Ultrasound Detection with Silicon Microring Resonators

Sarika Madhvapathy
Vladimir Stojanovic, Ed.
Ming C. Wu, Ed.

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-123
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-123.html

May 14, 2021
Ultrasound Detection with Silicon Microring Resonators

by

Sarika Madhvapathy

A thesis submitted in partial satisfaction of the requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Vladimir Stojanovic, Chair
Professor Ming Wu

Spring 2021
Ultrasound Detection with Silicon Microring Resonators

by Sarika Madhvapathy

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of Master of Science, Plan II.

Approval for the Report and Comprehensive Examination:

Committee:

[Signature]
Professor Vladimir Stojanovic
Research Advisor
5/14/21
(Date)

[Signature]
Professor Ming Wu
Second Reader
5/10/2021
(Date)
Abstract

Ultrasound Detection with Silicon Microring Resonators

by

Sarika Madhvapathy

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Vladimir Stojanovic, Chair

Ultrasound imaging systems are essential for breast tumor detection, stenting operations, and other endoscopic imaging applications. A beamforming array of silicon microring resonators (MRRs) is a promising solution for ultrasound imaging, as MRRs have been shown to have high sensitivity and high bandwidth. The power hungry circuitry of the analog front-end can be easily remoted outside of the body through a probe tube due to the small diameter of optical fibers. Using a comb laser, it is possible to interrogate an entire row with just one input. This helps reduce the probe tube size even further, as each input for a row requires a single optical fiber. With small diameters and remoted analog front end, the ring sensors can be compacted into a much smaller area, thus enabling higher bandwidth and higher resolution images. This technical report presents ultrasound receiver measurement results as well as photonic layout and analog designs for potential future generations of the project.
# Contents

## Contents

List of Figures ii

List of Tables iv

## 1 Introduction

1.1 Project overview ............................................. 1
1.2 Report overview ............................................. 2

## 2 Optical Ultrasound Receiver Measurements

2.1 Experimental setup .......................................... 4
2.2 Ultrasonic measurements .................................... 9
2.3 Electrical measurements .................................... 15

## 3 Photonic Layout

3.1 Berkeley Photonic Generator ................................. 18
3.2 Fanout for a commercial fiber block ..................... 18
3.3 Generator design .............................................. 20
3.4 Generated layouts ........................................... 28

## 4 Analog Frontend Redesign

4.1 Existing Topology ............................................ 31
4.2 Proposed Topology .......................................... 33
4.3 Simulation Results .......................................... 38

## 5 Conclusion and Future Work

Bibliography 42
## List of Figures

2.1 Immersion PMUT impinging pressure through water onto the receiver chip   3  
2.2 Hydrophone measurement setup.                                        5  
2.3 One 64-bit data packet, which contains the ADC output for a receiver quad 6  
2.4 FMC breakout board.                                                   7  
2.5 Ultrasound measurement setup.                                         8  
2.6 Transfer functions for various transducers, where $h$ is the vertical distance between each transducer and the hydrophone.     10  
2.7 Angular measurements.                                                11  
2.8 Bandwidth measurements using various transducers, where $h$ is the vertical distance between each transducer and the chip and the PMUT is centered over the ring.     12  
2.9 Block diagram for oscilloscope data processing code to recover analog data from each ring in a quad.                              13  
2.10 Received signals recovered from serialized data packets captured on oscilloscope.                                         14  
2.11 Schematic of AFE.                                                    15  
2.12 Result of gain/$I_{DAC}$ sweeping script.                            17  
3.1 Microscope image of the face of the PROFA tip with 61 channels.  [4, p. 610] 19  
3.2 BPG generated fanouts for PROFA fiber blocks.                         29  
3.3 BPG generated top level layouts to sense ultrasound with PROFA fiber blocks.                                             30  
4.1 Schematic of current AFE implementation.                              31  
4.2 Small signal model of current TIA implementation with equivalent noise generator, where $i_{dn}^2 = 4kT\gamma g_{mn}\Delta f$ and $i_{dp}^2 = 4kT\gamma g_{mp}\Delta f$.   32  
4.3 Schematic of proposed AFE implementation with feedback resistor omitted.                                                 33  
4.4 Schematic for $G_m$ calculation.                                       34  
4.5 Schematic for $R_{out}$ calculation.                                   35  
4.6 Equivalent MOSFET resistances.                                        35  
4.7 Schematic of proposed AFE implementation with equivalent noise generators, where $i_d^2 = 4kT\gamma g_{mk}\Delta f$.                              37  
4.8 Schematic of testbench with both TIA implementations (including DC operating points) with feedback resistor omitted.     38  
4.9 Noise simulation setup.                                               39
4.10 Voltage gain $A_V$ of each TIA implementation. ........................................... 40
4.11 Noise simulation results. .................................................................................. 40
List of Tables

3.1 Fiber array fanout parameters. .................................................. 20
3.2 MMI splitter tree + ring rows parameters. ................................. 24
3.3 Top level parameters. ............................................................... 26
3.4 Current fiber block vs. PROFA implementation (assuming the PROFA implemen-
   tation uses 16 rings per row). ................................................... 28

Listings

2.1 Pseudocode to find optimal AFE settings, loosely written in Python. ... 16
3.1 Generator for grating coupler fanout. ........................................ 20
3.2 Generator for splitter tree + ring rows. .................................... 24
3.3 Generator for top level layout. .................................................. 26
Acknowledgments

Thanks to Panagiotis Zarkos for all of his help and guidance, to Sidney Buchbinder for helping me debug all my BPG errors, to Professor Vladimir Stojanovic for advising me, to Christos Adamopoulos and the rest of the group, to the CITRIS Invention Lab for 3D-printing several water tanks for us, and to my friends and family for their support.
Chapter 1

Introduction

1.1 Project overview

Motivation

Ultrasound imaging systems are essential for breast tumor detection, stenting operations, and other endoscopic imaging applications. However, while they use established technology, commercial ultrasound imagers with piezoelectric micromachined ultrasound transducers (PMUTs) or capacitive micromachined ultrasound transducers (CMUTs) have several drawbacks:

• high probe form factor,
• high power consumption inside the body,
• large probe tube diameter,
• and limited bandwidth due to pitch constraints ($\lambda/2$ to avoid formation of grating lobes, where $\lambda$ is the wavelength corresponding to the operating frequency).

In the past, sub-array beamforming [2] and element level digitization [5] have been proposed in order to address the difficulty of minimizing the number of interfacing cables for a large number of sensors in order to maintain an adequately small probe tube size.

Proposed solution

A silicon microring resonator (MRR) experiences pressure-induced modulation of its resonant frequency due to waveguide deformation, ring elongation, and the opto-elastic effect [7]. This resonant frequency modulation results in modulation of the output optical power, which can be converted to a voltage through a analog front end (AFE) consisting of a ring photodetector and a transimpedance amplifier (TIA). MRRs have also been shown to have high sensitivity [6] and high bandwidth.
CHAPTER 1. INTRODUCTION

Furthermore, the power hungry circuitry of each AFE can be easily remoted outside of
the body through the probe tube in a dual-chip scheme, as the diameter of optical fibers is
significantly smaller than that of the micro-coaxial cables used in PMUT-based ultrasound
imaging systems.

Ring resonators behave like a notch filter with the notch located at the ring’s resonant
wavelength. Using a comb laser with comb teeth parked at the flanks of the resonant wave-
lengths for each ring in a row, it is possible to interrogate an entire row with just one input.
This helps reduce the probe tube size even further, as only a single optical fiber is required
as input for all of the rows.

