Volume 6 | Spring 2025

IEEE Student Magazine

Works from Students at the University of Cincinnati

Contents

Foreword from Marc Cahay

Dr. Marc Cahay, *ECE Dept. Head*

IT is hard to believe that this is already the 6th edition of the IEEE Student magazine. I started this initiative after becoming Department Head of Electrical and Computer Science back in 2017. At the beginning students started working on articles under my supervision and a few Faculty in the department. Now, they have the ball rolling. They have all the templates for the articles and the entire magazine all figured out and they are running the show. There is still faculty feedback, but they deserved all the credits for soliciting the articles and getting this final version in print. I would like to thank all the contributors, the reviewers, and especially, Mettika Ukey, *B.S. Electrical Engineering, Class of 2027*, the IEEE Magazine Editor-In-Chief, for running the show and giving me the opportunity to write this foreword. Putting the final touch to the edition always comes at the end of the spring semester which is loaded with other events and commitments, and the release of the magazine is always a tour de force.

I hope you will enjoy the articles as much as I did. I will make sure to let as many people as I can about this wonderful issue. This IEEE Student magazine has been a great success. In the future, we need to make sure a similar magazine could be put together by all engineering student groups in the college. This would be a great advertisement about the strength of our educational and research programs in the College of Engineering and Applied Science at the University of Cincinnati.

I would greatly appreciate your feedback about the IEEE Student magazine. Please send me an email at marc.cahay@uc.edu.

Foreword from Mettika Ukey

Mettika Ukey, *B.S. Elec. Eng., Editor In Chief*

I AM honored to present the first issue of the magazine under my leadership as Editor-in-Chief. Being a part of this magazine for the past 2 years is one of the most rewarding experiences I have had throughout college, and I am eternally grateful for this opportunity.

This issue covers a diverse range of topics within the engineering department here at UC. From new breakthroughs in computer and biological research to hands-on projects reflecting the creativity and technical depth of our student community.

I'd like to thank each author and editor that helped with the creation of this magazine. Your efforts in this magazine helps keep curiosity alive at the University of Cincinnati. I'd also like to thank the readers of the IEEE Student Magazine for supporting this project. Without you, this magazine wouldn't have been possible!

As we continue to grow, I encourage every student reading this to get involved. If you have done something interesting during your time here at UC that you would like to share, please consider writing for the magazine! Additionally if you are a reader, editor or designer that would like to sharpen your skills, we would love to have you contribute to next year's edition. For more information, please email me at ukeymn@mail.uc.edu.

Special Thanks to the Editor

Ansh Mendiratta, *B.S., Comp. Sci., 2028*

The Process of Implementing an 8-Point FFT in Verilog

Forrest Bushstone, *BSc, Comp. Eng.,*

Abstract

The Fast Fourier Transform (FFT) is an efficient algorithm that transforms a function in the time domain to a function in the frequency domain. The FFT has applications in nearly all signal processing, with communications being the largest beneficiary of the algorithm. This paper walks through solving an 8-point FFT, allowing more easily the explanation of concepts such as the butterfly and twiddle factors, which can be extended to build an FFT of 2^n points. A 1.15.16 two's-complement fixed point format will be used to represent fractionals.

I. 8-Point FFT Solution

THE FFT is fundamentally a divide-and-conquer algorithm: at each stage, the algorithm divides the previous stage into two parts. The algorithm runs recursively until a function of N samples is broken down into a function with one sample, after which point the transform is considered completed. Since we divide the number of samples each function deals with at each stage by two every time, there are $\log_2(N)$ stages in the FFT (for an 8-point FFT, there are 3 stages, since $2^3 = 8$).

A. Twiddle Factor Computation

The first step towards implementing an FFT is to calculate the twiddle factors, the numerical coefficients used in conjunction with the smaller discrete Fourier transforms when computing the Fast Fourier Transform.

The equation is as follows (where k represents the frequency and N represents the number of points):

$$W_N^k = \exp(-2kj\pi/N)$$

The maximum number of twiddle factors in an FFT is $N/2$: values $k = 1$ to $(N/2) - 1$. Solving for all four of our twiddle factors, we get the following:

$$\begin{aligned} W_8^0 &= 1 \\ W_8^1 &= \frac{\sqrt{2}}{2} - i\frac{\sqrt{2}}{2} \\ W_8^2 &= -i \\ W_8^3 &= -\frac{\sqrt{2}}{2} - i\frac{\sqrt{2}}{2} \end{aligned}$$

where $i = \sqrt{-1}$.

B. Butterfly Operation: Stages

1) Stage 1: Consider a time-domain function $x(t)$ with n samples. Three stages detailed below are necessary to transform it into the frequency domain function X.

$$\begin{aligned} S_1(0) &= x(0) + x(4) \\ S_1(1) &= W_8^0(x(0) - x(4)) \\ S_1(2) &= x(1) + x(5) \\ S_1(3) &= W_8^1(x(1) - x(5)) \\ S_1(4) &= x(2) + x(6) \\ S_1(5) &= W_8^2(x(2) - x(6)) \\ S_1(6) &= x(3) + x(7) \\ S_1(7) &= W_8^2(x(3) - x(7)) \end{aligned}$$

2) Stage 2:

$$\begin{aligned} S_2(0) &= S_1(0) + S_1(4) \\ S_2(1) &= S_1(1) - S_1(5)) \\ S_2(2) &= W_8^0(S_1(0) + S_1(4)) \\ S_2(3) &= W_8^0(S_1(1) - S_1(5)) \\ S_2(4) &= S_1(2) + S_1(6) \\ S_2(5) &= S_1(3) + S_1(7) \\ S_2(6) &= W_8^2(S_1(2) - S_1(6)) \\ S_2(7) &= W_8^2(S_1(3) - S_1(7)) \end{aligned}$$

3) Stage 3:

$$\begin{aligned} X(0) = S_3(0) &= S_2(0) + S_2(4) \\ X(1) = S_3(1) &= S_2(1) + S_2(5) \\ X(2) = S_3(2) &= S_2(2) + S_2(6) \\ X(3) = S_3(3) &= S_2(3) + S_2(7) \\ X(4) = S_3(4) &= W_8^0(S_2(0) - S_2(4)) \\ X(5) = S_3(5) &= W_8^0(S_2(1) - S_2(5)) \\ X(6) = S_3(6) &= W_8^0(S_2(2) - S_2(6)) \\ X(7) = S_3(7) &= W_8^0(S_2(3) - S_2(7)) \end{aligned}$$

C. Implementation Details

Internally, each point is represented as a complex number that uses 32 bits for the real portion and 32 bits for the imaginary portion (64 bits total). The high 32 bits of each point stores the imaginary component, and the low 32 bits of each point stores the real component. Both the real and the imaginary components each use the twos-complement 16.16 fixed point format (16 bits for the integer part, and 16 bits for the fractional part).

One drawback of fixed point is that multiplication doubles the width of both the integer and fractional components. As it stands, this fact would require a reimplementation of the butterfly module to output 32.32 and 64.64 fixed point numbers. FPGAs have a limited amount of logic they can perform, and so to circumvent the reimplementation, we simply discard the high 16 bits of the integer and the low 16 bits of the fractional, reducing the data we need to store. We consider this loss of precision to be acceptable for this application.

