Real-time 3-d Localization using Radar and Passive Surface Acoustic Wave Transponders

by

Jason Michael LaPenta

Submitted to the Department of Media Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of
Masters of Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
August 2007

© Massachusetts Institute of Technology 2007. All rights reserved.

Author . . . .
Department of Media Arts and Sciences,
School of Architecture and Planning
August 11th, 2007

Certified by . . . .
Professor Joseph A. Paradiso
MIT Media Laboratory
Thesis Supervisor

Read by . . . .
Professor Henry I. Smith
Department of Electrical Engineering & Computer Science
Thesis Reader

Read by . . . .
Principal Research Scientist V. Michael Bove, Jr.
MIT Media Laboratory
Thesis Reader

Accepted by . . . .
Professor Deb Roy
Chairperson, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Real-time 3-d Localization using Radar and Passive Surface Acoustic Wave Transponders

by

Jason Michael LaPenta

Submitted to the Department of Media Arts and Sciences,
School of Architecture and Planning
on August 11th, 2007, in partial fulfillment of the
requirements for the degree of
Masters of Science

Abstract

This thesis covers ongoing work into the design, fabrication, implementation, and characterization of novel passive transponders that allow range measurements at short range and at high update rates. Multiple RADAR measurement stations use phase-encoded chirps to selectively track individual transponders by triangulation of range and/or angle measurements. Nanofabrication processes are utilized to fabricate the passive surface acoustic wave transponders used in this thesis. These transponders have advantages over existing solutions with their small size ($mm \times mm$), zero-power, high-accuracy, and kilohertz update rates. Commercial applications such as human machine interfaces, virtual training environments, security, inventory control, computer gaming, and biomedical research exist.

A brief review of existing tracking technologies including a discussion of how their shortcomings are overcome by this system is included. Surface acoustic wave (SAW) device design and modeling is covered with particular attention paid to implementation of passive transponders. A method under development to fabricate SAW devices with features as small as $300nm$ is then covered in detail. The electronic design of the radar chirp transmitter and receiver are covered along with the design and implementation of the test electronics. Results from experiments conducted to characterize device performance are given.

Thesis Supervisor: Professor Joseph A. Paradiso
Title: MIT Media Laboratory

Biography

Jason LaPenta received his B.S. in Electrical engineering from the University of Illinois in Urbana–Champaign (UIUC) in January of 2001. He specialized in control systems and worked at a research assistant on vision tracking projects. While at UIUC, Jason's summers were spent interning at Philips Semiconductors (Zürich, Switzerland), Boeing (Seattle, Washington), and Procter and Gamble (Cincinnati, Ohio). After graduation, Jason worked for five years in the control systems group at MIT Lincoln Laboratories on such projects as the Mars laser communication experiment.

Jason left Lincoln Laboratory in 2005 to pursue graduate research at the MIT Media Laboratory in the Responsive Environments Group headed by Professor Joseph Paradiso.

While that demands of graduate school leave little free–time, when available Jason enjoys Underwater Hockey, caving, kayaking, sailing, and woodworking.

Acknowledgments

This thesis, and all the wonderful opportunities I have had at MIT were possible because of my adviser Professor Joe Paradiso, the Responsive Environments Group, and the Media Laboratory community. Without professor Paradiso's commitment to this project, especially obtaining the substantial resources necessary, none of this work would have been possible. I would like to especially thank Professor Paradiso for all of the hard work put into this project.

Microfabrication would have been an insurmountable obstacle without the generous support and dedication of the faculty, scientist, students, and staff of the Experimental Microfabrication Laboratory (EML), Nanostructure Laboratory (NSL), and Research Laboratory of Electronics (RLE). From the very first day I stepped into a fabrication laboratory, knowing absolutely nothing about applied fabrication, Kurt Broderick patiently mentored me through every step of fabrication to realize my first operational devices. I want to thank all the people at the the Nanostructures Laboratory, who demonstrated a truly exceptional level of collaboration, helpfulness, and openness that I'm am very lucky to have been a part of. Laboratory directors Professor Karl Berggren and Professor Hank Smith provided invaluable insight into many of the fabrication problems I encountered. Their hard work and dedication provided the laboratory, tools, and environment of collaboration essential to fabricating my high-frequency devices. James Daley worked very hard to keep all the temperamental laboratory machinery running smoothly and promptly provided me with all the supplies and processing I needed. Many long hours were spent by Mark Mondol checking and writing my designs using the VS-26 SEBL tool and teaching me to use the Raith–150 SEBL tool. Whenever I had a quick question, Tim Savas would drop what he was doing to answer and show me the proper methods. In consulting with Tim I learned a lot about practical fabrication considerations that are only learned

in application. There are too many students who helped me in NSL to mention here, though I greatly thank all of them and the NSL community.

Daniel Smalley, a friend and fellow Media Arts and Sciences graduate student, gave an enormous amount of advice during our many discussions of our similar projects. I would like to thanks Daniel for the countless hours of conversation through which his suggestions and insights saved even more hours of work. A special thanks to Roshni Cooper, the undergraduate research assistant whose hard work implementing the user interface and FPGA code allowed me to focus on fabrication while collecting data.

And I especially thank my wife Cindy for all she has done during the most demanding time of my life.

Contents

1 Introduction **23**

1.1 Analysis 30

1.2 Motivation 34

1.3 Related Work 35

2 Surface Acoustic Wave Devices **39**

2.1 Properties of Lithium Niobate 42

2.2 Modeling 43

2.3 Design 48

3 Micro–Fabrication **57**

3.1 Mask Design and Layout 62

3.2 Substrates 69

3.3 Substrate Preparation and Cleaning 71

3.4 Lithography 74

3.4.1 Photoresists 76

3.4.2 Coating 77

3.4.3 Contact Lithography 78

3.4.4 Conformable Vacuum Mask Lithography 79

3.4.5 Scanning Electron Beam Lithography 80

3.4.6 Exposure 82
3.4.7 Resist Developing 85
3.5 Descum 86
3.6 Deposition 88
3.7 Lift–off and Etching 90
3.8 Mounting/Packaging 96

4 Electronic Design 99
4.1 System Architecture 102
4.2 Analog Components and Test Equipment 106
4.3 Impedance Matching 111
4.4 SAW Filter Measurements 113
4.5 SAW Transponder Measurements 115
4.6 FPGA Hardware Description 116
4.7 Software 123

5 Testing and Characterization 127
5.1 SAW Device `0x2C9` 127
5.2 SAW Device `0x2E8` 133
5.3 SAW Device `0x0E3` 135

6 Conclusion 137

Bibliography 141

A Lithography Masks 151
A.1 Mask 1 Generation Scripts 151
A.2 Mask 2 & 3 Generation Scripts 178

B Fabrication Processes **189**

B.1 Device Fabrication Process Details 189

B.1.1 Cleaning 189

B.1.2 Lithography 191

B.1.3 Lift–Off and Etching 194

B.1.4 Vacuum Mask Fabrication 196

B.1.5 Raith 150 Operational Procedure 198

C OAI documentation **205**

C.1 Housing Drawings 205

C.2 Mask Chuck Drawings 214

D Auxiliary Electronics **217**

E FPGA VHDL/C/XPS **227**

F MATLAB Source Code **257**

List of Figures

1-1 Notational diagram of three measurement stations tracking a single SAW transponder. 25
1-2 Hypothetical SAW transponder to illustrate the approach taken for identification and common mode rejection. The IDTs are shown with phase–shift encoding. 26
1-3 Illustrates the signal flow of RF received and transmitted signal and surface waves, along with the effects of the signal processing. 27
1-4 Macro illustration of signals received by measurements stations at different locations (not to scale either in time or magnitude). 28
1-5 Digitization scheme for enhanced low–cost identification and timing. Each digital channel, a or b, represents the thresholded receive signal at a particular cutoff. 28
1-6 Time measurement to peak of the correlated signal. 29
1-7 Illustrates the time–of–flight of the interrogation pulse being returned by a multipath object and a transponder. 30

2-1 Simplified diagram of basic SAW transponder functional components[1]. 40
2-2 Saw correlator with corresponding code and output pulse waveform. *"time / position"* is the waveform over time at a single point between IDTs, or the waveform over position between IDTs at a particular time. 41

2-3 Illustration of the crystal cut of a 128°–RY wafer[2], with an X–Flat, from Crystal Technologies Inc. 42

2-4 Illustration of the crystal cut of a Y wafer [2], with a Z–Flat, from Crystal Technologies Inc. 43

2-5 SAW filter made with mask 1. Device `0x2C9`, 25 finger pairs that are $200\mu m$ long. 44

2-6 MATLAB™ simulation of the normalized impulse response of a $323MHz$ SAW band–pass filter shown if figure 2-5. 45

2-7 Result of an impulse through device `0x2C9` as recorded on an oscilloscope. 47

2-8 Parameters for the design of a SAW IDT in this project. 48

2-9 The three types of SAW reflectors were implemented on mask 1. . . . 50

2-10 Narrow–band output correlator with $n = 5$ Barker code illustration. . 50

2-11 Narrow–band output trace for figure 2-10. 51

2-12 Narrow–band output correlator with code $[1, 1, 1, 0, 1, 1, 1, 1, 1, 1]$ utilizing one finger–pair encoding. 52

2-13 Narrow–band correlator response for device fabricated on mask 3, shown in figure 2-12. Generated with `nb.m` in appendix F. 52

2-14 Wide–band output correlator illustration with $n = 5$ Barker code with two finger–pair encoding. 53

2-15 Wide–band output trace for figure 2-14. 53

2-16 Wide–band output correlator. 54

2-17 Wide–band correlator response for device fabricated on mask 3, shown in figure 2-16. Generated with `wb.m` in appendix F. 54

2-18 Dual narrow–band transponder with a Barker code of length five with a five finger encoding. 55

2-19 Dual narrow–band transponder's simulated waveform. The *"rx signal"* was transmitted by the measurement station and is what the input IDT sees. The *"first response"* and *"second response"* are the waveforms from each output IDT. 55
2-20 Simulation including the effect of surface waves originating from the output IDTs of the transponder shown in figure 2-18. 56

3-1 Simplified one–step lift–off process. 60
3-2 Simplified wet–etch process. 61
3-3 Artwork for mask 1 showing a die section. 63
3-4 Typical layout of a device on Mask 1. 63
3-5 Artwork for mask 2. 65
3-6 Artwork for mask 3 version 2. Bond–pads and leads are on a different layer than the devices. 68
3-7 Artwork for mask 3 version 3. 69
3-8 Micrography of PMMA scum on a quartz light–field mask with chrome. 73
3-9 Typical lithography process steps used in this project. SEBL or contact printing can be used to transfer a pattern to the photoresist. 75
3-10 Spin curve of 3 parts Anisole to 1 part PMMA 950k A11. 77
3-11 Fringe patterns resulting from imperfect contact between mask and substrate while under vacuum and illuminated with a green monochromatic light. This shows the OAI 77mm chuck with a top–down rubber gasket over the mask. The mask was 1mm quartz and the substrate was a 77mm silicon wafer. 80
3-12 Simplified diagram of a SEBL tool. The detector is used for imaging test particles that are used to adjust for focus, aberration, stigmation, and write field alignment. 81

3-13 Micrographs of field stitching problems. 82

3-14 Exposure test features allow inspection of focus, dose, and development parameters. 83

3-15 Micrograph of a dose matrix written in $100nm$ of PMMA on Quartz. Exposed with the Raith 150 SEBL tool. 83

3-16 The top micrograph shows clearing problems with PMMA on Quartz. The bottom two exposure show improperly cleared (left) and properly cleared (right) $4\mu m$ grating of PMMA on Silicon. The darker areas are PMMA and the light areas are substrate. These exposures were made with the OAI deep–UV exposure tool in NSL. 84

3-17 Developed exposure/developing test features showing the degree of over/under developing. The left panel shows underdeveloped clearing problems, and the right panel shows overdeveloping. The center panel show ideal exposure, with the two boxes coming into perfect contact. 85

3-18 Changes in width of $1.1\mu m$ lines due to developer temperature. . . . 86

3-19 Exaggerated illustration of the effect on resist profile from RIE or plasma asher descum processes. The top figure is the photoresist profile after development and before any descum step. 87

3-20 Etch depth vs. time of $106nm$ PMMA on Si in O_2 plasma asher using $50W$ power and $0.305psi$ average pressure. Thickness measured with an ellipsometer. 88

3-21 Results from $2sec$ of O_2 plasma asher and RIE descum methods. The figures show Cr on Quartz and Cr on PMMA before the liftoff step. $10nm$ of PMMA were taken off with the O_2 asher. $7nm$ of PMMA were taken off with the RIE. 89

3-22 Effect of off–angle deposition on side–wall coating. 89

3-23 Micrograph of a problem encounter when depositing 40nm Cr onto 100nm of PMMA on an SiO_2 wafer using the NSL evaporation tool. . 90
3-24 Undercut allows solvent to dissolve photoresist and remove metal overlay. 90
3-25 Damage likely resulting from too much sonication ($> 1min$) during lift–off. 92
3-26 Incomplete liftoff problems as a result of too much chrome being deposited. Shows 500nm line–widths with 210nm of chrome on quartz. The desired thickness of chrome was 100nm. The PMMA thickness was 250nm. 93
3-27 SEM of bridging caused by poor profile and photoresist that was too thin for the thickness of metal deposited. 93
3-28 Illustration of lift–off problems when the metal on top of the photoresist bridges to the metal on the substrate. 94
3-29 SEM showing a wafer after deposition and before lift–off. The images of the 100nm gold particles in the bottom left are given as a reference for image quality. 94
3-30 Micrograph showing a patch of incomplete lift-off due to inspection by the SEM before liftoff to take the image shown in figure 3-29. 95
3-31 Test patterns of Cr on Quartz and Cr on PMMA on Quartz. Same sample as shown in figures 3-29 & 3-30. 95
3-32 Show the change in image contrast before and after the etch has been completed. PMMA was used as a photoresist on top of chrome and quartz. 96
3-33 SAW test devices mounted to a package, wire-bonded, and soldered to a test PCB. 98

4-1 Conceptual measurement station transceiver block diagram. 100

4-2 Measurement station FPGA and transmitter block diagram. 102
4-3 Photo of NECL clock generator (part number PRL-174ANT-1350) and NECL to LVPECL logic level translator (part number PRL-460ANLPD). 103
4-4 Photo of Xilinx ML405 evaluation board. 104
4-5 Illustration of signal conversion from digital to RF. 105
4-6 Photo of analog components from Mini–Circuits®. 108
4-7 $323MHz$ antennas made from 16–AWG copper wire and an SMA connector. Used for preliminary isolated indoor tests. 110
4-8 RF Network Vector Analyzer test setup for reflectance and impedance measurements. 111
4-9 Simple L matching network [3, 4]. 111
4-10 Measured Smith chart on the RF Network Vector Analyzer of a SAW transponder. 112
4-11 RF Network Vector Analyzer test setup for Transmittance measurements. 113
4-12 Measured transmission chart of a SAW filter on the RF Network Vector Analyzer. Notice the anti–resonance artifact, which is ignored for this work. 114
4-13 Setup for transient measurements of a SAW filter using an oscilloscope. 114
4-14 Test setup for SAW transponder measurements. 115
4-15 SAW transponder measurements using antennas. 115
4-16 Block diagram of FPGA PowerPC processor configuration. 119
4-17 Top level block diagram of chirp2 VHDL code. 120
4-18 VHDL code organizational block diagram of the *"chirp2"* peripheral implementing the rocketIO multi–gigabit–transceiver and support FPGA cores. 121
4-19 `chirp_ui.pl` User interface program written in PERL. 124

5-1 Drawing of SAW filter device `0x2C9` 129
5-2 Frequency response from $50MHz$ to $1GHz$. The pass band is at $322.249MHz$. 129
5-3 Pass–band bandwidth shown as $\approx 20MHz$. 130
5-4 Smith chart before impedance matching circuit. 130
5-5 Smith chart after impedance matching circuit with $27nH$ and $10pF$. . 131
5-6 Internal reflection between two IDTs in device `0x2C9`. 131
5-7 Example of inter–finger reflections within a single IDT. 132
5-8 Drawing of SAW filter device `0x2E8` 133
5-9 The sloped frequency response is due to electromagnetic feed–through[5]. 134
5-10 Impulse response of SAW filter Device `0x2E8`. The number of cycles has been labeled (1,10,and 20). Channel Z2 is the zoomed view of Channel C2. 134
5-11 Drawing of SAW filter device `0x0e3` 136
5-12 The four cycle chirp response of device `0x0E3`. Channel Z3 is the zoomed view of the first reflection on Channel C3. 136

B-1 Illustrates the process for making devices by first writing a pattern to a $1mm$ quartz mask master, then transferring the pattern using contact lithography to a conformal mask, and finally patterning the lithium niobate substrate to make the desired devices. 196

C-1 OAI deep–UV aligner radiation shield. Made from Standard Cast Acrylic, $0.236inch$ thick Clear with Bronze Tint, McMaster part numbers. 8505K921, 8505K941, and 8505K951. 206
C-2 OAI $3inch$ chuck assembly. 214
C-3 OAI $3inch$ chuck assembly. 215

D-1 Analog test board. Holds SAW test package, low–noise amplifiers, and power regulators. 217
D-2 Closeup of one low–noise amplifier channel. 217
D-3 PCT top–side silkscreen. 218
D-4 PCB top–side metal layer. 218

List of Tables

1.1 Comparison between widely used localization systems and SAW transponder [6, 7, 8, 9]. 36

3.1 Steps to convert a file to `.kic` for use on the VS-26. 66
3.2 Processing costs and time estimates to process one mask with lift–off. This does not include meteorology. 67
3.3 Suppliers and specifications of substrates used. *Amorphous SiO_2. . . 70

4.1 Components used in this project that were purchased from Mini–Circuits®. 109

5.1 Specifications for device `0x2C9` 128
5.2 Specifications for device `0x2E8` 133
5.3 Specifications for device `0x0E3` 135

B.1 Alconox™ cleaning procedure. 189
B.2 Ultrasonic cleaning with Acetone. 190
B.3 Piranha cleaning procedure. 190
B.4 RCA or standard clean 1 (SC–1) procedure. 190
B.5 Photoresist coating procedure. 191
B.6 Exposing using the EML high–resolution MJB3 aligner. 192

B.7 Exposing using the NSL OAI. 192
B.8 PMMA development procedure. 193
B.9 NR7-1000P development procedure. 193
B.10 Lift–off of PMMA. 194
B.11 Lift–off of NR7–1000P. 194
B.12 Etching chrome to fabricated a patterned mask. 195
B.13 Process steps for making a conformal mask. 197

D.1 Parts list for test board 219

Chapter 1

Introduction

Unhindered localization of objects is desirable for many applications from human computer interaction to product tracking to security. Many researchers and companies have sited tracking of people with special needs, prisoners, or workers in hazardous situations as useful applications of localization[10]. What this project proposes is the use of Surface Acoustic Wave (SAW) devices engineered in a particular fashion to function as transponders for a distributed radar tracking network. The radar network would track the location of the SAW transponders in 3-d with an update rate on the order of $10kHz$ with $10cm$ accuracy. Improvements over these estimations are conceivable in an optimized implementation. A patent application covering the work in this thesis was filed on March 7$^{\text{th}}$, 2006 under the title "Real-time ranging and angle measurements using Radar and Surface Acoustic Wave Transponders."

This thesis covers all of the work completed to date on this project. While ranging results have not yet been obtained, a lot of important progress and experience has been garnered. About two to four months of work remains before initial results are ready. The approach taken for the overall system is covered. The transponder design progress that has been made to date is given as well as future plans for investigation. Micro–

fabrication of the devices is extensively covered, however processes for 1.1μm line–widths have not yet been attained (we are only two to three weeks from fabrication of working 1.1μm devices). The development of a measurement station is underway. The FPGA programs, firmware, and approach are described in detail. Analog RF circuits have been constructed, and recent results will be discussed. There is some work that was done for this project that will not be discussed and is as follows; An analog time–to–digital converter was designed and built, but is impractical and unsuited for time–of–flight ranging in this project. Analog circuits were built to generate a chirp before the FPGA was available. A low–noise amplifier was designed, analyzed, extensively simulated, and built; the details of this amplifier have been left–out because the design is being debugged.

There are two major components of this system; the SAW transponder and the radar measurement station.

A SAW transponder reflects a short pulse of received RF energy back to the measurement station as a phase-encoded pattern of data. An arrangement of correlators and impedance–modulated reflectors on the surface of the SAW transponder return encoded data patterns back to the antenna for retransmission to the measurement station. Several published techniques for selection, identification, and encoding of data with the tags will be applied as well as some novel designs. [11, 12, 13]

SAW devices were originally developed in the 1960's[5, 14, 15]. Their function is based on the principle of generating surface acoustic waves in a piezoelectric substrate. SAW devices are extensively used as passive narrow-band-pass filters in commercial communications equipment such as cellular telephones. Innovation continues with the use of SAW devices for RF-ID applications[12, 16, 17, 10]. The unique properties of SAW devices theoretically makes them an ideal technology for this novel real-time tracking solution.

Radar has been utilized for decades to track a multitude of targets such as aircraft, weather patterns, and cars. Active transponders assist radar identification by responding with an encoded signature. This allows the radar to determine exactly what object is being tracked. Transponders are used in almost all airplanes to aid air traffic controllers in identification [18, 19].

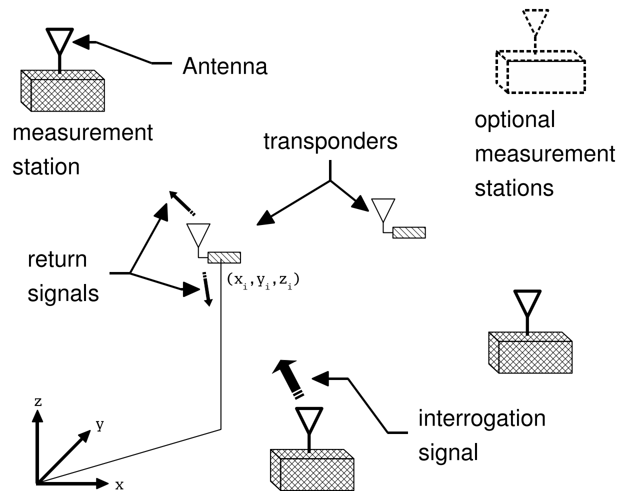


Figure 1-1: Notational diagram of three measurement stations tracking a single SAW transponder.

Each radar measurement station will measure time-of-flight and/or angle of arrival of the signal returned by the SAW transponder, figure 1-1. Simple triangulation calculations are performed on the data from three or more measurement stations to localize a tag. Only one measurement from all of the measurement stations, either time-of-flight or angle-of-arrival, is necessary to localize. However, using more than the minimum necessary signals will improve accuracy. For the simplified system described in this work, the location of the measurement stations must be fixed and known by the algorithm.

Figure 1-2 shows a simplified saw tag with phase encoding used to illustrate the approach we intend to take. The inter–digital transducers (IDTs) convert the electrical

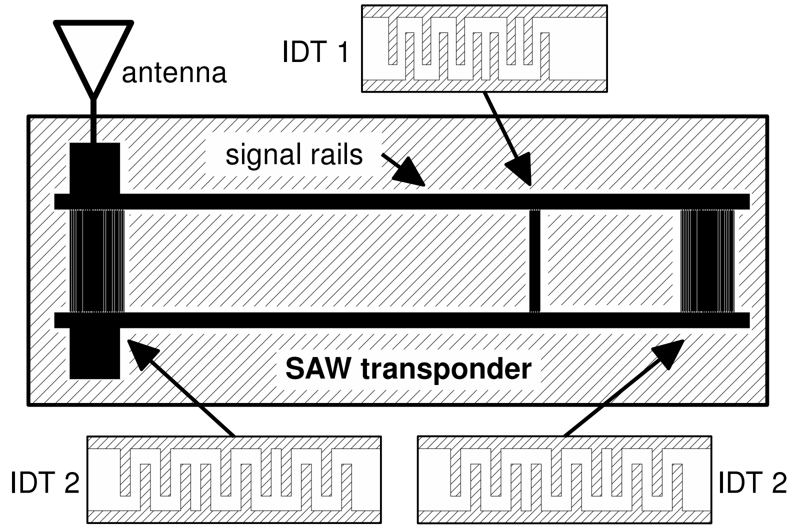


Figure 1-2: Hypothetical SAW transponder to illustrate the approach taken for identification and common mode rejection. The IDTs are shown with phase–shift encoding.

energy into a surface wave (details will be covered in chapter 2). A SAW transponder must be selective to a particular signal, thus the phase encoding. To enhance signal–to–noise and reduce background radar clutter, a built–in delay between the received signal and retransmitted signal must be implemented. A mechanism must also be provided to compensate for effects of temperature on the surface velocity of the SAW device. In this example three different IDTs implement the phase encoding. The codes for this example are very short (five and six bits in length). An actual tag would have up to 128bit code lengths. This particular tag has different receive and transmit codes.

The distance between the IDTs builds in a delay between when the radar signal is received and retransmitted. In this case, IDT 1 has been designed to be selected by the radar chirp from the measurement station, while IDTs 2 & 3 do not correlate to this signal. Because the distance between IDT 1 and IDT 2 is different than the distance between IDT 1 and IDT 3, and the propagation speed is the same, the time between them may be used to take out common mode surface velocity variations.

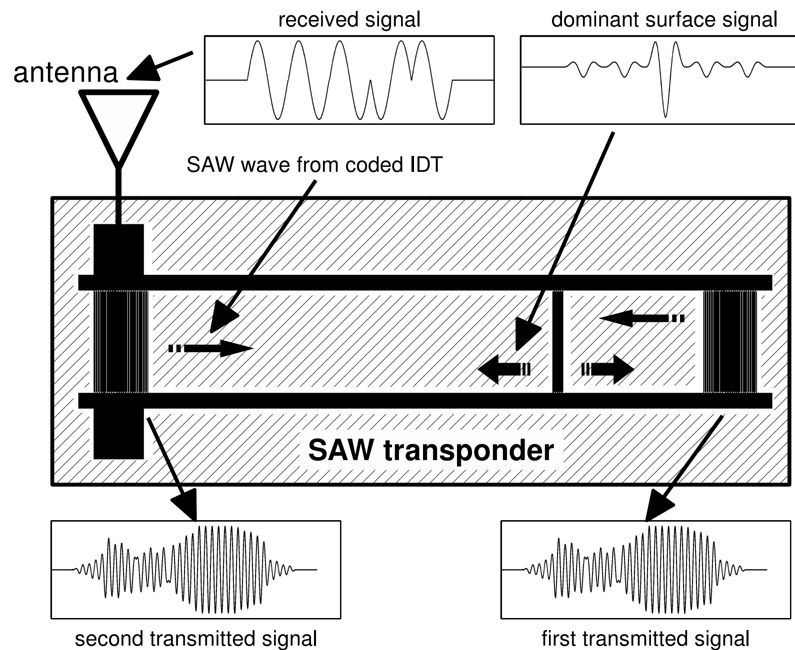


Figure 1-3: Illustrates the signal flow of RF received and transmitted signal and surface waves, along with the effects of the signal processing.

The various signal processing at the different places along the transponder are illustrated in figure 1-3. For the following discussion secondary reflections and uncorrelated signals are not accounted for. Chapter 2 will cover secondary reflections as well as other higher–order effects. The signal received by the transponder's antenna from the measurement station in this example is a Barker code[5, 14]. A Barker code correlated by the proper IDT will create a SAW pulse with unity side–lobes. Other codes are generated pseudo–randomly and the simulated and selected for acceptable side–lobe attenuation[14]. In this case only IDT 1 correlates to this received code. IDTs 2 & 3 do not correlate and launch only a small surface wave. When the surface wave launched by IDT 1 reaches IDT 2 & 3 the transponded signal is transmitted as a convolution of the surface wave and the phase encoded IDTs. This waveform is represented below their respective IDTs. In this case the same waveform is transmitted

by both IDTs, but they may have different encodings if desired.

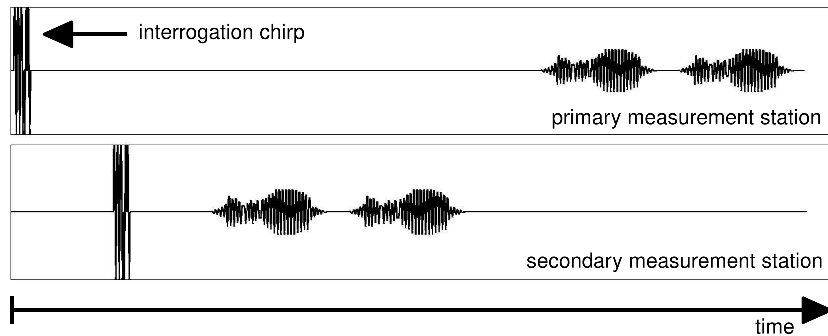


Figure 1-4: Macro illustration of signals received by measurements stations at different locations (not to scale either in time or magnitude).

The transmitted and transponded signals may be received by multiple measurement stations. Each additional measurement station would relay the station's position and the time measured from the interrogation chirp to the received signals to the primary measurement station. Figure 1-4 shows the temporal waveforms from two measurement stations. The primary measurement station in this example is further away from the transponder than the secondary measurement station.

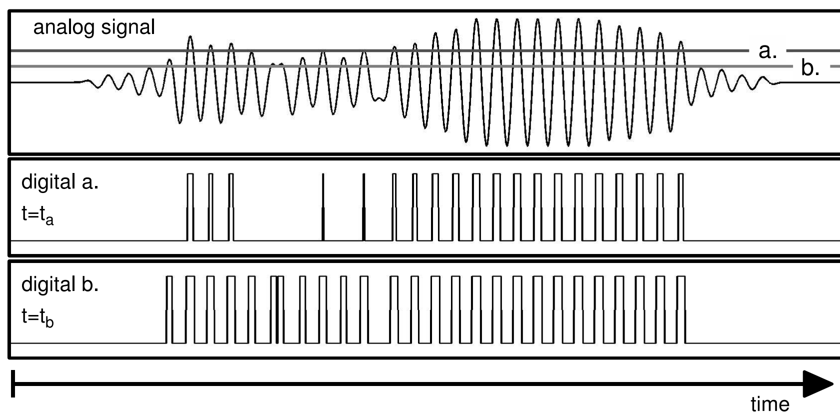


Figure 1-5: Digitization scheme for enhanced low–cost identification and timing. Each digital channel, a or b, represents the thresholded receive signal at a particular cutoff.

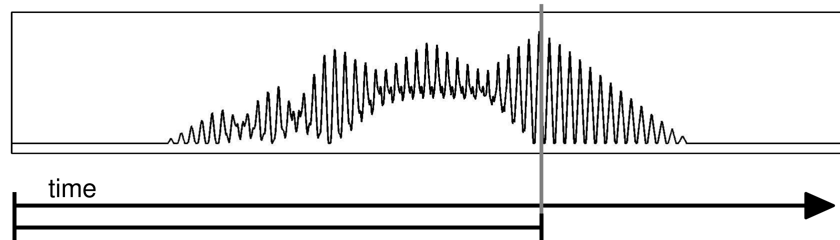


Figure 1-6: Time measurement to peak of the correlated signal.

The approach taken to accurately measure the time delay is illustrated with figures 1-5 & 1-6. The waveform will be measured by clipping and digitizing the received signal at high–speed. The digitized waveform will be captured using the multi–gigabit–receiver (covered in section 4.6). In this example, one digitization channel is used to capture data at two times, represented by (a.) and (b.) in figure 1-5. The digitized waveforms are then auto–correlated to the expected waveform as calculated using a SAW transponder model[20]. The correlated output waveform may look like the one presented in figure 1-6. The correlated waveform will have a peak and that time displacement measurement will be used to derive the time–of–fight for triangulation. Pseudo-random transponder codes will be selected for auto–correlated signals that have significant correlated peaks to allow for improved signal–to–noise performance. The time measurements from the different digitized channels will be approximated into one measurement by averaging the results. Of course, what is achievable in practice depends on many factors, some of which are the signal–to–noise ratio of the sampling circuitry, operating environment, multipath effects, and interference of other tags.

1.1 Analysis

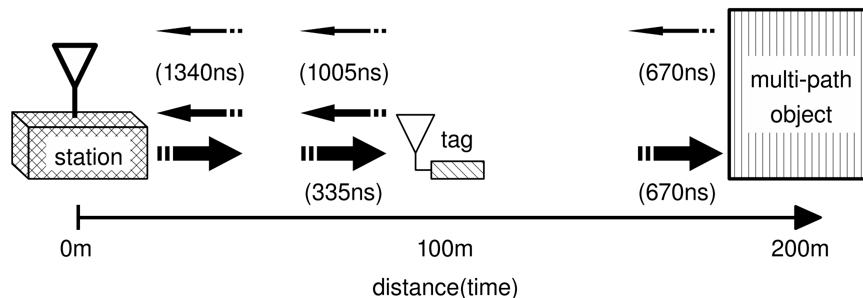


Figure 1-7: Illustrates the time–of–flight of the interrogation pulse being returned by a multipath object and a transponder.

The following is a simple analysis of the requirements and expected performance of the proposed system. The limiting factors for how fast a tag's position can be read is the maximum distance to a tag and the read–out / read–in time. For a maximum range of $100m$, the round trip time for a pulse is $2 \cdot 100m\frac{1}{c} = 670ns$, where c is the speed of light. To allow for background radar clutter to dissipate, a delay is added to the maximum round–trip time. For all signals within $200m$ to dissipate, we need a total delay of $1\mu s$ before the SAW transponders return the signal. Thus the total delay between the time the radar chirp is transmitted and the return signal is received must be at least $1.3\mu s$. A 32–bit code at $2.4GHz$ will take $13ns$ and considering the returned signal is a 64–bit convolved waveform, the total time for both is only $40ns$. Using three measurement stations with one transmitting and two listening, a single update can take place in $1.4\mu s$, or at a rate of $700kHz$. This is a very impressive 3-d localization rate in terms of available technology. Resolution improvements could be attained by integrating the result over time.

The demonstration system will be shifting received data in at a rate of $2.4GHz$. Two phase–locked multi–gigabit–transceivers (explained in section 4.6) channels will be used to double the sampling rate, thus satisfying Nyquist to achieve $2.4GHz$

overall. A very stable clock with $100ppm$ stability over half an hour drives the receiver, which has no significant effect on range measurements. The peak–to–peak clock jitter is $20ps$, which results in an error of $3mm$. Given the round–trip time to and from the target is measured, a factor of two advantage is realized. The best achievable range resolution given the hardware available to us would then be:

$$2c\frac{1}{f} = 2 \cdot 3.8 \cdot 10^8 \frac{1}{2.4 \cdot 10^9} = 6.25cm \tag{1.1}$$

Temperature effects on the surface wave velocity of the transponder is a concern. The properties of $LiNbO_3$, including temperature dependent effects, will be discussed in more detail in chapter 2. For the purposes of this example, a typical SAW substrate has a characteristic surface velocity of $3992m/s$ and temperature coefficient of $75ppm/$℃. Effect due to temperature changes may be compensated for through measurement and calibration.

To analyze the performance improvement with temperature measurement compensation consider the dual reflector configuration. The amount of error for the uncompensated design, e_{uncomp}, will be calculated first as a comparison for the compensated error, e_{comp}. For the $100m$ maximum range, a delay of $d_1 = 670ns$ will be required (Refer to figure 1-7). As a starting point, a $d_2 = 100ns$ delay between the temperature measurement dual reflectors will be chosen. Assuming a temperature discrepancy between the measurement station and the SAW transponder to be $t_{\text{error}} = 10$℃(a very conservative estimate), the approximate error in the distance measurement would then be;

$$\begin{aligned} dv &= v_{\text{sub}} t_{\text{error}} t_{\text{coeff}} \\ &= 3992\frac{m}{s} \cdot 10℃ \cdot 75\frac{ppm}{℃} \\ &= 3\frac{m}{s} \end{aligned} \tag{1.2}$$

Where v_{sub} is the substrate surface velocity of $128°LiNbO_3$, and t_{coeff} is the associated temperature coefficient[5]. With a surface length between the input IDT and the first reflector of

$$\begin{aligned} l &= v_{\text{sub}} \cdot d_1 \\ &= 3992m/s \cdot 670ns \\ &= 2.7mm \end{aligned} \tag{1.3}$$

lending to a distance measurement error of approximately;

$$\begin{aligned} e_{\text{uncomp}} &= c \cdot \left[d_1 - \left(\frac{l}{v_{\text{sub}} - dv} \right) \right] \\ &= 3 \cdot 10^8 \cdot \left[670ns - \left(\frac{2.7mm}{3992m/s - 3m/s} \right) \right] \\ &= 151mm \end{aligned} \tag{1.4}$$

An error of $151mm$ is decent, and probably much worse than what would be expected under normal conditions. However, such an error would be a significant degradation of achievable performance. As briefly discussed at the beginning of chapter 1, a multi–reflector scheme will be used to determine the temperature of the SAW transponder. Several schemes for using SAW transponders to measure temperature and or compensate for temperature have been developed and implemented by others[21, 22]. By using temperature compensation methods much better results should be attainable. SAW transponders have been demonstrated to measure temperature with a resolution of 0.02℃[21]. With such precise temperature measurements the error due to temperature may be reduced through compensation to;

$$\begin{aligned} dv &= v_{\text{sub}} t_{\text{error}} t_{\text{coeff}} \\ &= 3992\frac{m}{s} \cdot 0.02℃ \cdot 75\frac{ppm}{℃} \\ &= 6\frac{mm}{s} \end{aligned} \tag{1.5}$$

lending to a distance measurement error of approximately;

$$
\begin{aligned}
e_{\text{comp}} &= c \cdot \left[d_1 - \left(\frac{l}{v_{\text{sub}} - dv} \right) \right] \\
&= 3 \cdot 10^8 \cdot \left[670ns - \left(\frac{2.7mm}{3992m/s - 6mm/s} \right) \right] \\
&= 0.3mm
\end{aligned}
\tag{1.6}
$$

Therefore, with only 0.3mm of error due to temperature effects, they can be safely ignored for the application and design discussed in this thesis.

Given we are using a low–cost FPGA to digitize the received signal, measuring temperature will be a problem. By utilizing the fast oscilloscope we will be able to measure the temperature accurately enough to prove the concept. Later more capable receiver electronics will be designed to create a cost–effective independent solution.

Another issue with system utilizing RF for tracking is the degradation and spoofing due to multipath[18]. The $2.4GHz$ carrier frequency is susceptible to multi-path in environments where the transponder does not have line–of–sight to all the measurement stations. The problem of accurate localization becomes one of observability, which is out of the scope of this thesis. There several approaches that would help with mitigating effects of multipath such as positioning or adding measurement stations or implementation of Kalman filtering or estimation techniques[23, 24, 25, 26, 27, 28, 29]. By using the same correlation techniques explained at the beginning of this chapter and in section 2.3, multiple echos due to multipath can be detected and corrected for[19]. Continuing work after the basic goals of this project are met will include implementation and research into algorithms that exploit system state and constraints to mitigate the effects of multipath.

1.2 Motivation

Probably the most lucrative application for a passive localization system is asset, people, and/or animal tracking. The ability to locate products and possessions in real–time and 3-d will improve security and retrieval of items with significant improvements over existing technologies, such as RFID.

Encoding human body motion is an example of a dynamic application which illustrates the utility of 3-d tracking[30]. Traditionally body motion has been recorded with vision instrumented rooms[31], inertial measurement units (IMU)[32], or ultrasonic sensors[33]. A person wearing chip-sized SAW transponders sewn into the wrists, ankles, and joints of their clothing would have their body motion tracked in real-time. Such encoding could be used for a variety of human computer interfaces. Two of the many applications which this system is uniquely suited for include allowing unhindered natural motion in a virtual reality environment, training a robot through intuitive movements, or interactive computer games.

Because of their small size, low-cost, and the ability to use numerous tags, SAW transponders are well suited for disposable applications such as tracking animals, for research, or domestic control. An additional benefit of SAW transponders is that they do not requiring external power and their useful life is virtually unlimited and maintenance free.

1.3 Related Work

Existing technologies are inadequate for the applications a SAW localization system is suited for. Ultrasonic transponders are bulky, require a power source, and have limited line-of-sight and are susceptible to environmental interference. IMUs are bulky, costly, require a power source, and are limited by drift, bandwidth, and accuracy. A vision-based tracking system requires a carefully controlled environment to ensure line-of-sight and sufficient contrast between the obtrusive visual markers and background objects. Differential GPS systems are bulky, costly, mainly limited to outdoors, and have marginal update rates[34]. The key advantages of SAW based transponders is an unintrusive, robust, and low-cost tracking system with high update rates that can be deployed with thousand of tags. The applications of localizing SAW transponders is broad and poorly met by existing solutions.

Many commercial indoor location and tracking systems have been implemented or are in development. Chipcon has developed a hardware core that uses signal strength and bit–error–rate to estimate the relative location of transceivers [35, 34]. A system of using small active tags was developed by Carios for the use of tracking objects at high–speed, such as soccer balls. An asset tracking system was developed by Ubisense utilizing ultra–wide–band technology[36]. All of these systems have one thing in common, bulky active tags, a major disadvantage in comparison to the passive SAW transponders.

Meaningful quantitative comparisons between the mentioned technologies is difficult as there are many application specific trade-offs which affect performance. A few of the inherent trade-offs are between accuracy, power, cost, update rate, and effective range. For simplicity, many assumptions were made while compiling the data shown in figure 1.1[31]. The best performance as stated from the vendor's documentation sheet was assumed. Minimal external interference under the best conditions were

	uTags	Differential GPS	Vision	Ultrasonic	Radar
Tag Size	1x1 cm	16x9x17 cm	4x4 cm	5x5x5 cm	NA
Tag Weight	$< 2g$	$1.4kg$	$< 1g$	$\approx 100g$	NA
Cost/Tag	$\approx$ \$1.00	\$1k–\$10k	\$0.30	$\approx$ \$100.00	NA
Reader Cost	$\approx$ \$5k	NA	$\approx$ \$10k	NA	$\approx$ \$2k+
Resolution	$< 10cm$	$< 5cm(xy), < 10cm(z)$	$< 2cm$	$< 1c,$	$\approx meters$
Sample Rate	1–100kHz	1–20Hz	30–60Hz	$\approx 120Hz$	$\approx 1Hz$
Power/Tag	0	$5W$	0	$200mW$	0
Range	1–100m	NA	1–10m	1–10m	*miles*
ID	Yes	Yes	no	Yes	Yes
In/Out Doors	In/Out	In/Out	In	In/Out	Out

	Chipcon	Carios	Ubisense		
Tag Size	varies	2x2x0.5 cm	5.8x9.2x1 cm		
Tag Weight	varies	$5g$ est.	$45g$		
Cost/Tag	unknown	57 EUR	50 EUR		
Reader Cost	unknown	200k EUR	15k EUR		
Resolution	$1 - 2m$	$1 - 2cm$	$20cm$		
Sample Rate	*varies*	$100kHz$	$39Hz$		
Power/Tag	100mW	$70mW$–$175mW$	unknown		
Range	$70m$	$300m$	$50m$		
ID	Yes	Yes	Yes		
In/Out Doors	In/Out	In/Out	In/Out		

Table 1.1: Comparison between widely used localization systems and SAW transponder [6, 7, 8, 9].

implied on some data sheets, and nominal specifications were given by other data sheets. A comprehensive comparison is out of the scope of this thesis, however table 1.1 illustrates how SAW transponders are uniquely suited for a niche which existing technologies do not fill.

Design and fabrication of SAW filters and correlators has been extensively researched. The uses of SAW devices is expanding in the area of chemical sensing, optics, and RFID applications[13, 37, 38]. SAW devices are emerging in such novel uses as an optical wave-guide scanning system to steer a laser beam using the birefringent properties of $LiNbO_3$[39]. Shortly after this project was started companies producing SAW devices for RFID application started to appear[17, 40]. The market

for utilizing SAW RFID technology is relatively untapped, probably for several reason including the amount invested in other RFID technologies, recent developments in SAW designs, and the prior difficulty in manufacturing and reading SAW devices in comparison to other technologies. A search of the literature has not yielded any projects relating SAW transponders to ranging and localization similar to the goals and methods of this project.

Chapter 2

Surface Acoustic Wave Devices

This chapter covers the basic concept of SAW functionality, the approach taken to model these devices for this project, and an explanation of the device designs implemented on masks 1 through 3. The approach taken to tackle the complexity of this project was to iterate on design, modeling, fabrication, and testing of SAW devices. Simple models that were used in the beginning will become more sophisticated and accurate with each iteration. Using real test data at each step will verify our increasingly sophisticated models and designs, as well as progressively build experience with fabrication, electronics, and design. Existing methods for encoding data into the SAW transponders will also be experimented with, but are not the primary focus of this project.

Acoustic surface waves are a physical, as opposed to electromagnetic, compression disturbance moving along the surface of a substrate. Visualizing waves on the surface of a SAW device as ripples on a pond can be helpful with understanding the transient operation of these devices. Surface waves can be initiated in many ways depending on the substrate. The SAW devices in this project consists of a piezoelectric substrate patterned with a thin film of aluminum. When electrical signals are applied to the

metal on surface of the piezoelectric substrate, the electric field causes a physical disturbance. A cyclical electrical field causes ripples to be generated. To magnify the response, many input features are actuated in resonance for constructive interference of the waves[5].

The SAW transponders in this project consists of a piezoelectric substrate patterned with a thin film of aluminum. Any type of conductive metal can be used to make the IDT, however there is a trade–off between adhesion, resistance, and weight. Some metals have poor adhesion to $LiNbO_3$. Even though they are very thin ($\approx 200nm$), the weight of the metal fingers causes signal attenuation and reflections. Aluminum was chosen for reasonable adhesion to $LiNbO_3$, low mass, and low resistivity.

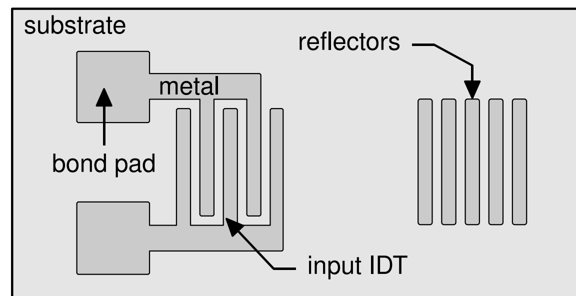


Figure 2-1: Simplified diagram of basic SAW transponder functional components[1].

A notational diagram of the patterning used for a simplified SAW transponder is shown in figure 2-1. An antenna is connected to the bond–pads which couples the RF chirp into the transponder. The inter–digital transducer (IDT) is a linear array of metal fingers which change incident electrical energy from the bond–pads into a surface acoustic wave on the piezoelectric substrate. This wave travels along at a slow velocity (in comparison to RF) until reflecting off discontinuities on the path of travel such as metallic strips of reflectors. The reflected signals return to the IDT and are retransmitted via the antenna to the base-station. The gap between the IDT and the

first reflector builds in a delay between the incident RF burst to the first reflector, which allows multi-path clutter to diminish before a rebroadcast, ensuring the base station receives a clear signal from the SAW transponder [13, 5, 14].

Anytime the surface wave encounters a discontinuity, three things may happen. Some of the energy in the wave may be reflected, continue on unaffected, or be dissipated. Any pattern of material may cause a disturbance or transformation of surface waves traveling underneath. Every finger in an IDT or reflector will reflect some of an incident wave. These reflected waves will then interact with other fingers to repeat the process until all of the energy has been dissipated. Secondary reflections can be a significant problem and pose a design challenge by ruining the desired response, or they may be exploited as part of the design.

Clever designs of SAW patterns allows sophisticated signal processing of the surface wave. Proper design of the patterns will even cause the substrate to steer light[39]. As well as being periodically resonant, SAW patterns are also designed to resonate to a specific sequence of patterns, as in the case of SAW correlators[5, 14].

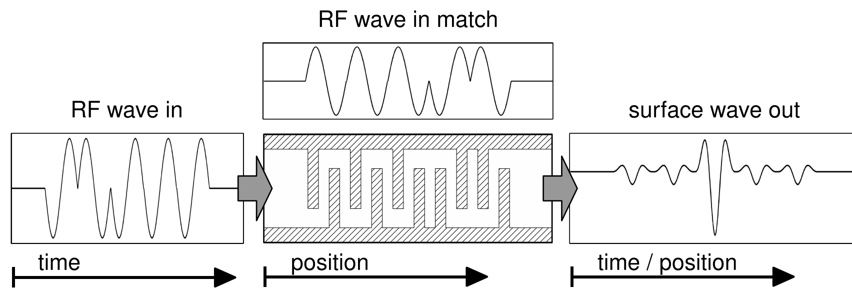


Figure 2-2: Saw correlator with corresponding code and output pulse waveform. *"time / position"* is the waveform over time at a single point between IDTs, or the waveform over position between IDTs at a particular time.

The design of phase–shift SAW correlators involves alternating the arrangement of the IDT finger pairs. Figure 2-2 shows a SAW correlator IDT with the respective matching waveform. As the waveform enters the IDT, at each cycle each finger

pair starts a wave of a certain polarity. These waves constructively or destructively interact. With the proper input waveform, the total sum of the waves interfering with each other will give the maximum response, which is launched along the surface towards another IDT or reflector. The "RF wave in match" in figure 2-2 illustrates how the input waveform matches to the IDT's finger pair polarity. In this example the correlated sequence is a Barker code which results in a pulse surface wave with side lobes of unity magnitude[14, 5]. Design and simulation of SAW correlators will be covered in is sections 2.2 & 2.3.

2.1 Properties of Lithium Niobate

The design of the SAW devices in this project is limited by the properties of $LiNbO_3$. Lithium niobate was chosen for several reasons. The high coupling–coefficient, or the amount of electrical energy transfered to the surface wave, of $LiNbO_3$ is much better than many other suitable materials[5, 2]. $LiNbO_3$ also has a relatively high characteristic surface wave velocity. Faster surface velocities enables higher frequency devices to have larger metal patterns, making fabrication easier.

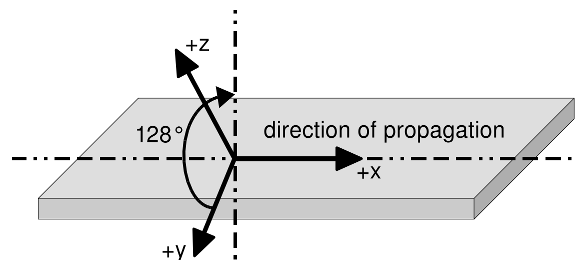


Figure 2-3: Illustration of the crystal cut of a 128°–RY wafer[2], with an X–Flat, from Crystal Technologies Inc.

The crystalline structures of $LiNbO_3$, constrains the surface waves to move along a particular direction. Varying the orientation of the crystalline structure with respect

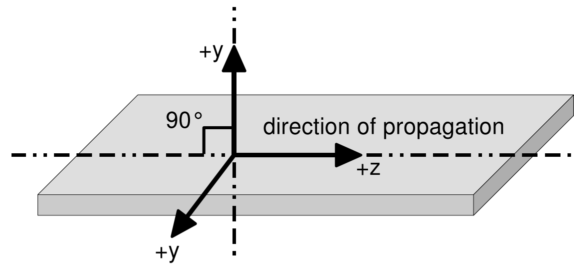


Figure 2-4: Illustration of the crystal cut of a Y wafer [2], with a Z–Flat, from Crystal Technologies Inc.

to the surface of the wafer changes the properties of substrate. Some of the properties affected by the crystalline orientation are the surface velocity, coupling coefficient, and temperature dependency [14, 41, 2]. The 128°–RY lithium niobate wafer cut has the best coupling coefficient ($K^2 = 5.3\%$) and surface velocity ($3992 m/s$) with a decent temperature degradation coefficient ($75 ppm/$℃). All of the devices discussed in this thesis were made with 128°–RY wafers, though devices were also fabricated out of Y–cut wafers.

The effects of temperature on the surface velocity of $LiNbO_3$ SAW devices has been well characterized[5]. Changes in temperature cause a shift in the band–pass SAW center frequencies of a SAW filter[14, 5]. Most implementations appear to safely ignore temperature effects. In cases where the surface velocity is critical, some designs use a heater to control temperature[14]. Some transducers exploit the temperature dependence of lithium niobate to measure temperature[21]. For this project, subtracting out the effects of temperature on the ranging solution will be important.

2.2 Modeling

Accurately modeling a SAW device's response to stimulus is very complicated. Higher order finger reflections are usually modeled using finite–element–analysis (FEA) or

boundary–element–analysis (BEA). Several methods of modeling finger pairs in FEA or BEA exist, such the P-Matrix, S-Matrix, coupling–of–modes (COM) modeling methods[5]. FEA and BEA are both computational intensive, time consuming, and not particularly intuitive. Intuition into the operation of SAW devices is essential for understanding the effects of design choices. The design of more complicated implementations lead to longer simulations, making visualizing SAW responses even more important.

A simple method of simulation was explored to quickly obtain some experience with the macro characteristics of SAW behavior. The approach taken estimates the loss in power resulting from first–order finger reflections given the initial acoustic waveform. The resulting surface waves are then convolved with the output IDT to simulate the electrical output generated by the surface waves.

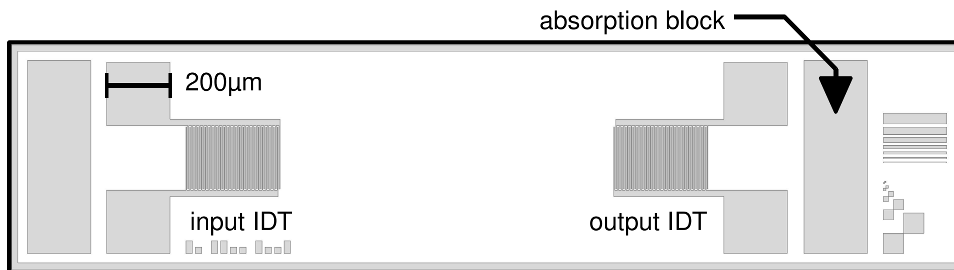


Figure 2-5: SAW filter made with mask 1. Device `0x2C9`, 25 finger pairs that are 200μm long.

A simple example illustrating simplified modeling is shown in figures 2-6. This is the simulated impulse response through a SAW filter as designed and implemented on mask 1, which is shown in figure 2-5. The surface wave response is modeled by a sine wave in which each finger pair corresponds to one cycle. The magnitude of the surface wave, at any given point between the two IDTs, decreases with time because of reflection caused by the input IDT fingers. Another way to visualize this is that at the moment the impulse couples into the IDT there is a surface wave generated

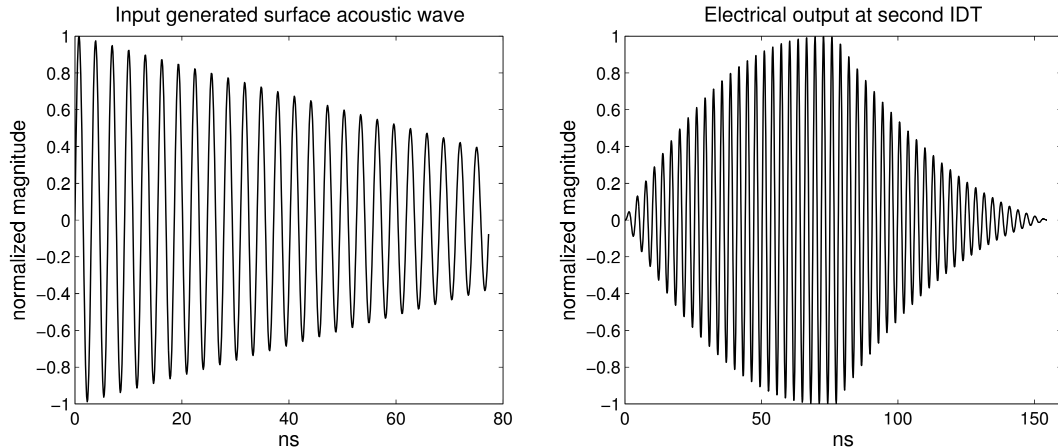


Figure 2-6: MATLAB™ simulation of the normalized impulse response of a $323MHz$ SAW band–pass filter shown if figure 2-5.

that spans the IDT with no attenuation of signal. This surface wave travels out from each IDT finger pair in both directions, and anytime an individual cycle encounters a finger some of the energy is lost due to reflection and absorption. As the surface wave couples into the output IDT, further attenuation occurs due to reflections. By convolving the two waveforms resulting from the geometry of the input and output IDTs, the electrical signal at the output can be estimated as shown to the right in figure 2-6. The code used to generate this simulation is given in MATLAB™ appendix F.