With diameters of approx. 10 µm and remoted analog front end, the ring sensors can be
compacted into a much smaller area, thus enabling higher bandwidth (λ/2 pitch) and higher
resolution images. For all of these reasons, implementing a beamforming array of MRRs is
a promising solution for ultrasound imaging.

1.2 Report overview

The purpose of this technical report is to provide results for the current generation of this
project as well as potential ideas for future iterations. This report surveys three different
aspects of this project:

1. measurement results,

2. photonic layout with Berkeley Photonic Generator (BPG),

3. and a potential redesign of the analog front end (AFE).
Chapter 2

Optical Ultrasound Receiver Measurements

Figure 2.1: Immersion PMUT impinging pressure through water onto the receiver chip.

This chapter presents the measurement setup and measurement results for the MRR ultrasound receivers using both the on-chip analog frontend as well as an external photodiode with transimpedance amplifier (PD-TIA) for bandwidth measurements. It also displays the measured frequency responses of commercial PMUT immersion transducers used to impinge pressure upon the MRR receivers (as in Figure 2.1) using a commercial hydrophone with known sensitivity.
2.1 Experimental setup

Calibration setup

The pressure applied onto the chip by the PMUT is not known a priori, thus requiring a measurement setup to characterize the pressure exerted by the transducer using a commercial hydrophone with known sensitivity. This enables an accurate receiver sensitivity estimate, as the sensitivity for a ring is a function of the applied pressure.

Figure 2.2 details the measurement setup for the hydrophone measurements. A hole with diameter slightly smaller than the base of a standard-size immersion ultrasonic transducer is drilled at the bottom of a sealed resin water tank. An immersion transducer inside the tank is screwed into a UHF-to-BNC connector on the outside of the tank through the hole, thus sealing the bottom of the tank. The tank is propped up on either side of the hole to make room at the bottom of the tank for the UHF-to-BNC connector and its attached BNC cable.

An Onda HGL-series hydrophone is positioned above the transducer using an external adjustable Thorlabs stage with the tip facing the transducer. The tank is filled with water such that both the transducer and the hydrophone are fully immersed.

The hydrophone output is fed into an Onda pre-amplifier with 20 dB voltage gain before being read by the oscilloscope. An arbitrary wave generator (AWG) is used to excite the ultrasound transducer through the BNC cable at the bottom of the tank. One signal from the AWG’s differential output is used to trigger the oscilloscope, while the other is used to excite the transducer. The peak-to-peak voltage output of the pre-amplifier is measured on the oscilloscope and recorded manually on a PC.
Figure 2.2: Hydrophone measurement setup.
Breakout board design and electrical testing setup

Figure 2.3: One 64-bit data packet, which contains the ADC output for a receiver quad.

The MRRs on the chip itself can both sense and receive, as each ring has dedicated analog frontend (AFE) circuitry to receive the modulated signal.

The AFE for a ring is routed to a successive-approximation (SAR) analog-to-digital converter (ADC) that converts the analog signal to a 9-bit digital signal. For each quad of sensing rings, the ADC outputs are concatenated and appended to a 28-bit preamble, as shown in Figure 2.3. The 64-bit packets for each quad are then serialized with a 64-to-1 serializer and output through the hostboard’s FPGA mezzanine card (FMC) connector, with the intent of receiving these packets through the GTX/GTH transceivers on a Xilinx field-programmable gate array (FPGA) board and processing the data in real time.

To circumvent delays in verifying the Xilinx FPGA boards’ transceiver functionality, it is possible to save the waveforms to an oscilloscope and process the data in post. This required the design of a custom SubMiniature version A (SMA) breakout board to read out the serial data, as most existing FMC breakout boards have the incorrect FMC connector type since they are designed to connect to an FPGA board rather than a daughterboard.

The PCB pictured in Figure 2.4 was designed with Altium Designer. The 2-layer board has 16 differential outputs (for a total of 32 SMA connectors) and one differential clock input (2 more SMA connectors) that is used to clock the on-chip ADCs. 16 SMA connectors are surface mount edge connectors and 18 are through-hole top connectors. The routing utilizes 50 Ohm coplanar ground waveguides; the ground planes on the top and bottom are stitched together using a via array. Each differential pair was designed to have matched lengths.
(a) Three-dimensional rendering of front of breakout board.

(b) Three-dimensional rendering of back of breakout board.

(c) Breakout board (left, without top SMAs soldered) connected to hostboard (right).

Figure 2.4: FMC breakout board.
Figure 2.5 depicts the ultrasound receiver measurement setup. A 3D printed tank made of PLA filament is attached to the chip board using silicone; the pins on the chipboard are covered with silicone as well to isolate them from the water that will fill the tank. The chipboard is attached to the hostboard, which connects to the breakout board via an FMC connector. A laser is parked at the left (stable) flank of the target ring’s Lorentzian and is input to the sensing array through an optical fiber. The tank is filled with water and a PMUT excited by either an AWG or an ultrasonic pulser is immersed in the water. Each measurement (a stream of the data packets in Figure 2.3) is captured on the oscilloscope and saved to an external drive for post-processing.
2.2 Ultrasonic measurements

Transducer calibration

The following measurements are included:

1. frequency response of ultrasound transducer,
2. angular response (varying horizontal distance between transducer and hydrophone),
3. and response to varying the vertical distance between the transducer and hydrophone.

The pressure in Pascals is calculated using the following formula:

\[ P_{pp} \, [\text{Pa}] = \frac{V_{pp} \, [\text{V}]}{\text{sensitivity} \, [\text{V/Pa}] \cdot G_{\text{pre-amp}}} \]

where \( V_{pp} \) is the peak-to-peak voltage of the pre-amplifier output and the hydrophone sensitivity at a particular frequency is given by Onda calibration data. \( G_{\text{pre-amp}} = 20 \, \text{dB} \) is the voltage gain of the pre-amplifier. Measurements to verify the linear response of each transducer were taken as well by varying the amplitude of the sinusoidal excitation while fixing its frequency.
CHAPTER 2. OPTICAL ULTRASOUND RECEIVER MEASUREMENTS

Figure 2.6: Transfer functions for various transducers, where $h$ is the vertical distance between each transducer and the hydrophone.

Figure 2.6 includes the frequency response of several immersion PMUTs. Figure 2.6a demonstrates that the transfer function of the immersion transducer degrades with extensive use and requires a rest period after a lengthy period of excitation.
CHAPTER 2. OPTICAL ULTRASOUND RECEIVER MEASUREMENTS

(a) Angular measurement setup. (b) Pressure vs. angle for A310S PMUT, \( h = 3.2 \) cm.

(c) Pressure vs. angle for V312 PMUT, \( h = 1.6 \) cm. (d) Pressure vs. height for V312 PMUT.

Figure 2.7: Angular measurements.

The adjustable stage in the transducer measurement setup shown in Figure 2.2 has three degrees of freedom — it can be adjusted in the x, y, and z directions. It is difficult to mount the hydrophone such that it can be accurately placed at an angle to the hydrophone while maintaining the same distance between the transducer and the hydrophone. Instead, the setup illustrated by Figure 2.7a is used. The adjustable stage is varied in the y-direction while the x and z directions remain constant. The angle, \( \theta \), between the hydrophone and the PMUT is calculated by \( \theta = \tan^{-1} \left( \frac{h}{d} \right) \). Measurements were also taken varying \( h \) by adjusting the stage only in the z-direction with the hydrophone centered over the transducer in the xy-plane. Figure 2.7 displays the results of these positional measurements. Each measurement was taken by exciting each transducer with a sinusoid at its resonant frequency.
Angular measurements were taken for the A313S transducer as well, but the plot has been omitted since the measured pressure was approximately 851 Pa for all angles.