D. Verilog Implementation

```
`timescale 1ns / 1ps
//////////////////////////////////////
// Engineer: Forrest Bushstone
// License: MIT
// Design name: fft8
// Description: An 8-point Fast Fourier Transform
// Design Notes:
//         * 16.16 twos-complement fixed point format is used for each point
//         * Each point is a 32-bit complex number- imag:real
// Warning: No mechanism is provided to prevent/detect overflow
//////////////////////////////////////

// Twiddle Factors
// 1
`define TWIDDLE_0_REAL 32'h0001_0000
// 0
`define TWIDDLE_0_IMAG 32'h0000_0000
// 0.707107
`define TWIDDLE_1_REAL 32'h0000_B505
// -j0.707107
`define TWIDDLE_1_IMAG 32'hFFFF_4AFA
// 0
`define TWIDDLE_2_REAL 32'h0000_0000
// -j
`define TWIDDLE_2_IMAG 32'hFFFF_0000
// -0.707107
`define TWIDDLE_3_REAL 32'hFFFF_4AFA
// -j0.707107
`define TWIDDLE_3_IMAG 32'hFFFF_4AFA

//butterfly unit naming convention: snbk, with n being the stage number and k being //
the butterfly number in the stage
// a and b are 64-bit internally should either be complex
module butterfly #(parameter twiddle_real = 32'h0001_0000,
                   parameter twiddle_imag = 32'h0000_0000)
                  (input rst,
                  input clk,
                  input [63:0] a,
                  input [63:0] b,
                  output reg [63:0] A,
                  output reg [63:0] B);

    // Used to evaluate the expression (twiddle_real + twiddle_imag) * (a + b)
    reg [63:0] B_poly_real, B_poly_imag, B_intermediate;
    // cannot use $signed on parameters
    reg [31:0] wr, wi;

    always @ (posedge rst, posedge clk) begin
        if (rst) begin
            A <= 64'h0000_0000_0000_0000;
            B <= 64'h0000_0000_0000_0000;
            B_poly_real <= 64'h0000_0000_0000_0000;
            B_poly_imag <= 64'h0000_0000_0000_0000;
        end
        else begin
            //Construct A
            A = a + b;
            B_intermediate = $signed(a) - $signed(b);
            wr = twiddle_real;
            wi = twiddle_imag;
```

```
            // Assuming B is a complex number, account for both the real and
            //     imaginary components
            // imag:real
            B_poly_real = $signed(wr) * $signed(B_intermediate[31:0]) -
                $signed(wi) * $signed(B_intermediate[63:32]);
            // real:imag
            B_poly_imag = $signed(wr) * $signed(B_intermediate[63:32]) +
                $signed(wi) * $signed(B_intermediate[31:0]);
            // Construct B
            B[63:48] = $signed(B_poly_imag[47:32]);
            B[47:32] = $signed(B_poly_imag[31:15]);
            B[31:16] = $signed(B_poly_real[47:32]);
            B[15:0] = $signed(B_poly_real[31:15]);
        end
    end
endmodule

module stage1(input rst,
              input clk,

              input [63:0] x0,
              input [63:0] x1,
              input [63:0] x2,
              input [63:0] x3,
              input [63:0] x4,
              input [63:0] x5,
              input [63:0] x6,
              input [63:0] x7,

              output [63:0] S1_0,
              output [63:0] S1_1,
              output [63:0] S1_2,
              output [63:0] S1_3,
              output [63:0] S1_4,
              output [63:0] S1_5,
              output [63:0] S1_6,
              output [63:0] S1_7);

    butterfly #(.twiddle_real(`TWIDDLE_0_REAL), .twiddle_imag(`TWIDDLE_0_IMAG))
        s1b0(.rst(rst), .clk(clk), .a(x0), .b(x4), .A(S1_0), .B(S1_1));
    butterfly #(.twiddle_real(`TWIDDLE_1_REAL), .twiddle_imag(`TWIDDLE_1_IMAG))
        s1b1(.rst(rst), .clk(clk), .a(x1), .b(x5), .A(S1_2), .B(S1_3));
    butterfly #(.twiddle_real(`TWIDDLE_2_REAL), .twiddle_imag(`TWIDDLE_2_IMAG))
        s1b2(.rst(rst), .clk(clk), .a(x2), .b(x6), .A(S1_4), .B(S1_5));
    butterfly #(.twiddle_real(`TWIDDLE_3_REAL), .twiddle_imag(`TWIDDLE_3_IMAG))
        s1b3(.rst(rst), .clk(clk), .a(x3), .b(x7), .A(S1_6), .B(S1_7));

endmodule

module stage2(input rst,
              input clk,

              input [63:0] S1_0,
              input [63:0] S1_1,
              input [63:0] S1_2,
              input [63:0] S1_3,
              input [63:0] S1_4,
              input [63:0] S1_5,
              input [63:0] S1_6,
              input [63:0] S1_7,

              output [63:0] S2_0,
              output [63:0] S2_1,
              output [63:0] S2_2,
              output [63:0] S2_3,
              output [63:0] S2_4,
              output [63:0] S2_5,
              output [63:0] S2_6,
              output [63:0] S2_7);
```

```
129
130     butterfly #(.twiddle_real(`TWIDDLE_0_REAL), .twiddle_imag(`TWIDDLE_0_IMAG))
131         s2b0(.rst(rst), .clk(clk), .a(S1_0), .b(S1_4), .A(S2_0), .B(S2_2));
132     butterfly #(.twiddle_real(`TWIDDLE_0_REAL), .twiddle_imag(`TWIDDLE_0_IMAG))
133         s2b1(.rst(rst), .clk(clk), .a(S1_1), .b(S1_5), .A(S2_1), .B(S2_3));
134     butterfly #(.twiddle_real(`TWIDDLE_2_REAL), .twiddle_imag(`TWIDDLE_2_IMAG))
135         s2b2(.rst(rst), .clk(clk), .a(S1_2), .b(S1_6), .A(S2_4), .B(S2_6));
136     butterfly #(.twiddle_real(`TWIDDLE_2_REAL), .twiddle_imag(`TWIDDLE_2_IMAG))
137         s2b3(.rst(rst), .clk(clk), .a(S1_3), .b(S1_7), .A(S2_5), .B(S2_7));
138
139 endmodule
140
141 module stage3(input rst,
142               input clk,
143
144               input [63:0] S2_0,
145               input [63:0] S2_1,
146               input [63:0] S2_2,
147               input [63:0] S2_3,
148               input [63:0] S2_4,
149               input [63:0] S2_5,
150               input [63:0] S2_6,
151               input [63:0] S2_7,
152
153               output [63:0] S3_0,
154               output [63:0] S3_1,
155               output [63:0] S3_2,
156               output [63:0] S3_3,
157               output [63:0] S3_4,
158               output [63:0] S3_5,
159               output [63:0] S3_6,
160               output [63:0] S3_7);
161
162     butterfly #(.twiddle_real(`TWIDDLE_0_REAL), .twiddle_imag(`TWIDDLE_0_IMAG))
163         s3b0(.rst(rst), .clk(clk), .a(S2_0), .b(S2_4), .A(S3_0), .B(S3_4));
164     butterfly #(.twiddle_real(`TWIDDLE_0_REAL), .twiddle_imag(`TWIDDLE_0_IMAG))
165         s3b1(.rst(rst), .clk(clk), .a(S2_1), .b(S2_5), .A(S3_1), .B(S3_5));
166     butterfly #(.twiddle_real(`TWIDDLE_0_REAL), .twiddle_imag(`TWIDDLE_0_IMAG))
167         s3b2(.rst(rst), .clk(clk), .a(S2_2), .b(S2_6), .A(S3_2), .B(S3_6));
168     butterfly #(.twiddle_real(`TWIDDLE_0_REAL), .twiddle_imag(`TWIDDLE_0_IMAG))
169         s3b3(.rst(rst), .clk(clk), .a(S2_3), .b(S2_7), .A(S3_3), .B(S3_7));
170
171 endmodule
172
173 module fft8(input rst,
174             input clk,
175             input [63:0] x0,
176             input [63:0] x1,
177             input [63:0] x2,
178             input [63:0] x3,
179             input [63:0] x4,
180             input [63:0] x5,
181             input [63:0] x6,
182             input [63:0] x7,
183
184             output [63:0] X0,
185             output [63:0] X1,
186             output [63:0] X2,
187             output [63:0] X3,
188             output [63:0] X4,
189             output [63:0] X5,
190             output [63:0] X6,
191             output [63:0] X7);
192
193     wire [63:0] S1_0, S1_1, S1_2, S1_3, S1_4, S1_5, S1_6, S1_7;
194     wire [63:0] S2_0, S2_1, S2_2, S2_3, S2_4, S2_5, S2_6, S2_7;
195
196     stage1 s1(.rst(rst),
197               .clk(clk),
```

```
                .x0(x0),
                .x1(x1),
                .x2(x2),
                .x3(x3),
                .x4(x4),
                .x5(x5),
                .x6(x6),
                .x7(x7),
                .S1_0(S1_0),
                .S1_1(S1_1),
                .S1_2(S1_2),
                .S1_3(S1_3),
                .S1_4(S1_4),
                .S1_5(S1_5),
                .S1_6(S1_6),
                .S1_7(S1_7));

    stage2 s2(.rst(rst),
                .clk(clk),
                .S1_0(S1_0),
                .S1_1(S1_1),
                .S1_2(S1_2),
                .S1_3(S1_3),
                .S1_4(S1_4),
                .S1_5(S1_5),
                .S1_6(S1_6),
                .S1_7(S1_7),
                .S2_0(S2_0),
                .S2_1(S2_1),
                .S2_2(S2_2),
                .S2_3(S2_3),
                .S2_4(S2_4),
                .S2_5(S2_5),
                .S2_6(S2_6),
                .S2_7(S2_7));

    stage3 s3(.rst(rst),
                .clk(clk),
                .S2_0(S2_0),
                .S2_1(S2_1),
                .S2_2(S2_2),
                .S2_3(S2_3),
                .S2_4(S2_4),
                .S2_5(S2_5),
                .S2_6(S2_6),
                .S2_7(S2_7),
                .S3_0(X0),
                .S3_1(X1),
                .S3_2(X2),
                .S3_3(X3),
                .S3_4(X4),
                .S3_5(X5),
                .S3_6(X6),
                .S3_7(X7));

endmodule
```