An oscilloscope trace from the actual device corresponding to this simulation is shown in figure 2-7. The test setup and data collection procedures are detailed in section 4.4. The first 25 cycles correspond very closely to the simulation, because only first order effects of finger reflections are dominant. As the surface wave moves further across the output IDT the secondary reflection start to have a visible impact, which is why the tail end of the test data deviates from the simulated output. Reflections will continue to bounce back and forth between the two IDTs, and internal IDT fingers, until all the surface waves have dissipated. This effect, shown in section 5.1, is also

MATLAB 1 : MATLAB™ source for simulation in figure 2-6

```
1  % Simple simulation of the impulse
2  % of a response 50 fingers, or 25
3  % finger pairs, saw filter
4
5  f = 323e6;      % frequency = 323 MHz
6  w = 2*pi*f;     % omega
7  p = 1/f;        % period
8  fingerPs = 25  % number of finger pairs
9
10 % time step is 100ps
11 t = 0:100e-12:fingerPs*p;
12
13 % model input pulse
14 in_p = (1-t*f/40).*sin(w*t);
15
16 % normalize
17 in_p = in_p./max(in_p);
18
19 % model output  pulse
20 out_p = conv(in_p, in_p);
21
22 % normalize
23 out_p = out_p./max(max(out_p),-min(out_p));
```

not covered by this model. The purpose of the higher order models is to take into account multiple finger reflections.

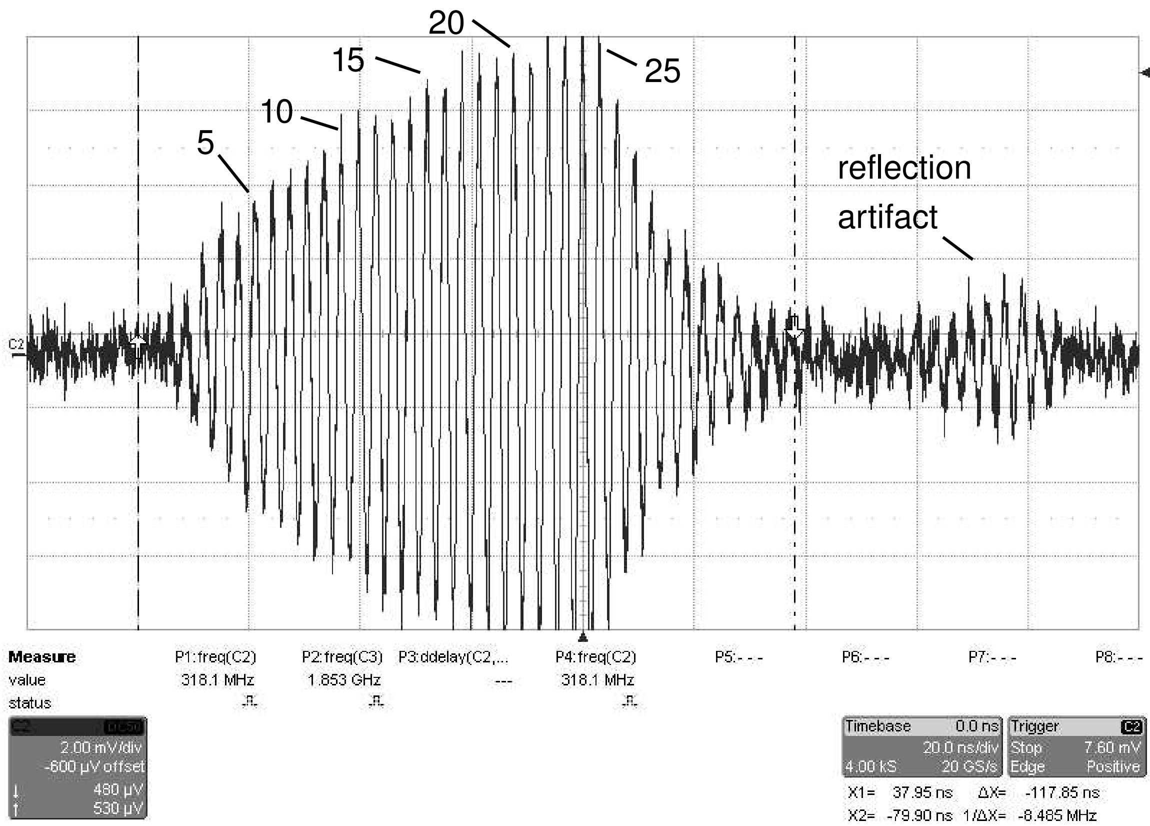


Figure 2-7: Result of an impulse through device `0x2C9` as recorded on an oscilloscope.

2.3 Design

Only a simplified number of design parameters were specified for the first devices. The center frequency of the mask 1 designs was primary specification. The complete frequency response has been ignored for simplicity.

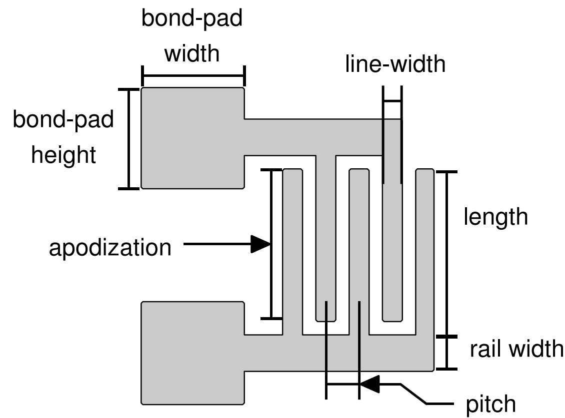


Figure 2-8: Parameters for the design of a SAW IDT in this project.

Figure 2-8 summarizes the basic parameters for the devices on mask 1. The finger pitch along with the characteristic surface velocity determines the resonant frequency of the IDT.

$$\begin{aligned} f &= \frac{v}{\lambda} \\ &= \frac{v}{2 \cdot p} \end{aligned} \tag{2.1}$$

where v is the characteristic surface wave velocity, λ is the surface wave–length, and f is the center frequency[5].

For the devices with $3\mu m$ line–widths, and a pitch of $6\mu m$ on a 128°–RY crystal cut with a surface velocity of $3992m/s$, the center frequency is $332MHz$. This differs from the measured center frequency response of $323MHz$, which may be due to secondary effects such as electromagnetic feedthrough, triple–transit–interference, capacitive effects of the fingers, or the RF-Network Vector Analyzer that we were

using was out of calibration (the calibration was last done over ten years ago). More investigation into the discrepancy is underway.

Apodization, or the amount of finger overlap, may be varied to evoke a certain response from a SAW filter. For example a sinc–function apodization of the IDT will yield a flat pass–band response. The simple filter devices on mask 1 had constant apodization.

The number of finger pairs and length of the apodization determines the input capacitance of the IDT. Capacitance per unit length is a fundamental property of the particular substrate and crystal cut used. For 128°–RY cut lithium niobate, the capacitance per finger pair per unit length is $5pF/cm/\textit{finger–pair}$. Therefore:

$$C_t = c_f \cdot 100(cm/m) \cdot a_p \cdot f_p \tag{2.2}$$

Where C_t is the total IDT capacitance, c_f is the fundamental capacitance per finger pair per unit length, a_p is the apodization in meters, and f_p is the number of finger pairs. For device $0x2C9$ detailed in section 5.1, $a_p = 200\mu m$, $c_f = 5pF/cm/pairs$, and $f_p = 25$ resulting in a C_t of $2.39pF$, which is close to the measured input capacitance of $2.13pF$.

The bond pads were made large enough for easy wire–bonding. They were much larger than necessary in an attempt to mitigate the problems caused by the thin metal and fragile substrate. The width of the rails only needs to be wide enough to ensure the signal is not attenuated substantially due to their internal resistance. An absorption block, see figure 2-5, stops unwanted signals from reflecting back towards the IDT. The $200\mu m$ width chosen was arbitrary for this design. Future designs will use angled reflectors that won't take up as much device area[14].

Three different reflectors for encoding of the return signature were tried on mask 1, shown in figure 2-9. The "open" reflector had the largest return response of the

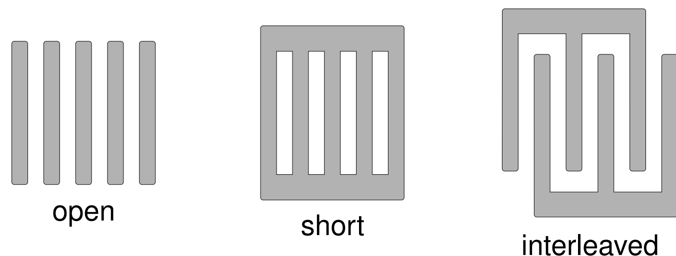


Figure 2-9: The three types of SAW reflectors were implemented on mask 1.

three designs and was the only design to return a significant signal. Not much time was taken to characterize the different reflectors for lack of equipment and because fabricating the 915MHz devices became the priority.

SAW correlators are extensively used in SAW filter and communications applications [11, 12, 13, 5]. Implementation of SAW correlators for encoding was one of the goals of mask 2 & 3. Several 915MHz narrow–band and wide–band correlator designs were made, with three of them explained here. To keep this stage of the project as uncomplicated as possible, secondary effects, such as reflections, have been ignored in the following examples and corresponding simulations. Secondary effects will need to be account for in future designs. The Matlab™ scripts used to generate the simulation for the following are located in appending F.

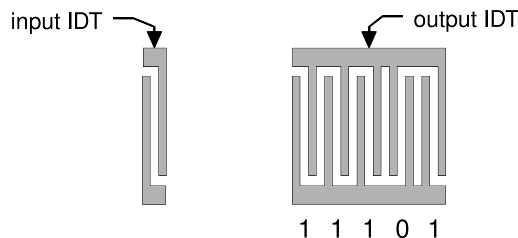


Figure 2-10: Narrow–band output correlator with $n = 5$ Barker code illustration.

A narrow–band correlator consists of a tightly packed set of IDT fingers for a continuous code. To encode '0' or '1', the phase of the finger is reversed from one to

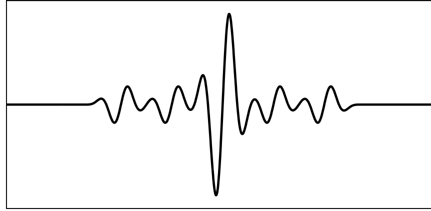


Figure 2-11: Narrow–band output trace for figure 2-10.

another (see the digits under the finger pairs of output IDT in figure 2-10). When the proper code couples into the input IDT, the resulting SAW waveform will create a peaked waveform at the output. Multiple finger–pairs may be used to implement the encoding which increases the total amount of energy in the signals and increases signal peaking. A simplified narrow–band SAW configuration is shown in figure 2-10 with a Barker code of length five encoding. The proper code received by the input IDT will result in the signal shown in figure 2-11 at the output. Notice the continuous nature of the output waveform.

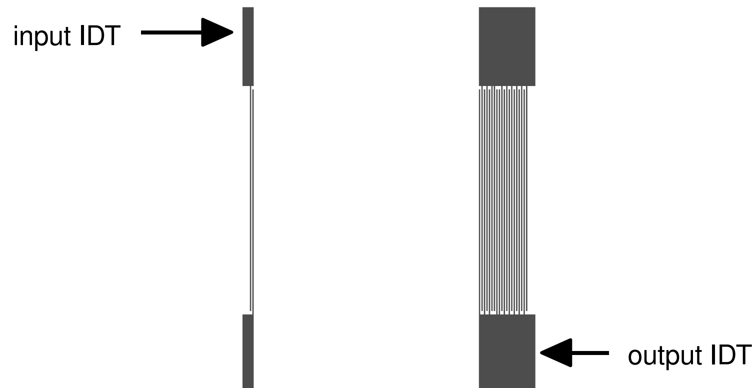


Figure 2-12: Narrow–band output correlator with code $[1, 1, 1, 0, 1, 1, 1, 1, 1, 1]$ utilizing one finger–pair encoding.

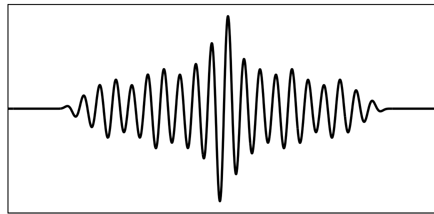


Figure 2-13: Narrow–band correlator response for device fabricated on mask 3, shown in figure 2-12. Generated with `nb.m` in appendix F.

Figure 2-12 shows a design for a narrow–band SAW correlator on mask 1. This devices has a code that is different from the Barker sets of codes, resulting in a unique signature to the optimal interrogation code. The expected output is shown in figure 2-13.

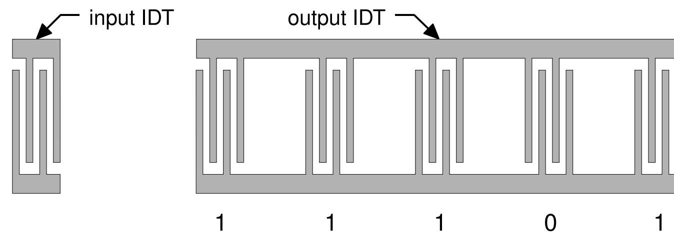


Figure 2-14: Wide–band output correlator illustration with $n = 5$ Barker code with two finger–pair encoding.

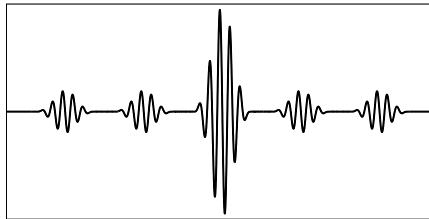


Figure 2-15: Wide–band output trace for figure 2-14.

A wide–band correlator differs from a narrow–band correlator by interspersing the phase encoded signal with a delay based on the number of fingers at the input. When the set of delayed pulses matches the output code the output waveform peaks, as shown in figure 2-14.

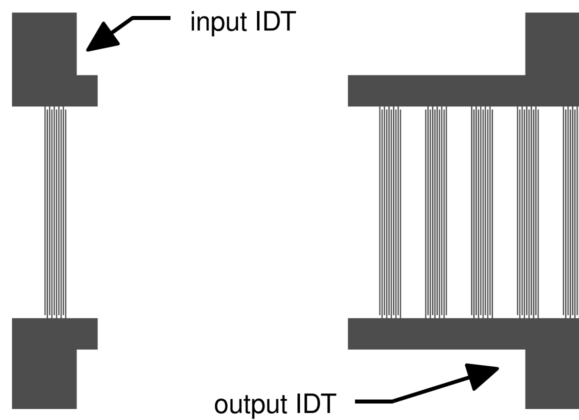


Figure 2-16: Wide–band output correlator.

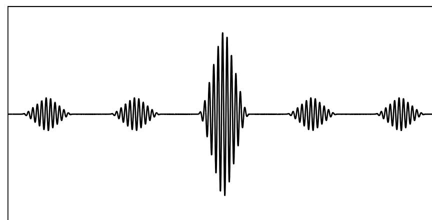


Figure 2-17: Wide–band correlator response for device fabricated on mask 3, shown in figure 2-16. Generated with `wb.m` in appendix F.

Figure 2-16 shows the layout of a wide–band SAW correlator on mask 3. The five finger input does not have an encoding. The output encoding is implemented in the output IDT. A Barker code of length five was used, which with the correct input sequence gives the output waveform in figure 2-17.

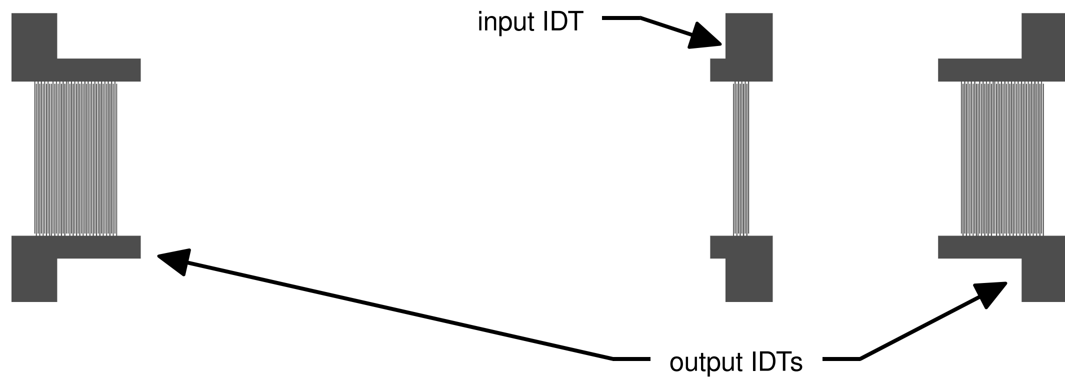


Figure 2-18: Dual narrow–band transponder with a Barker code of length five with a five finger encoding.

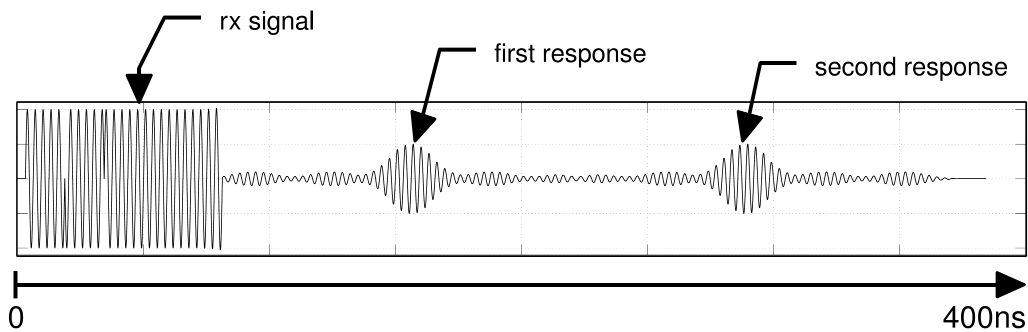


Figure 2-19: Dual narrow–band transponder's simulated waveform. The *"rx signal"* was transmitted by the measurement station and is what the input IDT sees. The *"first response"* and *"second response"* are the waveforms from each output IDT.

Measuring and compensating for temperature will be tried with the design in figure 2-18. This dual narrow–band transponder has three IDTs which will give two distinct responses as explained by the example in chapter 1. An actual implementation would have all three IDTs connected to the same antenna. They were left disconnected in this design to allow independent testing of each port. Figure 2-19 shows the expected waveform when the correct code sequence is received by the input IDT and probed at the output IDTs. The phase relationship and the distance between the two responses

will be proportional to the temperature of the substrate.

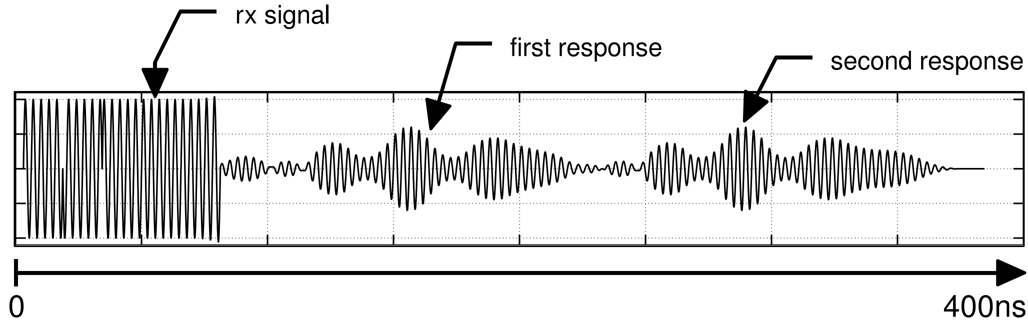


Figure 2-20: Simulation including the effect of surface waves originating from the output IDTs of the transponder shown in figure 2-18.

Figure 2-19 does not contain the effects of the initial input signal coupling into the output IDTs in the configuration where all IDTs are connected to the same source / antenna. When the input IDTs and output IDTs are connected to a common antenna (typical for a self–contained transponder), the output IDTs also generate a signal from the output to the input IDT. This signal combine with the signals originating from the input IDT to create the slightly distorted waveform shown in figure 2-20. In the future, designs we will exploit the ability to use all three IDTs as input and output transducers, to both increase the returned power and enhance encoding and temperature compensation.

Chapter 3

Micro–Fabrication

Fabricating devices with feature sizes in the rage of micrometers to nanometers requires specialized sciences, engineering, and a significant degree of art. High–resolution processing is carried out in a clean–room facility with a variety of chemical processing and meteorology tools. The path generally taken to constructing nano–structures as created in this project starts with design and layout of artwork describing the desired features with a CAD tool. A scanning–electron–beam–lithography (SEBL) tool then writes this CAD artwork into a photoresist on a wafer. The wafer may be the final substrate or an optical mask used to duplicate the artwork onto the desired substrate. The photoresist is then developed with a chemical solution to remove areas of photoresist. The substrate now contains a transfered pattern of exposed material onto which more material (i.e. metal, oxide) may be deposited or removed. After the deposition or etching step, the remaining photoresist is stripped from the wafer leaving the desired features. When the devices are completed, they are cut up into small rectangles, mounted to packages, and electrical leads are wire–bonded from pins on the package to pads on a device. Also note that nanofabrication may be carried out with processes other than lithography, such as nanoimprint and

atomic force patterning technologies, which use physical means to transfer a pattern [42, 43].

There are many emerging methods and applications of nanotechnology being developed on a variety of novel custom fabrication tools at MIT. The use of shared experimental fabrication tools, as opposed to commercial production tools, made developing a processes much more difficult. Many of the tools required a significant amount of experimentation to yield the desired results. Adding to existing challenges, some of the tools would behave differently after other lab users processed with them. Many factors and a array of processing variables made fabricating working devices a daunting challenge. As with most research projects, costs vary widely and are difficult to project ahead of time without significant experience. For this type of work a typical budget runs between $7000 and $8000 USD per semester. This project was my first exposure to micro–fabrication, and I have learned an enormous amount about the field. Topics in this chapter aim to explain all the challenging problems and their solutions, as well as all the process details that went into fabricating these devices necessary to reproduce this work.

The devices in this project were fabricated utilizing the resources of several MIT micro–fabrication facilities. The Nanostructures Laboratory (NSL), directed by Professor Henry Smith and Professor Karl Berggren, provided the high–resolution nano–lithography tools necessary to created the $915MHz$ and $2.4GHz$ transponders. The Research Laboratory of Electronics (RLE) provided the shared scanning–electron–beam–lithography facility (SEBL at RLE) to write the high–resolution lithography masks. The Microsystems Technologies Laboratory (MTL) provided the Experimental Micro–fabrication Laboratory (EML), where the $3\mu m$ devices were fabricated.

Fabrication started in the EML because of the ease of access and reduced cost while meeting our initial resolution requirements. A wide variety of experimental

micro–fabrication projects are supported by EML, where the research of over a hundred students is underway. EML is an excellent laboratory to inexpensively develop processes that don't require a high-degree of cleanliness and have feature sizes greater than $\approx 2\mu m$. Another advantage of EML is that more latitude is allowed for new processing methods and materials. The tools in EML utilized by this project were an electron–beam vapor deposition tool, the solvent and acid hoods, a resist coater, a profilometer, microscopes, and a mask aligning and exposure tool.

NSL is a community of researchers and students who emphasize collaboration and mentoring, which were invaluable to the progress of this project. The Nanostructures Laboratory is a cleaner cleanroom that provides the specialized lithography tools needed to achieve $1.1\mu m$ and $300nm$ feature sizes. A cleaner laboratory was critical for conformable vacuum–mask lithography, covered in detail in section 3.4. Conformable vacuum mask lithography was a more cost effective method of making many duplicate devices with line–widths as small as $300nm$[42]. A plasma asher and ellipsometer were also utilized in addition to all of the same tools used in EML.

Devices with line–widths of $300nm$ were required for operation in the $2.4GHz$ Instrumentation–Scientific–and–Medical (ISM[1]) frequency band. As a starting point, $323MHz$ devices with $3\mu m$ line–widths were made first for two reasons. Firstly, $3\mu m$ was the minimum resolution reasonably achievable using tools in EML. $3\mu m$ mask fabrication is also considerable less difficult and less expensive than $300nm$ masks fabrication. Many more devices may be patterned on a single $3\mu m$ mask at no extra cost. Experienced gleaned in learning the $3\mu m$ fabrication and testing processes directly transfer to the more difficult fabrication and testing processes of the $1.1\mu m$ and $300nm$.

Wafers were handled from tool to tool with several different types of specialized

[1]see FCC regulation 18.301

tweeters. The fragile wafers required a certain degree of skill to handle without breaking them. Wafer were stored in individual and 25–wafer plastic carriers. Die were put into sticky–gel containers[2].

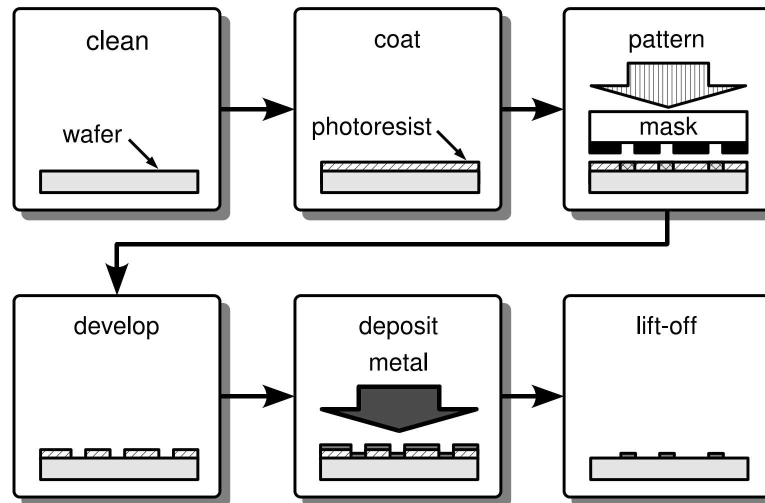


Figure 3-1: Simplified one–step lift–off process.

The SAW devices were made using a typical micro-fabrication one-step lift-off process. Refer to figure 3-1. A wafer coated with a photoresist is patterned with a mask and then developed to remove the exposed positive photoresist, covered in section 3.4.1. Then a thin layer of metal is deposited over the whole wafer using electron–beam vapor deposition. The wafer is then immersed in a solution to remove the photoresist and metal overlay from the surface. The remaining metal pattern forms the desired features.

A chemical wet-etch is another standard fabrication process that was used to make masks. In the wet etch process that was implemented, a layer of metal was deposited on the wafer before the photoresist. See figure 3-2. The photoresist is then patterned and developed in the same manner as in the lift-off processes. The wafer is then submerged in a liquid chemical etchant to remove the exposed metal. The wet-etch

[2]Wafer containers were purchased from MTI corporation `http://www.mtixtl.com/`

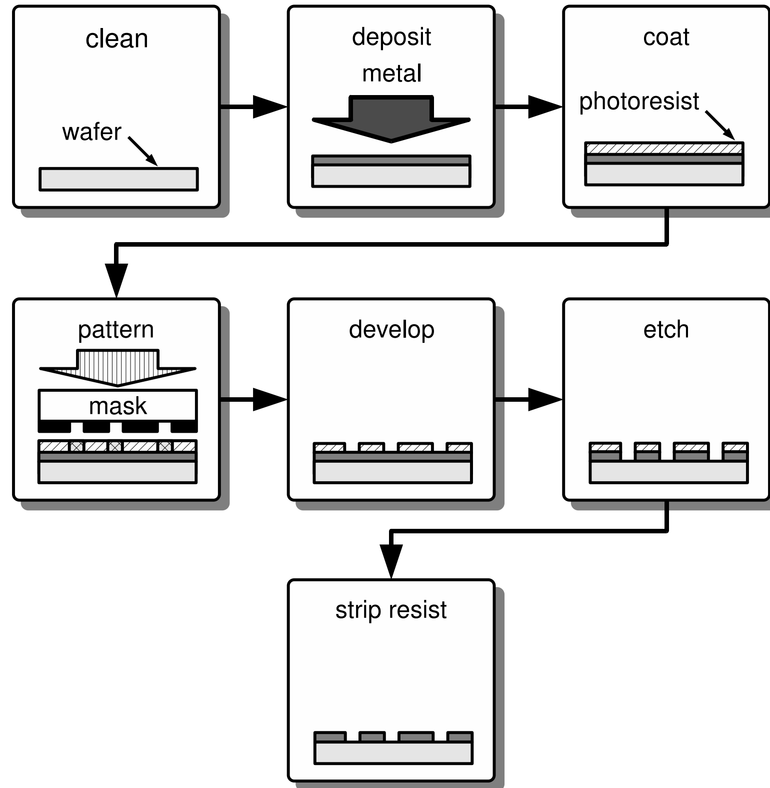


Figure 3-2: Simplified wet–etch process.

step, in some cases, can be much easier to implement than a lift-off depending on various factors and requirements.

To create working devices from a new design "always" requires iteration of the design, fabrication, and testing/evaluation tasks. At best, development of a fabrication process is a difficult and tedious task, much of which is skillful trial and error. A process designer not only relies on a vast collection of knowledge resources, but also ingenuity is essential for success. To design a fabrication process, one must choose values for a dizzying list of parameters that often do not have precisely determined specifications. Process parameters vary greatly with luck, time, temperature, humidity, and the chemicals used. Even minute particulate contamination will ruin a device or process step; thus success depends on cleanliness. The challenges of constructing

small structures is difficult not only because of the inherent physical limitations, but the laboratory skill required by a student to implement a process. The substrate used for SAW devices, $LiNbO_3$, caused many additional problems. $LiNbO_3$ is particularly fragile and inherent piezoelectric properties caused degradation of masks and adhesion problems. The challenges to developing a robust process with repeatable results yielding reliable devices are significant.

3.1 Mask Design and Layout

The lithography process often makes use of a *mask*, a piece of glass patterned with metal defining the structure of the devices[42, 43]. The metal blocks the light causing only illuminated parts of the photoresist to be exposed. Different mask substrates are commonly used such as Soda-lime (commonly called glass), which is amorphous SiO_2 with various additives. Pure, or nearly pure, SiO_2 is also used as a mask substrate[3]. The proper mask substrate must be chosen to match the wavelength of light emitted by aligner source. For example, soda–lime blocks almost all light with wavelengths less than 300nm and thus can not be used for $220nm$ deep–UV wavelengths. Pure SiO_2, which will transmit nearly all light with wavelengths above $200nm$[4], is used for deep–UV lithography.

The first mask layout was drawn using the open-source software package *MAGIC*[5]. The device geometries and parameters were chosen based on what I thought would be reasonably easy to design and fabricate given a broad survey of the literature in the field of SAW devices. The goal was to make as many different devices as possible

[3] SiO_2 masks are often referred to by the semiconductor industry as "Quartz masks." However, these substrates are not crystalline SiO_2 "Quartz," which can be confusing because crystalline SiO_2 "Quartz" is used in SAW device fabrication.

[4] Further information may be found at `http://www.nanofilm.com/SubstateMaterial.PDF`.

[5] MAGIC was originally developed by the University of California at Berkley. Additional information can be found at `http://opencircuitdesign.com/magic/`.

because the number of devices does not effect the cost of the mask, By automating the drawing of these devices using MAGIC's built–in TCL language[6] 773 different designs with various parameters where able to be put on a single mask. The TCL scripts used to generate the mask 1 artwork are located in appendix A.1. Simplicity was the driving factor of our first designs because I had no experience with the analysis, fabrication, or testing of SAW devices.

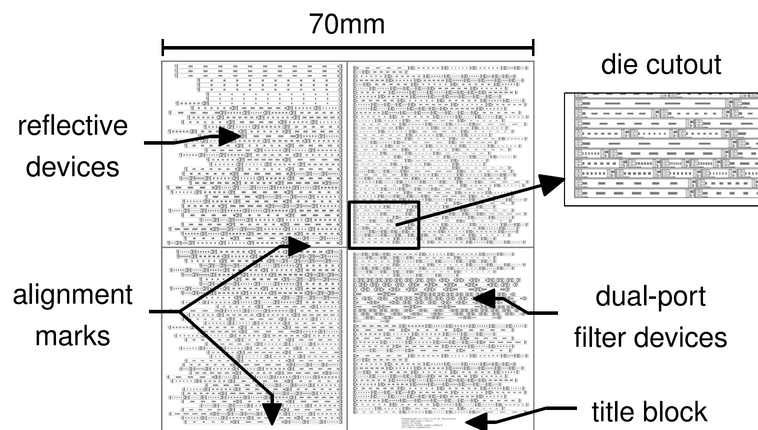


Figure 3-3: Artwork for mask 1 showing a die section.

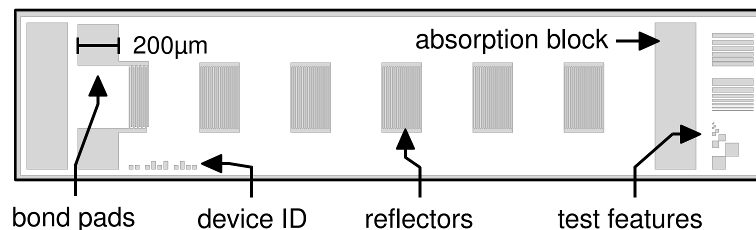


Figure 3-4: Typical layout of a device on Mask 1.

The artwork for the final mask is shown in figure 3-3. Each device includes two absorption blocks, which attenuate surface wave reflections, and a set of test structures, which are used to determine exposure and developing parameters. The resolution was

[6]TCL is a versatile open–source scripting language often chosen for simplicity.

limited by how well the mask would contact the substrate using the EML aligner. The hard–contact mechanism on the EML aligner would, with some major difficulty, achieve contact sufficient for $1 - 3\mu m$ line–widths. Mask 1 was made of a $100mm$ square, $2.29mm$ thick, soda-lime plate with a chrome coating. This mask would fit in the EML aligner, cover a $77mm$ $LiNbO_3$ wafer, and be compatible with the $\lambda = 365nm$ light source. The mask artwork filled a $70x70mm$ square at the center of the mask. A defect criteria was specified to the vendor as "zero defects greater than $5\mu m$ and less than one defect per square inch." Tolerances of $\pm 500nm$ were specified for line–widths and geometries. These specifications were decided though discussions with the mask vendor[7] and based primarily on our budget and the desired geometry of our devices. There are a variety of file formats supported by CAD tools and accepted by mask vendors. This design was submitted in a GDS2 file format and the order was placed March $15^{th}, 2006$ and cost $575.00.

A lot was learned though the problems encountered by using this first mask. Alignment of the devices to the crystal orientation of the $LiNbO_3$ wafer was problematic. The alignment flat on the wafer must be oriented to the devices using a microscope built into the aligner. To do the alignment, the substrate wafer needed to be shifted to the edge of the substrate chuck, while the mask was shifted to the limits of the microscope's xy–travel. The result of this shifting was that many of the devices on the side towards the wafer flat were unusable. In fact, all of the devices within a half inch of the wafer's edge were unusable due to downstream tool limitations and handling. Another problem was that the devices were not arranged into small squares that could be cut up into $\approx 5x5mm$ die, the size which fits the test packages. In hindsight a less expensive $77mm$ mask with fewer devices and proper layout would have been equally as productive as this $100mm$ mask. The original reason for elim-

[7] The masks were purchased from Advance Reproductions Corporation, 100 Flagship Drive, North Andover, Massachusetts 01845, `http://www.advancerepro.com`.

inating the area required to allow for partitioning the mask into die was that more room would be left for additional devices. Unfortunately, finding the desired devices and setting up the die–saw to cut out the correct area was very problematic.

The second series of masks, none of which were successfully made, incorporated both higher frequency and phase–shift encoded devices. See figure 3-5. The designs on these masks had line–widths of $300nm$ and $1.1\mu m$, which yield a center frequency of $915MHz$ and $2.4GHz$. See section 2.3. Fabrication of these devices proved to be very difficult and time consuming because of their small features.

These designs where drawn using another open-source software tool, *layout*[8]. *layout* was much easier to use than *MAGIC* and also provided a useful macro language. Macros were created to automate generation of devices using both *layout*'s built in macro language and PERL[9]. The macro source code is provided in appendix A.2.

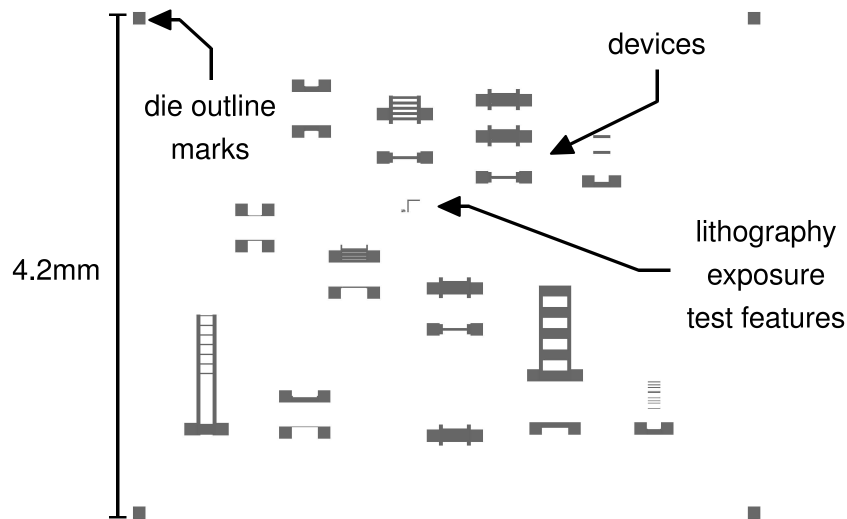


Figure 3-5: Artwork for mask 2.

This mask was designed with a bond pad layer, and a device layer. The device

[8]http://layout.sourceforge.net

[9]Practical Extraction and Reporting Language. A very popular, well established, open–source, and full featured scripting language.

1. export from layout in a single sell as a `.cif` file. Ensure layer names are less than four characters in length.
2. Remove duplicate layers with perl script.
3. run `ciftokic`. output is the name of the first layer.
4. Adjust number of kic units per μm in nanowriter .

Table 3.1: Steps to convert a file to `.kic` for use on the VS-26.

line–widths were small enough that they had to be written with the high–resolution VS-26 Scanning Electron–Beam Lithography (SEBL) tool. Higher–resolution requires lower power and a narrower electron beam, which causes the write time increase. By placing the bond–pads on a separate layer, they could be written much faster at a lower resolution, saving time and money. To ensure the bond–pads were connected to the devices between the two writes, as precise alignment can be a problem, some overlap between the layers was added to the design.

Two methods of fabricating these masks were tried. Lift–off was used with marginal results, discussed in detail later in this chapter. The complete process is detailed in Appendix ?? and ??. In summary, $\approx 120nm$ of PMMA[10] was deposited on a clean quartz wafer. A thin film of $7nm$ of Chromium was then evaporated onto the surface of the PMMA to shield the magnetic lens from electric fields during the SEBL write. See section 3.4.5. The artwork as then written to the wafer using a VS-26[11] SEBL tool. Alignment of the artwork to the wafer flat was important to ensure the devices would be aligned to the lithium–niobate crystal structure. The Chromium was then etched off and the PMMA developed using a dilution of one part MIBK[12] to two parts Isopropyl alcohol, followed by a descum to remove resid-

[10]PMMA is a very high–resolution positive photoresist from MicroChem Inc. `www.microchem.com`
[11]Operated by Mark Mondol at the Research Laboratory for Electronics at MIT.
[12]MIBK(methyl isobutyl ketone) or (4-Methyl 2-Pentanone) available from AlfaAesar PN#A11618

uals. $40nm$ of Chromium was then deposited by evaporation to make the actual mask features[44, 45]. The $40nm$ thickness of was chosen for the $17dB$ of attenuation provided at $220nm$ deep–UV light, which is sufficient for exposure of PMMA photoresist[44]. Unwanted Chromium was then removed via lift–off by striping the remaining PMMA with NMP[13], finishing the process and leaving the desired mask.

A wet–etch mask making process simplifies the number of fabrication steps. Masks made by wet–etch using SEBL were dark–field masks, and could be used to directly pattern to the $LiNbO_3$ substrate using the positive photoresist PMMA. First the mask was coated with $40nm$ of chrome. An $80nm$ of PMMA photoresist was deposited on top of the chrome. Only a thin layer of PMMA was necessary to mask the chrome from the etchant. The mask was submerged in the etchant after the mask was written and developed. More details on etching are provided in section 3.7.

Task	**Time**	**Cost**
Coat & Bake	$1hr$	\$80.00
Evaporate Cr	$30min$	\$40.00
Write Pattern (SEBL)	$2hr$	\$160.00
Strip Cr & Develop	$30min$	\$40.00
Evaporate Cr	$30min$	\$40.00
Lift–off	$30min$	\$40.00
Total	$5hr$	\$400.00

Table 3.2: Processing costs and time estimates to process one mask with lift–off. This does not include meteorology.

The reason we made these masks in–house was two fold. Commercially made masks with line–widths smaller than $1\mu m$ would have been very expensive, in the $\$1,500.00$ range. Also, by making the lithography masks at the NSL we were able to afford flexibility in testing different transponder designs more economically. The unprocessed clean quartz mask wafers cost $\approx \$50.00/ea.$. The bulk of the cost goes

[13]NMP (1-methyl 2-pyrrolidinone)

into processing the mask wafer to pattern the devices.

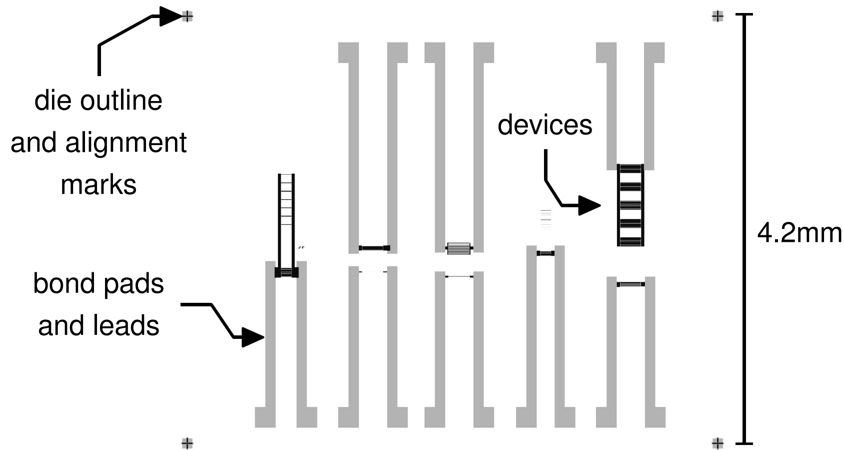


Figure 3-6: Artwork for mask 3 version 2. Bond–pads and leads are on a different layer than the devices.

The third mask was designed to eliminate the bond–pads and $100\mu m$ die boundary marking squares to save valuable and expensive SEBL machine time. Instead the large bond–pads would be added later in a separate step using a cheaply printed transparency film[14]. An added advantage is that the bond–pads will have a thicker layer of metal deposited making them more robust.

Mask 3 version 3, shown in figure 3-7 was written with a different SEBL tool, the Raith–150. The Raith–150 SEBL tool has an interferometric stage with a large range of motion, allowing larger areas to be written with little degradation in resolution. A large write area allowed separating the individual devices into separate die, making it easier to test them individually.

[14]Transparency films with $25\mu m$ resolution are supplied by `www.pageworks.com`.

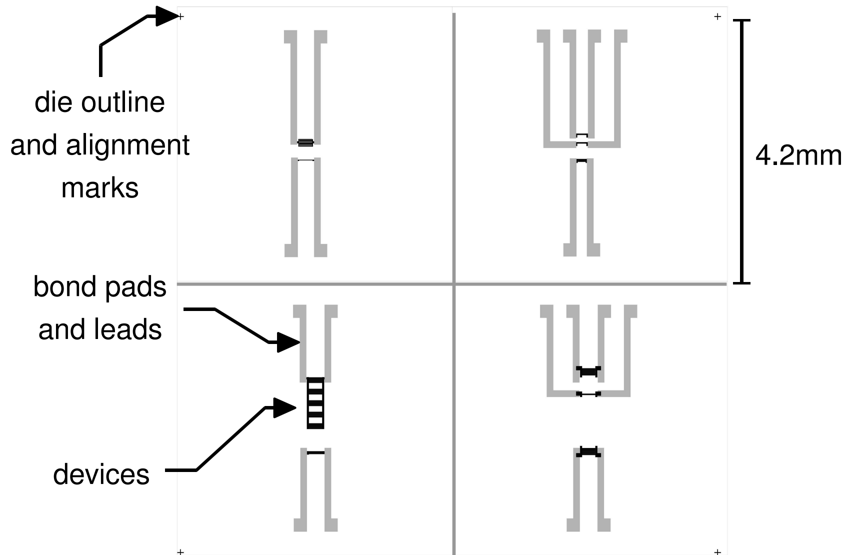


Figure 3-7: Artwork for mask 3 version 3.

3.2 Substrates

Two different substrates were used in the micro–fabrication part of this project, Quartz and lithium niobate wafers See table 3.3 substrate details.

Amorphous Silicon dioxide (also called fused SiO_2 and sometimes referred to incorrectly as Quartz) is used for deep UV photo-lithography masks because, unlike glass or other substrates, quartz is transparent to $220nm$ deep UV radiation. A relatively thick $1mm$ quartz wafers were chosen for the mask written by SEBL and used for hard contact printing or transfer to flex masks. The secondary mask has a thickness of $250\mu m$, to allow for conformal flexing during vacuum mask contact pattern transfer. Transferring the pattern to a secondary mask allows a flat and ridged structure for the SEBL to write to. Handling the thin $250\mu m$ quartz wafers was a challenge, as they are very fragile. If they are broken, transferring the pattern from the primary mask is much easier and less time consuming than writing a new mask via SEBL.

Mark Optics www.markoptics.com	
PN# WF3000X03931101	PN# WF3000X00131101
Quartz* w/ flat	Quartz* w/ flat
OD: $3.000 \pm 0.008 inch$	OD: $3.000 \pm 0.008 inch$
THK: $1mm$ +/- $25\mu m$	THK: $250 \pm 25\mu m$
Double Side Polished	Double Side Polished
Crystal Technology Inc. www.crystaltechnology.com	
Lithium Niobate	Lithium Niobate
Part Number : 99-60011-01	Part Number : 99-60011-01
128° RY-Cut	Y-Cut
OD: $3.000 \pm 0.008 inch$	OD: $3.000 \pm 0.008 inch$
THK: $0.0197 \pm 0.0008 inch$	THK: $0.0197 \pm 0.0008 inch$
Single Side Polished	Single Side Polished
X-flat and marker flat	Z-flat and marker flat

Table 3.3: Suppliers and specifications of substrates used. *Amorphous SiO_2.

Mark Optics specializes in semi–conductor lithography optical substrates and supplied the quartz wafers. These wafers arrived in a plastic canister with foam cushions and paper in between each wafer. Even though these optical wafers were supplied from a company selling semi–conductor grade substrates, they were especially dirty. Dust from the paper and other unidentified particles were on the surface and very difficult to remove. These particles caused problems with obtaining intimate contact between the conformal masks and the substrate. Cleaning these wafers was also costly. Contacting the vendor to ensure the wafers are manufactured and shipped from a clean–room in clean packaging is highly recommended if not essential[15].

[15]Mark Optics does supply clean–room cleaned and packed substrates upon request. Their clean process is advertised as; the wafers are SC-1 cleaned, followed by machine scrubbing. Then they

Lithium niobate is a manufactured in crystalline ingots and then cut and polished into flat wafers. The properties of surface acoustic waves traveling on lithium niobate are influenced by the crystalline orientation with respect to the wafer's surface. Each Lithium niobate wafer has a different set of flats to identify a particular crystal orientation. For this project two different crystal cuts (referring to crystal orientation) were used, Y-cut and 128°RY-cut. Refer to section 2.1 for details on the differences in the properties of these two crystal cuts [2, 41]. Lithium niobate wafers are also very fragile and must be handled with extra care.

The piezoelectric charging properties of lithium niobate caused problems in the fabrication process. Whenever the wafer was heated, either to cure photo resist or during processing such as e–beam evaporation, the wafer would charge and then arc near conductive surfaces. Unfortunate electrostatic discharges through the chrome on the mask occurred often, each time a millimeter size area of features on the mask would be destroyed. The arcing caused extensive damage to mask 1. The resulting degradation rendered many devices unusable.

3.3 Substrate Preparation and Cleaning

Ideally substrates arrive from the vendor clean and free of residues and particulate contamination. Contamination comes in the form of organic residues, metals, and particles. Residues, such as oils, photo resist, and scum cause adhesion problems for photo resist and metals. Particulate contamination causes intimate contact problems in contact mask lithography and will distort or ruin any device features in close proximity. Unwanted metals can cause the same problems as particles as well as shorting adjacent metal features, rendering the device inoperable. Before a new or

mega-sonically clean in recirculating filtered over-flow baths. The final step is scrubbing with machines using soft sponge brushes.

recycled wafer is processed, a cleaning procedure is used to remove contaminates unless the wafer was sufficiently cleaned by the vendor[42, 46].

Wafers that have chrome on them (such as the wafers used for the chrome masks) are stripped with CR-7[16] or equivalent chrome etchant. The chrome etch is repeated several times after the RCA[17] or Piranha[18] clean to ensure that all the chrome underneath the photoresist is removed.

A mixture of three parts sulfuric acid (H_2SO_4) to one part hydrogen peroxide (H_2O_2) is called "piranha." This cleaning step is is used to vigorously clean a wafer and remove heavy metals, particles, and organics such as aluminum and photoresist. The quartz and lithium niobate wafers were immersed in a piranha solution for $\approx 20min$[46]. Over time the heat generated by the sulfuric acid and hydrogen peroxide reaction dissipates. The mixture is carefully heated to maintain temperature and vigor.

Standard clean 1 (SC-1, also known as RCA-1 cleaning), was used to clean glassware, masks, and wafers before use. This was used as the last cleaning step before coating wafers with photo resist. Using a solvent clean after SC-1 is unnecessary and only increased the likelihood of contamination. Wafers were submerged in 4:1:1 solution of water:ammonium hydroxide:hydrogen peroxide heated to 80℃ for $20min$

After a dozen or so contact prints a thin residue film (scum) of photo resist would build up on the mask. To remove this scum the mask was cleaned in an oxygen plasma asher. The very reactive plasma–excited atomic oxygen reacts with the photoresist molecules taking them away from the surface. The plasma asher causes a small amount the resist molecules to clump together into very hard balls firmly stuck to the surface of the mask. A quick reactive ion etch (RIE) removes these clumps, but

[16] A standard 7% chromium etch solution.

[17] A cleaning process developed by RCA corporation and often referred to as SC–1, or standard clean 1. A mixture of ammonium–hydroxide, deionized water, and hydrogen–peroxide.

[18] A solution of sulfuric acid to hydrogen peroxide.

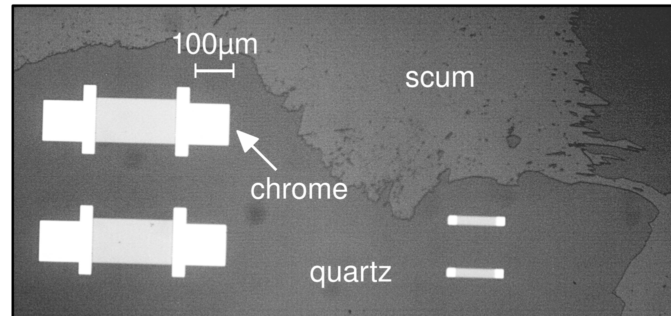


Figure 3-8: Micrography of PMMA scum on a quartz light–field mask with chrome.

care must be taken not to etch too far into the mask's metal[47, 42].

One clever solution to the buildup of scum that also resolves the problem with the mask sticking to the substrate is to Teflon coat the light–field masks. This process, suggested by another student[19], will be implemented as soon as new masks are ready. The material is Tridecafluoro–1,2,2–Tetrahydrooctyl, Trichlorosilane[20] ($CF_3 - (CF_2)_5 - CH_2 - CH_2SiCl_3$), a very durable self-assembling teflon monolayer that coats the exposed quartz[48]. Some Nanoimprint stamps use this coating to keep the resist from sticking to and breaking the relief structures. The coating is hydrophobic and tolerates RCA and solvent cleaning. A wafer is placed in a desiccator with a drop of the Teflon material. When the air is pumped out the Teflon coats the wafer in about 2mins. The wafer will turn cloudy if too much Teflon has been deposited.

Removing particles from quartz wafers was very difficult. In addition to SC-1 and Piranha cleaning, sonication was also tried with limited success. Wafers were also placed in a bath of Liqui–Nox™ heated to ≈ 50℃ for up to an hour. An ultrasonic acetone bath was also tried, with limited success. Spin drying wafers with high acceleration on the coater with acetone, methanol, and isopropanol[21] was marginally

[19] Filip Llievski of the MIT Materials Science and Engineering Department
[20] Available from Gelest Inc. part number : S1T8174.0–106M
[21] Often referred to as IPA or 2-proponal.

effective at removing stubborn particles. Acetone must be followed by methanol while the wafer is kept wet because Acetone displaces methanol. Likewise, isopropanol displaces methanol. If either acetone or methanol dries before being displaced, then dissolved organics may redeposit onto the surface of the sample. Some particles were removed by scraping them off with a needle under a microscope. Scraping is used as a last resort because of the hight likelihood of damaging the mask. Stubborn particles may also blasted off with a snow gun (compressed CO_2 gun), that creates tiny ice crystals which help sweep away particles.

Equipment and supplies, such as glassware and parts for the deep–UV aligner, must be throughly cleaned when they are brought into the lab. Liqui–Nox™, and the powdered form Alconox™, are strong detergents used to clean laboratory equipment in the same manner as one washes dishes. These detergents remove grease, oils, and other organic contaminates. Following washing and drying, the parts or glassware would be followed up by a solvent rinse before use. A quick rinse of the glassware with the solution to be used removes any contaminations that may dissolve in the chemical. Glassware, such as flasks used for photoresist, are cleaned with RCA–1 to insure absolute cleanliness.