**MRR ultrasound receiver measurements**

**Bandwidth measurements with external TIA**

Bandwidth measurements were taken using an external Thor Labs PD-TIA and measured directly on an oscilloscope. As the frequency of the transducer excitation is swept, the peak-to-peak voltage output of the PD-TIA is measured on the oscilloscope and recorded manually on a PC.

![Bandwidth measurements](image)

(a) V309 (5 MHz), \( h = 1.6 \) cm  
(b) A313S (15 MHz), \( h = 1.6 \) cm  
(c) V324-SU (25 MHz), \( h = 1.1 \) cm

Figure 2.8: Bandwidth measurements using various transducers, where \( h \) is the vertical distance between each transducer and the chip and the PMUT is centered over the ring.
Sensing ultrasonic pulse

The oscilloscope waveforms from the setup in Figure 2.5 are parsed using a Python script to recover the received signal from the serial data packets. Figure 2.9 summarizes the script in the form of a block diagram.

![Block diagram for oscilloscope data processing code to recover analog data from each ring in a quad.](image)

Figure 2.9: Block diagram for oscilloscope data processing code to recover analog data from each ring in a quad.

Explanation of each block:

- \(\int\): comparator, outputs 1 if the scope\(_p\) > scope\(_n\) or 0 if scope\(_p\) < scope\(_n\)
- \(\downarrow\) OSR: downsamples the bitstream by the oversampling rate (OSR), which is defined as follows:
  \[
  \text{OSR} = \frac{f_{\text{sample, scope}}}{2 \cdot f_{\text{clk, ADC}}}
  \]
- Pattern alignment: deserializes serial data into 64-bit packets aligned to the 28-bit preamble \(\{14\text{'}b1, 14\text{'}b0\}\), isolates the data by keeping only the first 36 bits of each packet, and separating the data into 4 9-bit chunks
- \(DA\): converts 9-bit binary ADC outputs to decimal (units: SAR least significant bits (LSBs), where 512 LSBs corresponds to the fullscale ADC output)
- \(\mathcal{F}\mathcal{F}\): bandpass finite impulse response (FIR) filter centered at 5 MHz with cutoff frequencies 2.5 MHz and 7.5 MHz
Figure 2.10: Received signals recovered from serialized data packets captured on oscilloscope.

Figure 2.10 shows the output of the Python script described in Figure 2.9 for a ring quad with a pulse excitation centered at 5 MHz. Figure 2.10b shows the output of filtering the signals in 2.10a with a 101-tap FIR bandpass filter with cutoff frequencies at 2.5 MHz and 7.5 MHz, which are the assumed lower and upper limits of the frequencies in the pulse.

**Bandwidth and sensitivity results**

The sensitivity of a ring is calculated using the following formula:

\[
S = \frac{\delta V_{\text{out}}}{\delta P_{\text{app}}} \text{ [V / Pa]}
\]

\[
= \frac{\delta \lambda_{\text{res}}}{\delta P_{\text{app}}} \frac{\delta P_{\text{opt}}}{\delta \lambda_{\text{res}}} R_{PD} G_{TIA}
\]

where \(V_{\text{out}}\) is the measured peak-to-peak output voltage, \(P_{\text{app}}\) is the pressure applied (calculated using the transducer calibration measurements), \(\lambda_{\text{res}}\) is the resonant wavelength of the ring, \(P_{\text{opt}}\) is the input optical power, \(R_{PD}\) is the responsivity of the photodetector, and \(G_{TIA}\) is the TIA gain. \(^7\)

A 5MHz sinusoidal excitation yields a fitted peak-to-peak output of 75 LSB codes, so the estimated receiver sensitivity is calculated as follows:

\[
S = \frac{600 \text{ mV}}{512 \text{ LSBs}} \cdot 75 \text{ LSBs} \cdot \frac{1}{12.1 \text{ kPa}}
\]

\[
= 7.3 \text{ mV/kPa} \quad \text{[8]}
\]
where 600 mV is the fullscale output of the 9-bit ADC corresponding to 512 LSB codes and 12.1 kPa is the pressure applied to the chip, back-calculated from the PMUT characterization measurements outlined in the previous section.

The bandwidth of the MRR sensors is $\geq$ 30 MHz (Figure 2.8c). The true bandwidth may be higher than 30 MHz, but the PMUT with the highest resonant frequency used was 25 MHz (V324-SU).

### 2.3 Electrical measurements

![Schematic of AFE.](image)

Figure 2.11: Schematic of AFE.

Figure 2.11 depicts part of the analog frontend (AFE). A 5-bit tunable pull-down current digital-to-analog converter (DAC), $I_{DAC}$, sets the common-mode voltage at the TIA output ($V_{out}$). The tunable TIA feedback resistor $R_f$ has 4 configuration bits, allowing for 16 gain settings from 50k$\Omega$ to 800k$\Omega$. AFE settings such as TIA gain ($\approx R_f$) and $I_{DAC}$ code can be adjusted to optimize the Lorentzian shape for each ring in order to maximize ring sensitivity by maximizing $\frac{\delta P_{out}}{\delta \lambda_{res}}$. These configuration bits can be updated using a Python script to interface with the scan chain.
Listing 2.1: Pseudocode to find optimal AFE settings, loosely written in Python.

Listing 2.1 contains a simplified version of a Python script that iterates over the gain and IDAC settings to find the optimal configuration. A Lorentzian is found for each AFE setting by sweeping the input laser wavelength around the ring resonance and recording the SAR output through the scan chain.
Figure 2.12: Result of gain/$I_{DAC}$ sweeping script.

The output of this sweeping script is plotted in Figure 2.12. The optimal combination of gain and $I_{DAC}$ code is the blue Lorentzian, which has a maximum slope of 5.365 $\frac{SAR\ LSBs}{pm}$. The Lorentzians corresponding to two non-optimal settings have been plotted as well for comparison.
Chapter 3

Photonic Layout

This chapter proposes an automated, fully parameterizable photonic layout design for future generations of the ultrasound sensing chip using a commercial fiber block in order to address the tradeoff between maximizing the number of sensing elements and minimizing the size of the probe tube.

3.1 Berkeley Photonic Generator

Berkeley Photonic Generator (BPG) is a Python-based photonic layout automation tool. BPG enables a user to parameterize a design such that it is easy to generate a new layout by simply changing design parameters.

Each layout can be specified using a Python class with the following methods:

- `get_params_info(cls)`: describes each parameter in the design
- `get_default_param_values(cls)`: returns a dictionary with the default parameter values for the design
- `draw_layout(self)`: instance method that denotes the physical layout of the design

Layouts can be generated hierarchically; a top level `draw_layout(self)` script can instantiate classes corresponding to lower-level layouts defined by the user.

3.2 Fanout for a commercial fiber block

The current layout has receiver arrays with 4 rows of 8 rings per row. To enable higher resolution images in the future, it is vital to implement the following:

1. Significantly reduce the pitch between the MRRs.
2. Increase the total number of rings in the array.
3. Minimize the diameter of the probe tube.

The pitch-reducing optical fiber arrays (PROFAs) proposed by Kopp et al. [4] enable items 2 and 3. With a minimum of 7 and a maximum of 61 channels, these hexagonal PROFAs can enable a single comb laser input to provide input to anywhere from 6 to 60 ring rows through a tree of 50-50 multi-mode interferometer (MMI) splitters. The unused outputs of the MMI splitter tree can be routed to other couplers to characterize the loss of the MMI tree as well as the MMI splitting capability. Furthermore, since the grating coupler pitch is between 35 and 45 µm, the maximum probe tube diameter is approximately 0.405 mm. This combination of reduced coupler pitch and a large number of channels addresses the tradeoff between minimizing the amount of probe tube cabling and maximizing the number of sensing elements on the probe tip. The face of a PROFA tip with 61 channels is illustrated in Figure 3.1.

