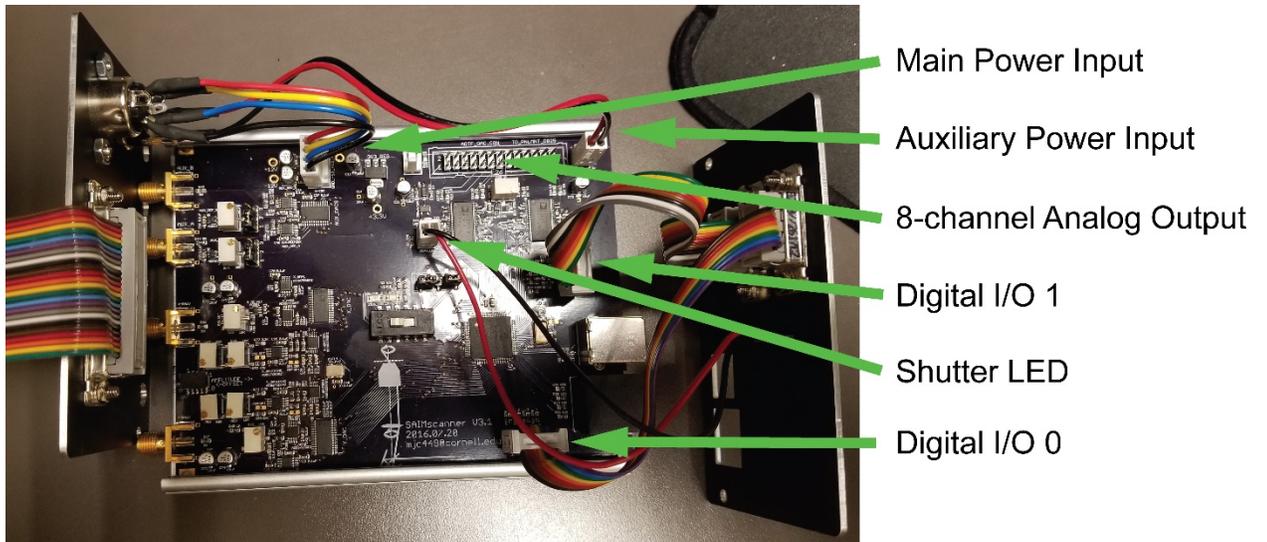
## Static Memory



## Dynamic Memory



**Supplementary Fig. A2.11 The firmware memory organization uses static storage for the experimental parameters and dynamic storage for experiment design.** Each angle sequence is stored as a 1-dimensional array of up to 128 independent values. Excitation profiles are stored as 1-dimensional arrays of the 8 channel values. Experiments are created in dynamic memory as linked lists of nodes, and the address of the first node is stored in a static array of experiments. Nodes within each experiment list contain pointers to the beginning of the angle sequence, excitation profile to be used, and the next node in the list. The final node contains a pointer to the node from which to continue the experiment, allowing steps at the beginning of the experiment to be skipped. The re-use of sequences and profiles greatly reduces the memory requirements for complex experiments.



**Supplementary Fig. A2.12 Internal connections in the completed controller assembly.** Top: image of the controller enclosure with the cover removed and internal connectors labeled. The 8-channel analog output ribbon cable has been removed for clarity. Bottom: same view with the 8-channel analog output ribbon cable installed.

# Appendix C

## **Firmware source code for the instrument controller as used in these studies**

The following is the source code used in performing the experiments in these studies. It was written in the CCS C-Aware IDE and compiled with the PCD compiler.

## File: firmware\_0\_0.h

```
/*
Firmware for the SAIMscannerV3.0 embedded microscope
controller.

Copyright 2019 Marshall J. Colville (mjc449@cornell.edu)

Redistribution and use in source and binary forms, with or
without modification,
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above
copyright notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above
copyright notice,
this list of conditions and the following disclaimer in the
documentation and/or
other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#include <24FJ256GB210.h>

#define NESTED_INTERRUPTS=TRUE

#define FUSES HS
#define FUSES WDT
#define FUSES NOJTAG //JTAG disabled
#define FUSES CKSFSM //Clock Switching is enabled,
fail Safe clock monitor is enabled
#define FUSES PLL96MHZ
#define FUSES NOPROTECT
```

```

#define BRDVER_MAJOR 3
#define BRDVER_MINOR 1
#define FWVER_MAJOR 1
#define FWVER_MINOR 3

#use delay(clock=32M, crystal=20M, USB_FULL)

#define USB_CONFIG_VID 0x04D8
#define USB_CONFIG_PID 0xF11A

#define USB_CONFIG_HID_TX_SIZE 64
#define USB_CONFIG_HID_RX_SIZE 64

#define USB_CONFIG_HID_TX_POLL 1
#define USB_CONFIG_HID_RX_POLL 1

#define USB_STRINGS_OVERWRITTEN

char USB_STRING_DESC_OFFSET[]={0,4,12,22};

const char const USB_STRING_DESC[]={
    //string 0 - language
    4, //length of string index
    0x03, //descriptor type (STRING)
    0x09,0x04, //Microsoft Defined for US-English
    //string 1 - manufacturer
    8, //length of string index
    0x03, //descriptor type (STRING)
    'M',0,
    'J',0,
    'C',0,
    //string 2 - product
    10, //length of string index
    0x03, //descriptor type (STRING)
    'S',0,
    'S',0,
    'v',0,
    '3',0,
    //string 3 - serial number
    12,
    0x03,
    '3',0,
    '0',0,
    '0',0,
    '0',0,
    '7',0
};

#include <pic24_usb.h>
#include <usb_desc_hid.h>

#include <usb.c>

```

```

#include <stdlib.h>
#include <stddef.h>

#use DYNAMIC_MEMORY
#include <stdlibm.h>

#pin_select SCK1OUT=PIN_F3
#pin_select SDI1=PIN_F8
#pin_select SDO1=PIN_F2
#use spi(MASTER, SPI1, MODE=1, BITS=16, ENABLE=PIN_F5,
stream=PIC2PIC)

#pin_select INT1=PIN_A14
#pin_select INT2=PIN_A15

//Direct control of TRIS registers to increase speed
#use FAST_IO(A)
#word TRISA=getenv("SFR:TRISA")
#word LATA=getenv("SFR:LATA")
#word PORTA=getenv("SFR:PORTA")
#use FAST_IO(B)
#use FAST_IO(C)
#word TRISC=getenv("SFR:TRISC")
#word LATC=getenv("SFR:LATC")
#use FAST_IO(D)
#word TRISD=getenv("SFR:TRISD")
#word LATD=getenv("SFR:LATD")
#use FAST_IO(E)
#word TRISE=getenv("SFR:TRISE")
#word TRISF=getenv("SFR:TRISF")
#word LATF=getenv("SFR:LATF")
#word TRISG=getenv("SFR:TRISG")
#word LATG=getenv("SFR:LATG")

#word Tmr1Loc=getenv("SFR:TMR1")

#bit LED_PWR=LATA.2
#bit LED_SHT=LATA.3
#bit LED_SCN=LATA.4
#bit LED_EXP=LATA.5
#bit AOTF_SHT=LATA.7
#bit FIRE_IN=PORTA.14
#bit ARM_IN=PORTA.15
#bit DO_0=LATC.13
#bit DO_1=LATC.14
#bit WAVE_MCLK=LATG.9

//Interrupt flag bits to simulate external triggers
struct {
    unsigned int SI2C1IF:1;
    unsigned int MI2C1IF:1;
    unsigned int CMIF:1;
    unsigned int CNIF:1;
    unsigned int INT1IF:1;
    unsigned int :1;

```

```

    unsigned int IC7IF:1;
    unsigned int IC8IF:1;
    unsigned int :1;
    unsigned int OC3IF:1;
    unsigned int OC4IF:1;
    unsigned int T4IF:1;
    unsigned int T5IF:1;
    unsigned int INT2IF:1;
    unsigned int U2RXIF:1;
    unsigned int U2TXIF:1;
} MCU_IFS1;
#word MCU_IFS1 = 0x086

//Command ID defines
//0x0X are control commands
#define CMD_BLINK_PWR 0x00
#define CMD_VISOR 0x01
#define CMD_BLINK_PWR_VAR 0x02

//0x1X are scan commands
#define CMD_SET_RADIUS 0x10
#define CMD_SET_CTR 0x11
#define CMD_SET_TIRF 0x12
#define CMD_CIRCLE_SCAN 0x13
#define CMD_TIRF_SCAN 0x14
#define CMD_LOC_PARK 0x15
#define CMD_DISC_SCAN 0x16

#define CMD_DISC_SCAN_OFF 0x1E
#define CMD_CENTER_PARK 0x1F

//0x2X are DDS commands
#define CMD_SET_FREQ 0x20
#define CMD_DEFAULT_FREQ 0x21
#define CMD_SET_PHASE 0x22
#define CMD_DEFAULT_PHASE 0x23
#define CMD_MCLK_TOGGLE 0x24
#define CMD_WAVE_RS 0x25
#define CMD_WAVE_CLR_RS 0x26
#define CMD_WAVE_SIN 0x27
#define CMD_WAVE_TRI 0x28
#define CMD_WAVE_SQ 0x29
#define CMD_SET_AXIS_FREQ 0x2A

//0x3X are AUX_DAC commands
#define CMD_CONST_AUX 0x30
#define CMD_MID_AUX 0x31
#define CMD_ZERO_AUX 0x32

//0x4X are AOTF commands
#define CMD_GLOBAL_HIGH 0x40
#define CMD_GLOBAL_LOW 0x41
#define CMD_CHANGE_CH 0x42
#define CMD_LOAD_PROFILE 0x43
#define CMD_OPEN_SHUTTER 0x44

```

```

#define CMD_CLOSE_SHUTTER 0x45
#define CMD_TOGGLE_OPEN 0x46
#define CMD_ADD_PROFILE 0x4C
#define CMD_AOTF_RESET 0x4F

//0x5X are interrupt commands
#define CMD_FIRE_ON 0x50
#define CMD_FIRE_OFF 0x51
#define CMD_SW_TRIGGER 0x5F

//0x6X are triggering commands

//0x7X are SC <-> SC communications

//0x8X are experiment commands
#define CMD_GET_SEQ_USB 0x80
#define CMD_DEL_SEQ 0x81
#define CMD_ADD_SEQ_LIN 0x82
#define CMD_ADD_EXP 0x83
#define CMD_ADD_NODE_START 0x84
#define CMD_ADD_NODE_END 0x85
#define CMD_ADD_LOOP 0x86
#define CMD_START_EXP 0x87
#define CMD_PAUSE_EXP 0x88
#define CMD_RESUME_EXP 0x89
#define CMD_RESTART_EXP 0x8A
#define CMD_DEL_EXP 0x8B
#define CMD_DEL_NODE_START 0x8C
#define CMD_DEL_NODE_END 0x8D
#define CMD_COUNT_STEPS 0x8E
#define CMD_STOP_EXP 0x8F

//0x9X are SimpleSAIM commands
#define CMD_LOAD_SIMPLE_HALF 0x90
#define CMD_LOAD_SIMPLE_FULL 0x91
#define CMD_DIRECTION 0x92
#define CMD_START_SIMPLE 0x93
#define CMD_START_DITHERED 0x94
#define CMD_STOP_SIMPLE 0x95
#define CMD_STEP_COUNT 0x96

//0xAX are Mirror Detector commands
#define CMD_SET_MD_RADIUS 0xA0
#define CMD_MD_ON 0xA1
#define CMD_MD_OFF 0xA2

//0xFX are special function commands
#define CMD_TS_PERIOD 0xF0
#define CMD_GET_SETTINGS 0xF1
#define CMD_GET_INFO 0xF2
#define CMD_CHECK_MEM 0xFD
#define CMD_SEND_STAT 0xFE
#define CMD_RESET_CPU 0xFF

//Macros

```

```

#define OPEN_SHUTTER DO_1 = LED_SHT = 1; //Open the mechanical
shutter
#define CLOSE_SHUTTER DO_1 = LED_SHT = AOTF_SHT = 0; //Close
the mechanical shutter

//Inline function prototypes
#define inline void FIRE_ON(void);
#define inline void FIRE_OFF(void);
#define inline void WAIT_TS(void);
#define inline void WAIT_SWTRIGGER(int);

//Prototypes
void process_command(void);
void check_memory_exists(int8* pCommand);
void visor(void);
void output_error(int Blinks);
void initialization(void);
void report_settings(int8 *pCommand);

//Global status flags
static struct Flag_Word{
    int Ts:1;
    int SAIM:1;
    int SAIMLoop:1;
    int LastFrame:1;
    int Paused:1;
    int Fire:1;
    int Arm:1;
    int DiscScan:1;
    int EndOfExp:1;
    int SimpleSAIM:1;
    int AlwaysOpen:1;
    int UseMirrorDetector:1;
    int SWTrigger:1;
    int SWTriggerState:1;
} Flags;

//Global Variables
static int Zero[] = {0, 0}; //Sometimes you need a variable
that's zero
static int ErrBlink = 0; //Count the number of blinks
static int ErrMSG = 0; //The number of blinks to be output
static int TsReset = 0xF8C0; //Timer1 reset value for ~120 us
Ts period
static int ArmDelay = 0x00F0; //Delay between Arm or Ts and
next trigger

#include "AD5547_dual_drvr.c"
#include "AD9833_dual_drvr.c"
#include "AD5440_drvr.c"
#include "AD5583_dual_drvr.c"
#include "SAIM_utilities.c"
#include "Simple_SAIM.c"

```

## File: firmware\_0\_0.c

```
/*
Firmware for the SAIMscannerV3.0 embedded microscope controller.

Copyright 2019 Marshall J. Colville (mjc449@cornell.edu)

Redistribution and use in source and binary forms, with or without
modification,
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright
notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice,
this list of conditions and the following disclaimer in the
documentation and/or
other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#include <firmware_0_0.h>

#define int_EXT1_LEVEL=5
//ISR that operates the AOTF global blank (gated illumination)
//and increments the AOTF profile and scan angle in a SAIM experiment
void camera_fire_isr(void)
{
    static SAIMnode* ThisNode;
    static int* ThisStep;
    //DO_1 = 1; //For debugging
    disable_interrupts(int_EXT1); //Necessary to check pin state and
change edge
```

```

    if((FIRE_IN || (Flags.SWTrigger && Flags.SWTriggerState))&&
Flags.Ts) //Fire signal is asserted and the galvos have settled
    {
        AOTF_SHT = 1; //Raise the AOTF global clear

        //We always have a ms or more between the start of exposure and
end, so this doesn't need to be super fast
        DO_0 = MirrorDetectorTrigger; //Fire the trigger signal to the
mirror detector camera
        ext_int_edge(1, H_TO_L); //Switch to falling edge detection
    }
    else if(!FIRE_IN || (Flags.SWTrigger && !Flags.SWTriggerState))
//Fire signal has gone low
    {
        if(!Flags.AlwaysOpen) AOTF_SHT = 0; //Lower the AOTF global
clear
        ext_int_edge(1, L_TO_H); //Switch to rising edge detection
        if(Flags.SAIM) //SAIM experiment has been started
        {
            if(StartNode) //If this is the first exposure in the
experiment the start node sill exists
            {
                ThisNode = StartNode;
                StartNode = NULL;
                ThisStep = StartStep;
                StartStep = NULL;
            }

            set_scan_radius(ThisStep); //Update the scan radius first,
can do everything else while settling
            GDAC_LOAD; //Load the new values into the DAC registers
            WAIT_TS(); //Start the settling timer
            DO_0 = 0; //Reset the mirror detector trigger
            //Check wheter or not to fire the mirror detector on the
next exposure
            MirrorDetectorTrigger =
                ((*ThisStep == *MirrorDetectorRadius)
&& Flags.UseMirrorDetector) ? 1 : 0;
            if(Flags.LastFrame && !Flags.EndOfExp)
            {
                update_ADAC_all(ThisNode->pAOTF); //Change the AOTF
output (Ts >> update)
                ADAC_LOAD;
                Flags.LastFrame = 0;
            }
            else if(Flags.LastFrame && Flags.EndOfExp)
            {
                Flags.SAIM = Flags.EndOfExp = Flags.LastFrame = LED_EXP =
0;

                return;
            }

            if(ThisStep >= ThisNode->pSeqEnd) //When we get to the end
of the AOI sequence...
            {

```

```

        Flags.LastFrame = 1; //Set the last frame flag
        if(Flags.SAIMLoop && ThisNode->pLoop != NULL) //If this
is a looping experiment and the loop node is defined
        {
            ThisNode = ThisNode->pLoop; //Update the current node
to the loop node
            ThisStep = ThisNode->pSeqStart; //Move the step
pointer to the beginning of the new sequence
        }
        else if(ThisNode->pNext != NULL)
        {
            ThisNode = ThisNode->pNext; //Go to the next node
            ThisStep = ThisNode->pSeqStart; //Move the step
pointer to the beginning of the new sequence
        }
        else
            Flags.EndOfExp = 1;
    }
    else
        ThisStep += 2; //This is not the end of the sequence, so
we increment the AOI pointer by 2
    }
    if(Flags.SimpleSAIM) //If a simple (non-circle) SAIM
experiment
    {
        if(currStep < steps) currStep++;
        if(currStep == steps)
        {
            Flags.SimpleSAIM = LED_EXP = 0;
            return;
        }
        if(direction!=0) x_offset(stepListX + currStep);
//Direction = 1, scan x, pointer arithmetic
        else y_offset(stepListY + currStep); //Direction = 0, scan
y, pointer arithmetic
        GDAC_LOAD;
        WAIT_TS();
    }
}
//In the event that the next exposure starts prior to the galvos
settling
//it will be ignored and an error output. This is most important
for high-speed
//SAIM experiments where Ts approaches 1% of the exposure time,
but up to 10%
//of the fire signal. Waiting for the settling time will alter
the effective
//length of illumination, leading to artifacts in some frames,
therefore the
//frames without settled galvos will not trigger the AOTF global
blank,
//and should be discarded. The following exposure will trigger
the next step in the experiment
else
{

```

```

        output_error(1); //1 blink for the dropped frame
        ext_int_edge(1, L_TO_H); //The acquisition is aborted, so
reset the edge detection
    }
    enable_interrupts(int_EXT1); //Re-enable the interrupt
    //DO_1 = 0; //For debugging
}

#int_EXT2 LEVEL=4
void camera_arm_isr(void)
{
    if(!Flags.Ts) //Ts hasn't passed since the last move
        set_timer4(get_timer1() - ArmDelay); //Delay before next
camera trigger
    else
        set_timer4(0xFFFF - ArmDelay); //Use the minimum delay
        enable_interrupts(int_TIMER4); //Start the delay timer
        disable_interrupts(int_EXT2); //This must be reset externally
}

//This holds the delay for the galvo settling time (~120 us)
//While this timer is active an exposure cannot be triggered
#int_TIMER1 LEVEL=5
void galvo_settling_isr(void)
{
    Flags.Ts = 1; //Set the Ts bit
    disable_interrupts(int_TIMER1); //This interrupt must be reset
externally
}

//This interrupt controls the point-scanning behavior of the galvos
#int_TIMER2 LEVEL=5
void point_scan_isr(void)
{
    set_timer2(Tmr2Reset); //Update rate = 0.25 us * (0xFFFF -
Tmr2Reset)
    x_offset(PointListX + ScanPosition); //Load the X value into the
XDAC data register
    y_offset(PointListY + ScanPosition); //Load the Y value into the
YDAC data register
    GDAC_LOAD; //Load the DAC registers simultaneously
    ScanPosition++; //Increment the circular position
    if(ScanPosition > ScanRollover) //When the circle is complete
        ScanPosition = 0; //Start back at 0
}

//This interrupt controls the ERR message output on the PWR LED
#int_TIMER3 LEVEL = 1
void error_output_isr(void)
{
    LED_PWR = ~LED_PWR; //Toggle the PWR LED on each call
    ErrBlink++; //Increment the blink counter
    if(ErrBlink > ErrMSG) //If the appropriate number of blinks has
been reached
    {

```

```

        disable_interrupts(int_TIMER3); //Clear the interrupt enable
bit
    ErrBlink = 0; //Reset the blink counter
    LED_PWR = 1; //Make sure to leave the light on
}
}

//Delay between arm signal and beginning the next exposure
//This will begin an exposure or sequence when using the microscope
camera
//in external trigger mode depending on the camera configuration
#int_TIMER4 LEVEL=3 //This is lower priority than the fire ISR
void arm_delay_isr(void)
{
    DO_0 = 1; //Trigger the start of the acquisition sequence
    disable_interrupts(int_TIMER4);
}

#int_TIMER5 LEVEL = 1 //This is the lowest priority
void general_timer_int(void)
{
    if(Flags.Paused)
        LED_EXP = ~LED_EXP;
    if(Flags.SWTrigger){
        Flags.SWTriggerState = 0;
        camera_fire_isr();
        Flags.SWTrigger = 0;
    }
}

void main()
{
    initialization() ;

    while (TRUE)
    {
        if(usb_kbhit(1))
            process_command();
        restart_wdt();
    }
}

//Called from main() on an incoming USB packet
//The command and data are first fetched into the command buffer,
then
//the appropriate action is taken
void process_command()
{
    disable_interrupts(GLOBAL);
    int ExpNum;
    int8 result; //Pass/Fail results
    int8 command[64]; //Buffer for incoming data
    usb_gets(1, command, 64, 100);
    switch (command[0]) //First byte is the command ID
    {

```

```

//0x0X control commands
case CMD_BLINK_PWR: //Simple 10 fast blinks of the PWR/ERR LED
    for(int i = 0; i < 10; i++)
    {
        LED_PWR = 0;
        delay_ms(100);
        LED_PWR = 1;
        delay_ms(100);
    }
    break;
case CMD_VISOR: //Run the visor() LED function
    visor();
    break;
case CMD_BLINK_PWR_VAR: //Use the ERR function
    output_error(make16(command[1], command[2]));
    break;
//0x1X scan commands
case CMD_SET_RADIUS:
    load_CS_radius(&command[1]);
    break;
case CMD_SET_CTR:
    load_CS_center(&command[1]);
    break;
case CMD_SET_TIRF:
    load_CS_TIRF(&command[1]);
    break;
case CMD_CIRCLE_SCAN:
    circle_scan();
    break;
case CMD_TIRF_SCAN:
    TIRF_scan();
    break;
case CMD_LOC_PARK:
    park_location(&command[1]);
    break;
case CMD_DISC_SCAN:
    discrete_circle_scan(&command[1]);
    break;
case CMD_DISC_SCAN_OFF:
    stop_discrete_scan();
    break;
case CMD_CENTER_PARK:
    park_center();
    break;
//0x2X DDS commands
case CMD_SET_FREQ:
    wave_set_freq(make16(command[1], command[2]));
    break;
case CMD_DEFAULT_FREQ:
    wave_set_freq(OnekHz);
    break;
case CMD_SET_PHASE:
    wave_set_phase_diff(make16(command[1], command[2]));
    break;
case CMD_DEFAULT_PHASE:

```

```

    wave_set_phase_diff(Phase90);
    break;
case CMD_MCLK_TOGGLE:
    WAVE_MCLK = ~WAVE_MCLK;
    break;
case CMD_WAVE_RS:
    wave_reset(1);
    break;
case CMD_WAVE_CLR_RS:
    wave_reset(0);
    break;
case CMD_WAVE_SIN:
    wave_shape(0);
    break;
case CMD_WAVE_TRI:
    wave_shape(1);
    break;
case CMD_WAVE_SQ:
    wave_shape(2);
    break;
case CMD_SET_AXIS_FREQ:
    wave_set_axis_freq(make16(command[1], command[2]),
make16(command[3],command[4]));
    break;
//0x3X auxiliary DAC
case CMD_CONST_AUX:
    int AUXValue = make16(command[2], command[3]);
    if(!command[1])
        write_aux_a(AUXValue);
    else
        write_aux_b(AUXValue);
    break;
case CMD_MID_AUX:
    write_aux_a(AUXMidscale[0]);
    write_aux_b(AUXMidscale[1]);
    break;
case CMD_ZERO_AUX:
    write_aux_a(Zero[0]);
    write_aux_b(Zero[1]);
    break;
//0x4X AOTF
case CMD_GLOBAL_HIGH:
    FIRE_OFF();
    AOTF_SHT = 1;
    OPEN_SHUTTER;
    break;
case CMD_GLOBAL_LOW:
    FIRE_OFF();
    AOTF_SHT = 0;
    break;
case CMD_CHANGE_CH:
    int ChValue = make16(command[2], command[3]);
    update_ADAC_channel(&(command[1]), &ChValue);
    ADAC_LOAD;
    break;