3.4 Lithography

Lithography is a technique used in microfabrication to transfer an image to a photoresist (photo sensitive polymer). The photoresist is then developed to remove areas which correspond to the artwork desired. Further processing uses the photoresist to mask off areas of the substrate to allow for deposition or etching. Several different processes exist for transferring artwork into a photoresist. The two methods of photo–lithography used in this project are scanning–electron–beam–lithography (SEBL) and contact printing. High–resolution masks and one–off devices are pat-

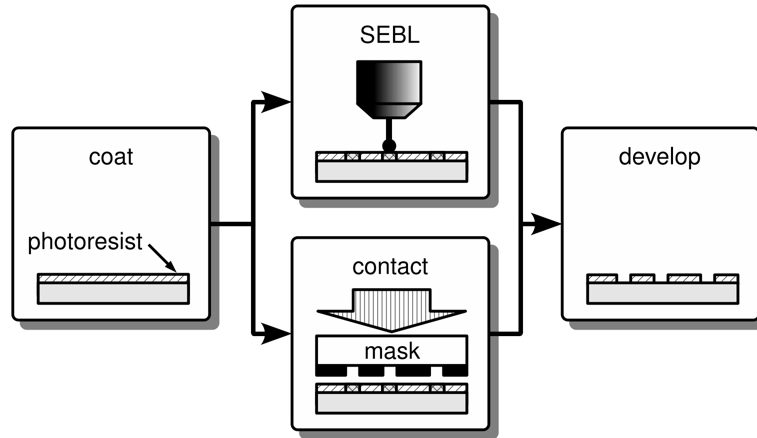


Figure 3-9: Typical lithography process steps used in this project. SEBL or contact printing can be used to transfer a pattern to the photoresist.

terned with SEBL, while photo–lithography is generally used when many duplicates of the same devices are needed. This section describes the entire lithography processes used, from applying photoresist to a wafer, through exposure, and development. Refer to figure 3-9.

A photoresist polymer is made of long chains of molecules and often comes dissolved in a solvent from a vendor. The wafer is coated with a uniformly thick layer of photoresist by spinning a small drop of liquid photoresist on the wafer and spinning at between 2000 and 4000rpm. Some of the solvent evaporates out of the photoresist while spinning the wafer. Before the wafer can be exposed, the photoresist must be cured though heating to completely remove the solvent, producing a durable layer that can resist etching, ion–bombardment, and evaporation. In the case of PMMA photoresist, exposing the photoresist to radiation causes the polymer chains break down. A developer is used to dissolve away the exposed photoresist. When a negative photoresist is used, the exposed area of the polymer are cross–linked after a post–bake step, causing unexposed area to develop away.

3.4.1 Photoresists

Photoresist are designed to be sensitive to different types of electromagnetic radiation (X-Ray, UV, deep-UV) and support different thicknesses, development rates, sensitivity, and maximum resolution. These parameters are all related and have corresponding trade–offs. Three different photoresists were used for this project; NR8, NR7, and PMMA[42].

The negative photoresists NR8-1000P[22] and NR7-1000P were used for the $3\mu m$ devices on mask 1. These resists are particularly suited for lift-off applications because exposure dose controls undercut, see section 3.7. See figure 3-24 on page 90.

The NR8-1000p was the first photoresist used and was based on a process already developed by another student working on a similar project. A similar resist, NR7-1000p, was later substituted for NR8-1000p for several reasons. NR8-1000p required cold storage and must be warmed to room temperature before use, while NR7-100p is stored at room temperature. NR8-1000p is an older resist that has a very limited shelf life where as NR7-1000p is a standard product with a longer shelf life.

NR7 is a specialty resist that is available in **P** and **PY** variants. NR7-1000PY is formulated for liftoff applications, whereas NR7-1000P is optimized for RIE/ION Milling masks. NR7-1000P was available in EML and is also suitable for the $3\mu m$ line width lift-off. The "1000" refers to a particular formulation that corresponds to a thickness of $1000nm$.

Polymethyl methacrylate (PMMA) was the third photoresist used. For this project, PMMA was used for SEBL and contact photo lithography to achieve line–widths as narrow as $300nm$. PMMA allows for higher resolution than NR7 and is also formulated to be exposed with SEBL or deep-UV radiation. The advantages of PMMA are higher contrast, higher resolution, high robustness, ease of use, and it can be open

[22]NR7-1000p and NR8-1000P are available from FUTURREX, INC. 12 Cork Hill Road Franklin, NJ 07416 www.futurrex.com

to sun–light if kept shielded in plastic containers. PMMA also works well in variety of environments, tolerating varying humidity and temperature well. Chlorobenzene and Anisole are two solvents used to dilute PMMA. The safer and environmentally friendly Anisole formulation was selected. Stock 11% 950k molecular weight PMMA solution dissolved in Anisole was diluted to achieve different film thicknesses. A solution of 3 parts Anisole to one part 11% 950PMMA results in the spin–curve shown in figure 3-10.

3.4.2 Coating

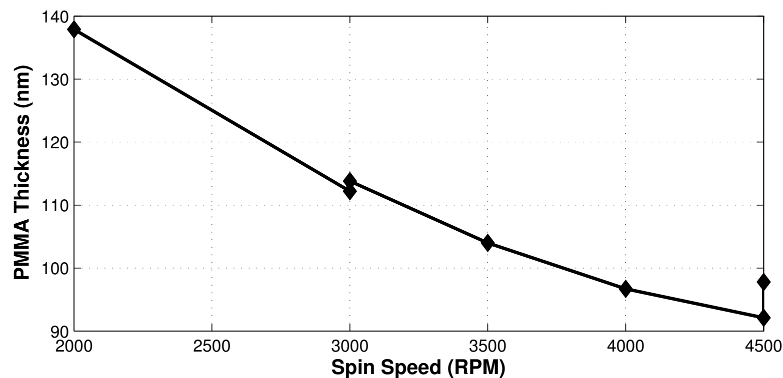


Figure 3-10: Spin curve of 3 parts Anisole to 1 part PMMA 950k A11.

A coater tool was used to deposit a thin uniform film of photoresist onto the substrate. Photoresist dilution and spin speed parameters determine the thickness of the photoresist. The procedure is to place $1 - 5mL$ of photoresist in the center of the wafer and then spin for $60sec$. A slow acceleration rate is used to allow the photoresist to spread out slowly, which yields a more uniform coating. When a new batch of photoresist is diluted, the first step is to run several silicon wafers through at various speeds to determine the thickness for a given speed. The thickness of each wafer is initially measured with an ellipsometer. After exposure and development,

which leaves a physical trench in the photoresist, a profilometer is used to measure the photoresist thickness.

3.4.3 Contact Lithography

Contact lithography brings the mask into contact with the photoresist. The equipment for contact lithography is relatively simple and much less costly compared to optical projection systems, making the process ideal for research applications. Contact lithography leaves a small gap between the mask and substrate, which limits the resolution due to near-field diffraction. For the best contact lithography systems, the resolution is limited to $\approx 1\mu m$[49]. Industry prefers other methods, because contact lithography wears down the expensive masks quickly in production systems. Contact mask lithography is more than adequate for research applications. Contact lithography system usually contain an (x, y, θ) stage, allowing precise alignment of a mask to previously patterned wafer or wafer flat. Contact lithography systems are referred to as "aligners," describing their alignment capabilities. The lithium–niobate wafer flats were aligned using an (x, y, θ) stage. Lithography systems are also referred to by their exposure wavelength, such as I–line for $\lambda = 365nm$[50].

The EML I-Line aligner was used for making the $3\mu m$ devices. This commercially produced tool had limitations and problems due to wear and age, which caused some difficulty. The exposure timer was not working properly, and with $6second$ exposures, accurate manual timing was difficult. Alignment was tricky as the (x, y, θ) stage's motion was limited. To view the $LiNbO_3$ wafer flat in the microscope, the wafer needed to be offset by a few centimeters, which leads to degraded contact between the mask and the wafer.

3.4.4 Conformable Vacuum Mask Lithography

A conformable vacuum mask lithography tool in NSL was used to achieve higher–resolution than the EML I-line aligner was capable of. Conformable contact lithography is a simple method of yielding very high resolution at low cost for research and low-volume production[15]. The vacuum aligner in NSL is a unique custom–built experimental system, which is capable of resolving line–widths of $100nm$[45, 44]. A $XeHg$ lamp provided $\lambda = 220nm$ deep–UV light. Air is evacuated from between the conformable vacuum mask and the substrate, causing them to be in intimate contact. This close contact allows features smaller than the diffraction limit to be resolved[15, 49]. The vacuum aligner uses very thin, $250\mu m$ thick, flexible quartz masks to allow conforming around contamination particles and to accommodate any flatness variation in the wafers.

The NSL deep–UV aligner required a new mask chuck to use the $3inch$ wafers in this project. The mask chuck holds the mask for alignment and makes vacuum contact with the mask. I designed a three inch chuck, which was fabricated at the MIT central machine shop. The existing vacuum system was controlled by on/off needle valves, which did not release the vacuum on the wafer or substrate. These valves were replaced with venting ball valves to allow proper operations. A light–tight housing made of acrylic was also fabricated to protect the lab space from hazardous deep–UV radiation. Details of these modifications are provided in Appendix C.2.

As air escapes from between the mask and the substrate when they are brought in closer contact with each other. Before the two wafers contact fully, fringe patterns appear, as shown in figure 3-11. The number of fringes directly relates to the wavelength of the light. There may still be a significant gap at $\lambda = 220nm$, even if no fringes are detected in the visible spectrum[15].

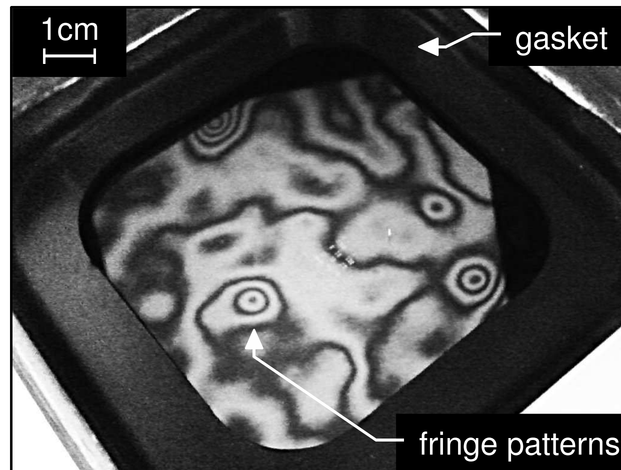


Figure 3-11: Fringe patterns resulting from imperfect contact between mask and substrate while under vacuum and illuminated with a green monochromatic light. This shows the OAI 77mm chuck with a top–down rubber gasket over the mask. The mask was 1mm quartz and the substrate was a 77mm silicon wafer.

3.4.5 Scanning Electron Beam Lithography

A scanning–electron–beam–lithography tool is essentially a scanning–electron–beam–microscope (SEM) used to aim electrons at a sample to expose patterns onto a photoresist. SEBL electron beam–widths can approach 1nm with 30nm resolution[42]. The Raith–150 is able to write lines as narrow as 17nm and gratings of 70nm. While SEBL is able to write very small features, the drawback is that this is a scanned beam which means writing large high–resolution patterns will be very time consuming. For example, to write a simple mask takes about two hours on a SEBL including 40$minutes$ of setup time. More complicated masks take much longer to write. In comparison, an exposure into PMMA on the OAI takes only 30$minutes$, no matter how complicated the pattern.

The first step to writing a pattern with the SEBL is to correct for misalignment, focus, aberrations, and stigmation. 100nm gold particles deposited from a liquid suspension were dried onto the surface of the sample and imaged. Adjustments are

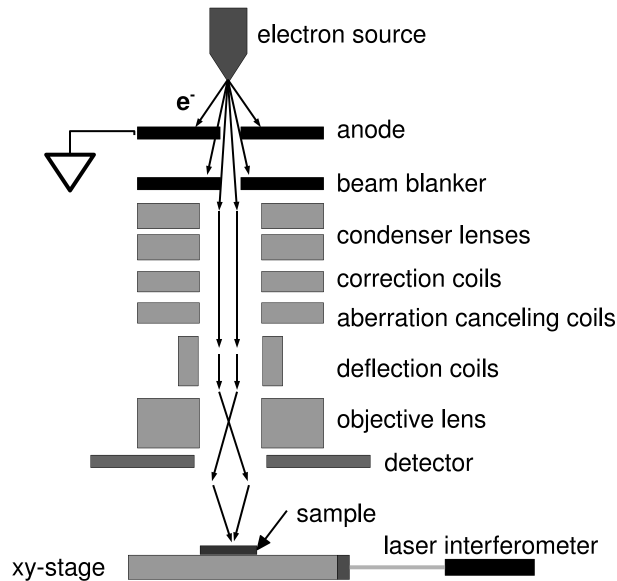


Figure 3-12: Simplified diagram of a SEBL tool. The detector is used for imaging test particles that are used to adjust for focus, aberration, stigmation, and write field alignment.

made to give a nice image of a gold particle. The area that the electron–beam scans is small, from $50\mu m$ to $500\mu m$ depending on beam– width. To increase the writable area an xy–stage repositions the sample after each scan field has been written. Moving the stage introduces further errors, and calibrations must be made to ensure proper field stitching alignment. Field stitching errors were a common problem and some of the results are shown in figure 3-13. Stitching problems were most likely due to improper execution of the write–field alignment procedure.

Electrons that impinge on the photoresist can build up a surface charge, which can cause an unwanted electric field to deflect the scanned beam and thus distort the written pattern. A thin layer of conducting film is used to shield the electric fields when writing on insulating substrates. Conductive samples, such as the chrome on the etched masks, do not need additional shielding. A thin $7nm$ chrome layer was deposited on top of the PMMA and later etched off with a standard chrome etch (CR-

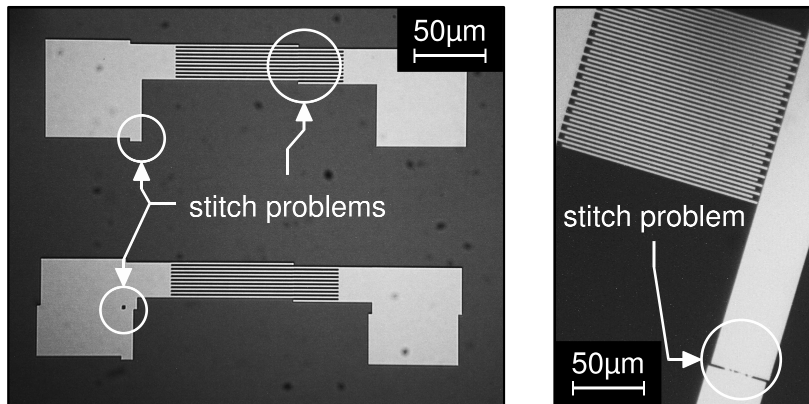


Figure 3-13: Micrographs of field stitching problems.

7) with no effect on the exposed PMMA. Chrome was selected for easy deposition and etching properties and low–cost in comparison to exotic conductive polymers, such as Aquasave[23].

3.4.6 Exposure

Exposing PMMA causes the long polymer chains to break down in a process referred to as scission. The amount of energy imparted in the photoresist determines how much the long polymers are chopped up. Determining the correct amount of energy to expose a photoresist is critical for the artwork to be exactly reproduced[42, 43]. An incorrect exposure dose will lead to widening or narrowing of features.

Test features, such as those shown in figure 3-14, were used to evaluated the effect of different doses. Nested L exposure test features allow for checking line uniformity and minimum achievable line separation. Diagonal boxes are useful for checking proper exposure and development. When the boxes just barely touch, the optimal parameters have been achieved.

[23]Aquasave™ can be spun on with a coater and is useful for coating samples that can not go into an e–beam evaporator. Available from Mitsubishi Rayon Co. `www.mrc.co.jp`

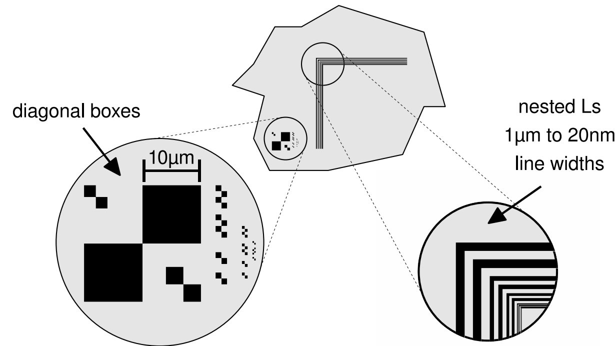


Figure 3-14: Exposure test features allow inspection of focus, dose, and development parameters.

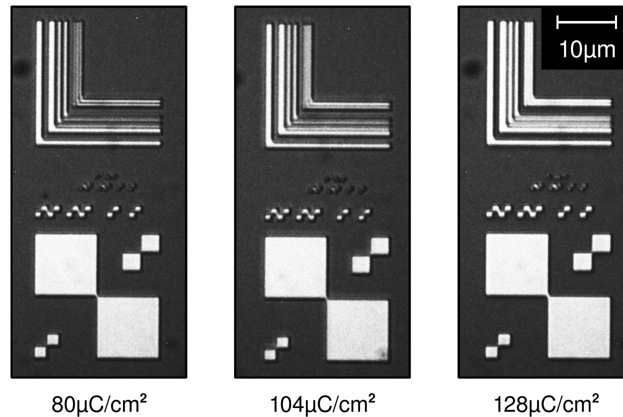


Figure 3-15: Micrograph of a dose matrix written in $100nm$ of PMMA on Quartz. Exposed with the Raith 150 SEBL tool.

Figure 3-15 shows the test structures with three different doses. The nested Ls in this figure are made up of a pair of $1\mu m$ lines, a pair of $0.5\mu m$ lines and three lines with $100nm$ widths. As the dose is increased the line–width widens, eventually causing them to join with adjacent lines. At a dose of $128\mu C/cm^2$ the $100nm$ lines have blurred together and the connection point between the diagonal boxes has widened.

On the other hand, exposure doses that are too low will cause line–widths to be narrower than desired. One method of checking for proper exposure is to use diagonally connected boxes. The exposure is correct when the boxes are just touching.

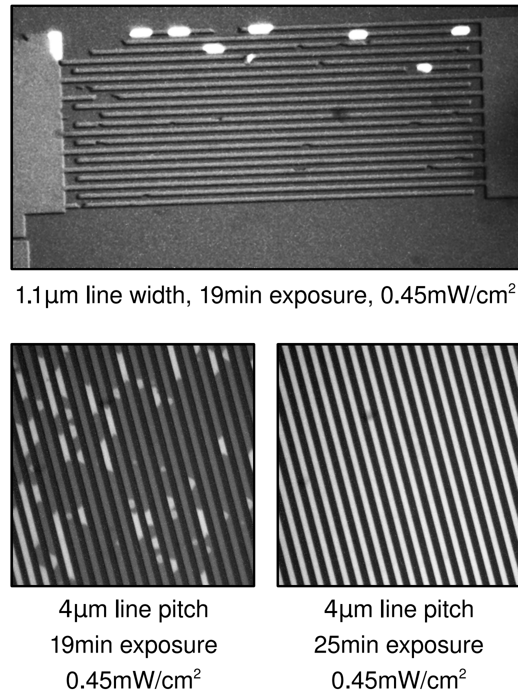


Figure 3-16: The top micrograph shows clearing problems with PMMA on Quartz. The bottom two exposure show improperly cleared (left) and properly cleared (right) $4\mu m$ grating of PMMA on Silicon. The darker areas are PMMA and the light areas are substrate. These exposures were made with the OAI deep–UV exposure tool in NSL.

In extreme cases of underexposure, the photoresist will not clear when developing. Figure 3-16 shows such clearing problems due to underexposure.

NR-7 photoresist was very sensitive to exposure dose. With a dose–rate of $4.5mW/cm^2$ at I–line ($\lambda = 365nm$), an exposure time of only $7seconds$ achieved good results. For contact lithography with a deep–UV ($\lambda = 220nm$) at $0.47mW/cm^2$, PMMA did not clear until after $1320seconds$. Good result where achieved with an exposure time of $1500seconds$[50].

3.4.7 Resist Developing

Developing photoresist involves submerging the wafer into a solution that will dissolve away exposed areas. The polymer chains in the exposed areas of PMMA are shorter due to scission, making removing this material easier for the developer in comparison to the unexposed PMMA. The proper temperature and development times are the two most critical parameters.

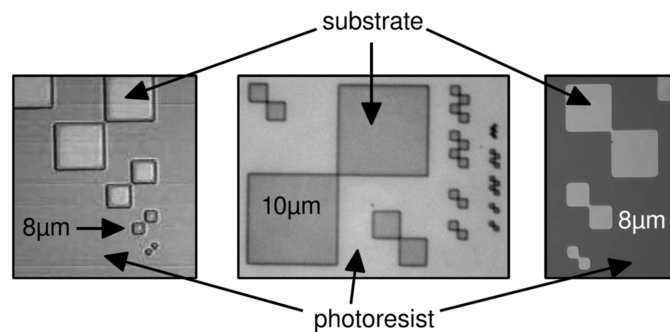


Figure 3-17: Developed exposure/developing test features showing the degree of over/under developing. The left panel shows underdeveloped clearing problems, and the right panel shows overdeveloping. The center panel show ideal exposure, with the two boxes coming into perfect contact.

Unfortunate temperature was not taken into account or measured when developing NR–7. In hindsight, the temperature of NR–7 should have been recorded and made consistent. Development time appeared to be the most important parameter for NR–7. Too short a development time caused insufficient clearing of resist from unexposed regions (NR–7 is a negative resist). A longer than necessary development time caused pitting of the resist and led to line-width widening, see figure 3-17. The correct development times for NR–7 were determined by trial and error. A 2% solution of resist developer called RD2[24] was used. When RD2 was not available RD6 was diluted 1:3 with deionized water into the equivalent of RD2. Development was stopped by

[24]RD2 is a standard 2% developer for resist supplied by Futurrex, Inc.

submerging the sample in deionized water.

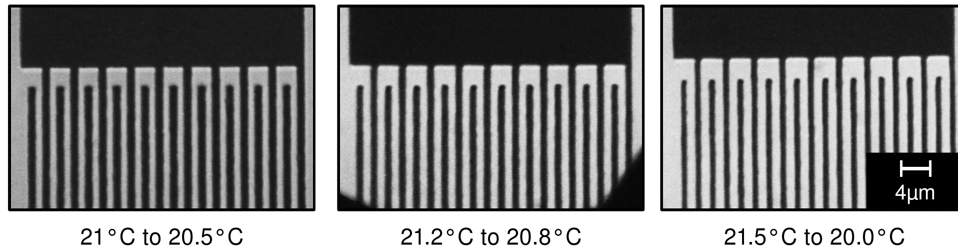


Figure 3-18: Changes in width of $1.1\mu m$ lines due to developer temperature.

PMMA was developed in a one part MIBK[25] diluted with 2 parts IPA. Temperature and development time for PMMA appeared to be very sensitive at the $100nm$ to $300nm$ scale. Development times were kept as accurate as possible to approximately $\pm 2seconds$. Line–widths would show noticeable differences with temperature variations as little as half a degree Celsius. A quantity of $500mL$ of developer was placed in a $1L$ flask and heated on a hotplate to exactly 21℃ using a thermometer accurate to a $10th$ of a degree. The flask was removed from the hotplate at exactly 21℃ and the $90second$ development was started. The temperature of the developer would drop about half a degree during the $90seconds$. The development was stopped by spraying isopropanol for $60seconds$. A spray was used over immersion to make sure isopropanol entered the small features quickly enough to avoid over development.

3.5 Descum

Besides cleaning and etching (as discussed in section 3.3), the O_2 plasma asher and reactive–ion–etching (RIE) can also be used for removing residuals from the exposed substrate areas on the wafer. Such a "descum" step helps fix certain adhesion problems. The drawbacks of using the O_2 asher for descum in liftoff applications is that

[25]MIBK(methyl isobutyl ketone) or 4–Methyl 2–Pentanone (From AlfaAesar #A11618)

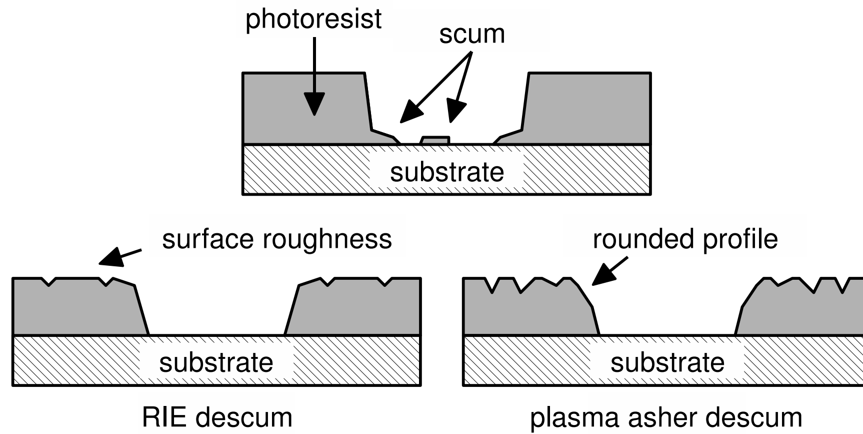


Figure 3-19: Exaggerated illustration of the effect on resist profile from RIE or plasma asher descum processes. The top figure is the photoresist profile after development and before any descum step.

the etch is isotropic. Thus the quality of the sidewalls can be degraded causing the metal to bridge, ruining liftoff. Descum results from using the RIE is generally better because the ions are accelerated under a bias voltage resulting in an etch nearly perpendicular to the substrate, leaving a better profile for liftoff[42, 47].

The etch rate of PMMA in the O_2 plasma asher was characterized using silicon test wafers. A single wafer was coated with $106nm$ of PMMA and then oven baked for $30min$ at 170℃. This wafer was then repeatedly put into the asher and measured with the ellipsometer until the thickness of PMMA was less than $70nm$ (the point at which the accuracy of the ellipsometer used becomes questionable).

To better characterize the difference between using the O_2 asher or O_2 RIE for descum, some silicon test wafers were processed. Preliminary results show roughing of the surface of the PMMA, see figure 3-21. The difference in surface roughness may be due to the amount of time or the parameters used in the descum step. More work is underway to better characterize and optimize the parameters for descum.

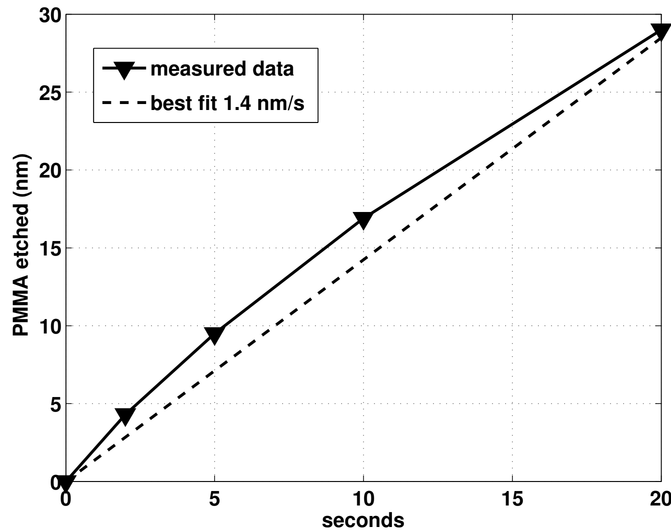


Figure 3-20: Etch depth vs. time of $106nm$ PMMA on Si in O_2 plasma asher using $50W$ power and $0.305psi$ average pressure. Thickness measured with an ellipsometer.

3.6 Deposition

Electron–beam evaporation was used to deposit aluminum and chrome onto substrates. Evaporation works by heating a source of material in a crucible with an electron beam in a vacuum [47]. The source material sublimates onto an opposing sample. Usually the source is at the bottom of the crucible and the target is at the top facing down. For liftoff applications, the target wafer must be located directly over the crucible to mitigate the effects of sidewall shadowing (see figure 3-22).

Aluminum was used to create the interdigital transducers on the lithium niobate for the SAW devices. The thickness of aluminum ranged from $50 - 200nm$. The SiO_2 masks used $40nm$ of chrome for the patterns and $7nm$ of chrome for shielding of the electron field for SEBL (see section 3.4.5).

Two different evaporation tools were used. A custom built evaporator assembled from older evaporator components in EML evaporated the aluminum for the $3\mu m$

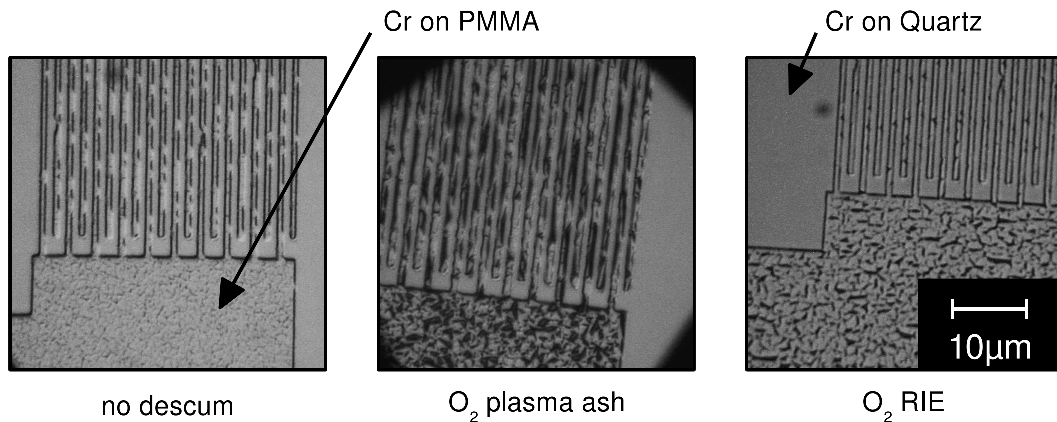


Figure 3-21: Results from $2sec$ of O_2 plasma asher and RIE descum methods. The figures show Cr on Quartz and Cr on PMMA before the liftoff step. $10nm$ of PMMA were taken off with the O_2 asher. $7nm$ of PMMA were taken off with the RIE.

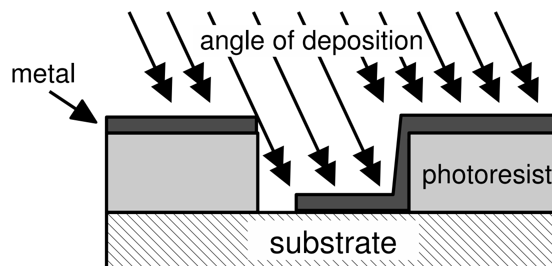


Figure 3-22: Effect of off–angle deposition on side–wall coating.

devices. The EML tool operation allows the tool to use a selection of many different materials[26]. Contamination from these materials was an issue due to the way the tool was designed, cleaned, and used. Another problem with the EML e–beam evaporator was that the film thickness was manually controlled with a shutter, yielding tolerances of $\pm 20nm$ for aluminum, and as bad as $\pm 100nm$ for chrome. Depositing chrome was problematic as intermittent spikes in deposition rates made controlling film thicknesses difficult. One theory was that the chrome would grow thin filaments

[26]Often evaporation tools are limited in their selection of materials to avoid contamination problems.

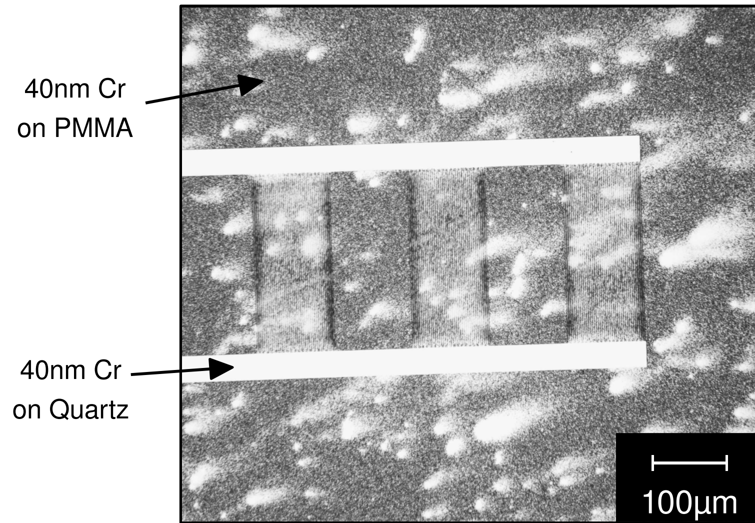


Figure 3-23: Micrograph of a problem encounter when depositing 40nm Cr onto 100nm of PMMA on an SiO_2 wafer using the NSL evaporation tool.

during evaporation, which would short to the electron gun under the magnetic field. The resulting shorts would cause spikes in power and deposition rate, referred to as “spitting.” Films put down under spitting conditions had poor uniformity. The comet spray patterns shown in figure 3-23 are believed to be the result of chrome spitting problems. The evaporation tool in NSL was under computer control and yielded much more consistent and higher quality films in comparison to the EML evaporator.

3.7 Lift–off and Etching

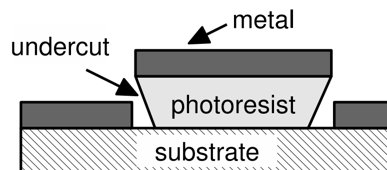


Figure 3-24: Undercut allows solvent to dissolve photoresist and remove metal overlay.

After the photoresist has been patterned, and metal has been evaporated onto the sample, the photoresist is removed taking away the unwanted metal in a process called lift–off. Lift–off takes place in a solvent bath that dissolves the photoresist and lifts the metal off of the substrate in the processes. The solvent needs access to the photoresist underneath the metal, which can be facilitated through sidewall undercut, photoresist thickness, or as a last resort sonication. See figure 3-24.

Acetone or Microstrip was used to lift–off NR7. Much better results were obtained using Microstrip, as the acetone took a lot of time to completely remove the NR7. The profile of NR7 has a decent amount of undercut aiding in lift-off. On the other hand, PMMA does not have undercut, which made executing a clean lift–off very difficult. NMP[27] is used for dissolving PMMA in lift–off and is slowly heated to 80℃. Heating NMP must be done very carefully as NMP's flash point is 93℃, which is clearly written on the bottle. Poor results were obtained when using unheated NMP.

Lift–off must be completed before removing the sample from the solvent, or there is the possibility that metal will fall into the trenches and be very difficult or impossible to remove. Gently scrubbing the area where stubborn metal pieces are stuck with a small fab–swab, while the wafer is wet with acetone, may remove them.

The process of lift–off, in theory, should only require immersion in a solvent. However imprecise profiles or deposition could leave bridging between metal on the substrate and metal on the photoresist. A hydrophobic photoresist may also prevent the solvent from entering very small trenches. Ultrasonic agitation of the sample in a solvent may aid in lift–off by removing stubborn photoresist and metal. This technique must be used sparingly and only when necessary, as too much sonication will damage devices. See figure 3-25.

Problems due to bridging were found in SEM micrographs of test devices patterned

[27]NMP (1-methyl 2-pyrrolidinone)

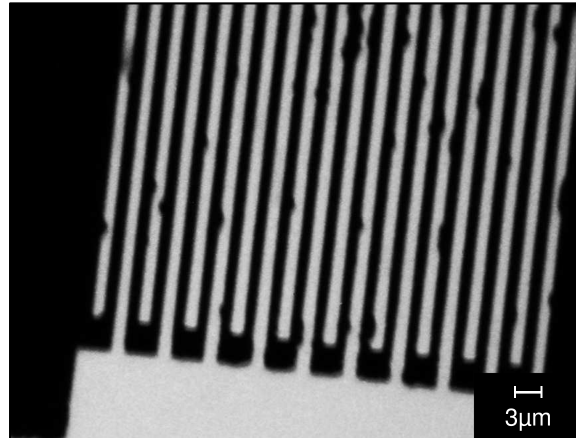


Figure 3-25: Damage likely resulting from too much sonication ($> 1min$) during lift–off.

onto silicon. Figure 3-26 shows how bridging between metal in the trenches and metal on the surface of the photoresist will ruin lift–off. The SEM image shown in figure 3-27 was a clear indication of the bridging that caused lift–off problems. A sectioned illustration of the side wall problem is shown in figure 3-28. The lift–off for this device was aided with sonication, and due to the brittle nature of chrome, some of the unwanted overhangs broke off.

Too better understand the lift–off problems, some SEM micrographs were taken of the sample after evaporation of less chrome ($30nm$ instead of $40nm$) and before lift–off, figures 3-29 & 3-31. These images show that there is adequate separation between the metal on the photoresist and the metal on the substrate, which should (and did) lead to a successful liftoff. Only minor bridging was observed at the corners of the metal fingers. Viewing a PMMA coated sample in the SEM will cause cracking and melting of the photoresist. This cracking is clearly shown in the figures. Lift–off in the areas scanned by the SEM failed due to the changes caused by the bombarding electrons. Figure 3-30 shows the area that was scanned by the SEM after lift–off when taking the image shown in figure 3-29.

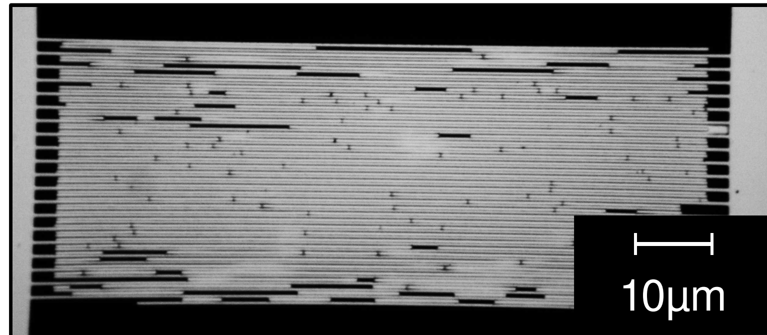


Figure 3-26: Incomplete liftoff problems as a result of too much chrome being deposited. Shows 500nm line–widths with 210nm of chrome on quartz. The desired thickness of chrome was 100nm. The PMMA thickness was 250nm.

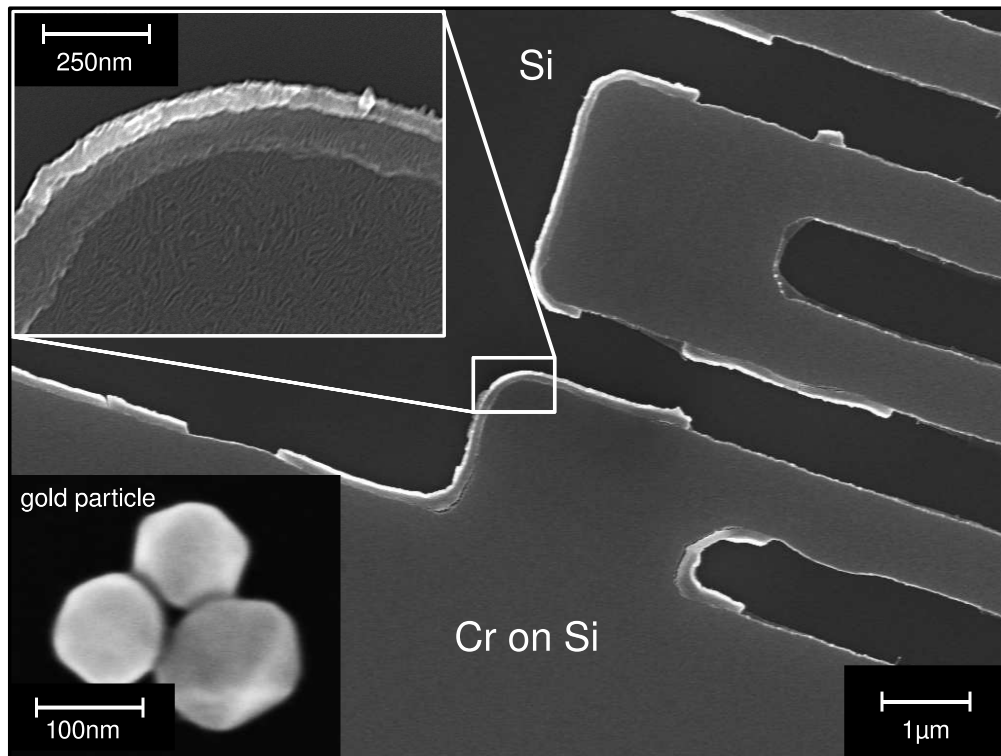
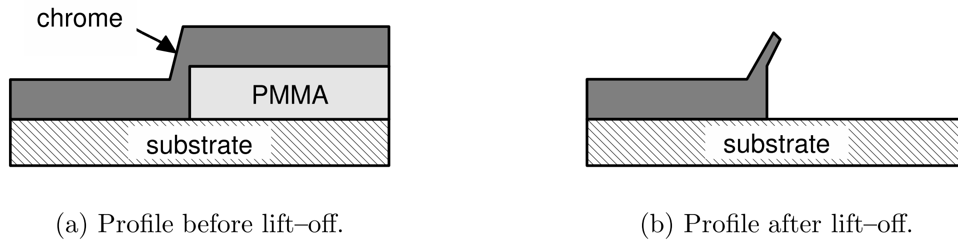


Figure 3-27: SEM of bridging caused by poor profile and photoresist that was too thin for the thickness of metal deposited.



(a) Profile before lift–off.

(b) Profile after lift–off.

Figure 3-28: Illustration of lift–off problems when the metal on top of the photoresist bridges to the metal on the substrate.

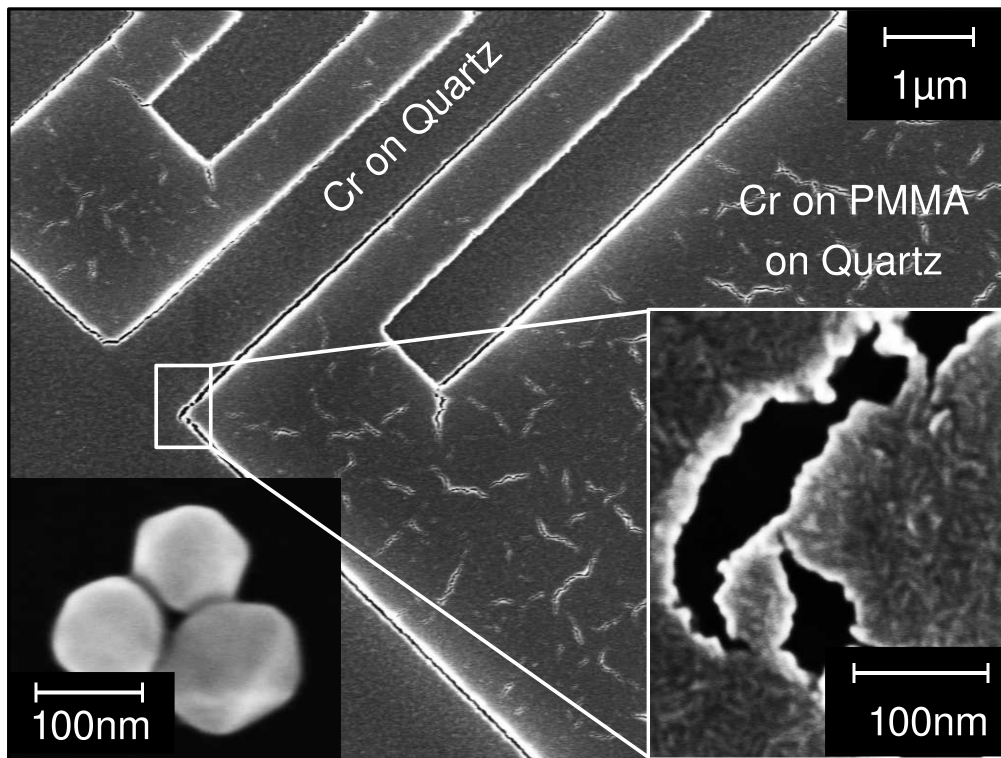


Figure 3-29: SEM showing a wafer after deposition and before lift–off. The images of the 100nm gold particles in the bottom left are given as a reference for image quality.

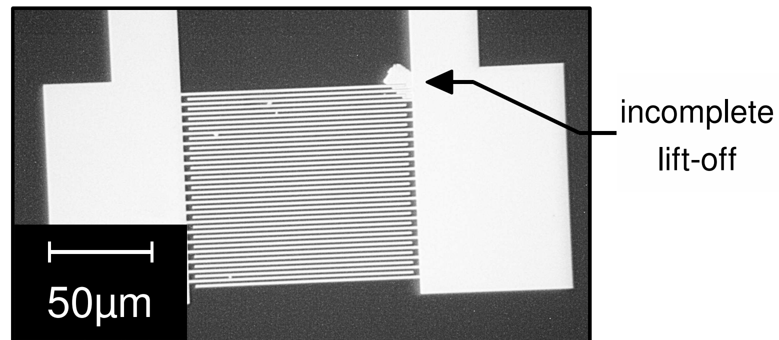


Figure 3-30: Micrograph showing a patch of incomplete lift-off due to inspection by the SEM before liftoff to take the image shown in figure 3-29.

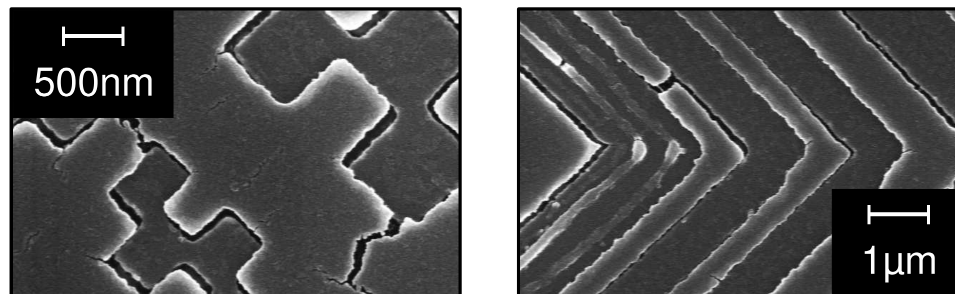


Figure 3-31: Test patterns of Cr on Quartz and Cr on PMMA on Quartz. Same sample as shown in figures 3-29 & 3-30.

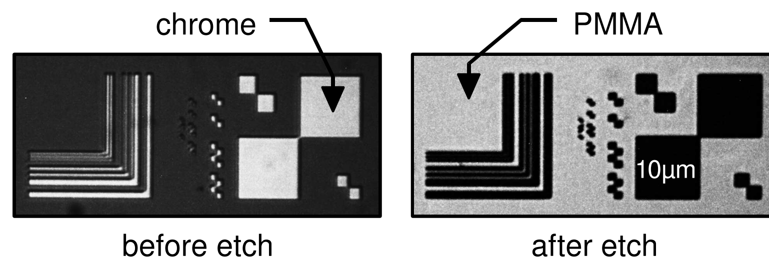


Figure 3-32: Show the change in image contrast before and after the etch has been completed. PMMA was used as a photoresist on top of chrome and quartz.

Wet etching with a standard electronics grade chromium etchant (CR–7) was used. Etching chrome is very simple, is done at room temperature, and only requires immersion into the chrome etchant. For etching mask artwork, CR–7 was diluted 1:2 with deionized water allowing for more precise control over the etch. With the diluted 1:2 solution, etching through $40nm$ of chrome takes about $40seconds$. Etching chromium is very similar to developing photoresist. Leaving the sample in the etchant for too long will cause an over–etch and line–width widening. If the sample is not left in the etchant long enough there will be clearing problems. In general, proper etching was very easy to accomplish.

3.8 Mounting/Packaging

A Disco DAD-2H/6T die cutting tool, provided by MTL, was used to cut a wafer into rectangular die. The devices on mask 1 were not positioned so they could easily be cut out. To ensure that the devices I wanted to test were not destroyed, the die size was adjusted between $\approx 5mm$ and $\approx 10mm$ on a side. The devices on mask 2 and 3 were positioned so that they could easily be cut out and would fit on a $4mm \times 5mm$ die. A 28–pin SOIC with a 165x205mil cavity was purchased from Spectrum Semiconductor

Materials, Inc.[28] part number `CS0028P2`. The test package, with the lid removed, was sized to take a $4mm \times 5mm$ die. This package was selected for size, ease of wire–bonding to the level cavity, and surface mount leads, which have lower parasitic impedance.

The die was super–glued[29] to the package and then wire–bonded. Care needed to be taken in application as any excess glue covering the leads would make wire–bonding impossible.

Wire–bonding proved to be very problematic. The tool was very difficult to set up, which involved threading a very thin wire through an ultrasonic tip and adjusting the bonding power and duration. To simplify the process for mask 1, the bond pads were made in the same step as the devices, and thus only had a thickness of $\approx 200nm$. These thin bond pads on top of the fragile lithium niobate caused cracking of the wafer underneath the pads, leaving the pads to break off. A very narrow range of settings would allow for proper bonding without ruining the pad and having a mechanically and electrically sturdy connection. In an attempt to make wire bonding easier, as well as reduce write time on the Raith 150 SEBL tool, masks 2 and 3 are designed to include a separate bond pad layer. These bond pads will be made thicker and out of gold or other easily bonded to material.

The test PCB was designed and drawn in Mentor Graphics® Design View™ and Expedition PCB™ software packages. See figure 3-33. Advanced Circuits fabricated the PCBs from the gerber files generated with Expedition PCB™. This high–frequency design incorporates several important features. The end–launch SMA connectors are in–line with the signal path to reduce impedance variation and signal reflections. The traces on the PCB are impedance matched to 50Ω, and a test trace was placed on this board to verify proper implementation. An impedance matching network

---

[28] `http://www.spectrum-semi.com`

[29] Tech Hold™ Cyanoacrylate adhesive by TechSpray.

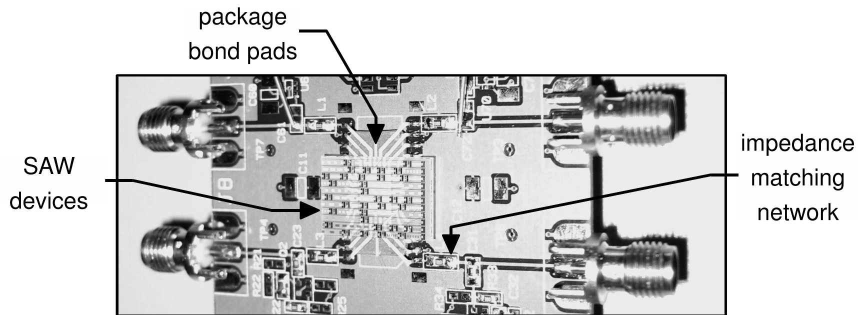


Figure 3-33: SAW test devices mounted to a package, wire-bonded, and soldered to a test PCB.

was placed as close to the SAW test package as possible to reduce parasitics. All the parts were surface mount and placed top–side as closely together as possible to reduce parasitics and prevent noise from coupling into the signal path. Rounded pads were used on RF packages so solder would flow with reduced discontinuity for better signal propagation[3]. Details, artwork, and schematics are available in appendix D.

Chapter 4

Electronic Design

Included in this section is a detailed description of the architecture, design, and implementation of all of the electronics and supporting software used in this project. Requirements of the design and the approach taken to meet those requirements are also covered. The field–programmable–gate–array (FPGA) design went through several iterations of increasing complexity. Only the highlights are reviewed here because each implementation is very complex. Programs written to control of the FPGA hardware peripherals running on FPGA's embedded processor are covered in section 4.7. User interface programs are also covered in section 4.7. To keep the size of the appendix manageable, only a few of the most important source files were included.

A short radar chirp containing a phase–encoded signal is transmitted by a measurement station to the SAW transponder (covered in detail in chapter 1). The measurement station measures the round–trip time–of–flight to and from the SAW transponder. Specialized high–speed electronics are necessary for a measurement station to operate at $2.4GHz$. Signal range, accuracy, and noise were difficult requirements for the measurement station's design to meet. On top of that, limitations in the both the project's budget and cost for commercially viable systems made the

measurement station's design even more challenging.

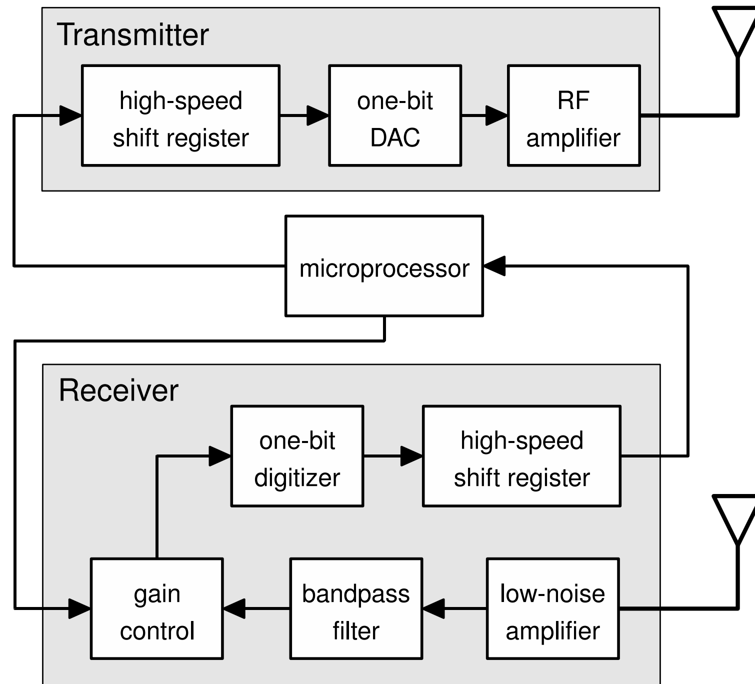


Figure 4-1: Conceptual measurement station transceiver block diagram.

The basic design of the first system is shown in figure 4-1. A simpified design was used first to make the system tractable. Better performing more complicated designs will be pursued once the concept has been proven. A microprocessor determines the correct digital code which needs to be transmitted to select a particular tag. The phase encoded digital code is then shifted out at the carrier frequency. Before transmission via the antenna, the digital signal is conditioned into suitable radio frequency waveform and amplified.

The receiver continually samples for both the transmitted signal and the SAW reflected signal. In this way, the microprocessor can determine the time between when the transmitted signal was received and the time the transponder's signal is received. The receiver's front end is typical, containing a low-noise amplifier (LNA), a bandpass

filter, and a gain controller. The gain controller is set by the microprocessor in such a way that the digitizer does not saturate when the high-power transmitted signal is received. A one-bit digitizer and high-speed shift-register captures the signal, which is processed by the microprocessor. To simplify the design, and more importantly reduce the cost of the demonstration electronics, only a one-bit digitizer (ADC) is used in the demonstration system. Future systems would utilize more sophisticated ADC's with multi-bit resolution.

The receiver is the most challenging component of the electronics to design and implement. To make the initial system evaluation easier a high-speed oscilloscope will be used to digitize the return signals. MATLAB® will then process and simulate different receiver configurations before the initial design and implementation of the custom receiver electronics.

The measurement station was implemented with a very fast oscilloscope, an FPGA development kit, and some custom RF electronics. While detecting a particular transponder and measuring time–of–flight would be possible using a low cost FPGA, the demanding challenges involved were outside of the practical goals of this project. Therefore we chose to leave an integrated solution for future work. Instead we chose to use a very fast oscilloscope with $5GHz$ of band–width and capable of 20–Gigisamples–per–second, to implement the receiver.

Commercial instruments[1] are available that can generate the radar chirps required. However, with prices ranging between $\$40 - 60k$ commercial instruments are prohibitively expensive for our budget. With a significant amount of work, the necessary functionality was achieved with an $\approx \$1k$ FPGA development board and $\approx \$1.5k$ of auxiliary electronics. A typical desktop computer with no additional hardware was utilized as the user interface. The programming languages were PERL for the user

[1] For example, the Agilent 81133A, $3.35GHz$, single channel Pulse Pattern Generator.

interface, MATLAB® for real–time signal analysis, C and VHDL for programming the FPGA.

One approach to measure time-of-flight that was initially investigated is to use one of a numerous collection of existing time-to-digital converter (TDC) designs. TDCs have been used in physics research and radar applications for over three decades. Accuracies of $30ps$ are tractable, and would lead to a range resolution of $c \cdot 30ps/2 = 4.5mm$. Existing TDC designs are not easily interfaced to correlated responses as required by our approach. Utilizing a high–speed multi–gigibit–transceiver appears to be the most flexible, easy to implement, and economical solution.