E. Verilog Testbench

```
`timescale 1ns / 1ps

module fft8_tb(output [31:0] X0r,
               output [31:0] X1r,
               output [31:0] X2r,
               output [31:0] X3r,
               output [31:0] X4r,
               output [31:0] X5r,
```

```
264                 output [31:0] X6r,
265                 output [31:0] X7r,
266
267                 output [31:0] X0i,
268                 output [31:0] X1i,
269                 output [31:0] X2i,
270                 output [31:0] X3i,
271                 output [31:0] X4i,
272                 output [31:0] X5i,
273                 output [31:0] X6i,
274                 output [31:0] X7i);
275     reg clk, rst;
276     reg [63:0] x0, x1, x2, x3, x4, x5, x6, x7;
277     wire [63:0] X0, X1, X2, X3, X4, X5, X6, X7;
278
279     initial begin
280         clk = 0;
281         rst = 0;
282
283         #5 rst = 1;
284         #10 rst = 0;
285         x0 = 64'h0000_0000_0004_192B;
286         x1 = 64'h0000_0000_0008_A0BC;
287         x2 = 64'h0000_0000_0003_57D2;
288         x3 = 64'h0000_0000_000F_B892;
289         x4 = 64'h0000_0000_0013_8243;
290         x5 = 64'h0000_0000_0007_1833;
291         x6 = 64'h0000_0000_0003_BF3D;
292         x7 = 64'h0000_0000_0009_4389;
293     end
294
295     always begin
296         #5 clk = ~clk;
297     end
298
299     fft8 fft_test_inst(.rst(rst), .clk(clk),
300          .x0(x0), .x1(x1), .x2(x2), .x3(x3), .x4(x4), .x5(x5), .x6(x6), .x7(x7),
301          .X0(X0), .X1(X1), .X2(X2), .X3(X3), .X4(X4), .X5(X5), .X6(X6), .X7(X7));
302
303     assign X0r = X0[31:0];
304     assign X1r = X1[31:0];
305     assign X2r = X2[31:0];
306     assign X3r = X3[31:0];
307     assign X4r = X4[31:0];
308     assign X5r = X5[31:0];
309     assign X6r = X6[31:0];
310     assign X7r = X7[31:0];
311
312     assign X0i = X0[63:32];
313     assign X1i = X1[63:32];
314     assign X2i = X2[63:32];
315     assign X3i = X3[63:32];
316     assign X4i = X4[63:32];
317     assign X5i = X5[63:32];
318     assign X6i = X6[63:32];
319     assign X7i = X7[63:32];
320
321 endmodule
```

The Making of PassBuddy

Hinna Parwez, *B.Sc, Comp. Sci.,*

***Abstract*—MakeUC is an annual hackathon organized by the University of Cincinnati's IEEE chapter each fall semester. This two-day event features 24 hours of hacking, where individuals and teams develop innovative projects in both software and hardware. Participants have the opportunity to attend workshops, network, and engage with sponsors. The event is held online and in person at the 1819 Innovation Hub.**

I. Introduction

OUR project, Pass Buddy, is a Flask-based password management application designed to enhance cybersecurity and user convenience by generating and storing passwords. The application ensures that passwords are strong and resistant to common attacks by cross-referencing them against a database of weak passwords. It features an email-based two-factor authentication (2FA) for added security and secure password storage to protect user data. Developed in 24 hours at MakeUC, Pass Buddy was designed with a focus on both security and ease of use.

II. Background

Weak passwords continue to be one of the leading causes of security breaches, with studies showing that 81% of hacking incidents involve compromised or weak credentials[1]. Despite increasing awareness of cybersecurity risks, many users continue to prioritize convenience over security, opting for easy-to-remember passwords or relying on auto-saved credentials on keychains[**RG**]. Keychains, while convenient, can be risky as they often unintentionally store sensitive information. If these keychains are compromised, hackers gain access to a wealth of personal data. For example, a study found that 40,880 mini-programs had inadvertently leaked their master keys, exposing users to a range of security breaches [2].

Pass Buddy addresses these issues by generating passwords based on user hobbies, ensuring that they are both secure and memorable. Unlike traditional keychain solutions which may inadvertently store weak credentials, Pass Buddy integrates robust security measures, including two-factor authentication and secure password storage, without sacrificing user experience.

III. PassBuddy & Objective

Pass Buddy was developed to help users generate and manage secure passwords in a personalized and user-friendly manner. The application focuses on mitigating the risks associated with weak or reused passwords, making password management easier and more engaging. Users can select their hobbies as the basis for generating passwords, ensuring that the passwords are memorable yet strong. The key objectives of the project were to:

- Assure strong password generation: implement a system that creates passwords that are both secure and personalized to the user's interests.
- Integrate two-factor authentication: Add an email-based 2FA mechanism to increase security.
- Secure password storage: use modern cryptographic techniques to store passwords securely.
- User-centered design: provide a simple and intuitive user interface while still providing robust security.

IV. Design

The system design centered around providing a secure and simple user experience. The application was built using Flask due to its lightweight nature and ease of deployment, making it an ideal choice for rapid development in a hackathon setting. However, setting up the Flask environment proved challenging due to the tight time frame, requiring careful attention to configuration and integration with required libraries.

A major design challenge was ensuring a balance between strong security measures and usability. We incorporated features such as two-factor authentication (2FA) and secure password storage while maintaining an intuitive user interface (UI). To strengthen password security, we used a public dataset from GitHub to identify weak passwords, ensuring that all generated passwords met security requirements and were resistant to common password attacks.

The user interface was designed with simplicity in mind. Due to the hackathon's time constraints, we focused on essential components, such as password generation and 2FA functionality. Future versions should also prioritize enhancing the UI/UX design to improve accessibility and user engagement.

V. Prototype & Testing

The development process followed an iterative prototyping approach, testing each feature as it was being developed. Key components of testing included:

- API testing: we used Postman to manually test the authentication endpoints and guarantee that the login, 2FA, and password generation functionalities were working correctly.
- Security Validation: passwords were hashed using bcrypt before being stored in the database. Additionally, we cross-referenced the generated passwords against a public dataset of commonly used passwords to verify they met security standards.
- Functional testing: real time debugging in Flask's built-in debug mode allowed us to quickly identify and resolve errors related to authentication, and password generation logic.

Project Structure

```
PassBuddyNEW/
├── app.py
├── auth/
│   └── auth.py
├── models/
│   └── models.py
├── static/
│   ├── styles.css
│   └── Pass Buddy.png
├── templates/
│   ├── login.html
│   ├── home.html
│   ├── register.html
│   ├── dashboard.html
│   └── verify.html
├── README.md
```

Fig. 1: Project Structure

VI. Improvements

Looking ahead, one key area for improvement is scalability. To address this, Pass Buddy could adopt a microservices architecture, breaking the application into smaller, more independent services that can be developed, deployed, and scaled individually. Each microservice could focus on a specific aspect of the app, such as user authentication, password generation, or data storage.