```

```

case CMD_LOAD_PROFILE:
    command[2] = load_AOTF_profile(command[1]);
    break;
case CMD_OPEN_SHUTTER:
    OPEN_SHUTTER;
    break;
case CMD_CLOSE_SHUTTER:
    FIRE_OFF();
    CLOSE_SHUTTER;
    break;
case CMD_TOGGLE_OPEN:
    if(Flags.AlwaysOpen)
    {
        AOTF_SHT = 0;
        CLOSE_SHUTTER;
        Flags.AlwaysOpen = 0;
    }
    else
    {
        AOTF_SHT = 1;
        OPEN_SHUTTER;
        Flags.AlwaysOpen = 1;
    }
    break;
case CMD_ADD_PROFILE:
    result = new_ADAC_profile(&command[1]);
    command[0] = make8(result, 0);
    break;
case CMD_AOTF_RESET:
    ADAC_RS;
    AOTF_SHT = 0;
    break;
//0x5X interrupt
case CMD_FIRE_ON:
    FIRE_ON();
    break;
case CMD_FIRE_OFF:
    FIRE_OFF();
    break;
case CMD_SW_TRIGGER:
    WAIT_SWTRIGGER(make16(command[1], command[2]));
    break;
//0x6X triggering

//0x7X SC<->SC communications

//0x8X experiment
case CMD_GET_SEQ_USB:
    result = get_seq_usb(&command[1]);
    command[0] = make8(result, 0);
    break;
case CMD_DEL_SEQ:
    int SeqNum = (int)command[1];
    delete_seq(SeqNum);
    break;

```

```

case CMD_ADD_SEQ_LIN:
    result = add_seq_linear(&command[1]);
    command[0] = make8(result, 0);
    break;
case CMD_ADD_EXP:
    result = create_new_node(&command[1]);
    command[0] = result;
    break;
case CMD_ADD_NODE_START:
    result = push_node(&command[1]);
    command[0] = result;
    break;
case CMD_ADD_NODE_END:
    result = add_last_node(&command[1]);
    command[0] = result;
    break;
case CMD_ADD_LOOP:
    ExpNum = (int)command[1];
    int LoopNode = (int)command[2];
    result = build_loop(ExpNum, LoopNode);
    command[0] = result;
    break;
case CMD_START_EXP:
    command[0] = start_experiment(&command[1]);
    break;
case CMD_PAUSE_EXP:
    pause_experiment();
    break;
case CMD_RESUME_EXP:
    resume_experiment();
    break;
case CMD_RESTART_EXP:
    restart_experiment();
    break;
case CMD_DEL_EXP:
    ExpNum = (int)command[1];
    delete_all_nodes(ExpNum);
    break;
case CMD_DEL_NODE_START:
    ExpNum = (int)command[1];
    pop_node(ExpNum);
    break;
case CMD_DEL_NODE_END:
    ExpNum = (int)command[1];
    remove_last_node(ExpNum);
    break;
case CMD_COUNT_STEPS:
    count_steps(&command[1]);
    break;
case CMD_STOP_EXP:
    stop_experiment();
    break;

//0x9X SimpleSAIM
case CMD_LOAD_SIMPLE_HALF:

```

```

        steps = make16(command[1], command[2]);
        int stepSize = make16(command[3], command[4]);
        command[0] = create_simple(stepSize);
        break;
    case CMD_LOAD_SIMPLE_FULL:
        steps = make16(command[1], command[2]);
        int stepSize1 = make16(command[3], command[4]);
        int startPosX = make16(command[5], command[6]);
        int startPosY = make16(command[7], command[8]);
        command[0] = create_simple(stepSize1, startPosX, startPosY);
        break;
    case CMD_DIRECTION:
        direction = command[1];
        break;
    case CMD_START_SIMPLE:
        start_simple();
        break;
    case CMD_START_DITHERED:
        int val = make16(command[1], command[2]);
        start_dithered(&val);
        break;
    case CMD_STOP_SIMPLE:
        stop_simple();
        break;
    case CMD_STEP_COUNT:
        int8* ptr = &steps;
        command[1] = *ptr++;
        command[2] = *ptr;
        ptr = &currStep;
        command[3] = *ptr++;
        command[4] = *ptr;
        break;

//0xAX Mirror Detector
    case CMD_SET_MD_RADIUS:
        *MirrorDetectorRadius = make16(command[1], command[2]);
        break;
    case CMD_MD_ON:
        Flags.UseMirrorDetector = 1;
        command[1] = Flags.UseMirrorDetector;
        break;
    case CMD_MD_OFF:
        Flags.UseMirrorDetector = 0;
        command[1] = Flags.UseMirrorDetector;
        break;

//0xFX special functions
    case CMD_TS_PERIOD:
        TsReset = make16(command[1], command[2]);
        break;
    case CMD_GET_SETTINGS:
        report_settings(&command[1]);
        break;
    case CMD_GET_INFO:
        command[1] = BRDVER_MAJOR;

```

```

        command[2] = BRDVER_MINOR;
        command[3] = FWVER_MAJOR;
        command[4] = FWVER_MINOR;
        break;
    case CMD_CHECK_MEM:
        check_memory_exists(&command[1]);
        break;
    case CMD_SEND_STAT:
        command[1] = Flags;
        break;
    case CMD_RESET_CPU:
        reset_cpu();
        break;

    default: //If the command is not recognized (i.e. doesn't
exist)
        output_error(10); //Blink the ERR LED 10 times
        command[0] = 0xFF; //Report the error
        break;
    }
    enable_interrupts(GLOBAL);
    usb_puts (1, command, 64, 100);
}

#inline void FIRE_ON(void)
{
    if(!Flags.Fire)
    {
        OPEN_SHUTTER;
        AOTF_SHT = 0;
        Flags.Fire = 1;
        enable_interrupts(int_EXT1);
    }
}

#inline void FIRE_OFF(void)
{
    if(Flags.Fire)
    {
        pause_experiment();
        disable_interrupts(int_EXT1);
        ext_int_edge(1, L_TO_H);
        AOTF_SHT = 0;
        Flags.Fire = 0;
    }
}

#inline void WAIT_TS(void)
{
    Flags.Ts = 0;
    set_timer1(TsReset);
    enable_interrupts(int_TIMER1);
}

#inline void WAIT_SWTRIGGER(int val)

```

```

{
    Flags.SWTrigger = 1;
    Flags.SWTriggerState = 1;
    camera_fire_isr();
    set_timer5(val);
    enable_interrupts(int_TIMER5);
}

void report_settings(int8 *pCommand)
{
    *pCommand++ = make8(CSCenter[0], 1);
    *pCommand++ = make8(CSCenter[0], 0);
    *pCommand++ = make8(CSCenter[1], 1);
    *pCommand++ = make8(CSCenter[1], 0);
    *pCommand++ = make8(CSTIRF[0], 1);
    *pCommand++ = make8(CSTIRF[0], 0);
    *pCommand++ = make8(Phase, 1);
    *pCommand++ = make8(Phase, 0);
    *pCommand++ = make8(FrequencyX, 1);
    *pCommand++ = make8(FrequencyX, 0);
}

//Asks whether the pointers in the head arrays are assigned
//Returns the number of assigned pointers, 0 otherwise
//Also overwrites the command buffer to show number of assignments
void check_memory_exists(int8* pCommand)
{
    int8 ExpInUse = 0;
    long ExpPattern = 0;
    int i;
    for(i = 0; i < MaxExp; i++)
    {
        if(ExpHeads[i])
        {
            ExpInUse++;
            bit_set(ExpPattern, i);
        }
    }
    *pCommand++ = ExpInUse;
    *pCommand++ = make8(ExpPattern, 3);
    *pCommand++ = make8(ExpPattern, 2);
    *pCommand++ = make8(ExpPattern, 1);
    *pCommand++ = make8(ExpPattern, 0);
}

void visor()
{
    int1 PWRstate = LED_PWR;
    int1 SHTstate = LED_SHT;
    int1 SCNstate = LED_SCN;
    int1 EXPstate = LED_EXP;
    for(int i = 0; i < 5; i++) //Visor output for panel LEDs
    {
        LED_PWR = 1;
        delay_ms(50);
    }
}

```

```

        LED_PWR = 0;
        LED_SHT = 1;
        delay_ms(50);
        LED_SHT = 0;
        LED_SCN = 1;
        delay_ms(50);
        LED_SCN = 0;
        LED_EXP = 1;
        delay_ms(50);
        LED_EXP = 0;
        LED_SCN = 1;
        delay_ms(50);
        LED_SCN = 0;
        LED_SHT = 1;
        delay_ms(50);
        LED_SHT = 0;
        LED_PWR = 1;
        delay_ms(50);
        LED_PWR = 0;
    }
    LED_PWR = PWRstate;
    LED_SHT = SHTstate;
    LED_SCN = SCNstate;
    LED_EXP = EXPstate;
}

//Output a specified number of blinks on the PWR/ERR LED
void output_error(int Blinks)
{
    ErrMSG = (2 * Blinks) - 1;
    set_timer3(0x0000);
    enable_interrupts(int_TIMER3);
}

//Function to setup the I/O ports and initialize the peripherals
void initialization()
/*
This function is called once at powerup to initialize peripherals and
put
the system in a known state. All unused pins are set as outputs and
driven
*/
{
    //Using FAST_IO directive requires manual TRIS config
    //Setup PORTA
    bit_set(TRISA,15); //TRIG_1
    bit_set(TRISA,14); //TRIG_0
    bit_clear(TRISA,10); //GDAC A1
    bit_clear(TRISA,9); //GDAC A0
    bit_clear(TRISA,7); //AOTF global blank
    bit_clear(TRISA,6); //Unused
    bit_clear(TRISA,5); //LED_EXP
    bit_clear(TRISA,4); //LED_SCN
    bit_clear(TRISA,3); //LED_SHT
    bit_clear(TRISA,2); //LED_PWR

```

```

    bit_clear(TRISA,1); //GDAC_RS
    bit_clear(TRISA,0); //Unused
    set_pullupdown(true, PIN_A15); //A15 and A14 have external
pulldowns
    set_pullupdown(true, PIN_A14); //Internal pulldowns might interfere
withinputs
    LATA = 0b0000000000000100; //Drive everything low except the
PWR/STATUS LED
    bit_set(LATA, 1); //Bring the GDAC out of reset

    //Setup PORTB
    SET_TRIS_B(0); //PORTB is the 16 bit GDAC data port, nothing to do
here
    output_b(0x7FFF); //The GDAC is already at midrange reset, but
might as well...

    //Setup PORTC
    bit_clear(TRISC,14); //C14 and C13 are GPIO pins connected to
panel BNCs
    bit_clear(TRISC,13); //We use these as trigger sources for other
hardware
    bit_clear(TRISC,4); //C4 is the X DDS data line
    bit_clear(TRISC,3); //Unused
    bit_clear(TRISC,2); //Unused
    bit_clear(TRISC,1); //Unused
    output_c(0b0000000000010000); //Drive everything low except the X
DDS data line

    //Setup PORTD
    SET_TRIS_D(0); //PORTD is the 10 bit ADAC data port and control
pins
    output_d(0b0111000000000000); //Bring, LOAD, and CS lines high,
everything else low
    bit_set(LATD, 15); //Bring the RS high to enable output

    //Setup PORTE
    SET_TRIS_E(0); //PORTE is the 10 bit AUXDAC data bus, nothing to
do here
    output_e(0x01FF); //Set data pins to midrange

    //Setup PORTF
    bit_clear(TRISF,13); //PORTF has PIC2PIC SPI as well as VBUS, so
we use STD
    bit_clear(TRISF,12); //IO on it, however we will set tris and
toggle LATF
    bit_clear(TRISF,4); //for the PINS that we are manually
controlling
    bit_clear(TRISF,1);
    bit_clear(TRISF,0);
    bit_set(LATF, 13); //GDAC_XWR is active low (latches data on
falling edge)
    bit_clear(LATF, 12); //GDAC_LD is active high (loads data on
rising edge)
    bit_set(LATF, 4); //GDAC_YWR
    bit_clear(LATF, 1); //Unused

```

```

    bit_clear(LATF, 0); //Unused

    //Setup PORTG
    bit_clear(TRISG,15); //PORTF has the D+/D- pins, so we have to
use STD IO
    bit_clear(TRISG,14); //We still set TRIS in the initialization
sequence
    bit_clear(TRISG,13); //on the data pins that we will manually
control
    bit_clear(TRISG,12); //for peripherals
    bit_clear(TRISG,9);
    bit_clear(TRISG,8);
    bit_clear(TRISG,7);
    bit_clear(TRISG,6);
    bit_clear(TRISG,1);
    bit_clear(TRISG,0);
    bit_clear(LATG, 15); //Set the startup states
    bit_clear(LATG, 14); //Unused pins should be left low
    bit_clear(LATG, 13); //Select channel A
    bit_set(LATG, 12); //Diasble input register
    bit_set(LATG, 9); //Begin with 25 MHz oscillator output enabled
    bit_set(LATG, 8); //Y DDS data line idle high
    bit_set(LATG, 7); //Active low DDS SYNC
    bit_set(LATG, 6); //DDS SCK, data is clocked on falling edge,
idle high
    bit_clear(LATG, 1); //Unused
    bit_clear(LATG, 0); //Unused
    //Setup the galvo DACs to center park the galvos
    center_park();
    //Set the AUXDACs to midscale (0V)
    write_aux_a(AUXMidscale[0]);
    write_aux_b(AUXMidscale[1]);

    delay_us(100);

    //Setup the DDS chips
    //We will initialize them to output two sine functions at 1 kHz
with a pi/2
    //phase shift between them, which is the default operating state
    wave_reset(1); //Enables the reset state of both DDS, disabling
output
    wave_freq_range(1); //while reset, select the freq MSB
    wave_set_freq(0x0000); //Write 0 to the MSB, limiting output
range to 1.5 kHz
    wave_freq_range(0); //Select the freq LSB
    wave_set_freq(OnekHz); //Set the output freq to 1 kHz
    wave_set_phase_diff(Phase90); //Set the phase offset to pi/2
    wave_reset(0); //Disables reset, output becomes available

    for(int i = 0; i < 5; i++) //Blink the shutter LED to verify DDS
have been programmed
    {
        LED_SHT = 1;
        delay_ms(100);
        LED_SHT = 0;
    }

```

```

        delay_ms(100);
    }

    usb_init(); //Start the USB module
    setup_timer1(TMR_INTERNAL | TMR_DIV_BY_1); //The galv settling
timer
    setup_timer2(TMR_INTERNAL | TMR_DIV_BY_1); //The galv move timer
for point scanning
    setup_timer3(TMR_INTERNAL | TMR_DIV_BY_64); //The error message
output timer
    setup_timer4(TMR_INTERNAL | TMR_DIV_BY_1); //Arm delay timer
    setup_timer5(TMR_INTERNAL | TMR_DIV_BY_8); //General purpose
timer/interrupt source
    ext_int_edge(1, L_TO_H); //Start looking for the rising edge of
the fire signal
    ext_int_edge(2, L_TO_H); //Camera is ready when arm goes high
//enable_interrupts(GLOBAL);

visor(); //Indicate the initialization is finished
LED_PWR = 1;

Flags.Ts = 1;
Flags.SAIM = 0;
Flags.Paused = 0;
Flags.Fire = 0;
Flags.Arm = 0;
Flags.DiscScan = 0;
Flags.SWTrigger = 0;
Flags.SWTriggerState = 0;

    setup_wdt(WDT_16S);
}

```

## File: SAIM\_utilities.c

```
/*
Function definitions and variables for loading and running SAIM
experiments
on the SAIMScanner_v3.1 hardware.

Copyright 2019 Marshall J. Colville (mjc449@cornell.edu)

Redistribution and use in source and binary forms, with or without
modification,
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright
notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice,
this list of conditions and the following disclaimer in the
documentation and/or
other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
*/

//Define the node structure for the linked lists that make up
experiments
typedef struct node {
    int* pSeqStart; //Start of the AOI sequence
    int* pSeqEnd; //End of the AOI sequence
    int* pAOTF; //Head of the AOTF profile
    struct node* pNext; //Pointer to the next node in the experiment
    struct node* pLoop; //Pointer to the node to jump to in a looping
experiment
} SAIMnode;
```

```

//Define the experiment setup parameters structure type
//for storing and initiating experiments
typedef struct experiment {
    int ExpNum; //Number of the experiment list in the head array
    ptrdiff_t StepNum; //Step to begin the experiment at
    int LoopOn; //T/F use loop if defined
    int ActivationTime; //Time for pre-experiment photoactivation (if
0 this is skipped)
    int ActivationIntensity; //Intensity to use for pre-experiment
photoactivation
} ExpSetup;

//Define some global variables
const size_t MaxExp = 32; //Can be changed later if need be
static SAIMnode* ExpHeads[MaxExp]; //Locations of the experiment
begin nodes
const size_t MaxSeq = 32; //Can be changed later if need be
static int* SeqTails[MaxSeq]; //Location of the last angle (HBYTE x)
in the sequences
static int SeqArray[MaxSeq][ 2 * 128];
static SAIMnode* StartNode = NULL;
static int* StartStep = NULL;
static int* MirrorDetectorRadius = NULL;
static int1 MirrorDetectorTrigger = 0;

void delete_seq(int& SeqNum)
{
    SeqTails[SeqNum] = NULL;
}

//Creates a new sequence
//Returns 0 if the operation succeeds, 1 if allocation fails, 2 if
usb error
int8 get_seq_usb(int8* pCommand)
{
    int SeqNum = (int)*pCommand++; //Second byte in the command is
the list number
    int SeqLen = make16(*pCommand++, *pCommand++); //Third and fourth
bytes are the length of the sequence (number of angles)
    int PacketsInbound = make16(*pCommand++, *pCommand++); //The
number of USB packets required for the transfer
    int PacketsRead = 0;
    int8 Data[64] = {0};
    int size = 0;

    if(!SeqLen || (SeqLen > 128))
    {
        output_error(2);
        return(2);
    }

/*****/
//The following cannot fail or undefined behavior results on
StartExperiment

```

```

    delete_seq(SeqNum); //Make sure to clean up in case the rest
fails

    int* pSeq = &SeqArray[SeqNum][0]; //Pointer to the array that
will hold the

    SeqTails[SeqNum] = &SeqArray[SeqNum][(SeqLen - 1) * 2];

/*****
*****/

usb_puts(1, Data, 64, 100);
int lastPacket = 0;
do //Loop to read in data
{
    while(!usb_kbhit(1)){}; //Wait for the packet to be sent
    usb_gets(1, Data, 64, 100); //Store in the data buffer
    PacketsRead++;
    if(PacketsRead < PacketsInbound)
        size = 32; //Full report length in words
    else if(PacketsRead == PacketsInbound)
    {
        size = (SeqLen * 2) - ((PacketsRead - 1) * 32);
        lastPacket = 1;
    }
    else
    {
        output_error(2);
        return 2;
    }
    for(int i = 0; i < size; i++)
    {
        *pSeq++ = makel6(Data[i * 2], Data[i * 2 + 1]); //Put the
values into the sequence
    }
    usb_puts(1, Data, 64, 100); //Response to signal ready for
next transfer
    restart_wdt();
}while(lastPacket == 0);

    return(0);
}

int8 add_seq_linear(int8* pCommand)
{
    int SeqNum = makel6(0, *pCommand++); //Second byte in the command
is the list number
    int SeqLen = makel6(*pCommand++, *pCommand++); //Third and fourth
bytes are the number of angles in the sequence
    int SeqStep = makel6(*pCommand++, *pCommand++); //Fourth and
fifth bytes are the step size for the sequence in DAC units
    int SeqStart = makel6(*pCommand++, *pCommand ++); //Sixth and
seventh bytes are the first angle in the sequence

```

```

    int YScale = make16(*pCommand++, *pCommand++); //Eighth and ninth
bytes are a linear adjustment to the y values to account for
ellipticity
    int* pSeq = NULL; //Pointer to the new sequence
    int MidPt = 0x7FFF;
    int XValue = SeqStart;

    if(!SeqLen || (SeqLen > 128))
    {
        output_error(2);
        return(2);
    }

    //Use the scale factor to calculate the y start
    int YValue = (int)((float)SeqStart * (float)YScale /
(float)MidPt);

    //Calculate the step value for the Y DAC. This is the most basic
correction and may not be adequate
    int YStep = (int)((float)SeqStep * (float)YScale / (float)MidPt);

/*****/
//The following cannot fail or undefined behavior results on
StartExperiment
delete_seq(SeqNum); //Make sure to clean up in case the rest
fails

    pSeq = &SeqArray[SeqNum][0]; //Pointer to the array that will
hold the

    SeqTails[SeqNum] = &SeqArray[SeqNum][(SeqLen - 1) * 2];

/*****/

    for(int i = 0; i < SeqLen; i++)
    {
        *pSeq++ = XValue; //Put the current values into the sequence
array
        *pSeq++ = YValue;
        XValue += SeqStep; //Increment the values by the step sizes
        YValue += YStep;
    }
    return(0);
}

//Pops a node from the list,
void pop_node(int& ExpNum)
{
    SAIMnode* NextNode = NULL; //Put the address of the second node
in the list here
    SAIMnode* head = ExpHeads[ExpNum];

```

```

    if(!head) //Check that there is at least one node
        return;

    NextNode = head->pNext; //Fill in the temp. variable
    free(head); //Deallocated the block for the first node
    ExpHeads[ExpNum] = NextNode; //Point the experiment head at the
new first node
}

void delete_all_nodes(int& ExpNum)
{
    while(ExpHeads[ExpNum])
        pop_node(ExpNum); //Keep popping nodes (and freeing them)
until the head points to NULL
}

//Makes a new SAIMnode list, deleting any existing nodes before
beginning
//Returns 0 if successful, 1 if allocation fails, 2 if seq is empty
and 3 if profile is empty
int8 create_new_node(int8* pCommand)
{
    int ExpNum = (int)*pCommand++;
    int SeqNum = (int)*pCommand++;
    int AOTFNum = (int)*pCommand;

    if(ExpNum >= MaxExp)
    {
        output_error(2);
        return(2);
    }
    if(!SeqTails[SeqNum]) //If the sequence doesn't exist return 2
    {
        output_error(2);
        return(3);
    }

    delete_all_nodes(ExpNum); //Make sure to free any existing memory

    SAIMnode* head = malloc(sizeof(SAIMnode)); //Allocate space for
the first node
    if(!head) //Check that the allocation was successful
    {
        output_error(2);
        return(1);
    }

    ExpHeads[ExpNum] = head; //Add the address of the experiment to
the list

    head->pSeqEnd = SeqTails[SeqNum]; //Fill in the end of the AOI
sequence pointer
    head->pSeqStart = &SeqArray[SeqNum][0]; //First AOI value
location

```

```

    head->pAOTF = &AOTFArray[AOTFNum][0]; //Fill in the AOTF profile
pointer
    head->pNext = NULL; //This is the first and last node
    head->pLoop = NULL; //Always initialize to NULL

    return(0);
}

//Pushes a SAIMnode onto the list of the given experiment
//Returns 0 if successful, 1 if allocation fails, 2 if seq is empty
and 3 if profile is empty
//Can be used to create an experiment on an empty list
int8 push_node(int8* pCommand)
{
    int ExpNum = (int)*pCommand++;
    int SeqNum = (int)*pCommand++;
    int AOTFNum = (int)*pCommand;

    if(ExpNum >= MaxExp)
    {
        output_error(2);
        return(2);
    }
    if(!SeqTails[SeqNum]) //If the sequence doesn't exist return 2
    {
        output_error(2);
        return(2);
    }

    SAIMnode* NewNode = malloc(sizeof(SAIMnode)); //Allocate space
for the first node
    if(!NewNode) //Check that the allocation was successful
    {
        output_error(2);
        return(1);
    }

    NewNode->pSeqEnd = SeqTails[SeqNum];
    NewNode->pSeqStart = &SeqArray[SeqNum][0];
    NewNode->pAOTF = &AOTFArray[AOTFNum][0];
    NewNode->pNext = ExpHeads[ExpNum];
    NewNode->pLoop = NULL;
    ExpHeads[ExpNum] = NewNode;

    return(0);
}

//Removes the last node in the list at ExpNum
//Returns 0 if there are nodes left, 1 if the list is now empty, and
2 if the list was empty to begin with
int8 remove_last_node(int& ExpNum)
{
    SAIMnode* head = ExpHeads[ExpNum];

    if(!head) //The list is empty