4.1 System Architecture

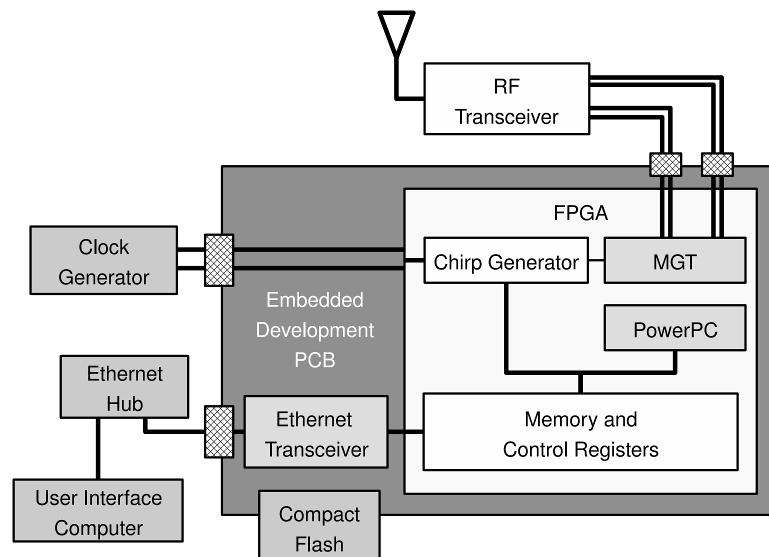


Figure 4-2: Measurement station FPGA and transmitter block diagram.

The envisioned architecture called for an integrated measurement station as shown in figure 4-2. A Xilinx® embedded development kit (EDK) printed circuit board makes up the backbone of this system. The EDK PCB consists of a versatile FPGA

with various peripherals such as Ethernet, a multi-gigabit-transceiver (MGT), and a compact flash interface. Ethernet allows the measurement station to communicate with other devices including other measurement stations and the user interface running on a remote desktop, laptop, or oscilloscope computers. An external clock generator provides the precise frequency used by the FPGA to synthesize the radar chirps. The EDK PCB uses files written to the Compact Flash card by the developer to configure the FPGA. Configuration data includes the FPGA hardware code and software running on the FPGA's built–in PowerPC processor. The RF Transceiver was implemented with custom electronics. Phase–shift encoded data from the FPGA is transmits though an amplifier and antenna as radar chirps.

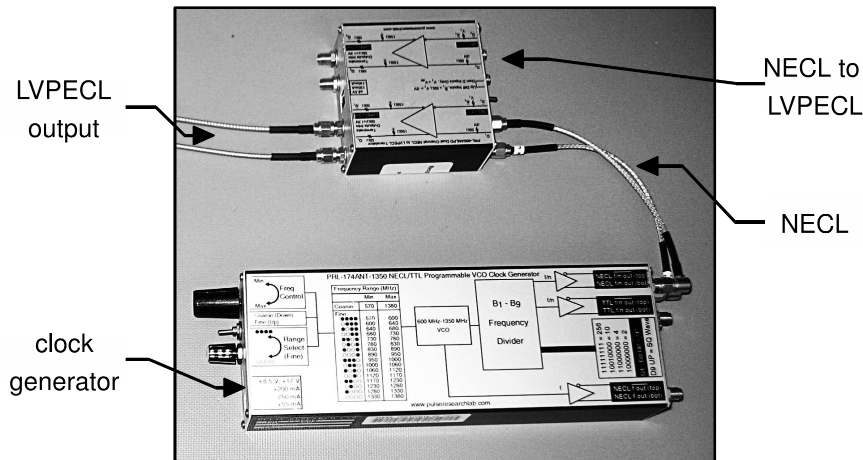


Figure 4-3: Photo of NECL clock generator (part number PRL-174ANT-1350) and NECL to LVPECL logic level translator (part number PRL-460ANLPD).

The clock generation hardware consists of two commercial modules. The configurable clock source is generated from a Pulse Research Lab[2] model PRL-174ANT-1350 programmable VCO clock source. A PRL-460ANLPD NECL to LVPECL logic level translator module was used to convert the negative-emitter-coupled-logic (NECL) outputs from the clock source to the low-voltage-positive-emitter-coupled-

[2] www.pulseresearchlab.com

logic (LVPECL) required by the Xilinx Virtex-4 FPGA. NECL and LVPECL are both widely used differential logic standards advantageous for their high-speed and superior immunity to noise.

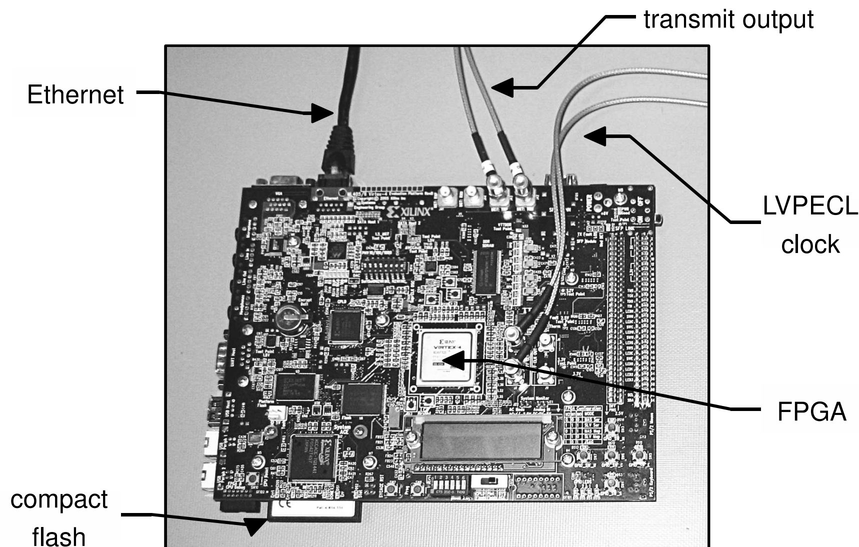
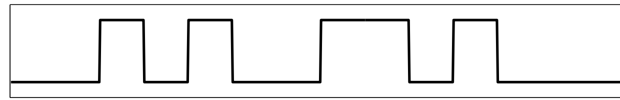


Figure 4-4: Photo of Xilinx ML405 evaluation board.

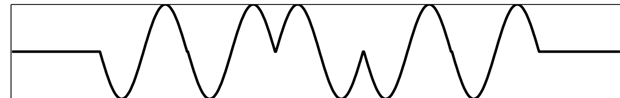
FPGAs allow the implementation of very fast custom electronics hardware that is reconfigurable. The performance provided by FPGAs is far greater than software implemented on a DSP, and yet FPGAs are flexible and low–cost compared with application–specific–integrated–circuits (ASICs). The Xilinx® ML405[3] Evaluation Platform was chosen for the on–board Ethernet controller, Virtex-4 FPGA, which includes multi–gigabit–transceiver (MGT), on–chip PowerPC, and the systems $\$1k$ low–cost price. The MGT is a versatile port capable of many different communications standards such as Gigabit Ethernet and PCI Express™. RocketIO™ is Xilinx's implementation of an MGT.

Radar chirps are generated using a custom configuration of the MGT, as opposed to standard communications protocols, such as gigabit-ethernet. Data describing the

[3]More information available at `http://www.xilinx.com`.



(a) Digital output from FPGA's MGT port.



(b) Ideal analog output after RF signal conditioning of digital signal.

Figure 4-5: Illustration of signal conversion from digital to RF.

chirp pattern is loaded into the MGT and then shifted out serially as a phase–shift encoded digital signal. The digital signal is filtered, conditioned, amplified, and then transmitted via the antenna by the custom RF electronics. The ideal digital to RF transformation is illustrated by figure 4-5.

The PowerPC embedded processor in the FPGA runs software written in C to communicate with the user interface via Ethernet. The user software running on a remote PC is able to collect data from the system as well as to control the specialized FPGA radar chirp–generating logic interfaced with the MGT. Using an embedded processor has the advantage of flexibility in implementation of control and data collecting algorithms, while keeping the high-speed signal processing implemented in FPGA logic. The PowerPC software is easy to quickly change without the time consuming compilation of FPGA logic from VHDL source code.

Several software tools are necessary to develop systems using Xilinx FPGAs. A suite of tools contained in the Xilinx Embedded Development Kit (EDK) is organized through the Xilinx Platform Studio (XPS) integrated development environment. XPS runs external tools that configure the PowerPC and system buses, compile the soft-

ware running on the embedded processors, compile the VHDL code to configure the FPGA, and generate the bit-level code used to program the FPGA with the software and hardware configuration. Another collection of Xilinx® software tools is packaged as the ISE Foundation integrated development environment. ISE provides low-level FPGA layout and planing tools, many of which can be called from XPS. The FPGA simulation software provided by Xilinx® was rudimentary. Thus the industry standard simulation suite, called ModelSim®, developed by Mentor Graphics®, was used for simulating before and after the FPGA code was compiled[4].

The ML405 development board loads FPGA configuration bit–files and programs to run on the embedded processor from the compact flash card. Once the FPGA code is compiled using either XPS or ISE, the generated `project_file.bit` file needed to be converted to a `project_file.ace` with the `ml40x_bit2ace` program. The `project_file.ace` is copied to the compact flash card in the `/ml405/ace/` directory and is loaded when the "My own ACE file" is select from the boot loader menu. See references [54] and the section "My Own ACE File" in reference [55] for detailed information.

4.2 Analog Components and Test Equipment

This section briefly details the variety of test equipment and electronic components used in this project. A custom printed circuit board (photos and schematics are given in Appendix D), was designed and fabricated to allow testing of the SAW devices. RF amplifiers and design components were purchased for modularly prototyping the many test configurations covered in chapter 5. Cables, connectors, and power supplies needed to be carefully selected, as this system is very sensitive to noise and losses

[4]Developing FPGA code is very complicated with many different stages to implementation. Readers are encouraged to consult with [51, 52, 53]

at high–frequency. The electronics design for both testing and implementation is preliminary. While results are being gathered, there is still more work to be done on the electronic design. The components listed here will be added to as the testing and development progresses.

The most important piece of equipment for this project is a high–speed and very sensitive digital–sampling–oscilloscope (DSO). A fast oscilloscope is necessary is to accurately capture the transponder's return responses with enough resolution to allow for proper signal processing. A fast DSO was also necessary to debug signals at the operating frequency of $2.4GHz$, and to pinpoint unwanted parasitic oscillations at even higher frequencies. Capturing the subtle higher–order return signals from even the slower $323MHz$ SAW transponders is critical for proper characterization and understanding of these designs. Observing the return signal is essential for the design, implementation, and debugging of the RF receiver. Before the receiver is designed the DSO will allow us to complete functional testing and evaluation of the system, including time–of–flight calculations.

A LeCroy WaveMaster 8500A DSO with $5GHz$ of bandwidth was purchased in the spring of 2007 for this project. With a DSO capable of a 20 giga–samples–per–second rate, evaluation of the round–trip time–of–flight to accuracies of approximately $10cm$ should be practical. A vertical sensitivity of $2mV/division$ with $8bits$ of resolution made this the most sensitive oscilloscope we could afford and will exceed the requirements of a demonstration system. The sensitivity of the DSO limited the input voltage range to $\pm 4V$. This limitation required attenuators for some of the larger signals. This scope also has a direct interface to MATLAB®, allowing real–time control of this oscilloscope as a highly configurable RF–digitizer. The wide bandwidth allowed inspection of oscillations on the low–noise–amplifier that appeared to be noise on previously available DSOs. The data capture and report generating features have

greatly increased the efficiency of data collection, analysis, and debugging.

An Agilent 8714ET RF Network Analyzer was used for measuring and compensating for input transducer impedance as well as measuring the transmissibility of the SAW filters. This was a relatively old unit ($> 10yrs$) with no record of calibration. One device was characterized with this unit, and then measured on a newer and recently calibrated RF Network Analyzer, and the results were reasonable close. Details on the use of the RF network analyzer for impedance matching are given in section 4.3. The section 4.4 covers testing the SAW filter transmissibility with RF Network Analyzer.

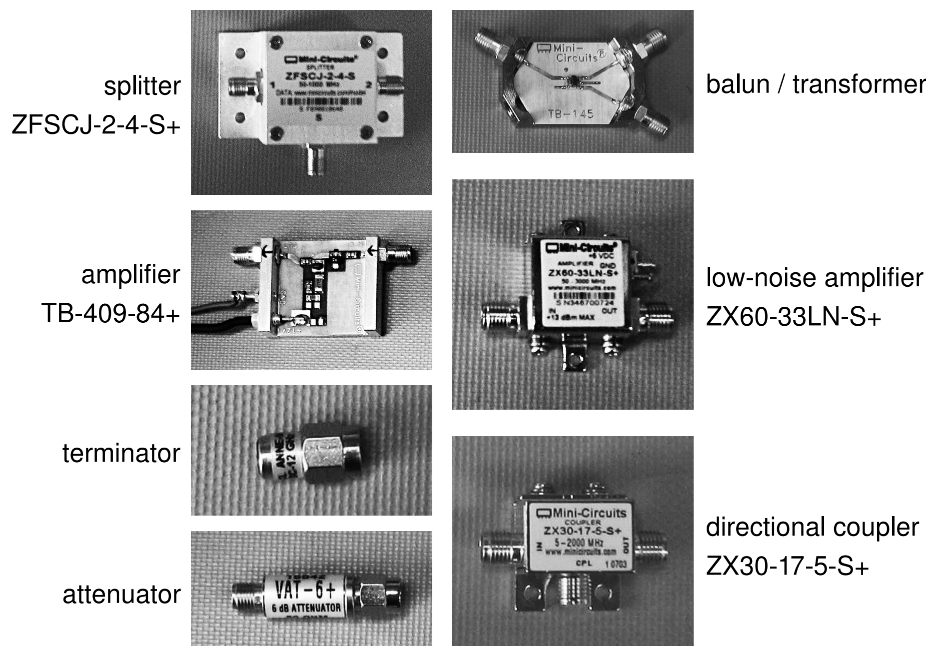


Figure 4-6: Photo of analog components from Mini–Circuits®.

Several packaged RF prototyping circuits were purchased from Mini–Circuits®. Low–noise amplifiers were needed for the RF receiver front–end and power amplifiers were need for the RF transmitter. Several amplifiers were purchased based on availability and price. A selection of couplers were necessary for interfacing devices

Part Number	**Description**
TB-409-84+	Test board with Gali–84+ DC to $6GHz$, $25.6dB$ of gain amplifier with $38dBm$ output.
TB-145	Transformer test board.
ZX30-17-5-S+	Directional Coupler, DC to $2GHz$.
ZX60-33LN-S+	Low–noise amplifier with $20dB$ of gain and output power up to $17.5dBm$.
ZFSCJ-2-4-S	Power Splitter / Combiner, 2 way 180°, 50 to $1000MHz$.
K1-TMC+	Designer's transformers kit.
K2–VAT+	Designer's fixed attenuator kit. DC to 6GHz, 1 to 10dB.
ANNE-50L+	50Ω DC to $12GHz$ Attenuators.

Table 4.1: Components used in this project that were purchased from Mini–Circuits®.

to various test equipment; more details are provided in section 4.4 & 4.5. Transforms and baluns were used for decoupling signals and matching signals such as the RocketIO™ output to the RF transmitter impedance. The transformers and baluns were purchased in a kit, and then the desired one was installed on the Mini–Circuits® test board TB–145. A few 50Ω terminators and a set of attenuators were purchased to terminate unused connections and attenuate large signals for test equipment, especially the DSO. We found that at the time Mini–Circuits® was the most cost effective source of these components in SMA–compatible packages. RF filters have not yet been purchased but will be required as the measurement system is developed. See table 4.1 for a summary of components.

In addition to providing a carrier for the SAW devices, the custom PCB was created to amplify the minute signals as close to the devices a possible. The idea was to have the amplifier close to the device so the signals could be amplified with as little noise as possible added before amplification. Unfortunately, the very fast $18GHz$ transistors that were selected caused oscillations in the design. These problems were

not worked out at the time of writing this thesis. A possible solution to removing the oscillations would be to add a ferrite bead to the base of the transistor, which will be tried soon.

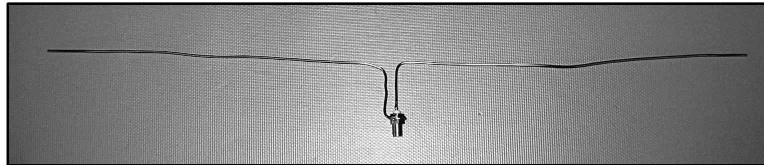


Figure 4-7: $323MHz$ antennas made from $16\text{–}AWG$ copper wire and an SMA connector. Used for preliminary isolated indoor tests.

Several other components and auxiliary equipment were used for this project. A low–noise fixed linear power supply from Power–One®, part number HAA15-0.8-A, was used to power many of the Mini–Circuits® active components. Custom $323MHz$ antennas were built, as pictured in figure 4-7, for a few isolated indoor tests. Other antennas were purchased for use on the $915MHz$ instrumentation–scientific–and–medical (ISM) band. A small $58mm$ Helical Antenna, part number WTH430003 from S.P.K Electronics Co. LTD., was purchased along with a $236mm$ monopole antenna, part number WAN-00514-D8 from Wanshih Electronics Co. LTD. 50Ω SMA patch cables were purchased for their low–loss SMA connectors and RG–316 coax cable. Many other discrete components, such as inductors and capacitors used for impedance matching, were selected for low–loss/high–Q. Surface mount inductors from EPCOS, series B82498, were used for most implementations in this project. NPO/COG rated capacitors were used for their low–loss. Selection of discrete components for RF design is very important and the reader is referred to [3].

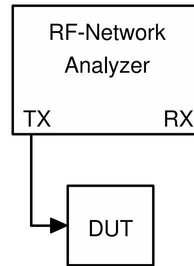


Figure 4-8: RF Network Vector Analyzer test setup for reflectance and impedance measurements.

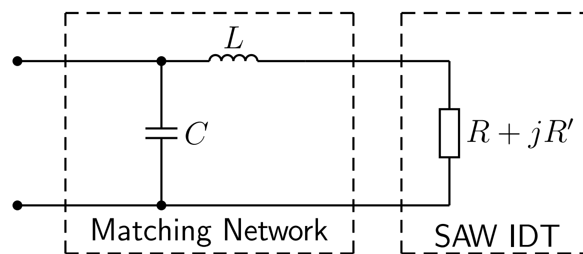


Figure 4-9: Simple L matching network [3, 4].

4.3 Impedance Matching

Matching the characteristic impedance at higher frequencies is very important. Any of a number of problems, such as reflections and oscillations, may occur without a proper impedance match[3, 56]. Loss of power due to an improper match can also distort the characterization of a design. The IDTs of SAW devices are normally capacitive, thus a matching network must be added to bring the input port back to 50Ω. The procedure was to measure the real and complex impedances of the input IDT with the RF network vector analyzer using the setup shown in figure 4-8. The measured complex impedance was then used to determine the values of components needed to make a an impedance match. The match was made by using an inductor and capacitor in an L–match network configuration that is added to the front of the input port of the IDT (as shown in figure 4-9). Low–loss (high-Q) high–frequency components were necessary to make the match without signal degradation[3, 4].

Calculating the correct values is straight forward and may be done with a Smith chart or by solving a system of equations. The equivalent resistance looking into the network is a simple voltage divider with L and $R + jR'$ in parallel with C (equation 4.1). By setting this to our desired characteristic impedance, R_{match}, and solving, we are given the proper values for L and C. Equation 4.1 is split into a real and imaginary equations 4.2 and 4.3 and then solved numerically.

$$\frac{\frac{1}{sC}(sL + R + jR')}{\frac{1}{sC} + (sL + R + jR')} \tag{4.1}$$

$$j\frac{\omega L + R'}{\omega R_{\text{match}} R} = jC \tag{4.2}$$

$$\frac{R - R_{\text{match}}}{R_{\text{match}}(-\omega^2 L - \omega R')} = C \tag{4.3}$$

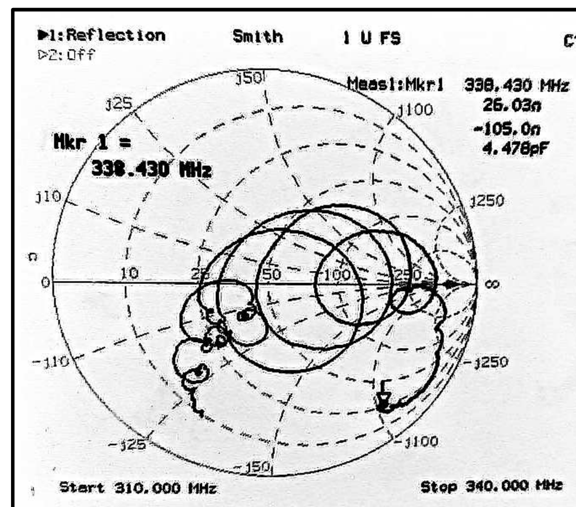


Figure 4-10: Measured Smith chart on the RF Network Vector Analyzer of a SAW transponder.

One problem with measuring the characteristic input impedance of a SAW IDT is that there are reflections and resonances due to the device characteristics that appear to cause problems with measurements taken with the RF Network Vector Analyzer.

Figure 4-10 shows the measurement taken of a simple SAW transponder on mask 1. The curly Qs are supposedly from a combination of reflections from input fingers and off of the downstream reflectors. One way around this that has yet to be explored (for lack of access to the proper equipment), is making the measurements instead with a time–domain–reflectometer. Another possibility that will be tried is to make a bare input IDT that is not connected to anything (i.e. an output IDT as in the case of a SAW filter, or coupled reflectors as in the case of a transponder) for measuring the input impedance of similar transducers.

4.4 SAW Filter Measurements

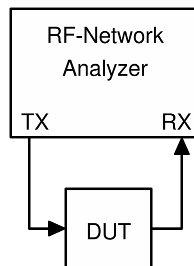


Figure 4-11: RF Network Vector Analyzer test setup for Transmittance measurements.

Transmissibility measures the amount of loss between a transmitter and receiver at a particular frequency through a black box. The transmissibility of the SAW filters was measured to determine correct function as well as the loss and frequency response of the device. Transmissibility measurements were also used to determine how well the impedance matching adjustments worked. A better match yields lower transmission losses at the pass–band frequencies. The same RF Network Vector Analyzer as was used to determine input impedance was used for transmissibility measurements. Figure 4-11 shows the setup used, and figure 4-12 gives an example of

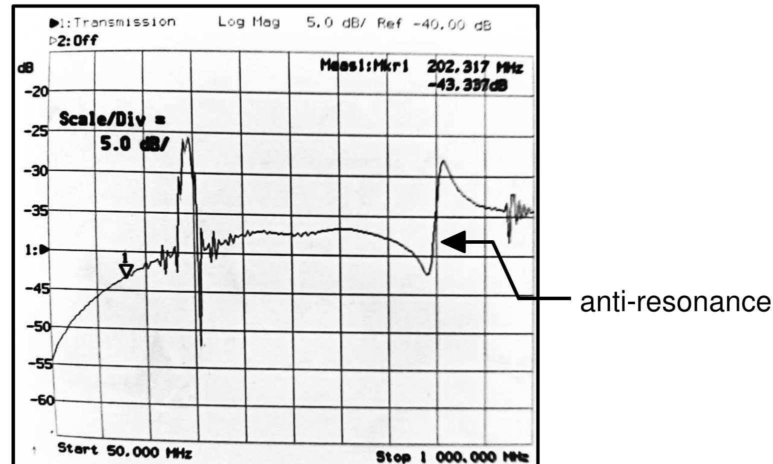


Figure 4-12: Measured transmission chart of a SAW filter on the RF Network Vector Analyzer. Notice the anti–resonance artifact, which is ignored for this work.

a measured response. For convenience, a logarithmic magnitude scale was selected.

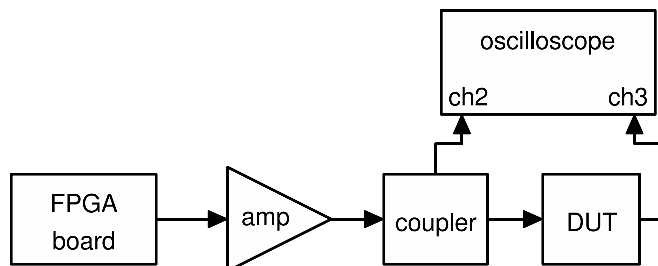


Figure 4-13: Setup for transient measurements of a SAW filter using an oscilloscope.

The setup shown in figure 4-13 was used to measure the impulse response and the chirp response of a SAW filter. Both measurements helped with understanding SAW IDT operation and characterization of the devices. The impulse response was taken by generating a pulse at the resonance frequency with the FPGA. One channel of the oscilloscope measures and triggers off of the transmitted impulse. The response was measured on the other channel.

4.5 SAW Transponder Measurements

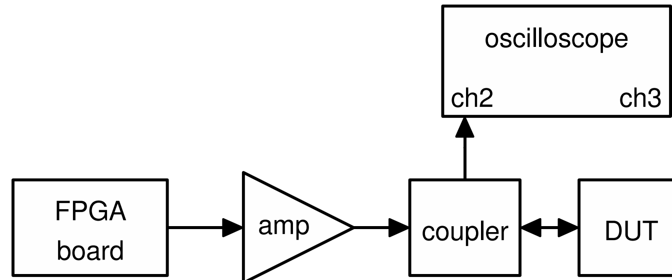


Figure 4-14: Test setup for SAW transponder measurements.

The early $323MHz$ SAW transponders have only one port from which to interrogate and measure the return response. A directional coupler[5] allows triggering off of the FPGA–generated chirp pulse while observing the transponder's return signal. The setup is shown in figure 4-14. There are some problems with the FPGA's transmitted signal saturating the input of the oscilloscope. Solutions to these and other measurement problems are in the works.

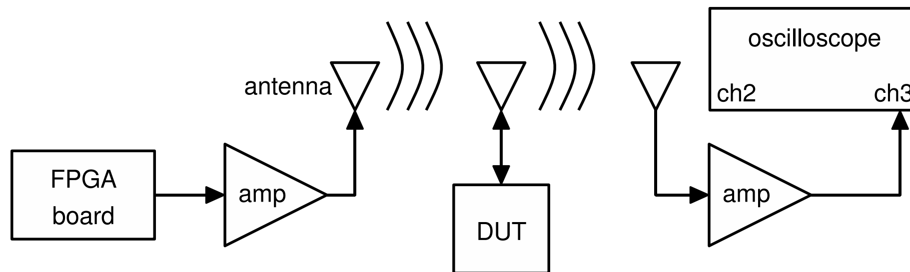


Figure 4-15: SAW transponder measurements using antennas.

Measuring of the SAW transponders in their intended configuration, with the transponder connected to an antenna, is shown in figure 4-15. The operation is similar to the previous setup, except the coupler is replaced by three antennas. Some

[5]Mini–Circuits® has a few useful application notes on directional couplers (`http://www.minicircuits.com/`)

input filtering (not shown) after the receiver amplifier will probably be necessary. The best receiver design is currently being investigated.

4.6 FPGA Hardware Description

The internal logic of an FPGA can be specified through several different languages (such as VHDL, verilog, even MATLAB) or with high–level schematic capture programs. VHDL was one of the first FPGA programming languages and resembles ADA. The project used VHDL exclusively, based on personal preference. Many other FPGA languages would have worked equally as well.

VHDL 1 : Example of IO pad description used VHDL and the Xilinx UNISIM library.

```
1  ...
2
3  ---- Xilinx primitives
4  library UNISIM;
5
6  use UNISIM.VComponents.all;
7
8  ...
9
10   -- ************************************************
11   IBUFGDS_usrclk : IBUFGDS
12     generic map (
13       IBUF_DELAY_VALUE => "0",
14       IOSTANDARD       => "LVPECL_25",
15       DIFF_TERM        => TRUE)
16     port map (
17       O  => usrclk_b,
18       I  => usrclk_ip,  -- Diff_p clock
19       IB => usrclk_in   -- Diff_n clock
20       );
21
22
23   -- ************************************************
24   IBUF_reset : IBUF
25     generic map (
26       IBUF_DELAY_VALUE => "0",
27       IFD_DELAY_VALUE  => "AUTO",
28       IOSTANDARD       => "LVTTL")
29     port map (
30       O => reset_n_b,
31       I => reset_n
32       );
33 ...
```

Developers take many different approaches to organizing their system designs[57]. The VHDL source for this project was organized to be hierarchically tailored for two configurations, simulation and implementation. The top-level simulation files are

known as "testbenches", and are denoted by the naming convention of `filename_tb.vhd`. A testbench provides simulated simulation signals to the VHDL code that will become the description for the real hardware as well as checking the outputs. The implementation's top-level source file is denoted by `filename_pad.vhd`, which represents the physical "pads" of the FPGA's package. The `filename_pad.vhd` file uses Xilinx device family dependent logic black–boxes to select specifically how an input or output pad should be configured. For example, the `IBUFGDS` and `IBUF` are both black boxes in VHDL listing 1. The pad configuration parameters can also be specified using a constraints file, however specifying the configuration in a `filename_pad.vhd` allows further compatibility among tools from different vendors, such as ModelSim™. Only VHDL code that describes the entire implementation uses the `filename_pad.vhd` file. The chirp–generating implementations that do not use the PowerPC have a `filename_pad.vhd` file.

The VHDL code specifies the physical connections between Configurable Logic Blocks (CLBs), the basic building block of an FPGA. Programming the CLBs configures them as flip–flops, encoders, and other logic elements, which are connected in a fixed configuration implementing the desired functionality . There are also many different types of specialized fixed ASIC components on an FPGA, such as the MGTs and PowerPC core. The proper way to correctly specify the connections and topology to imply specifically the hardware desired must be done by following vendor–specific VHDL coding standards. Often these specification are provided in a separate "Synthesis Manual," which may be provided by the FPGA vendor and/or the simulation and synthesis vendor (i.e. Mentor Graphics® or Simplicity®). For example, the FPGA provided by Xilinx® may have the VHDL configuration code compiled by Xilinx® software, or Precision Synthesis™ from Mentor Graphics®, or similar product from Synopsis®. Following the synthesis specification provided by the vendor of the

tool being used for the particular FPGA selected is essential for an accurate simulation and successful implementation[51]. The UNISIM VHDL libraries provide all the primitive black–boxes along with the timing parameters for accurate simulation and are supplied by Xilinx®.

As mentioned earlier, there are several increasingly complicated implementation of the chirp–generating FPGA code. The first chirp–generating implementation did not use the MGT or a graphical user interface. For simplicity, the buttons on the ML405 development board increased or decrease the number of cycles in a chirp by five. The chirp output was a gated clock limited to 400MHz chirps without any encoding functionality. A gated clock is a poor implementation because the first and last cycle of a chirp would be randomly truncated, causing degraded performance. The output frequency was fixed by the clock synthesizer.

In the next design iteration the MGT configuration was added. The MGT allowed phase encoded chirps to be transmitted at the maximum speed capable of the FPGA development board. The output chirp was hard–coded for simplicity (the configurable implementation was to use the PowerPC processor and Ethernet controller) and could only be changed by recompiling the FPGA source code, a time consuming task.

The internal FPGA processor architecture is very flexible and very complex, with several different buses, memory architectures, bridges, input–output controllers, and processors (MicroBlaze™ and PowerPC™). Figure 4-16 shows a simplified depiction of the organization of the FPGA's embedded processor bus as configured for this implementation. The PowerPC has direct access to on–chip SRAM configured for both program instructions and program data. High–speed peripherals are connected directly to the PowerPC on the processor local bus. While all peripherals can be placed on any bus, good design practices separate high–speed components that only communicate with the processor from slower peripherals that may communicate with

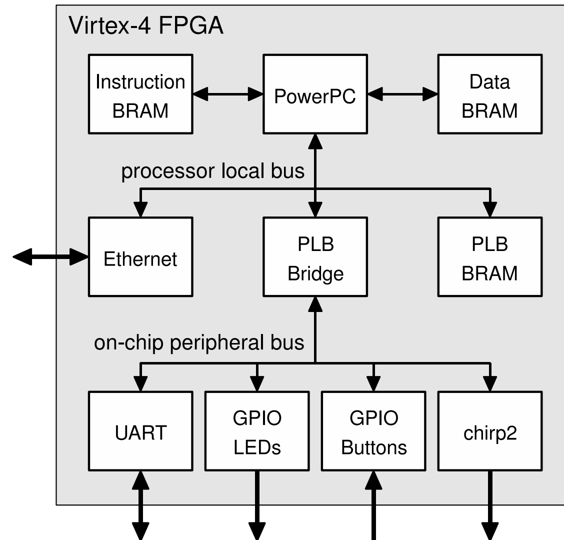


Figure 4-16: Block diagram of FPGA PowerPC processor configuration.

each other as well as the processor.

The processor local bus (PLB) bridge connects the PowerPC to the on–chip peripheral bus (OPB). The *"UART"* allows the program running on the PowerPC to send simple debug messages via RS–232 to a terminal. Debugging problems with Ethernet communication was greatly aided by the *"UART"*. The general purpose input output (GPIO) peripherals are generated automatically via a peripheral wizard in XPS and can be connected to anything. The *"GPIO LEDs"* peripheral was configured to set LEDs as another simple debug status output. The *"GPIO buttons"* allowed the program running on the processor to check the buttons on the ML405 development board. The only custom peripheral was *"chirp2"*, the second implementation which transmits the phase–encoded chirp.

The organization of the *"chirp2"* IP is shown in figure 4-17. Many of these source files were automatically generated skeleton template files from XPS that were later appended and modified as needed. The `chirp2.vhd` and `user_logic` sources were automatically generated by the XPS peripheral wizard. They provide a register transfer

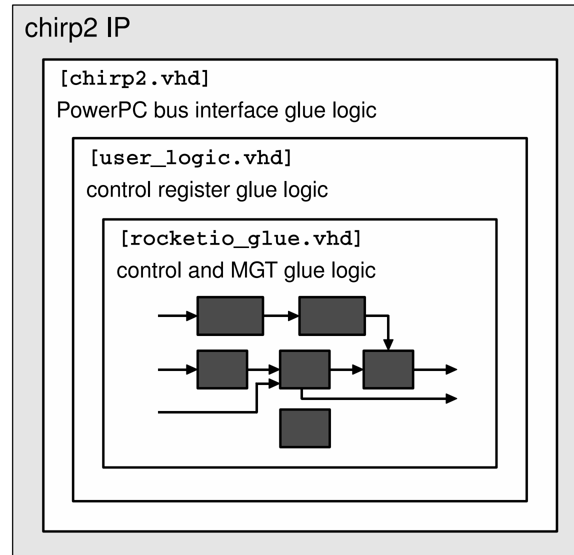


Figure 4-17: Top level block diagram of chirp2 VHDL code.

interface to the PowerPC processor that allows the program to write to registers that configure and control this peripheral. These files must be modified to bring input and output signals to the custom peripheral. All of the custom logic is placed in the `user_logic.vhd` source file.

Specialized hardware, or "cores", in the FPGA are configured with a Xilinx® tool called CoreGen. Optimized implementation of complex functions, such as a fast–fourier–transform, are also provided through CoreGen as black–boxes. The MGT was configured through CoreGen, which generated a VHDL template from all of the setting specified in the CoreGen Wizard. These files are then integrated into the design by instantiating the provided component code into a higher level component. Figure 4-17 shows the MGT configured by CoreGen integrated into the chirp2 IP wrapped by the `rocketio_glue.vhd` source file. The CoreGen generated code needs several modifications to work properly in this design. The clock distribution and connections were not setup as required. Details on setting up the MGT clocks is out of the scope of this thesis and requires careful study of [58].

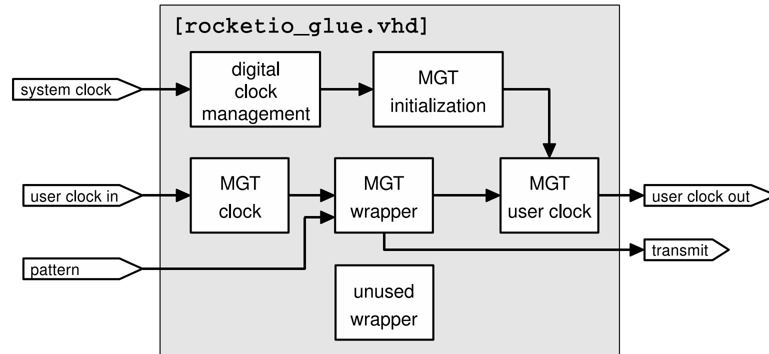


Figure 4-18: VHDL code organizational block diagram of the *"chirp2"* peripheral implementing the rocketIO multi–gigabit–transceiver and support FPGA cores.

All of the custom peripheral management code as well as the implementation of the MGT are contained in `rocketio_glue.vhd`. This code makes all the proper connections between the MGT hard–wired cores and the higher–level custom logic. VHDL listing fragment 2 shows the custom logic that periodically sends the configured chirp by loading the output register with data when the `counter` reached `delay_time`. The majority of the `rocketio_glue.vhd` is shown in block diagram 4-18. The *"MGT clock"* black–box buffers the user clock (set by the external VCO), separating the clock domains, and phase locking to the MGT user clock. Synchronization with adjacent MGT blocks as well as converting the MGT clocks to the proper frequency is also handled by the *"MGT clock"*. The MGT wrapper encapsulates the hundreds of RocketIO configuration parameters to simplify the interface for higher–level components. Most of the complexity of the MGT configuration is due to the hardware implementation of different high–speed encoding and decoding standards. The *"user clock out"* signal is generated to synchronize the high–speed parallel data to the rest of the FPGA logic. The *"digital clock management"* (DCM) buffers and reduces the clock rate of the *"system clock"* to drive the MGT power–on reset and initialization hardware. The receiver is not connected in this implementation. The MGT hardware comes in pairs of transceivers. The *"unused wrapper"* component is

VHDL 2 : `txdata_p` process in `rocketio_glue.vhd`

```
1  -- purpose: provides data to MGT0_TX
2  -- type    : sequential
3  -- inputs  : usr_clk_i, reset
4  -- outputs:
5  txdata_p: process (usr_clk_i, tx_system_ready_i, chirp_pattern, delay_time)
6
7  begin -- process txdata_p
8    if tx_system_ready_i = '0' then                   -- asynchronous reset (active low)
9
10     counter       <= (others => '0');
11     mgt0_txdata_i <= (others => '0');
12
13   elsif usr_clk_i'event and usr_clk_i = '1' then  -- rising clock edge
14
15     if counter >= delay_time then
16
17       counter       <= (others => '0');
18       mgt0_txdata_i <= chirp_pattern;
19
20     else
21
22       counter <= counter + 1;
23       mgt0_txdata_i <= (others => '0');
24
25     end if;
26
27   end if;
28 end process txdata_p;
```

the unused transceiver in the same ASIC MGT block as the utilized MGT [58].

Some of the status and logic indicators were brought out to the LEDs to help with debugging. System ready, clock management phase locking, reset signals, and buffer ready status conditions were monitored via the LEDs.

Normally every level of VHDL code is simulated individually to check correct operation, and simulated as a system to check integration. The complexity of the MGT made simulation infeasible due to the complexity of setup and amount of time to run. Sometime compiling the VHDL code and running on the FPGA, with carefully chosen debug signals made available to test ports, is the best way to debug an implementation. Other options for debugging code are available, such as ChipScope™ from Xilinx, however they were not within our budget.

4.7 Software

There are two software platforms currently under development. The PowerPC on the FPGA runs an Internet server that services requests from other devices communicating via the TCP/IP sockets port. A PC runs a user interface that allows changes to the transmitted pattern and frequency of periodic chirp. The software was developed with the help of an undergraduate, Roshni Cooper, as a part of the Undergraduate Research Opportunities Program (UROP)[59]. While the code is usable, and has basic functionality required to configure the chirp peripheral, much more work will be done as this project progresses. Future software development will allow MATLAB® running on the LeCroy oscilloscope, through the Java sockets library, to signal the FPGA to transmit a particular code. MATLAB® will then trigger the oscilloscope and collect and analyze the data from the return RF signal.

The current implementation has the embedded PowerPC processor running a very simple program to service requests from the user interface. The source code is located in Appendix E. There are three libraries and associated source code used for this program;

- `enet.c` : A customized high–level Ethernet interface which calls the Xilinx provided network APIs.
- `xromlcd.c` : Provides the interface to the ML405's LCD display. This source code was a part of some Xilinx demo code.
- `chirp2.c` : Interface device driver for the chirp2 peripheral.

The top level source file is `chirpnet.c`. First the *"chirp2"* and *"Ethernet"* peripherals are initialized. Then the main program loop services connection requests with `enet_accept()`. Up to `MAX_NET_CONNS` from different computers are allowed.

This feature will be useful when multiple measurement stations must communicate to triangulate a tag's position. Commands are processed and parsed with the `enet_proc_conns()` function, which checks each connection and executes any pending requests. Only one command, "`chirp2,delay,high,low;`", has been implemented; where delay is an integer specifying the number of cycles between chirp transmissions, and high and low give the binary code for the chirp in two 16*bit* words.

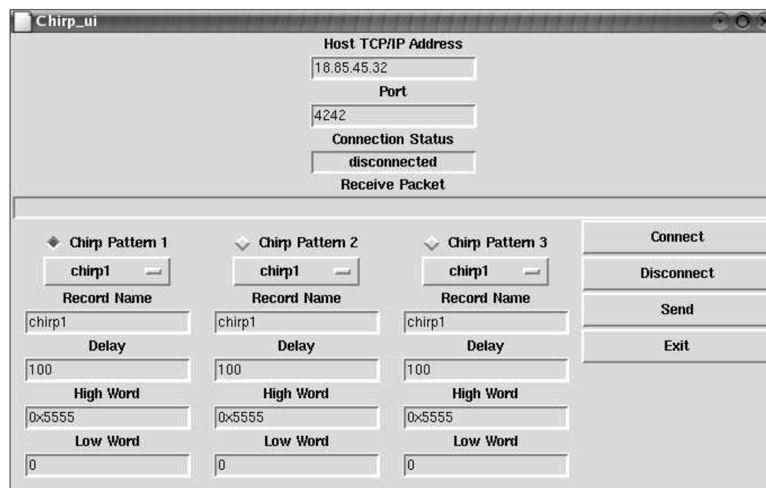


Figure 4-19: `chirp_ui.pl` User interface program written in PERL.

The simple user interface, shown in figure 4-19, communicates over Ethernet with a TCP/IP sockets library. Programmed in PERL, the interface is portable, simple, and easily extended. The IP address and the port number of the measurement station is entered under "Host TCP/IP Address" and "Port." Two buttons are provided to connect and disconnect to the measurement station. Many patterns may be specified by the three columns of configurations, which are stored on disk for later use. The selected pattern is sent to the measurement station when the "Send" button is pressed.

The user interface, while functional and in use, is preliminary, and therefore the full source code is not provided. However the most important code fragments are given in PERL listing 1. A TCP/IP socket is opened to the measurement station

with `IO::Socket::INET->new( ... )`. The handle `$sock` is then written to, read from, and closed the same as any file handle. The `set_chirp( ... )` subroutine demonstrates how the "`chirp2,delay,high,low;`" command is assembled and sent.

PERL 1 : Code fragments illustrating how to use TCP/IP sockets.

```
1  use IO::Handle;
2  use IO::Socket;
3  use Net::hostent;
4  use POSIX qw/setsid/;
5  use subs qw/client_connect/;
6
7  #-------------------------------------------------------
8  sub host_connect {
9    $sock = IO::Socket::INET->new( PeerAddr => $host,
10                                  Proto    => 'tcp',
11                                  PeerPort => $port );
12   $sock->autoflush( 1 );
13 }
14
15 #-------------------------------------------------------
16 sub host_disconnect {
17   return unless defined $sock;
18   close( $sock );
19 }
20
21 #-------------------------------------------------------
22 sub set_chirp {
23   my $delay = shift;
24   my $high  = shift;
25   my $low   = shift;
26   host_send( "chirp2,$delay,$high,$low" );
27 }
28
29 #-------------------------------------------------------
30 sub host_send {
31   my $data = shift;
32   print $sock $data."\n";
33 }
34
35 #-------------------------------------------------------
36 sub host_recv {
37   my $data = '';
38   $sock->recv( $data,128 );
39   return $data;
40 }
```

Chapter 5

Testing and Characterization

All of the results presented here are very preliminary. The $915MHz$ devices of interest have not yet been successfully fabricated (these devices should ready withing the first two months of the PhD program). The purpose of the data taken thus far is to confirm that the design is on the right track. To check our first simplified designs, the $3\mu m$ fabrication process, and the test setup, some data was taken. Until recently, we did not have the necessary test equipment (oscilloscope, amplifiers, RF prototyping components, etc.) therefore very little presentable data was collected. However, the following is useful as an illustration of the operation of three SAW devices along with some insight into their performance.

5.1 SAW Device `0x2C9`

Device `0x2C9` was a band–pass filter with a reasonably good filter response. See table 5.1 for the detailed specifications. Figure 5-1 shows the CAD artwork (layout was done in MAGIC). This is a bi–directional device, and for the purposes of this data, one port is the input and one is the output. The transmissibility (measurement setup

is explained in section 4.4) shows very good band–pass filtering with a pass–band of $\approx 20MHz$ (figures 5-2 and 5-3). A simple impedance match was tried and seemed to work. As shown in figures 5-4 and 5-5, the matching network appears to improve the impedance. The pass–band loss also improved about $3dB$ after the match was made. One possible explanation for the curlicues shown in figure 5-5 is that the resonant reflections were confusing the RF Network Vector Analyzer (more investigation is underway).

Figure 5-6 shows the effect of internal back–and–forth reflections between the input and output IDTs. The magnitude of the internal reflections may be in part due to an impedance mismatch. The data is taken at the output port, and a short 4–cycle chirp is sent into the input port. The first signal that is detected is unwanted feed–through from the input to output port. The main signal is the largest waveform to be received at the output. The signals that show up after the main signal are caused by additional reflections off of the input and output IDTs. These macro reflections are a set of waves bouncing between the input and output IDTs. There are also intra–finger reflections in each IDT, which are shown in figure 5-7. The secondary reflections are discussed in section 2.2.

IDT line–width and spacing	$3\mu m$
IDT fingers	50
IDT Length	$200\mu m$
IDT Apodization	$191\mu m$
IDT Rail Width	$20\mu m$
Pad width and height	$200\mu m$
Input to output separation	$1050\mu m$

Table 5.1: Specifications for device `0x2C9`

Figure 5-1: Drawing of SAW filter device 0x2C9

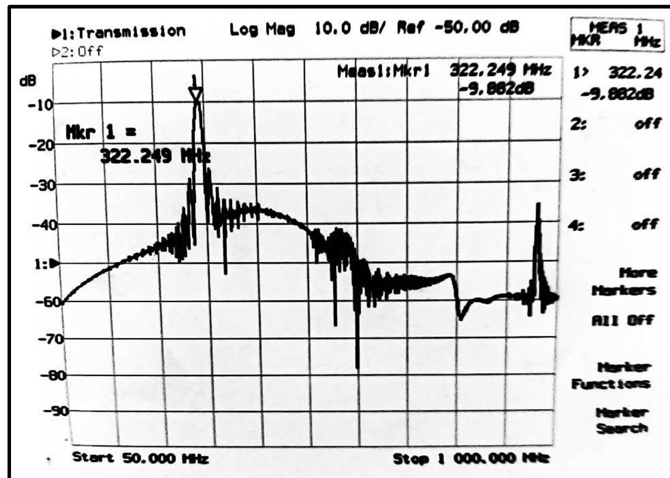


Figure 5-2: Frequency response from $50MHz$ to $1GHz$. The pass band is at $322.249MHz$.

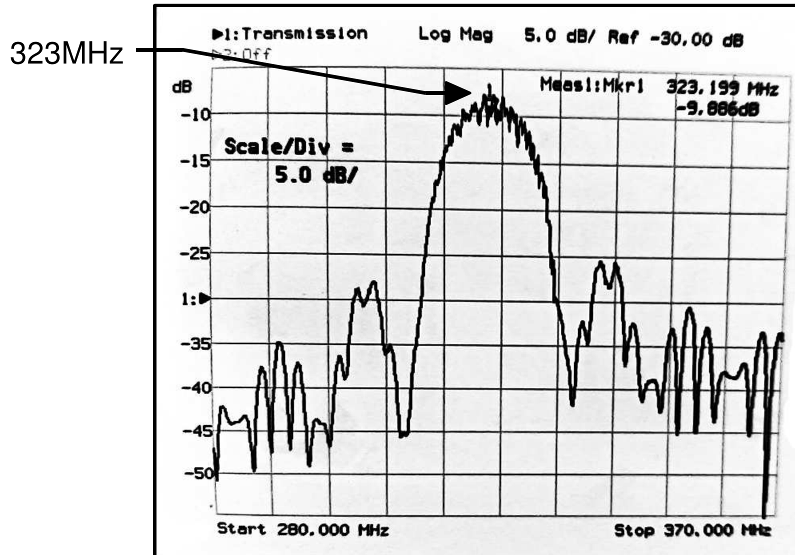


Figure 5-3: Pass–band bandwidth shown as $\approx 20MHz$.

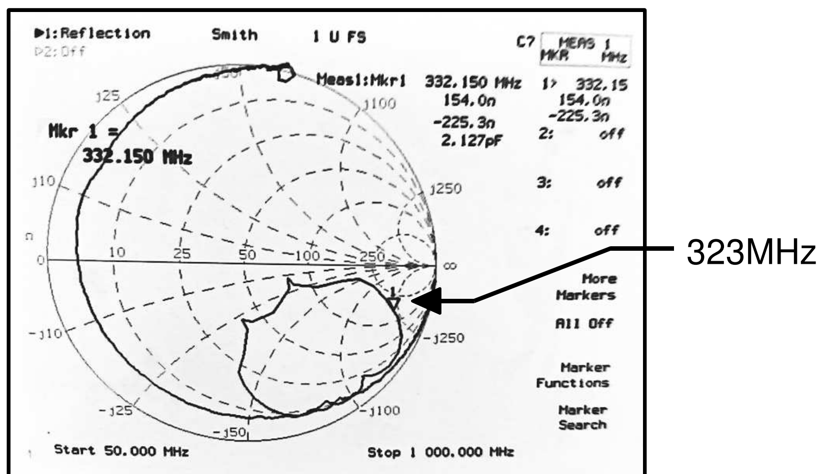


Figure 5-4: Smith chart before impedance matching circuit.

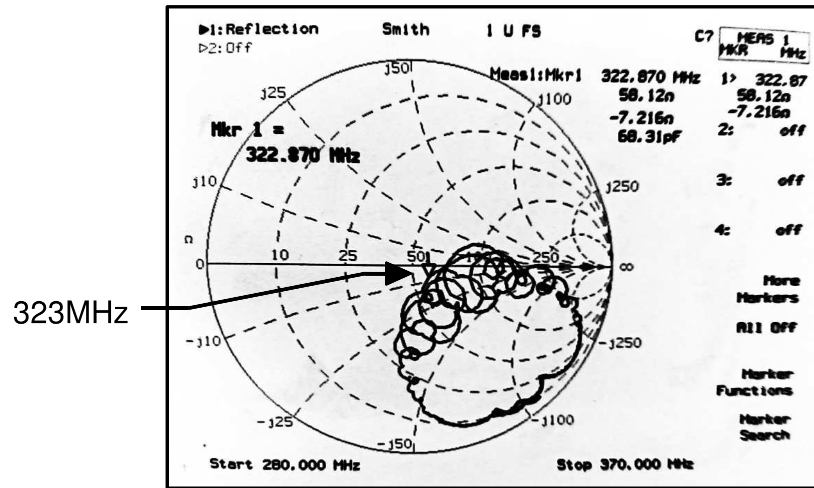


Figure 5-5: Smith chart after impedance matching circuit with $27nH$ and $10pF$.

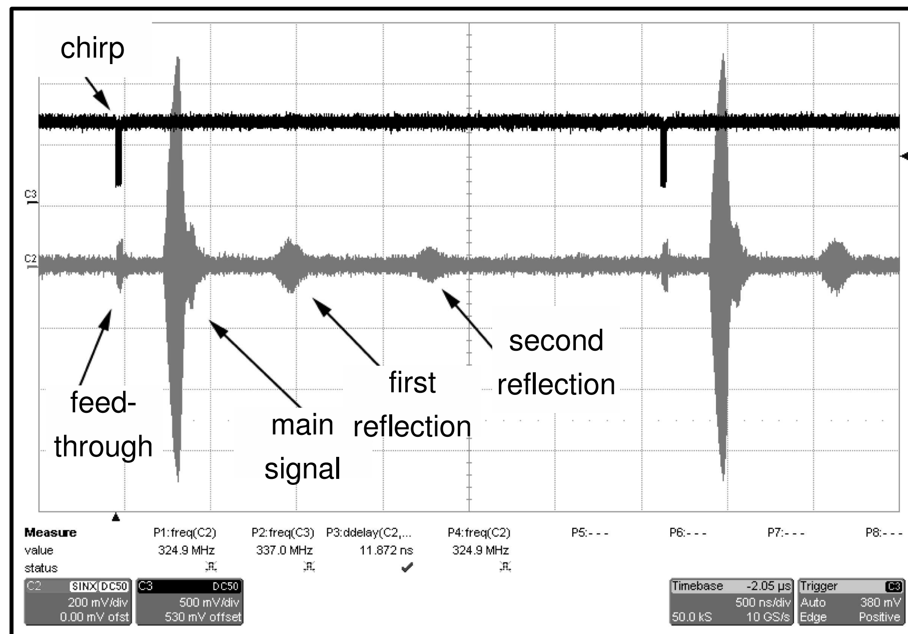


Figure 5-6: Internal reflection between two IDTs in device 0x2C9.

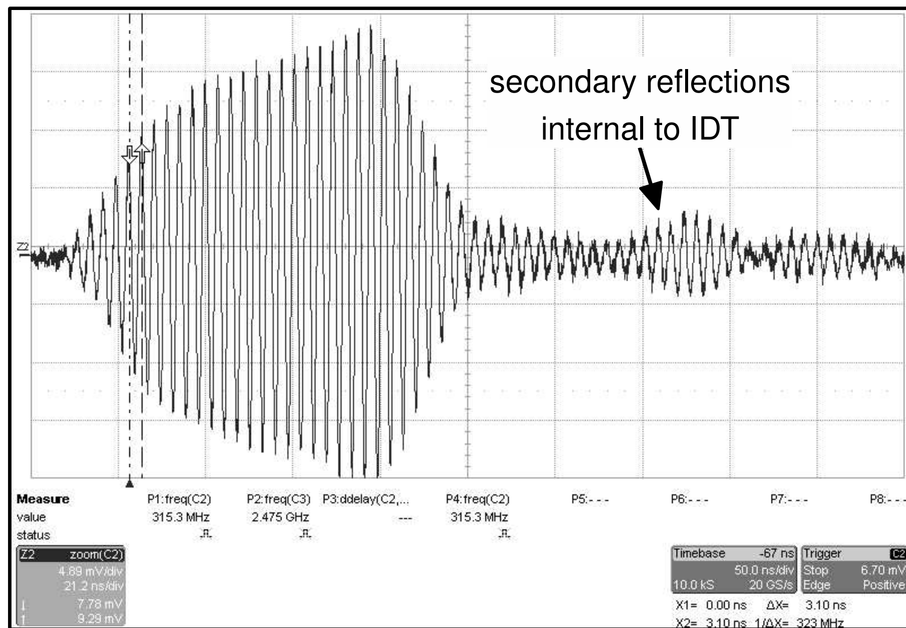


Figure 5-7: Example of inter–finger reflections within a single IDT.