In addition, implementing a distributed cloud infrastructure would help further scale the application. By spreading services across multiple cloud servers, we can better manage load balancing and ensure high availability. This setup also enhances fault tolerance as the failure of one service will not impact the entire application. I would also explore integrating GraphQL, because it would provide more flexible and efficient data retrieval.

VII. Thoughts

The development of Pass Buddy provided an invaluable learning experience, especially in backend development and web security. Reflecting on the project, I now recognize many aspects that I would approach differently in the future. One key area I would prioritize more is scalability. Ensuring the application can handle increased traffic and growth from the outset is crucial, and I would allocate more time to addressing this in future projects.

In addition, I would plan more time to integrate UI/UX design. The actual functionality of the app was my priority during the 24-hour time constraint of the hackathon, but it is important to create a seamless and intuitive user experience. In future hackathons, I plan to structure our approach more efficiently, ensuring that my team spends less time brainstorming and conceptualizing and more time on actual development. This will allow us to maximize our time to code and refine the product. Ultimately, the experience of building Pass Buddy under such time pressure was rewarding. It was a unique opportunity to see what can be accomplished when working on a project in real time, and I look forward to applying these lessons to future hackathons.

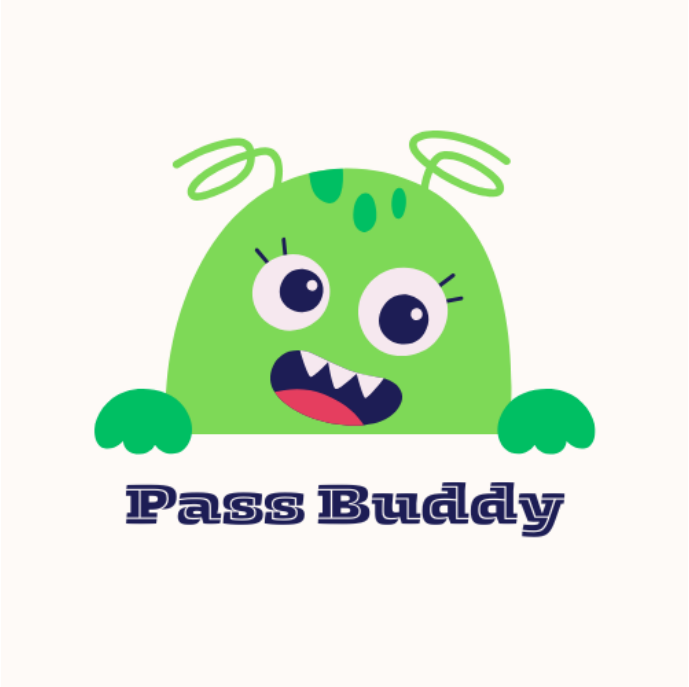


Fig. 2: Pass Buddy Logo

VIII. Conclusion

MakeUC provided a unique environment for rapid development, allowing for the quick creation of innovative projects under real-world constraints. While Pass Buddy has achieved its core objectives of secure password generation and storage, future work will focus on scalability, enhanced security features, and a more refined user interface. The hackathon experience was invaluable in demonstrating how quickly software solutions can be developed while maintaining a high standard of security and user experience.

References

[1] *Password policy: Updating your approach*. National Cyber Security Centre, Nov. 2018. URL: https://www.ncsc.gov.uk/collection/passwords/updating-your-approach.

[2] Zhiqiang Lin Yue Zhang Yuqing Yang. *Don't Leak Your Keys: Understanding, Measuring, and Exploiting the AppSecret Leaks in Mini-Programs*. Nov. 2023. URL: %5Curl%7Bhttps://dl.acm.org/doi/10.1145/3576915.3616591%7D.

Early Detection of Alzheimer's Disease Using Micro-Biochips

Somkene Egwudo, *BS, Biomed. Eng.,* Ally Daher *MS, Biommed. Eng.,* Colleen Arrasmith, *MS, Biomed. Eng.,* Anna Franchi, *BS, Elec. Eng.,* Isabella Keller, *MS, Elec. Eng*

***Abstract*—Alzheimer's Disease [AD] is a progressive neurodegenerative disease that is caused by widespread damage starting in the hippocampus and entorhinal cortex. This extensive damage stems from both neurofibrillary tangles, and neuritic plaques. As the disease progresses, it causes issues with memory, cognitive abilities and completing daily tasks. There is no cure for AD, the best course of action is early detection to allow the patient to establish a support system to maintain quality of life. Current testing methods are not targeted towards early detection. They are often only done once the patient has already begun to decline and is presenting symptoms[1]. This results in high costs for the patient and limits the amount of time the patient has to establish a support system after getting a diagnosis. The following reviews a novel lab-on-a-chip device that will utilize key AD progression biomarkers and Surface plasmon resonance technology to help diagnose a patient up to 8-10 years prior to the onset of symptoms[2].**

I. Introduction

ALZHEIMER'S disease is a neurodegenerative disease that is characterized by a gradual loss of nerve cells and brain tissue. Clinically, the disease presents itself as a decline in cognitive function and encompasses psychiatric disturbances. These psychiatric disturbances can present as depression, over-activity, and/or aggressive behavior[1][3].

This disease has a high mortality rate, with the number of deaths due to AD between 2000 and 2019 having increased by 145%. As of 2023, there is an estimated 6.7 million people in the United States that are living with AD[1]. There is no present cure or prevention for AD. For the roughly 7 million people that are living with AD in the United States, early detection is key in preventing and managing its progression[2][4].

Biomarkers are playing an increasingly crucial role in disease detection. They provide an insight into assessing both an individual's susceptibility to the disease and predicting the time of AD onset. Different studies have focused on various biofluids to isolate the biomarkers for detection. Cerebrospinal fluid, saliva, and blood have been in the forefront for AD biomarkers detection[5].

Microscopically, AD is characterized by both neuritic plaques and neurofibrillary tangles [NFT]. The NFT are fibrillar deposits of hyperphosphorylated tau protein that form inside the cells forming the fibers and tangles branching between the cells. Similarly, neuritic plaques are formed between the cells due to an abnormal concentration of Amyloid beta 42 [Aβ - 42]. Both the NFT and neuritic plaques result in synapse elimination and contribute to cell death. Synaptic dysfunction, inflammation and/or vascular dysregulation are other pathological events that have also been proven to provoke disease progression[5].

In this literature review, a novel lab-on-a-chip device is presented. This device would utilize exosome isolation to obtain the protein biomarkers related to AD progression. A binary count for 2 different proteins phosphorylated tau 181 [p-tau 181], and Aβ-42 is then determined using Surface plasmon resonance [SPR]. These counts are compared against a determined standard level and provide insight into whether or not the patient is trending towards an AD diagnosis.

II. Literature Review

There are many ways to evaluate an individual who is suspected of having AD. Many of these assessments are done in conjunction with one another to provide the diagnosis. Those assessments include a detailed medical history – to see if any relatives have had AD in the past – which is then used with physical and mental status examinations to see if there are any other symptoms the patient is exhibiting that could point to a different diagnosis than AD. The assessments also include basic labs, neuroimaging studies, and neuropsychological testing, which are done to test the patient's cognitive function. Examples of neuropsychological testing include testing the patient's reading, concentration, memory, learning, and fine motor skills.

The main techniques that will be discussed are positron emission tomography [PET], magnetic resonance imaging [MRI], cerebrospinal fluid [CSF] tests, and blood tests[6][7].

A. Pet Scans

A PET scan can assess the patient's brain for any abnormal accumulation of both Aβ and p-tau proteins. Those proteins accumulate in the brain when someone has AD and serve as a reliable biomarker for cognitive decline[6]. The p-tau protein has been shown to accumulate in the brain proportionally with exhibiting cognitive decline in AD patients, which is a benefit of tau PET scans[6]. However, amyloid PET scans have higher sensitivity than other types of PET scans for detecting AD, such as the tau PET scans. Approximately 70-90% of patients that meet the clinical criteria for AD have positive amyloid PET results, so visual interpretations of amyloid PET scans are the current standard in the clinical environment[8].