```

```

        return(2);
    if(!head->pNext) //The list only contains one node
    {
        pop_node(ExpNum);
        return(1);
    }

    SAIMnode* CurrNode = ExpHeads[ExpNum]; //Starting from the first
node

    while(CurrNode->pNext->pNext) //While the current node is not
next-to-last
        CurrNode = CurrNode->pNext; //Increment through the list

    free(CurrNode->pNext->pNext); //Deallocate the last node
    CurrNode->pNext = NULL; //The current node is now the last node
    return(0);
}

//Add a node to the end of the list
//Returns 0 if successful, 1 if allocation fails, 2 if seq is empty
and 3 if profile is empty
//Can be used to create an experiment on an empty list
int8 add_last_node(int8* pCommand)
{
    int ExpNum = (int)*pCommand++;
    int SeqNum = (int)*pCommand++;
    int AOTFNum = (int)*pCommand;

    if(ExpNum >= MaxExp)
    {
        output_error(2);
        return(2);
    }
    if(SeqTails[SeqNum] == NULL) //If the sequence doesn't exist
return 2
    {
        output_error(2);
        return(3);
    }

    SAIMnode* NewNode = malloc(sizeof(SAIMnode)); //Allocate space
for the new node
    if(!NewNode) //Check that the allocation was successful
    {
        output_error(2);
        return(1);
    }

    NewNode->pSeqEnd = SeqTails[SeqNum];
    NewNode->pSeqStart = &SeqArray[SeqNum][0];
    NewNode->pAOTF = &AOTFArray[AOTFNum][0];
    NewNode->pNext = NULL;
    NewNode->pLoop = NULL;

```

```

    if(!ExpHeads[ExpNum])
    {
        ExpHeads[ExpNum] = NewNode;
        return(0);
    }

    SAIMnode* CurrNode = ExpHeads[ExpNum];
    while(CurrNode->pNext != NULL) //We're looking for the end of the
list
        CurrNode = CurrNode->pNext; //Increment the current pointer
until we find it
    CurrNode->pNext = NewNode; //Add the new node to the end
    return(0);
}

//Counts and returns the number of nodes and steps in a given
experiment
void count_steps(int8* pCommand)
{
    SAIMnode* CurrNode = ExpHeads[(int)*pCommand++];
    int NSteps = 0, NNodes = 0;

    while(CurrNode)
    {
        NSteps += (CurrNode->pSeqEnd - CurrNode->pSeqStart + 2) / 2;
        NNodes++;
        CurrNode = CurrNode->pNext;
    }
    *pCommand++ = make8(NNodes, 1);
    *pCommand++ = make8(NNodes, 0);
    *pCommand++ = make8(NSteps, 1);
    *pCommand++ = make8(NSteps, 0);
}

//Clears any existing loops in the given experiment
void clear_loop(int& ExpNum)
{
    if(!ExpHeads[ExpNum])
        return;
    SAIMnode* CurrNode = ExpHeads[ExpNum];
    while(CurrNode->pNext != NULL) //As long as pLoop is NULL and
pNext exists
    {
        CurrNode->pLoop = NULL;
        CurrNode = CurrNode->pNext; //Go to the next node in the list
    }
    return;
}

//Adds a loop to an experiment by linking the last node to the node
number passed
//in the function call. Returns 0 if successful, 1 if the experiment
doesn't
//exist and 2 if the loop step is out of bounds
int8 build_loop(int& ExpNum, int& nLoopNode)

```

```

{
    if(!ExpHeads[ExpNum]) //If the experiment doesn't exist throw an
error
    {
        output_error(2);
        return(1);
    }
    clear_loop(ExpNum); //Make sure to clear any pre-existing loops
    SAIMnode* pCurrNode = ExpHeads[ExpNum]; //Temp storage for
looking for the loop and end
    SAIMnode* pLoopNode = NULL; //Temp storage for saving the loop
address
    int nCurrNode = 0; //Node counter
    while(nCurrNode < nLoopNode) //Look for the node before the loop
    {
        if(pCurrNode->pNext == NULL) //If the loop is out of bounds
        {
            output_error(2); //Throw an error
            return(2);
        }
        pCurrNode = pCurrNode->pNext; //Otherwise, increment the node
an counter
        nCurrNode++;
    }
    pLoopNode = pCurrNode; //The current node is the loop node
    while(pCurrNode->pNext != NULL)
        pCurrNode = pCurrNode->pNext;
    pCurrNode->pLoop = pLoopNode; //The current node is the last in
the experiment
    return(0);
}

//Begin a SAIM experiment with a given setup
void setup_experiment(ExpSetup* pSetup)
{
    int ExpNum = pSetup->ExpNum;
    ptrdiff_t StepNum = pSetup->StepNum;
    int LoopOn = pSetup->LoopOn;
    int ActivationTime = pSetup->ActivationTime;
    int ActivationIntensity = pSetup->ActivationIntensity;

    FIRE_OFF(); //Turn off Fire interrupt while setting up experiment
    AOTF_SHT = 0; //AOTF global low

    if(!ExpHeads[ExpNum]) //Check the experiment exists
    {
        output_error(2);
        return;
    }

    if(ActivationTime) //If there is a non-zero pre-sequence
activation step time
    {
        set_scan_center(CSCenter);
        set_scan_radius(CSTIRF); //Scan TIRF
    }
}

```

```

GDAC_LOAD; //Update the DAC registers
WAIT_TS();
ADAC_RS; //Reset the AOTF to all zeros
int8 Ch = 0; //UV channel (405 nm on our scope)
update_ADAC_channel(&Ch, &ActivationIntensity); //Set the UV
intensity
ADAC_LOAD;
do{}while(!Flags.Ts); //Wait for the galvs to settle
AOTF_SHT = 1; //Begin the activation
delay_ms(ActivationTime);
AOTF_SHT = 0; //End of activation
}

Flags.SAIMLoop = 0;
if(LoopOn)
    Flags.SAIMLoop = 1;
StartNode = ExpHeads[ExpNum];
StartStep = StartNode->pSeqStart;
ptrdiff_t StepCount = 0;
do //Find the starting step for the experiment
{
    if(!StartNode) //If we've run out of nodes
    {
        output_error(3); //Throw an error and exit
        return;
    }
    if((ptrdiff_t)((StartNode->pSeqEnd - StartNode->pSeqStart) / 2)
>= (StepNum - StepCount))
    {
        StartStep = StartNode->pSeqStart + 2 * StepNum;
        StepCount = StepNum;
    }
    else
    {
        StepCount += (StartNode->pSeqEnd - StartNode->pSeqStart) /
2;
        StartNode = StartNode->pNext;
    }
}while(StepCount < StepNum);

//Put the system in the appropriate condition for the first
exposure
set_scan_center(CSCenter);
set_scan_radius(StartStep);
GDAC_LOAD; //Load the new values into the DAC registers
WAIT_TS();
update_ADAC_all(StartNode->pAOTF); //Change the AOTF output (Ts
>> update)
ADAC_LOAD;
LED_EXP = LED_SCN = Flags.SAIM = 1; //Set the appropriate bits
Flags.Paused = Flags.LastFrame = Flags.EndOfExp = 0;
MirrorDetectorTrigger = 0;
if(StartStep == StartNode->pSeqEnd)
{
    Flags.LastFrame = 1;

```

```

    if((Flags.SAIMLoop == 1) && (StartNode->pLoop != NULL))
    {
        StartNode = StartNode->pLoop;
        StartStep = StartNode->pSeqStart;
    }
    else if(StartNode->pNext != NULL)
    {
        StartNode = StartNode->pNext;
        StartStep = StartNode->pSeqStart;
    }
    else
        Flags.LastFrame = Flags.EndOfExp = 1; //This is a single
frame experiment (probably just testing), so make sure the the
experiment ends
    }
    else
        StartStep += 2;
    FIRE_ON();
}

void stop_experiment(void)
{
    if(Flags.Paused || Flags.SAIM)
    {
        Flags.SAIM = Flags.Paused = 0;
        LED_EXP = 0;
    }
}

/*Setup a SAIM experiment at a given step
//Command structure is:
//{CMD, ExpNum, MSBStepNum, LSBStepNum, BoolLoop, Activation
MSBTime(ms), LSBTime(ms), MSBIntensity, LSBIntensity}
*/
int8 start_experiment(int8* pCommand)
{
    static ExpSetup PreviousSetup;
    if(Flags.SAIM) stop_experiment(); //This prevents segfault caused
by having two experiments running
    if(!(*pCommand))
    {
        setup_experiment(&PreviousSetup);
        return 0;
    }
    pCommand++;
    PreviousSetup.ExpNum = (int)*pCommand++;
    if(!ExpHeads[PreviousSetup.ExpNum])
        return 1;
    PreviousSetup.StepNum = make16(*pCommand++, *pCommand++);
    PreviousSetup.LoopOn = (int)*pCommand++;
    PreviousSetup.ActivationTime = make16(*pCommand++, *pCommand++);
    PreviousSetup.ActivationIntensity = make16(*pCommand++,
*pCommand);
    setup_experiment(&PreviousSetup); //Run the setup
    return 0;
}

```

```

}

void pause_experiment(void)
{
    if(Flags.SAIM && !Flags.Paused)
    {
        Flags.SAIM = 0;
        Flags.Paused = 1;
        enable_interrupts(int_TIMER5);
    }
}

void resume_experiment(void)
{
    if(Flags.Paused)
    {
        disable_interrupts(int_TIMER5);
        LED_EXP = 1;
        Flags.Paused = 0;
        Flags.SAIM = 1;
    }
}

void restart_experiment(void)
{
    if(Flags.Paused)
    {
        Flags.Paused = 0;
        disable_interrupts(int_TIMER5);
        LED_EXP = 0;
    }
    if(Flags.SAIM)
    {
        Flags.SAIM = 0;
        LED_EXP = 0;
    }
    start_experiment(NULL);
}

```

## File: Simple\_SAIM.c

```
/*
Driver for doing a simple SAIM experiment walking through
a set of angles from 0 deg to a user defined upper limit.

Copyright 2019 Marshall J. Colville (mjc449@cornell.edu)

Redistribution and use in source and binary forms, with or without
modification,
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright
notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice,
this list of conditions and the following disclaimer in the
documentation and/or
other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
*/

//Global variables
static int currStep;           //Current step number
static int steps;             //Number of steps in the experiment
static int8 direction;        //Direction to step in
const int maxSteps = 128;     //Maximum size of the experiment
int stepListX[maxSteps];      //List of positions to step through along X
int stepListY[maxSteps];      //List of positions to step through along Y

int create_simple(int& stepSize)
{
    int code = 0;
    if(steps > maxSteps)
```

```

    {
        steps = maxSteps;    //Change the number of steps to the max
array size
        code = 1;           //let the caller know steps were truncated
    }
    //If the steps are too big or will go past TIR
    int limit = CSCenter[0] + CSTIRF[0];
    if((stepSize > 0x0200) || (steps * stepSize > limit))
    {
        output_error(5); //Visual error indicator
        return 2;       //Exit without building the experiment
    }
    stepListX[0] = CSCenter[0]; //If no center is specified use the
stepListY[0] = CSCenter[1]; //current CSCenter
    for(int i = 1; i < steps; i++)
    {
        stepListX[i] = stepListX[i-1]-stepSize;
        stepListY[i] = stepListY[i-1]-stepSize;
    }
    return code;
}

//Overloaded function to specify a starting position different
//from the CSCenter position
int create_simple(int& stepSize, int& startPosX, int&startPosY)
{
    int code = 0;
    int highLim = CSCenter[0] + CSTIRF[0];
    int lowLim = CSCenter[0] - CSTIRF[0];
    if(steps > maxSteps)
    {
        steps = maxSteps;
        code = 1;
    }
    if((stepSize > 0x0200) ||
        (startPosX < lowLim) ||
        (startPosY < lowLim) ||
        (startPosX + steps * stepSize > highLim) ||
        (startPosY + steps * stepSize > highLim))
    {
        output_error(5); //Visual error indicator
        return 2;
    }
    stepListX[0] = startPosX;
    stepListY[0] = startPosY;
    for(int i = 1; i < steps; i++)
    {
        stepListX[i] = stepListX[i-1]-stepSize;
        stepListY[i] = stepListY[i-1]-stepSize;
    }
    return code;
}

void start_simple()
{

```

```

    center_park();
    stop_experiment();
    currStep = 0;
    Flags.SimpleSAIM = LED_EXP = 1;
    FIRE_ON();
}

void start_dithered(int* dither)
{
    center_park();
    stop_experiment();
    currStep = 0;
    Flags.SimpleSAIM = LED_EXP = LED_SCN = 1;
    if(direction) y_amplitude(dither); //Direction = 1, scan x,
dither y
    else x_amplitude(dither); //Direction = 2, scan y, dither x
    FIRE_ON();
}

void stop_simple()
{
    if(Flags.SimpleSAIM)
    {
        Flags.SimpleSAIM = LED_EXP = LED_SCN = 0;
        currStep = 0;
        park_center();
    }
}

```

## File: AD5440\_drvr.c

```
/*
Library for using AD5440 dual channel 10 bit DAC on the SAIMScannerV3
hardware
The outputs are configured for bipolar, 4-quadrant multiplication
with +10V
references. The references provide trimmers on jumpers for < 10V
values.
```

Copyright 2019 Marshall J. Colville (mjc449@cornell.edu)

Redistribution and use in source and binary forms, with or without  
modification,  
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright  
notice,  
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright  
notice,  
this list of conditions and the following disclaimer in the  
documentation and/or  
other materials provided with the distribution.

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

```
//Macros
```

```
#define AUX_LOAD bit_clear(LATG, 12); bit_set(LATG, 12);
```

```
//Global variables
```

```
static int AUXMidscale[2] = {0x01FF, 0x01FF}; //Midscale output for
Aux DACs
```

```
//Writes the referenced value to the DAC A output
```

```

void write_aux_a(int& value)
{
    bit_clear(LATG, 13);
    if(value > 0x3FFF) //Check that the value is within the bounds
        output_e(0x3FFF); //If not, output max code
    else
        output_e(value);
    AUX_LOAD;
}

//Writes the referenced value to the DAC B output
void write_aux_b(int& value)
{
    bit_set(LATG, 13);
    if(value > 0x3FFF)
        output_e(0x3FFF);
    else
        output_e(value);
    AUX_LOAD;
}

```

## **File: AD5547\_dual\_drvr.c**

```
/*
Library for using two AD5547 dual channel 16 bit DACs with AD9833 DDS
chips
for sine output and 10V DC refs on the SAIMScannerV3 hardware
The chips are configured for simultaneous synchronous loads,
maintaining
circularity of the output scan
Functionality includes point scanning, raster scanning, and circle
scanning
with ellipticity control
Basic IO macros are defined in firmwave_0_0.h:
    X_AMP sets the X amplitude address bits
    X_DC sets the X DC bias address bits
    Y_AMP sets the Y amplitude address bits
    Y_DC sets the Y DC bias address bits

    X_WRITE pulses the X write pin
    Y_WRITE pulses the Y write pin
    GDAC_LOAD pulses the load pin on both DACs, updating all 4 outputs
```

Copyright 2019 Marshall J. Colville (mjc449@cornell.edu)

Redistribution and use in source and binary forms, with or without  
modification,  
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright  
notice,  
this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright  
notice,  
this list of conditions and the following disclaimer in the  
documentation and/or  
other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  
"AS IS" AND  
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
IMPLIED  
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
ARE DISCLAIMED.  
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR  
ANY DIRECT,  
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES  
(INCLUDING,  
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS  
OF USE,  
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  
THEORY OF  
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING  
NEGLIGENCE

```

OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
*/

```

```

//Quick macros to handle basic operations
#define X_AMP bit_set(LATA, 9); bit_set(LATA, 10);
#define X_DC bit_clear(LATA, 9); bit_clear(LATA, 10);
#define Y_AMP bit_clear(LATA, 9); bit_clear(LATA, 10);
#define Y_DC bit_set(LATA, 9); bit_set(LATA, 10);
#define X_WRITE bit_clear(LATF, 13); bit_set(LATF, 13);
#define Y_WRITE bit_clear(LATF, 4); bit_set(LATF, 4);
#define GDAC_LOAD bit_set(LATF, 12); bit_clear(LATF, 12);
#define GDAC_RESET bit_clear(LATA, 1); bit_set(LATA, 1);

//Global variables
static int CSCenter[] = {0x7FFF, 0x7FFF}; //Storage for the circle-
scan center
static int CSRradius[] = {0x0000, 0x0000}; //Storage for the circle-
scan radius
static int CSTIRF[] = {0x3D00, 0x3D00}; //Initialized to approximate
TIRF for 100x NA 1.49, 300 mm TL, 10x BE
static int DiscSineX [32]; //Storage for a discretized sine wave
static int DiscSineY [32];
static int Tmr2Reset = 0x0000; //Reset value on TMR2 overflow for
point scanning
static int* PointListX = NULL; //Pointers to the lists of points to
scan
static int* PointListY = NULL;
static int ScanPosition = 0; //Current value in the point scanning
loop
static int ScanRollover = 0; //Rollover value for the point scanning
loop

static int32 UnitCircle[] =
{
    0xFFFF, 0xFD89, 0xF640, 0xEA6C, 0xDA81, 0xC71C, 0xB0FB, 0x98F8,
    0x7FFF, 0x6706, 0x4F03, 0x38E2, 0x257D, 0x1592, 0x09BE, 0x0275,
    0x0000, 0x0275, 0x09BE, 0x1592, 0x257D, 0x38E2, 0x4F03, 0x6706,
    0x7FFF, 0x98F8, 0xB0FB, 0xC71C, 0xDA81, 0xEA6C, 0xF640, 0xFD89
};

//Function to set the DC offset of the x galvo
void x_offset(int* pValue)
{
    X_DC;
    output_b(*pValue); //Write value to the parallel port
    X_WRITE;
}

//Function to set the DC offset of the y galvo
void y_offset(int* pValue)
{
    Y_DC;
    output_b(*pValue);
}

```

```

    Y_WRITE;
}

//Function to set the amplitude of the x sine output
void x_amplitude(int* pValue)
{
    X_AMP;
    output_b(*pValue);
    X_WRITE;
}

//Function to set the amplitude of the y sine output
void y_amplitude(int* pValue)
{
    Y_AMP;
    output_b(*pValue);
    Y_WRITE;
}

//Function to center the galvos and park them
void center_park(void)
{
    x_amplitude(&Zero[0]);
    y_amplitude(&Zero[1]);
    x_offset(&CSCenter[0]);
    y_offset(&CSCenter[1]);
    GDAC_LOAD;
}

//Function to be called from "firmware_0_0.c" to set the scan center
void set_scan_center(int* pArray)
{
    x_offset(&pArray[0]);
    y_offset(&pArray[1]);
}

//Function to be called from "firmware_0_0.c" to set the scan radius
void set_scan_radius(int* pArray)
{
    x_amplitude(&pArray[0]);
    y_amplitude(&pArray[1]);
}

//Function to calculate a discretized sine wave for comparison to DDS
output
void compute_circle (int32 Radius)
{
    int32 Offset = 0x00008000 - (Radius / 2);
    int8 YPos = 8;
    for (int i = 0; i < 32; i++)
    {
        //int32 circleValueX = unitCircle16[i];
        //int32 circleValueY = unitCircle16[y_pos];

        int XValue =

```

```

        ( (Radius * UnitCircle[i]) / 0x0000FFFF) + Offset;

        int32 YValue =
        ( (Radius * UnitCircle[YPos]) / 0x0000FFFF) + Offset;
        DiscSineX[i] = make16(make8(XValue, 1), make8(XValue, 0));
        DiscSineY[i] = make16(make8(YValue, 1), make8(YValue, 0));
        YPos++;
        if (YPos == 32)
        {
            YPos = 0;
        }
    }
}

//Set and update the scan radius
void load_CS_radius(int8* pValues)
{
    int1 resume = 0;
    if(Flags.Fire)
        resume = 1;
    FIRE_OFF();
    CSRadius[0] = make16(*pValues++, *pValues++);
    CSRadius[1] = make16(*pValues++, *pValues);
    set_scan_radius (CSRadius);
    set_scan_center (CScenter);
    GDAC_LOAD;
    LED_SCN = 1;
    if(resume)
        FIRE_ON();
}

//Set and update the scan radius
void load_CS_center(int8* pValues)
{
    int1 resume = 0;
    if(Flags.Fire)
        resume = 1;
    FIRE_OFF();
    CSCenter[0] = make16 (*pValues++, *pValues++);
    CSCenter[1] = make16 (*pValues++, *pValues);
    set_scan_center (CSCenter);
    GDAC_LOAD;
    if(resume)
        FIRE_ON();
}

//Set and update the TIRF scan radius
void load_CS_TIRF(int8* pValues)
{
    int1 resume = 0;
    if(Flags.Fire)
        resume = 1;
    FIRE_OFF();
    CSTIRF[0] = make16(*pValues++, *pValues++);
    CSTIRF[1] = make16(*pValues++, *pValues);
}

```

```

    set_scan_radius(CSTIRF);
    GDAC_LOAD;
    LED_SCN = 1;
    if(resume)
        FIRE_ON();
}

//Set the scan at the stored CS values
void circle_scan(void)
{
    int1 resume = 0;
    if(Flags.Fire)
        resume = 1;
    FIRE_OFF();
    set_scan_radius (CSRadius);
    set_scan_center (CSCenter);
    GDAC_LOAD;
    LED_SCN = 1;
    if(resume)
        FIRE_ON();
}

//Set the scan at the stored TIRF values
void TIRF_scan(void)
{
    int1 resume = 0;
    if(Flags.Fire)
        resume = 1;
    FIRE_OFF();
    set_scan_radius(CSTIRF);
    set_scan_center(CSCenter);
    GDAC_LOAD;
    LED_SCN = 1;
    if(resume)
        FIRE_ON();
}

//Park the beam at a designated coordinate pair
void park_location(int8* pValues)
{
    int1 resume = 0;
    if(Flags.Fire)
        resume = 1;
    FIRE_OFF();
    int Point[2];
    Point[0] = make16(*pValues++, *pValues++);
    Point[1] = make16(*pValues++, *pValues);
    set_scan_radius(Zero);
    set_scan_center(Point);
    GDAC_LOAD;
    LED_SCN = 0;
    if(resume)
        FIRE_ON();
}

```

```

//Stop a discretized circle scan
void stop_discrete_scan(void)
{
    if(Flags.DiscScan)
    {
        disable_interrupts(int_TIMER2);
        center_park();
        delay_us(500); //Delay to allow galvs to settle before next
move
        LED_SCN = 0;
        Flags.DiscScan = 0;
    }
}

//Load and begin a discretized circle scan
void discrete_circle_scan(int8* pValues)
{
    int1 resume = 0;
    if(Flags.Fire)
        resume = 1;
    FIRE_OFF();
    stop_discrete_scan();
    Flags.DiscScan = 1;
    compute_circle(makel6(*pValues++, *pValues++));
    Tmr2Reset = makel6(*pValues++, *pValues);
    ScanRollover = 31;
    PointListX = &DiscSineX[0];
    PointListY = &DiscSineY[0];
    enable_interrupts(int_TIMER2);
    LED_SCN = 1;
    if(resume)
        FIRE_ON();
}

//Park the beam at the center position
void park_center(void)
{
    FIRE_OFF();
    center_park();
    LED_SCN = 0;
}