5.2 SAW Device 0x2E8

With only 20 fingers and half the apodization (IDT finger overlap) of device 0x2C9, device 0x2E8 has a much lossier response. Not as much data was collected as for 0x2E8 due to lack of time. Figure 5-9 shows the measured band–pass responses as well as signs of electromagnetic feed–through, which is the coupling of the input signal to the output via electromagnetics[5]. The impulse response shown in figure 5-10 has been labeled to show the relationship of the output signal to the number of fingers in the device. For this device, 20 fingers or 10 finger pairs on each IDT yields a 20cycle output.

IDT line–width and spacing	$3\mu m$
IDT fingers	20
IDT Length	$100\mu m$
IDT Apodization	$91\mu m$
IDT Rail Width	$20\mu m$
Pad width and height	$200\mu m$
Input to output separation	$360\mu m$

Table 5.2: Specifications for device 0x2E8

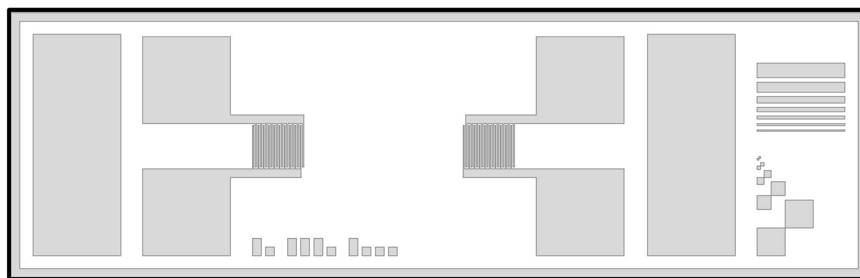


Figure 5-8: Drawing of SAW filter device 0x2E8

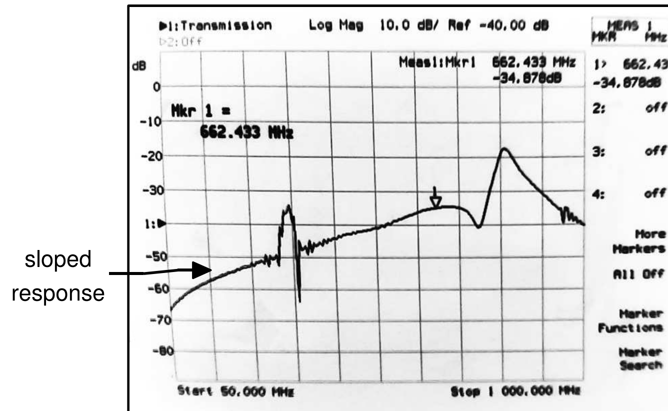


Figure 5-9: The sloped frequency response is due to electromagnetic feed–through[5].

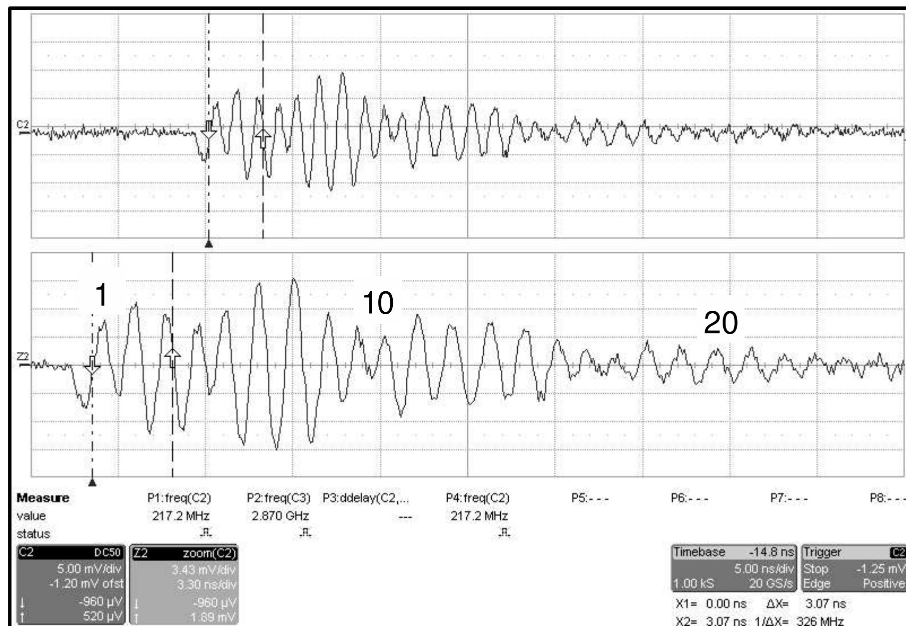


Figure 5-10: Impulse response of SAW filter Device `0x2E8`. The number of cycles has been labeled (1,10,and 20). Channel Z2 is the zoomed view of Channel C2.

5.3 SAW Device 0x0E3

One of the four saw reflectors that were tested is presented here. 0x0E3 was the only SAW reflector device that was tested with the fast $5GHz$ oscilloscope. Limited data was taken on three other SAW reflectors with a $500MHz$ oscilloscope. Due to the inadequate bandwidth of the $500MHz$ oscilloscope, that data is not worth presenting. Figure 5-11 show the device with five sets of reflector gratings. The response from an 11–cycle chirp signal is shown in figure 5-12. Two groups of reflected signals are shown, though more are there, just out of view. Only the reflections from the first three gratings are measurable, as too much of the signal is lost by the time the surface wave reaches the fourth and fifth reflector. The delay between the interrogation chip and the round trip to and back from the first reflector is $171ns$. This delay corresponds to the surface velocity and distance between the input IDT and first grading. The surface velocity is $\approx 4000m/s$ and the distance between the gratings is $300\mu m$. The very first return response will start at $150ns$. However the first detectable response will be shortly after because more than one wave will need to couple into the IDT. As shown in figure 5-12, the time until signal is significant is $171ns$, as expected.

IDT line–width and spacing	$3\mu m$
IDT fingers	50
IDT Length	$300\mu m$
IDT Apodization	$291\mu m$
IDT Rail Width	$20\mu m$
Pad width and height	$200\mu m$
Reflector spacing	$300\mu m$

Table 5.3: Specifications for device 0x0E3

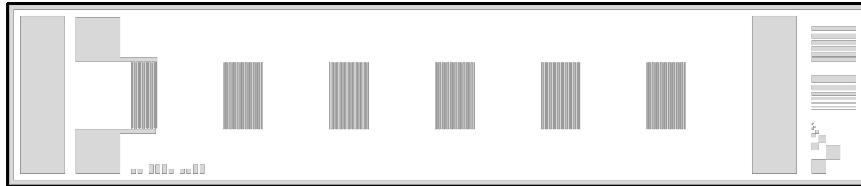


Figure 5-11: Drawing of SAW filter device 0x0e3

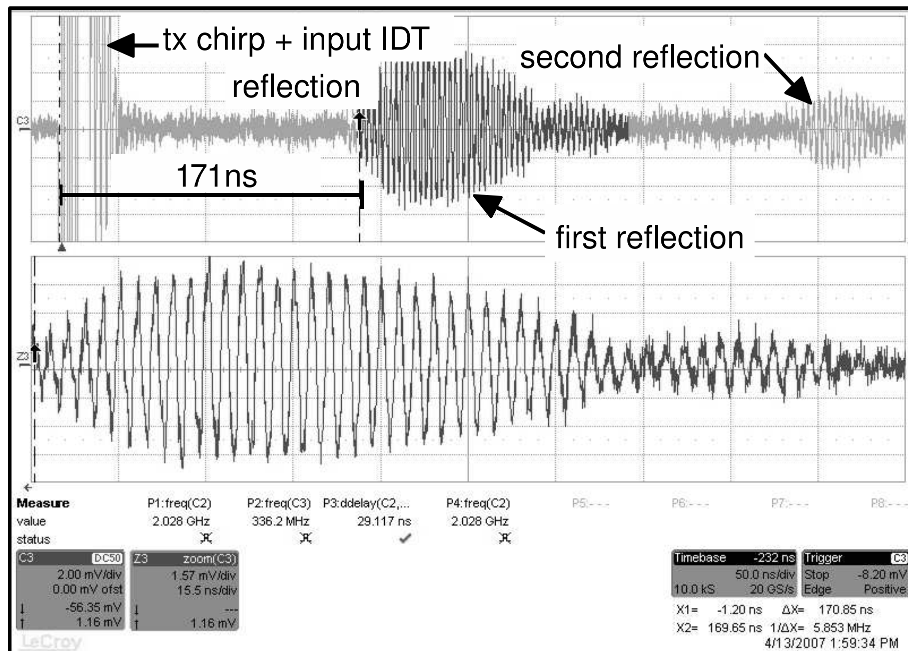


Figure 5-12: The four cycle chirp response of device 0x0E3. Channel Z3 is the zoomed view of the first reflection on Channel C3.

Chapter 6

Conclusion

From the beginning we knew this project was very ambitious. Micro–fabrication projects are particularly difficult to complete over the short duration of a masters degree. Additional challenges to this project were the high–frequency electronics design and testing. An an enormous amount has been accomplished and learned in the two years since the start of this project. Extensive investigations into the state–of–the–art of existing solutions and an analysis of probable performance of SAW devices were completed. The idea of a SAW based transponder with ranging capabilities was conceptualized and a patent was filed on the technology. The implementation of the SAW transponders is now progressing quickly and should conclude with positive data and results soon.

Significant progress has been made understanding the properties of lithium niobate and utilizing this substrate for SAW applications. The first simple models of SAW operation were developed, implemented, and verified with our initial devices. The designs of new devices, currently being fabricated, were based on these models. Three masks were drawn using two CAD tools, "layout" and "Magic." The foundation has been laid for quickly implementing new designs using the tools and methods developed

in the thesis.

The most difficult aspect of this project was attaining proficiency in microfabrication of SAW devices. The majority of the problems were due to inexperience with the complicated processes and laboratory methods. The use of numerous tools were learned and successfully utilized. Amorphous silicon dioxide masks and lithium niobate SAW substrates were selected and purchased based on our requirements. Cleaning and preparation processes were learned and modified for our uses, allowing us to recycle wafers from unsuccessful runs, saving a significant amount of money.. I became proficient in the use of two lithography aligners, one of which significant upgrades were made to meet requirement of this project. Knowledge necessary to use the scanning electron beam lithography tool has allowed inspection and writing of masks. After debugging, the lift–off processes we achieved reliable fabrication of devices. $3\mu m$ devices were successfully fabricated. Several $3inch$ quartz masks with $1.1\mu m$ and $300nm$ devices on them were successfully fabricated. Debugging of the mask making process is nearly complete. The tools and hardware necessary for mounting the SAW devices to suitable packages and wire–bonding them to the leads has been figured out and implemented. A lot of work as been accomplished in learning the details of microfabrication necessary to make our desired devices. The foundation for fabricating SAW transponders has been made, and the skill necessary for making our desired devices is ready for the final implementation.

The architecture necessary for the electronics test suite was designed and developed to a point where devices are able to be tested. A prototype test suite has been assembled, with an FPGA signal source, supporting high–frequency electronics such as low–noise amplifiers, and a high–speed oscilloscope capturing return signals and debugging. Significant progress has been made in verifying that the electronics will work for the ultimate goals of this project. The FPGA hardware and software

has been developed sufficiently to generate the required signals. An Ethernet capable user interface program was written to communicate and control the FPGA. Impedance matching of the SAW devices to the standard 50Ω electronics has been figured out. Topologies for testing the various SAW devices has been prototyped and has collected some useful data. The equipment acquired should be nearly everything necessary to attain our initial results for ranging measurements. Data taken thus far with the electronics looks encouraging and good results are expected soon.

Basic testing of devices made to date has been accomplished, and the results are reasonable. The test setup and methods for testing SAW filters has given us result from four $3\mu m$ devices. Frequency and impulse responses of the $3\mu m$ devices have been measured and correspond to the expected results. The effect of impedance matching the SAW IDTs has been verified to be an improvement. The response of several $3\mu m$ reflector devices have also been investigated. Initial test results show that the available test equipment, such as the RF Network Vector analyzer and oscilloscope, are adequate.

The work left to be done before ranging results can be attained are as follows. Fabrication will continue with making the $915MHz$ and $2.4GHz$ devices. The processes to make the devices have been developed and only need to be tweaked before working devices are made. Once the high–frequency devices are made, the electronics suite will be put to the test. Some significant debugging is expected and different approaches to testing the devices will be explored. Testing of all of the different devices will take place after the electronics have been fully debugged. Following the collection of the test data, a paper containing the results will be published. The end goal of this project is to transfer SAW transponder ranging and localization technology to industry.

Localization using small passive transponders has excited several sponsors of the

MIT Media Laboratory. The usefulness of the project is evident in the amount of work being put into other less elegant solutions to the tracking problem, summarized in section 1.3. Successful completion of this project will solve many tracking problems for which there is currently no practical solution.

Bibliography

[1] K. Finkenzeller, *RFID Handbook*. West Susses, England: John Wiley & Sons, second ed., 2003.

[2] A. Slobodnik, E. Conway, , and R. Delmonico, *Microwave Acoustics Handbook*, vol. 1A. Surface Wave Velocities. L.G. Hanscom Field, Bedford, Massachusetts: Air Force Cambridge Research Laboratories, 1973.

[3] T. H. Lee, *Planar microwave engineering: a practical guide to theory, measurement, and circuits*. Cambridge UK: Cambridge University Press, 2004.

[4] D. M. Pozar, *Microwave Engineering Third Edition*. Hoboken, NJ: John Wiley & Sons, Inc., 2005.

[5] C. K. Campbell, *Surface Acoustic Wave Devices for Mobile and Wireless Communications*. San Diego: Academic Press, 1998.

[6] *OmniSTAR 8400HP User Manual*, (`http://www.omnistar.com.au`).

[7] G. Loegering, "A performance comparison between a SCAT–1 DGPS system and the Omnistar DGPS system," *Position Location and Navigation Symposium*, vol. 48, pp. 310–314, APRIL 2004.

[8] C. Shuxin, W. Yongsheng, and C. Fei, “A study of differential GPS positioning accuracy,” in *Microwave and Millimeter Wave Technology 3rd International Conference on*, vol. 17, pp. 361 – 364, August 2002.

[9] *Carios information*, (`vdg@iis.fraunhofer.de`).

[10] A. Pohl, R. Steindl, and L. Reindl, “The “intelligent tire” utilizing passive SAW sensors measurement of tire friction,” *Instrumentation and Measurement, IEEE Transactions on*, vol. 48, no. 6, pp. 1041–1046, December 1999.

[11] R. W. Brocato, “Passive microwave tags,” *Sandia National Laboratories*, no. SAND2004–4924, 2004.

[12] R. W. Brocato, “Programmable SAW development,” *Sandia National Laboratories*, no. SAND2004–5255, 2004.

[13] L. Reindl, “Wireless passive SAW identification marks and sensors,” in *2nd Int. Symp. Acoustic Wave Devices for Future Mobile Communication Systems*, (Freiburg, Germany), pp. 1–15, Institute for Microsystem Technology, Albert-Ludwigs-University Freiburg, March 2004.

[14] H. Matthews, ed., *Surface Wave Filters: Design, Construction, and Use.* New York: John Wiley & Sons, 1977.

[15] H. Smith, “Fabrication techniques for surface–acoustic–wave and thin–film optical devices,” *Proceedings of the IEEE*, vol. 62, pp. 1361–1387, October 1974.

[16] *RF Code*, (`http://rfcode.com/`).

[17] *RF SAW*, (`http://rfsaw.com/`).

[18] M. I. Skolnik, *Introduction to Radar Systems.* Boston Massachusetts: McGraw Hill, 1980.

[19] J. D. Taylor, ed., *Introduction to Ultra–wideband Radar Systems*. Boca Raton: CRC Press, 1995.

[20] A. V. Oppenheim and R. W. Schafer, *Discrete–time signal processing*. New Jersey: Prentice–Hall Inc., 1998.

[21] L. Reindl, G. Scholl, T. Ostertag, H. Scherr, U. Wolff, and F. Schmidt, "Theory and application of passive SAW radio transponders as sensors," *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, vol. 45, no. 5, pp. 1281–1292, 1998.

[22] B. K. Sinha, "A stress and temperature compensated orientation and propagation direction for surface acoustic wave devices," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 34, no. 1, pp. 64–74, 1987.

[23] M. Rlinami, H. Morikawa, and T. Aoyama, "An adaptive multipath mitigation technique for gps signal reception," *IEEE Vehicular Technology Conference Proceedings*, vol. 2, no. 15-18, pp. 1625–1629, 2000.

[24] C. J. Comp and P. Axelrad, "Adaptive SNR-based carrier phase multipath mitigation technique," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 34, no. 1, pp. 264–276, 1998.

[25] R. E. Phelts and P. Enge, "Multipath mitigation for narrowband receivers," *IEEE Position Location and Navigation Symposium*, no. 13-16, pp. 30–36, 2000.

[26] W.-J. Chang and J.-H. Tarng, "Effects of bandwidth on observable multipath clustering in outdoor/indoor environments for broadband and ultrawideband wireless systems," *Vehicular Technology, IEEE Transactions on*, vol. 56, no. 4, pp. 1913–1923, 2007.

[27] D. C. K. Lee, K. W. Sowerby, and M. J. Neve, "Extracting fine multipath detail from measured data at 5.8 GHz," *IEEE Vehicular Technology Conference*, vol. 1, no. 17-19, pp. 74–78, 2004.

[28] P. C. Ching and H. C. So, "Two adaptive algorithms for multipath time delay estimation," *Oceanic Engineering, IEEE Journal of*, vol. 19, no. 3, pp. 458–463, 1994.

[29] C.-T. Chen, *Linear system theory and design.* New York: Oxford university press, third ed., 1999.

[30] P. Maes, T. Darrell, B. Blumberg, and A. Pentland, "The ALIVE system: wireless, full-body interaction with autonomous agents," *Multimedia Syst.*, vol. 5, no. 2, pp. 105–112, 1997.

[31] J. Hightower and G. Borriello, "Location systems for ubiquitous computing," *Computer, IEEE*, vol. 34, no. 8, pp. 57–66, 2001.

[32] A. Y. Benbasat and J. A. Paradiso, "An inertial measurement framework for gesture recognition and applications," in *Gesture Workshop*, pp. 9–20, 2001.

[33] A. Smith, H. Balakrishnan, M. Goraczko, and N. Priyanthae, "Tracking moving devices with the cricket location system," in *2nd International Conference on Mobile Systems, Applications and Services (Mobisys 2004)*, (Boston, MA), June 2004.

[34] A. Savvides, M. Srivastava, L. Girod, and D. Estrin, *Wireless sensor networks, chapter Localization in sensor networks.* Norwell, MA, USA: Kluwer Academic Publishers, 2004.

[35] *Chipcon Application Note AN042 – CC2431 Location Engine*, (`http://www.chipcon.com/`).

[36] *Ubisense*, (`http://www.ubisense.net/`).

[37] M. Penza, G. Cassano, P. Aversa, F. Antolini, A. Cusano, A. Cutolo, M. Giordano, and L. Nicolais, "Recognition of organic solvents molecules by simultaneous detection using SAW oscillator sensors and optical fiber devices coated by Langmuir–Blodgett Cadmium Arachidate films," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 58, no. 8, pp. 1493–1502, 2006.

[38] R. W. Brocato, "Passive wireless sensor tags," *Sandia National Laboratories*, no. SAND2004–1288, 2006.

[39] D. E. Smalley, Q. Y. J. Smithwick, and V. M. Bove Jr., "Holographic video display based on guided-wave acousto-optic devices," in *Proceedings of SPIE - Practical Holography XXI*, vol. 6488, February 2007.

[40] *Thoronics*, (`http://www.thoronics.com/`).

[41] K. K. Wong, ed., *Properties of Lithium Niobate*. London, UK: INSPEC, 2002.

[42] K. K. Berggren and H. I. Smith, *Class Notes for MIT Course 6.781 "Submicrometer and Nanometer Technology"*. Cambridge Massachusetts: Massachusetts Institute of Technology, 2007.

[43] J. D. Plummer, M. D. Deal, and P. B. Griffin, *Silicon VLSI Technology*. Australia: Pearson Education International, 2000.

[44] J. G. Goodberlet, "Patterning 100nm features using deep-ultraviolet contact photolithography," *Applied Physics Letters*, vol. 76, no. 6, pp. 667–669, 2000.

[45] J. G. Goodberlet and B. L. Dunn, "Deep-ultraviolet contact photolithography," *Microelectronic Engineering*, vol. 53, pp. 95–99, 2000.

[46] W. Kern, *Handbook of Semiconductor Wafer Cleaning Technology – Science, Technology, and Applications*. William Andrew Publishing/Noyes, 1993.

[47] J. L. Vossen and W. Kern, *Thin Film Processes*. New York: Academic Press Inc., 1978.

[48] T. Bailey, B. J. Choi, M. Colburn, M. Meissl, S. Shaya, J. G. Ekerdt, S. V. Sreenivasan, and C. G. Willson, "Step and flash imprint lithography: Template surface treatment and defect analysis," *Vacuum Science Technology*, vol. 18, no. 6, pp. 3572–3576, 2000.

[49] C. P. Fucetolao, "Resolution limits and process latitude of conformable contact nano-lithography," Master's thesis, Massachusetts Institute of Technology, 2007.

[50] S. Berek, U. Knauer, and H. Zottl, "I-line lithography for highly reproducible fabrication of surface acoustic wave devices," *SPIE Optical/Laser Microlithography IV*, vol. 1463, pp. 515–520, 1991.

[51] *Virtex-4 User Guide*. San Jose, CA: Xilinx®, 2006.

[52] *EDK 9.1 Documentation*. San Jose, CA: Xilinx®, 2006.

[53] *ModelSim® SE User Manual*. Wilsonville, Oregon: Mentor Graphics®, 2007.

[54] *ML405 User Guide*. San Jose, CA: Xilinx®, 2007.

[55] *ML40x Getting Started Tutorial*. San Jose, CA: Xilinx®, 2006.

[56] P. R. Gray, P. J. Hurst, S. H. Lewis, and R. G. Meyer, *Analysis and design of analog integrated circuits*. New York: John Wiley & Sons, Inc., 2001.

[57] P. J. Ashenden, G. D. Peterson, and D. A. Teegarden, *The Systems Designer's Guild to VDHL–AMS*. San Francisco California: Morgan Kaufmann, 2003.

[58] *Virtex-4 RocketIO Multi-Gigabit Transceiver User Guide*. San Jose, CA: Xilinx®, 2006.

[59] R. Cooper, "Initial implementation of the μTags localization transponders," Master's thesis, Massachusetts Institute of Technology, 2007.

[60] K. Pahlavan, X. Li, and J.-P. Mäkelä, "Indoor geolocation science and technology," *Communications Magazine, IEEE*, vol. 40, no. 2, pp. 112–118, 2002.

[61] C. S. Hartmann, D. T. Bell Jr., and R. C. Rosenfeld, "Impulse model design of acoustic surface–wave filters," *Sonics and Ultrasonics, IEEE Transactions on*, vol. 20, pp. 80–93, April 1973.

[62] P. Ventura, J. M. Hodé, M. Solal, J. Desbois, and J. Ribbe, "Numerical methods for SAW propagation characterization," *IEEE Ultrasonics Symposium*, pp. 175–186, 1998.

[63] K. Uehara, C.-M. Yang, T. Shibata, S.-K. Kim, S. Kameda, H. Nakase, and K. Tsubouchi, "Fabrication of 5–GHz–Band SAW filter with atomically–flat-surface $A1N$ on sapphire," *IEEE Ultrasonics Symposium*, vol. 1, no. 23-27, pp. 203–206, 2002.

[64] F. Moeller, A. Rabah, M. A. Belkerdid, and D. C. Malocha, "Differential phase shift keying direct sequence spread spectrum single SAW based correlator receiver," *IEEE Ultrasonics Symposium*, pp. 189–193, 1994.

[65] J. Temmyo and S. Yoshikawa, "On the fabrication and performance of SAW delay

line filters for GHz SAW oscillators," *Sonics and Ultrasonics, IEEE Transactions on*, vol. 25, no. 6, pp. 367–371, 1978.

[66] L. B. Milstein and P. K. Das, "Surface acoustic wave devices," *Communications Magazine, IEEE*, vol. 17, no. 5, pp. 25–33, 1979.

[67] R. W. Brocato, "A zero–power radio receiver," *Sandia National Laboratories*, no. SAND2004–4610, 2004.

[68] K.Yamanouchi, G.Shimizu, and K.Morishitat, "2.5 GHz-range SAW propagation and reflection characteristics and application to passive electronic tag and matched filter," *IEEE Ultrasonics Symposium*, pp. 1267–1270, 1993.

[69] Skeie et al, "Surface acoustic wave passive transponder having acoustic wave reflectors," *United States Patent* 4, 625, 208, November 25, 1986.

[70] Hartmann et al, "Single phase unidirectional surface acoustic wave transducer and improved reflectors," *United States Patent* 7, 173, 360, February 6, 2007.

[71] Lanzl et al, "Dual mode tracking system," *United States Patent* 6, 353, 3406, March 5, 2002.

[72] Nysen et al, "Passive interrogator label system having offset compensation and temperature compensation for a surface acoustic wave transponder," *United States Patent* 4, 734, 698, March 29 1988.

[73] Skeie, "Surface acoustic wave passive transponder having amplitude and phase-modifying surface pads," *United States Patent* 4, 625, 207, November 25, 1986.

[74] Nysen et al, "System for interrogating a passive transponder carrying phase-encoded information," *United States Patent* 4, 725, 841, February 16, 1988.

[75] Epstein, "Passive transponders using acoustic surface wave devices," *United States Patent* 4,059,831, November 22, 1977.

[76] L. Reindl, C. C. W. Ruppel, S. Berek, U. Knauer, M. Vossiek, P. Heide, and L. Oreans, "Design, fabrication, and application of precise SAW delay lines used in an FMCW radar system," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 49, no. 4, pp. 787–794, 2001.

[77] U. Wolff, F. L. Dickert, G. K. Fischerauer, W. Greibl, and C. C. W. Ruppel, "SAW sensors for harsh environments," *IEEE Sensors Journal*, vol. 1, pp. 4–13, June 2001.

[78] J. Bachrach and C. Taylor, *Book Chapter: Localization in Sensor Networks*. Cambridge Massachusetts: Computer Science and Artificial Intelligence Laboratory at Massachusetts Institute of Technology, 2004.

[79] D. Smalley and T. Pastureaud, "An eigenmode method for fast cascade of P-matrices," Master's thesis, SAW Research and Development, TEMEX Microsonics, 2005.

[80] S. Lindebner, H. H. Frühauf, J. Heubeck, R. Wansch, and M. Schühler, "Evaluation of direction of arrival location with a $2.45GHz$ smart antenna system," in *Proceedings of Signals, Sensors and Devices*, (Sousse), 2005.

[81] K. Yamanouchi, G. Shimizu, and K. Morishita, "2.5 ghz-range saw propagation and reflection characteristics and application to passive electronic tag and matched filter," in *Ultrasonics Symposium, 1993. Proceedings., IEEE 1993*, vol. 2, pp. 1267 – 1270, October 1993.

[82] L. Reindl and W. Ruile, "Programmable reflectors for SAW-ID-tags," in *Ultrasonics Symposium, IEEE Proceedings*, vol. 1, pp. 125 – 130, October 1993.

[83] D. P. Morgan, "Reflective array modeling for saw transducers," in *Ultrasonics Symposium, 1995. Proceedings., 1995 IEEE*, vol. 1, pp. 215 – 220, November 1995.

[84] *OPB PCI v1.02.a User Guide*. San Jose, CA: Xilinx®, 2006.

[85] *PowerPC 405 Processor Block Reference Guide*. San Jose, CA: Xilinx®, 2007.

Appendix A

Lithography Masks

A.1 Mask 1 Generation Scripts

Mask 1 was generated using TCL scripts located in the

`./msk/saw.1_3.10.2006/scripts/`

directory. The `make_all.tcl` script was used to layout the majority of the mask. This script uses the `saw.tcl` library of procedures. After the `make_all.tcl` script was run, some manual touch–up was necessary. First the devices were bumped around to make room for a horizontal alignment mark. Then the devices near the edges of the mask were duplicated in the remaining area. The devices on the left side were rotated and flipped so their common edge was towards the center of the mask. The `make_string.tcl` script was then run to make the title block at the bottom of the mask. This script makes use of a `char.glyph` data file which is generated by the `make_char.pl` script. Postscript documentation was automatically created. Some bugs in Magic 7.3 necessitate the use of a `fix.pl` script to fix the postscript output. A comma–separated–variable listing of all the parameters for each device is output to the `specs.csv` file. Note that there is an error in the `make_all.tcl` script. When

the first 9 saw devices are made the `REF_Apodization` is not set correctly for the first device. The half–fingers are not shorted correctly. This was not worth the time to correct before making the mask.

TCL : `make_all.tcl`

```
1  #==================================================
2  #
3  #       Make All : Make the mask
4  #
5  #       filename : make_all.tcl
6  #       authors  : Jason LaPenta (JL)
7  #       date     : 3/4/2006 (JL)
8  #       target   : Magic v7.3
9  #            [http://opencircuitdesign.com/magic/]
10 #       version  : 0.3 alpha (branch 0)
11 #
12 #       abstract : generates parameterized SAW
13 #                  gratings and test features
14 #
15 #==================================================
16 # Mask Layout
17 #
18 #            ------------- <- (70mm,70mm)
19 #            |     |     |
20 #            |     |     |
21 #            |     |     |
22 #            |     |     |
23 #            |     |     |
24 #            |     |     |
25 # (0,0) ->   -------------
26 #
27 #  the bottom corner of 7 is (0,0)
28 #  the mask is 4" square, with an inset to
29 #  the artwork of 15mm on each side, giving
30 #  a square of 70x70mm for the artwork
31 #==============================================
32 #
33 #
34 # Changes :
35 # Date      Who   Description
36 # ------------------------------------------------
37 # 2/12/06   JL    New File
38 # 3/5/06    JL    Added in dual port saws
39 # 3/6/06    JL    redid layout to pack many
40 #                 devices in a 70mm x 70mm area
41 #-------------------------------------------------
42
43 # saw creation library
44 source saw.tcl
45
```

```
46
47  #=================================================
48  #  next
49  #
50  #  moves to the next available space on the die
51  #=================================================
52
53  proc next { } {
54
55      global offset_x
56      global offset_y
57      global id
58
59      incr id
60
61      set pos [magic::box pos]
62      set size [magic::box size]
63
64      set x [lindex $pos 0]
65      set y [lindex $pos 1]
66
67      set w [lindex $size 0]
68      set h [lindex $size 1]
69
70      # we've run out of room, stop moving
71      if { $y < $h && $offset_x > [expr 70000/2] } {
72          puts "\n\n_****_OUT_OF_ROOM_***\n\n"
73          return
74      }
75
76      # check if last device fits
77      # if not then move
78      if { $x + $w > [expr (70000 / 2) + $offset_x - 2000] } {
79          magic::select visible
80          magic::move to $offset_x [expr $y - $h - $offset_y]
81          magic::box move east [expr $w + $offset_y]
82      } else {
83          magic::box move east [expr $w + $offset_y]
84      }
85
86      if { $y < 2*$h } {
87          set offset_x [expr 70000/2 + 1000]
88          magic::box position $offset_x [expr 70000 - 1500]
89      }
90
91  }
92
93
94
95  #=================================================
96  #  output
97  #
98  #  makes postscript outputs
99  #=================================================
```

```
100
101 proc output { x y bw bh id } {
102
103     # limit width so documentation is readable
104     set bw2 $bw
105     if { $bw2 > 4000 } {
106         set bw2 4000
107     }
108
109     magic::box position $x [expr $y - 450]
110     magic::box size $bw2 [expr $bh + 450]
111     magic::plot postscript saw_[format "%\#05x" $id].ps {metal1 labels}
112
113     # delete documentation labels
114     magic::box size $bw2 450
115     magic::select visible
116     magic::delete
117
118     magic::box position $x $y
119     magic::box size $bw $bh
120 }
121
122 #==================================================
123 #  make_cell
124 #
125 #  x     - x position of first finger
126 #  y     - y position of first finger
127 #  h     - height of surrounding box
128 #  w     - width of surrounding box
129 #==================================================
130
131 proc make_cell { } {
132
133     global saw_spec
134
135     # save the box position for later
136     set pos  [magic::box position]
137     set x [lindex $pos 0]
138     set y [lindex $pos 1]
139
140     set s       $saw_spec(IDT_Spacing)
141     set w       $saw_spec(IDT_Width)
142     set l       $saw_spec(IDT_Length)
143     set a       $saw_spec(IDT_Apodization)
144
145     set lw      $saw_spec(Leg_Width)
146     set po      $saw_spec(Pad_Offset)
147     set pw      $saw_spec(Pad_width)
148     set ph      $saw_spec(Pad_height)
149
150     set in_f    $saw_spec(IN_Fingers)
151     set r_f     $saw_spec(REF_Fingers)
152     set r_s     $saw_spec(REF_Spacing)
153     set r_n     $saw_spec(REF_Number)
```

```
154     set s_l     $saw_spec(Stop_Length)
155
156     set id      $saw_spec(ID)
157
158     # calculate the box width of the surrounding box
159     # 50um of padding
160     set pad_x 50
161     # width of the test features
162     set test_w 250
163
164     set bw [expr 2*$pad_x + $pw + $po + ($s+$w)*$in_f]
165     set bw [expr $bw + $r_n*($r_f*($s+$w)+$r_s)]
166     set bw [expr $bw + 2*$s_l + $po + $r_s + $test_w]
167
168     # calculate the box height of the surrounding box
169     # 50um of padding
170     set pad_y 50
171
172     set bh [expr 2*$pad_y + 2*$ph + (2*$l-$a)]
173
174     # make the surrounding box
175     make_outline $bw $bh 20
176
177     # make the device
178     magic::box position [expr $x + $pad_x] [expr $y + $pad_y]
179     make_saw saw_spec $x $y $bw $bh $pad_x $pad_y
180
181     # make test structures
182     magic::box position [expr $x + $bw - $test_w ] [expr $y + $pad_y]
183     make_test $bh
184
185     # make index number
186     magic::box position [expr $x + $pad_y + $s_l + 2*$po + $pw] [expr $y + $pad_y]
187     make_id $id
188
189     # document parameters
190     magic::box position $x $y
191     magic::box move south 50
192     document saw_spec $x $y $bw $bh
193
194     # output postscript drawing
195     output $x $y $bw $bh $id
196 }
197
198
199 #===================================================
200 #  make_dual
201 #
202 #  x     - x position of first finger
203 #  y     - y position of first finger
204 #  h     - height of surrounding box
205 #  w     - width of surrounding box
206 #===================================================
207 proc make_dual {  } {
```

```
208
209     global saw_spec
210
211     # save the box position for later
212     set pos  [magic::box position]
213     set x [lindex $pos 0]
214     set y [lindex $pos 1]
215
216     set s      $saw_spec(IDT_Spacing)
217     set w      $saw_spec(IDT_Width)
218     set l      $saw_spec(IDT_Length)
219     set a      $saw_spec(IDT_Apodization)
220
221     set lw     $saw_spec(Leg_Width)
222     set po     $saw_spec(Pad_Offset)
223     set pw     $saw_spec(Pad_width)
224     set ph     $saw_spec(Pad_height)
225
226     set in_f   $saw_spec(IN_Fingers)
227     set r_s    $saw_spec(REF_Spacing)
228     set s_l    $saw_spec(Stop_Length)
229
230     set id     $saw_spec(ID)
231
232     # calculate the box width of the surrounding box
233     # 50um of padding
234     set pad_x 50
235     # width of the test features
236     set test_w 250
237
238     set bw [expr 2*$pad_x + 2*$pw + 2*$po + 2*($s+$w)*$in_f]
239     set bw [expr $bw + $r_s + 2*$s_l + 2*$po + $test_w]
240
241     # calculate the box height of the surrounding box
242     # 50um of padding
243     set pad_y 50
244
245     set bh [expr 2*$pad_y + 2*$ph + (2*$l-$a)]
246
247     # make the surrounding box
248     make_outline $bw $bh 20
249
250     # make left stop block
251     magic::box position [expr $x + $pad_x] [expr $y + $pad_y]
252     magic::box size $s_l [expr $bh - 2*$pad_y]
253     magic::paint metal1
254
255     # Make Input and Output IDTs
256     set saw_spec(IDT_Fingers)   $saw_spec(IN_Fingers)
257     set saw_spec(Include_Pads)      1
258     magic::box move north [expr $ph]
259     magic::box move east  [expr $s_l + 2*$po + $pw]
260     make_idt saw_spec
261     magic::box move east [expr ($s+$w)*$in_f +  $r_s ]
```

```
262         set saw_spec(Include_Pads)      -1
263         make_idt saw_spec
264
265         # make right stop block
266         magic::box move east [expr  $pw + 2*$po + ($s+$w)*$in_f]
267         magic::box move south [expr $ph]
268         magic::box size $s_l [expr $bh - 2*$pad_y]
269         magic::paint metal1
270         magic::paint metal1
271
272
273         # make test structures
274         magic::box position [expr $x + $bw - $test_w ] [expr $y + $pad_y]
275         make_test $bh
276
277         # make index number
278         magic::box position [expr $x + $pad_y + $s_l + 2*$po +$pw] [expr $y + $pad_y]
279         make_id $id
280
281         # document parameters
282         magic::box position $x $y
283         magic::box move south 50
284         document saw_spec $x $y $bw $bh
285
286         # output postscript drawing
287         output $x $y $bw $bh $id
288     }
289
290     #=================================================
291     #=================================================
292
293
294     #=================================================
295
296     # offset spacing of each device
297     set offset_x 1000
298     set offset_y 50
299
300     # device id
301     set id 0
302
303     # start position
304     magic::box position 1000 [expr 70000  - 1500]
305
306     # delete any old postscript files
307     exec "./delps"
308
309     #..............................................
310     # open a file to print out docs
311     exec rm -f specs.csv
312     set fileId [open specs.csv w 0600]
313
314     set fileVars {
315         IDT_Spacing
```

```
316         IDT_Width
317         IDT_Length
318         IDT_Apodization
319         IN_Fingers
320         REF_Fingers
321         REF_Spacing
322         REF_Number
323         REF_Apodization
324         REF_Leg_Width
325         Stop_Length
326         Leg_Width
327         Pad_Offset
328         Pad_width
329         Pad_height
330         Include_Pads
331     }
332
333     set buf "ID, x, y, w, h,"
334     foreach v $fileVars {
335         set buf "$buf$v, "
336     }
337     puts $fileId "$buf \n"
338
339
340
341     #==================================================
342     #           Make Devices
343     #==================================================
344
345
346
347
348     # make a few very long SAW devices
349
350     array set saw_spec {
351         {ID}                 0
352         {IDT_Spacing}       10
353         {IDT_Width}         10
354         {IDT_Length}       300
355         {IDT_Apodization} 270
356         {IN_Fingers}        20
357         {REF_Fingers}       40
358         {REF_Spacing}     2000
359         {REF_Number}        10
360         {REF_Apodization} 270
361         {REF_Leg_Width}     20
362         {Stop_Length}      200
363         {Leg_Width}         20
364         {Pad_Offset}        50
365         {Pad_width}        200
366         {Pad_height}       200
367         {Include_Pads}       1
368     }
369
```

```
370  set len $saw_spec(IDT_Length)
371
372  foreach w { 10 5 3 } {
373
374      set saw_spec(IDT_Spacing) $w
375      set saw_spec(IDT_Width)   $w
376      set saw_spec(IDT_Apodization)   [expr $len - 3 * $w]
377
378      set saw_spec(REF_Leg_Width)     20
379      set saw_spec(ID) $id
380      make_cell
381      next
382
383      set saw_spec(REF_Apodization)   300
384      set saw_spec(ID) $id
385      make_cell
386      next
387
388      set saw_spec(REF_Apodization)   300
389      set saw_spec(REF_Leg_Width)     0
390      set saw_spec(ID) $id
391      make_cell
392      next
393
394  }
395
396
397  #----------------------------
398  # For testing only
399
400  #...............................................
401  # make vertical alignment marks
402  # set saw_spec(REF_Spacing) 300
403  # set saw_spec(ID) $id
404  # make_cell
405  # next
406
407  # set saw_spec(IDT_Length)        100
408  # set saw_spec(IDT_Apodization)    90
409  # set saw_spec(REF_Apodization)    90
410  # make_cell
411  #  next
412
413  # set saw_spec(REF_Spacing)      1000
414  # set saw_spec(IDT_Length)        100
415  # set saw_spec(IDT_Apodization)  90
416  # set saw_spec(REF_Apodization)  90
417
418  # make_dual
419
420  #return
421  # testing
422  #----------------------------
423
```

```
424
425 array set saw_spec {
426     {ID}                0
427     {IDT_Spacing}       10
428     {IDT_Width}         10
429     {IDT_Length}       300
430     {IDT_Apodization} 296
431     {IN_Fingers}        20
432     {REF_Fingers}       40
433     {REF_Spacing}        0
434     {REF_Number}         5
435     {REF_Apodization} 296
436     {REF_Leg_Width}     20
437     {Stop_Length}      200
438     {Leg_Width}         20
439     {Pad_Offset}        50
440     {Pad_width}        200
441     {Pad_height}       200
442     {Include_Pads}       1
443 }
444
445
446 # finger length
447 foreach len { 300 200 100 } {
448     foreach w { 10 5 3 } {
449
450         # set reflector apodiztion for fully shorted(0), 1/2 shorted(1)
451         # and grating(2)
452         foreach a { 0 1 2 } {
453
454             set saw_spec(IDT_Length)  $len
455             set saw_spec(IDT_Spacing) $w
456             set saw_spec(IDT_Width)   $w
457             set saw_spec(IDT_Apodization)   [expr $len - 3 * $w]
458
459             if { $a < 2 } {
460                 set saw_spec(REF_Apodization)   [expr $len - 3 * $w * $a]
461                 set saw_spec(REF_Leg_Width)     $saw_spec(Leg_Width)
462             } else {
463                 set saw_spec(REF_Apodization)   $len
464                 set saw_spec(REF_Leg_Width)     0
465             }
466
467
468             # spacing
469             foreach s {2.0 1.3 2.5} {
470                 # number of in fingers
471                 foreach i_f {10 50 20} {
472                     # number of reflector fingers
473                     foreach r_f  {2.0 1.0 1.5} {
474
475                         set saw_spec(REF_Number)  5
476
477                         # limit width with number of reflectors to no more than
```

```
478                             # 10000um
479                             set we [expr int(($r_f * $i_f)*2*$w + ($s * $i_f * 2 * $w))*5]
480                             while { $we > 8000 } {
481                                 incr saw_spec(REF_Number) -1
482                                 if { $saw_spec(REF_Number) <= 2 } {
483                                     break;
484                                 }
485                                 set we [expr (int($r_f * $i_f)*2*$w + int($s * $i_f * 2 * $w))*$saw_spec(REF_Numbe
486                             }
487
488                             if { $saw_spec(REF_Number) > 2 } {
489                                 set saw_spec(IN_Fingers)  $i_f
490                                 set saw_spec(REF_Fingers) [expr int($r_f * $i_f)]
491                                 set saw_spec(REF_Spacing) [expr int($s * $i_f * 2 * $w)]
492                                 set saw_spec(ID) $id
493                                 make_cell
494                                 next
495                             }
496
497                         }
498                     }
499                 }
500             }
501         }
502     }
503
504
505     # move to next line before making dual port IDTs
506     set pos [magic::box pos]
507     set y [lindex $pos 1]
508     magic::box position $offset_x  [expr $y - 1000]
509
510
511     foreach len { 300 200 100 } {
512         foreach w { 3 5 10 } {
513
514             set saw_spec(IDT_Length)  $len
515             set saw_spec(IDT_Spacing) $w
516             set saw_spec(IDT_Width)   $w
517             set saw_spec(IDT_Apodization)   [expr $len - 3 * $w]
518
519             foreach i {10 20 50} {
520                 set saw_spec(IN_Fingers) $i
521
522                 foreach s {2.0 2.5 3.0 3.5} {
523                     set saw_spec(REF_Spacing) [expr int($s*2*$w*$i)]
524                     set saw_spec(ID) $id
525                     make_dual
526                     next
527                 }
528             }
529
530         }
531     }
```

```
532
533
534 #..............................................
535 # make vertical alignment marks
536 magic::box size 250 70000
537
538 #..............................................
539 # make vertical alignment marks
540 magic::box size 250 70000
541
542 foreach i { 0 34750 69750 } {
543     magic::box position $i 0
544     magic::paint metal1
545 }
546
547 magic::box size 70000 250
548
549 foreach i { 0 69750 } {
550     magic::box position 0 $i
551     magic::paint metal1
552 }
553
554
555 magic::select
556 magic::plot postscript all.ps {metal1}
557
558 puts "fixing postscript files"
559 #exec "./fix"
560
561 close $fileId
```

TCL : saw.tcl

```
1  #==================================================
2  #
3  #        SAW Library
4  #
5  #        filename : saw.tcl
6  #        authors  : Jason LaPenta (JL)
7  #        date     : 3/2/2006 (JL)
8  #        target   : Magic v7.3
9  #           [http://opencircuitdesign.com/magic/]
10 #        version  : 0.2 alpha (branch 0)
11 #
12 #        abstract : generates parameterized SAW gratings
13 #                   and test features
14 #
15 #  Changes :
16 #  Date      Who    Description
17 # --------------------------------------------------
18 #  2/12/06   JL     New File
19 #  3/5/06    JL     added in right pads on IDT
20 #--------------------------------------------------
21
22
23 #-------------------------------------------------
24 # make_saw
25 #
26 # makes a SAW at the cursor's location
27 #
28 # pass saw spec
29
30 proc make_saw { sawSpecName x y bw bh pad_x pad_y } {
31
32     upvar $sawSpecName saw_spec
33
34     set s      $saw_spec(IDT_Spacing)
35     set w      $saw_spec(IDT_Width)
36     set l      $saw_spec(IDT_Length)
37     set a      $saw_spec(IDT_Apodization)
38     set lw     $saw_spec(Leg_Width)
39     set po     $saw_spec(Pad_Offset)
40     set pw     $saw_spec(Pad_width)
41     set ph     $saw_spec(Pad_height)
42     set in_f   $saw_spec(IN_Fingers)
43     set r_f    $saw_spec(REF_Fingers)
44     set s_l    $saw_spec(Stop_Length)
45
46
47     # save the box position for later
48     set box_pos  [magic::box position]
49
50     # reflector spacing
51     set r_space $saw_spec(REF_Spacing)
52
```

```
53      # make left stop block
54      magic::box size $s_l [expr $bh - 2*$pad_y]
55      magic::paint metal1
56
57      # make input IDT
58      magic::box move north [expr $ph]
59      magic::box move east  [expr $s_l + 2*$po + $pw]
60
61      set saw_spec(Include_Pads)  1
62      set saw_spec(IDT_Fingers)   $saw_spec(IN_Fingers)
63      make_idt saw_spec
64
65      # setup for reflectors
66      set saw_spec(Include_Pads)      0
67      set saw_spec(IDT_Fingers)       $saw_spec(REF_Fingers)
68      set saw_spec(IDT_Apodization)  $saw_spec(REF_Apodization)
69      set saw_spec(Leg_Width)        $saw_spec(REF_Leg_Width)
70
71      # make reflectors
72      set r_num  $saw_spec(REF_Number)
73
74      for {set i 0} {$i < $r_num} {incr i} {
75          set size [magic::box size]
76          magic::box move east [expr [lindex $size 0] + $r_space]
77          make_idt saw_spec
78      }
79
80      # restore led width and apodization
81      set saw_spec(Leg_Width)        $lw
82      set saw_spec(IDT_Apodization)  $a
83
84      # add in stop block
85      magic::box move east [expr [lindex $size 0] + $r_space]
86      magic::box move south [expr $ph]
87  #    magic::box move south [expr $l/2]
88  #    magic::box size $s_l [expr 2*$l]
89      magic::box size $s_l [expr $bh - 2*$pad_y]
90      magic::paint metal1
91
92      # box around the whole device
93      # draw box around new idt
94      # todo y overshoots by a small amount
95      magic::box position [lindex $box_pos 0] [lindex $box_pos 1]
96      magic::box move south [expr $ph]
97      magic::box move west  [expr $pw+$po]
98      set r_x_dist [expr (($s + $w)*$r_f + $r_space)*$r_num + ($s + $w)*$in_f]
99      magic::box size [expr $r_x_dist + $pw + $po + 2*[lindex $size 0] +
100         [lindex $size 0]] [expr 2*$l-$a + 2*$ph]
101 }
102
103 #------------------------------------------------
104 # make_idt
105 #
106 # makes and IDT at the cursor's location
```

```
107  #
108  # pass spec
109
110  proc make_idt { specName } {
111
112      # save box position and size
113      set box_pos  [magic::box position]
114      set box_size [magic::box size]
115
116      upvar $specName spec
117
118      set s     $spec(IDT_Spacing)
119      set w     $spec(IDT_Width)
120      set l     $spec(IDT_Length)
121      set a     $spec(IDT_Apodization)
122      set num   $spec(IDT_Fingers)
123      set lw    $spec(Leg_Width)
124      set po    $spec(Pad_Offset)
125      set pw    $spec(Pad_width)
126      set ph    $spec(Pad_height)
127
128  #     puts "\n  Making SAW IDT ( $s, $w, $l, $a, $num, $lw )\n"
129
130      set start_pos [magic::box position]
131      set start_pos_x [lindex $start_pos 0]
132      set start_pos_y [lindex $start_pos 1]
133
134      # apodization offset
135      set ao [expr ($l - $a)/2]
136
137      # make one finger at 0,0
138      magic::box size $w $l
139      magic::paint metal1
140
141      # make second offset finger
142      magic::box move right [expr $w + $s]
143      magic::box move up $ao
144      magic::paint metal1
145
146      # array 2 fingers number of finger times
147      magic::box move left [expr $w + $s]
148      magic::box move down $ao
149      magic::box size [expr 2 * ($w+$s)] [expr $l + $ao]
150      magic::select visible
151
152      if { [expr $num % 2] > 0 } {
153          # odd number of finger
154          magic::array [expr ($num + 1) / 2] 1
155          # delete last finger
156          magic::select clear
157          magic::box move right [expr ($w + $s)*$num]
158          magic::box size $w [expr $l + $ao]
159          magic::select visible
160          magic::delete
```

```
161     } else {
162         # even number of finger
163         magic::array [expr $num / 2] 1
164     }
165
166     # make bottom leg
167     magic::box position $start_pos_x  $start_pos_y
168
169     magic::box move up [expr -$lw]
170     if { [expr $num % 2] > 0 } {
171         # odd number of finger
172         magic::box size [expr ($w + $s) * $num - $s] $lw
173     } else {
174         # even number of finger
175         magic::box size [expr ($w + $s) * ($num - 1) - $s] $lw
176     }
177
178     # correct for shorted grating
179     if { $a >= $l } {
180         if { [expr $num % 2] > 0 } {
181             # odd number of finger
182             magic::box size [expr ($w + $s) * ($num - 1) + $w] $lw
183         } else {
184             # even number of finger
185             magic::box size [expr ($w + $s) * ($num - 1) + $w] $lw
186         }
187     }
188
189     magic::paint metal1
190
191     # make top leg
192     magic::box position [expr $start_pos_x + $w + $s] [expr $start_pos_y + $l + $ao]
193     if { [expr $num % 2] > 0 } {
194         # odd number of finger
195         magic::box size [expr ($w + $s) * ($num - 2) - $s] $lw
196     } else {
197         # even number of finger
198         magic::box size [expr ($w + $s) * ($num - 1) - $s] $lw
199     }
200
201     # correct for shorted grating
202     if { $a >= $l } {
203         magic::box position [expr $start_pos_x] [expr $start_pos_y + $l + $ao]
204         if { [expr $num % 2] > 0 } {
205             # odd number of finger
206             magic::box size [expr ($w + $s) * ($num - 2)] $lw
207         } else {
208             # even number of finger
209             magic::box size [expr ($w + $s) * ($num - 1)] $lw
210         }
211         magic::box size [expr ($w + $s) * $num - $s] $lw
212     }
213
214     magic::paint metal1
```

```
215
216     # Make bond pads if requested to the left
217     if {$spec(Include_Pads) == 1} {
218 #         puts "  Making pads ( $po, $pw, $ph )\n"
219
220         # make bottom leg
221         magic::box position [expr $start_pos_x - $po ] [expr $start_pos_y -$lw]
222         magic::box size [expr $w + $s + $po] $lw
223         magic::paint metal1
224
225         # make top leg
226         magic::box move up [expr $l + $ao + $lw]
227         magic::box size [expr $w + $s + $po] $lw
228         magic::paint metal1
229
230         # make bottom pad
231         magic::box position [expr $start_pos_x -$po -$pw ] [expr $start_pos_y -$ph]
232         magic::box size $pw $ph
233         magic::paint metal1
234
235         # make top leg
236         magic::box position [expr $start_pos_x -$po -$pw] [expr $start_pos_y + $l + $ao]
237         magic::box size $pw $ph
238         magic::paint metal1
239     }
240
241     # Make bond pads if requested to the right
242     if {$spec(Include_Pads) == -1} {
243 #         puts "  Making pads ( $po, $pw, $ph )\n"
244
245         magic::box position $start_pos_x $start_pos_y
246         # make bottom leg
247         magic::box move east [expr ($w + $s) * ($num - 1) - $s]
248         magic::box size [expr $w + $s + $po] -$lw
249         magic::paint metal1
250
251         # make bottom pad
252         magic::box move east [expr $w + $s + $po]
253         magic::box move north $lw
254         magic::box size $pw -$ph
255         magic::paint metal1
256
257         # make top leg
258         magic::box move up [expr $l + $ao + $ph]
259         magic::box move west [expr $w + $s + $po]
260         magic::box size [expr $w + $s + $po] $lw
261         magic::paint metal1
262
263         # make top pad
264         magic::box move east [expr $w + $s + $po]
265         magic::box size $pw $ph
266         magic::paint metal1
267     }
268
```

```
269     # draw box around new idt
270     magic::box position [lindex $box_pos 0] [lindex $box_pos 1]
271     magic::box size [expr ($s + $w)*$num - $s] [expr 2*$l-$a]
272 }
273
274 #-------------------------------------------------
275 # document
276 #
277 # labels SAW with paramaters
278 # w - width
279 # h - height
280
281 proc document { specName x y w h } {
282
283     upvar $specName spec
284     global offset_x
285     global offset_y
286
287
288     # vertical offset between lines
289     set os      100
290
291     # check if last device fits
292     # if not then move
293     if { $x + $w > (70000 / 2) - $offset_x } {
294         set x $offset_x
295         set y [expr $y - $h - $offset_y]
296     }
297
298     # document with lables
299
300     set vals {
301         IDT_Spacing
302         IDT_Width
303         IDT_Length
304     }
305
306     set doc "[format_"%\#05x"_$spec(ID)]($x,$y)\[$w_x_$h\]"
307     foreach var $vals {
308         set doc "$doc:$var=$spec($var)"
309     }
310
311     magic::box size 1 1
312     regsub -all {\s} $doc {_} string
313     puts $string
314     magic::label $string southeast
315     magic::box move south $os
316
317     set vals {
318         Leg_Width
319         Pad_Offset
320         Pad_width
321         Pad_height
322         IDT_Apodization
```

```
323         }
324
325         set doc ""
326         foreach var $vals {
327             set doc "$doc:$var=$spec($var)"
328         }
329
330         regsub -all {\s} $doc {_} string
331         puts $string
332         magic::label $string southeast
333         magic::box move south $os
334
335         set vals {
336             IN_Fingers
337             REF_Fingers
338             REF_Spacing
339         }
340
341         set doc ""
342         foreach var $vals {
343             set doc "$doc:$var=$spec($var)"
344         }
345
346         regsub -all {\s} $doc {_} string
347         puts $string
348         magic::label $string southeast
349
350         # add to CSV file
351         global fileId
352         global fileVars
353
354         set buf "[format_"%\#05x"_$spec(ID)],_$x,_$y,_$w,_$h,"
355         foreach v $fileVars {
356             set buf "$buf$spec($v),_"
357         }
358         puts $fileId "$buf"
359
360     }
361
362     #------------------------------------------------------------
363     # make_test_boxes
364     #
365     # make two boxes
366
367     proc make_test_boxes { } {
368
369         set size  64
370
371         while { $size > 2 } {
372             magic::box size $size $size
373             magic::paint metal1
374             magic::box move northeast $size
375             magic::paint metal1
376             magic::box move west $size
```

```
377
378             set size [expr $size / 2]
379             magic::box move north [expr $size + 10]
380         }
381     }
382
383     #------------------------------------------------------------
384     # make_test_grating
385     #
386     # horizontal test lines
387
388     proc make_test_grating { y bh } {
389
390         set size 3
391         set total 0
392         set width 200
393
394         while { $size < 50 } {
395             magic::box size $width $size
396             magic::paint metal1
397
398             magic::box move north [expr $size + 10]
399             set total [expr $total + $size + 10]
400             set size [expr $size * 1.5]
401         }
402
403         if { $bh < 750 } {
404             return
405         }
406
407         magic::box move north $size
408
409         set size 3
410         while { $size < 30 } {
411
412
413             magic::box size $width 20
414             magic::paint metal1
415
416             magic::box move north [expr $size + 20]
417             set size [expr $size * 1.5]
418         }
419     }
420
421     #------------------------------------------------------------
422     # make_test
423     #
424     # makes both boxes and gratings
425
426     proc make_test { bh } {
427
428         set pos [magic::box pos]
429         set y   [lindex $pos 1]
430
```

```
431  #     puts "Making_Test_Features"
432
433       make_test_boxes
434       magic::box move north 50
435       make_test_grating $y $bh
436  }
437
438  #---------------------------------------------------
439  # make_ticbox
440  #
441  # Makes corner and outline dashes
442  #
443  # w - width
444  # h - height
445  # ls - size  (tic length)
446  # ts - size  (tic thickness)
447
448  proc make_ticbox {w h ls ts} {
449
450  #     puts "Making_Tic_Box"
451
452       # bottom left
453       magic::box size $ls $ts
454       magic::paint metal1
455       magic::box size $ts $ls
456       magic::paint metal1
457
458       # bottom
459       magic::box move east [expr $w/2 - $ls/2]
460       magic::box size $ls $ts
461       magic::paint metal1
462
463       # bottom right
464       magic::box move east [expr $w/2 - $ls/2]
465       magic::box size $ls $ts
466       magic::paint metal1
467       magic::box move east $ls
468       magic::box size -$ts $ls
469       magic::paint metal1
470
471       # right
472       magic::box move north [expr $h/2 - $ls/2]
473       magic::box size $ts $ls
474       magic::paint metal1
475
476       # top right
477       magic::box move north [expr $h/2 - $ls/2]
478       magic::paint metal1
479       magic::box move north [expr $ls - $ts]
480       magic::box move west  [expr $ls - $ts]
481       magic::box size $ls $ts
482       magic::paint metal1
483
484       # top
```

```
485       magic::box move west [expr $w/2 - $ls/2]
486       magic::box size $ls $ts
487       magic::paint metal1
488
489       # top left
490       magic::box move west [expr $w/2 - $ls/2]
491       magic::box size $ls $ts
492       magic::paint metal1
493       magic::box size $ts  [expr -$ls + $ts]
494       magic::paint metal1
495
496       # left
497       magic::box move south [expr $h/2 - $ls/2]
498       magic::box size $ts $ls
499       magic::paint metal1
500   }
501
502   #-----------------------------------------------
503   # make_outline
504   #
505   # makes a box outline
506   #
507   # w  - width
508   # h  - height
509   # lw - line width
510
511   proc make_outline {w h lw} {
512
513   #     puts "Making Tic Box"
514
515       # bottom
516       magic::box size $w $lw
517       magic::paint metal1
518
519       # top
520       magic::box move north [expr $h - $lw]
521       magic::paint metal1
522
523       # left
524       magic::box move south [expr $h - $lw]
525       magic::box size $lw $h
526       magic::paint metal1
527
528       # right
529       magic::box move east [expr $w - $lw]
530       magic::box size $lw $h
531       magic::paint metal1
532
533   }
534
535   #-----------------------------------------------
536   # make_id
537   #
538   # Makes a binary identification tic feature
```

```
539 #
540 # bin = list of 0 and 1
541
542 proc make_id { id } {
543
544     set bin {0 0 0 0 0 0 0 0 0 0}
545
546     set i 0
547     while { $id > 0 } {
548
549         set t [expr $id % 2]
550         set id [expr $id / 2]
551         lset bin [expr 9 - $i] $t
552         incr i
553     }
554
555
556     # size of tics
557     set s 20
558
559     magic::box size $s $s
560     set i 3
561     foreach var $bin {
562
563         magic::paint metal1
564
565         if { $var == 1 } {
566             magic::move north [expr $s]
567             magic::paint metal1
568             magic::move south [expr $s]
569         }
570
571         magic::move east [expr $s + $s/2]
572         if { $i % 4 == 0 } {
573             magic::move east [expr $s]
574         }
575         incr i
576     }
577
578 }
579
580
581 #------------------------------------------------
582 # make text
583 #
584
585 proc make_text { str } {
586
587     source char.glyph
588
589     puts "mtext"
590
591
592     # set pixel width and height
```

```
593        set pw 20
594        set ph 20
595
596        magic::move north [expr 8*$ph]
597        magic::box size $pw $ph
598
599        # wright out text
600        foreach chr [split $str ""] {
601
602             if { $chr == "_" } {
603                 magic::move east  [expr 6*$pw]
604                 continue
605             }
606
607            puts $chr
608
609            set glyph [lindex [array get cmap $chr ] 1 ]
610
611            foreach dot [split $glyph ""] {
612                if { $dot == "." } {
613                    magic::move east $pw
614                } elseif { $dot == "O" } {
615                    magic::paint metal1
616                    magic::move east $pw
617                } elseif { $dot == "\n" } {
618                    magic::move south $ph
619                    magic::move west [expr 9*$pw]
620                }
621            }
622
623            magic::move east  [expr 16*$pw]
624            magic::move north [expr 8*$ph]
625
626        }
627
628    }
```