![Microscope image of the face of the PROFA tip with 61 channels.](image)

This chapter details the design and layout of this concept using BPG. This generator lays out a fanout for a PROFA using one fiber as an input and the remaining fibers as outputs, where the fiber pitch and number of fibers are fully parameterizable. The input grating coupler is routed to the input of an MMI splitter tree; each output of the MMI splitter tree is routed to a ring row, the output of which is routed back to the PROFA fanout. The ring array pitch is parameterizable as well.
3.3 Generator design

The hierarchy of the generator is as follows:

```
Top level generator
  Grating coupler hexagonal array + fanout
  MMI splitter tree + ring rows
```

**Grating coupler hexagonal array + fanout**

This generator lays out a hexagonal array of grating couplers with the specified grating coupler pitch; it then fans out the port of each grating coupler to the right using the specified waveguide pitch. The script calculates how many waveguides can fit between two rows of grating couplers, and routes the remaining coupler ports that can't fit between two rows around the fiber block.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_layout_package</td>
<td>Layout package for a ring row</td>
</tr>
<tr>
<td>row_class_package</td>
<td>Class package for a ring row</td>
</tr>
<tr>
<td>row_params</td>
<td>List of params for a ring row</td>
</tr>
<tr>
<td>gc_layout_package</td>
<td>Layout package for the grating coupler</td>
</tr>
<tr>
<td>gc_class_package</td>
<td>Class package for the grating coupler</td>
</tr>
<tr>
<td>wg_bend_radius</td>
<td>Waveguide bend radius</td>
</tr>
<tr>
<td>wg_pitch</td>
<td>Minimum pitch between adjacent waveguides</td>
</tr>
<tr>
<td>rout</td>
<td>List of ring radii in a row</td>
</tr>
<tr>
<td>gc_pitch</td>
<td>Pitch between grating couplers</td>
</tr>
<tr>
<td>min_coupler_row</td>
<td>Number of couplers in the top row of the array</td>
</tr>
</tbody>
</table>

Table 3.1: Fiber array fanout parameters.

draw_layout(self) code:

```python
1    gc_pitch = self.params['gc_pitch']
2    min_coupler_row = self.params['min_coupler_row']
3    wg_pitch = self.params['wg_pitch']
4    wg_bend_radius = self.params['wg_bend_radius']
5    num_rows = 2 * min_coupler_row - 1
6    gc_layout_package = self.params['gc_layout_package']
7    gc_cls_name = self.params['gc_class_package']
8    gc_lay_module = importlib.import_module(gc_layout_package)
9    gc_cls = getattr(gc_lay_module, gc_cls_name)
10   gc_master = self.new_template(params=dict(), temp_cls=gc_cls)
11   num_coupes = min_coupler_row
12   start_offset_x = 0
```
offset_y = 0

# Calculate maximum number of waveguides that will fit between coupler rows
max_wgs = math.floor(gc_pitch / np.sqrt(2) / wg_pitch) - 1
around_offset_y = min(num_couplers - 1, max_wgs - 1) * wg_pitch \
+ wg_pitch - 2 * wg_bend_radius
around_offset_x = gc_pitch
end_x = (2 * min_coupler_row - 1) * gc_pitch
idx = 0
min_actual_y = 0
top_gc_rows = []
bottom_gc_rows = []

# Iterate over array rows to generate grating coupler array
for i in range(num_rows):
    dir = 1
    num_around = num_couplers - max_wgs - 1

    # Iterate over all grating couplers in a row
    for j in range(num_couplers):
        if i < min_coupler_row - 1:
            dir = -1
        elif i > min_coupler_row - 1:
            dir = 1
        offset_x = start_offset_x + j * gc_pitch
        gc_inst = self.add_instance(master=gc_master, 
                                    loc=(offset_x, offset_y), 
                                    orient='R180')
        gc_port = gc_inst['INPUT']

        # Begin route at output of grating coupler
        router = WgRouter(gen_cls=self, init_port=gc_port, 
                          layer=('rx1phot', 'drawing'), name='route {}{}'.format(i, j), 
                          wg_params={'bend_type': 'trajECE', 'radius': wg_bend_radius, 
                                      'AngleTurnEuler': .5})

        if dir == 1:
            dir1 = 'left'
            dir2 = 'right'
        else:
            dir2 = 'left'
            dir1 = 'right'

        # Last coupler in a row
        if j == num_couplers - 1:
            loc = router.port.center
            router.cardinal_router(points=[(end_x, loc[1])], 
                                    bend_params=dict(radius=wg_bend_radius, 
                                                    bend_type='trajCirc'))
            if i == min_coupler_row - 1:
                router_start = router

        # Can't fit all waveguides between rows
        elif (j < num_couplers - max_wgs - 1 and i != min_coupler_row - 1) \
            or (i == min_coupler_row - 1 and (num_couplers - j) // 2 > max_wgs):
            if dir == 1:
                around_offset_x = around_offset_x - wg_pitch
                around_offset_y = around_offset_y
            dist = 1 + j * wg_pitch
            (router
.add_straight_wg(length=0.5)
.add_90_bend(direction=dir1,
    bend_params=dict(radius=wg_bend_radius, bend_type='circular'))
.add_straight_wg(length=dist))
loc = router.port.center
if i == min_coupler_row - 1:
    num_around = num_couplers - 2 * max_wgs
    if dir == -1:
        actual_x = around_offset_x + (num_around - 1 - 2 * j // 2) * wg_pitch
        actual_y = around_offset_y - 2 * wg_bend_radius + i * gc_pitch / np.sqrt(2) +
                    (idx - 3 * j // 2 + num_around - 1) * gc_pitch / np.sqrt(2) +
                    wg_pitch * (idx - j // 2 - 1)
    else:
        actual_x = around_offset_x + wg_pitch
        actual_y = around_offset_y - 2 * wg_bend_radius + (num_rows - 1 - i) * gc_pitch / np.sqrt(2) +
                    wg_pitch * (-j // 2 + idx + 1)
elif dir == -1:
    actual_x = around_offset_x + (num_around - 1 - 2 * j) * wg_pitch
    actual_y = around_offset_y - 2 * wg_bend_radius + i * gc_pitch / np.sqrt(2) +
                wg_pitch * (-j // 2 + idx) +
    if loc[0] > -around_offset_x:
        if dir == 1:
            idx -= 1
        router.add_90_bend(direction=dir1,
            bend_params=dict(radius=wg_bend_radius, bend_type='circular'))
        loc = router.port.center
        (router
            .cardinal_router(points=[(-actual_x + wg_bend_radius, loc[1])],
            bend_params=dict(radius=wg_bend_radius, bend_type='trajCirc'))
            .add_90_bend(direction=dir2, bend_params=dict(radius=wg_bend_radius, bend_type='circular'))
            .add_straight_wg(length=actual_y)
            .add_90_bend(direction=dir2, bend_params=dict(radius=wg_bend_radius, bend_type='circular')))
        loc = router.port.center
        router.cardinal_router(points=[(end_x, loc[1])],
            bend_params=dict(radius=wg_bend_radius, bend_type='trajCirc'))
        if dir == -1:
            idx += 1
        else:
            (router
                .add_straight_wg(length=around_offset_y + min(i, num_rows - 1 - i) * gc_pitch / np.sqrt(2))
                .add_90_bend(direction=dir2, bend_params=dict(radius=wg_bend_radius, bend_type='circular')))
            loc = router.port.center
            router.cardinal_router(points=[(end_x, loc[1])],
                bend_params=dict(radius=wg_bend_radius, bend_type='trajCirc'))
            if actual_y > min_actual_y and dir == -1:
                min_actual_y = actual_y
            if dir == -1:
                around_offset_x = around_offset_x + wg_pitch
                around_offset_y = around_offset_y
# Can fit waveguide between rows

else:
    if i == min_coupler_row - 1:
        dist = 1 + (num_couplers - j - 2) // 2 * wg_pitch + wg_pitch - 2 * wg_bend_radius
    else:
        dist = 1 + (num_couplers - j - 2) * wg_pitch + wg_pitch - 2 * wg_bend_radius