```

**File: AD5583\_dual\_drvr.c**

```
/*
Library for using two AD5583 quad channel 10 bit DACs for DC output
to the AOTF controller on the SAIMScannerV3 hardware.
The chips are configured for simultaneous synchronous loads, data
must be
latched serially

Basic IO macros are defined in firmwave_0_0.h:
  X_AMP sets the X amplitude address bits
  X_DC sets the X DC bias address bits
  Y_AMP sets the Y amplitude address bits
  Y_DC sets the Y DC bias address bits

  X_WRITE pulses the X write pin
  Y_WRITE pulses the Y write pin
  GDAC_LOAD pulses the load pin on both DACs, updating all 4 outputs

Copyright 2019 Marshall J. Colville (mjc449@cornell.edu)

Redistribution and use in source and binary forms, with or without
modification,
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright
notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice,
this list of conditions and the following disclaimer in the
documentation and/or
other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

```

#define ADAC_LOAD bit_clear(LATD, 14); bit_set(LATD, 14);
#define ADAC0_WR bit_clear(LATD, 12); bit_set(LATD, 12);
#define ADAC1_WR bit_clear(LATD, 13); bit_set(LATD, 13);
#define ADAC_RS bit_clear(LATD, 15); bit_set(LATD, 15);
#define CHA bit_clear(LATD, 11); bit_clear(LATD, 10);
#define CHB bit_clear(LATD, 11); bit_set(LATD, 10);
#define CHC bit_set(LATD, 11); bit_clear(LATD, 10);
#define CHD bit_set(LATD, 11); bit_set(LATD, 11);

//Global variables
const size_t MaxAOTF = 32;
static int* AOTFArray[MaxAOTF][8]; //Addresses of the AOTF profile
locations on the heap

//Update the data registers on a ADAC channel, must be followed an
external call to ADAC_LOAD
void update_ADAC_channel(int8* pChannel, int* pValue)
{
    if(*pValue > 0x03FF) //Check that the value is within range
        *pValue = 0x03FF; //If it's too big, set it to max

    switch(*pChannel)
    {
        case 7:
            output_d(0b1111100000000000 | *pValue);
            ADAC1_WR;
            break;
        case 6:
            output_d(0b1111101000000000 | *pValue);
            ADAC1_WR;
            break;
        case 5:
            output_d(0b1111110000000000 | *pValue);
            ADAC1_WR;
            break;
        case 4:
            output_d(0b1111111000000000 | *pValue);
            ADAC1_WR;
            break;
        case 3:
            output_d(0b1111100000000000 | *pValue);
            ADAC0_WR;
            break;
        case 2:
            output_d(0b1111101000000000 | *pValue);
            ADAC0_WR;
            break;
        case 1:
            output_d(0b1111110000000000 | *pValue);
            ADAC0_WR;
            break;
        case 0:
            output_d(0b1111111000000000 | *pValue);
    }
}

```

```

        ADAC0_WR;
        break;
    default:
        output_error(5);
        break;
    }
}

//Update all 8 AOTF channels, must be followed by an external call to
ADAC_LOAD
void update_ADAC_all(int* pProfile)
{
    if(pProfile) //Check that the profile exists
    {
        int8 channel = 0;
        for(int i = 0; i <= 7; i++)
        {
            update_ADAC_channel(&channel, pProfile++);
            channel++;
        }
    }
    else //If the profile is NULL (deleted or unassigned)
        output_error(5); //AOTF errors are 5 blinks
}

//Creates a new ADAC profile on the heap and returns a pointer to the
profile
//Returns 0 if the operation succeeds, 1 if it fails
int new_ADAC_profile(int8* pCommand)
{
    int num = *pCommand++;
    for (int i = 0; i <= 7; i++) //Add the value for each channel
        AOTFArray[num][i] = make16(*pCommand++, *pCommand++);
    return(0);
}

int load_AOTF_profile(int Profile)
{
    if(Profile >= MaxAOTF)
    {
        output_error(5);
        return 2;
    }
    update_ADAC_all(&AOTFArray[Profile][0]); //Load the requested
profile
    ADAC_LOAD; //Update the DAC output
    return 0;
}

```

## File: AD9833\_dual\_drvr.c

```
/*
Library for using two AD9833 DDS chips
The DDSs share SYNC and SCK lines, but have separate SDI line
Serial data is shifted into both DDS units simultaneously

For the SAIMScannerV3 hardware we only ever want a sine output at <
1.1 kHz,
so control options are limited to these conditions and reset

Copyright 2019 Marshall J. Colville (mjc449@cornell.edu)

Redistribution and use in source and binary forms, with or without
modification,
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright
notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice,
this list of conditions and the following disclaimer in the
documentation and/or
other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#bit WAVE_XDO=LATC.4
#bit WAVE_YDO=LATG.8

//Global Variables
static int ControlWord = 0;
static int OnekHz = 0x0090; //Default scan frequency
```

```

static int Phase90 = 0x0400; //Default phase between x and y DDS
outputs
static int FrequencyX = OnekHz;
static int FrequencyY = OnekHz;
static int Phase = Phase90;

void wave_xfer(int& XMessage, int& YMessage) //Function to transfer
data
{
    bit_clear(LATG, 7); //Active low serial frame sync, stays low for
16 clocks
    for (int i = 0; i < 16; i++)
    {
        bit_set(LATG, 6);
        //Write the leftmost bit to the appropriate pin and shift in a
0 from right
        WAVE_XDO = shift_left(&XMessage, 2, 0);
        WAVE_YDO = shift_left(&YMessage, 2, 0);
        //Pulse the clock
        bit_clear(LATG, 6);
    }
    bit_set(LATG, 7); //End of transfer, raise sync
    bit_set(LATG, 6); //Idle the SCK
}

//Function to set the reset bit
void wave_reset(int1 RSState)
{
    if(RSState) //State = 1 enables reset, 0 clears reset
        bit_set(ControlWord, 8);
    else
        bit_clear(ControlWord, 8);
    int XMessage = ControlWord;
    int YMessage = ControlWord;
    wave_xfer(XMessage, YMessage);
}

//Function to set the MSB or LSB of freq registers
//For the SAIM scanner with 25 MHz MCLK writing 0x00 to the MSB
limits the
//maximum output freq to 1525 Hz, which is faster than we can run the
galvs.
//Therefore, we write 0x00 once to the MSB at startup, then all
writes
//during operation are to the LSB
void wave_freq_range(int1 HighLow)
{
    if(HighLow) //HighLow=1 writes to 14 MSB of freq reg, 0 writes to
LSB
        bit_set(ControlWord, 12);
    else
        bit_clear(ControlWord, 12);
    int XMessage = ControlWord;
    int YMessage = ControlWord;
    wave_xfer(XMessage, YMessage);
}

```

```

}

//Function to write to the frequency registers
//We only use FREQ0, and both DDS units share MCLK, so the same
frequency is
//written to both chips
void wave_set_freq(int value)
{
    if(value > 0x2E23) //This corresponds to an output freq of 1.1
kHz
        value = 0x2E23; //To avoid damaging the galvs, limit the
maximum freq
    value |= 0x4000;
    FrequencyX = FrequencyY = value;
    int XMessage = value;
    int yMessage = value;
    wave_xfer(XMessage, YMessage);
}

//Write a new frequency value to a single axis
void wave_set_axis_freq(int xValue, int yValue)
{
    if(xValue > 0x2E23)
        xValue = 0x2E23;
    if(yValue > 0x2E23)
        yValue = 0x2E23;
    xValue |= 0x4000;
    yValue |= 0x4000;
    wave_xfer(xValue, yValue);
}

//Function to write the phase offset between the DDS chips
//Because we are scanning a circle the phase offset in Y begins at
pi/2
//Therefore, the y phase will be value + pi/2
void wave_set_phase_diff(int value)
{
    if(value > 0x0FFF) //Maximum allowable phase
        value = 0x0FFF; //Limit to maximum
    int XMessage = 0xC000; //Phase X = 0;
    int YMessage = value | 0xC000; //Phase Y = value + pi/2
    wave_xfer(XMessage, YMessage);
    Phase = value;
}

void wave_shape(int shape)
{
    switch(shape)
    {
    {
    case 0:
        bit_clear(ControlWord, 5);
        bit_clear(ControlWord, 1);
        break;
    case 1:
        bit_clear(ControlWord, 5);

```

```
        bit_set(ControlWord, 1);
        break;
    case 2:
        bit_set(ControlWord, 5);
        bit_clear(ControlWord, 1);
        break;
    }
    int XMessage = ControlWord;
    int YMessage = ControlWord;
    wave_xfer(XMessage, YMessage);
}
```

## Appendix D

Instrument controller API source code. The following was written in C/C++ to the C++17 standard in Microsoft Visual Studio and compiled with the Visual C++ 2017. The code compiles to a dynamic library which forms an intermediary layer between the controller firmware and graphical interface. The API is designed to abstract the details of the communications protocol, providing a convenient interface for building custom graphical interfaces or for integration into existing acquisition software. Compiling the code requires the open-source and free HIDAPI library available at <http://github.com/signal11/hidapi>.

## File: SAIMScannerV3.h

```
/**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
/  
//  
//  
// SAIMScannerV3 - an open-source microscope controller providing an  
//  
// embedded solution for hardware synchronization.  
//  
// This library provides a compiler independent interface  
//  
// with the controller hardware on Windows systems.  
//  
// Many functions are Scanning Angle Interference Microscopy  
//  
// (SAIM) specific, however the hardware and underlying  
//  
// functionality is designed to be versatile and applicable  
//  
// in a variety of applications.  
//  
//  
// While devices can be created in applications using a  
//  
// variety of methods, the recommended approach is using a  
//  
// smart pointer, e.g.:  
//  
//  
// #include <functional>  
//  
// typedef std::shared_ptr<SSV3::Controller> pScanCard;  
//  
// pScanCard scancard(SSV3::CreateDevice(),  
//  
//     std::mem_fn(&(SSV3::Controller::Destroy)));  
//  
//  
// In this way the controller is shutdown to a safe state  
//  
// when the application ends or in the case of an error  
//  
//  
//  
// Copyright(c) 2018, Marshall Colville mj449@cornell.edu  
//  
// All rights reserved.  
//  
//  
//
```



```

#ifndef SAIMSCANNERV3_H_
#define SAIMSCANNERV3_H_

#ifdef MAKE_DLL_
#define SSV3API_ __declspec(dllexport)
#else
#define SSV3API_ __declspec(dllimport)
#endif //MAKE_DLL_

namespace SSV3
{
    typedef class Manager * SSV3Manager;
    typedef class Controller * SSV3Controller;

    /**Base Ssv3 controller class wraps the HID communications into opaque\n
    //function calls facilitating rapid, easy integration*/
    class Controller
    {
    public:
        virtual ~Controller() = default;

        /**Return error codes*/
        enum class SSV3ERROR
        {
            SSV3ERROR_OK,
            SSV3ERROR_NO_DEVICES_FOUND,
            SSV3ERROR_DEVICE_UNAVAILABLE,
            SSV3ERROR_NO_RESPONSE,
            SSV3ERROR_UNEXPECTED_RETURN,
            SSV3ERROR_XMIT_FAIL,
            SSV3ERROR_READ_TOO_SHORT,
            SSV3ERROR_EXCITATION_FULL,
            SSV3ERROR_ALLOC_FAIL,
            SSV3ERROR_EXCITATION_PROFILE_OUT_OF_RANGE,
            SSV3ERROR_EXCITATION_PROFILE_DOESNT_EXIST,
            SSV3ERROR_SEQUENCES_FULL,
            SSV3ERROR_SEQUENCE_ALLOCATION_FAIL,
            SSV3ERROR_SEQUENCE_LENGTH_ZERO,
            SSV3ERROR_SEQUENCE_DOESNT_EXIST,
            SSV3ERROR_SEQUENCE_LOAD_FAILED,
            SSV3ERROR_NO_EXPERIMENT,
            SSV3ERROR_INVALID_LOOP,
            SSV3ERROR_STEP_OUTSIDE_EXPERIMENT_RANGE,
            SSV3ERROR_COULDNT_OPEN_ERR_LOG
        };

        /**Disconnects the controller and destroys the instance.\n
        //Handle must not be used after calling Destroy\n
        //To reattach the controller create a new instance*/
        virtual void Destroy() = 0;

        /**Check for hidapi errors\n
        //Returns a descriptive string of the error or NULL*/
        virtual const wchar_t * HidError() = 0;
    };
}

```

```

/**Sets the controller to a safe beginning state\n
//and prints the manufacturer and product strings to the command out*/
virtual SSV3ERROR Initialize() = 0;

/**Change the response timeout value
//@param ms = timeout value in ms*/
virtual void Timeout(unsigned int ms) = 0;

/**Set the maximum number of retries on read calls
//@param attempts = number or attempts before failure, default is 2*/
virtual void ReadRetries(unsigned int attempts) = 0;

/**Runs a quick visor across the status LEDS on the front\n
//of the controller*/
virtual SSV3ERROR Visor() = 0;

/**Centers the beam, stops scanning, and closes the shutter*/
virtual SSV3ERROR CenterPark() = 0;

/**Stops scanning and parks the beam\n
//location[0] = x inclination, location[1] = y inclination\n
//@param location = park location in DAC units*/
virtual SSV3ERROR LocationPark(unsigned short *location) = 0;

/**Open or close the mechanical shutter\n
//@param state - true for open, false for close*/
virtual SSV3ERROR Shutter(bool state) = 0;

/**Blanks all laser lines via the AOTF\n
//@param state = false blanks lasers (off), true enables lasers (on)*/
virtual SSV3ERROR AOTFBlank(bool state) = 0;

/**Set the absolute intensity for a specific laser line\n
//@param line = laser to be adjusted\n
//@param value = power (max 1023)*/
virtual SSV3ERROR SingleLaserPower(unsigned char line, unsigned short
value) = 0;

/**Create an excitation profile object\n
//The newly initialized profile will be all 0s (all lasers off)\n
//If no number for the profile is specified the next available\n
//number will be used,\n
//If all profiles are full and no number is specified the call will
fail\n
//@param profile = the number ID of the profile being created\n
//@param values = optional pointer to array of 8 laser values\n*/
virtual SSV3ERROR MakeExcitationProfile(const unsigned char profile,
unsigned short *values = nullptr) = 0;

/**Set the value of a line in a excitation profile and switch to\n
// the profile\n
//@param profile = handle for the profile being changed\n
//@param line = laser line number\n
//@param value = new value (max 1023)\n*/

```

```

    virtual SSV3ERROR SetProfilePower(const unsigned char profile, const
unsigned char line, const unsigned short value) = 0;

    /**Load an excitation profile\n
    //@param profile = excitation settings handle to load*/
    virtual SSV3ERROR LoadExcitationProfile(const unsigned char profile) = 0;

    /**Create a new profile with the current laser settings\n
    //If profile already exists the settings will be overwritten\n
    //@param profile = handle to the new excitation profile*/
    virtual SSV3ERROR MakeProfileFromCurrentExcitation(const unsigned char
profile) = 0;

    /**Turn on or off the camera triggered excitation\n
    //@param state = true for on, false for off*/
    virtual SSV3ERROR Fire(bool state) = 0;

    /**Zeros all 8 laser lines without changing any of the stored profiles*/
    virtual SSV3ERROR ClearExcitation() = 0;

    /**Adjust the phase value\n
    //0 deg = 0x000, 90 deg ~0x400, 180 deg ~0x800\n
    //@param phase = waveform phase offset*/
    virtual SSV3ERROR AdjustPhase(unsigned short phase) = 0;

    /**Adjust the scan frequency\n
    //0 Hz = 0x00, 1 kHz = 0x29f1\n
    //Maximum allowed value is 1 kHz\n
    //@param frequency = circle scan frequency*/
    virtual SSV3ERROR AdjustFrequency(unsigned short frequency) = 0;

    /**Set scan amplitude offset in the y axis.\n
    //Used to correct ellipticity in the scan profile.\n
    //0x7fff corresponds to 0 offset\n
    //@param value = relative scaling of the Y-axis*/
    virtual SSV3ERROR YAmplitudeCorrection(unsigned short value) = 0;

    /**Update the current scan radius\n
    //Radius is limited to 0x4200 to protect the galvos from being
overdriven\n
    //@param radius = desired scan radius in DAC units*/
    virtual SSV3ERROR ScanRadius(unsigned short radius) = 0;

    /**Changes the DC bias on one of the galvos\n
    //@param axis = axis to move, true for x, false for y\n
    //@param value = center (0 bias) is 0x7fff*/
    virtual SSV3ERROR ScanCenter(bool axis, unsigned short value) = 0;

    /**Change the TIR scan value\n
    //Also update current scan radius to new TIR value\n
    //If value == 0 or default TIR value is not update and the scan\n
    // radius is set to the current TIR value\n
    //@param value = new scan radius (limited to 0x4000)*/
    virtual SSV3ERROR TIRF(unsigned short value = 0) = 0;

    /**Load a set of radii into the controller memory\n

```

```

//The radii are specified as a 2-byte DAC value corresponding to the\n
// x-galvo amplitude. The y-offset will be used to calculate\n
// the y-galvo amplitude internally.\n
//Returns an error the number parameter is specified and sequence\n
// doesn't exist\n
//@param sequence = sequence number in internal memory\n
//@param length = size of the sequence (number of steps)\n
//@param values = array of values to be loaded*/
virtual SSV3ERROR LoadAngles(const unsigned char sequence, const unsigned
short length, unsigned short *values) = 0;

/**Add an angle sequence and a excitation setting to the end of the\n
// experiment.\n
//@param sequence = set of angles to add\n
//@param excitation = excitation profile to add\n
//@param step = the step number in the sequence if step < 0 step will be
inserted at the end if a step already exists it will be overwritten*/
virtual SSV3ERROR AddExperimentStep(const unsigned char sequence, const
unsigned char excitation, const int step = -1) = 0;

/**Clears the experiment of all steps*/
virtual SSV3ERROR ClearExperiment() = 0;

/**Create or destroy a loop in the experiment.\n
//When the end of the experiment list is reached it will loop back to
the\n
// step specified.\n
//@param onOff = turn the loop on or off\n
//@param loopTo = number of the experiment step to loop to*/
virtual SSV3ERROR Loop(bool onOff, const unsigned int loopTo = 0) = 0;

/**Begin the currently programmed experiment.*/
virtual SSV3ERROR StartExperiment() = 0;

/**Stop the current experiment and return to the state at the beginning\n
// of the experiment*/
virtual SSV3ERROR StopExperiment() = 0;

/**Send a single pulse to simulate a camera frame (positive polarity)\n
//@param period = the pulse length (0x00 ~80 us, 0xff ~32 ms)*/
virtual SSV3ERROR SendSWTrigger(unsigned short period = 0xffff) = 0;

/**Send the input array to the controller.\n
//Low level direct access to the controller functions.\n
//Not meant for general use\n
//No check is made on the response other than that it was received\n
//The *msg argument is overwritten with the returned values\n
//@param array = byte array to be sent\n
//@param length = length of the packet, must not be greater than 64*/
virtual SSV3ERROR SendArray(unsigned char *msg, const size_t length = 0)
= 0;

/**Queries the device for the current scan settings from the controller
onboard memory. All parameters are returned in DAC units. This is
particularly useful at startup so that the current state of the controller can
be indentified.

```

```

    @param xCenter = scan center x value (DC offset of waveform)
    @param yCenter = scan center y value (DC offset of waveform)
    @param tirRadius = stored TIR value
    @param phase = stored phase difference between the output waveforms
    @param frequency = stored scan (waveform generator) frequency*/
    virtual SSV3ERROR QueryInternalSettings(unsigned short *xCenter, unsigned
short *yCenter, unsigned short *tirRadius, unsigned short *phase, unsigned
short *frequency) = 0;

    /**Queries the hardware and firmware versions of the controller
    @param brdMajor = major hardware revision
    @param brdMinor = minor hardware revision
    @param fwMajor = firmware major revision
    @param fwMinor = firmware minor revision*/
    virtual SSV3ERROR QueryDevVer(unsigned char *brdMajor, unsigned char
*brdMinor, unsigned char *fwMajor, unsigned char *fwMinor) = 0;

    /**Detailed error logging for debugging. This adds overhead to API
function calls so should be left off in normal operation*/
    virtual SSV3ERROR DetailedErrorReporting(bool onOff) = 0;

    /**Get the last error description, returns null if there are no errors*/
    virtual const char * GetLastError() = 0;

    /**Resets the device erasing all data and calls the deleter Destroy().\n
//The controller instance is no longer valid after call.\n
//It is the caller's responsibility to destroy any references.*/
    virtual void Reset() = 0;
};

    /**Simple wrapper around the hidapi library to add specificity for the SSV3
device. Makes attaching and tracking multiple devices easier.*/
    class Manager
    {
    public:
        /**Return error codes*/
        enum class SSV3MANAGER_ERROR
        {
            SSV3MANAGER_ERROR_OK,
            SSV3MANAGER_ERROR_INIT_FAILED,
            SSV3MANAGER_ERROR_NO_DEVICES,
            SSV3MANAGER_ERROR_DEVICE_INVALID
        };

        virtual ~Manager() = default;

        /**Disconnects the manager and frees the enumerated devices.
It is the caller's responsibility to ensure that the manager is
no longer referenced*/
        virtual void Destroy() = 0;

        /**Initializes the SSV3 device manager
        @param nDevs = number of detected devices*/
        virtual SSV3MANAGER_ERROR Enumerate(int *nDevs) = 0;

```

```

    /**Get the number of attached devices
    @param nDevs = number of devices found*/
    virtual SSV3MANAGER_ERROR DeviceCount(int *nDevs) = 0;

    /**Query the device ID
    @param dev = device number
    @param man = manufacturer string
    @param prod = product string
    @param sn = serial number string*/
    virtual SSV3MANAGER_ERROR GetDeviceInfo(const int dev, wchar_t *man,
    wchar_t *prod, wchar_t *sn) = 0;

    /**Refreshes the device list.
    Previous device counts and information may no longer be valid
    @param nDevs = number of devices found*/
    virtual SSV3MANAGER_ERROR RefreshDevices(int *nDevs) = 0;
};

#ifdef __cplusplus
extern "C" {
#endif
    SSV3API_ SSV3Controller __cdecl CreateDevice();
    SSV3API_ SSV3Controller __cdecl CreateDeviceFromSN(wchar_t *sn);
    SSV3API_ SSV3Controller __cdecl CreateDemoDevice(bool demo = true);
    SSV3API_ SSV3Manager __cdecl CreateManager();
#ifdef __cplusplus
}
#endif
}

#endif //SAIMSCANNERV3_H_

```

**File: SSV3Manager.cpp**

```
////////////////////////////////////  
/  
//  
//  
// SAIMScannerV3 - an open-source microscope controller providing an  
//  
// embedded solution for hardware synchronization.  
//  
// This library provides a compiler independent interface  
//  
// with the controller hardware on Windows systems.  
//  
// Many functions are Scanning Angle Interference Microscopy  
//  
// (SAIM) specific, however the hardware and underlying  
//  
// functionality is designed to be versatile and applicable  
//  
// in a variety of applications.  
//  
//  
//  
// Copyright(c) 2018, Marshall Colville mjc449@cornell.edu  
//  
// All rights reserved.  
//  
//  
//  
// Redistribution and use in source and binary forms, with or without  
//  
// modification, are permitted provided that the following conditions are  
//  
// met :  
//  
//  
//  
// 1. Redistributions of source code must retain the above copyright notice,  
//  
// this list of conditions and the following disclaimer.  
//  
// 2. Redistributions in binary form must reproduce the above copyright  
//  
// notice, this list of conditions and the following disclaimer in the  
//  
// documentation and/or other materials provided with the distribution.  
//  
//  
//  
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  
//  
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED  
//  
// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A  
//  
//
```



```

}

ERR Enumerate(int *nDevs)
{
    if (hid_init() == -1)
        return ERR::SSV3MANAGER_ERROR_INIT_FAILED;
    _devInfo = hid_enumerate(1240, 61722);
    _devCount = 0;
    hid_device_info *info = _devInfo;
    while (info != nullptr)
    {
        _devCount++;
        info = info->next;
    }
    *nDevs = _devCount;
    return ERR::SSV3MANAGER_ERROR_OK;
}