TCL : make_string.tcl

```
1  proc mtext { str } {
2
3      source char.glyph
4
5      set pos [magic::box pos]
6      set x [lindex $pos 0]
7      set y [lindex $pos 1]
8
9      # set pixel width and height
10     set pw 40
11     set ph 40
12
13     magic::move north [expr 8*$ph]
14     magic::box size $pw $ph
15
16     # wright out text
17     foreach chr [split $str ""] {
18
19          if { $chr == "␣" } {
20              magic::move east  [expr 6*$pw]
21              continue
22          }
23
24         set glyph [lindex [array get cmap $chr ] 1 ]
25
26         foreach dot [split $glyph ""] {
27             if { $dot == "." } {
28                 magic::move east $pw
29             } elseif { $dot == "O" } {
30                 magic::paint metal1
31                 magic::move east $pw
32             } elseif { $dot == "\n" } {
33                 magic::move south $ph
34                 magic::move west [expr 9*$pw]
35             }
36         }
37
38         magic::move east  [expr 16*$pw]
39         magic::move north [expr 8*$ph]
40     }
41
42     magic::box position $x [expr $y - 10*$ph]
43 }
44
45 mtext "Massachusetts␣Institute␣of␣Technology"
46 mtext "uTags␣SAW␣Mask␣1␣Rev"
47 mtext "March␣10␣2006"
48 mtext "copyright␣2006␣Jason␣LaPenta"
49 mtext "MIT␣Media␣Laboratory"
```

TCL : `make_char.pl`

```
1   #!/usr/bin/perl -w
2
3
4   use Text::Banner;
5   $a = Text::Banner->new;
6
7
8   $a->size(1);
9   $a->fill('O');
10  $a->rotate('h');
11
12
13  print "array set cmap {\n";
14
15  # generate a string of all characters
16  foreach $c (40..126){
17  #  print "[$c]\n";
18    $c = chr($c);
19    next if $c =~ /[\[\](){}#\\*+,"-.\/]/;
20    $a->set( "$c " );
21    print "{$c} {\n";
22    $s = $a->get;
23    $s =~ s/^\W*$//;
24    chop $s;
25    chop $s;
26    chop $s;
27    $s =~ s/        \n/\n/mg;
28    $s =~ s/ /./mg;
29    print $s;
30    print "}\n";
31  }
32
33  print "}\n";
```

TCL : fix.pl

```
1   #!/usr/bin/perl -w
2   #
3   # fix postscript files generated by Magic so they
4   # will be displayed properly.
5
6   foreach $fn (<*.epsi>) {
7
8     print "\n processing $fn\n";
9
10    open( $fh_in,   "<$fn" );
11    open( $fh_out1, ">out1.temp" );
12    open( $fh_out2, ">out2.temp" );
13
14    foreach $l (<$fh_in>) {
15      if ( $l =~ /fb$/ ) { print $fh_out1 $l; }
16    }
17
18    close( $fh_out1 );
19    open( $fh_in1, "<out1.temp" );
20    seek( $fh_in, 0, 0 );
21
22    foreach $l (<$fh_in>) {
23
24      next if $l =~ /fb$/ ;
25
26      if ( $l =~ /^\/col17/ ) {
27        print $fh_out2 "/col18 {0.279 0.128 0.000 0.000 sc} bind def\n";
28      }
29
30      print $fh_out2 $l;
31
32      if ( $l =~ /^l1$/ ) {
33        # insert fb lines
34        print $fh_out2 "5 sl\ncol18\n";
35        foreach $l2 (<$fh_in1>) { print $fh_out2 $l2; }
36        print $fh_out2 "col12\n";
37      }
38    }
39
40    close( $fh_in );
41    close( $fh_in1 );
42    close( $fh_out2 );
43
44    `mv out2.temp $fn`;
45  }
```

A.2 Mask 2 & 3 Generation Scripts

PERL : `fix.cif.layers.pl`

```
1  #!/usr/bin/perl -w
2
3  $ARGV[0] =~ /(.*?)\.cif$/;
4  $fn = $1;
5
6  print "fixing $fn.cir\n output is $fn.out.cif\n\n";
7
8  system( "dos2unix $fn.cif" );
9
10 open( $fhi, "<$fn.cif" ) or die $!;
11 open( $fho, ">$fn.out.cif" ) or die $!;
12
13 foreach $line (<$fhi>) {
14
15   if ( $line =~ /^L (.*?);$/ ) {
16
17     $last_layer = $1;
18     unless ( defined $layers{ $last_layer } ) {
19       $layers{ $last_layer } = $line;
20     }
21
22   } elsif ( $line =~ /^B/ ) {
23     $layers{ $last_layer } .= $line;
24
25   } elsif ( $line =~ /^(DF|E)/ ) {
26     next;
27
28   } else {
29     print $fho $line;
30
31   }
32
33 }
34
35 foreach $key (keys %layers) {
36
37   print $fho $layers{$key};
38
39 }
40
41 print $fho "DF;\nE\n";
42
43 close( $fhi );
44 close( $fho );
```

PERL : layout.pm

```
1  #!/usr/bin/perl -w
2  #
3  # Package to simplify generating scripts for
4  # 'layout' CAD tool
5  #==================================================
6
7  package layout;
8
9  #==================================================
10 BEGIN {
11   require Exporter;
12   our ( @ISA, @EXPORT, @EXPORT_OK, %EXPORT_TAGS );
13   @ISA     = qw(Exporter);
14   @EXPORT = qw( head     end   mac     make_idt  setup
15                 box   mirror   copy point2 selectAll
16                 clearPts pt flatAll deselectAll
17                 newCell );
18 }
19
20 my $mac_fn = ''; # macro filename
21
22 #--------------------------------------------------
23 sub head {  # generate the macro heading
24
25   my $fn = shift;
26
27   open( $mac_file, ">$fn" ) || die $!;
28   mac( '#!/usr/bin/layout' );
29   mac( "#name=Macro:_$fn_Perl_Generated" );
30   mac( '#help=Recorded_Thu_Jan_7_2007');
31   mac( 'int_main(){');
32 }
33 #--------------------------------------------------
34 sub end { # end the macro
35   mac('return_0;}');
36 }
37 #--------------------------------------------------
38 sub mac { # insert a macro
39   print $mac_file (shift)."\n";
40 }
41 #--------------------------------------------------
42 sub make_idt { # generate macro sequence for an IDT
43
44   my $name      = shift;
45   my $polarity = shift;
46   my $c          = shift; # config hash
47
48   my $w = $c->{-Width };
49   my $l = $c->{-Len };
50   my $g = $c->{-Gap };
51   my $f = $c->{-Fingers };
52   my $a = $c->{-Ap }; # Apodization
```

```
53
54    # delete existing idt
55    mac( "layout->drawing->setCell(\"$name\");");
56    mac( 'layout->drawing->deleteActuellCell();' );
57    #--------------------------------------------------
58    setup();
59    #--------------------------------------------------
60    # draw IDT finger pairs
61    newCell();
62    mac( "layout->drawing->currentCell->cellName=\"$name\";" );
63    box( 0, 0, $w, $l );
64    selectAll();
65    copy( 0, 0, ($w +$g), ($l-$a)/2 );
66    selectAll();
67
68    foreach ( 2..$f ) {
69      copy( 0, 0, 2*($w + $g), 0 );
70    }
71
72    if ( $polarity eq 'odd' ) {
73      clearPts();
74      selectAll();
75      mirror( 0, (3*$l - $a)/4,
76              1, (3*$l - $a)/4 );
77    }
78  }
79  #--------------------------------------------------
80  sub setup { # setup drawing mode for metal features
81    deselectAll();
82    clearPts();
83    mac( 'layout->drawing->activeLayer=0;' );
84    mac( 'layers::num[0].name="metal1";' );
85    mac( 'layers::num[0].setStyle(6);' );
86    mac( 'layers::num[0].setColor(130,43,43);' );
87  }
88  #--------------------------------------------------
89  sub box { # draw a box
90    point2( @_ );
91    mac( 'layout->drawing->box();' );
92  }
93  #--------------------------------------------------
94  sub mirror { # mirror the selected objects
95    point2( @_ );
96    mac( 'layout->drawing->mirror();' );
97  }
98  #--------------------------------------------------
99  sub copy { # copy selected objects
100   point2( @_ );
101   mac( 'layout->drawing->copy();' );
102 }
103 #--------------------------------------------------
104 sub point2 { # select objects between two points
105   my ($x1,$y1,$x2,$y2) = @_;
106
```

```
107     mac( 'layout->drawing->clearPoints();' );
108     pt( $x1, $y1);
109     pt( $x2, $y2);
110   }
111   #-----------------------------------------------
112   sub selectAll {
113     mac( 'layout->drawing->selectAll();' );
114   }
115   #-----------------------------------------------
116   sub clearPts { # clear selection
117     mac( 'layout->drawing->clearPoints();' );
118   }
119   #-----------------------------------------------
120   sub pt { # start a selection
121     mac( 'layout->drawing->point('.(shift).','.(shift).');' );
122   }
123   #-----------------------------------------------
124   sub flatAll { # flatten cell
125     mac( 'layout->drawing->flatAll();' );
126   }
127   #-----------------------------------------------
128   sub deselectAll {
129     mac( 'layout->drawing->deselectAll();' );
130   }
131   #-----------------------------------------------
132   sub newCell {
133     mac( 'layout->newCell();' );
134   }
135
136   1;
```

PERL : idt generator : `idt.pl`

```
1  #!/usr/bin/perl -w
2  #
3  #                  generates IDTs
4  #-----------------------------------------------------
5
6  use layout;
7
8  $mac_fn = $ARGV[0];
9  head( $mac_fn );
10
11 # initial setup
12 setup();
13 # all parameters in nm
14 $g = $IDT{ -Gap }    =  1100;
15 $w = $IDT{ -Width } =  1100;
16 $l = $IDT{ -Len }    = 100000;
17 $f = $IDT{ -Fingers } = 5;
18
19 $IDT{ -Offset }   = 2;
20 $IDT{ -Ap }       = 100000;
21
22 # make IDTs
23 make_idt( 'CODE0', 'even', \%IDT );
24 draw_rails( 2*($g+$w)*$f );
25 $name = "IDT_".($w/1000)."um/".$f."f/".($l/1000)."L";
26 mac( "layout->drawing->currentCell->cellName=\"$name\";" );
27
28 # finish up
29 selectAll();
30 flatAll();
31 deselectAll();
32 end();
33 exit;
34
35 #-----------------------------------------------------
36 sub draw_rails {
37
38   my $len = shift;
39
40   $rw    = 30000;                # Rail Width
41   $rlen = $len + $rw;
42   $l     = $IDT{ -Len };
43   $a     = $IDT{ -Ap };
44
45   clearPts();
46   # bottom rail
47   box( -$rw, 0, $rlen, -$rw );
48   # top rail
49   box( -$rw, $l + ($l - $a)/2,
50         $rlen, $rw + $l + ($l - $a)/2 );
51 }
```

PERL : wide–band idt generator : `wb.pl`

```
1  #!/usr/bin/perl -w
2  #
3  #  generates Wide Band SAW IDTS
4  #------------------------------------------------------------------------
5
6  use layout;
7
8  $mac_fn = $ARGV[0];
9  head( $mac_fn );
10
11 # initial setup
12 setup();
13
14 # all parameters in nm
15 $IDT{ -Gap }     =   3000;
16 $IDT{ -Width } =   3000;
17 $IDT{ -Len }     = 200000;
18 $IDT{ -Ap }      = 194000;
19 $IDT{ -Fingers } = 20;
20 $IDT{ -Offset }  = 2;
21
22 $code = [1,0,1,1,1];
23
24 foreach my $f ( 1, 5, 20 ){
25
26   $IDT{ -Fingers } = $f;
27
28   # make IDTs
29   make_idt( 'CODE0', 'even', \%IDT );
30   make_idt( 'CODE1', 'odd', \%IDT );
31
32   ( $wt, $len ) = encode( $code, \%IDT );
33   draw_RandP( $wt, $len );
34
35   # finish up
36   selectAll();
37   flatAll();
38   deselectAll();
39 }
40 end();
41 exit;
42
43 #----------------------------------------------
44 # Draw Rails and pads
45 sub draw_RandP {
46
47   my $wt = shift;
48   my $len = shift;
49
50   $rw    = 30000;  # Rail Width
51   $rlen = $wt * $len + $rw;
52   $l     = $IDT{ -Len };
```

```
53    $a      = $IDT{ -Ap };
54
55    clearPts();
56    # bottom rail
57    box( -$rw, 0, $rlen, -$rw );
58    # top rail
59    box( -$rw, $l + ($l - $a)/2,
60         $rlen, $rw + $l + ($l - $a)/2 );
61
62    # Draw Pads
63    $pad_size = 100000;
64    # bottom pad
65    box( 0, -$rw, $pad_size, (-$pad_size - $rw) );
66    # top pad
67    box( 0, $rw + $l + ($l - $a )/2,
68         $pad_size, $rw + $l + ($l - $a)/2 + $pad_size );
69  }
70  #-------------------------------------------------
71  sub encode {
72
73    my $en = shift; # encoding
74    my $c  = shift; # config hash
75
76    my $w = $c->{-Width };
77    my $l = $c->{-Len };
78    my $g = $c->{-Gap };
79    my $f = $c->{-Fingers };
80    my $s = $c->{-Offset }; #offset spacing between IDTs
81    my $name = '';
82
83    newCell();
84
85    # Draw Finger Encodings
86    my $len = 0; # total lenthing in X
87    my $wt  = 2*$f*($w + $g); # width per idt total
88
89    foreach my $encoding ( @$en ) {
90      $name .= $encoding;
91      clearPts();
92      pt( $len*$wt, 0);
93
94      mac( 'layout->drawing->cellref("CODE0");' ) if $encoding == 0;
95      mac( 'layout->drawing->cellref("CODE1");' ) if $encoding == 1;
96
97      $len += $s;
98      print $len*$wt." $len \n";
99    }
100
101   $name = "WB [$name] ";
102   $name .= ($w/1000)."um/".$f."f/".($l/1000)."L";
103   mac( "layout->drawing->currentCell->cellName=\"$name\";" );
104   return ($wt, $len);
105 }
```

PERL : narrow–band idt generator : `nb.pl`

```
1  #!/usr/bin/perl -w
2  #
3  #                 generates Narrow Band SAW IDTS
4  #-------------------------------------------------
5
6  use layout;
7
8  #-------------------------------------------------
9  $mac_fn = $ARGV[0];
10 head( $mac_fn );
11
12 # initial setup
13 setup();
14
15 # all parameters in nm
16 $IDT{ -Gap }     =   3000;
17 $IDT{ -Width } =   3000;
18 $IDT{ -Len }     = 200000;
19 $IDT{ -Ap }      = 194000;
20 $IDT{ -Fingers } = 1;
21 $IDT{ -Offset }  = 1;
22
23 # make IDTs
24 make_idt( 'CODE0', 'even', \%IDT );
25 make_idt( 'CODE1', 'odd', \%IDT );
26
27 @encode = ( [1,0,1,1,1],
28             [0,1,1,1,1],
29             [1,1,0,1,1],
30             [1,0,0,1,1] );
31
32 foreach my $code (@encode) {
33
34   ( $wt, $len ) = encode( $code, \%IDT );
35
36   draw_RandP( $wt, $len );
37
38   # finish up
39   selectAll();
40   flatAll();
41   deselectAll();
42 }
43 end();
44 exit;
45
46 #-------------------------------------------------
47 sub draw_RandP { # Draw Rails and pads
48
49   my $wt = shift;
50   my $len = shift;
51
52   $rw    = 30000;                  # Rail Width
```

```
53     $rlen = $wt * $len + $rw;
54     $l    = $IDT{ -Len };
55     $a    = $IDT{ -Ap };
56
57     clearPts();
58
59     # bottom rail
60     box( -$rw, 0, $rlen, -$rw );
61     # top rail
62     box( -$rw, $l + ($l - $a)/2,
63          $rlen, $rw + $l + ($l - $a)/2 );
64
65     # Draw Pads
66     $pad_size = 100000;
67     # bottom pad
68     box( 0, -$rw, $pad_size, (-$pad_size - $rw) );
69     # top pad
70     box( 0, $rw + $l + ($l - $a )/2,
71          $pad_size, $rw + $l + ($l - $a)/2 + $pad_size );
72   }
73   #------------------------------------------------
74   sub encode {  # make incoded finger pattern
75
76     my $en = shift; # encoding
77     my $c  = shift; # config hash
78
79     my $w = $c->{ -Width };
80     my $l = $c->{ -Len };
81     my $g = $c->{ -Gap };
82     my $f = $c->{ -Fingers };
83     my $s = $c->{ -Offset }; #offset spacing between IDTs
84
85     my $name = '';
86     mac( 'layout->newCell();' );
87
88     # Draw Finger Encodings
89     my $len = 0; # total lenthing in X
90     my $wt  = 2*$f*($w + $g); # width per idt total
91
92     foreach my $encoding ( @$en ) {
93       $name .= $encoding;
94       clearPts();
95       pt( $len*$wt, 0);
96
97       mac( 'layout->drawing->cellref("CODE0");' ) if $encoding == 0;
98       mac( 'layout->drawing->cellref("CODE1");' ) if $encoding == 1;
99       $len += $s;
100    }
101
102    $name = "NB_[$name]_";
103    $name .= ($w/1000)."um/".$f."f/".($l/1000)."L";
104    mac( "layout->drawing->currentCell->cellName=\"$name\";" );
105    return ($wt, $len);
106  }
```

layout : `outline.layout`

```
1  #!/usr/bin/layout
2  #name=Magic Outline
3  #help=Adds a box around selected items on current layer
4
5  int maxX,maxY,minX,minY;
6  int selected = 0; /* was something selected */
7
8  int getCellExtents( cell *cellp , int baseX , int baseY , bool selectall ){
9
10   elementList *elements;
11   element   *element;
12   cell        *tmp;
13
14   pointArray pa;
15   point p;
16
17   if( cellp != NULL )
18     {
19       for(elements = cellp->firstElement; elements != NULL; elements = elements->nextElement)
20         {
21           element = elements->thisElement;
22           if( element != NULL )
23             if( element->select || selectall )
24               {
25                 selected++;
26
27                 // cell reference
28                 if( element->isCellref() )
29                   {
30                     tmp = element->depend();
31                     pa = element->getPoints();
32                     p  = pa.point(0);
33                     getCellExtents( tmp, p.x(), p.y(), true );
34                   }
35                 /* only change selected boxes */
36                 if(element->isBox())
37                   {
38                     /* get coordinates */
39                     pa = element->getPoints();
40
41                     int idx;
42                     for( idx = 0; idx < pa.size(); idx++ ){
43                       p = pa.point(idx);
44
45                       if( p.x() + baseX > maxX )
46                         maxX = p.x() + baseX;
47                       if( p.y() + baseY > maxY )
48                         maxY = p.y() + baseY;
49                       if( p.x() + baseX < minX )
50                         minX = p.x() + baseX;
51                       if( p.y() + baseY < minY )
52                         minY = p.y() + baseY;
```

```
53                     }
54                   };
55               }; /* end if element selected */
56         };
57     };
58 }
59 %------------------------------------------------
60 int getExtents( )
61 {
62   cellList  *cells;
63   maxX = -99999999;
64   minX = 99999999;
65   maxY = -99999999;
66   minY = 99999999;
67
68   /* check if something was selected at all */
69   for(cells = layout->drawing->firstCell; cells!=NULL; cells=cells->nextCell)
70     {
71       getCellExtents(cells->thisCell, 0, 0, false );
72     };
73
74   if( selected == 0)
75     return 0;
76
77   return 1;
78 }
79 %------------------------------------------------
80 void drawBox( int x1,int y1,int x2,int y2 ){
81   layout->drawing->point(x1,y1);
82   layout->drawing->point(x2,y2);
83   layout->drawing->box();
84 }
85 %------------------------------------------------
86 int border ()
87 {
88   int width  = layout->getInteger( "Input", "width(um)" ) * 1000;
89   int offset = layout->getInteger( "Input", "offset(um)" ) * 1000;
90
91   minX -= offset;
92   minY -= offset;
93   maxX += offset;
94   maxY += offset;
95
96   drawBox( minX, minY, maxX, minY + width );
97   drawBox( minX, minY + width, minX + width, maxY - width );
98   drawBox( maxX - width, minY + width, maxX, maxY - width );
99   drawBox( minX, maxY - width, maxX, maxY );
100 }
101 %------------------------------------------------
102 int main(){
103   getExtents();
104   border();
105   return 0;
106 }
```

Appendix B

Fabrication Processes

B.1 Device Fabrication Process Details

B.1.1 Cleaning

1. Ultrasonic immersion in Alconox Detergent 10min, unheated.
2. Deionized H_2O spray, then 10min deionized H_2O soak.
3. Spin dry at 3000rpm.

This procedure is used to remove stubborn particles and organics from heavily contaminated dirty wafers or parts.

Table B.1: Alconox™ cleaning procedure.

1. Ultrasonic cleaning in Acetone 20min, unheated.
2. Spray clean with Acetone then Methanol then Isopropanol, and Spin dry at 3000rpm.

Acetone may also be used to remove stubborn particles and organics. While the wafer is on the coater the Acetone then Methanol then Isopropanol may be applied at $\approx 1000rpm$ to help dislodge particles, and then accelerated to 3000rpm to dry.

Table B.2: Ultrasonic cleaning with Acetone.

1. Mix a 3:1 H_2SO_4:H_2O_2 solution by slowly adding sulfuric acid to hydrogen peroxide
2. Gently heat on hotplate at low temperature to keep the reaction at constant vigor.
3. Immerse sample for 20$minutes$. After 20$minutes$ the efficacy of the piranha mixture is reduced.

Table B.3: Piranha cleaning procedure.

1. Select a large Pyrex dish for spill containment and a sufficiently large Pyrex beaker for the SC-1 solution. Place on a hotplate.
2. Fill the beaker with 4 parts deionized H_2O, then fill the Pyrex dish a 1/4 full of H_2O. Heat to 55℃.
3. Add one part ammonium–hydroxide (NH_4OH) when solution reaches 55℃.
4. Add one part hydrogen–peroxide (H_2O_2) when the water reaches 65℃
5. Insert sample when solution reaches 75℃. Maintain temperature between 75℃ and 80℃.
6. When finished, remove substrate, let solution cool and then dump down drain with lots of water. Rinse the beaker three times before drying.

Table B.4: RCA or standard clean 1 (SC–1) procedure.

B.1.2 Lithography

1. Ready coater by checking the acceleration is at the minimum setting, that the target speed is $3500rpm \pm 100rpm$, and the timer is set for $0 : 60s$. Install the proper containment insert and chuck (check for cleanliness).

2. Mount wafer of the chuck, dust with N_2, and then apply $\approx 0.5mL$ of photo resist covering the useful area (where devices will be located) plus about $1cm$ extra. Wait $10sec$ for photoresist to spread out and interact with the wafer's surface.

3. Spin the wafer, remove and place in a Quartz wafer boat and then bake in the oven at 170℃ for $30minutes$ (PMMA).

General procedure for coating a wafer with photoresist. Bake times and temperatures vary with resists. This procedure is for PMMA. For NR7–1000P bake for $10minute$ at 130℃. The oven was used for SiO_2 and lithium niobate wafers for even heating because they have low thermal conductivity, causing poor results when using the hotplate. An optional $0.45\mu m$ filter may be used when applying the PMMA with a calibrated syringe.

Table B.5: Photoresist coating procedure.

1. Insert the *long–wavelength–filter*. Measure optical power output. Should be $3.5mW/cm^2$ measured at a wavelength of $320nm$.
2. Select the $4inch$ mask chuck, mount the mask pattern–side out (to contact the wafer). Press the `[VAC Mask]` button. Insert and tighten the mask chuck nuts.
3. Insert the wafer with the contact lever disengaged (towards the operator). Adjust position and alignment and press the `[Hard]` contact button.
4. Set the exposure time to $6seconds$ (nominal dose for NR7–1000P, a sweep of exposure times is often necessary to optimize the dose). Engage the contact lever (by moving away from the operator), then press the `[Expose]` button.
5. NR7–1000P must be post–baked for $10minutes$ at 100℃.

Table B.6: Exposing using the EML high–resolution MJB3 aligner.

1. Turn Lamp ON, wait $10sec$. Press Lamp Start, wait $5sec$, green led to the right of the on button should illuminate. If the lamp does not come on, wait $30sec$ and then retry starting the Lamp. Wait $5 - 10min$ for the lamp warm up.
2. Set optical power meter to use the $220nm$ sensor with the switch located on the back of connector. Slide back vacuum chuck with optical power meter sensor. Replace housing door. Turn on exposer timer (the switch is located on the back of the box) and take reading from the power meter. Typical values are in the $0.45mW/cm^2$. Turn off exposer and turn off optical power meter.
3. Adjust exposure time and decimal point. Use multiple exposures for times in excess of $1000sec$.
4. Proceed with substrate/mask alignment. See section C.2. Push back chuck, replace housing door and turn on exposer.

Table B.7: Exposing using the NSL OAI.

1. Ensure enough IPA is available in the spray bottle to stop the developer.
2. Prepare $500mL$ of 2 part IPA to 1 part MIBK developer in $1L$ beaker. Place on a hotplate set initially to $\approx$ 50℃. Heat the solution to exactly 21℃, lowering the hotplate temperature to 30℃ as the solution's temperature rises. Mixing IPA with MIBK causes an endothermic reaction, thus the temperature must be carefully monitored.
3. Remove the developer from the hotplate and immediately start developing the sample for exactly $90seconds$. Immediately spray with IPA for $60seconds$ after wards. Dry with N_2.
4. Oven bake at 95℃ for $30minutes$ to remove residuals. Make sure the oven temperature is below the glass transition temperature to avoid distortions. By increasing the temperature, some scalloping may be removed.

The developer may be reused many times. The $1L$ beaker should be rinsed with a small amount of developer to remove any containments before pouring in the rest of the developer. The developer may be stored at room temperature.

Table B.8: PMMA development procedure.

1. Develop in RD2 at room temperature for $50seconds$ with light agitation.
2. Submerge in deionized H_2O for $2minutes$ to stop development, then dry with N_2.
3. Bake at 90℃ for $1minute$.

This procedure was poorly implemented. The temperature of the developer should be monitored, which may explain the inconsistent results.

Table B.9: NR7-1000P development procedure.

B.1.3 Lift–Off and Etching

---

1. Place sample in a bath of NMP carefully heated to 80℃ (NMP has a low flash–point).

2. Gently agitate the wafer for 20*min*.

3. Carefully remove by spraying a sheet of NMP over the surface of the sample as it is removed from the beaker. Rinse clean with Acetone, Methanol, IPA and then N_2 and spin dry.

Ensure lift–off has completed as any material that has not finished liftoff may fall back onto the wafer and be impossible to remove. Light rubbing with a swab saturated with acetone may help stubborn lift–off and remove unwanted metal that has adhered to surface. Only sonicate as a last resort for no more than 10*seconds* with < 3*seconds* usually being sufficient.

Table B.10: Lift–off of PMMA.

---

1. Place sample in a bath of Microstrip™ and Gently agitate the wafer for 20*min*.

2. Carefully remove by spraying a sheet of Acetone over the surface of the sample as it is removed from the beaker. Rinse clean with Acetone, Methanol, IPA and then N_2 and spin dry.

Table B.11: Lift–off of NR7–1000P.

1. Prepare a $75mL$ of one part deionized H_2O to one part CR–7 chrome etchant in a $250mL$ processing dish at room temperature.

2. Submerse PMMA coated sample in solution for $40seconds$. Spray rinse in deionized H_2O for $2minutes$.

Table B.12: Etching chrome to fabricated a patterned mask.

B.1.4 Vacuum Mask Fabrication

This process involves making a $1mm$ mask master using E-Beam Lithography. The master mask is then used to create $250\mu m$ daughter masks using vacuum mask exposure tool. The daughter masks are then used for making the devices. Daughter masks are very fragile, which is the reason a $1mm$ master mask is used.

The end–to–end PMMA process allows image reversal of the daughter mask. Because the daughter mask is mostly metal, Cr, the conduction should help with the sparking caused by the $LiNbO_3$ handling.

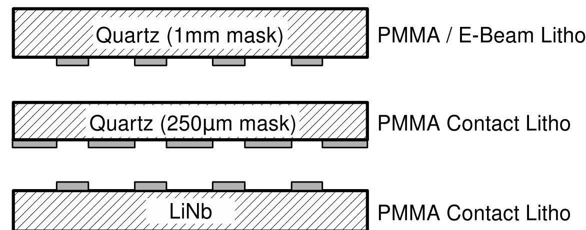


Figure B-1: Illustrates the process for making devices by first writing a pattern to a $1mm$ quartz mask master, then transferring the pattern using contact lithography to a conformal mask, and finally patterning the lithium niobate substrate to make the desired devices.

1. Clean 1mm master mask.
2. Deposit 100nm PMMA.
3. Deposit 7nm chrome.
4. Pattern with SEBL.
5. Remove Cr.
6. Develop PMMA with IPA:MIBK.
7. Descum with 2s ash (optional).
8. Deposit 40nm of chrome.
9. Lift-off with NMP to finish making master mask.
10. Clean 250um daughter mask.
11. Deposit 100nm PMMA onto daughter mask
12. RCA clean and plasma ash master mask as required.
13. Expose daughter mask with OAI.
14. Develop daughter mask with IPA:MIBK.
15. Deposit 40nm of chrome on daughter mask.
16. Lift-off with NMP to finish making daughter mask.

Table B.13: Process steps for making a conformal mask.

B.1.5 Raith 150 Operational Procedure

Loading :

1. Load sample into **load–lock** chamber.
   (a) Deposit $100nm$ gold particles in identifiable location on conductive surface. Make a sketch.
   (b) Ensure all screws are secure and gold solution is dry.
   (c) Check continuity is $< 3Meg\Omega$. aluminum foil.
   (d) Make sure wafer flat is parallel to *x-axis* for alignment.
   (e) N_2 dust and load sample tray ensuring counter–bore hole is aligned, then rotate clockwise until stop.
   (f) **DOUBLE CHECK THAT THE CHUCK HAS BEEN INSERTED CORRECTLY!**
   (g) Then close load–lock door. Ensure **transfer rod** is all the way out.
2. Log into Raith machine. Passwords are not used.
3. `[Desktop 1→Sample Loading→Load Sample]`
4. Ensure **Roughing Pump** becomes **quiet**.
5. **Slowly** insert **transfer rod** when prompted. Then **check insertion** on video screen before continuing.
6. `[OK]` **Move to Upper Exchange Position**.
7. **Slowly** remove **transfer rod** when prompted.
8. `[OK]` **Reset Coordinates System**.
9. Record **Extractor Current** and **Gun Vacuum**.

10. `[OK]` **Switch On Beam**.

11. `[OK]` **Set voltage to** $10keV$.

12. `[OK]` **Set aperture to** $30\mu m$.

13. `[OK]` **Set working distance to default**.

14. `[OK]` **Set focus and stigmation**.

15. Goto the **Stage Control** window's **Destination** tab.

    (a) LENGTH = **mm**
    (b) BASE = **XYZ**
    (c) POSITION = **absolute**
    (d) Z = $16.8mm$

16. Put SEM into TV mode. **DOUBLE CHECK PREVIOUS STEP**

17. `[Start]` , watch motion on TV, if there is a **problem** click the red **STOP** button.

18. `[File→Open Script→SHAKEIT.js]` , press green run button. View in **Height Control** window.

19. `[Tools→Goto Control Panel→SEM Control→Detector→InLens]`
    `[Detector→Auto BC]`
    un-blank beam, check for stable current on Keithley picoampmeter.

20. **Record the Extractor I current**

Focus, Align, and Stigmation :

1. Zoom to 23X and find gold particles

2. Zoom to 100kX, Focus, Stigmation
3. Focus Wobble, Align, then re-Focus.

Align Write Field :

1. Adjust UV coordinates and rotation.
   (a) TV mode, goto wafer flat. Pick first point for rotation, move, pick second point for rotation, `[Adjust Rotation→Adjust]` **Make Sure Rotation Coordinates go from LEFT to RIGHT**
   (b) Blank beam and move to desired write location.
   `[Adjust UVW(Global)→Origin Correction→Adjust]` Sets UV position to (0,0)
2. Image and center gold particles, zoom out to 620X.
3. `[Desktop 2→Microscope Control]`
   `[select (''set'' 620x100`μm`)→Set Button]`
4. `[Detectors→Auto BC Off]` **Ensure Detector is InLens NOT TV.**
5. `[File→New Image]` , `[ImageScan→Single]`
   To move stage, `[CTRL-right-click→F5]`
6. `[File→New Position List]`
   (a) Zoom to features with `[right-click]` menu
   (b) `[Scan Manager→Align write field procedures]`
   drag course, medium, and fine to **Position List**
   (c) Add 100μm WF - Auto ALWF 1μm window
   (d) Select and run each procedure from position list one with `[F9]`

   (e) `[ctrl-left]` and drag to reference feature.

7. Check convergence in Web Browser.

Exposing :

1. Layout a GDSII file, then import to CleWin and the save from CleWin. This corrects various file format problems.

2. Move stage to Faraday Cup
   `[Stage Control→Command→User Positions→OCT2005-CHUCK]`

3. `[Desktop 4]` , Zoom to 100kX, record picoampmeter ≈ $145pA$ at $10keV$, $30\mu m$ aperture. `[Measure]`

4. Add GDSII file and setup exposure

   (a) `[File→New Position List]`
   `[GDSII Database→load file]`
   Drag cell to **Position List**.

   (b) `[Position List→right-click Properties→Expose Properties]`

   (c) `[Expose Layer]` select layers to expose.

   (d) Set UV position to expose at.

   (e) Set `[dose factor→1]`

   (f) `[Calculate Dialog]` fill in step size and dose.
   `[dwell--time calculate icon]`

   (g) `[Times]` to calculate write time.

   (h) `[Desktop 4→Exposure→uncheck line and dot]`

   (i) `[OK]` to close **Expose Properties**

5. Add GDSII dose matrix file and setup exposure

(a) Add GDSII file to **Position List** and setup exposure as in previous step.
(b) `[Expose Properties→Calculate→Dose]` less than main file.
(c) Select Dose Matrix cell in **Position List**
`[Filter→Matrix Copy]`
{selected,1x6,0x(> device height),V-first,dose factor 0.1, add, auto}

6. Check UV coordinates in **Position List**.

7. Record all settings.

8. **MAKE SURE SEM IS IN SEM MODE (NOT TV MODE)!!**

9. `[Beam Blank]` Check Raith beam is **blanked**.

10. `[Position List→Highlight Entries]`
`[Scan→All]`

11. Check

    (a) Proper XY and UV coordinates
    (b) High Performance Scan Processor, Beam On
    (c) Picoampmeter.

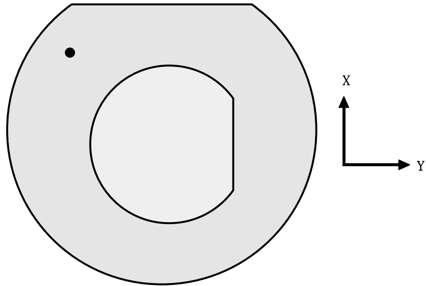
Unloading :

1. `[Desktop 1→Sample Loading→UnLoad Sample]`

2. Insert **transfer rod** when prompted. Then **check insertion** on video screen before continuing.

3. `[OK]` **Move to Lower Exchange Position**.

4. Remove **transfer rod** when prompted.



Date	
Wafer	
Write #	

Vacuum Before ON			
Extract-I Before ON			
Extract-I After ON			
Voltage			
Working Distance			
Stigmation *(x,y)*			
Aperture			
Beam Current			
Gold (x,y)			
Field Size			
Area Step			
Dwell Time			
Area Dose			
Beam Speed			
Est. Write Time			
Start Write Time			
Write Time			
Dose Matrix			
(U,V)→(x,y)			
Rotation			

Appendix C

OAI documentation

C.1 Housing Drawings

The OAI deep–UV vacuum mask lithography tool in NSL was upgraded to include a radiation shield over the working area. The shield was made of acrylic which attenuates $\lambda = 200nm$ by $60dB$ and $\lambda = 440nm$ by $40dB$. To simplify the design no fasteners were used. Instead the housing was designed with press–fit slot in hole joints. 4-40 tapped holes were drilled along the edges for gaskets. Two clean room corrosion–resistant magnetic catches (McMaster part number 6680A15) were used to keep the front door in place. The patterns were drawn in Pro–Engineer, then exported as DXF, imported into CorelDraw, and then printed using a laser–cutter at the Media Laboratory's fabrication facilities.

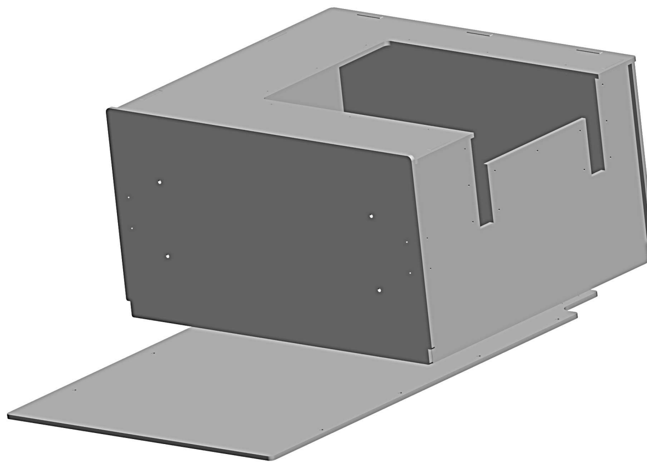
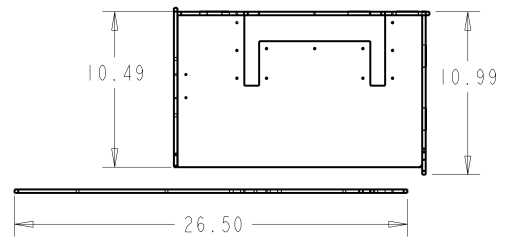
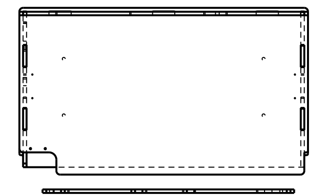
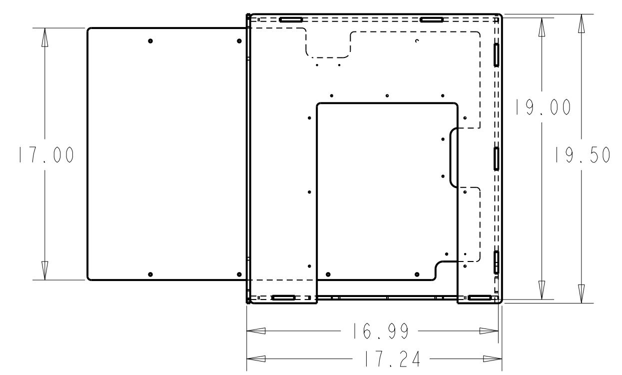


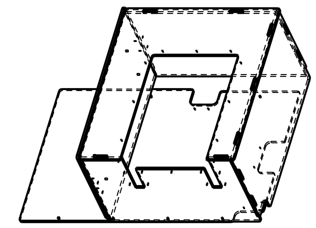
Figure C-1: OAI deep–UV aligner radiation shield. Made from Standard Cast Acrylic, 0.236*inch* thick Clear with Bronze Tint, McMaster part numbers. 8505K921, 8505K941, and 8505K951.





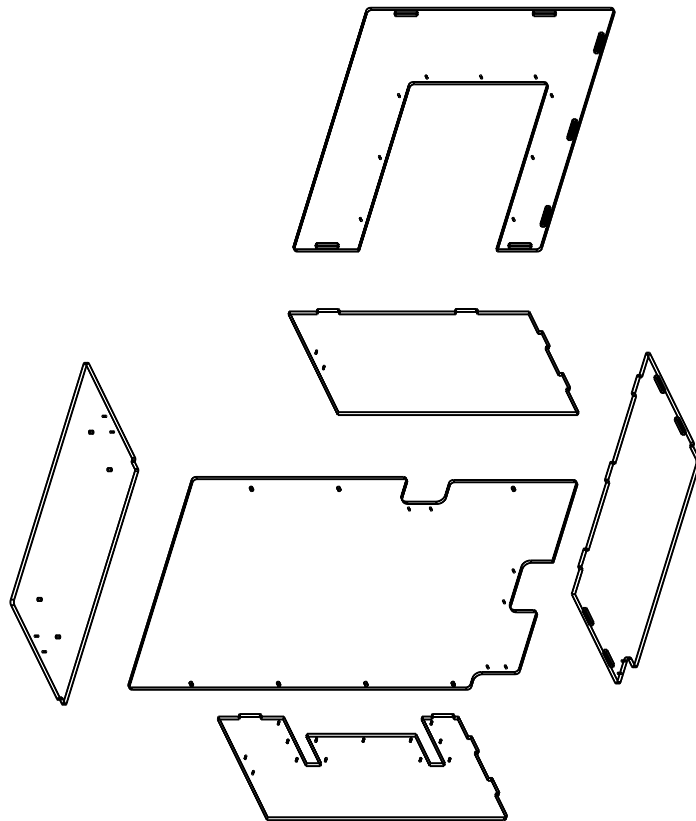
SCALE 0.150



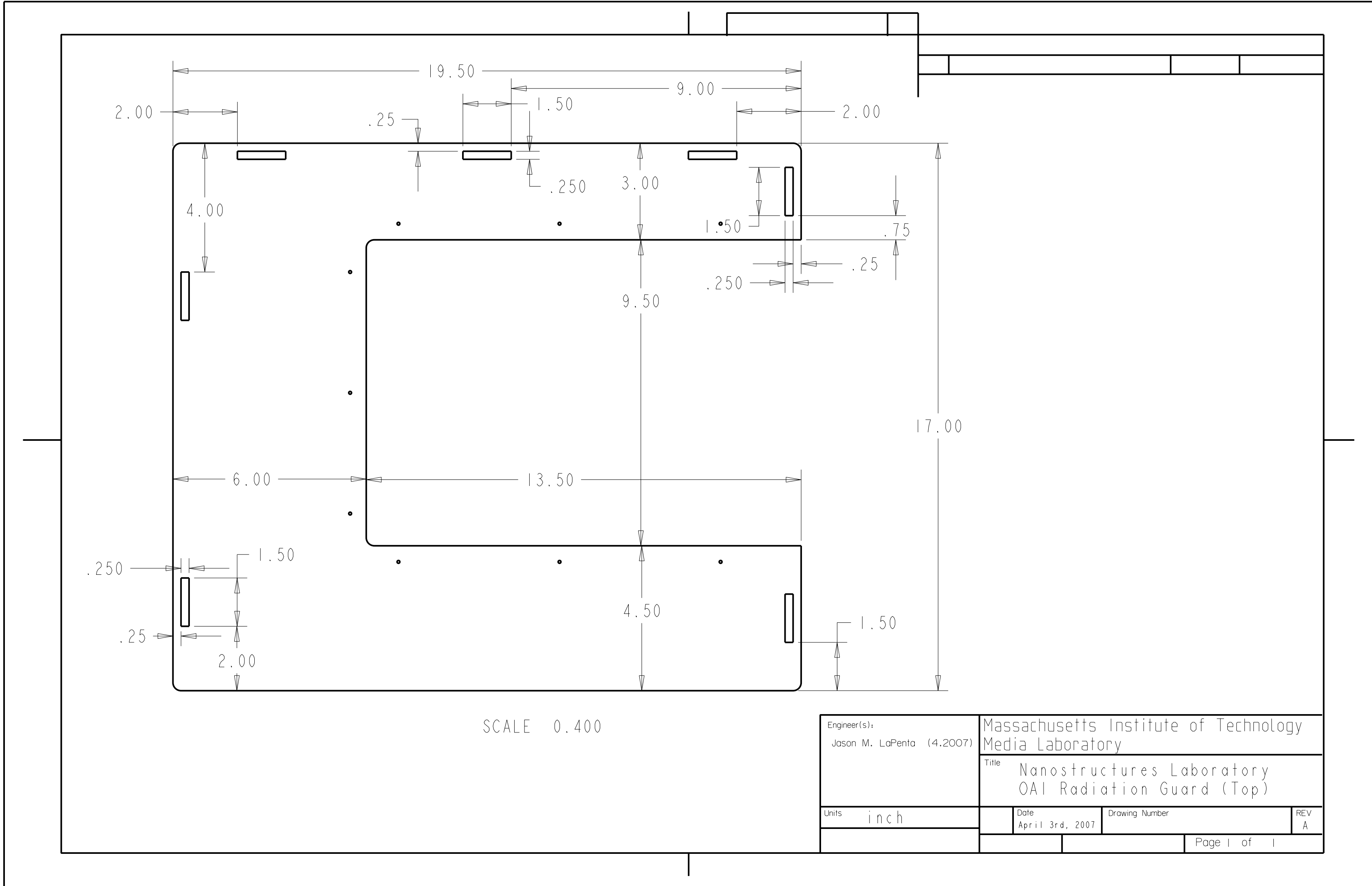


SCALE 0.100

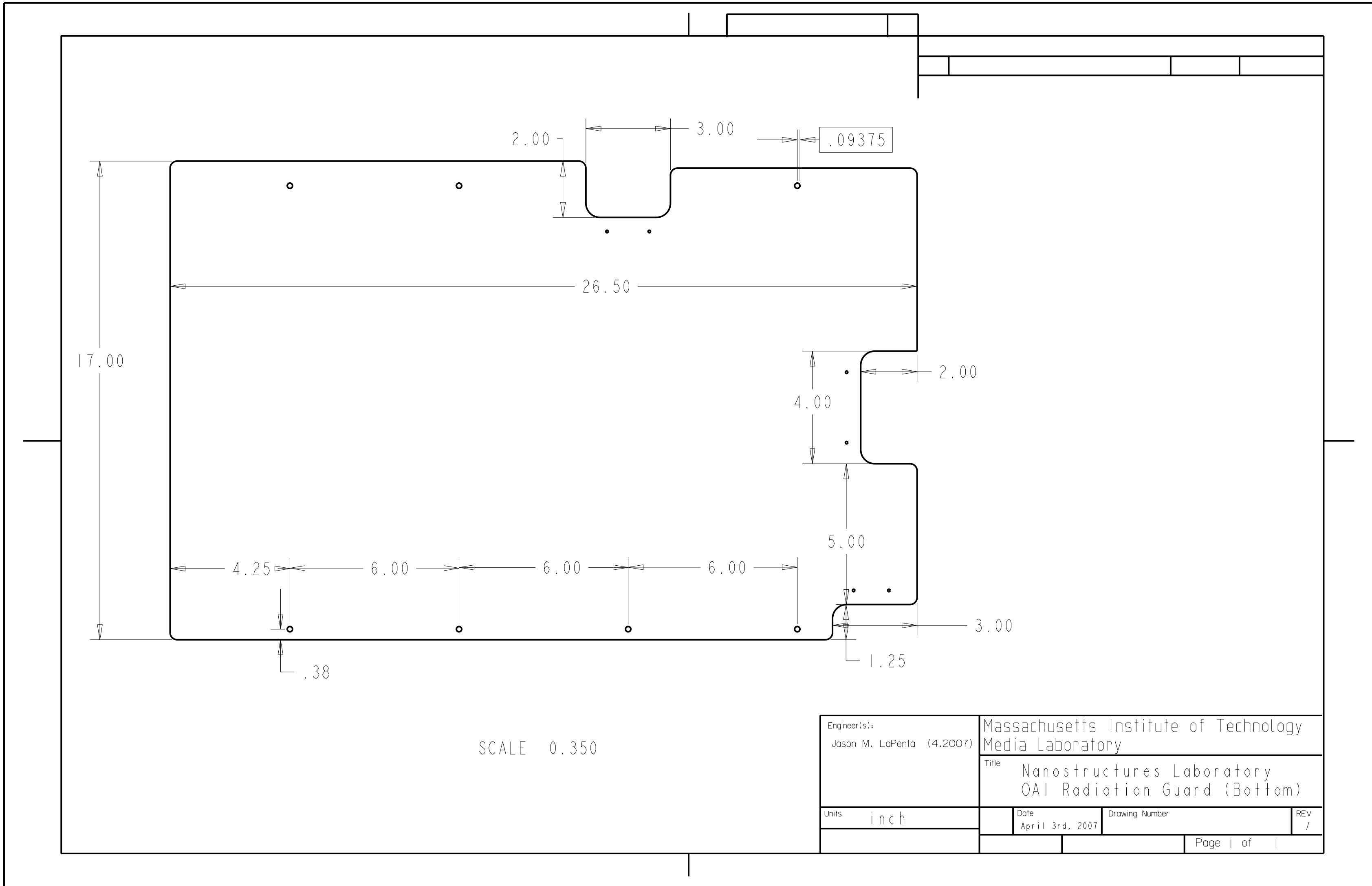
Engineer(s): Jason M. LaPenta (4.2007)	Massachusetts Institute of Technology Media Laboratory		
	Title: Nanostructures Laboratory OAI Radiation Guard		
Units: inch	Date: April 3rd, 2007	Drawing Number	REV: A
		Page 1 of 2	