    (router
        .add_straight_wg(length=0.5)
        .add_90_bend(direction=dir1,
            bend_params=dict(radius=wg_bend_radius, bend_type='circular'))
        .add_straight_wg(length=dist)
        .add_90_bend(direction=dir2,
            bend_params=dict(radius=wg_bend_radius, bend_type='circular')))  

    loc = router.port.center
    router.cardinal_router(points=[(end_x, loc[1])],
        bend_params=dict(radius=wg_bend_radius, bend_type='trajCirc'))

    if dir == 1:
        top_gc_rows.append(router)
    else:
        bottom_gc_rows.append(router)
    dir = -dir

if i < min_coupler_row - 1:
    start_offset_x = start_offset_x - gc_pitch / 2 
    num_couplers = num_couplers + 1
else:
    start_offset_x = start_offset_x + gc_pitch / 2 
    num_couplers = num_couplers - 1
    offset_y = offset_y + gc_pitch * 1 / np.sqrt(2)

top_gc_rows = sorted(top_gc_rows, key=lambda x: x.port.center[1])
bottom_gc_rows = sorted(bottom_gc_rows, key=lambda x: x.port.center[1])

# Extract output ports
for idx, row in enumerate(top_gc_rows):
    row.extract_port('TOP_' + str(idx))
for idx, row in enumerate(bottom_gc_rows[:: -1]):
    row.extract_port('BOTTOM_' + str(idx))

# Extract input port
router_start.extract_port('CENTER')

Listing 3.1: Generator for grating coupler fanout.
CHAPTER 3. PHOTONIC LAYOUT

MMI splitter tree + ring rows

This generator lays out a binary MMI splitter tree based on the number of possible outputs in the fiber array fanout. It calculates the tree depth as follows:

$$\text{depth} = \lceil \log_2 n \rceil$$

where $n$ is the number of outputs. It then lays out $n$ MRR rows, which means that there are $2^{\text{depth}} - n$ unused splitter tree outputs.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch_tree_layout_package</td>
<td>Layout package for the binary switch tree</td>
</tr>
<tr>
<td>switch_tree_class_package</td>
<td>Class package for the binary switch tree</td>
</tr>
<tr>
<td>switch_elem_pitch</td>
<td>Element pitch for the binary switch tree</td>
</tr>
<tr>
<td>row_layout_package</td>
<td>Layout package for a ring row</td>
</tr>
<tr>
<td>row_class_package</td>
<td>Class package for a ring row</td>
</tr>
<tr>
<td>row_params</td>
<td>List of params for a ring row</td>
</tr>
<tr>
<td>wg_pitch</td>
<td>Minimum pitch between adjacent waveguides</td>
</tr>
<tr>
<td>wg_bend_radius</td>
<td>Waveguide bend radius</td>
</tr>
<tr>
<td>contact_name_start_ind</td>
<td>The start index for contact label names</td>
</tr>
<tr>
<td>num_rings_per_row</td>
<td>The number of rings per row in this array</td>
</tr>
<tr>
<td>coupling_gap_base_1</td>
<td>Initial coupling gap of left array</td>
</tr>
<tr>
<td>coupling_gap_pitch_1</td>
<td>Pitch with which coupling gap increments - left array</td>
</tr>
<tr>
<td>rout</td>
<td>List of ring radii in a row</td>
</tr>
<tr>
<td>min_coupler_row</td>
<td>Number of couplers in the top row of the array</td>
</tr>
</tbody>
</table>

Table 3.2: MMI splitter tree + ring rows parameters.

draw_layout(self) code:

1. `switch_tree_layout_package = self.params[‘switch_tree_layout_package’]`
2. `switch_tree_cls_name = self.params[‘switch_tree_class_package’]`
3. `switch_tree_lay_module = importlib.import_module(switch_tree_layout_package)`
4. `switch_tree_cls = getattr(switch_tree_lay_module, switch_tree_cls_name)`
5. `row_layout_package = self.params[‘row_layout_package’]`
6. `row_cls_name = self.params[‘row_class_package’]`
7. `row_lay_module = importlib.import_module(row_layout_package)`
8. `row_cls = getattr(row_lay_module, row_cls_name)`
9. `min_coupler_row = self.params[‘min_coupler_row’]`
10. `num_rows = 3 * sum([2 * i for i in range(1, min_coupler_row)])`
11. `tree_depth = math.ceil(math.log2(num_rows))`
12. `row_offset = (2 ** tree_depth - num_rows) // 2`
\textbf{CHAPTER 3. PHOTONIC LAYOUT}

```
num_rings_tot = num_rows * self.params[\'num_rings_per_row\'] - 1

switch_tree_params = dict(element_pitch=self.params[\'switch_elem.pitch\'],
                          input_port_name=\'IN\_TOP\', tree_depth=tree_depth)

switch_tree_master = self.new_template(params=switch_tree_params, temp_cls=switch_tree_cls)
switch_tree_inst_left = self.add_instance(master=switch_tree_master, loc=(0, 0))

self.extract_photonic_ports(inst=switch_tree_inst_left, port_names=[\'INPUT\'])

contact_name_ind = 0

for ind in range(num_rows):
    curr_dict = deepcopy(self.params[\"row_params\"])
    name_str = \'Mod1_\'
    coupling_gap_var = dict(
        ring_params_list=[dict(rout=self.params[\'rout\'][x],
                               coupling_slot=self.params[\'coupling_slot_base\'] + x *
                               self.params[\'coupling_slot_pitch\']) for x in
                               range(self.params[\'num_rings_per_row\'])])

    contact_name_dict = dict(
        ring_labels=[dict(N=name_str + \' RingCat \'+ \'<\' + str(num_rings_tot - x) + \ ?>' ,
                       P=name_str + \' RingAn \'+ \'<\' + str(num_rings_tot - x) + \>}' ) for
                       x in range(contact_name_ind, contact_name_ind
                       + self.params[\'num_rings_per_row\'], 1)],
        heater_labels=[dict(N=name_str + \' HeaterP \'+ \'<\' + str(num_rings_tot - x) + \>}' ,
                       P=name_str + \' HeaterN \'+ \'<\' + str(num_rings_tot - x) + \>(') for
                       x in range(contact_name_ind, contact_name_ind
                       + self.params[\'num_rings_per_row\'], 1)])

    if ind % 2 == 1:
        contact_name_dict[\'ring_labels\'].reverse()
        contact_name_dict[\'heater_labels\'].reverse()

    curr_dict.update(contact_name_dict)
    contact_name_ind += self.params[\'num_rings_per_row\']

    curr_dict.update(coupling_gap_var)

    row_master = self.new_template(params=curr_dict, temp_cls=row_cls)

    switch_port = switch_tree_inst_left[\'OUTPUT_\'+ str(ind + row_offset)]

    router = WgRouter(gen_cls=self,
                      init_port=switch_port,
                      layer=(\'rxphot\', \'drawing\'),
                      name=\'route\'+ str(ind + 1))

    row_inst = self.add_instance_port_to_port(inst_master=row_master,
                                              instance_port_name=\'PORT1\',
                                              self_port=router.port,
                                              reflect=True)

    self.extract_photonic_ports(inst=row_inst, port_names=[\'PORT0\'],
                                 port_renaming={\'PORT0\': \'OUTPUT\_(\{}\).format(ind + 1)\}})
```