ERR DeviceCount(int *nDevs)
{
    if (_devCount < 0)
        return ERR::SSV3MANAGER_ERROR_INIT_FAILED;
    *nDevs = _devCount;
    return ERR::SSV3MANAGER_ERROR_OK;
}

ERR GetDeviceInfo(const int dev, wchar_t *man, wchar_t *prod, wchar_t
*sn)
{
    if (_devCount < 1)
        return ERR::SSV3MANAGER_ERROR_NO_DEVICES;
    hid_device_info *device = _devInfo;
    for (int i = 0; i < dev; i++)
    {
        if (device->next == nullptr)
            return ERR::SSV3MANAGER_ERROR_DEVICE_INVALID;
        device = device->next;
    }
    size_t manlen = wcsnlen_s(device->manufacturer_string, 64);
    size_t prodlen = wcsnlen_s(device->product_string, 64);
    size_t snlen = wcsnlen_s(device->serial_number, 64);
    wcsncpy_s(man, 63, device->manufacturer_string, manlen);
    wcsncpy_s(prod, 63, device->product_string, prodlen);
    wcsncpy_s(sn, 63, device->serial_number, snlen);
    return ERR::SSV3MANAGER_ERROR_OK;
}

ERR RefreshDevices(int *nDevs)
{
    _devCount = -1;
    hid_free_enumeration(_devInfo);
    return Enumerate(nDevs);
}

private:

```

```

////////////////////////////////////
// Data members
//
////////////////////////////////////
    hid_device_info *_devInfo{ nullptr };
    int _devCount{ -1 };
};

#ifdef __cplusplus
extern "C" {
#endif
    SSV3API_ SSV3Manager __cdecl CreateManager()
    {
        SSV3Manager p = new ScanCardManager;
        return p;
    }
#ifdef __cplusplus
}
#endif
}

```

**File: SSV3Device.cpp**

```
////////////////////////////////////  
/  
//  
//  
// SAIMScannerV3 - an open-source microscope controller providing an  
//  
// embedded solution for hardware synchronization.  
//  
// This library provides a compiler independent interface  
//  
// with the controller hardware on Windows systems.  
//  
// Many functions are Scanning Angle Interference Microscopy  
//  
// (SAIM) specific, however the hardware and underlying  
//  
// functionality is designed to be versatile and applicable  
//  
// in a variety of applications.  
//  
//  
//  
// Copyright(c) 2018, Marshall Colville mjc449@cornell.edu  
//  
// All rights reserved.  
//  
//  
//  
// Redistribution and use in source and binary forms, with or without  
//  
// modification, are permitted provided that the following conditions are  
//  
// met :  
//  
//  
//  
// 1. Redistributions of source code must retain the above copyright notice,  
//  
// this list of conditions and the following disclaimer.  
//  
// 2. Redistributions in binary form must reproduce the above copyright  
//  
// notice, this list of conditions and the following disclaimer in the  
//  
// documentation and/or other materials provided with the distribution.  
//  
//  
//  
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  
//  
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED  
//  
// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A  
//  
//
```



```

{
public:
    ScanCard()
    {
        _dev = hid_open(1240, 61722, NULL);
    }

    ScanCard(wchar_t *sn)
    {
        _dev = hid_open(1240, 61722, sn);
    }

    ScanCard(bool demo)
    {
        _demo = true;
    }

~ScanCard(){}

////////////////////////////////////
/* API functions */
////////////////////////////////////

void Destroy()
{
    //Make sure to stop any experiments, center the galvos
    // and close the shutter
    SSV3ERROR ret = OK_;
    _oBuffer[0] = 0x8F;
    SendAndListen(&ret, 1, true);
    _oBuffer[0] = 0x1F;
    SendAndListen(&ret, 1, true);
    _oBuffer[0] = 0x45;
    SendAndListen(& ret, 1, true);
    if(!_demo)
        hid_close(_dev);
    delete[] _errStr;
    delete this;
}

const wchar_t * HidError() { return hid_error(_dev); }

SSV3ERROR Initialize()
{
    SSV3ERROR ret{ OK_ };
    if (!_demo)
    {
        //Set the read attempts to a reasonable large value in case the
        // controller is a little slow
        _readAttempts = 100;
        ret = Visor();
        _readAttempts = 2;
        if (_detailedReporting)
        {
            _errMsg.str("");
            wchar_t string[64];

```

```

        std::wstring str;
        if (hid_get_manufacturer_string(_dev, string, 255) != 0)
        {
            _errMsg << "No response to manufacturer inquiry\n";
            _newErr = true;
        }
        if (hid_get_product_string(_dev, string, 255) != 0)
        {
            _errMsg << "No response to product inquiry\n";
            _newErr = true;
        }
    }
}
for (int i = 0; i < 8; i++)
    _currentExcitation.push_back(0);
for (int i = 0; i < 32; i++)
{
    _illuminationProfiles.push_back(std::vector<unsigned short>());
    _angleSequences.push_back(std::vector<unsigned short>());
}
return ret;
}

void Timeout(unsigned int ms) { _timeout = ms; }

void ReadRetries(unsigned int attempts) { _readAttempts = attempts; }

SSV3ERROR Visor()
{
    SSV3ERROR ret{ OK_ };
    _oBuffer[0] = 0x01;
    SendAndListen(&ret, 1, true);
    return ret;
}

SSV3ERROR CenterPark()
{
    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;
    _oBuffer[0] = 0x1F;
    SendAndListen(&ret, 1, true);
    if (ret != OK_)
        return ret;
    _oBuffer[0] = 0x45;
    SendAndListen(&ret, 1, true);
    return ret;
}

SSV3ERROR Shutter(bool state)
{
    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
}

```

```

        if (ret != OK_)
            return ret;
        _oBuffer[0] = state ? 0x44 : 0x45;
        SendAndListen(&ret, 1, true);
        return ret;
    }

SSV3ERROR AOTFBlank(bool state)
{
    SSV3ERROR ret{ OK_ };
    if (!_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;
    _oBuffer[0] = state ? 0x40 : 0x41;
    SendAndListen(&ret, 1, true);
    return ret;
}

SSV3ERROR SingleLaserPower(unsigned char line, unsigned short value)
{
    //Load the output buffer
    _oBuffer[0] = 0x42;
    _oBuffer[1] = line;
    _oBuffer[2] = HByte(value);
    _oBuffer[3] = LByte(value);
    SSV3ERROR ret{ OK_ };
    SendAndListen(&ret, 4, true);
    if (ret != OK_)
        return ret;
    //Update the current excitation values in case these will be assigned
a profile
    _currentExcitation[line] = value;
    return ret;
}

SSV3ERROR MakeExcitationProfile(const unsigned char profile, unsigned
short *values = nullptr)
{
    SSV3ERROR ret{ OK_ };
    if (!_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;

    if (profile > 31)
        return SSV3ERROR::SSV3ERROR_EXCITATION_PROFILE_OUT_OF_RANGE;
    _illuminationProfiles.at(profile).clear();

    unsigned short power;
    unsigned char *msg = _oBuffer;
    *msg++ = 0x4c;
    *msg++ = profile;
    for (int i = 0; i < 8; i++)
    {
        if (values == nullptr)

```

```

        power = 0;
    else
        power = values[i];
        _illuminationProfiles[profile].push_back(power);
        *msg++ = HByte(power);
        *msg++ = LByte(power);
    }
    SendAndListen(&ret, 0, true);
    if (_iBuffer[0] == 1)
        return SSV3ERROR::SSV3ERROR_ALLOC_FAIL;
    _experimentModified = true;
    return ret;
}

SSV3ERROR SetProfilePower(const unsigned char profile, const unsigned
char line, const unsigned short value)
{
    if (profile > 31 || _illuminationProfiles.at(profile).empty())
        return SSV3ERROR::SSV3ERROR_EXCITATION_PROFILE_DOESNT_EXIST;
    unsigned short newVal = value;

    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;

    //Check that the requested value is valid and adjust if needed
    newVal = newVal > 0x03ff ? 0x03ff : newVal;
    //Update the line value
    _illuminationProfiles.at(profile).at(line) = newVal;

    //Put the new values in the output buffer
    unsigned char *msg = _oBuffer;
    *msg++ = 0x4c;
    *msg++ = profile;
    //Load the entire value array into the output buffer
    for (int i = 0; i < 8; i++)
    {
        *msg++ = HByte(_illuminationProfiles[profile][i]);
        *msg++ = LByte(_illuminationProfiles[profile][i]);
        _currentExcitation.at(i) = _illuminationProfiles[profile][i];
    }
    SendAndListen(&ret, 0, true);
    if (ret != OK_)
        return ret;
    if (_iBuffer[0] == 1)
        return SSV3ERROR::SSV3ERROR_ALLOC_FAIL;
    for (int i = 1; i < 18; i++)
        if (_iBuffer[i] != _oBuffer[i])
            return SSV3ERROR::SSV3ERROR_UNEXPECTED_RETURN;

    msg = _oBuffer;
    *msg++ = 0x43;
    *msg = profile;
    SendAndListen(&ret, 1, true);
}

```

```

        if (ret != OK_)
            return ret;
        _usingExcitation = profile;

        _experimentModified = true;
        return ret;
    }

SSV3ERROR LoadExcitationProfile(const unsigned char profile)
{
    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;

    if (profile > 31 || _illuminationProfiles[profile].empty())
        return SSV3ERROR::SSV3ERROR_EXCITATION_PROFILE_DOESNT_EXIST;

    _oBuffer[0] = 0x43;
    _oBuffer[1] = profile;
    SendAndListen(&ret, 1, true);
    if (ret != OK_)
        return ret;
    if (_iBuffer[2] == 0x01)
        return SSV3ERROR::SSV3ERROR_EXCITATION_PROFILE_DOESNT_EXIST;
    else if (_iBuffer[2] == 0x02)
        return SSV3ERROR::SSV3ERROR_EXCITATION_PROFILE_OUT_OF_RANGE;
    _usingExcitation = profile;

    for (int i = 0; i < 8; i++)
        _currentExcitation.at(i) = _illuminationProfiles.at(profile).at(i);
    return ret;
}

SSV3ERROR MakeProfileFromCurrentExcitation(const unsigned char profile)
{
    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;

    //Max number of profiles exists
    if (profile > 31)
        return SSV3ERROR::SSV3ERROR_EXCITATION_FULL;
    else
        _illuminationProfiles.at(profile).clear();

    unsigned char *msg = _oBuffer;
    *msg++ = 0x4c;
    *msg++ = profile;
    for (int i = 0; i < 8; i++)
    {
        _illuminationProfiles.at(profile).push_back(_currentExcitation[i]);
        *msg++ = HByte(_currentExcitation[i]);
    }
}

```

```

        *msg++ = LByte(_currentExcitation[i]);
    }
    SendAndListen(&ret, 1, true);
    if (ret != OK_)
        return ret;
    if (_iBuffer[0] == 1)
        return SSV3ERROR::SSV3ERROR_ALLOC_FAIL;

    _usingExcitation = profile;
    _experimentModified = true;
    return ret;
}

SSV3ERROR Fire(bool state)
{
    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;

    _oBuffer[0] = state ? 0x50 : 0x51;
    SendAndListen(&ret, 1, true);
    return ret;
}

SSV3ERROR ClearExcitation()
{
    SSV3ERROR ret{ OK_ };
    //Loop through all 8 lines and set to 0;
    for (unsigned char i = 0; i < 8; i++)
    {
        unsigned char* msg = _oBuffer;
        *msg++ = 0x42;
        *msg++ = i;
        *msg++ = 0x00;
        *msg++ = 0x00;
        _currentExcitation[i] = 0;
        SendAndListen(&ret, 4, true);
        if (ret != OK_)
            return ret;
    }
    return ret;
}

SSV3ERROR AdjustPhase(unsigned short phase)
{
    SSV3ERROR ret{ OK_ };
    unsigned char *msg = _oBuffer;
    *msg++ = 0x22;
    *msg++ = HByte(phase);
    *msg++ = LByte(phase);
    SendAndListen(&ret, 3, true);
    return ret;
}

```

```

SSV3ERROR AdjustFrequency(unsigned short frequency)
{
    SSV3ERROR ret{ OK_ };
    unsigned short value = frequency > 0x2e23 ? 0x2e23 : frequency;
    unsigned char *msg = _oBuffer;
    *msg++ = 0x20;
    *msg++ = HByte(value);
    *msg++ = LByte(value);
    SendAndListen(&ret, 3, true);
    return ret;
}

SSV3ERROR YAmpCorrection(unsigned short value)
{
    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;
    //Just calculate the signed offset value and update radius
    _yOffset = (float)value / (float)0x7fff;
    //Send x-radius, y offset will be applied in ScanRadius()
    return SetRadius(_currentRadius);
}

SSV3ERROR ScanRadius(unsigned short radius)
{
    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;
    unsigned short value = radius > 0x8000 ? 0x8000 : radius;
    _currentRadius = value;
    return SetRadius(value);
}

SSV3ERROR ScanCenter(bool axis, unsigned short value)
{
    SSV3ERROR ret{ OK_ };
    if (axis)
        _scanCenter[0] = value;
    else
        _scanCenter[1] = value;
    unsigned char *msg = _oBuffer;
    *msg++ = 0x11;
    *msg++ = HByte(_scanCenter[0]);
    *msg++ = LByte(_scanCenter[0]);
    *msg++ = HByte(_scanCenter[1]);
    *msg = LByte(_scanCenter[1]);
    SendAndListen(&ret, 5, true);
    return ret;
}

SSV3ERROR LocationPark(unsigned short *location)
{

```

```

SSV3ERROR ret{ OK_ };
unsigned char *msg = _oBuffer;
*msg++ = 0x15;
*msg++ = HByte(location[0]);
*msg++ = LByte(location[0]);
*msg++ = HByte(location[1]);
*msg = LByte(location[1]);
SendAndListen(&ret, 5, true);
return ret;
}

SSV3ERROR TIRF(unsigned short value = 0)
{
    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;

    unsigned short newVal = value > 0x8000 ? 0x8000 : value;
    //If we just want to go to the currently saved TIRF value
    if (newVal == 0)
    {
        //Pass the current x TIR value, yoffset will be applied
        _oBuffer[0] = 0x14;
        SendAndListen(&ret, 1, true);
        return ret;
    }
    //Otherwise we will program a new TIR value into the scancard
    _tirRadius = newVal;
    unsigned short yVal = (unsigned short)((float)newVal * _yOffset);
    unsigned char *msg = _oBuffer;
    *msg++ = 0x12;
    *msg++ = HByte(newVal);
    *msg++ = LByte(newVal);
    *msg++ = HByte(yVal);
    *msg++ = LByte(yVal);
    SendAndListen(&ret, 5, true);
    return ret;
}

SSV3ERROR LoadAngles(const unsigned char sequence, const unsigned short
length, unsigned short *values)
{
    SSV3ERROR ret{ OK_ };
    if (_demo)
        return ret;
    int readWriteVal{};
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;

    if (sequence > 31)
        return SSV3ERROR::SSV3ERROR_SEQUENCE_DOESNT_EXIST;
}

```

```

//Calculate the number of packets required to transmit the sequence
//Each angle requires 4 bytes (2 for x, 2 for y)
//Maximum packet length is 64 bytes (32 2-byte DAC values)
unsigned short nPackets = (unsigned short)ceil(((float)length * 4.0) /
64.0);
unsigned char *msg = _oBuffer;
*msg++ = 0x80;
*msg++ = sequence;
*msg++ = HByte(length);
*msg++ = LByte(length);
*msg++ = HByte(nPackets);
*msg++ = LByte(nPackets);

SendAndListen(&ret, 0, true);
if (ret != OK_)
    return ret;
if (_iBuffer[0] == 1)
    return SSV3ERROR::SSV3ERROR_SEQUENCE_ALLOCATION_FAIL;
else if (_iBuffer[0] == 2)
    return SSV3ERROR::SSV3ERROR_SEQUENCE_LENGTH_ZERO;

_angleSequences.at(sequence).clear();
int counter = 0;
_newErr = false;
for (int i = 0; i < (nPackets - 1); i++)
{
    msg = _oBuffer;
    for (int j = 0; j < 16; j++)
    {
        _angleSequences.at(sequence).push_back(values[counter]);
        unsigned short yval = (unsigned short)((float)values[counter] *
_yOffset);
        *msg++ = HByte(values[counter]);
        *msg++ = LByte(values[counter]);
        *msg++ = HByte(yval);
        *msg++ = LByte(yval);
        counter++;
    }
    SendAndListen(&ret, 64, false);
    if (ret != OK_)
        return ret;
}

int remaining = length - counter; //Number of pairs left to send
msg = _oBuffer;
for (int i = 0; i < remaining; i++)
{
    _angleSequences.at(sequence).push_back(values[counter]);
    unsigned short yval = (unsigned short)((float)values[counter] *
_yOffset);
    *msg++ = HByte(values[counter]);
    *msg++ = LByte(values[counter]);
    *msg++ = HByte(yval);
    *msg++ = LByte(yval);
    counter++;
}

```

```

    }
    SendAndListen(&ret, 4 * remaining, false);
    if (ret != OK_)
        return ret;

    _experimentModified = true;
    if (hid_read(_dev, _iBuffer, 64) < 64)
        return SSV3ERROR::SSV3ERROR_NO_RESPONSE;
    if (_iBuffer[0] != 0)
        return SSV3ERROR::SSV3ERROR_SEQUENCE_LOAD_FAILED;
    return ret;
}

SSV3ERROR AddExperimentStep(const unsigned char sequence, const unsigned
char excitation, const int step = -1)
{
    //If the default -1 or some other negative value was passed
    // set the step number to one greater than the last valid index
    int stepNum = step;
    if (stepNum < 0)
        stepNum = (int)_experimentList.size();

    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;

    if (sequence > 31 || sequence > _angleSequences.size())
        return SSV3ERROR::SSV3ERROR_SEQUENCE_DOESNT_EXIST;
    else if (_angleSequences[sequence].empty())
        return SSV3ERROR::SSV3ERROR_SEQUENCE_DOESNT_EXIST;
    if (excitation > 31 || excitation > _illuminationProfiles.size())
        return SSV3ERROR::SSV3ERROR_EXCITATION_PROFILE_DOESNT_EXIST;
    else if (_illuminationProfiles[excitation].empty())
        return SSV3ERROR::SSV3ERROR_EXCITATION_PROFILE_DOESNT_EXIST;

    if (stepNum > _experimentList.size())
        return SSV3ERROR::SSV3ERROR_STEP_OUTSIDE_EXPERIMENT_RANGE;

    Node newNode;
    newNode.number = stepNum;
    newNode._sequence = sequence;
    newNode._exSetting = excitation;

    if (stepNum == _experimentList.size())
        _experimentList.emplace_back(newNode);
    else
        _experimentList.emplace(_experimentList.begin() + step, newNode);
    _experimentModified = true;
    return ret;
}

SSV3ERROR ClearExperiment()
{
    SSV3ERROR ret{ OK_ };
}

```

```

    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;
    _experimentList.clear();
    _loopOnOff = false;
    _loopTo = 0;
    _experimentModified = true;
    return ret;
}

SSV3ERROR Loop(bool onOff, const unsigned int loopTo = 0)
{
    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;

    if (_experimentList.size() == 0)
        return SSV3ERROR::SSV3ERROR_NO_EXPERIMENT;
    if (loopTo < 0 || loopTo >= _experimentList.size())
        return SSV3ERROR::SSV3ERROR_INVALID_LOOP;

    _loopOnOff = onOff;
    _loopTo = (unsigned char)loopTo;
    _experimentModified = true;
    return ret;
}

SSV3ERROR StartExperiment()
{
    SSV3ERROR ret{ OK_ };
    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;
    if (_experimentList.size() == 0)
        return SSV3ERROR::SSV3ERROR_NO_EXPERIMENT;

    //This ensures that the current experiment design is programmed
    if (_experimentModified)
    {
        ret = ResendExperiment();
    }

    unsigned char *msg = _oBuffer;
    *msg++ = 0x87;
    *msg++ = 0x01;
    *msg++ = _defaultExperiment;
    *msg++ = 0x00;
    *msg++ = 0x00;
    *msg++ = _loopOnOff;
    *msg++ = 0x00;
}

```

```

    *msg++ = 0x00;
    SendAndListen(&ret, 0, true);
    if (ret != OK_)
        return ret;
    if (_iBuffer[0] == 1)
        return SSV3ERROR::SSV3ERROR_NO_EXPERIMENT;
    if (_iBuffer[0] != 0)
        return SSV3ERROR::SSV3ERROR_UNEXPECTED_RETURN;
    _experimentRunning = true;
    return ret;
}

SSV3ERROR StopExperiment()
{
    SSV3ERROR ret{ OK_ };
    if (!_experimentRunning)
        return ret;
    _oBuffer[0] = 0x8F;
    SendAndListen(&ret, 1, true);
    if (ret != OK_)
        return ret;
    _experimentRunning = false;
    for (unsigned char i = 0; i < 8; i++)
    {
        SingleLaserPower(i, _currentExcitation[i]);
    }
    SetRadius(_currentRadius);
    Fire(true);
    return ret;
}

SSV3ERROR SendSWTrigger(unsigned short period = 0xffff)
{
    SSV3ERROR ret{ OK_ };
    unsigned short resetVal = 0xffff - period;
    unsigned char* msg = _oBuffer;
    *msg++ = 0x5f;
    *msg++ = HByte(resetVal);
    *msg++ = LByte(resetVal);
    SendAndListen(&ret, 0, true);
    return ret;
}

SSV3ERROR SendArray(unsigned char* msg, const size_t length = 0)
{
    SSV3ERROR ret{ OK_ };
    for (size_t i = 0; i < length; i++)
        _oBuffer[i] = msg[i];
    SendAndListen(&ret, 0, true);
    if (ret != OK_)
        return ret;
    for (size_t i = 0; i < length; i++)
        msg[i] = _iBuffer[i];
    return ret;
}

```

```

SSV3ERROR QueryInternalSettings(
    unsigned short *xCenter,
    unsigned short *yCenter,
    unsigned short *tirRadius,
    unsigned short *phase,
    unsigned short *frequency)
{
    SSV3ERROR ret{ OK_ };
    _oBuffer[0] = 0xF1;
    SendAndListen(&ret, 1, true);
    if (ret != OK_)
        return ret;
    *xCenter = _iBuffer[1];
    *xCenter = (*xCenter << 8) | (_iBuffer[2] & 0xff);
    *yCenter = _iBuffer[3];
    *yCenter = (*yCenter << 8) | (_iBuffer[4] & 0xff);
    *tirRadius = _iBuffer[5];
    *tirRadius = (*tirRadius << 8) | (_iBuffer[6] & 0xff);
    *phase = _iBuffer[7];
    *phase = (*phase << 8) | (_iBuffer[8] & 0xff);
    *frequency = _iBuffer[9];
    *frequency = (*frequency << 8) | (_iBuffer[10] & 0xff);
    return ret;
}

SSV3ERROR QueryDevVer(unsigned char *brdMajor, unsigned char *brdMinor,
unsigned char *fwMajor, unsigned char *fwMinor)
{
    SSV3ERROR ret{ OK_ };
    _oBuffer[0] = 0xF2;
    SendAndListen(&ret, 1, true);
    if (ret != OK_)
        return ret;
    *brdMajor = _iBuffer[1];
    *brdMinor = _iBuffer[2];
    *fwMajor = _iBuffer[3];
    *fwMinor = _iBuffer[4];
    return ret;
}

SSV3ERROR DetailedErrorReporting(bool onOff)
{
    _detailedReporting = onOff;
    return OK_;
}

const char * GetLastError()
{
    if (_newErr)
    {
        if (_errStr != nullptr)
            delete[] _errStr;
        _errStr = new char[_errMsg.str().size()];
        strcpy(_errStr, _errMsg.str().c_str());
        _newErr = false;
        return _errStr;
    }
}

```