Another benefit of amyloid PET scans is that the Aβ protein accumulates in the brain prior to any clinically significant cognitive changes in the patient which can lead to an early diagnosis of AD. Unfortunately, PET scans are not

a routine neuroimaging technique, therefore, most patients exhibit symptoms of AD prior to any scans , so AD can be caught early in the patient; however, an issue with PET scans is that they are not a routine neuroimaging technique, so most people are already exhibiting symptoms of Alzheimer's disease prior to any scans being performed[6]. Amyloid PET imaging can precisely detect cerebral Aβ deposition with a good specificity for Alzheimer's disease. However, another drawback for PET scans is that an amyloid PET is unable to distinguish Alzheimer's disease from any other disorders that involve Aβ, such as dementia with Lewy bodies [DLB][8]. PET scans also have high costs associated with them due to the expensive equipment required[9].

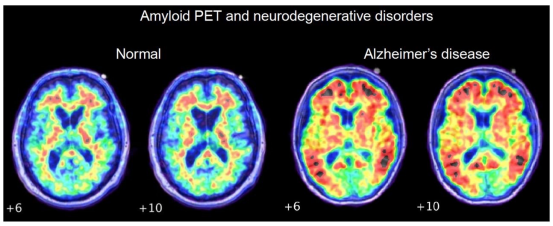


Fig. 1: Differences Between PET Scans of a Patient's Brain with Alzheimer's Disease and a Brain without Alzheimer's Disease[8].

B. MRI Scans

An MRI scan can also contribute to a patient's AD diagnosis. The MRI gives valuable visuals of the volume of specific areas of the brain. One specific area, the hippocampus, can undergo significant volume changes, and a smaller hippocampal volume is considered an important Alzheimer's structural marker[6]. The hippocampus reduces on average by 25% in volume with Alzheimer's patients[10]. Once taken, the MRIs can be assessed by radiologists or with an FDA-approved MRI volumetric data software package, such as Neuroreader® or NeuroQuant®.

Hippocampal atrophy is associated with cognitive decline from any disease, not just AD, and volumetric MRIs also have a limited sensitivity with diagnosing AD, so the MRIs can contribute to a diagnosis, but they must be used with other AD-specific biomarker tests to confirm a diagnosis[6][11][12]. However, volumetric MRI scans have good sensitivity and specificity for determining the specific type of dementia a patient has developed[11]. The hippocampal atrophy maps have 90% sensitivity and 84% specificity for AD when compared to other types of dementia, such as DLB[12]. Hippocampal volume atrophy is a well-known biomarker for memory impairment, but it is less specific for Alzheimer's disease compared with Aβ and tau imaging, so volumetric MRIs are not used as much in the field as PET scans[11].

C. Cerebrospinal Fluid Tests

Cerebrospinal fluid surrounds the brain and is obtained through a lumbar puncture, which many patients describe as a highly uncomfortable and invasive procedure. CSF testing

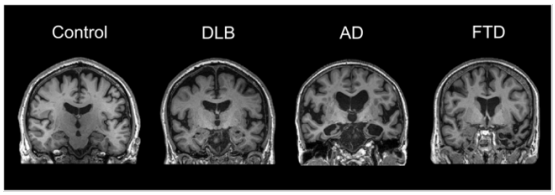


Fig. 2: MRI Scans from a Patient with Alzheimer's ("AD") Compared to the Control and Other Types of Dementia, such as Dementia with Lew Bodies ("DLB") and Frontotemporal Dementia ("FTD")[13].

for the p-tau protein has been shown to have a high sensitivity and specificity[6]. There currently are FDA-approved CSF immunoassays for Aβ-42/40 and p-tau181/Aβ-42, which compare the ratio of Aβ-42 to Aβ-40 and the ratio of p-tau181 to Aβ-42 in the CSF, respectively[9]. Patients with a lower Aβ-42/40 ratio or a higher p-tau181/Aβ-42 ratio show worse cognitive and functional prognosis[14].

The benefit of CSF tests is that the levels of both Aβ-42 and p-tau proteins change in the CSF decades before the onset of clinically significant Alzheimer's disease symptoms, which is earlier detection than provided by the amyloid PET scans. However, an issue with CSF testing is that it is also not a routine test, so most patients get it done while already exhibiting symptoms for Alzheimer's disease[6][8]. CSF tests also have high costs associated with them due to their perceived invasiveness and requirement of specially trained personnel[9].

D. Blood Tests

Blood is more accessible than CSF so there have been several blood tests developed for AD. Blood tests utilize mass spectrometry to detect the different concentrations of Aβ and p-tau proteins within the plasma[6]. The plasma assays that measure the levels of Aβ-42 and Aβ-40 show results of a lower Aβ-42/40 ratio consistent with amyloid PET and CSF tests[8].

Also, the increased plasma levels of p-tau in the p-tau181/Aβ-42 ratio are strongly associated with PET and CSF biomarkers and the eventual development of Alzheimer's disease[9]. These blood tests have 88% to 92% accuracy with predicting the patient receiving a future diagnosis of AD[15]. However, a current issue with blood tests is that they are not covered by insurance, so it might not be an accessible diagnostic route for some patients. Blood tests also are currently not a stand-alone diagnostic tool and need to be assessed with other diagnostic tools as well[6].

III. Methods/Materials

A. Plasma Separation

One of the most common ways to separate plasma from blood is through using a centrifuge. This is a method where solutions can be sorted into their components based on their densities[16]. According to Moller et al., the protocol of 5 minutes at 3000 x g with an Eppendorf 5810 R swing bucket centrifuge at room temperature (21°C) is suitable for plasma separation[17].

B. EV Isolation

In order to separate the extracellular vesicles [EV's] from the plasma sample, a di-electrophoretic [DEP] isolation process will be used. A non-uniform electric field will be applied along the length of a micropipette in order to isolate the EV's from the sample. The micropipette will be fabricated from a glass capillary utilizing a micropipette puller. Specifically, the capillary selected was BF100-50-15 type Borosilicate Glass and the Sutter Instrument's P-2000 puller was chosen [18].

To supply the voltage needed to create the non-uniform field along the micropipette a bench top power supply is necessary in order to carry out the experiment. The power supply chosen will be the Array Agilent 33210A Power Supply[19]. This was chosen as it has a high level of precision and will be able to supply the voltage and amperage needed. The figure below depicts the high electric field region generated at the tip of the pipette from the power supply in order to isolate the exosomes [20].

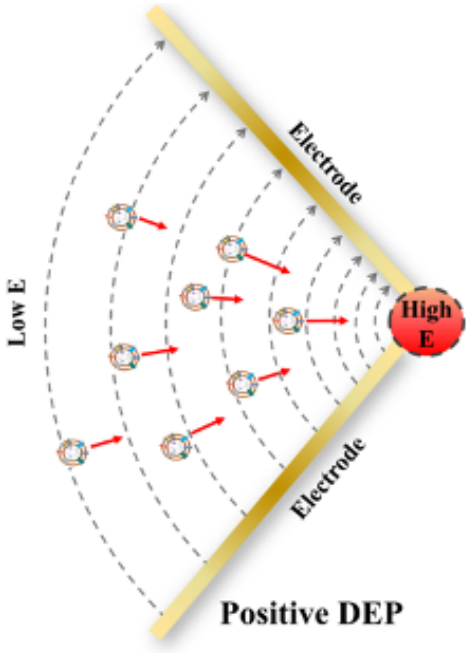


Fig. 3: A Depiction of Exosome Isolation using an Electric Field[20].

C. Protein Isolation and Quantification

In order to obtain the proteins from within the EVs, a lysing solution is required. Radioimmunoprecipitation assay [RIPA] was selected after reviewing study completed by Subedi et al. on the quantitative comparison of the proteins obtained from different lysing solutions on isolated EV's. The lysing solutions break down the cell membrane and release the proteins inside. The RIPA solution was selected as it did not denature the proteins and had the highest number of identified proteins and peptides from the isolated EV's[21].