Listing 3.2: Generator for splitter tree + ring rows.
Top-level generator

The top-level generator instantiates the fiber array as well as the MMI splitter tree and ring rows. It then iteratively routes all ring row outputs back to the ports of the grating coupler fanout.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fiber_arr_layout_package</td>
<td>Layout package (path) to the fiber array</td>
</tr>
<tr>
<td>fiber_arr_class_package</td>
<td>Class package (class name) to the fiber array</td>
</tr>
<tr>
<td>fiber_arr_params</td>
<td>dict of params for the fiber array</td>
</tr>
<tr>
<td>splitter_rows_layout_package</td>
<td>Layout package (path) to the splitter + ring rows</td>
</tr>
<tr>
<td>splitter_rows_class_package</td>
<td>Class package (class name) to the splitter + ring rows</td>
</tr>
<tr>
<td>wg.bend_radius</td>
<td>Radius of the waveguide bends</td>
</tr>
<tr>
<td>wg.width</td>
<td>Waveguide width</td>
</tr>
<tr>
<td>wg.pitch</td>
<td>Minimum waveguide pitch</td>
</tr>
<tr>
<td>min_coupler_row</td>
<td>Number of couplers in the top row of the array</td>
</tr>
</tbody>
</table>

Table 3.3: Top level parameters.

draw_layout(self) code:

```python
# Add fiber array first as (0, 0)
fiber_arr_layout_package = self.params['fiber_arr_layout_package']
fiber_arr_class_package = self.params['fiber_arr_class_package']
gen_params = dict(min_coupler_row=self.params['min_coupler_row'], wg_pitch=self.params['wg_pitch'])
fiber_arr_lay_module = importlib.import_module(fiber_arr_layout_package)
fiber_arr_cls = getattr(fiber_arr_lay_module, fiber_arr_cls_name)
fiber_arr_master = self.new_template(params=gen_params, temp_cls=fiber_arr_cls)
fiber_arr_inst = self.add_instance(master=fiber_arr_master, loc=(0, 0))

# Calculating total number of ring rows and MMI splitter tree depth
num_rows = 3 * sum([2 * i for i in range(1, fiber_arr_master.params['min_coupler_row'])])
tree_depth = math.ceil(math.log2(num_rows))

wg_pitch = self.params['wg_pitch']

# Route center-most grating coupler to input of MMI splitter tree
router = WgRouter(gen_cls=self,
                  init_port=fiber_arr_inst['CENTER'],
                  layer=('rxiphot', 'drawing'),
                  name='route0',
                  wg_params={'bend_type': 'trajECE', 'radius': 10, 'AngleTurnEuler': .5})

splitter_rows_layout_package = self.params['splitter_rows_layout_package']
splitter_rows_cls_name = self.params['splitter_rows_class_package']

splitter_rows_lay_module = importlib.import_module(splitter_rows_layout_package)
splitter_rows_cls = getattr(splitter_rows_lay_module, splitter_rows_cls_name)
```
CHAPTER 3. PHOTONIC LAYOUT

splitter_rows_master = self.new_template(params=gen_params, temp_cls=splitter_rows_cls)

row_pitch = splitter_rows_master.params['switch_elem_pitch']

router.add_straight_wg(length=50 + num_rows // 2 * wg_pitch)
router.add_s_bend(length=30, shift_left=row_pitch // 2)

splitter_rows_inst = self.add_instance_port_to_port(inst_master=splitter_rows_master,
instance_port_name='INPUT',
self_port=router.port)

row_offset = (2 ** tree_depth - num_rows) // 2  # Calculate number of unused splitter outputs

# Route all ring row outputs to fiber array fanout
for route_num in range(1, num_rows + 1):
    if route_num < (num_rows) // 2 + 1:
        dir = 'left'
        out_len = route_num + 1
        row = fiber_arr_inst['TOP_{0}'.format(route_num)]
        end_y = row.center[1]
    else:
        dir = 'right'
        out_len = num_rows - route_num + 2
        row = fiber_arr_inst['BOTTOM_{0}'.format(num_rows - route_num)]
        end_y = row.center[1]

    router = WgRouter(gen_cls=self,
    init_port=splitter_rows_inst['OUTPUT_{0}'.format(route_num)],
    layer=('rxphot', 'drawing'),
    name='route0',
    wg_params={'bend_type': 'trajECE', 'radius': 10, 'AngleTurnEuler': .5})

    (router
    .add_straight_wg(length=wg_pitch)
    .add_straight_wg(length=wg_pitch * out_len)
    .add_90_bend(direction=dir,
        bend_params=dict(radius=self.params['wg_bend_radius'], bend_type='circular'))
    .add_straight_wg(length=100 + (out_len + row_offset) * row_pitch + out_len * wg_pitch)
    .add_90_bend(direction=dir,
        bend_params=dict(radius=self.params['wg_bend_radius'], bend_type='circular'))
    .add_straight_wg(length=row_pitch * splitter_rows_master.params['num_rings_per_row']
    + 2 * out_len * wg_pitch + 90 * tree_depth)
    .add_90_bend(direction=dir,
        bend_params=dict(radius=self.params['wg_bend_radius'], bend_type='circular'))
    )

diff = abs(router.port.center[1] - end_y) - 2 * self.params['wg_bend_radius']

(router
    .add_straight_wg(length=diff)
    .cardinal_router(points=[row.center],
        bend_params=dict(radius=self.params['wg_bend_radius'], bend_type='trajCirc'))))

Listing 3.3: Generator for top level layout.
3.4 Generated layouts

Figure 3.2 displays fanouts for PROFA tips of different sizes with the grating couplers omitted. Figure 3.2a has 19 channels with \( \text{wg\_pitch} = 5 \, \mu\text{m} \), Figure 3.2b has 37 channels with a waveguide pitch of \( 5 \, \mu\text{m} \), and Figure 3.2c has 61 channels with \( \text{wg\_pitch} = 6 \, \mu\text{m} \). The grating coupler pitch for each fanout is 40 \( \mu\text{m} \). Note that all the coupler outputs that do not fit between coupler rows are routed around the fiber block. Therefore, the maximum \( \text{wg\_pitch} \) is reached when the total number of couplers is equal to \( 2 \cdot \text{max\_wgs} \), where \( \text{max\_wgs} \) (see line 21 in Listing 3.1) is the maximum number of waveguides that will fit between coupler rows with the given \( \text{gc\_pitch} \) and \( \text{wg\_pitch} \). Using too small a value for \( \text{wg\_pitch} \) may result in unwanted coupling. The reduced pitch between the couplers limits the maximum waveguide bend radius (\( \text{wg\_bend\_radius} \)) as well, which could potentially result in lossy waveguides.

Figure 3.3 displays the generated top-level layouts for each possible number of PROFA channels. In each image, the coupler array and fanout (with couplers omitted) is at the bottom. The coupler output at the center of the fanout routes to the input of the MMI splitter tree in the middle of the image. The MMI splitter outputs route to the ring rows at the top of the image; the ring row outputs are then routed back to the coupler fanout. In each generated layout, \( \text{num\_rings\_per\_row} = 16, \text{wg\_pitch} = 5 \, \mu\text{m} \), and the pitch between rings (\( \text{switch\_elem\_pitch} \)) is \( 50 \mu\text{m} \).