```

    }
    else
        return NULL;
}

```

```

void Reset()
{
    _oBuffer[0] = 0xff;
    hid_write(_dev, _xmit, 65);
}

```

private:

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Member variables                                                                                               */
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

unsigned char _xmit[65]{ 0 };
unsigned char *_oBuffer{ &_xmit[1] };
unsigned char _iBuffer[64]{ 0 };
unsigned int _timeout{ 2000 };
unsigned char _usingExcitation{0xff};
hid_device *_dev;
float _yOffset{ 1 };
unsigned short _currentRadius{ 0 };
unsigned short _scanCenter[2]{ 0x7fff, 0x7fff };
unsigned short _tirRadius{ 0x2d00 };
unsigned int _readAttempts{ 2 };
unsigned char _defaultExperiment{ 16 };
int _loopTo{ 0 };
bool _loopOnOff{ false };
bool _experimentRunning{ false };
std::vector<unsigned short> _currentExcitation;
std::vector<std::vector<unsigned short>> _angleSequences;
std::vector<std::vector<unsigned short>> _illuminationProfiles;
std::vector<Node> _experimentList;
bool _experimentModified{ false };
bool _demo{ false };
bool _detailedReporting{ false };
std::stringstream _errMsg;
bool _newErr{ false };
char *_errStr{ nullptr };

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Internal Functions                                                                                           */
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

//Send the contents of the output buffer and wait for a response
void Transmit(SSV3ERROR *err, int nCheck, bool retry, const char *fun)
{
    if (_demo)
    {
        *err = OK_;
        return;
    }
    //Ensure that the report sent is 0x00
    _xmit[0] = 0x00;
}

```

```

*err = OK_;
int ret = 0;
bool success{ false };
if (!retry)
    success = true;

int nRet = hid_write(_dev, _xmit, 65);
if (nRet < 65)
{
    if (_detailedReporting)
    {
        _newErr = true;
        _errMsg.str("");
        _errMsg << "Error in function ";
        _errMsg << fun;
        if (nRet == -1)
        {
            _errMsg << " - Write failed.\n";
        }
        else
        {
            _errMsg << " - Wrote " << nRet << " of 65 bytes.\n";
        }
        _errMsg << "Attempted to write: ";
        for (int i = 0; i < 64; i++)
            _errMsg << (int)_oBuffer[i] << " ";
    }
    *err = SSV3ERROR::SSV3ERROR_XMIT_FAIL;
    return;
}

//Read the received buffer
unsigned int rAttempts = 1;
do {
    ret = hid_read_timeout(_dev, _iBuffer, 64, _timeout);
    //When we get the response break both loops
    if (ret == 64)
    {
        success = true;
        break;
    }
    //Otherwise, keep trying to read
    rAttempts++;
    if (rAttempts > _readAttempts)
        break;
} while (!success);

//If there was no response
if (!success)
{
    if (_detailedReporting)
    {
        _newErr = true;
        _errMsg.str("");
        _errMsg << "No reponse from device in function " << fun << "\n";
    }
}

```

```

    }
    *err = SSV3ERROR::SSV3ERROR_NO_RESPONSE;
}

//If the response is truncated
else if (ret < 64)
{
    if (_detailedReporting)
    {
        _newErr = true;
        _errMsg.str("");
        _errMsg << "Error in function " << fun << " received " << ret <<
" of 64 bytes.\n";
    }
    *err = SSV3ERROR::SSV3ERROR_READ_TOO_SHORT;
}

//If a response check was specified check the number of bytes
indicated
else if (nCheck > 0)
{
    bool mismatch{ false };
    for (int i = 0; i < nCheck; i++)
        if (_iBuffer[i] != _oBuffer[i])
            mismatch = true;
    if (mismatch && _detailedReporting)
    {
        _newErr = true;
        _errMsg.str("");
        _errMsg << "Error in function " << fun << " response did not
match expected response.\n";
        *err = SSV3ERROR::SSV3ERROR_UNEXPECTED_RETURN;
    }
}
if (*err != OK_)
{
    _errMsg << "Wrote :";
    for (int i = 0; i < 64; i++)
        _errMsg << (int)_oBuffer[i] << " ";
    _errMsg << "\nReceived :";
    for (int i = 0; i < 64; i++)
        _errMsg << (int)_iBuffer[i] << " ";
    return;
}
return;
}

unsigned char HByte(unsigned short x) { return (unsigned char)(x >> 8); }
unsigned char LByte(unsigned short x) { return (unsigned char)x; }

//Set a new scan radius
SSV3ERROR SetRadius(unsigned short value)
{
    SSV3ERROR ret{ OK_ };
    unsigned short yVal = (unsigned short)((float)value * _yOffset);
    yVal = yVal > 0x8000 ? 0x8000 : yVal;
}

```

```

    unsigned char *msg = _oBuffer;
    *msg++ = 0x10;
    *msg++ = HByte(value);
    *msg++ = LByte(value);
    *msg++ = HByte(yVal);
    *msg++ = LByte(yVal);
    SendAndListen(&ret, 5, true);
    return ret;
}

//Reprograms the current experiment
//This is necessary after any call that changes the excitation profiles
// or angle sequences
SSV3ERROR ResendExperiment()
{
    SSV3ERROR ret{ OK_ };
    if (_experimentList.size() == 0)
        return ret;

    if (_experimentRunning)
        ret = StopExperiment();
    if (ret != OK_)
        return ret;

    //Basically we just reattach all the nodes
    //Otherwise the controller may have nonexistent pointers to the data
    unsigned char *msg = _oBuffer;
    *msg++ = 0x83; //Add a node
    *msg++ = _defaultExperiment;
    *msg++ = _experimentList[0]._sequence;
    *msg++ = _experimentList[0]._exSetting;
    SendAndListen(&ret, 0, false);
    if (ret != OK_)
        return ret;
    if (_iBuffer[0] == 1)
        return SSV3ERROR::SSV3ERROR_ALLOC_FAIL;
    if (_iBuffer[0] == 3)
        return SSV3ERROR::SSV3ERROR_SEQUENCE_DOESNT_EXIST;
    if (_iBuffer[0] == 4)
        return SSV3ERROR::SSV3ERROR_EXCITATION_PROFILE_DOESNT_EXIST;

    //Now build the rest of the experiment
    for (std::vector<Node>::iterator itr(_experimentList.begin() + 1); itr
!= _experimentList.end(); itr++)
    {
        msg = _oBuffer;
        *msg++ = 0x85;
        *msg++ = _defaultExperiment;
        *msg++ = itr->_sequence;
        *msg = itr->_exSetting;
        SendAndListen(&ret, 0, false);
        if (ret != OK_)
            return ret;
        if (_iBuffer[0] == 1)
            return SSV3ERROR::SSV3ERROR_ALLOC_FAIL;

```

```

        if (_iBuffer[0] == 3)
            return SSV3ERROR::SSV3ERROR_SEQUENCE_DOESNT_EXIST;
        if (_iBuffer[0] == 4)
            return SSV3ERROR::SSV3ERROR_EXCITATION_PROFILE_DOESNT_EXIST;
    }

    //Rebuild the loop
    if (_loopOnOff)
    {
        if (_loopTo < _experimentList.size())
        {
            msg = _oBuffer;
            *msg++ = 0x86;
            *msg++ = _defaultExperiment;
            *msg++ = _loopTo;
            SendAndListen(&ret, 0, true);
            if (ret != OK_)
                return ret;
            if (_iBuffer[0] == 1)
                ret = SSV3ERROR::SSV3ERROR_NO_EXPERIMENT;
            if (_iBuffer[0] == 2)
                ret = SSV3ERROR::SSV3ERROR_INVALID_LOOP;
        }
        else
            ret = SSV3ERROR::SSV3ERROR_INVALID_LOOP;
    }
    _experimentModified = false;
    return ret;
}
};

#ifdef __cplusplus
extern "C" {
#endif
SSV3API_ SSV3Controller __cdecl CreateDevice()
{
    SSV3Controller p = new ScanCard;
    return p;
}

SSV3API_ SSV3Controller __cdecl CreateDeviceFromSN(wchar_t *sn)
{
    SSV3Controller p = new ScanCard(sn);
    return p;
}

SSV3API_ SSV3Controller __cdecl CreateDemoDevice(bool demo)
{
    SSV3Controller p = new ScanCard(demo);
    return p;
}
#ifdef __cplusplus
}
#endif
}

```

## Appendix E

### SAIM analysis source code

The following is the source code for the custom SAIM analysis software. The code is written in C/C++ under the C++17 standard in Microsoft Visual Studio and compiled with Intel Parallel Studio XE 2018. It makes use of the Boost and OpenCV libraries for file I/O and image manipulation. The code compiles to a Microsoft Windows executable with a command line interface. For optimization parameters are entered into the file `analysis_testbed.h` and the program is compiled on the target computer prior to execution. Files to be analyzed are passed as command line arguments for batch processing.

**File: saim\_model\_cpu.h**

```
/**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
/  
//  
//  
//  
//  
// Copyright(c) 2018, Marshall Colville mjc449@cornell.edu  
//  
// All rights reserved.  
//  
//  
//  
// Redistribution and use in source and binary forms, with or without  
// modification, are permitted provided that the following conditions are  
// met :  
//  
//  
//  
// 1. Redistributions of source code must retain the above copyright notice,  
// this list of conditions and the following disclaimer.  
//  
// 2. Redistributions in binary form must reproduce the above copyright  
// notice, this list of conditions and the following disclaimer in the  
// documentation and/or other materials provided with the distribution.  
//  
//  
//  
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED  
// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A  
// PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER  
// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,  
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,  
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR  
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF  
// LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING  
// NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS  
// SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  
//
```



```

FitTask(CPUModel *, int, int, int);
~FitTask();
volatile void operator()(const tbb::blocked_range<int> &index) const;
void operator()(int);

extern friend void objective(MKL_INT *n, MKL_INT *m, double *, double
*, void *);

private:
CPUModel *l_parent;
int l_count;
double *l_xvec, *l_fvec, *l_jvec;
MKL_INT l_nVars{ 3 };
MKL_INT l_nPoints{ 0 };
MKL_INT l_iterations{ 1000 };
MKL_INT l_stepIterations{ 100 };
double l_initialStep{ 00.0 };
MKL_INT l_rciRequest{ 0 };
MKL_INT l_successful{ 0 };
MKL_INT l_actualIterations{ 0 };
MKL_INT l_stopCrit{ 0 };
double l_initialRes{ 0 }, l_finalRes{ 0 };
MKL_INT l_counter{ 0 };
MKL_INT l_fitInfo[6];
double l_eps[6] = { 0.000000001, 0.000000001, 0.000000001,
0.000000001, 0.000000001, 0.000000001 };
double l_jeps{ 0.000000001 };
};

/*****
 * @brief Calculates the function value at the current xvec
*****/
int CalculateFunction(int, double *, double *);

/*****
 * @brief Calculates the Jacobian value at the current xvec
*****/
int CalculateJacobian(int, double *, double *);

private:
std::vector<cv::Mat> _rawImgs, _outputImgs;
volatile int _n;
volatile int _m;
int _grainSize, _nGrains, _emptyPixels;
bool _initialized;
size_t _datasz, _fnasz, _xsz, _jacs;
unsigned short *_data;
double *_constvec;
double _guesses[3]{ 0.8, 1.0, 6.0 };
};
}
#endif //SAIM_MODEL_CPU_H

```

**File: saim\_model\_cpu.cpp**

```
/**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
/  
//  
//  
//  
// Copyright(c) 2018, Marshall Colville mjc449@cornell.edu  
//  
// All rights reserved.  
//  
//  
// Redistribution and use in source and binary forms, with or without  
// modification, are permitted provided that the following conditions are  
// met :  
//  
//  
// 1. Redistributions of source code must retain the above copyright notice,  
// this list of conditions and the following disclaimer.  
//  
// 2. Redistributions in binary form must reproduce the above copyright  
// notice, this list of conditions and the following disclaimer in the  
// documentation and/or other materials provided with the distribution.  
//  
//  
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED  
// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A  
// PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER  
// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,  
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,  
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR  
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF  
// LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING  
// NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS  
// SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  
//
```

```

//
//
// The views and conclusions contained in the software and documentation are
//
// those of the authors and should not be interpreted as representing
//
// official policies, either expressed or implied, of the SAIMScannerV3
//
// project, the Paszek Research Group, or Cornell University.
//
//
//*****
/

#include "saim_model_cpu.h"

#include <chrono>
#include <stdio.h>
#include <iostream>

#include <opencv2/core/core.hpp>

namespace cpu_model
{
    CPUModel::CPUModel() {};
    CPUModel::~CPUModel() {};

    int CPUModel::RegisterImages(std::vector<cv::Mat> &imStack)
    {
        _rawImgs = imStack;
        _outputImgs.clear();
        for (int i = 0; i < 7; i++)
        {
            _outputImgs.push_back(cv::Mat(_rawImgs[0].rows, _rawImgs[0].cols,
CV_32F));
        }
        return 0;
    }

    int CPUModel::SetGrainSize(int grain)
    {
        _grainSize = grain;
        return 0;
    }

    int CPUModel::InitializeBuffers()
    {
        _m = _rawImgs[0].rows * _rawImgs[0].cols;
        _n = _rawImgs.size();
        _nGrains = _m % _grainSize == 0 ? _m / _grainSize : _m / _grainSize + 1;
        _emptyPixels = _m % _grainSize == 0 ? 0 : _grainSize - (_m % _grainSize);
        _datasz = _m * _n;
        _fnsz = _n;
        _jacsiz = _n * 3;
        _data = (unsigned short *)MKL_malloc(_datasz * sizeof(unsigned short),
64);
        if (_data == nullptr)

```

```

    {
        _initialized = false;
        return 1;
    }
    _constvec = (double *)MKL_malloc(_n * 3 * sizeof(double), 64);
    if (_constvec == nullptr)
    {
        mkl_free(_data);
        _initialized = false;
        return 1;
    }
    for (int i = 0; i < _raw_imgs.size(); i++)
    {
        unsigned short *ptr = _raw_imgs[i].ptr<unsigned short>();
        for (int j = 0; j < _m; j++)
        {
            _data[j * _n + i] = *ptr++;
        }
    }
    _initialized = true;
    return 0;
}

int CPUModel::ReleaseBuffers(void)
{
    if (_data != nullptr)
    {
        mkl_free(_data);
        _data = nullptr;
    }
    if (_constvec != nullptr)
    {
        mkl_free(_constvec);
        _constvec = nullptr;
    }
    _initialized = false;
    return 0;
}

int CPUModel::CalculateConstants(double wavelength, double d0x, double nB,
double n0x, double nSi, double *angles)
{
    double angle0x, angleSi, pB, p0x, pSi, k0x, m11, m12, m21, m22;
    std::complex<double> num, denom, rTE;
    for (int i = 0; i < _n; i++)
    {
        angle0x = asin(sin(angles[i]) * nB / n0x);
        angleSi = asin(sin(angle0x) * n0x / nSi);
        k0x = 2.0 * CV_PI * n0x / wavelength;
        pB = nB * cos(angles[i]);
        p0x = n0x * cos(angle0x);
        pSi = nSi * cos(angleSi);
        m11 = cos(k0x * d0x * cos(angle0x));
        m12 = -1.0 / p0x * sin(k0x * d0x * cos(angle0x));
        m21 = -p0x * sin(k0x * d0x * cos(angle0x));
        m22 = m11;
    }
}

```

```

        num = std::complex<double>(m11 * pB - m22 * pSi, m12 * pSi * pB -
m21);
        denom = std::complex<double>(m11 * pB + m22 * pSi, m12 * pSi * pB +
m21);
        rTE = num / denom;
        _constvec[i * 3] = rTE.real();
        _constvec[i * 3 + 1] = rTE.imag();
        _constvec[i * 3 + 2] = 4 * CV_PI * nB * cos(angles[i]) / wavelength;
    }
    return 0;
}

int CPUModel::RunFit(void)
{
    FitTask task(this, _n, 0, 1);
    for (int i = 0; i < _m; i++)
        task(i);
    return 0;
}

int CPUModel::ParforRunFit(void)
{
    std::chrono::high_resolution_clock::time_point earlier, later;
    std::chrono::duration<double> timeTaken;
    earlier = std::chrono::high_resolution_clock::now();
    FitTask task(this, _n, 0, 1);
    tbb::parallel_for(tbb::blocked_range<int>(0, _m), FitTask(this, _n, 0,
1));
    later = std::chrono::high_resolution_clock::now();
    timeTaken = later - earlier;
    std::cout << "Took " <<
std::chrono::duration_cast<std::chrono::milliseconds>(timeTaken).count() <<
std::endl;
    return 0;
}

int CPUModel::ThreadedRunFit(void)
{
    return 0;
}

int CPUModel::CalculateFunction(int pixel, double *xvec, double *fvec)
{
    double A{ xvec[0] }, B{ xvec[1] }, H{ xvec[2] };
    //double *dataVec = new double[_n];
    //double *funVec = new double[_n];
    for (int i = 0; i < _n; i++)
    {
        double c{ _constvec[3 * i] }, d{ _constvec[3 * i + 1] }, phi{
        _constvec[3 * i + 2] };
        double value = A * (1.0 + 2.0 * c * cos(phi * H) - 2.0 * d * sin(phi *
H) + c * c + d * d) + B;
        fvec[i] = (double)_data[_n * pixel + i] - value;
        //dataVec[i] = (double)_data[_n * pixel + i];
        //funVec[i] = value;
    }
}

```

```

        //delete[] dataVec;
        //delete[] funVec;
        return 0;
    }

    int CPUModel::CalculateJacobian(int pixel, double *xvec, double *jvec)
    {
        double A{ xvec[0] }, H{ xvec[2] };
        for (int i = 0; i < _n; i++)
        {
            double c{ _constvec[3 * i] }, d{ _constvec[3 * i + 1] }, phi{
            _constvec[3 * i + 2] };
            jvec[i] = -1.0 * (1.0 + 2.0 * c * cos(phi * H) - 2.0 * d * sin(phi *
            H) + c * c + d * d);
            jvec[i + _n] = -1.0;
            jvec[i + 2 * _n] = 2.0 * A * phi * (c * sin(phi * H) + d * cos(phi *
            H));
        }

        return 0;
    }

    std::vector<cv::Mat> CPUModel::GetImages(void)
    {
        return _outputImgs;
    }

    CPUModel::FitTask::FitTask(CPUModel *parent, int frames, int startIdx, int
    count) : l_parent(parent), l_nPoints(frames), l_count(count) {}

    CPUModel::FitTask::~FitTask() {}

    void CPUModel::FitTask::operator()(int index)
    {
        l_xvec = (double *)mkl_malloc(3 * sizeof(double), 64);
        if (l_xvec == nullptr)
            return;
        l_fvec = (double *)mkl_malloc(l_parent->fnsz * sizeof(double), 64);
        if (l_fvec == nullptr)
        {
            mkl_free(l_xvec);
            return;
        }
        l_jvec = (double *)mkl_malloc(l_parent->jacsz * sizeof(double), 64);
        if (l_jvec == nullptr)
        {
            mkl_free(l_xvec);
            mkl_free(l_fvec);
            return;
        }
        std::chrono::high_resolution_clock::time_point earlier, later;
        std::chrono::duration<double> timeTaken;
        for (int i = 0; i < l_count; i++)
        {
            earlier = std::chrono::high_resolution_clock::now();

```

```

l_successful = 0;
l_xvec[0] = l_parent->guesses[0];
l_xvec[1] = l_parent->guesses[1];
l_xvec[2] = l_parent->guesses[2];
int pixel = index + i;

    _TRNSP_HANDLE_t solverHandle;

    if (dtrnlsp_init(&solverHandle, &l_nVars, &l_nPoints, l_xvec, l_eps,
&l_iterations, &l_stepIterations, &l_initialStep) != TR_SUCCESS)
    {
        std::cerr << "Error initializing solver" << std::endl;
        MKL_Thread_Free_Buffers();
        return;
    }
    if (dtrnlsp_check(&solverHandle, &l_nVars, &l_nPoints, l_jvec, l_fvec,
l_eps, l_fitInfo) != TR_SUCCESS)
    {
        std::cerr << "Error checking solver" << std::endl;
        MKL_Thread_Free_Buffers();
        return;
    }
    else
    {
        if (l_fitInfo[0] != 0 ||
            l_fitInfo[1] != 0 ||
            l_fitInfo[2] != 0 ||
            l_fitInfo[3] != 0)
        {
            std::cerr << "Invalid array passed to solver: " << std::endl;
            MKL_Thread_Free_Buffers();
            return;
        }
    }
l_successful = 0;
l_counter = 0;
while (l_successful == 0)
{
    if (dtrnlsp_solve(&solverHandle, l_fvec, l_jvec, &l_rciRequest) !=
TR_SUCCESS)
    {
        std::cerr << "Error solving solver" << std::endl;
        MKL_Thread_Free_Buffers();
        return;
    }
    if (l_rciRequest == -1 ||
        l_rciRequest == -2 ||
        l_rciRequest == -3 ||
        l_rciRequest == -4 ||
        l_rciRequest == -5 ||
        l_rciRequest == -6)
        l_successful = 1;
    if (l_rciRequest == 1)
    {
        l_parent->CalculateFunction(pixel, l_xvec, l_fvec);
    }
}

```

```

        if (l_rciRequest == 2)
            l_parent->CalculateJacobian(pixel, l_xvec, l_jvec);
        //std::cout << "RCI cycle: " << _counter++ << std::endl;
        l_counter++;
    }
    if (dtrnlsp_get(&solverHandle, &l_actualIterations, &l_stopCrit,
&l_initialRes, &l_finalRes) != TR_SUCCESS)
    {
        std::cerr << "Error getting solver results" << std::endl;
        MKL_Thread_Free_Buffers();
        return;
    }
    if (dtrnlsp_delete(&solverHandle) != TR_SUCCESS)
    {
        std::cerr << "Error deleting the solver" << std::endl;
        MKL_Thread_Free_Buffers();
        return;
    }

    *(l_parent->_outputImgs[0].ptr<double>() + pixel) = l_xvec[0];
    *(l_parent->_outputImgs[1].ptr<double>() + pixel) = l_xvec[1];
    *(l_parent->_outputImgs[2].ptr<double>() + pixel) = l_xvec[2];
    *(l_parent->_outputImgs[3].ptr<double>() + pixel) = l_stopCrit;
    *(l_parent->_outputImgs[4].ptr<double>() + pixel) = l_finalRes;

    later = std::chrono::high_resolution_clock::now();
    timeTaken = later - earlier;
    std::cout << "Pixel " << pixel << " finished " << l_counter << "
cycles in " <<
std::chrono::duration_cast<std::chrono::microseconds>(timeTaken).count() << "
microseconds." << std::endl;
    }
    mkl_free(l_xvec);
    mkl_free(l_fvec);
    mkl_free(l_jvec);
    MKL_Thread_Free_Buffers();
    l_xvec = l_fvec = l_jvec = nullptr;
}

volatile void CPUModel::FitTask::operator()(const tbb::blocked_range<int>
&index) const
{
    double *xvec = (double *)mkl_malloc(3 * sizeof(double), 64);
    if (l_xvec == nullptr)
        return;
    double *fvec = (double *)mkl_malloc(l_parent->_fnsz * sizeof(double),
64);
    if (l_fvec == nullptr)
    {
        mkl_free(xvec);
        return;
    }
    double *jvec = (double *)mkl_malloc(l_parent->_jacsz * sizeof(double),
64);
    if (l_jvec == nullptr)
    {