Once the proteins have been isolated, they will then be quantified using SPR technology. On top of the thin gold sheet, from the SPR technology, will be 2 channels functionalized with appropriate aptamers for the specific proteins.

To detect Aβ-42, the first channel will be functionalized with c-abp2. This is a peptide aptamer that has a binding affinity of 35.80 ± 18.22 pM. This peptide aptamer has both high specificity and strong binding to ensure accurate detection[22]. The second channel will then be functionalized with an IT2 aptamer to detect p-tau 181. This single stranded DNA aptamer has an off rate of ((5.9 ± 1.2) [Equation] and a binding affinity of 5.5 ± 1.1 nM. Both the off rate and binding affinity ensure a strong and stable binding interaction between the protein and the aptamer probe[23].

IV. Theory

A. Exosome Isolation

A technique for exosome isolation is characterized by the balance of three forces, Di-electrophoresis [DEP], Electroosmosis [EOF] and Electrophoresis [EP]. The DEP force is the motion of a neutral or charged particle in a non-uniform electric field[20]. The EOF force involves the moment of a solution within a charged channel[24]. The EP force considers the induced movement of a charged particle within a solution in a non-uniform electric field[25].

As the non-uniform electric field is applied within the micro-pipette the exosomes are pulled to the tip of the pipette. With this motion the "trapping zone" is deemed at the point where the DEP and EOF forces are able to counterbalance the EP force, as represented in Figure 1. At this point, the exosome is deemed to be in a state of equilibrium, with no motion, and remains at the tip of the pipette[18].

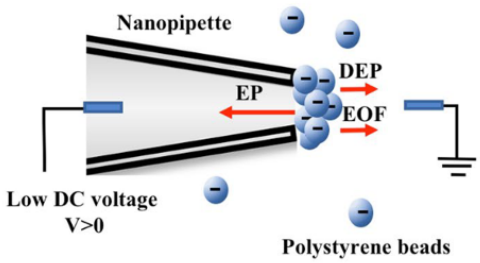


Fig. 4: Balance of Three Forces at the Pipette Tip [18].

Once the exosome isolation process has been run to trap the exosomes, the non-uniform electric field voltage polarity is reversed. By reversing the polarity, the exosomes are then sent out of the pipette and can be collected in a buffer solution for future analysis[18].

B. P-Tau181

P-tau 181 is a protein biomarker that has been previously used for both AD and mild cognitive impairment detections. An elevated level of p-tau 181 is correlated to the formation of NFT. The NFT causes disruption in the normal function of neurons which leads to neuronal damage and eventually cognitive decline. In a plasma sample, p-tau 181 is preferred largely for its dynamic range and sensitivity. Elevated p-tau

181 levels have been correlated with cognitive decline and has demonstrated specificity to AD diagnosis when compared to other neurodegenerative diseases[26].

To quantify the measured levels of p-tau 181, Karikari et al. reported three different groups, cognitively unimpaired [CU], Mild cognitive impairment [MCI], and AD patients. Mean values were reported for each group as 14.9 pg/mL, 18.3 pg/mL, and 23.6 pg/mL respectively, acting as the baseline for this report[27]. Previous studies have reported that the elevated p-tau 181 levels in a plasma sample is associated with hippocampal atrophy and gray matter loss related to neuronal loss from the disease[28].

C. Aβ-42

Amyloid β 42 [Aβ-42] in conjunction with Amyloid β 40 [Aβ-40] have been shown to accumulate farther into AD disease progression. When abnormal concentrations of this protein accumulate in the brain, it results in plaques forming between the cells. It is theorized that this plaque formation results in a cascade of events leading to further neuronal damage in AD. It has previously been reported that Aβ-42 is able to bi-directionally traverse the blood-brain barrier and in CSF decreased Aβ-42 levels have been correlated with a decrease in cognitive function.

This pattern has also been observed in plasma samples. Seppa¨la¨ et al. reported that, for a baseline, an average of 19 pg/mL was found in cognitively stable patients compared to an average of 12 pg/mL in patients who had cognitively declined 3 years after AD diagnosis. The measurements were then repeated 6 years after AD diagnosis, and an average of 10 pg/mL was reported compared to an average of 18 pg/mL for patients who remained cognitively stable. The inverse trend acts as a baseline for Aβ-42 for this report. Previous studies reported that when an abnormal amount of Aβ-42 builds up in the brain, less is able to leak out into the bloodstream through the blood brain barrier. This explains why there is a de-elevated amount of the protein found in a plasma sample from an AD patient[29].

D. Surface Plasmon Resonance

SPR is a spectroscopic technique for real time monitoring of analyte interactions on a biomolecular recognition element. As the p-tau 181 and Aβ-42 molecules bind to the aptamers on the substrate surface, light is passed through a prism. At the point where the light reaches the sensor surface, a small portion travels through to the gold film. Both light portions are then reflected back through the prism, as represented in Figure 2. The change in refraction angle and light intensity for both portions of the reflected light can then be quantitatively measured. The measured change is then used to quantitatively count the number of proteins that are bound to the aptamers on the sensor surface[30].

It has been reported that, typically, a SPR biosensor has a detection limit down to 10 pg/mL. Using SPR technology in conjunction with the specified aptamers will allow for accurate detection based on the reported baselines for elevated and de-elevated protein levels for p-tau 181 and Aβ-42 respectively[31].

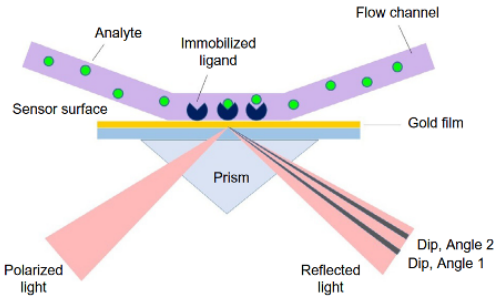


Fig. 5: An SPR Depiction to Measure the Change in Reflected Light Intensity[30].

V. Solution

A. Protein Isolation

The solution starts with extracting blood from a patient. This is done with a needle and syringe. The blood is transferred into smaller vials and placed in a centrifuge to separate the plasma from white blood cells, ficoll, and other liquids and macromolecules[32]. Because the process uses very little blood, a portion of the blood can be preserved upon the patient's request for future testing.

EV's within the plasma hold the proteins sought out for isolation. By applying a non-uniform electric field through the micropipette, as detailed in the theory section, the EV's can be trapped at the tip of the micropipette due to their unique dielectric properties. After trapping, the voltage polarity is reversed and the EV's can then be deposited into a buffer solution for further analysis.

Following the EV isolation, the EV's will be lysed. Radioimmune Precipitation Assay [RIPA] will be used to extract the desired proteins from the EV's. The process works by destroying the cell wall, releasing the molecules that are inside of the EV's. The RIPA solution was chosen because it is specific enough to destroy the cell wall without denaturing the proteins found inside. This was confirmed in a study completed by Subedi et al[21].

The proteins will then be transferred to an aqueous solution such as Phosphate buffered saline before getting transferred to the biosensor. The sensor will include 2 channels to provide a binary count for both Aβ-42 and p-tau 181. As the solution is sent through the channels, the proteins in the solution bind to the c-abp-2 and IT2 aptamers for Aβ-42 and p-tau 181 respectively. As the proteins are binding for a continuous measurement, it will be quantified using SPR[21][23]. This technique will provide a binary count for the amount of binding events and produce the data on a graph, as represented below in Figure 6[31].

B. Quantification

Using a microchip with SPR technology, proteins can be quantified, and a diagnosis can be made. SPR is a label-free technique used to quantify molecules such as proteins, lipids, and DNA markers[31]. Using a label-free detection process lowers the amount of time the process takes and decreases non-specific binding, increasing specificity.

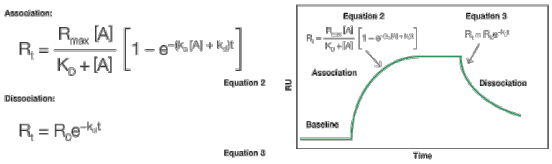


Fig. 6: SPR Graph Showing the Change in Binding Over Time[33].