<table>
<thead>
<tr>
<th>Feature</th>
<th>Current fiber block</th>
<th>PROFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of input couplers</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Number of output couplers/ring rows</td>
<td>4</td>
<td>6-60</td>
</tr>
<tr>
<td>Grating coupler pitch</td>
<td>250( \mu\text{m} )</td>
<td>35-45( \mu\text{m} )</td>
</tr>
<tr>
<td>Ring pitch</td>
<td>220( \mu\text{m} )</td>
<td>50( \mu\text{m} )</td>
</tr>
<tr>
<td>Total number of rings</td>
<td>32</td>
<td>96-960</td>
</tr>
<tr>
<td>Approximate sensing array dimensions</td>
<td>0.75mm x 1.54mm</td>
<td>0.75mm x 0.25mm-2.95mm</td>
</tr>
</tbody>
</table>

Table 3.4: Current fiber block vs. PROFA implementation (assuming the PROFA implementation uses 16 rings per row).
Figure 3.2: BPG generated fanouts for PROFA fiber blocks.
Figure 3.3: BPG generated top level layouts to sense ultrasound with PROFA fiber blocks.
Chapter 4

Analog Frontend Redesign

This chapter explores a potential alternate transimpedance amplifier (TIA) implementation for the chip analog frontend (AFE). It is important to minimize TIA noise to avoid thermal noise from the metal-oxide-semiconductor field-effect transistors (MOSFETs) dominating the system over the shot noise from the photodetector; it is also desirable to maximize the TIA’s open-loop voltage gain $A_V$. The gain $\frac{V_{out}}{I_{in}}$ of a TIA is defined as follows:

$$G_{TIA} = -\frac{A_V}{1 + A_V R_f}$$

Therefore, as $A_V \to \infty$, $G_{TIA} \to -R_f$, where $R_f$ is the feedback resistance.

4.1 Existing Topology

![TIA schematic](image)

Figure 4.1: Schematic of current AFE implementation.
The existing AFE consists of a MRR photodetector (PD) in series with an inverter-based TIA with a variable feedback resistance $R_f$. The schematic for the existing AFE is illustrated in Figure 4.1.

This TIA is a **current-reuse** TIA; as the input is connected to the gates of both the NMOS and the PMOS, the effective transconductance $G_m$ is the sum of the NMOS and PMOS $g_m$’s. For the following analysis, the DC operation points $g_m$ and $r_o$ are approximated to be the same for the NMOS and the PMOS.

**Gain analysis**

To find the gain ($V_{out}/I_{in}$) of the TIA, the voltage gain $A_V = G_m R_{out}$ is found first in order to evaluate the TIA gain as follows:

$$\frac{V_{out}}{I_{in}} = -\frac{A_V}{1 + A_V} R_f = -\frac{G_m R_{out}}{1 + G_m R_{out}} R_f$$

![Figure 4.2: Small signal model of current TIA implementation with equivalent noise generator, where $i_{dn}^2 = 4kT\gamma g_{mn} \Delta f$ and $i_{dp}^2 = 4kT\gamma g_{mp} \Delta f$.](image)

By inspection of figure [4.2](#), the effective transconductance $G_m$ of the circuit is $g_{mp} + g_{mn}$ and the output resistance $R_{out} = r_{op}||r_{on} = (g_{dsp} + g_{dsn})^{-1}$. Therefore,

$$\frac{V_{out}}{I_{in}} = -\frac{(g_{mp} + g_{mn})(r_{op}||r_{on})}{1 + (g_{mp} + g_{mn})(r_{op}||r_{on})} R_f$$

$$\approx -\frac{2g_m \cdot \frac{1}{2}r_o}{1 + 2g_m \cdot \frac{1}{2}r_o} R_f$$

$$\approx -\frac{g_m r_o}{1 + g_m r_o} R_f$$
CHAPTER 4. ANALOG FRONTEND REDESIGN

33

Noise spectral density analysis

Ignoring the thermal noise from the feedback resistor, the total noise due to the thermal noise from the MOSFETs at the output is as follows:

\[
\begin{align*}
\overline{i_{\text{noise, out}}^2} &= i_{dn}^2 + i_{dp}^2 \\
&= 4kT\gamma(g_{mn} + g_{mp})\Delta f \\
\rightarrow \overline{i_{\text{noise, out}}^2} &= i_{\text{noise, out}}^2 \cdot R_{out}^2 \\
&= 4kT\gamma\frac{g_{mn} + g_{mp}}{(g_{ds} + g_{ds})^2} \Delta f \\
&\approx 2kT\gamma g_m r_o^2 \Delta f
\end{align*}
\]

4.2 Proposed Topology

The proposed TIA topology for the AFE is utilized in the piezoelectric ultrasound imaging scheme presented by D’Urbino et al [3]. It is illustrated in Figure 4.3 with feedback resistor and photodetector omitted. A feedback resistor \(R_f\) is connected between \(V^-\) and \(V_{out}\). The photodetector output is routed to \(V^-\) and \(V^+\) is connected to a reference voltage \((V_{ref})\). This TIA is also a current-reuse TIA, so the effective transconductance \(G_m\) will be approximately the sum of the NMOS and PMOS \(g_m\)’s. Unlike the existing TIA topology, this has a differential input.

![Figure 4.3: Schematic of proposed AFE implementation with feedback resistor omitted.](image)

For the following analysis, \(g_m\) and \(r_o\) are assumed to be the same for M2, M3, M5, and M6. M1 and M4 are assumed to have the same \(g_m\) (but different from the \(g_m\) of the
complementary transistors, as M1 and M4 are assumed to be in triode). Since M1 and M4 are assumed to be in triode, they can be replaced with resistances $R_{on,p}$ and $R_{on,n}$. The effect of the feedback from the drains of M2 and M3 to the gate of M1 is assumed to be negligible.

**Gain analysis**

**Transconductance**

To find the transconductance $G_m$, the output is shorted and a differential input ($V^+ = \frac{1}{2}v_{id}, V^- = -\frac{1}{2}v_{id}$) is applied to the circuit. $G_m$ is evaluated as the output current $I_{out}$ divided by $v_{id}$. The node between the sources of M3 and M5 is incremental ground because M3 and M5 have equal and opposite currents due to the differential input. The node between the sources of M3 and M6 is incremental ground for the same reason. This means that the circuit can be split into two differential half-circuits. Figure 4.4 displays the righthand half circuit.

![Figure 4.4: Schematic for $G_m$ calculation.](image)

$$I_{out} = g_{m5} \frac{v_{id}}{2} + g_{m6} \frac{v_{id}}{2}$$

$$= \frac{1}{2}v_{id}(g_{m5} + g_{m6})$$

$$\Rightarrow G_m = \frac{1}{2}(g_{m5} + g_{m6})$$

**Output resistance**

To find the output resistance, all independent sources are shorted and a test voltage is attached to the output as shown in Figure 4.5.
Figure 4.5: Schematic for $R_{out}$ calculation.

Figure 4.6: Equivalent MOSFET resistances.