```

```

        mkl_free(xvec);
        mkl_free(fvec);
        return;
    }
    int fitInfo[6]{ 0, 0, 0, 0, 0, 0 };
    std::chrono::high_resolution_clock::time_point earlier, later;
    std::chrono::duration<double> timeTaken;
    for (size_t i = index.begin(); i != index.end(); i++)
    {
        if (l_parent->_data[i * l_nPoints] == 0)
            continue;
        earlier = std::chrono::high_resolution_clock::now();
        double maxval = (double)l_parent->_data[i * l_nPoints];
        double minval{ maxval };
        for (int j = 0; j < l_nPoints; j++)
        {
            fvec[j] = 0;
            jvec[j] = 0;
            jvec[j + l_nPoints] = 0;
            jvec[j + l_nPoints * 2] = 0;
            double thisval = (double)l_parent->_data[i * l_nPoints + j];
            maxval = maxval < thisval ? thisval : maxval;
            minval = minval > thisval ? thisval : minval;
        }

        int successful = 0;
        xvec[0] = l_parent->_guesses[0] * (maxval - minval);
        xvec[1] = l_parent->_guesses[1] * minval;
        xvec[2] = l_parent->_guesses[2];
        int pixel = i;

        _TRNSP_HANDLE_t solverHandle;

        if (dtrnlsp_init(&solverHandle, &l_nVars, &l_nPoints, xvec, l_eps,
            &l_iterations, &l_stepIterations, &l_initialStep) != TR_SUCCESS)
        {
            std::cerr << "Error initializing solver" << std::endl;
            MKL_Thread_Free_Buffers();
            return;
        }
        if (dtrnlsp_check(&solverHandle, &l_nVars, &l_nPoints, jvec, fvec,
            l_eps, fitInfo) != TR_SUCCESS)
        {
            std::cerr << "Error checking solver" << std::endl;
            MKL_Thread_Free_Buffers();
            return;
        }
        else
        {
            if (fitInfo[0] != 0 ||
                fitInfo[1] != 0 ||
                fitInfo[2] != 0 ||
                fitInfo[3] != 0)
            {
                std::cerr << "Invalid array passed to solver: " << fitInfo[0] <<
                    fitInfo[1] << fitInfo[2] << fitInfo[3] << std::endl;
            }
        }
    }
}

```

```

        MKL_Thread_Free_Buffers();
        return;
    }
}
successful = 0;
int counter = 0;
int rciRequest = 0;
while (successful == 0)
{
    if (dtrnlsp_solve(&solverHandle, fvec, jvec, &rciRequest) !=
TR_SUCCESS)
    {
        std::cerr << "Error solving solver" << std::endl;
        MKL_Thread_Free_Buffers();
        return;
    }
    if (rciRequest == -1 ||
        rciRequest == -2 ||
        rciRequest == -3 ||
        rciRequest == -4 ||
        rciRequest == -5 ||
        rciRequest == -6)
        successful = 1;
    if (rciRequest == 1)
        l_parent->CalculateFunction(pixel, xvec, fvec);
    if (rciRequest == 2)
        l_parent->CalculateJacobian(pixel, xvec, jvec);
    //std::cout << "RCI cycle: " << _counter++ << std::endl;
}
MKL_INT actualIterations{ 0 };
MKL_INT stopCrit{ 0 };
double initialRes{ 0 }, finalRes{ 0 };
if (dtrnlsp_get(&solverHandle, &actualIterations, &stopCrit,
&initialRes, &finalRes) != TR_SUCCESS)
{
    std::cerr << "Error getting solver results" << std::endl;
    MKL_Thread_Free_Buffers();
    return;
}
if (dtrnlsp_delete(&solverHandle) != TR_SUCCESS)
{
    std::cerr << "Error deleting the solver" << std::endl;
    MKL_Thread_Free_Buffers();
    return;
}

double res{ 0.0 }, avg{ 0.0 }, sst{ 0.0 }, ssr{ 0.0 }, ssc{ 0.0 };

for (int j = 0; j < l_nPoints; j++)
{
    avg += l_parent->_data[pixel * l_nPoints + j];
    ssr += fvec[j] * fvec[j];
}
avg /= l_nPoints;
for (int j = 0; j < l_nPoints; j++)
{

```

```

        double dataval = l_parent->_data[pixel * l_nPoints + j];
        dataval -= avg;
        sst += dataval * dataval;
    }
    for (int j = 1; j < l_nPoints; j++)
    {
        double scval = fvec[j] - fvec[j - 1];
        ssc += scval * scval;
    }

    double rmsNoise{ 0.0 }, rmsSignal{ 0.0 }, snr{ 0.0 };
    double *prediction = new double[l_nPoints];

    for (int j = 0; j < l_nPoints; j++)
    {
        double c{ l_parent->_constvec[3 * j] }, d{ l_parent->_constvec[3 *
j + 1] }, phi{ l_parent->_constvec[3 * j + 2] };
        prediction[j] = xvec[0] * (1.0 + 2.0 * c * cos(phi * xvec[2]) - 2.0
* d * sin(phi * xvec[2]) + c * c + d * d);
        rmsSignal += prediction[j] * prediction[j];
    }

    double d, r;
    d = ssc / ssr;
    r = 1 - ssr / sst;
    rmsNoise = sqrt(ssr / l_nPoints);
    rmsSignal = sqrt(rmsSignal / l_nPoints);
    snr = (rmsSignal * rmsSignal) / (rmsNoise * rmsNoise);

    *(l_parent->_outputImgs[0].ptr<float>() + pixel) = (float)xvec[0];
    *(l_parent->_outputImgs[1].ptr<float>() + pixel) = (float)xvec[1];
    *(l_parent->_outputImgs[2].ptr<float>() + pixel) = (float)xvec[2];
    *(l_parent->_outputImgs[3].ptr<float>() + pixel) = (float)stopCrit;
    *(l_parent->_outputImgs[4].ptr<float>() + pixel) = (float)r;
    *(l_parent->_outputImgs[5].ptr<float>() + pixel) = (float)d;
    *(l_parent->_outputImgs[6].ptr<float>() + pixel) = (float)snr;

    later = std::chrono::high_resolution_clock::now();
    timeTaken = later - earlier;
    delete[] prediction;
    //std::cout << "Pixel " << pixel << " finished in " <<
std::chrono::duration_cast<std::chrono::microseconds>(timeTaken).count() << "
microseconds." << std::endl;
    }
    mkl_free(xvec);
    mkl_free(fvec);
    mkl_free(jvec);
    MKL_Thread_Free_Buffers();
    xvec = fvec = jvec = nullptr;
}

void objective(MKL_INT *pixel, MKL_INT *m, double *x, double *f, void
*instance)
{
    CPUModel::FitTask *task = (CPUModel::FitTask *)instance;
    task->l_parent->CalculateFunction(*pixel, task->l_xvec, task->l_fvec);
}

```

```
        f = task->l_fvec;  
    }  
}
```

File: analysis\_testbed.cpp

```
/**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
/  
//  
//  
//  
// Copyright(c) 2018, Marshall Colville mjc449@cornell.edu  
//  
// All rights reserved.  
//  
//  
//  
// Redistribution and use in source and binary forms, with or without  
// modification, are permitted provided that the following conditions are  
// met :  
//  
//  
//  
// 1. Redistributions of source code must retain the above copyright notice,  
// this list of conditions and the following disclaimer.  
//  
// 2. Redistributions in binary form must reproduce the above copyright  
// notice, this list of conditions and the following disclaimer in the  
// documentation and/or other materials provided with the distribution.  
//  
//  
//  
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED  
// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A  
// PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER  
// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,  
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,  
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR  
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF  
// LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING  
// NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS  
// SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  
//
```



```

    double linangles[31] = { -39.911025,
-37.25029,
-34.589555,
-31.92882,
-29.268085,
-26.60735,
-23.946615,
-21.28588,
-18.625145,
-15.96441,
-13.303675,
-10.64294,
-7.982205,
-5.32147,
-2.660735,
0.0,
2.660735,
5.32147,
7.982205,
10.64294,
13.303675,
15.96441,
18.625145,
21.28588,
23.946615,
26.60735,
29.268085,
31.92882,
34.589555,
37.25029,
39.911025,
};

```

```

double angles[31] =
{ -43.81494106,
- 39.98737856,
- 36.28932555,
- 32.72078203,
- 29.281748,
- 25.97222346,
- 22.79220841,
- 19.74170285,
- 16.82070678,
- 14.02922019,
- 11.3672431,
- 8.834775503,
- 6.431817392,
- 4.158368772,
- 2.014429641,
0.0,
2.014429641,
4.158368772,
6.431817392,
8.834775503,
11.3672431,
14.02922019,

```

```

16.82070678,
19.74170285,
22.79220841,
25.97222346,
29.281748,
32.72078203,
36.28932555,
39.98737856,
43.81494106 };

for (int i = 0; i < 31; i++)
    linangles[i] = linangles[i] * 2.0 * CV_PI / 360.0;

/*
>>>>>> 3519e08061486c7d7904e57dd88568bdbaa23dfd
double start = firstAngle * 2.0 * CV_PI / 360.0;
for (int i = 0; i < samples; i++)
    angles[i] = start + i * step * 2.0 * CV_PI / 360.0;
*/

cpu_model::CPUModel model;

model.RegisterImages(imstack);
model.SetGrainSize(1);
model.InitializeBuffers();
model.CalculateConstants(560.0, 1910.5, 1.34, 1.463, 4.3638, linangles);
model.ParforRunFit();
//model.RunFit();
std::vector<cv::Mat> outputs = model.GetImages();
model.ReleaseBuffers();

//Fit A
outputs[0].convertTo(outputs[0], CV_16UC1);
//Fit B
outputs[1].convertTo(outputs[1], CV_16UC1);
//Multiply H by 100 to get 2 decimals
outputs[2] *= 100;
outputs[2].convertTo(outputs[2], CV_16UC1);
//Fit stop criteria
outputs[3].convertTo(outputs[3], CV_16UC1);
//Multiply R2 by 1000 to get 3 decimals
outputs[4] *= 1000;
outputs[4].convertTo(outputs[4], CV_16UC1);
//Multiply d by 1000 to get 3 decimals
outputs[5] *= 1000;
outputs[5].convertTo(outputs[5], CV_16UC1);
//Multiply S/N by 100 to get 2 decimals
outputs[6] *= 100;
outputs[6].convertTo(outputs[6], CV_16UC1);

fs::path outputPath = inputPath.parent_path() /= inputPath.stem();
cv::imwrite(outputPath.string() + "_A.png", outputs[0]);
cv::imwrite(outputPath.string() + "_B.png", outputs[1]);
cv::imwrite(outputPath.string() + "_H.png", outputs[2]);
cv::imwrite(outputPath.string() + "_stopCrit.png", outputs[3]);
cv::imwrite(outputPath.string() + "_R2.png", outputs[4]);

```

```
cv::imwrite(outputPath.string() + "_d.png", outputs[5]);  
cv::imwrite(outputPath.string() + "_snr.png", outputs[6]);  
  
//delete[] angles;  
return 0;  
}
```

# Appendix F

## Bootloader documentation for the instrument controller project

The following is an example of the project documentation in development for the instrument controller. The bootloader covered in this manual is currently available in the project repository and the associated firmware is in development at the time of this writing. A functional bootloader will aid in adoption and adaptation of the controller in a broader range of applications.

# Micro-Controller: Bootloader installation and operation

Ver 1.0

2019/03/05

Author: Marshall Colville

Email: [mjc449@cornell.edu](mailto:mjc449@cornell.edu)

Copyright (c) 2019, Marshall Colville

All rights reserved.

This documentation, relevant source code and hardware designs are released under the FreeBSD

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of product specifications, source code, and documentation must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS PRODUCT IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the hardware, software, and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Micro-Controller project, Paszek or Zipfel Labs or Cornell University.

## Contents

<a href="#">Abbreviations</a>	4
<a href="#">Introduction</a>	5
<a href="#">Hardware</a>	6
<a href="#">External connections</a>	6
<a href="#">Internal connections</a>	7
<a href="#">Bootloader Installation</a>	9
<a href="#">Prerequisites</a>	9
<a href="#">Precautions:</a>	9
<a href="#">From a pre-built binary</a>	10
<a href="#">Connect the Programmer</a>	10
<a href="#">Installing the bootloader</a>	11
<a href="#">From source code</a>	14
<a href="#">Connect the Programmer</a>	14
<a href="#">Building the bootloader</a>	14

## Abbreviations

<b>Abbreviation</b>	<b>Name</b>
<b>I/O</b>	Input/output
<b>HID</b>	Human interface device
<b>ICSP</b>	In-circuit serial programming
<b>MLA</b>	Microchip Libraries for Applications
<b>PCB</b>	Printed circuit board
<b>UC</b>	microcontroller
<b>USB</b>	Universal serial bus
<b>ANx</b>	Analog output x
<b>DIx</b>	Digital input x
<b>DOx</b>	Digital output x
<b>GND</b>	Ground / VDD
<b>DCx</b>	Bipolar analog output x
<b>FNx</b>	Function/waveform generator output x
<b>IPE</b>	Integrated Programming Environment
<b>IDE</b>	Integrated Development Environment

## Introduction

The Micro-Controller Project is an open-source solution for hardware control and synchronization of scientific instruments. The focus of the project is to provide a general-purpose platform for integrating multiple peripheral devices in custom optical microscopes, enabling rapid prototyping and development of new tools and techniques. The hardware portions of the project are designed to be flexible and generic to increase the range of applications and portability of the device.

The hardware, software, and firmware are released under the permissive FreeBSD License to encourage contributions from users and developers, expanding upon the expertise and applications of the project's originators. The bootloader source code was derived from the Microchip® USB HID bootloader included in the MLA and is licensed under the Apache License, v2.0 (<http://www.apache.org/licenses/LICENSE-2.0>). Source code, design files, and documentation are hosted online at <https://github.com/mjc449/SAIMscannerV3.git>. For the latest documentation and source files please visit the project repository.

This manual covers the USB HID bootloader portions of the project. The bootloader is a small piece of code that resides below the application firmware. On a device reset the bootloader will either enter programming mode or direct execution to the application firmware. Typically, reprogramming the microcontroller is done via ICSP with a dedicated hardware programmer. Reprogramming via ICSP requires the user to open the controller enclosure, connect the programmer hardware and place the device in programming mode via a switch on the PCB. In programming mode, the bootloader can rewrite the flash memory segments reserved for application code. The updated firmware is received from the host system over the USB connection, providing a convenient update mechanism that eliminates the additional hardware and physical access to the PCB required for ICSP.

## Hardware

### External connections

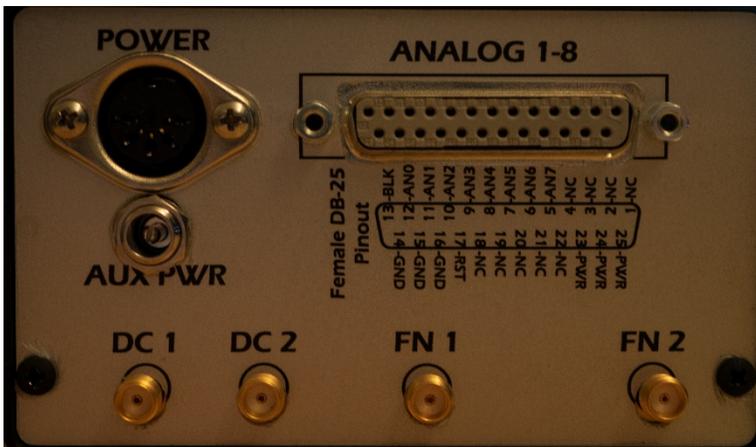


SER/DIO3 – Reconfigurable serial port (SPI, UART) or digital I/O. The function of the port is set in the application firmware and can be changed programmatically at runtime.

USB – USB full-speed type B connection to host

DIO1 – Digital I/O port 1. Reconfigurable at runtime as inputs or outputs

DIO2/TRIG - Digital I/O port 2. D2-D3 are reconfigurable as hardware interrupt sources in application firmware and can be changed programmatically at runtime to switch between interrupt and general purpose I/O.



POWER – Five-pin 180° DIN power jack. Pinout going counter-clockwise from left: +5V, +12V, GND, -12V, GND

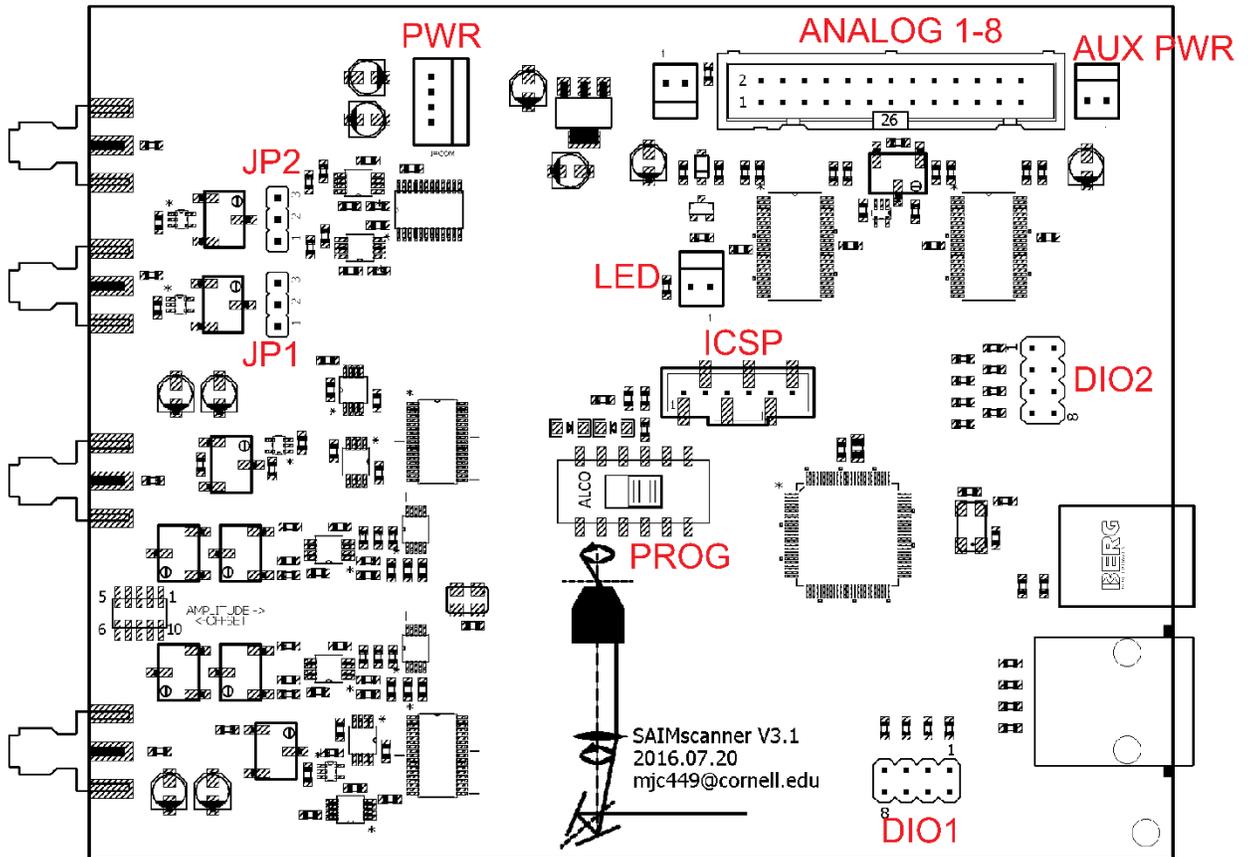
AUX PWR – Power supply pass-through to pins 23, 24, 25 on DB-25 connector.

ANALOG1-8 – 8 Channel analog output (0-10V) with 10V digital blank, ground and power.

DC1, DC2 – Bipolar analog outputs

FN1, FN2 – Function generator outputs

## Internal connections



PWR – Power connector

GND

-12 V

+12 V

+5 V

JP1/JP2 – DC output reference selection. Jump pin 1 to 2 for 10 V fixed reference or 2 to 3 for 0-10 V settable through the trim pot.

ANALOG 1-8 – Multiplex +10V analog outputs.

NC 1, 3, 5, 7, 10, 12, 14, 16, 18, 26

GND 2, 4, 6

AUX PWR 20, 22, 24

+5 V RESET 8

+10 V BLANK 25

ANALOG 1-8 9, 11, 13, 15, 17, 19, 21, 23

AUX PWR – External power supply passthrough to ANALOG connector.

LED – Front panel LED, connected to ANALOG +10 V BLANK pin.

DIO1/DIO2 – digital I/O headers.

PROG – slide switch to select between ICSP program mode and application run mode. In program mode the red LED is on. In run mode the red LED is off and the green LED is on. Also used to place the device in bootloader mode.

ICSP – Serial programming port. With the PROG switch in run mode all pins are NC. The following functions are available in run mode.

MCLEAR – active low master clear

VDD - +3.3 V logic level

VSS – Digital ground level

PGED – serial data

PGEC – serial clock line

## Bootloader Installation

### Prerequisites

The bootloader must be programmed into the controller via ICSP before application code can be programmed over USB. In the case of a new controller that has not had the bootloader installed, a corrupted installation or upgrades to the bootloader firmware the following tools will be required:

A compatible serial programmer. This manual assumes a PICkit4 available from Microchip is being used. Alternatives exist, however it is up to the user to determine how to install the bootloader if other hardware is being used.

The Microchip Libraries for Applications available for free download from

<https://www.microchip.com/mplab/microchip-libraries-for-applications>

Microchip MPLAB IPE available for free download from

<https://www.microchip.com/mplab/mplab-integrated-programming-environment>

The following components are only required for building the bootloader from source code. Development and customization of the bootloader requires only the compiler, however this manual uses the MPLAB X IDE in the examples and does not cover command-line compilation. These components are not required for installing a pre-build binary.

Microchip's MPLAB X IDE available for free download from

<https://www.microchip.com/mplab/mplab-x-ide>

Microchip's XC16 C Compiler available for free download from

<https://www.microchip.com/mplab/compilers>

Precautions:

Opening the enclosure exposes the PCB. To prevent damage from static discharge do not remove the PCB from the enclosure base.

Handle the unit by the enclosure base.

Unplug the power supply and disconnect any peripherals before opening the enclosure.

Take care not to touch or allow anything conductive to contact the ICSP header when the controller is in program mode (red LED illuminated), as this can damage or destroy the controller.

### From a pre-built binary

#### Connect the Programmer

Remove all connections to the micro-controller unit.

Remove the two phillips head screws from the front of the micro-controller, freeing the front panel from the enclosure.

Gently pull the front panel outward and downward from the rest of the enclosure.

With the ribbon cables and wires clear, slide the enclosure top free of the base and set aside

Disconnect the 25 conductor ribbon cable from the header on the PCB assembly.

Remove jumpers from the the ICSP port (if present)

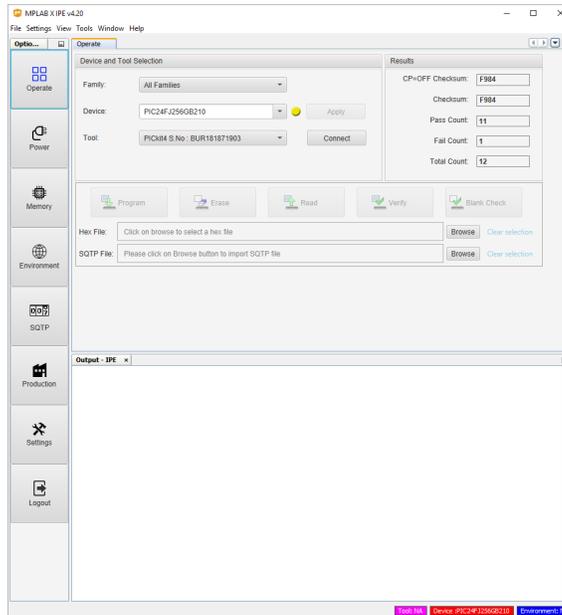
Connect the programmer to the ICSP header. Pin 1 on the header should correspond to the arrow on the programmer inline socket.

Slide the program switch to the program position (toward the ICSP port).

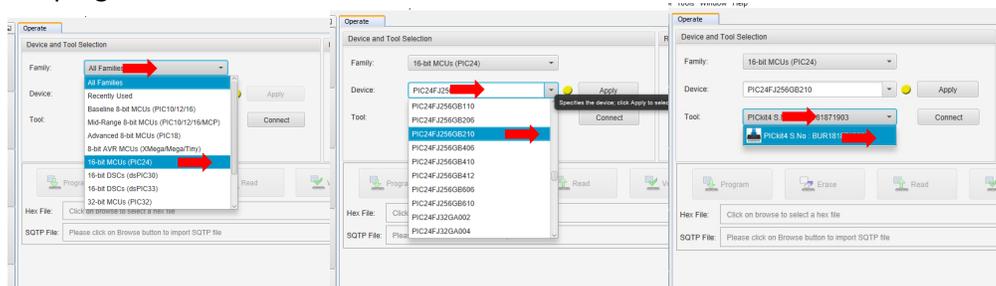
Connect the power supply to the controller, ensuring the red program LED is illuminated.