SPR works by placing probes antithetic to the p-tau 181 and Aβ-42 proteins on a thin metal sheet, then energizing that sheet with light. A gold sheet is often used because of its common use as a detection instrument. We will use IT2 aptamers and c-abp2 aptamers for the probes because of their specificity and reversibility. The probes were also selected due to having the detection ability needed for the baseline ranges set to diagnose an AD patient from a cognitively unimpaired patient[22].

As the light is shown onto a prism, it excites the gold sheet behind it. When light hits the prism, it is reflected by a specific angle, called the SPR angle. The SPR angle represents the minimum observed reflectivity. The angle is changed due to the binding or unbinding action that is taking place on the gold film, as represented in Figure 7 below. This change can also be represented as a change in light intensity reflecting off the gold sheet. A decision to choose between measuring the light intensity or change in angle of the reflected light was made. Light intensity was chosen as the devices are relatively cheap and easy to integrate[34].

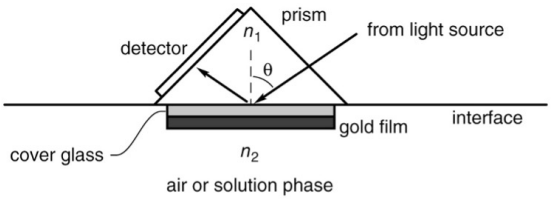


Fig. 7: . The Angle of Refraction for an SPR Sensor[34].

The use of aptamers with the SPR technology also provides an additional advantage over antibodies. Due to the reversibility of the aptamers, the sensor can be a continuous monitoring system. This allows for fast analysis times and reduces required sample volume. The aptamers also allow for a longer shelf-life, and less bath-to-batch variation. The aptamers are also both cheaper and easier to produce, making running the experiments easier. Lastly, the aptamers are smaller than antibodies, allowing them to detect molecules on the picomolar level, which is well within the range of detection for determining if a patient has AD or not[31].

VI. Gaps in the Proposed Solution

A. Potential to diagnose the wrong disease

It should be mentioned that there are some shortcomings with the proposed solution outlined in the previous section. For starters, biomarkers found in blood can indicate several different health conditions. An example of this is P-tau, a biomarker for several different cognitive conditions. When looking at p-tau specifically, one must ensure that the amount present is greatly increased up to 23.6 pg/mL. Although any amount of P-tau above 14.9 pg/mL is considered elevated, it is only an indication of AD specifically when it is at or above 23.6 pg/mL[27].

Aβ-42 is often elevated for different types of dementia. For example, vascular dementia can mimic AD[35]. While both diseases are a type of dementia, the causes are completely different, but they could still yield the same results.

B. EVs combined with other molecules

Another significant challenge is the presence of large amounts of soluble proteins and aggregates in bodily fluids, which can lead to contamination during EV isolation processes. This contamination may obscure the specific signals from EVs, making it difficult to accurately interpret results and potentially leading to misleading conclusions about disease states. By minimizing contamination, researchers can more reliably analyze the biomolecular cargo of EVs, such as RNA, proteins, and lipids, which are critical for understanding their role in intercellular communication and their potential as diagnostic biomarkers. This is particularly important in studying diseases like cancer or neurodegenerative disorders, where the precise composition of EVs could reveal early pathological changes or therapeutic targets.

Furthermore, the high selectivity of some isolation methods may result in the loss of potentially relevant EV populations, thereby limiting biomarker detection. These factors collectively reduce the reliability and reproducibility of EV-based biomarker studies, slow the development of standardized protocols, and improved isolation techniques to enhance their clinical usability[36].

C. Drawback of blood

Our solution is a blood test, but spinal fluid tends to yield better results. We have traded accuracy for patient comfort. The process of retrieving spinal fluid is invasive and painful. The process for retrieving spinal fluid involves a syringe with a long needle, about a few inches, inserted into the spine between disks. While both blood and spinal fluid are collected using a needle, the needle for spinal fluid is much larger and penetrates deeper into the body. This greatly increases discomfort.

The reason plasma is a worse option than CSF, is that it has less selectivity and sensitivity when reacting with the sensor. Selectivity is the of a biosensor's ability to distinguish between the target analyte and other substances. Sensitivity describes the ability of a biosensor to detect small changes in the concentration of the target analyte. Having a high selectivity and sensitivity means that the effectiveness of how the sensor performs is increased[9].

D. Complexity with steps

Another potential gap is that a multi-step process will inherently become more complicated. Using a large combination of complex steps and devices can lead to unforeseen

problems. When isolating extracellular vesicles, processes like differential centrifugation, density gradient separation, and filtration each introduce opportunities for variability and error. This can affect the purity, yield, or integrity of EVs, which are critical for downstream applications like biomarker diagnostics, or therapeutic use. Streamlining EV isolation methods or adopting integrated approaches, such as microfluidic devices or automated systems, could mitigate these issues and improve reproducibility[37].

E. Issues with new technology

There are risks that come with such a novel solution. The integration of biosensors with advanced chemistry techniques represents a new approach that has yet to be tested in a laboratory setting. Given its unprecedented nature, this new combination is likely to encounter some challenges and potential flaws during initial implementation. With any innovative method, it requires several adjustments and refinements before it can operate smoothly and effectively, highlighting the need for patience and thorough analysis in early trials. Because this novel process has never been officially tested before, some assumptions have been made. Based on the successes of the technology mentioned above, it is inferred that the different techniques will be compatible with each other.

Unfortunately, the field of microfluidic devices and biosensors is still in development and not easily available. Microfluidic devices, while promising for the isolation of extracellular vesicles, come with several disadvantages that can limit their widespread application. One significant challenge is the potential for channel blockage, which can occur when the analyzed sample contains particulates or aggregates, leading to reduced efficiency and reliability of the isolation process.

Additionally, the sample input and overall yield of EVs from these devices are often lower compared to traditional methods, which can hinder their diagnostic potential. The complexity of microfluidic systems may also require specialized knowledge for operation and maintenance, making them less accessible for routine use in many laboratories. Furthermore, the initial investment in microfluidic technology can be substantial, and ongoing development is needed to optimize these devices for various applications, which may not yet be fully realized in the current market.

F. High costs

The implementation of microfluidic devices for the isolation of EVs involves several cost considerations that can be a barrier to their widespread adoption. Initially, the investment for acquiring microfluidic systems can be significant, because devices often require advanced technology and specialized components, such as pumps, valves, and sensors, which can drive up costs.

Additionally, the development and fabrication of custom microfluidic chips may incur further expenses, particularly if specific designs are needed for medical applications. Beyond the initial purchase, there are ongoing costs associated with maintenance, calibration, and potential repairs of the equipment, which can add to the overall financial burden. Furthermore, researchers may need to invest in training personnel to effectively operate and troubleshoot these sophisticated systems, which can divert resources from other research activities. As a result, while microfluidic devices offer innovative solutions for EV isolation, the associated costs can pose challenges, specifically for smaller laboratories or private institutions with limited budgets[38].

VII. Conclusions/Recommendations

AD is a neurodegenerative disease that increasingly deteriorates brain tissue as the disease progresses. This disease is extremely detrimental to those who have the disease and those around them. There is currently no cure for AD, therefore, treatment is the only viable option for people with this disease. The earlier treatment can start, the more effective it will be, meaning that accurate testing is imperative. Without proper testing, people with AD can go years without a proper diagnosis and are many times only diagnosed after symptoms are already present[1][3].

The current solutions for diagnosing AD involve a multitude of tests involving physical and mental exams as well as brain imaging, CSF tests and blood tests. The most common tests for diagnosis are PET scans and MRI scans[6]. PET scans work by measuring the accumulation of Aβ-42 in the brain, whereas MRI scans work by determining volume decrease in the brain[6][8]. However, none of the current diagnostic tests are viable to be routine, meaning that symptoms are likely already present when they are performed[1]. Furthermore, neither of these scans are conducive for a stand-alone test for a diagnosis[6][13].