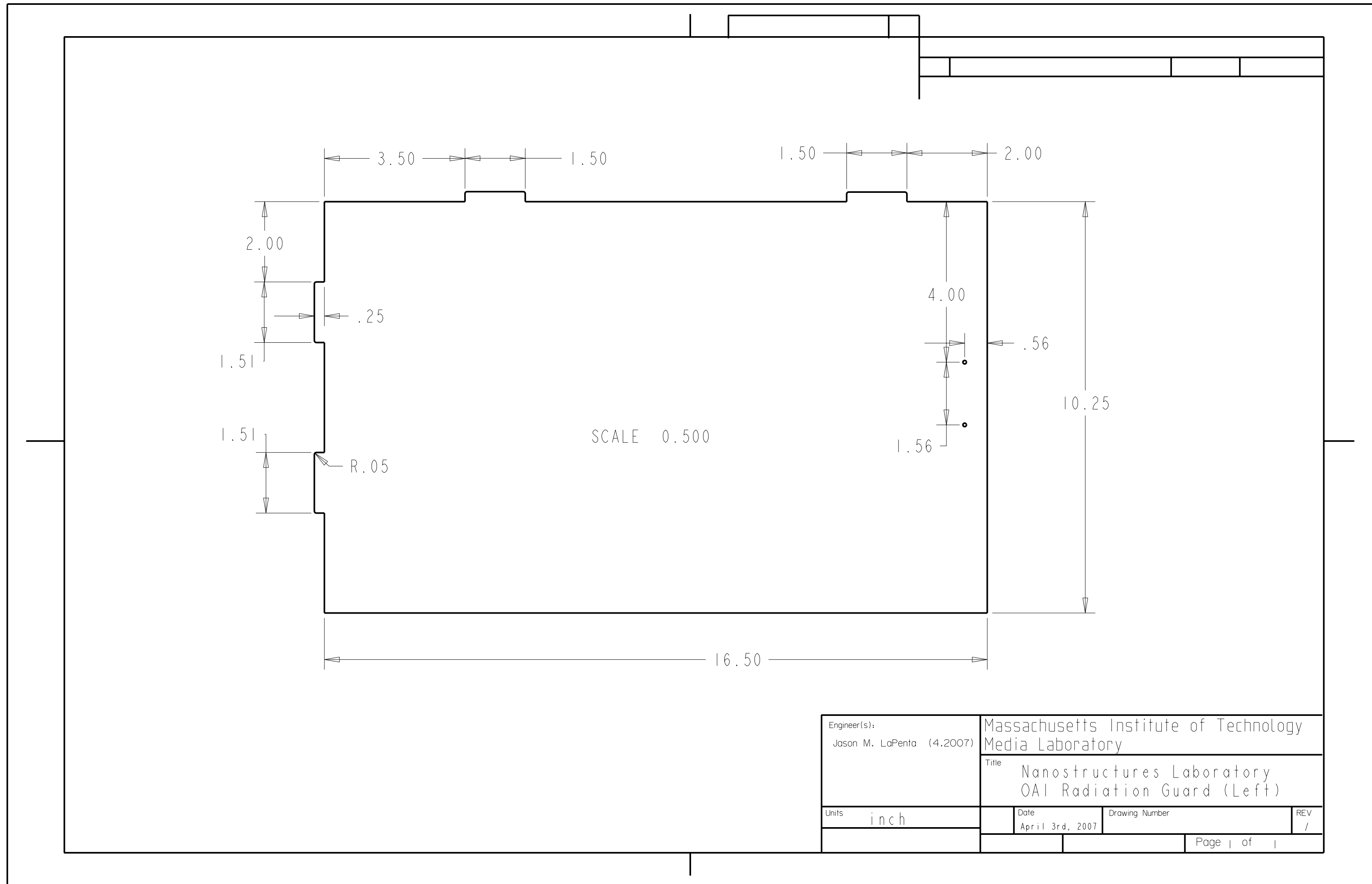
Engineer(s):	Massachusetts Institute of Technology Media Laboratory
Jason M. LaPenta (4.2007)	Title Nanostructures Laboratory OAI Radiation Guard
Units inch	Date April 3rd, 2007 · Drawing Number · REV A
	Page 2 of 2

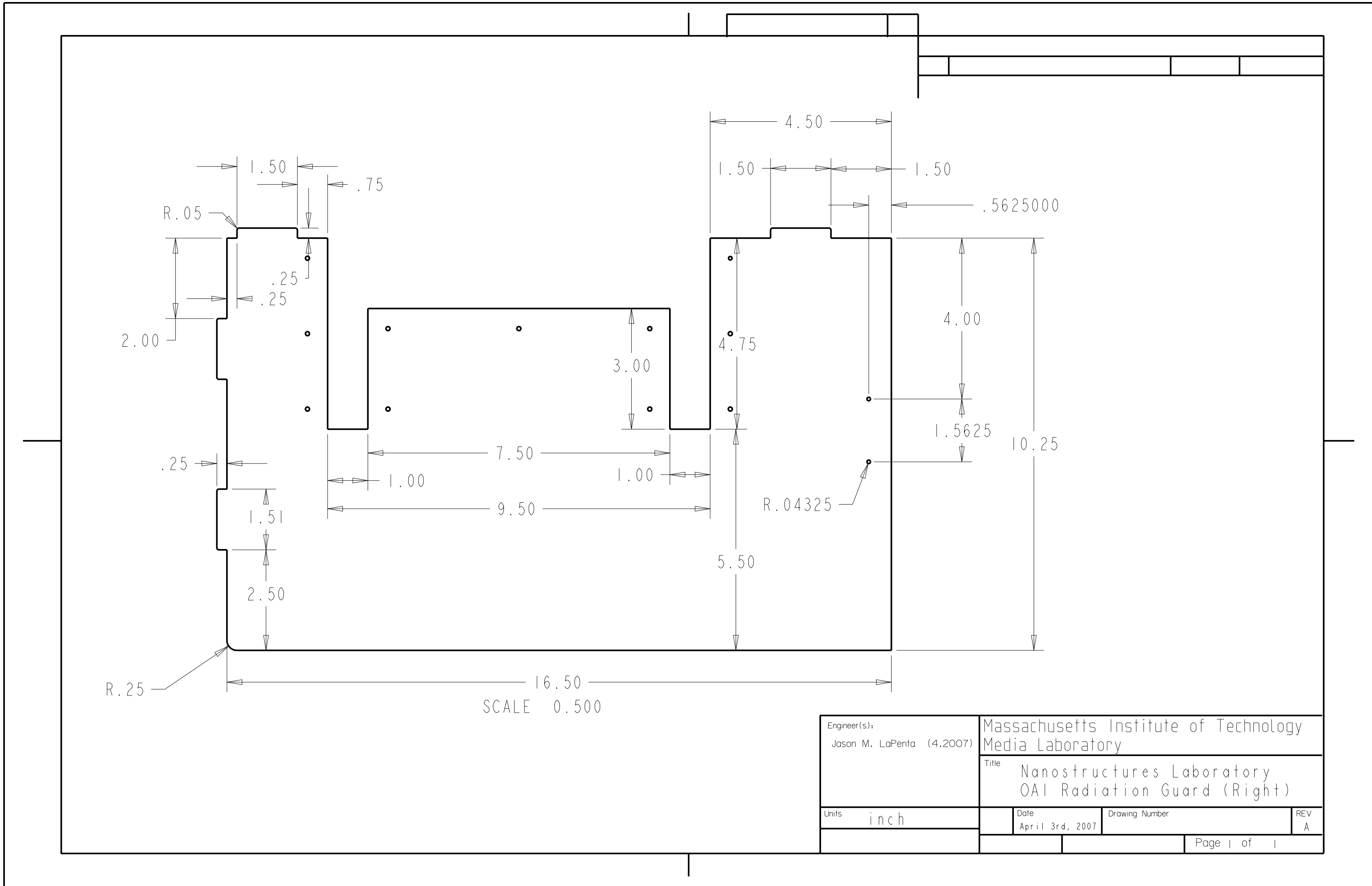


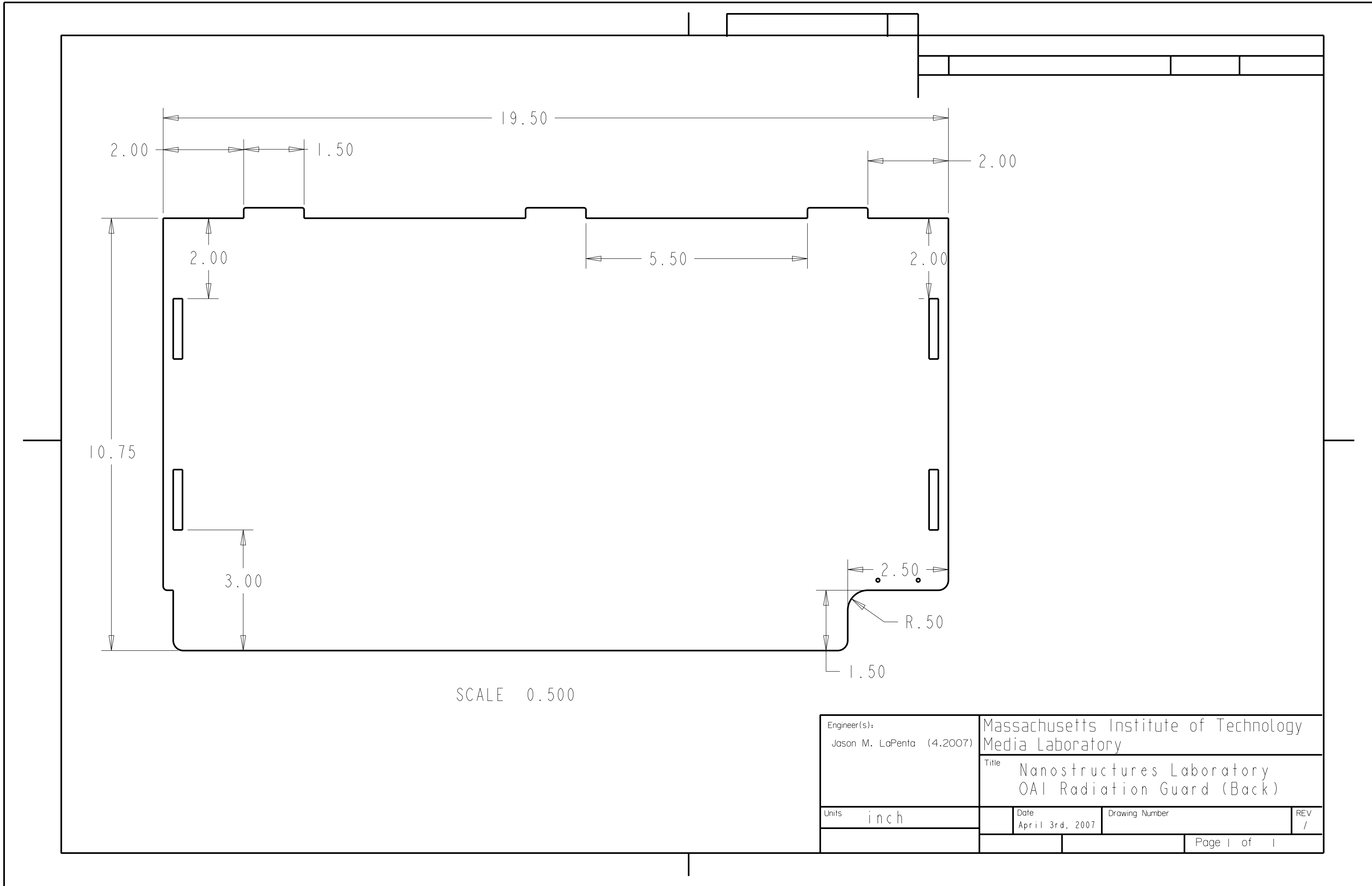
19.50
9.00
2.00
1.50
2.00
.25
.250
3.00
4.00
1.50
.75
.25
.250
9.50
17.00
6.00
13.50
.250
1.50
4.50
1.50
.25
2.00
SCALE 0.400
Engineer(s):
Jason M. LaPenta (4.2007)
Massachusetts Institute of Technology
Media Laboratory
Title
Nanostructures Laboratory
OAI Radiation Guard (Top)
Units
inch
Date
April 3rd, 2007
Drawing Number
REV
A
Page 1 of 1



2.00
3.00
.09375
26.50
17.00
2.00
4.00
5.00
4.25
6.00
6.00
6.00
3.00
1.25
.38
SCALE 0.350
Engineer(s):
Jason M. LaPenta (4.2007)
Massachusetts Institute of Technology
Media Laboratory
Title
Nanostructures Laboratory
OAI Radiation Guard (Bottom)
Units
inch
Date
April 3rd, 2007
Drawing Number
REV
/
Page 1 of 1


3.50
1.50
1.50
2.00
2.00
.25
1.51
4.00
.56
10.25
1.51
SCALE 0.500
1.56
R.05
16.50
Engineer(s):
Jason M. LaPenta (4.2007)
Massachusetts Institute of Technology
Media Laboratory
Title
Nanostructures Laboratory
OAI Radiation Guard (Left)
Units
inch
Date
April 3rd, 2007
Drawing Number
REV
/
Page 1 of 1


4.50
1.50
1.50
.75
1.50
.5625000
R.05
.25
.25
2.00
4.00
3.00
4.75
1.5625
10.25
7.50
.25
1.00
1.00
9.50
R.04325
1.51
5.50
2.50
R.25
16.50
SCALE 0.500
Engineer(s):
Jason M. LaPenta (4.2007)
Massachusetts Institute of Technology
Media Laboratory
Title
Nanostructures Laboratory
OAI Radiation Guard (Right)
Units
inch
Date
April 3rd, 2007
Drawing Number
REV
A
Page 1 of 1


19.50
2.00
1.50
2.00
2.00
5.50
2.00
10.75
3.00
2.50
R.50
1.50
SCALE 0.500
Engineer(s):
Jason M. LaPenta (4.2007)
Massachusetts Institute of Technology
Media Laboratory
Title
Nanostructures Laboratory
OAI Radiation Guard (Back)
Units
inch
Date
April 3rd, 2007
Drawing Number
REV
/
Page 1 of 1

C.2 Mask Chuck Drawings

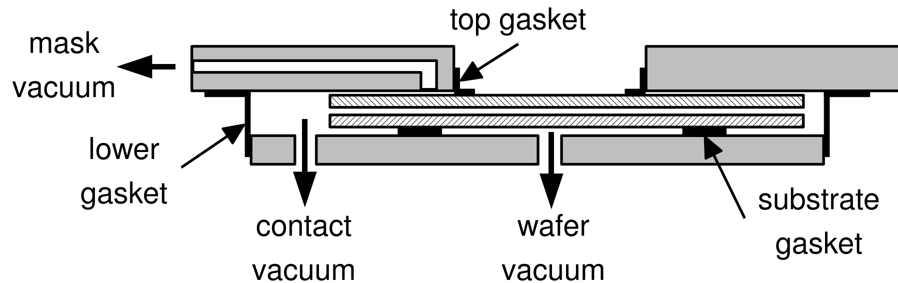


Figure C-2: OAI 3*inch* chuck assembly.

A new OAI 3*inch* vacuum mask chuck was designed for use on this project. The existing mask chuck required 5*inch* mask and operated in a different manner. The 3*inch* chuck assembly is illustrated in figure C-2. The lower plate (shown in figure **??** but not in figure C-2) and substrate chuck were pre–existing. The 3*inch* upper chuck plate was the new part. This part was machined in the MIT Central Machine Shop. The substrate gasket allows atmospheric pressure to be applied to the backside of the substrate, as well as keep the substrates back side from contacting the chuck (useful when making a mask). The top gasket makes the seal to the mask and the lower gasket make the seal to the chuck for the vacuum contact. The mask vacuum holds the mask to the chuck during alignment. There are three supports underneath the chuck as part of the lower plate used to level the mask to the substrate. An xy–level is used to make sure the mask chuck and substrate chuck are perfectly parallel (usually only done once).

The operational procedure for alignment and contact is as follows;

- Lower substrate chuck to bottom of limit.
- Put down substrate gasket, substrate, and apply substrate vacuum.
- Mount mask to chuck and apply mask vacuum.

- Place mask chuck onto 3–point support.
- Raise substrate stage until fringe patterns are visible, then back off a turn.
- Use the microscope to align mask to substrate.
- Raise substrate stage until to contact mask to substrate.
- Engage contact vacuum, slowly increasing pressure with the adjustable ball valve.
- Turn off substrate vacuum to allow backside pressure on the substrate.

The valves originally on the OAI were single throw needle valves. These valves do not vent, meaning the sealed vacuum chambers would not release after an exposure. To allow for self–venting and some amount of vacuum throttling, the valves were replace with SwageLok part number B-41VS2 brass vented ball valves.

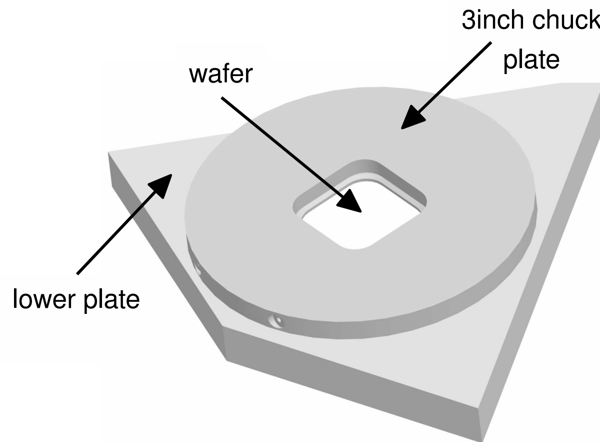
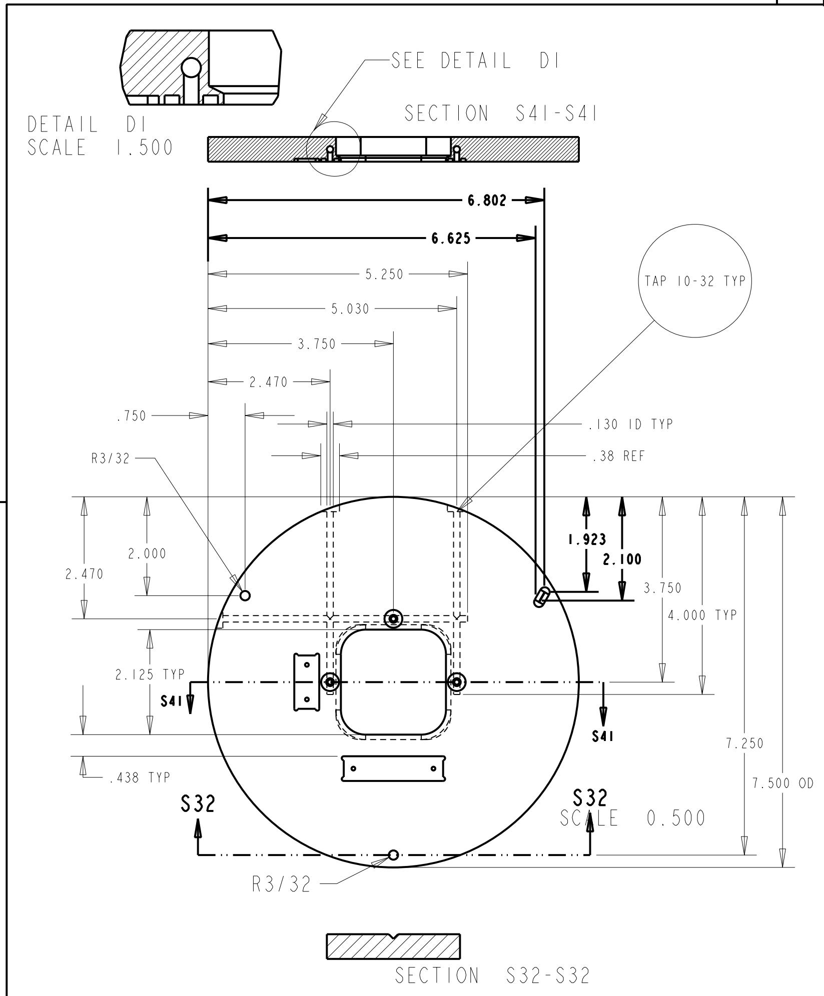
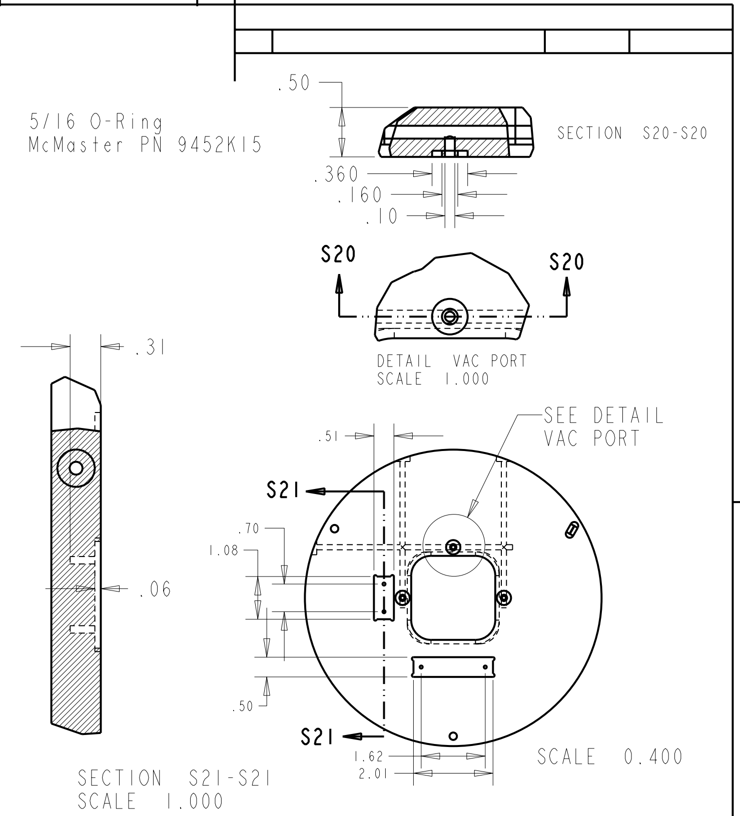


Figure C-3: OAI 3*inch* chuck assembly.

The machining cost for this chuck was nearly $1600.00. An important note is that parts with more drawings cost more than parts with fewer drawings. The more sheets a machinist has to flip through the higher likelihood of a mistake and the more time is taking matching up features on different pages. For this reason, only one drawing page is used to specify this part.





Engineer(s):
Jason M. LaPenta (4.2007)
the5@mit.edu

Massachusetts Institute of Technology
Media Laboratory

Title
Nanostructures Laboratory
OAI 3inch Mask Chuck

Units

Date
April 10, 2007

Drawing Number

REV
D

Page 1 of 1

Appendix D

Auxiliary Electronics

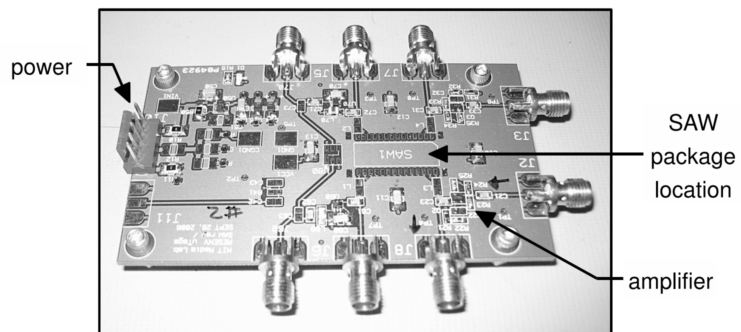


Figure D-1: Analog test board. Holds SAW test package, low–noise amplifiers, and power regulators.

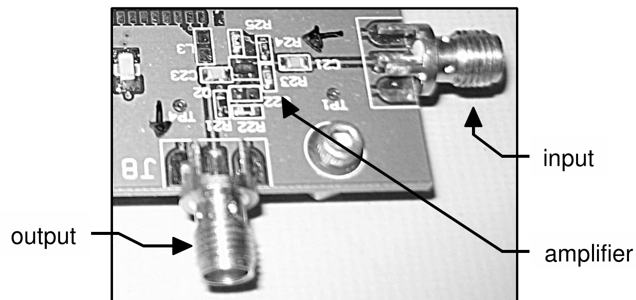


Figure D-2: Closeup of one low–noise amplifier channel.

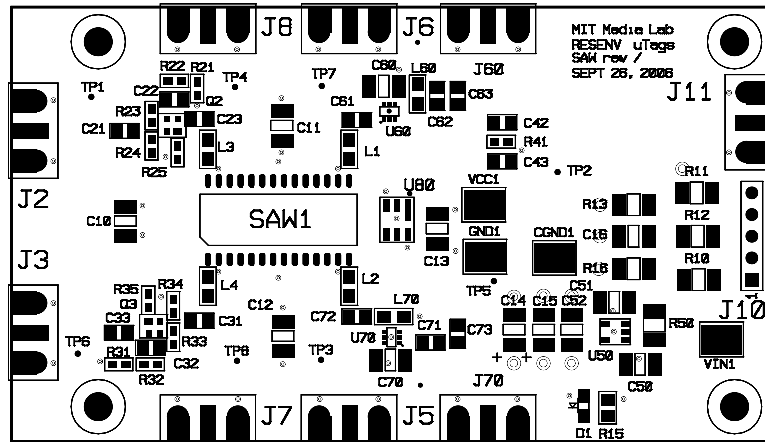


Figure D-3: PCT top–side silkscreen.

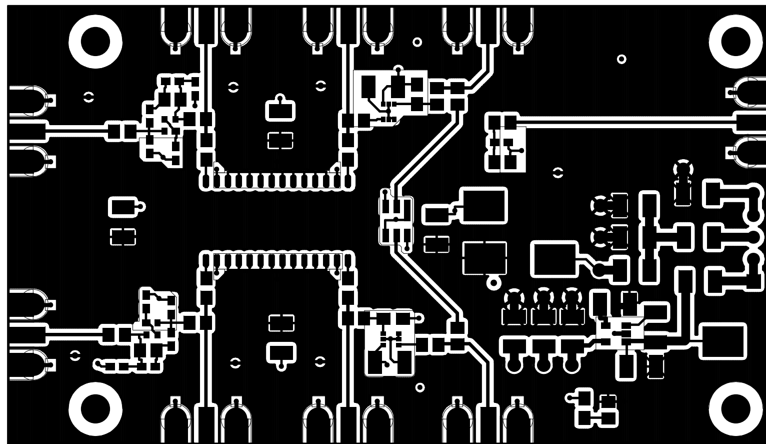
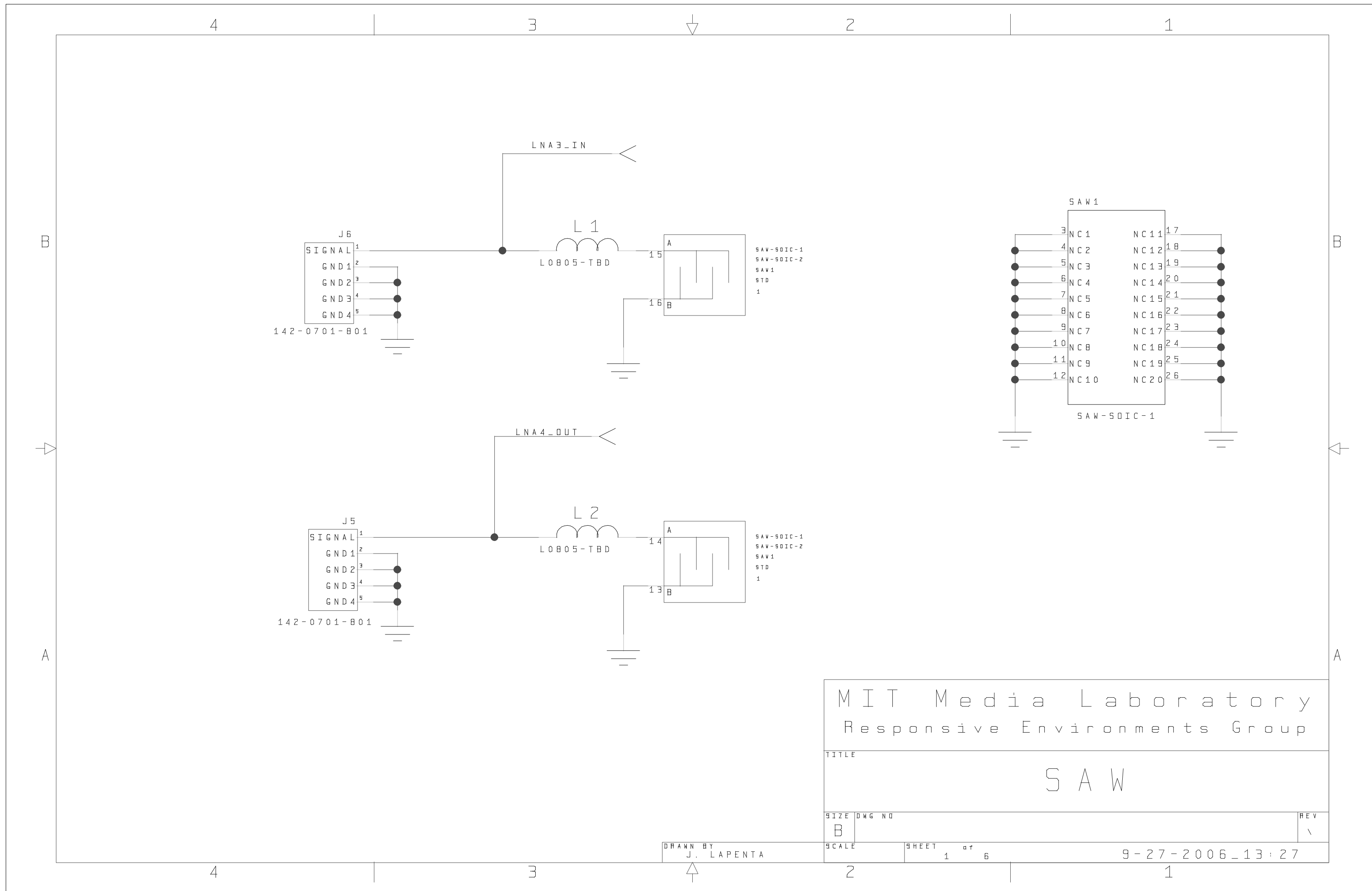
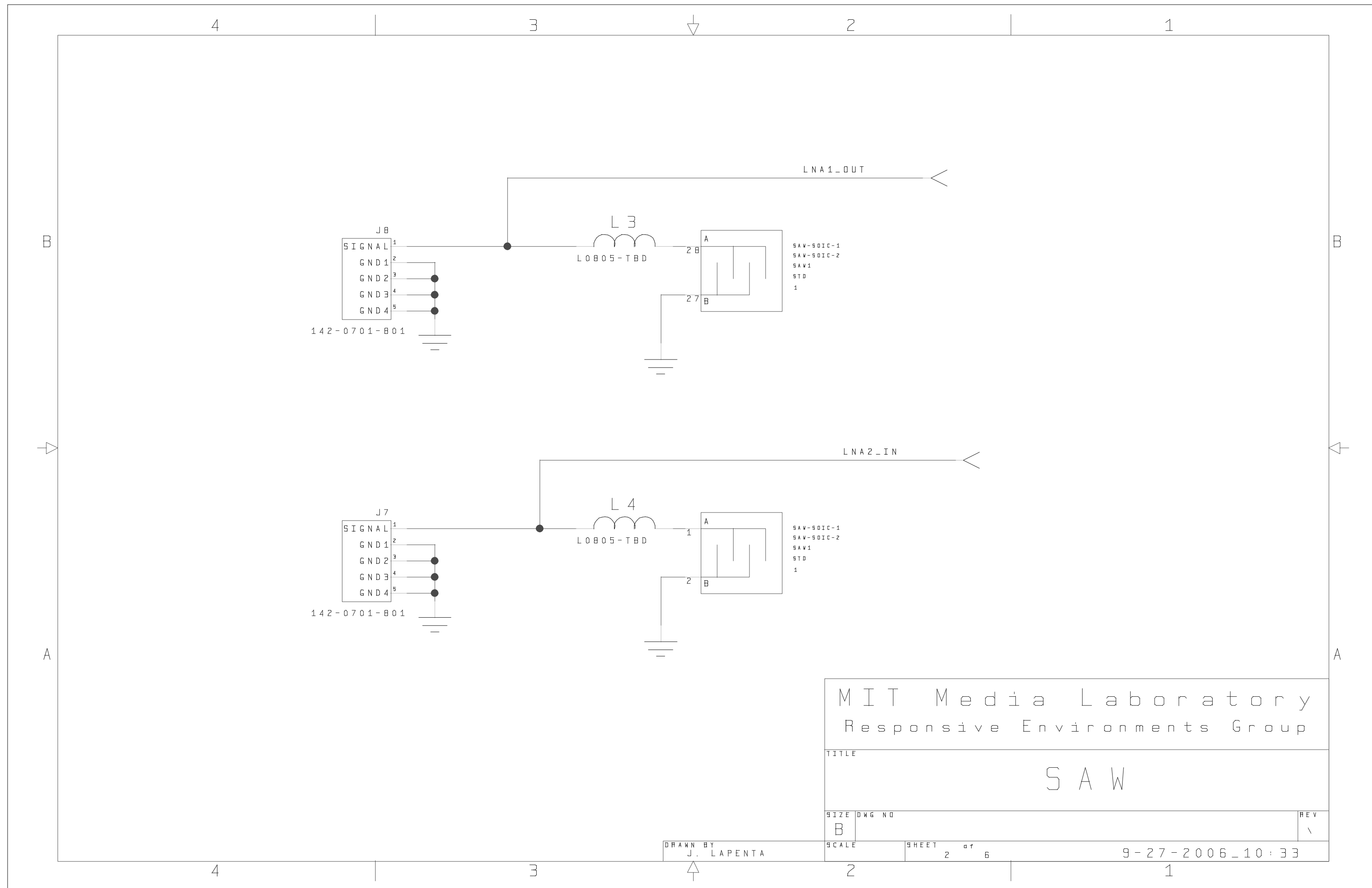


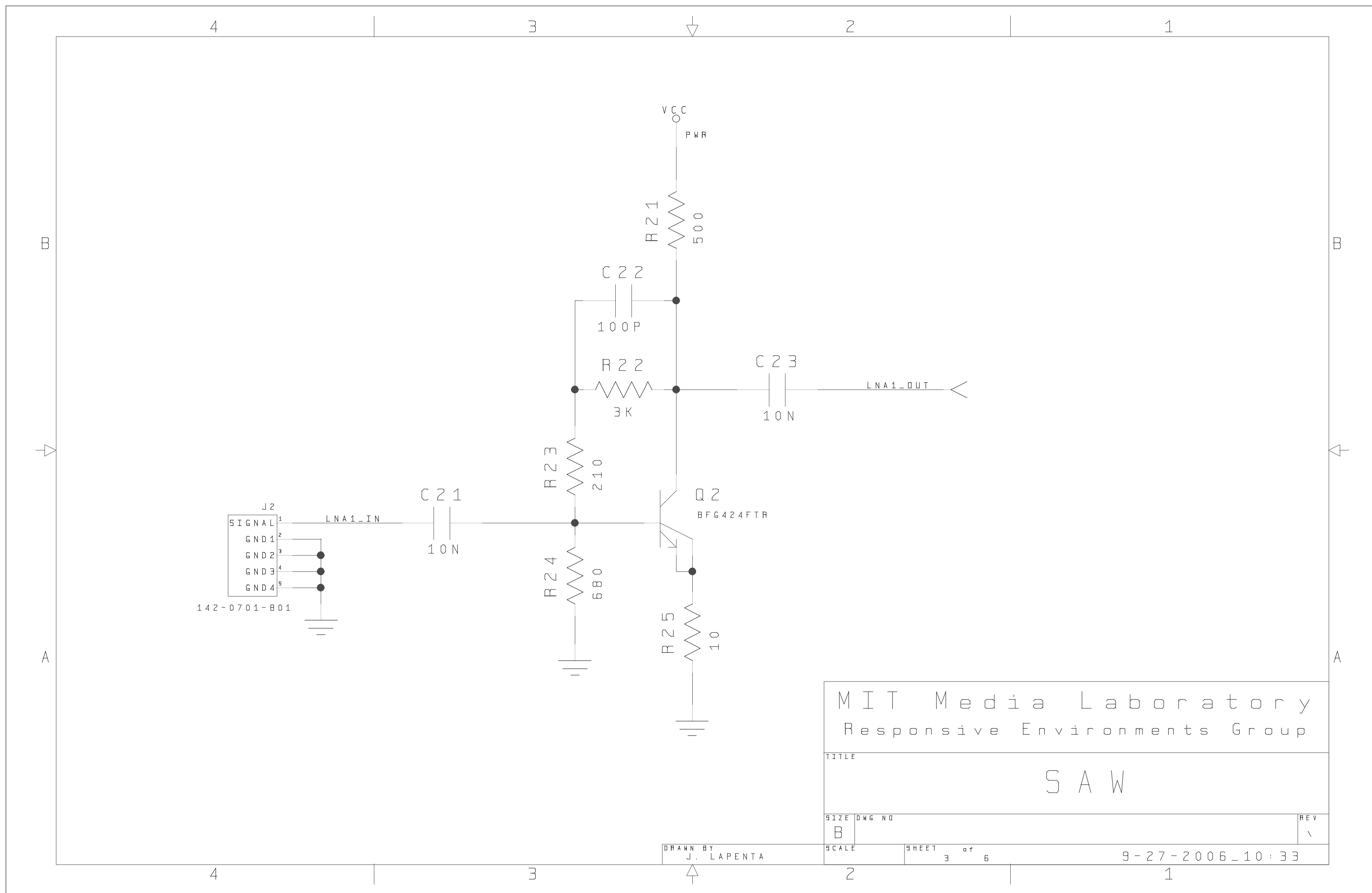
Figure D-4: PCB top–side metal layer.

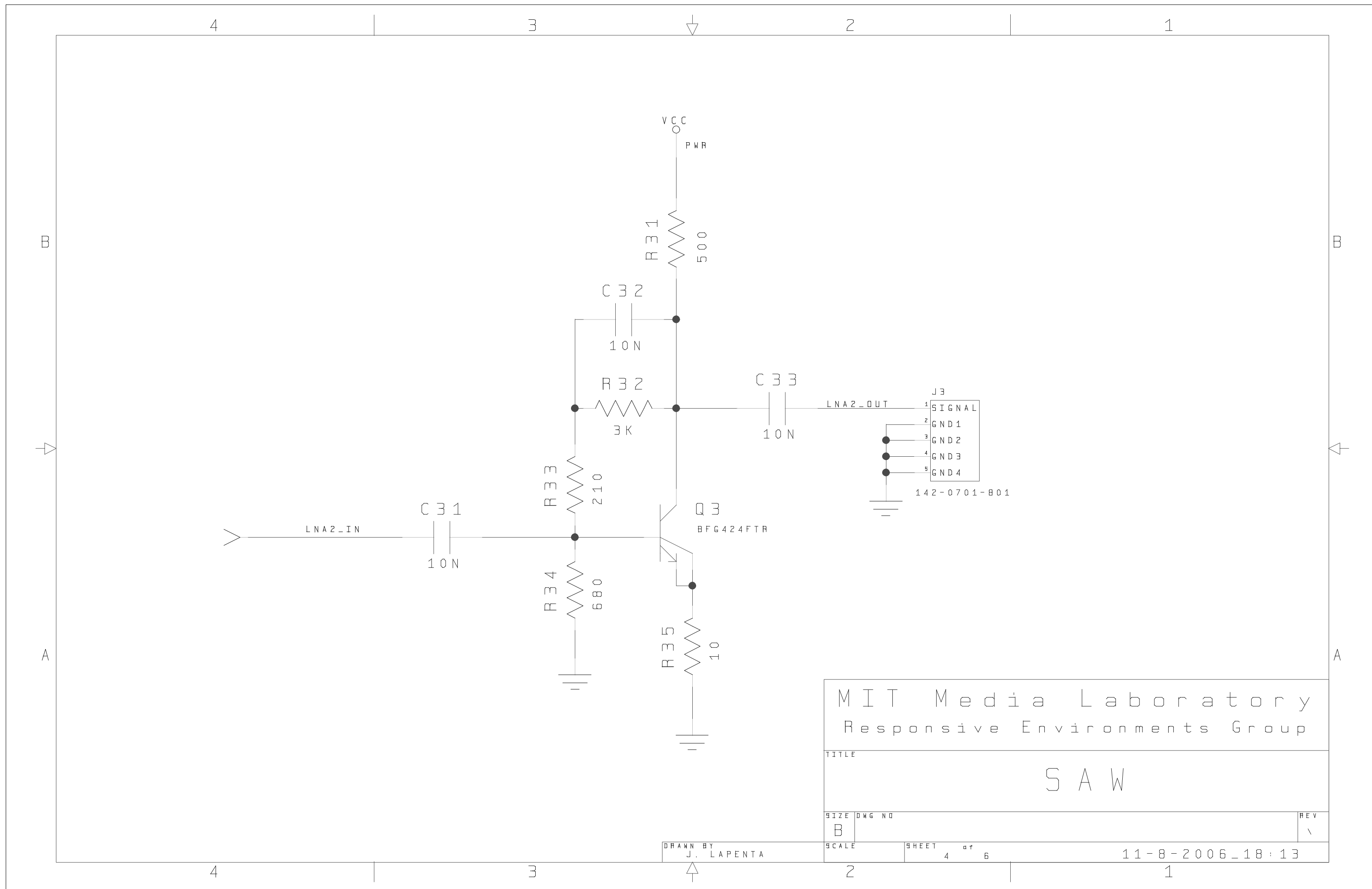
	QTY	REFDES	DEVICE	PACKAGE	VALUE
1	5	C10,C11,C12,C13,C16	C1210-TBD	CAP	$2.2\mu F$
2	2	C14,C15	C1210P-TBD	CAP	$22\mu F$
3	13	C21,C23,C31,C32,C33 C42,C43,C61,C62,C63 C71,C72,C73	C0805-TBD	CAP	$10nF$
4	1	C22	C0805-TBD	CAP	$100pF$
5	1	C50	C1210-TBD	CAP	$1\mu F$
6	1	C51	C1210-TBD	CAP	$10nF$
7	1	C52	C1210-TBD	CAP	$2.2\mu F$
8	2	C60,C70	C1210-TBD	CAP	$1nF$
9	4	CGND1,GND1,VCC1,VIN1	5016KCT	TP1	
10	1	D1	LTST-C170KGKT	TO-5	
11	9	J2,J3,J5,J6,J7,J8 J11,J60,J70	142-0701-801	COAX-1	
12	1	J10	640453-5	SIP-5P	
13	4	L1,L2,L3,L4	L0805-TBD	94Z	
14	2	L60,L70	L0805-TBD	94Z	1U
15	2	Q2,Q3	BFG424FTR	TO-18	
16	6	R10,R11,R12,R13,R16 R50	R1210-TBD	RES	0Ω
17	1	R15	R0805-TBD	RES	210Ω
18	2	R21,R31	R0603-TBD	RES	500Ω
19	2	R22,R32	R0603-TBD	RES	$3K\Omega$
20	2	R23,R33	R0603-TBD	RES	210Ω
21	2	R24,R34	R0603-TBD	RES	680Ω
22	2	R25,R35	R0603-TBD	RES	10Ω
23	1	R41	R0603-TBD	RES	0Ω
24	1	SAW1	SAW-SOIC-1	SAW-1-NC	
25	1	U50	LP2985A-18DBVR		
26	2	U60,U70	UPC2711TB	AMP-6PINS-1	
27	1	U80	B3710	SAW_6	

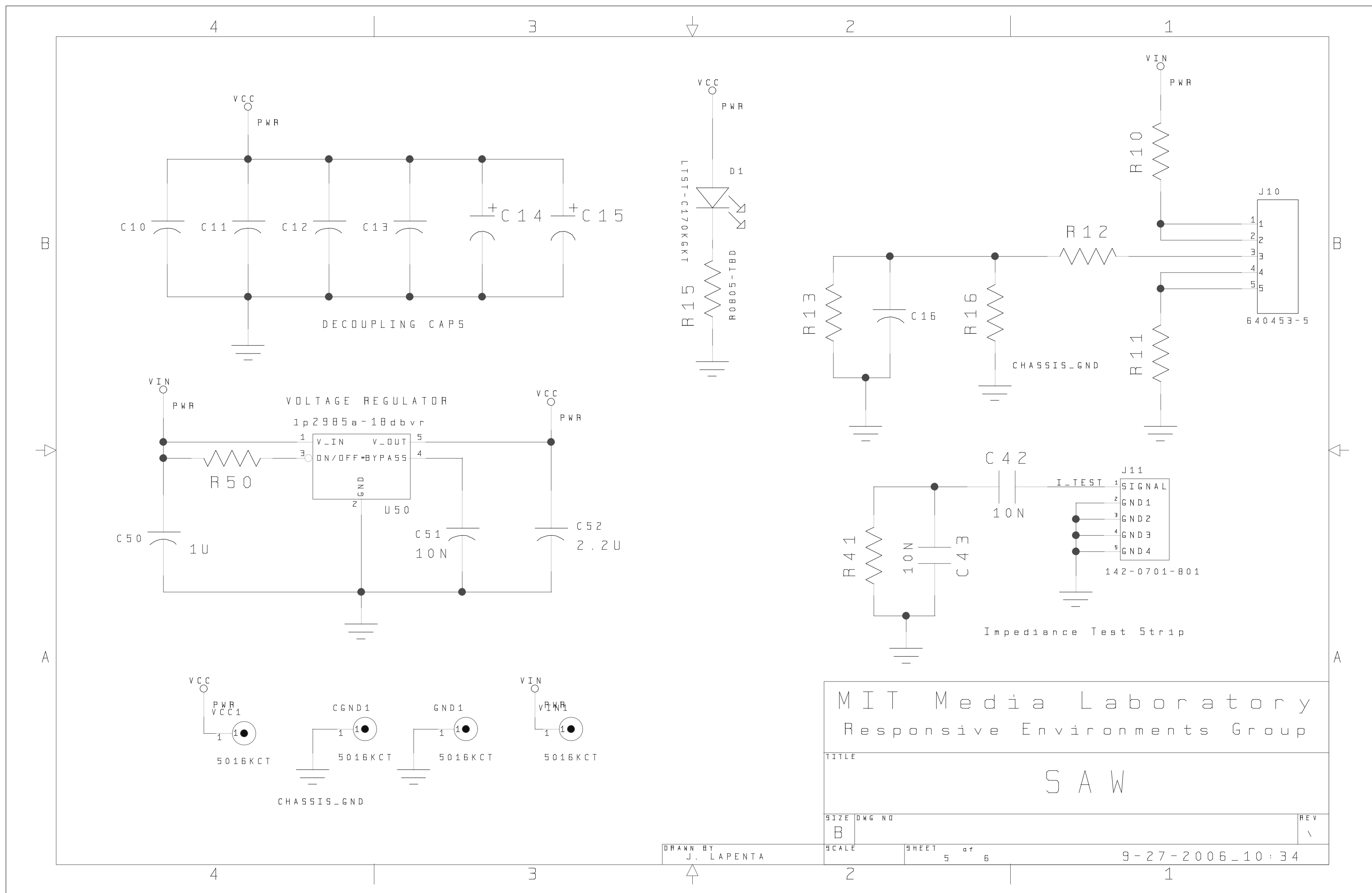
Table D.1: Parts list for test board


LNA3_IN
J6
SIGNAL
GND1
GND2
GND3
GND4
142-0701-801
L1
L0805-TBD
SAW-SOIC-1
SAW-SOIC-2
SAW1
STD
LNA4_OUT
J5
L2
SAW1
SAW-SOIC-1
MIT Media Laboratory
Responsive Environments Group
TITLE
SAW
SIZE
B
DWG NO
REV
DRAWN BY
J. LAPENTA
SCALE
SHEET 1 OF 6
9-27-2006_13:27

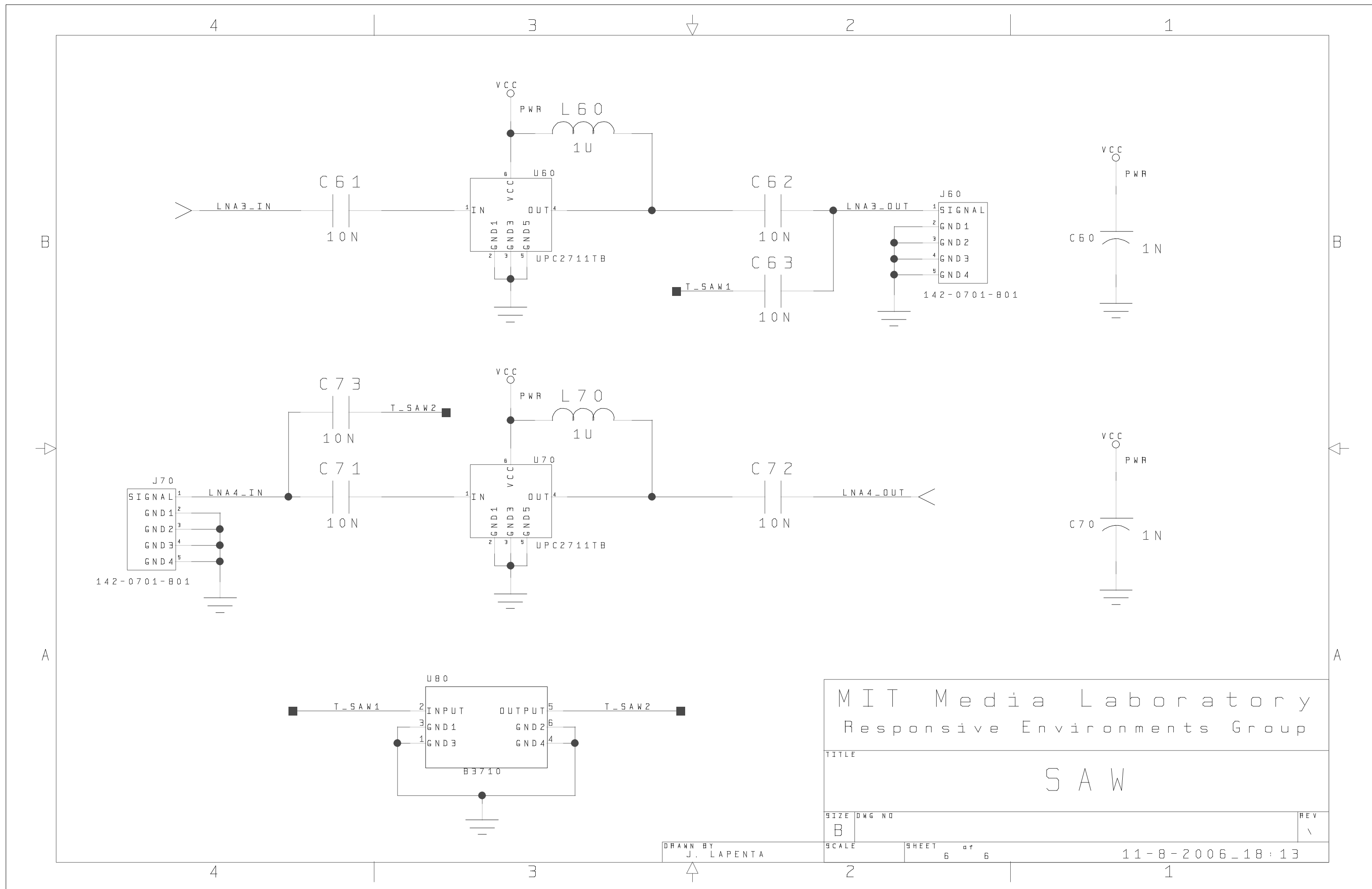

LNA1_OUT
J8
SIGNAL
GND1
GND2
GND3
GND4
142-0701-801
L3
L0805-TBD
28
27
A
B
SAW-SOIC-1
SAW-SOIC-2
SAW1
STD
LNA2_IN
J7
SIGNAL
GND1
GND2
GND3
GND4
142-0701-801
L4
L0805-TBD
MIT Media Laboratory
Responsive Environments Group
TITLE
SAW
SIZE
B
DWG NO
REV
DRAWN BY
J. LAPENTA
SCALE
SHEET 2 of 6
9-27-2006_10:33


VCC
PWR
R21
500
C22
100P
R22
3K
C23
10N
LNA1_OUT
R23
210
C21
10N
Q2
BFG424FTR
J2
SIGNAL
GND1
GND2
GND3
GND4
142-0701-801
LNA1_IN
R24
680
R25
10
MIT Media Laboratory
Responsive Environments Group
TITLE
SAW
SIZE
B
DWG NO
REV
DRAWN BY
J. LAPENTA
SCALE
SHEET
3 of 6
9-27-2006_10:33


VCC
PWR
R31
500
C32
10N
R32
3K
C33
10N
LNA2_OUT
J3
SIGNAL
GND1
GND2
GND3
GND4
142-0701-801
R33
210
C31
10N
LNA2_IN
Q3
BFG424FTR
R34
680
R35
10
MIT Media Laboratory
Responsive Environments Group
TITLE
SAW
SIZE
B
DWG NO
REV
DRAWN BY
J. LAPENTA
SCALE
SHEET 4 of 6
11-8-2006_18:13



DECOUPLING CAPS
VOLTAGE REGULATOR
lp2985a-18dbvr
Impediance Test Strip
MIT Media Laboratory
Responsive Environments Group
SAW
DRAWN BY J. LAPENTA
9-27-2006_10:34



VCC
PWR
L60
1U
C61
U60
LNA3_IN
IN
OUT
VCC
GND1
GND3
GND5
10N
UPC2711TB
C62
LNA3_OUT
J60
SIGNAL
GND1
GND2
GND3
GND4
10N
C63
T_SAW1
10N
142-0701-801
C60
1N
C73
T_SAW2
10N
L70
1U
C71
U70
J70
LNA4_IN
10N
UPC2711TB
C72
LNA4_OUT
10N
142-0701-801
C70
1N
U80
T_SAW1
INPUT
GND1
GND3
OUTPUT
GND2
GND4
T_SAW2
B3710
MIT Media Laboratory
Responsive Environments Group
TITLE
SAW
SIZE
B
DWG NO
REV
DRAWN BY
J. LAPENTA
SCALE
SHEET
6 of 6
11-8-2006_18:13