## Installing the bootloader

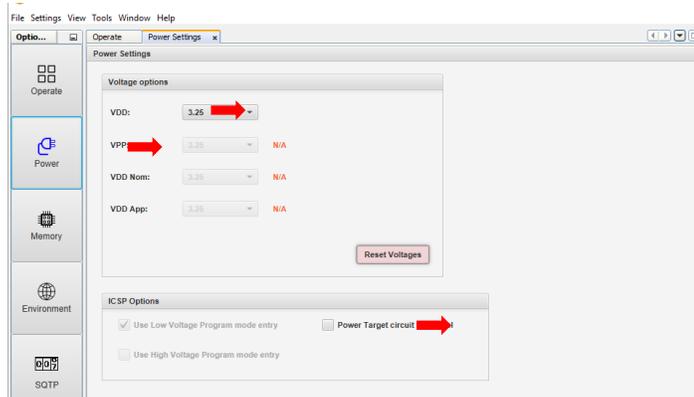
With the programmer connected and the controller powered in program mode, launch MPLAB X IPE



In Device and Tool Selection select “16-bit MCUs (PIC24)” for Family, “PIC24FJ256GB210” from Device, and PICkit4 S.No : XXXXXXXX” from Tool, where XXXXXXXX is the serial number of the programmer.

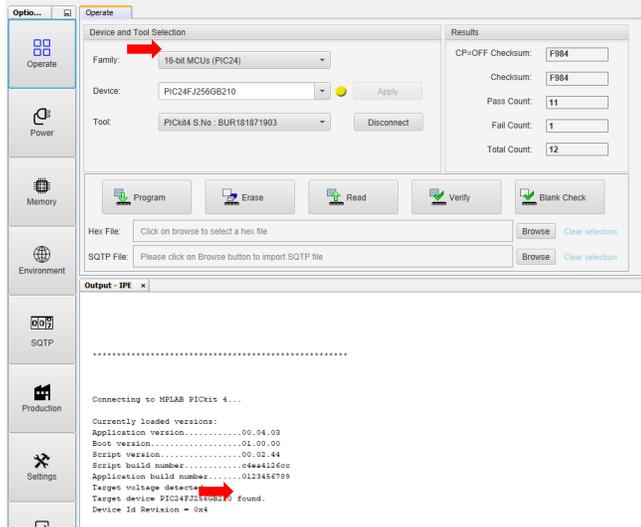


From the toolbar select “Settings>Advanced mode”. An options toolbar will appear to the left of the window. Select “Power” and check that VDD is set to 3.25 V and “Power Target circuit from Toll” is not selected.

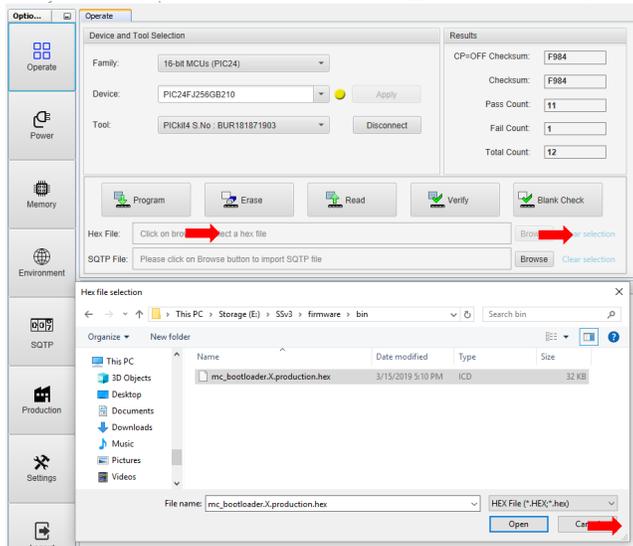


On the Operate tab click “Connect”. The device information will be printed to the output

window. The specific output may vary depending on the tool and version, however ensure that “Target voltage detected” and “Target device PIC24FJ256GB210 found.” are present.



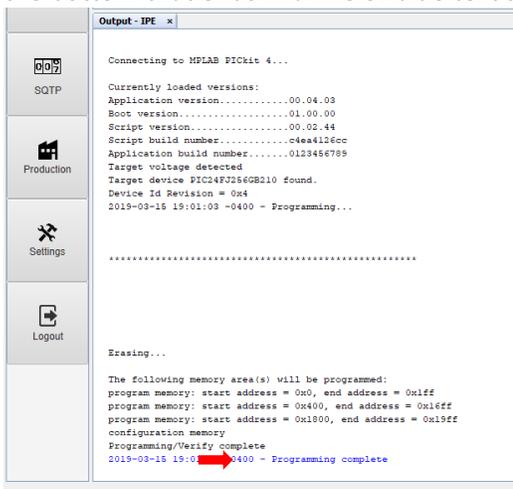
Select “Browse” or enter the path to the bootloader Hex file. The pre-built hex file can be found in the project repository at:  
SAIMscannerV3\firmware\bin\mc\_bootloader.X.production.hex



Check the output window to make sure the file loaded correctly. If so there will be a line of blue text ending in “ – Hex file loaded successfully.” Once the file is loaded click “Program”



Once programming has finished check the output window. There should be a block of text at the bottom that ends in a line of blue text stating “- Programming complete”.



Click “Disconnect”, the unplug the programmer from the board, disconnect the power supply and return the programming switch to the run position.

Reassembling the controller in the enclosure is the reverse of disassembly from the previous section. At this point the bootloader has been installed and all further programming can be done via the USB interface.

## From source code

### Connect the Programmer

Follow the instructions above in “From a pre-built binary” for connecting the PICkit4 to the target controller. The procedure is much the same for other tools, however make sure that the correct connections are made on the ICSP port or the controller could be damaged or destroyed.

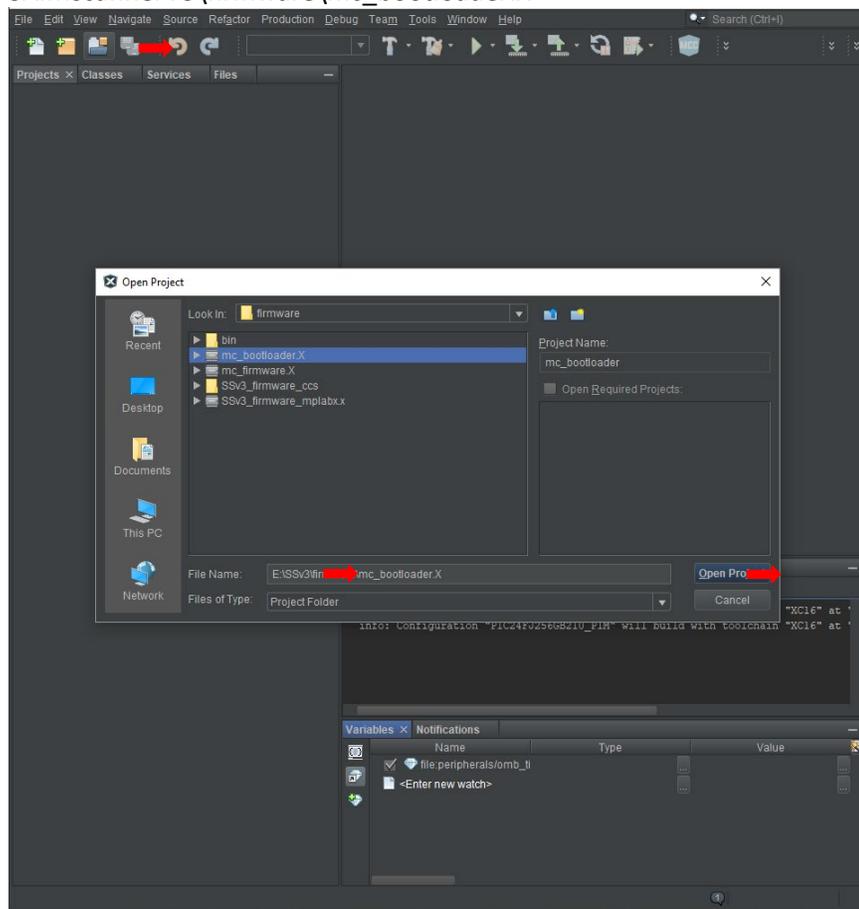
### Building the bootloader

Clone or download the latest version of the project repository from

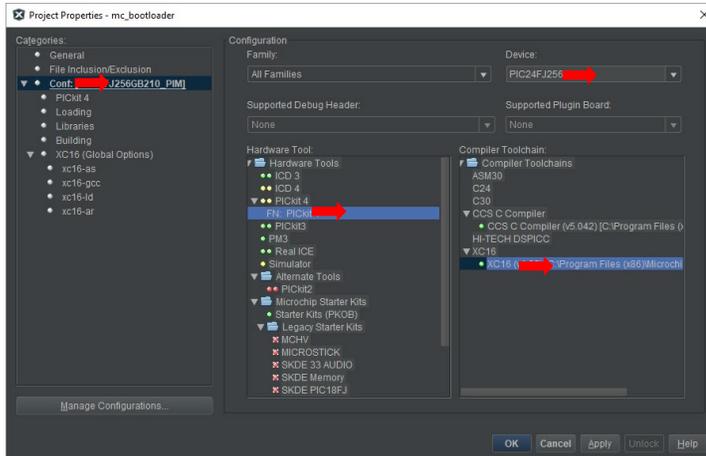
<https://github.com/mjc449/SAIMscannerV3.git>

Open MPLAB X IDE and click “Open Project” in the toolbar at the top of the window or select “File > Open Project...” from the menu. Select the project folder in the dialog window and click “Open Project”. The bootloader project can be found at:

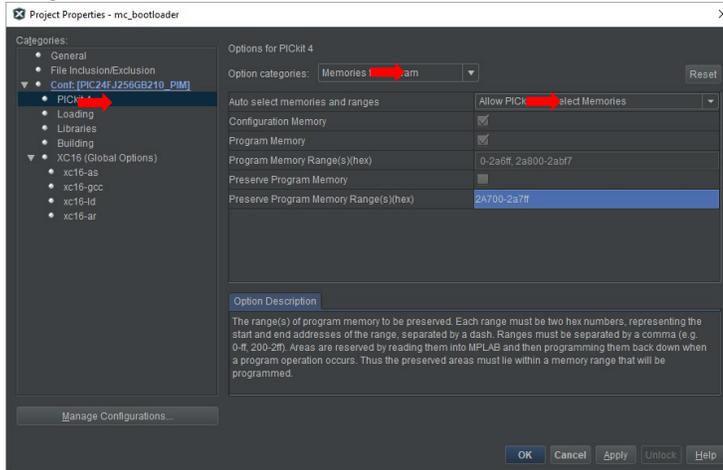
SAIMscannerV3\firmware\mc\_bootloader.X



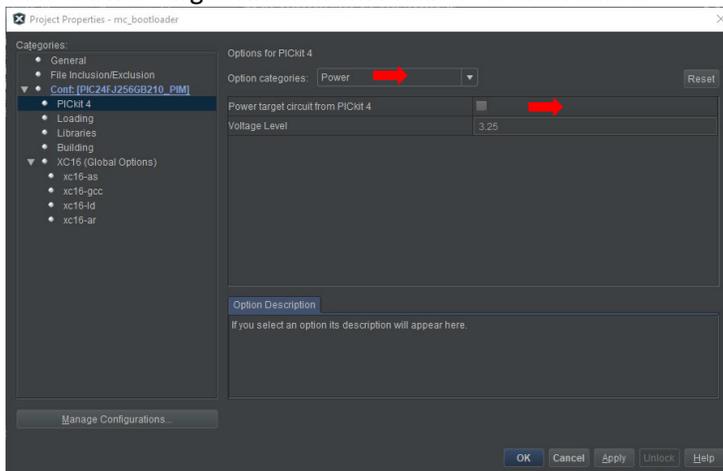
Select “File > Project Properties – mc\_bootloader” from the menu. In the configuration window ensure that PIC24FJ256GB210 is selected for “Device”, XC16 for “Compiler Toolchain”, and PICkit 4 for “Hardware Tool”. If you are using a different programmer select it from the list, or if it is not available you will have to use the programmer’s own software to load the bootloader hex file to the controller.



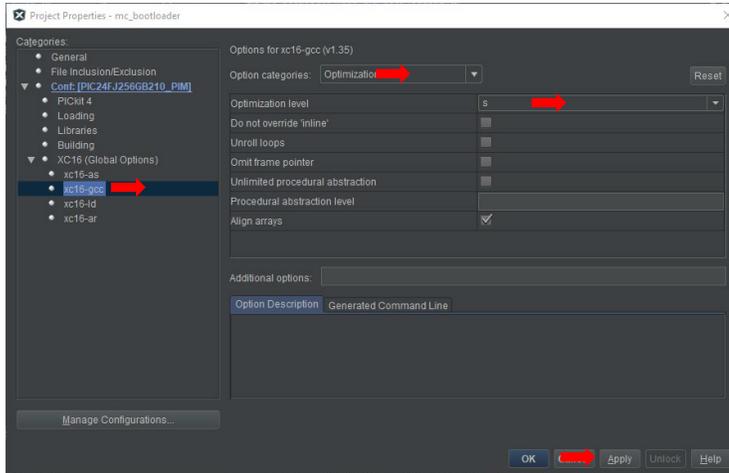
In the properties tree select PICkit4. In the dropdown at the top select “Memories to Program” and ensure that “Allow PICkit 4 to Select Memories” is selected.



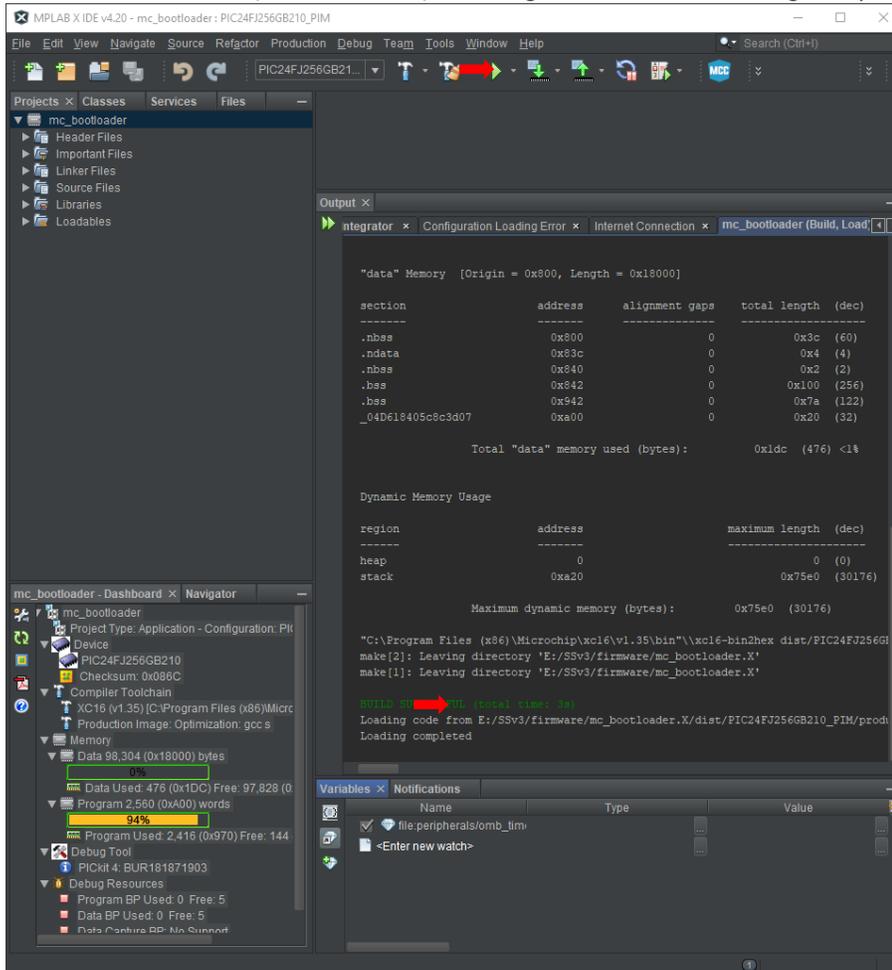
Choose “Power” from the dropdown at the top of the PICkit 4 properties window and ensure that “Power target circuit from PICkit 4” is unchecked.



Select select xc16-gcc from the properties tree. Select “Optimizations” from the dropdown menu at the top of the page. Ensure that the optimization level is set to “s”. Click “OK” to close the options window.

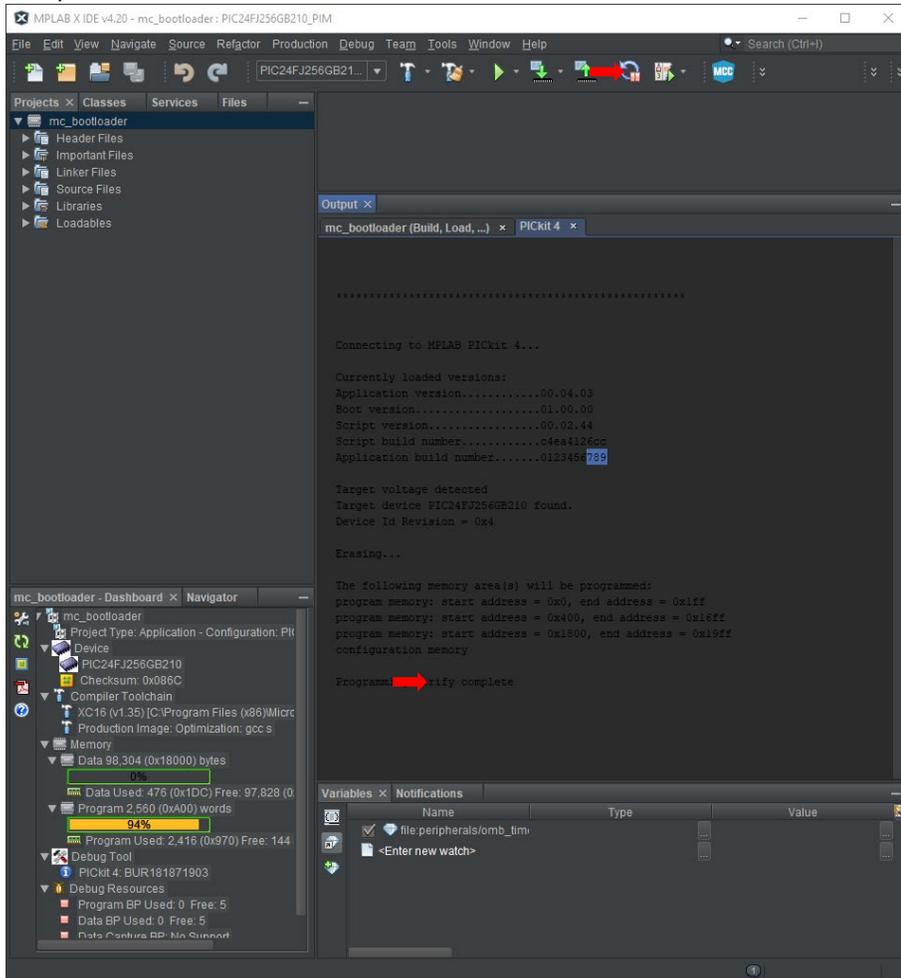


To build the project only without concurrently programming the device, select “Production > Build Project” from the menu or from the toolbar click on the hammer icon to build the project. The output window will display the build info. The output should end with the lines “BUILD SUCCESSFUL (total time: XX); Loading code from ...; Loading completed”.



To build and program the controller in a single step make sure that the PICkit 4 is connected to the controller. Place the controller in program mode with the switch on the PCB. Connect

the power supply to the controller, and ensure that the red LED next to the ICSP port is on. In MPLAB click the “Make and Program Device” button (green arrow pointing toward chip icon) on the toolbar. The build output should look similar to that below. A second output tab will open for the programmer. Check that the output ends with “Programming/Verify complete”.



Disconnect the power supply and programmer from the controller. Move the programming switch back to run mode and replace any jumpers (if present) on the ICSP pins. Reassemble the controller.

## Bootloader operation

### Host software

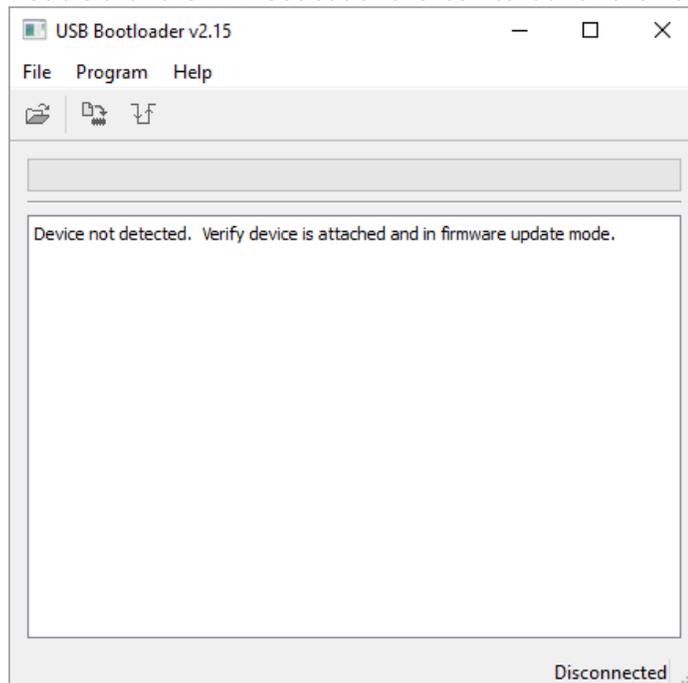
The bootloader firmware is a modified version of Microchip's USB HID bootloader firmware provided in the Microchip Libraries for Applications (MLA). For this reason the easiest way to program the micro-controller unit is using the host program included in the MLA.

The MLA is available at: <https://www.microchip.com/mplab/microchip-libraries-for-applications>

Install the MLA according to the prompts. The only component needed for this project is the USB Demo Projects, and the others can be skipped to save disc space. To do this, at the "Select Components" step uncheck everything except "USB Demo Projects" then click "Next". Continue through to complete the installation.

Once the installation has finished navigate to the bootloader folder. If you opted for the default path this will be: C:\microchip\mla\<version>\apps\usb\device\bootloaders\. The host-side software can be found in: bootloaders\utilities\bin\<platform>\HIDBootloader.exe.

Double click the HIDBootloader.exe icon to launch the host software.



## Placing the micro-controller unit into bootloader mode

There are 3 means of entering bootloader mode on the device:

Through the application firmware by setting a flag in flash memory and restarting the device.

Through hardware. This option requires disassembling the enclosure and should only be used as a last resort.

If the controller detects application code corruption or if the application code reset is blank.

These mechanisms are not accessible to the user.

### Firmware method

The final instruction in application memory (address 0x2ABF6) is reserved for system flags.

On startup the address is read into data memory and compared to determine whether to enter the bootloader or proceed to application code. To enter the bootloader the 24<sup>th</sup> bit is set to 0. Once set the bit can only be cleared by issuing a reset command over USB. A hard reset, loss of power, or a reset trap will not clear the flag.

The following block of code will set the goto bootloader flag:

```
#include <libpic30.h>
//Start by reading the flash block into RAM
uint32_t programBlockBegin = 0x02A7FE;
uint16_t memIdx;
uint32_t wMemSave[512];
//Blocks are 512 instructions = 1024 words
//Read the entire block into RAM
For (memIdx = 0; memIdx < 512; memIdx++){
    wMemSave[memIdx] = ReadProgramMemory(programBlockBegin + 2 * memIdx) &
        0x0FFFFFFF;
}
//Set the bootloader flag
wMemSave[511] &= 0x0FFFFFFE;
//Erase the entire block
_erase_flash(0x02A7FE);
//Write back the program memory
for(memIdx = 0; memIdx < 512; memIdx++){
    if(wMemSave[memIdx] & 0x0FFFFFFF != 0x0FFFFFFF){
        _write_flash_word24(programBlockBegin + 2 * memIdx, wMemSave[memIdx]);
    }
}
```