The equivalent resistances in Figure 4.6 are $R_{eqS} = \frac{r_o + R_D}{g_m r_o + 1}$ (4.6a) and $R_{eqD} = r_o + g_m r_o R_S$.
These equivalences are used to solve for the output resistance of the TIA.

\[ R_{\text{out}} = \frac{V_{\text{test}}}{I_{\text{test}}} \]

\[ I_{\text{test}} = i_5 + i_6 = \frac{V_{\text{test}}}{R_{eq5}} + \frac{V_{\text{test}}}{R_{eq6}} \]

\[ R_{eq5} = r_{o5} + g_{m5}r_{o5} \cdot (R_{on,p} || R_{eq,S,3}) = r_{o5} + g_{m5}r_{o5} \cdot \left( R_{on,p} \parallel \frac{r_{o3} + g_{m2}r_{o2}R_{on,n}}{g_{m}r_{o3} + 1} \right) \approx r_{o5} + g_{m5}r_{o5} \cdot \left( R_{on,p} \parallel \frac{1}{g_{m}} + R_{on,n} \right) \quad (g_{m}r_{o} \gg 1) \]

The transconductance \( g_{m} \) of a MOSFET in saturation is defined as

\[ g_{m} = \frac{\delta I_{DS}}{\delta V_{GS}} \bigg|_{V_{DS}} = \mu C_{ox} \frac{W}{L} (|V_{GS}| - |V_{T}|) \]

and the on resistance of a MOSFET in triode is

\[ R_{on} = \frac{\delta V_{DS}}{\delta I_{DS}} \bigg|_{V_{GS}} = \frac{1}{\mu C_{ox} \frac{W}{L} (|V_{GS}| - |V_{T}|)} \]

assuming the square law model for a MOSFET. It can therefore be assumed that \( R_{on} \) and \( \frac{1}{g_{m}} \) are of similar magnitude and \( R_{on} \ll R_{on} + \frac{1}{g_{m}} \). Therefore,

\[ R_{eq5} \approx r_{o5}(1 + g_{m5}R_{on,p}) \]
\[ R_{eq6} \approx r_{o6}(1 + g_{m6}R_{on,n}) \quad \text{(symmetry)} \]
\[ R_{out} = r_{o5}(1 + g_{m5}R_{on,p})||r_{o6}(1 + g_{m6}R_{on,n}) \]
\[ = \frac{1}{2}r_{o}(1 + g_{m}R_{on}) \]

Therefore, the voltage gain is \( A_{V} = (g_{m1} + g_{m2})(r_{o5}(1 + g_{m5}R_{on,p})||r_{o6}(1 + g_{m6}R_{on,n})) \) and the TIA gain is

\[ \frac{V_{out}}{I_{in}} \approx - \frac{2g_{m} \cdot \frac{1}{2}r_{o}(1 + g_{m}R_{on})}{1 + 2g_{m} \cdot \frac{1}{2}r_{o}(1 + g_{m}R_{on})}R_{f} = - \frac{g_{m}r_{o}(1 + g_{m}R_{on})}{1 + g_{m}r_{o}(1 + g_{m}R_{on})}R_{f} \]

**Noise analysis**

To calculate the noise current variance at the TIA output, the individual contributions from each equivalent noise generator shown in Figure 4.7 is considered using superposition.
Intuitively, it can be seen in Figure 4.7 that this circuit has twice the number of noise sources due to the two branches of the circuit in addition to the noise contributions from the tail transistors. Therefore, the noise spectral density at the output is expected to be significantly larger than that of the current TIA implementation ($2kT\gamma g_m r_o \Delta f$, calculated in Section 4.1). This will be verified in simulation in Section 4.3.

**Comparison of the two topologies**

Both topologies utilize current reuse and are able to leverage the transconductance of both the NMOS and PMOS inputs. The proposed TIA has a differential input, which allows for common mode noise cancellation. However, the output noise spectral density of the proposed topology is significantly larger than noise of the current topology. The proposed topology has a higher voltage gain due to its larger output resistance, so its TIA gain is closer to $-R_f$ as desired.

The tail NMOS and PMOS transistors limit the output voltage headroom, but since they can be operated in triode, the drain voltage can be close to the rail [1] and headroom is not affected significantly.

As the proposed topology has 4 more transistors than the current topology, each AFE will take up a larger area. Assuming the complementary transistors (M2, M3, M5, and M6) in the proposed topology have the same $g_m = \frac{2I_D}{V_T}$ as the input pair in the current topology,
the proposed topology would require twice as much current as the existing TIA due to its differential nature. However, the larger area and higher power consumption do not matter too much, as the AFE is remoted outside of the body where size and power consumption are less of a constraint.

The bias voltage in the proposed topology enables greater user control over the tail current and power consumption. Furthermore, the self-biased tail transistor M1 eliminates the need for a complex common mode feedback loop [3]. If the common mode input voltage range is limited, M4 can be self-biased as well by connecting its gate to the drains of M2 and M3 [1].

4.3 Simulation Results

Simulations were run to verify the gain and noise analyses from the previous section and help compare the relative performance of the two topologies.

Figure 4.8: Schematic of testbench with both TIA implementations (including DC operating points) with feedback resistor omitted.

Figure 4.8 includes a schematic of the testbench for both topologies. The noise contribution of the feedback resistor was neglected. The transistors were sized such that $g_{mp}$ and $r_{op}$ for the PMOS transistors in each complementary pair are the same and $g_{mn}$ and $r_{on}$ for the NMOS transistors in each complementary pair are the same across both circuits. The tail NMOS at the bottom was sized with twice the width of the complementary NMOS.
transistors and the tail PMOS at the top was sized with twice the width of the complementary PMOS transistors. The following parameters were used: \( V_{DD} = 1.2 \text{V}, \) \( V_b = 0.8 \text{V}, \) \( V_{ref} = 0.6 \text{V}, \) and \( f = 30 \text{MHz}, \) where \( f \) is the frequency of the sinusoidal voltage input.

![Figure 4.9: Noise simulation setup.](image)

(a) PSS simulation setup. (b) PNOISE simulation setup

Figure 4.9: Noise simulation setup.

Figure 4.9 displays the simulation setup for the noise analysis. A periodic steady state (PSS) simulation is run first to determine the periodic operating point and a PNOISE simulation is run subsequently to calculate the total noise.
(a) Integrated noise for current topology (V).  (b) Integrated noise for proposed topology (V).

Figure 4.11: Noise simulation results.

As shown in Figure 4.10, the voltage gain of the proposed TIA (≈ 5.45) is slightly higher than the gain of the existing TIA (≈ 5.2). This aligns with the calculations in Sections 4.1 and 4.2:

\[
\frac{A_{V,\text{proposed}}}{A_{V,\text{current}}} = 1 + g_m R_{on} > 1
\]

Figure 4.11 displays the integrated noise at the output for each TIA topology. As expected, the noise at the output of the proposed topology (10.397 mV) is significantly higher than the noise at the output of the current topology (6.9402 mV) by a factor of about 1.5.
Chapter 5

Conclusion and Future Work

Silicon MRRs are a promising alternative to established PMUT and CMUT based ultrasound imaging systems. By demonstrating proof of concept through measurement results and optimizing both the photonic layout and circuit design of the system, MRR-based ultrasound imaging presents a solution for the disadvantages of commercial systems.

An important next step in testing the current prototype chip is to transition to processing data in real-time on an FPGA as well as implement beamforming in order to generate real-time images (as opposed to using the script detailed in Figure 2.9). In terms of photonic layout, implementing a smaller size, lower power frontend for the sensing rings will help accommodate a 50 µm pitch between the MRRs and thus enable the layout demonstrated in Chapter 3. Furthermore, as the topology proposed in Chapter 4 does not appear significantly advantageous over the current inverter-based TIA due to its greater thermal noise, it is worth exploring alternate TIA implementations.
Bibliography