Appendix E

FPGA VHDL/C/XPS

VHDL : rocketio_glue.vhd

```
1   --------------------------------------------------------------------------------
2   -- Copyright 2007 Massachusetts Institute of Technology : All rights reserved
3   --
4   -- Company:      Massachusetts Institute of Technology
5   -- Engineer:   Jason M. LaPenta (JML)
6   --
7   -- Create Date:      13:30:08 02/13/2007
8   -- Design Name:      uTags - chirp2_rio
9   -- Module Name:      rocketio_test - Behavioral
10  -- Project Name:    uTags
11  -- Target Devices: Virtex4 ML405 devel board
12  -- Tool versions:
13  --
14  -- Description: Test configuration of a rocket_io port. Transmitts a pattern
15  -- cyclically
16  --
17  --
18  --
19  -- Dependencies:
20  --
21  -- Revision:
22  -- Revision 0.01 - File Created
23  --
24  --    Changes :
25  --
26  --     Date        Who       Description
27  --    ----------------------------------------------------------------
28  --
29  -- Additional Comments:
30  --
31  --------------------------------------------------------------------------------
32
```

```
33  library IEEE;
34  use IEEE.STD_LOGIC_1164.all;
35  use IEEE.STD_LOGIC_ARITH.all;
36  use IEEE.STD_LOGIC_UNSIGNED.all;
37  use IEEE.numeric_std.all;
38
39  ---- Uncomment the following library declaration if instantiating
40  ---- any Xilinx primitives in this code.
41  library UNISIM;
42  use UNISIM.VComponents.all;
43
44  entity rocketio_glue is
45    port (
46      usrclk_ip : in std_logic;          -- varies
47      usrclk_in : in std_logic;          -- varies
48
49      system_clk : in std_logic;         -- 100 MHz
50      reset      : in std_logic;
51
52      chirp_pattern : in std_logic_vector( 63 downto 0 );
53      delay_time : in unsigned( 31 downto 0 );
54
55      RX1N_IN  : in  std_logic_vector(0 to 1);
56      RX1P_IN  : in  std_logic_vector(0 to 1);
57      TX1N_OUT : out std_logic_vector(0 to 1);
58      TX1P_OUT : out std_logic_vector(0 to 1);
59
60      led          : out std_logic_vector(1 to 10);
61      usrclk_out_p : out std_logic;
62      usrclk_out_n : out std_logic
63      );
64  end rocketio_glue;
65
66  ----------------------------------------------------
67  --  ARCHITECTURE
68  ----------------------------------------------------
69  architecture Behavioral of rocketio_glue is
70
71    -- ..............................
72    component ROCKETIO_WRAPPER
73      generic
74        (
75          MGT0_GT11_MODE_P : string  := "A";  -- Default Location
76          MGT0_MGT_ID_P    : integer := 0     -- 0=A, 1=B
77          );
78      port
79        (
80
81          ---------------------------------------------------------------------------
82          ---------------------------------------------------------------------------
83          --MGT0  (X0Y3)
84
85          ------------------------------- Global Ports ------------------------------
86          MGT0_POWERDOWN_IN  : in  std_logic;
```

```
87          MGT0_TXINHIBIT_IN   : in   std_logic;
88          ---------------------------------- PLL Lock ----------------------------------
89          MGT0_RXLOCK_OUT     : out std_logic;
90          MGT0_TXLOCK_OUT     : out std_logic;
91          --------------------------- Polarity Control Ports ---------------------------
92          MGT0_TXPOLARITY_IN  : in   std_logic;
93          ---------------------------- Ports for Simulation ----------------------------
94          MGT0_COMBUSIN_IN    : in   std_logic_vector(15 downto 0);
95          MGT0_COMBUSOUT_OUT  : out std_logic_vector(15 downto 0);
96          ------------------------------ Reference Clocks ------------------------------
97          MGT0_REFCLK1_IN     : in   std_logic;
98          ----------------------------------- Resets -----------------------------------
99          MGT0_RXPMARESET_IN  : in   std_logic;
100         MGT0_TXPMARESET_IN  : in   std_logic;
101         MGT0_TXRESET_IN     : in   std_logic;
102         -------------------------------- Serial Ports --------------------------------
103         MGT0_RX1N_IN        : in   std_logic;
104         MGT0_RX1P_IN        : in   std_logic;
105         MGT0_TX1N_OUT       : out std_logic;
106         MGT0_TX1P_OUT       : out std_logic;
107         ----------------------------------- Status -----------------------------------
108         MGT0_TXBUFERR_OUT   : out std_logic;
109         ---------------------- Transmit Data Path and Control Ports ----------------------
110         MGT0_TXDATA_IN      : in   std_logic_vector(63 downto 0);
111         -------------------------------- User Clocks ---------------------------------
112         MGT0_TXOUTCLK1_OUT  : out std_logic;
113         MGT0_TXOUTCLK2_OUT  : out std_logic;
114         MGT0_TXUSRCLK_IN    : in   std_logic;
115         MGT0_TXUSRCLK2_IN   : in   std_logic
116
117         );
118  end component;
119
120
121  --  ............................
122  component UNUSED_MGT
123    generic
124      (
125         GT11_MODE_P : string   := "B";    -- Default Location
126         MGT_ID_P    : integer := 1        -- Default Location
127         );
128    port
129      (
130
131         ------------------------------------------------------------------------------
132         ------------------------------------------------------------------------------
133         --MGT0
134
135         ---------------------------------- PLL Lock ----------------------------------
136         MGT0_RXLOCK_OUT     : out std_logic;
137         MGT0_TXLOCK_OUT     : out std_logic;
138         ---------------------------- Ports for Simulation ----------------------------
139         MGT0_COMBUSIN_IN    : in   std_logic_vector(15 downto 0);
140         MGT0_COMBUSOUT_OUT  : out std_logic_vector(15 downto 0);
```

```
141         ------------------------------ Reference Clocks ------------------------------
142         MGT0_REFCLK1_IN     : in  std_logic;
143         ------------------------------------- Resets -------------------------------------
144         MGT0_RXPMARESET_IN  : in  std_logic;
145         MGT0_TXPMARESET_IN  : in  std_logic;
146         MGT0_TXRESET_IN     : in  std_logic;
147         --------------------------------- Serial Ports ---------------------------------
148         MGT0_RX1N_IN        : in  std_logic;
149         MGT0_RX1P_IN        : in  std_logic;
150         MGT0_TX1N_OUT       : out std_logic;
151         MGT0_TX1P_OUT       : out std_logic;
152         --------------------------------- User Clocks ---------------------------------
153         MGT0_TXOUTCLK1_OUT  : out std_logic;
154         MGT0_TXOUTCLK2_OUT  : out std_logic;
155         MGT0_TXUSRCLK_IN    : in  std_logic;
156         MGT0_TXUSRCLK2_IN   : in  std_logic
157
158         );
159   end component;
160
161   -- ..............................
162   component MGT_USRCLK_SOURCE
163     port
164       (
165         USRCLK_SOURCE   : out std_logic;
166         USRCLK2_SOURCE  : out std_logic;
167         DCM_LOCKED      : out std_logic;
168         TXOUTCLK1_IN    : in  std_logic;
169         DCM_RESET       : in  std_logic
170
171         );
172   end component;
173
174   -- ..............................
175   component GT11_INIT_TX
176     generic
177       (
178         C_SIMULATION : integer := 0      -- Set to 1 for simulation
179         );
180     port
181       (
182         CLK            : in  std_logic;
183         START_INIT     : in  std_logic;
184         LOCK           : in  std_logic;
185         USRCLK_STABLE  : in  std_logic;
186         PCS_ERROR      : in  std_logic;
187         PMA_RESET      : out std_logic;
188         PCS_RESET      : out std_logic;
189         READY          : out std_logic
190
191         );
192   end component;
193
194   ----------------------------------------------------------------
```

```
195     -- Instantiations
196     ------------------------------------------------------------
197
198     signal usrclk_b   : std_logic;
199     signal usrclk_b2  : std_logic;
200     signal clk_fb     : std_logic;
201     signal clk_33MHz  : std_logic;
202     alias dcm_clk_b   : std_logic is clk_33MHz;
203
204     ------------------------ Reference Clock Signals ----------------------------
205     signal refclk_b            : std_logic;
206     signal dcm_locked          : std_logic;  -- TODO
207     signal dcm_reset_i         : std_logic;  -- TODO
208     ----------------------------------- PLL Lock --------------------------------
209     signal mgt0_txlock_i       : std_logic;
210     ----------------------------------- Resets ----------------------------------
211     signal mgt0_tx_pmareset_c  : std_logic;
212     signal mgt0_tx_reset_c     : std_logic;
213     ----------------------------------- Status ----------------------------------
214     signal mgt0_txbuferr_i     : std_logic;
215     -------------------- Transmit Data Path and Control Ports -------------------
216     signal mgt0_txdata_i       : std_logic_vector(63 downto 0);
217     -------------------------------- User Clocks --------------------------------
218     signal mgt0_txoutclk1_i    : std_logic;
219     signal usr_clk_i           : std_logic;
220     signal usr_clk2_i          : std_logic;
221
222
223     -------------------------------- Reset System -------------------------------
224     signal tx_system_ready_i : std_logic;  -- currently unused, for reset of user
225                                            -- logic
226
227     signal counter : unsigned( 31 downto 0 );
228
229
230   --*************************** Main Body of Code ******************************
231   begin
232
233     --  Static signal Assigments
234
235     led(1)  <= dcm_locked;
236     led(2)  <= dcm_reset_i;
237     led(3)  <= mgt0_tx_pmareset_c;
238     led(4)  <= mgt0_tx_reset_c;
239     led(5)  <= mgt0_txlock_i;
240     led(6)  <= mgt0_txbuferr_i;
241     led(7)  <= tx_system_ready_i;
242     led(8)   <= '1';
243     led(9)   <= '1';
244     led(10) <= '1';
245
246     usrclk_out_p <= usr_clk_i;
247     usrclk_out_n <= usr_clk2_i;
248
```

```
249
250    -- Processes
251
252    -- purpose: provides data to MGT0_TX
253    -- type    : sequential
254    -- inputs : usr_clk_i, reset
255    -- outputs:
256    txdata_p: process (usr_clk_i, tx_system_ready_i, chirp_pattern, delay_time)
257
258
259    begin  -- process txdata_p
260      if tx_system_ready_i = '0' then            -- asynchronous reset (active low)
261
262        counter        <= (others => '0');
263        mgt0_txdata_i <= (others => '0');
264
265      elsif usr_clk_i'event and usr_clk_i = '1' then  -- rising clock edge
266
267        if counter >= delay_time then
268
269          counter        <= (others => '0');
270          mgt0_txdata_i <= chirp_pattern;
271
272        else
273
274          counter <= counter + 1;
275          mgt0_txdata_i <= (others => '0');
276
277        end if;
278
279      end if;
280    end process txdata_p;
281
282
283    -- Components
284
285    ---------------user clocks (for refclk_b)-------------
286    IBUFGDS_usrclk : IBUFGDS
287      generic map (
288        IBUF_DELAY_VALUE => "0",
289        IOSTANDARD       => "LVPECL_25",
290        DIFF_TERM        => true)
291      port map (
292        O  => usrclk_b,
293        I  => usrclk_ip,                    -- Diff_p clock
294        IB => usrclk_in                     -- Diff_n clock
295        );
296
297    BUFG_userclk : BUFG
298      port map (
299        O => usrclk_b2,                     -- Clock buffer output
300        I => usrclk_b                       -- Clock buffer input
301        );
302    ---------------Instantiate PMCD/DCM module for the user clocks-------------
```

```
303
304     mgt_usrclk_source_i : MGT_USRCLK_SOURCE
305       port map
306       (
307         USRCLK_SOURCE   => usr_clk_i,
308         USRCLK2_SOURCE => usr_clk2_i,
309         DCM_LOCKED      => dcm_locked,
310         TXOUTCLK1_IN    => mgt0_txoutclk1_i,
311         DCM_RESET       => dcm_reset_i
312         );
313
314     dcm_reset_i    <= not mgt0_txlock_i;
315
316     ---------------------- Instantiate an GT11_INIT module --------------------
317     -- This module is responsible for performing the correct reset and lock procedure
318     -- as recommended in the user guide
319
320     gt11_init_tx_i : GT11_INIT_TX
321       generic map
322       (
323         C_SIMULATION => 0
324         )
325       port map
326       (
327         CLK            => dcm_clk_b,
328         START_INIT     => reset,
329         USRCLK_STABLE => dcm_locked,
330         LOCK           => mgt0_txlock_i,
331         PCS_ERROR      => mgt0_txbuferr_i,
332         PMA_RESET      => mgt0_tx_pmareset_c,
333         PCS_RESET      => mgt0_tx_reset_c,
334         READY          => tx_system_ready_i
335         );
336
337     ------------ GT11CLK_MGT Instantiations for upper reference clocks --------
338
339     upper_gt11clk_mgt_inst : GT11CLK
340       generic map
341       (
342         SYNCLK1OUTEN => "ENABLE",
343         SYNCLK2OUTEN => "DISABLE",
344         REFCLKSEL     => "REFCLK"
345         )
346       port map
347       (
348         MGTCLKN      => '0',                 -- unused
349         MGTCLKP      => '1',                 -- unused
350         REFCLK       => usrclk_b2,
351         RXBCLK       => '0',                 -- unused
352         SYNCLK1IN    => '0',                 -- unused
353         SYNCLK2IN    => '0',                 -- unused
354         SYNCLK1OUT => refclk_b,
355         SYNCLK2OUT => open
356         );
```

```
357
358 ---------------------- Instantiate MGT Wrapper ----------------------------
359
360 ----------------------------------------------------
361 -- MGT0    (X0Y3)
362
363 rocketio_wrapper_i : ROCKETIO_WRAPPER
364   generic map
365   (
366     MGT0_GT11_MODE_P => "B",
367     MGT0_MGT_ID_P    => 0
368     )
369   port map
370   (
371     -------------------------------- Global Ports ---------------------------
372     MGT0_POWERDOWN_IN    => '0',
373     MGT0_TXINHIBIT_IN    => '0',
374     ---------------------------------- PLL Lock ------------------------------
375     MGT0_RXLOCK_OUT      => open,
376     MGT0_TXLOCK_OUT      => mgt0_txlock_i,  -- unused?
377     -------------------------- Polarity Control Ports ------------------------
378     MGT0_TXPOLARITY_IN => '0',
379     --------------------------- Ports for Simulation -------------------------
380     MGT0_COMBUSIN_IN     => X"0000",
381     MGT0_COMBUSOUT_OUT => open,
382     ----------------------------- Reference Clocks ---------------------------
383     MGT0_REFCLK1_IN      => refclk_b,
384     ------------------------------------ Resets --------------------------------
385     MGT0_RXPMARESET_IN => '1',
386     MGT0_TXPMARESET_IN => mgt0_tx_pmareset_c,
387     MGT0_TXRESET_IN      => mgt0_tx_reset_c,
388     -------------------------------- Serial Ports ----------------------------
389     MGT0_RX1N_IN         => RX1N_IN(0),
390     MGT0_RX1P_IN         => RX1P_IN(0),
391     MGT0_TX1N_OUT        => TX1N_OUT(0),
392     MGT0_TX1P_OUT        => TX1P_OUT(0),
393     ------------------------------------ Status --------------------------------
394     MGT0_TXBUFERR_OUT   => mgt0_txbuferr_i,
395     ------------------- Transmit Data Path and Control Ports ------------------
396     MGT0_TXDATA_IN       => mgt0_txdata_i,
397     -------------------------------- User Clocks -----------------------------
398     MGT0_TXOUTCLK1_OUT => mgt0_txoutclk1_i,
399     MGT0_TXOUTCLK2_OUT => open,
400     MGT0_TXUSRCLK_IN     => usr_clk_i,
401     MGT0_TXUSRCLK2_IN    => usr_clk2_i
402
403     );
404
405
406
407 ----------------------------------------------------
408 -- Unused MGT paired with MGT0
409
410 unused_mgt_0_i : UNUSED_MGT
```

```
411      generic map
412      (
413        GT11_MODE_P => "B",                  -- Based on MGT Location
414        MGT_ID_P     => 1
415        )
416      port map(
417
418        ----------------------------------- PLL Lock -----------------------------------
419        MGT0_RXLOCK_OUT     => open,
420        MGT0_TXLOCK_OUT     => open,
421        ----------------------------- Ports for Simulation -----------------------------
422        MGT0_COMBUSIN_IN     => X"0000",
423        MGT0_COMBUSOUT_OUT => open,
424        ------------------------------- Reference Clocks -------------------------------
425        MGT0_REFCLK1_IN     => refclk_b,
426        ------------------------------------ Resets ------------------------------------
427        MGT0_RXPMARESET_IN => '1',
428        MGT0_TXPMARESET_IN => mgt0_tx_pmareset_c,
429        MGT0_TXRESET_IN     => mgt0_tx_reset_c,
430        --------------------------------- Serial Ports ---------------------------------
431        MGT0_RX1N_IN         => RX1N_IN(1),
432        MGT0_RX1P_IN         => RX1P_IN(1),
433        MGT0_TX1N_OUT        => TX1N_OUT(1),
434        MGT0_TX1P_OUT        => TX1P_OUT(1),
435        --------------------------------- User Clocks ----------------------------------
436        MGT0_TXOUTCLK1_OUT => open,
437        MGT0_TXOUTCLK2_OUT => open,
438        MGT0_TXUSRCLK_IN    => usr_clk_i,
439        MGT0_TXUSRCLK2_IN   => usr_clk2_i
440        );
441
442
443    ----------------------------------------------------------------------
444    -- divides down system clock by 3.0 to create
445    -- 100MHz / 3 = 33 MHz clock
446
447    DCM_BASE_inst : DCM_BASE
448      generic map (
449        CLKDV_DIVIDE            => 3.0,      -- Divide by: 3.0
450        CLKFX_DIVIDE            => 1,        -- Can be any interger from 1 to 32
451        CLKFX_MULTIPLY          => 4,        -- Can be any integer from 2 to 32
452        CLKIN_DIVIDE_BY_2       => false,    -- TRUE/FALSE to enable CLKIN divide by
453                                             -- two feature
454        CLKIN_PERIOD            => 10.0,     -- Specify period of input clock in ns
455                                             -- from 1.25 to 1000.00
456        CLKOUT_PHASE_SHIFT      => "NONE",   -- Specify phase shift mode of NONE or FIXED
457        CLK_FEEDBACK            => "1X",     -- Specify clock feedback of NONE or 1X
458        DCM_AUTOCALIBRATION     => true,     -- DCM calibrartion circuitry TRUE/FALSE
459        DCM_PERFORMANCE_MODE    => "MAX_SPEED",  -- Can be MAX_SPEED or MAX_RANGE
460        DESKEW_ADJUST           => "SYSTEM_SYNCHRONOUS",  -- SOURCE_SYNCHRONOUS,
461                                             -- SYSTEM_SYNCHRONOUS or
462                                             --   an integer from 0 to 15
463        DFS_FREQUENCY_MODE      => "LOW",    -- LOW or HIGH frequency mode for frequency
464        synthesis
```

```
465          DLL_FREQUENCY_MODE     => "LOW" ,    -- LOW, HIGH, or HIGH_SER frequency mode for
466          DLL
467          DUTY_CYCLE_CORRECTION => true ,      -- Duty cycle correction , TRUE or FALSE
468          FACTORY_JF             => X"F0F0" , -- FACTORY JF Values Suggested to be set to
469          X"F0F0
470          PHASE_SHIFT           => 0,        -- Amount of fixed phase shift from -255 to
471          1023
472          STARTUP_WAIT          => false)    -- Delay configuration DONE until DCM_LOCK,
473                                             -- TRUE/FALSE
474      port map (
475          CLK0     => clk_fb ,               -- 0 degree DCM CLK ouptput
476          CLK180   => open,                  -- 180 degree DCM CLK output
477          CLK270   => open,                  -- 270 degree DCM CLK output
478          CLK2X    => open,                  -- 2X DCM CLK output
479          CLK2X180 => open,                  -- 2X, 180 degree DCM CLK out
480          CLK90    => open,                  -- 90 degree DCM CLK output
481          CLKDV    => clk_33MHz,             -- Divided DCM CLK out (CLKDV_DIVIDE)
482          CLKFX    => open,                  -- DCM CLK synthesis out (M/D)
483          CLKFX180 => open,                  -- 180 degree CLK synthesis out
484          LOCKED   => open,                  -- DCM LOCK status output
485          CLKFB    => clk_fb ,               -- DCM clock feedback
486          CLKIN    => system_clk ,           -- Clock input (from IBUFG, BUFG or DCM)
487          RST      => reset                  -- DCM asynchronous reset input, active high
488          );
489
490  end Behavioral;
```

VHDL : chirp3_pads.vhd

```
1   ------------------------------------------------------------------------------
2   -- Copyright 2007 Massachusetts Institute of Technology : All rights reserved
3   --
4   -- Company:    Massachusetts Institute of Technology
5   -- Engineer:   Jason M. LaPenta
6   --
7   -- Create Date:      12:34:27 08/12/2006
8   -- Design Name:      uTags - chirp1
9   -- Module Name:      chirp3 - Behavioral
10  -- Project Name:     uTags
11  -- Target Devices: Virtex4 ML405 devel board
12  -- Tool versions:
13  -- Description: A simple chirp demonstration the sends out a configurable
14  --              make an ace file with
15  --              # ./ml40x_bit2ace chirp3_pads.bit chirp3.ace
16  --
17  -- Dependencies:
18  --
19  -- Revision:
20  -- Revision 0.01 - File Created
21  -- Additional Comments:
22  --
23  ------------------------------------------------------------------------------
24  library IEEE;
25  use IEEE.STD_LOGIC_1164.all;
26  use IEEE.STD_LOGIC_ARITH.all;
27  use IEEE.STD_LOGIC_UNSIGNED.all;
28
29  ---- Xilinx primitives
30  library UNISIM;
31  use UNISIM.VComponents.all;
32
33  -----------------------------------------------------------------------------
34  -- Entity chirp3
35  -----------------------------------------------------------------------------
36  entity chirp3_pads is
37    port (
38      sysclk_i  : in std_logic;          -- 100 mhz system clock
39      usrclk_ip : in std_logic;          -- user supplied clock for chirp freq
40      usrclk_in : in std_logic;          -- user supplied clock for chirp freq
41      reset_n   : in std_logic;          -- active low reset button A10
42      trig_i    : in std_logic;          -- trigger, starts chirp
43
44      pb_n_i : in std_logic;             -- push button north
45      pb_s_i : in std_logic;             -- push button south
46      pb_c_i : in std_logic;             -- push button center
47
48      led_reset   : out std_logic;       -- illuminates when device reset
49      led_trig    : out std_logic;       -- illuminates on trigger
50      led_timeout : out std_logic;       -- illuminates on trigger timeout
51      test_o      : out std_logic;       -- test output H6 diff_clk_out_n
52
```

```
53     chirp_op      : out std_logic;   -- chirp output
54     chirp_on      : out std_logic;   -- chirp output
55
56     chirp_o : out std_logic             -- chirp output signal
57     );
58
59 end chirp3_pads;
60
61 -----------------------------------------------------------------------------
62 --  architecture
63 -----------------------------------------------------------------------------
64 architecture Behavioral of chirp3_pads is
65
66   -------------------------------------------------
67   component chirp3_top
68     port(
69       sysclk_i : in std_logic;          -- 100 mhz system clock
70       usrclk_i : in std_logic;          -- user supplied clock for chirp freq
71       reset_n  : in std_logic;
72       trig_i   : in std_logic;          -- trigger, starts chirp
73
74       pb_n_i : in std_logic;               -- push button north
75       pb_s_i : in std_logic;               -- push button south
76       pb_c_i : in std_logic;               -- push button center
77
78       led_reset   : out std_logic;      -- illuminates when device reset
79       led_trig    : out std_logic;      -- illuminates on trigger
80       led_timeout : out std_logic;      -- illuminates on trigger timeout
81       test_o      : out std_logic;        -- test output H6 diff_clk_out_n
82
83       chirp_o     : out std_logic;   -- chirp output
84
85       chirp_en_o : out std_logic
86       );
87   end component;
88   -------------------------------------------------
89
90
91   signal sysclk_b  : std_logic;         -- 100 mhz system clock
92   signal usrclk_b  : std_logic;         -- user supplied clock for chirp freq
93   signal reset_n_b : std_logic;
94   signal trig_b    : std_logic;         -- trigger, starts chirp
95
96   signal pb_n_b : std_logic;            -- push button north
97   signal pb_s_b : std_logic;            -- push button south
98   signal pb_c_b : std_logic;            -- push button center
99
100  signal led_reset_b   : std_logic;     -- illuminates when device reset
101  signal led_trig_b    : std_logic;     -- illuminates on trigger
102  signal led_timeout_b : std_logic;     -- illuminates on trigger timeout
103  signal test_b        : std_logic;     -- test output H6 diff_clk_out_n
104
105  signal chirp_b      : std_logic;   -- chirp output
106
```

```
107    signal chirp_en_b : std_logic;
108
109  begin
110
111    ------------------------------------------------------------
112    -- top component
113    top_c : chirp3_top port map(
114      sysclk_i     => sysclk_b,
115      usrclk_i     => usrclk_b,
116      reset_n      => reset_n_b,
117      trig_i       => trig_b,
118      pb_n_i       => pb_n_b,
119      pb_s_i       => pb_s_b,
120      pb_c_i       => pb_c_b,
121      led_reset    => led_reset_b,
122      led_trig     => led_trig_b,
123      led_timeout => led_timeout_b,
124      test_o       => test_b,
125      chirp_o      => chirp_b,
126      chirp_en_o   => chirp_en_b
127      );
128
129
130    -- ************************************************
131    IBUFG_sysclk : IBUFG
132      generic map (
133        IBUF_DELAY_VALUE => "0",
134        IOSTANDARD       => "LVTTL")
135      port map (
136        O => sysclk_b,
137        I => sysclk_i
138        );
139
140    -- ************************************************
141    IBUFGDS_usrclk : IBUFGDS
142      generic map (
143        IBUF_DELAY_VALUE => "0",
144        IOSTANDARD       => "LVPECL_25",
145        DIFF_TERM        => TRUE)
146      port map (
147        O  => usrclk_b,
148        I  => usrclk_ip,  -- Diff_p clock
149        IB => usrclk_in   -- Diff_n clock
150        );
151
152    -- ************************************************
153    OBUF_chirp_o : OBUFDS
154      generic map (
155        IOSTANDARD => "LVPECL_25"
156        )
157      port map (
158        O  => chirp_op,
159        OB => chirp_on,
160        I  => chirp_b
```

```
161         );
162
163
164   -- ***********************************************
165   IBUF_reset : IBUF
166     generic map (
167       IBUF_DELAY_VALUE => "0",
168       IFD_DELAY_VALUE  => "AUTO",
169       IOSTANDARD       => "LVTTL")
170     port map (
171       O => reset_n_b,
172       I => reset_n
173       );
174
175   -- ***********************************************
176   IBUF_trig : IBUF
177     generic map (
178       IBUF_DELAY_VALUE => "0",
179       IFD_DELAY_VALUE  => "AUTO",
180       IOSTANDARD       => "LVTTL")
181     port map (
182       O => trig_b,
183       I => trig_i
184       );
185
186   -- ***********************************************
187   IBUF_pb_n : IBUF
188     generic map (
189       IBUF_DELAY_VALUE => "0",
190       IFD_DELAY_VALUE  => "AUTO",
191       IOSTANDARD       => "LVTTL")
192     port map (
193       O => pb_n_b,
194       I => pb_n_i
195       );
196
197   -- ***********************************************
198   IBUF_pb_s : IBUF
199     generic map (
200       IBUF_DELAY_VALUE => "0",
201       IFD_DELAY_VALUE  => "AUTO",
202       IOSTANDARD       => "LVTTL")
203     port map (
204       O => pb_s_b,
205       I => pb_s_i
206       );
207
208   -- ***********************************************
209   IBUF_pb_c : IBUF
210     generic map (
211       IBUF_DELAY_VALUE => "0",
212       IFD_DELAY_VALUE  => "AUTO",
213       IOSTANDARD       => "LVTTL")
214     port map (
```

```
215         O => pb_c_b ,
216         I => pb_c_i
217         );
218
219     -- *************************************************
220     OBUF_led_reset : OBUF
221       generic map (
222         DRIVE       => 12,
223         IOSTANDARD => "LVTTL",
224         SLEW        => "SLOW")
225       port map (
226         O => led_reset ,
227         I => led_reset_b
228         );
229
230     -- *************************************************
231     OBUF_led_trig : OBUF
232       generic map (
233         DRIVE       => 12,
234         IOSTANDARD => "LVTTL",
235         SLEW        => "SLOW")
236       port map (
237         O => led_trig ,
238         I => led_trig_b
239         );
240
241     -- *************************************************
242     OBUF_led_timeout : OBUF
243       generic map (
244         DRIVE       => 12,
245         IOSTANDARD => "LVTTL",
246         SLEW        => "SLOW")
247       port map (
248         O => led_timeout ,
249         I => led_timeout_b
250         );
251
252     -- *************************************************
253     OBUF_test : OBUF
254       generic map (
255         DRIVE       => 12,
256         IOSTANDARD => "LVTTL",
257         SLEW        => "SLOW")
258       port map (
259         O => test_o ,
260         I => test_b
261         );
262
263     -- *************************************************
264     OBUF_chirp : OBUFT
265       generic map (
266         IOSTANDARD => "LVTTL",
267         DRIVE       => 24,
268         SLEW        => "FAST")
```

```
269      port map (
270        T => chirp_en_b,
271        O => chirp_o,
272        I => usrclk_b
273        );
274
275
276  end Behavioral;
```

VHDL : chirp3_top.vhd

```vhdl
------------------------------------------------------------------------------
-- Copyright 2007 Massachusetts Institute of Technology : All rights reserved
--
-- Company:    Massachusetts Institute of Technology
-- Engineer:   Jason M. LaPenta
--
-- Create Date:     12:34:27 08/12/2006
-- Design Name:     uTags - chirp1
-- Module Name:     chirp3 - Behavioral
-- Project Name:    uTags
-- Target Devices: Virtex4 ML405 devel board
-- Tool versions:
-- Description:
--
--          Sends a chirp on a trigger signal, or at 10Hz in the absence of
--          a trigger.
--
--          The number of cycles per chirp can be changed using the push buttons
--
--          This version utilized OSERDES to generate phase shifted wide-band chirps.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01       - File Created
-- Additional Comments:
--
------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.numeric_std.all;

---- Xilinx primitives
library UNISIM;
use UNISIM.VComponents.all;

-----------------------------------------------------------------------------
-- Entity chirp3
-----------------------------------------------------------------------------
entity chirp3_top is
  port (sysclk_i : in std_logic;        -- 100 MHz system clock
        usrclk_i : in std_logic;        -- user supplied clock for chirp freq
        reset_n  : in std_logic;
        trig_i   : in std_logic;        -- trigger, starts chirp

        pb_n_i : in std_logic;          -- push button north
        pb_s_i : in std_logic;          -- push button south
        pb_c_i : in std_logic;          -- push button center

        led_reset   : out std_logic;    -- Illuminates when device reset
```

```
53          led_trig     : out std_logic;     -- illuminates on trigger
54          led_timeout : out std_logic;     -- illuminates on trigger timeout
55          test_o       : out std_logic;     -- test output H6 diff_clk_out_n
56
57          chirp_o       : out std_logic;   -- chirp output
58
59          chirp_en_o : out std_logic);
60  end chirp3_top;
61
62  ------------------------------------------------------------------------------
63  --  architecture
64  ------------------------------------------------------------------------------
65  architecture Behavioral of chirp3_top is
66
67    ----------------------------------------------------
68    component chirp3_gen
69      port(
70        clock_i   : in std_logic;           -- variable input clock
71        reset_b   : in std_logic;
72        le_i      : in std_logic;           -- data latch enable
73        data_i    : in std_logic_vector( 5 downto 0 );           -- data latch enable
74
75        chirp_o   : out std_logic           -- chirp signal output
76        );
77    end component;
78    ----------------------------------------------------
79
80    signal clk_5MHz : std_logic;           -- internal divided clock
81    signal clk_fb   : std_logic;           -- DCM clock Feed Back
82    signal reset_b  : std_logic;           -- Active High Reset for DCM
83
84    signal trig_timeout : std_logic;      -- signals tigger timeout
85    signal trigger      : std_logic;  -- signals to trigger a chirp on rising edge
86
87    signal chirp_en  : std_logic;
88    signal chirp_len : unsigned(7 downto 0);  -- chirp length in cycles
89
90  begin
91
92
93    -- combinational
94    led_timeout <= trig_timeout;
95    led_trig    <= trigger;
96    led_reset   <= not reset_n;
97    test_o      <= clk_5MHz;
98    reset_b     <= not reset_n;
99    chirp_en_o  <= chirp_en;              -- active low
100
101   ----------------------------------------------------
102   --
103   chirp3_gen1 : chirp3_gen
104     port map (
105       clock_i  => usrclk_i,
106       reset_b  => reset_b,
```

```
107         le_i       => '1',
108         data_i    => "111111",
109         chirp_o   => chirp_o
110         );
111   --------------------------------------------------------
112
113
114   --------------------------------------------------------
115   --  uses trigger to gate chirp output
116   chirp_p : process (usrclk_i, reset_n)
117
118     variable chirp_cntr : unsigned(7 downto 0);
119     variable trig_last   : std_logic;
120
121
122   begin  -- process chirp_p
123
124     if reset_n = '0' then
125       chirp_cntr := (others => '0');
126       chirp_en    <= '1';
127
128     elsif usrclk_i'event and usrclk_i = '1' then  -- rising clock edge
129
130       if trigger = '1' and trig_last = '0' then
131         chirp_en    <= '0';
132         chirp_cntr := (others => '0');
133       else
134
135         if chirp_cntr /= chirp_len then
136           chirp_cntr := chirp_cntr + 1;
137           chirp_en    <= '0';
138         else
139           chirp_en <= '1';
140         end if;
141
142       end if;
143       trig_last := trigger;
144
145     end if;
146
147   end process chirp_p;
148
149
150   --------------------------------------------------------
151   -- purpose: create internal triger if necessary
152   trig_p : process (clk_5MHz, reset_n, trig_i)
153
154     -- one second timeout
155     variable trig_to_cntr : unsigned (23 downto 0);
156     variable trig_last     : std_logic;
157
158     -- trig_cntr used for 10kHz timout
159     variable trig_cntr      : unsigned (7 downto 0);
160     variable trig_internal : std_logic;
```

```
161
162   begin  -- process trig_p
163
164     if reset_n = '0' then
165       trig_last    := '0';
166       trig_to_cntr := (others => '0');
167
168       trig_cntr     := (others => '0');
169       trig_internal := '0';
170
171       trig_timeout <= '0';
172       trigger      <= '0';
173
174     elsif clk_5MHz'event and clk_5MHz = '1' then  -- rising clock edge
175
176       -- edge detect TRIG, reset timeout if detected
177       if trig_i = '1' and trig_last = '0' then
178         trig_to_cntr := (others => '0');
179       else
180         if trig_timeout = '0' then
181           trig_to_cntr := trig_to_cntr + 1;
182         end if;
183       end if;
184       trig_last := trig_i;
185
186       -- 1 second signal timeout to start 10kHz signal
187       if trig_to_cntr = 2000 then       -- 5e6 then
188         trig_timeout <= '1';
189       else
190         trig_timeout <= '0';
191       end if;
192
193       -- internal trigger counter
194       if trig_cntr = 250 then
195
196         trig_cntr     := (others => '0');
197         trig_internal := not trig_internal;
198       else
199         trig_cntr     := trig_cntr + 1;
200         trig_internal := trig_internal;
201       end if;
202
203       -- trigger select
204       if trig_timeout = '0' then
205         trigger <= trig_i;
206       else
207         trigger <= trig_internal;
208       end if;
209
210     end if;
211
212   end process trig_p;
213
214
```

```
215   --------------------------------------------------------
216   -- purpose: debounce buttons
217   buttons_p : process (clk_5MHz, reset_n)
218
219     -- one second timeout, disable buttons for
220     -- one second after a press
221     variable buttons_to_cntr : unsigned(21 downto 0);
222
223   begin  -- process buttons_p
224
225     if reset_n = '0' then
226
227       buttons_to_cntr := (others => '0');
228       chirp_len       <= unsigned(to_unsigned(50, chirp_len'length));
229
230     elsif clk_5MHz'event and clk_5MHz = '1' then  -- rising clock edge
231
232       if buttons_to_cntr = 0 then
233
234         -- increase chirp length by 5
235         if pb_n_i = '1' and chirp_len < 250 then
236           chirp_len       <= chirp_len + 5;
237           buttons_to_cntr := (others => '1');
238
239           -- decrease chirp length by 5
240         elsif pb_s_i = '1' and chirp_len > 10 then
241           chirp_len       <= chirp_len - 5;
242           buttons_to_cntr := (others => '1');
243
244           -- decrease chirp length by 1
245         elsif pb_s_i = '1' and chirp_len > 1 and chirp_len < 10 then
246           chirp_len       <= chirp_len - 1;
247           buttons_to_cntr := (others => '1');
248
249           -- reset chirp length to 50
250         elsif pb_c_i = '1' then
251           chirp_len       <= unsigned(to_unsigned(50, chirp_len'length));
252           buttons_to_cntr := (others => '1');
253
254         end if;
255
256       else
257         buttons_to_cntr := buttons_to_cntr - 1;
258       end if;
259
260     end if;
261
262   end process buttons_p;
263
264
265   --------------------------------------------------------
266   -- divides down system clock by 32 to create
267   -- 100MHz / 20 = 5 MHz clock
268   DCM_BASE_inst : DCM_BASE
```

```
269       generic map (
270         CLKDV_DIVIDE            => 10.0,  -- Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
271         --    7.0,7.5,8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0 or 16.0
272         CLKFX_DIVIDE            => 1,        -- Can be any interger from 1 to 32
273         CLKFX_MULTIPLY          => 4,        -- Can be any integer from 2 to 32
274         CLKIN_DIVIDE_BY_2       => true,  -- TRUE/FALSE to enable CLKIN divide by two feature
275         CLKIN_PERIOD            => 10.0,  -- Specify period of input clock in ns from 1.25 to 1000.00
276         CLKOUT_PHASE_SHIFT      => "NONE",  -- Specify phase shift mode of NONE or FIXED
277         CLK_FEEDBACK            => "1X",     -- Specify clock feedback of NONE or 1X
278         DCM_AUTOCALIBRATION     => true,  -- DCM calibrartion circuitry TRUE/FALSE
279         DCM_PERFORMANCE_MODE    => "MAX_SPEED",  -- Can be MAX_SPEED or MAX_RANGE
280         DESKEW_ADJUST           => "SYSTEM_SYNCHRONOUS",  -- SOURCE_SYNCHRONOUS, SYSTEM_SYNCHRONOUS or
281                                                           --   an integer from 0 to 15
282         DFS_FREQUENCY_MODE      => "LOW",  -- LOW or HIGH frequency mode for frequency synthesis
283         DLL_FREQUENCY_MODE      => "LOW",  -- LOW, HIGH, or HIGH_SER frequency mode for DLL
284         DUTY_CYCLE_CORRECTION => true,     -- Duty cycle correction, TRUE or FALSE
285         FACTORY_JF              => X"F0F0",  -- FACTORY JF Values Suggested to be set to X"F0F0
286         PHASE_SHIFT             => 0,  -- Amount of fixed phase shift from -255 to 1023
287         STARTUP_WAIT            => false)  -- Delay configuration DONE until DCM LOCK, TRUE/FALSE
288       port map (
289         CLK0      => clk_fb,              -- 0 degree DCM CLK ouptput
290         CLK180    => open,                -- 180 degree DCM CLK output
291         CLK270    => open,                -- 270 degree DCM CLK output
292         CLK2X     => open,                -- 2X DCM CLK output
293         CLK2X180 => open,                 -- 2X, 180 degree DCM CLK out
294         CLK90     => open,                -- 90 degree DCM CLK output
295         CLKDV     => clk_5MHz,            -- Divided DCM CLK out (CLKDV_DIVIDE)
296         CLKFX     => open,                -- DCM CLK synthesis out (M/D)
297         CLKFX180 => open,                 -- 180 degree CLK synthesis out
298         LOCKED    => open,                -- DCM LOCK status output
299         CLKFB     => clk_fb,              -- DCM clock feedback
300         CLKIN     => SYSCLK_I,          -- Clock input (from IBUFG, BUFG or DCM)
301         RST       => reset_b            -- DCM asynchronous reset input, active high
302         );
303
304   end Behavioral;
```

C : chirpnet.c

```
1  /*-------------------------------------------------------------------------------
2     -- Copyright 2007 Massachusetts Institute of Technology : All rights reserved
3     --
4     -- Create Date:     3/13/2007
5     -- Design Name:     uTags - xps/chirp_rio/chirp2
6     -- File Name:       chirp2.c
7     -- Target Devices: Virtex4 ML405 devel board PowerPC 405
8     -- Tool versions:  Xilinx XPS 8.2
9     --
10    -- Description: Simple program to access UART debug interface,
11    --              Ethernet User Interface, and the rocket_io chirp2
12    --              Peripherial.
13    -------------------------------------------------------------------------------*/
14
15 #include <stdio.h>
16 #include <stdlib.h>
17 #include <string.h>
18 #include <xgpio_l.h>
19 #include <xbasic_types.h>
20
21 #include "xparameters.h"
22 #include "xromlcd.h"
23 #include "chirp2.h"
24 #include "enet.h"
25
26 #define VERSION "ChirpNet_v0.10"
27
28 //=====================================================
29 int main (void) {
30
31   /* Setup LCD screen */
32   XromLCDOn();
33   XromLCDInit();
34   XromLCDPrintString ( VERSION, "Running..." );
35   xil_printf( VERSION "\n");
36
37   // initialize chirp2 peripheral
38   chirp2_reset();
39   chirp2_set_delay( 100 );
40   chirp2_set_pattern( 0x88AA55AA, 0xFF00FFFF );
41   chirp2_enable();
42
43   // open Ethernet and Accept New Connections
44   enet_open();
45
46   for (;;) {
47     enet_accept();
48     enet_proc_conns();
49   }
50   return 0;
51 }
52
```

```
53  /* Stack Overflow & Interrupt Handler Dummy Functions */
54  void _stack_overflow_exit(void) {
55    xil_printf("Stack Overflow \r\n");
56  }
57
58  void _interrupt_handler() {}
```

C : enet.c

```
1  /*---------------------------------------------------------------------------------
2    -- Copyright 2007 Massachusetts Institute of Technology : All rights reserved
3    --
4    -- Create Date:     3/13/2007
5    -- Design Name:     uTags - xps/chirp_rio/chirp2
6    -- Target Devices: Virtex4 ML405 devel board PowerPC 405
7    -- Tool versions:   Xilinx XPS 8.2
8    --
9    -- Description: Supports the Server TCP/IP SocketIO Interface
10   ---------------------------------------------------------------------------------*/
11
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <string.h>
15 #include <net/xilnet_config.h>
16 #include <net/xilsock.h>
17 #include <net/mac.h>
18 #include <xemac_l.h>
19 #include <xbasic_types.h>
20
21 #include "xromlcd.h"
22 #include "enet.h"
23 #include "chirp2.h"
24
25 // Server States for connections
26 #define NET_GET          1    // Got a Get request
27 #define NET_GET_PROC     2    // processing a GET request
28 #define NET_DONE         3    // done with a GET request processing
29 #define NET_CONN_FREE   -1    // conn free to assign to client
30
31 struct net_conn_ent {
32   int sock;           // socket descriptor
33   int state;        // state of the connection
34 };
35
36 /****************************************
37  * Global Variables */
38 struct globals {
39   int isnetinit; // Net Connections Management
40   int sock;
41   struct net_conn_ent net_conns[MAX_NET_CONNS];
42   struct sockaddr_in addr;
43 } g;
44
45 /****************************************
46  * Local Function Definitions */
47 void enet_parse_buf ( char *buf );
48 void enet_free_conn ( int idx );
49 int  enet_add_conn( int sock );
50 void enet_proc_req ( int cidx );
51
52 /****************************************
```

```
53    * open */
54   int enet_open ( void ) {
55
56     int err;
57     int idx;
58
59     g.isnetinit = 0;
60     g.sock = 0;
61
62     // Initialise all net conns
63     for (idx = 0; idx < MAX_NET_CONNS; idx++) {
64       g.net_conns[idx].sock =-1;
65       g.net_conns[idx].state = NET_CONN_FREE;
66     }
67     g.isnetinit = 1;
68
69     // Set up MAC & IP addresses and intialise Ethernet HW Table
70     xilnet_eth_init_hw_addr( "00:00:00:00:22:38" );
71     xilnet_eth_init_hw_addr_tbl();
72     xilnet_ip_init( ENET_ADDR );
73
74     // Initialize MAC Baseaddress, set MAC address and enable it
75     xilnet_mac_init( XPAR_MYETHER_BASEADDR );
76     XEmac_mSetMacAddress( XPAR_MYETHER_BASEADDR, mb_hw_addr );
77     XEmac_mEnable( XPAR_MYETHER_BASEADDR );
78
79     // Create, Bind, Listen on Server Socket
80     g.sock = xilsock_socket( AF_INET, SOCK_STREAM, 0 );
81     if (g.sock == -1) {
82       xil_printf("socket() error \r\n");
83       return 0;
84     }
85
86     g.addr.sin_family = AF_INET;
87     g.addr.sin_addr.s_addr = INADDR_ANY;
88     g.addr.sin_port = SERVER_PORT;
89
90     err = xilsock_bind( g.sock, (struct sockaddr *)&(g.addr), sizeof(struct sockaddr) );
91     if (err == -1) {
92       xil_printf("bind() error \r\n");
93       return 0;
94     }
95
96     err = xilsock_listen( g.sock, 4 );
97     if (err == -1) {
98       xil_printf("listen() error \r\n");
99       return 0;
100    }
101
102    return 1;
103  }
104
105  /****************************************
106   * close */
```

```
107 int enet_close ( void ) {
108   xil_printf("warning : enet_close() not implemented  \r\n");
109   return 1;
110 }
111
112 /****************************************
113  * accept - Accept new connections */
114 int enet_accept ( void ) {
115
116   int alen = 0;
117   int rtn  = 0;
118
119   alen = sizeof(struct sockaddr);
120   rtn  = xilsock_accept( g.sock, (struct sockaddr *)&(g.addr), &alen );
121
122   if ( xilsock_status_flag & XILSOCK_NEW_CONN ) {
123     xil_printf( "New Connection \r\n" );
124     enet_add_conn( rtn );
125   }
126
127   return 1;
128 }
129
130 /****************************************
131  * Add a net connection */
132 int enet_add_conn( int sock ) {
133
134   int idx;
135
136   for ( idx = 0; idx < MAX_NET_CONNS; idx++ ) {
137     if ( g.net_conns[idx].state == NET_CONN_FREE )
138       break;
139   }
140   if ( idx <= MAX_NET_CONNS) {
141     g.net_conns[idx].sock = sock;
142     g.net_conns[idx].state = NET_GET;
143     return idx;
144   }
145   xil_printf("no free conns \r\n");
146   return -1;
147 }
148
149 /****************************************
150  * Process a Net Connection */
151 void enet_proc_req ( int cidx ) {
152
153   unsigned char *sndptr = (unsigned char*) \
154                           (sendbuf+LINK_HDR_LEN+IP_HDR_LEN*4+(TCP_HDR_LEN*4));
155   unsigned char *buf    = xilsock_sockets[ g.net_conns[cidx].sock ].recvbuf.buf;
156
157   if ( !buf )
158     return;
159
160   if ( xilsock_status_flag & XILSOCK_TCP_DATA ) {
```

```
161       XromLCDPrintString( strtok(buf,"\n"),"" );
162       enet_parse_buf( buf );
163       memset( sendbuf, 0, LINK_FRAME_LEN );
164       memcpy( sndptr, buf, strlen(buf) );
165       xilsock_send( g.net_conns[cidx].sock, sendbuf, strlen(sndptr)+1 );
166     }
167
168     // reset buf ptrs of sockets
169     if ( g.net_conns[cidx].sock != -1 ) {
170       xilsock_sockets[ g.net_conns[cidx].sock ].recvbuf.buf = NULL;
171       xilsock_sockets[ g.net_conns[cidx].sock ].recvbuf.size = 0;
172     }
173   }
174
175   /****************************************
176    * Free a net connection */
177   void enet_free_conn ( int idx ) {
178
179     xil_printf("Free Connection %d \r\n", idx);
180     xilsock_close( g.net_conns[ idx ].sock );
181     g.net_conns[idx].sock = -1;
182     g.net_conns[idx].state = NET_CONN_FREE;
183   }
184
185   /****************************************
186    * Process all net connections */
187   void enet_proc_conns( void ) {
188
189     int idx;
190     for (idx = 0; idx < MAX_NET_CONNS; idx++){
191       enet_proc_req( idx );
192     }
193   }
194
195   /****************************************
196    * enet_parse_buf -
197    * parse, process, and then execute
198    * commands sent by client */
199   void enet_parse_buf ( char *buf ){
200
201     int   idx = 0;
202     char *tok_array[10];
203
204     // parse no more than 10 tokens
205     tok_array[0] = strtok( buf, ",");
206     xil_printf("Token 0 = '%s' \r\n", tok_array[0] );
207
208     for ( idx = 1; idx < 10; idx++ ){
209       tok_array[idx] = strtok( 0, ",");
210       if ( tok_array[idx] == 0 ){
211         break;
212       }
213       xil_printf("Token %d = %s \r\n", idx, tok_array[idx] );
214     }
```

```
215
216     //-------------------------------------------------
217     // parse commands - "chirp2,delay,high,low;"
218     if( !strcmp( tok_array[0], "chirp2" ) ){
219
220       Xuint32 delay = strtoul( tok_array[1], 0, 0 );
221       Xuint32 high  = strtoul( tok_array[2], 0, 0 );
222       Xuint32 low   = strtoul( tok_array[3], 0, 0 );
223
224       xil_printf( "chirp2'%d'0x%08X'0x%08X;\r\n", delay, high, low );
225
226       chirp2_set_delay( delay );
227       chirp2_set_pattern( high, low );
228       chirp2_enable();
229       return;
230     }
231   }
```

Appendix F

MATLAB Source Code

MATLAB : `pschirpWB.m` Source code for simulations in section 2.3

```
1  function chirp = pschirpWB ( encoding )
2  % 11/29/2006
3  %
4  % creates a phase shifted chirp
5  % the encoding is an array of finger
6  % polarties. 0 = even, 1 = odd phase
7  % 2 = no signal
8
9  f = 915e6;    % frequency = 915 MHz
10 w = 2*pi*f;   % rad/second
11 p = 1/f;      % period
12
13 len = size(encoding,2);
14
15 t = 0:(p/100):p;
16
17 even = sin(w*t);
18 odd  = -even;
19 chirp = [0];
20
21 % model input pulse
22 for i = 1:len
23   if encoding(i) == 0
24     chirp = [chirp odd];
25   elseif encoding(i) == 1
26     chirp = [chirp even];
27   else
28     chirp = [chirp zeros(size(even))];
29   end
30 end
```

MATLAB : `nb.m` Source code for simulations in section 2.3

```
1  % This code simulates narrow--band
2  % barker codes for SAW correlators.
3
4  %...............................................
5  % n-5 barker code simulation
6
7  % 11101 receive code, narrow band
8  ea = [ 2 1 0 1 1 1 2];
9  rx = pschirpWB( ea );
10
11 % 10111 transmit output code, narrow band
12 ea = [ 2 1 1 1 0 1 2];
13 tx = pschirpWB( ea );
14
15 % narrow band input idt
16 ec = [2 1 2];
17 inidt = pschirpWB( ec );
18
19 figure
20 t1 = conv( tx,conv(rx,inidt));
21 plot( t1 )
22
23 %...............................................
24 % example narrow--band device on mask3.v3
25
26 % 1111110111 receive code, narrow band
27 ea = [ 2 1 1 1 1 1 1 0 1 1 1 2];
28 rx = pschirpWB( ea );
29
30 % 1011111111 transmit output code, narrow band
31 ea = [ 2 1 1 1 0 1 1 1 1 1 1 2];
32 tx = pschirpWB( ea );
33
34 % narrow band input idt
35 ec = [2 1 2];
36 inidt = pschirpWB( ec );
37
38 figure
39 t2 = conv( tx,conv(rx,inidt));
40 plot( t2 )
```

MATLAB : `wb.m` Source code for simulations in section 2.3

```
1  % This code simulates wide--band and wide band
2  % barker codes for SAW correlators.
3
4  %...............................................
5  % n-5 barker code simulation with 2-cycle codes
6
7  l0 = zeros(1,2);
8  l1 = ones(1,2);
9  ss = 2*ones(1,2);
10
11 % 10111 code wide band
12 ea = [ 2 l1 ss l0 ss l1 ss l1 ss l1 2];
13 rx = pschirpWB( ea );
14
15 % 11101 code wide band
16 ea = [ 2 l1 ss l1 ss l1 ss l0 ss l1 2];
17 tx = pschirpWB( ea );
18
19 % narrow band input idt
20 ec = [2 1 1 2];
21 inidt = pschirpWB( ec );
22
23 figure
24 t1 = conv( tx,conv(rx,inidt));
25 plot( t1 )
26
27 %...............................................
28 % example wide--band device on mask3.v3
29
30 l0 = zeros(1,5);
31 l1 = ones(1,5);
32 ss = 2*ones(1,5);
33
34 % 11101 code wide band
35 ea = [ 2 l1 ss l0 ss l1 ss l1 ss l1 2];
36 rx = pschirpWB( ea );
37
38 % 11101 code wide band
39 ea = [ 2 l1 ss l1 ss l1 ss l0 ss l1 2];
40 tx = pschirpWB( ea );
41
42 % narrow band input idt
43 ec = [2 l1 2];
44 inidt = pschirpWB( ec );
45
46 figure
47 t2 = conv( tx,conv(rx,inidt));
48 plot( t2 )
```

MATLAB : `dual.m` Source code for simulations in section 2.3

```
1  % This code simulates the dual reflector
2  % narrow-band SAW correlator transponder
3
4  v  = 3992;   % surface velocity (m/s)
5  r1 = 280e-6; % distance to reflector 1
6  r2 = 810e-6; % distance to reflector 2
7  d1 = r1/v;   % delay (s) to reflector 1
8  d2 = r2/v;   % delay (s) to reflector 2
9
10 f = 323e6;   % frequency = 323 MHz
11 w = 2*pi*f;  % rad/second
12 p = 1/f;     % period
13 t = 0:(p/100):400e-9;  % 0 to 400ns
14
15 % + and - patterns, 5 in cycles in length
16 l0 = zeros(1,5);
17 l1 = ones(1,5);
18
19 % 11101 code narrow band
20 ea = [ 2 l1 l0 l1 l1 l1 2 ];
21 rx = pschirpWB( ea );
22
23 % 11101 code wide band
24 ea = [ 2 l1 l1 l1 l0 l1 2 ];
25 tx = pschirpWB( ea );
26
27 % narrow band input idt
28 ec = [2 l1 2];
29 inidt = pschirpWB( ec );
30
31 %------------------------------
32 % signal from inidt to tx
33 s = conv( tx,conv(rx,inidt));
34
35 % amount of time to add to first reflector delay
36 add = (d1 - p*3)*100/p;
37 s1 = [zeros(1,floor(add)), s];
38
39 % amount of time to add to second reflector delay
40 add = (d2 - p*3)*100/p;
41 s2 = [zeros(1,floor(add)), s];
42
43 s1 = [s1, zeros(1,(size(s2,2)-size(s1,2)))];
44 st = (s1 + s2);%/(2*max(s1));
45
46 % plot of one way signal path
47 figure
48 plot( t(1:size(st,2)), st )
49
50 %--------------------------
51 % signal form tx to inidt
52 s = conv( inidt,conv(rx,rx) );
```

```
53
54  % amount of time to add to first reflector delay
55  add = (d1 - p*3)*100/p;
56  s1 = [zeros(1,floor(add)), s];
57
58  % amount of time to add to second reflector delay
59  add = (d2 - p*3)*100/p;
60  s2 = [zeros(1,floor(add)), s];
61
62  % conbine signals
63  s1 = [s1, zeros(1,(size(s2,2)-size(s1,2)))];
64  st2 = (s1 + s2 + st)/(2*max(st));
65
66  len = size(rx,2);
67  st2(1:len) = st2(1:len) + rx;
68
69  figure
70  plot( t(1:size(st,2)), st2 )
```