The device presented in this paper illustrates a solution for early AD diagnosis. The process consists of taking blood from a patient and separating out the plasma through a centrifuge. The EVs are then isolated, lysed, and both Aβ-42 and p-tau will be sent through their own channel so that they can be sensed. Then, using a SPR sensor, these biomarkers will be detected. The SPR sensor will contain aptamers that are specific to each biomarker and will measure the change in light intensity as binding to the sensor occurs.

Some proposed improvements for the device are to either improve the process or the specificity. The process proposed is a multi-step process which contains many of its steps outside of the sensor itself. A device that is able to perform more steps within the device would be more ideal. This could include moving more steps that are outside of the sensor onto the same platform. One idea was to build out a single platform that would include EV isolation, EV lysing and protein quantification. In doing so, this would make it easier for the technician running the experiments and reduce the risk for impurities being introduced in the sample.

Similarly, although the aptamers chosen are individually conducive probes for Aβ-42 and p-tau, it is possible there are more specific and sensitive aptamers that could be more effective for this implementation. Additionally, by reducing the EV isolation sample to only the neuronal derived EV's, it can help improve the specificity of the test. This would include another isolation step and would need to be included onto the device on the single platform.

Moreover, this device has yet to be tested for accuracy and precision. Once testing has been completed, the advantages and disadvantages of CSF should be weighed. Although our sensor aims to improve the specificity and sensitivity of blood testing, CSF, accessed through a lumbar puncture, is inherently more specific and sensitive. The efficacy of the proposed design should be weighed against the accessibility of the fluid used for measurement.

REFERENCES

[1] Tianchi Zhuang et al. "Novel plasma protein biomarkers: A time-dependent predictive model for Alzheimer's disease". en. In: *Arch. Gerontol. Geriatr.* 129.105650 (Feb. 2025), p. 105650.

[2] Jill Rasmussen and Haya Langerman. "Alzheimer's disease - why we need early diagnosis". en. In: *Degener. Neurol. Neuromuscul. Dis.* 9 (Dec. 2019), pp. 123–130.

[3] Weili Cao et al. "Comparison of the efficacy of updated drugs for the treatment on the improvement of cognitive function in patients with Alzheimer 's disease: A systematic review and network meta- analysis". en. In: *Neuroscience* 565 (Jan. 2025), pp. 29–39.

[4] Zeinab Breijyeh and Rafik Karaman. "Comprehensive review on Alzheimer's disease: Causes and treatment". en. In: *Molecules* 25.24 (Dec. 2020), p. 5789.

[5] Manuel H Janeiro et al. "Biomarkers in Alzheimer's disease". en. In: *Adv. Lab. Med.* 2.1 (Mar. 2021), pp. 27–50.

[6] Emily Bomasang-Layno and Rachel Bronsther. "Diagnosis and treatment of Alzheimer's disease:: An update". en. In: *Dela. J. Public Health* 7.4 (Sept. 2021), pp. 74–85.

[7] Grace J Lee and Julie A Suhr. "Principles and practices of neuropsychological assessment". In: *Comprehensive Clinical Psychology*. Elsevier, 2022, pp. 167–178.

[8] Marianne Chapleau et al. "The role of amyloid PET in imaging neurodegenerative disorders: A review". en. In: *J. Nucl. Med.* 63.Suppl 1 (June 2022), 13S–19S.

[9] Nicolas R Barthélemy et al. "Highly accurate blood test for Alzheimer's disease is similar or superior to clinical cerebrospinal fluid tests". en. In: *Nat. Med.* 30.4 (Apr. 2024), pp. 1085–1095.

[10] Avinash Vijayakumar and Abhishek Vijayakumar. "Comparison of hippocampal volume in dementia subtypes". en. In: *ISRN Radiol.* 2013 (2013), p. 174524.

[11] Bernard J Hanseeuw et al. "Association of pathologic and volumetric biomarker changes with cognitive decline in clinically normal adults". en. In: *Neurology* 101.24 (Dec. 2023), e2533–e2544.

[12] Y Lakshmisha Rao et al. "Hippocampus and its involvement in Alzheimer's disease: a review". en. In: *3 Biotech* 12.2 (Feb. 2022), p. 55.

[13] Leonidas Chouliaras and John T O'Brien. "The use of neuroimaging techniques in the early and differential diagnosis of dementia". en. In: *Mol. Psychiatry* 28.10 (Oct. 2023), pp. 4084–4097.

[14] Constance Delaby et al. "The Aβ1-42/Aβ1-40 ratio in CSF is more strongly associated to tau markers and clinical progression than Aβ1-42 alone". en. In: *Alzheimers. Res. Ther.* 14.1 (Feb. 2022), p. 20.

[15] Sebastian Palmqvist et al. "Blood biomarkers to detect Alzheimer disease in primary care and secondary care". en. In: *JAMA* 332.15 (Oct. 2024), pp. 1245–1257.

[16] Giuseppe Lippi et al. "Influence of the centrifuge time of primary plasma tubes on routine coagulation testing". en. In: *Blood Coagul. Fibrinolysis* 18.5 (July 2007), pp. 525–528.

[17] Mette F Møller et al. "Evaluation of a reduced centrifugation time and higher centrifugal force on various general chemistry and immunochemistry analytes in plasma and serum". en. In: *Ann. Clin. Biochem.* 54.5 (Sept. 2017), pp. 593–600.

[18] Leilei Shi, Ankit Rana, and Leyla Esfandiari. "A low voltage nanopipette dielectrophoretic device for rapid entrapment of nanoparticles and exosomes extracted from plasma of healthy donors". en. In: *Sci. Rep.* 8.1 (Apr. 2018).

[19] Haoqing Zhang, Honglong Chang, and Pavel Neuzil. "DEP-on-a-chip: Dielectrophoresis applied to microfluidic platforms". en. In: *Micromachines (Basel)* 10.6 (June 2019), p. 423.

[20] Mei Lan and Fang Yang. "Applications of dielectrophoresis in microfluidic-based exosome separation and detection". en. In: *Chem. Eng. J.* 491.152067 (July 2024), p. 152067.

[21] Prabal Subedi et al. "Comparison of methods to isolate proteins from extracellular vesicles for mass spectrometry-based proteomic analyses". en. In: *Anal. Biochem.* 584.113390 (Nov. 2019), p. 113390.

[22] Yan Zheng et al. "Advances in aptamers against Aβ and applications in Aβ detection and regulation for Alzheimer's disease". en. In: *Theranostics* 12.5 (Jan. 2022), pp. 2095–2114.

[23] I-Ting Teng et al. "Identification and characterization of DNA aptamers specific for phosphorylation epitopes of tau protein". en. In: *J. Am. Chem. Soc.* 140.43 (Oct. 2018), pp. 14314–14323.

[24] Jean Aupiais and Frédéric Chartier. "Capillary electrophoresis". In: *Sample Introduction Systems in ICPMS and ICPOES*. Elsevier, 2020, pp. 299–356.

[25] Aditya S Khair. "Nonlinear electrophoresis of colloidal particles". en. In: *Curr. Opin. Colloid Interface Sci.* 59.101587 (June 2022), p. 101587.

[26] Giulia Giacomucci et al. "Plasma p-tau181 as a promising non-invasive biomarker of Alzheimer's Disease pathology in Subjective Cognitive Decline and Mild Cognitive Impairment". en. In: *J. Neurol. Sci.* 453.120805 (Oct. 2023), p. 120805.

[27] Thomas K Karikari et al. "Diagnostic performance and prediction of clinical progression of plasma phospho-tau181 in the Alzheimer's Disease Neuroimaging Initiative". en. In: *Mol. Psychiatry* 26.2 (Feb. 2021), pp. 429–442.

[28] Yan-Li Wang et al. “Plasma p-tau181 level predicts neurodegeneration and progression to Alzheimer’s dementia: A longitudinal study”. en. In: *Front. Neurol.* 12 (Sept. 2021), p. 695696.

[29] T T Seppälä et al. “Plasma Abeta42 and Abeta40 as markers of cognitive change in follow-up: a prospective, longitudinal, population-based cohort study”. en. In: *J. Neurol. Neurosurg. Psychiatry* 81.10 (Oct. 2010), pp. 1123–1127.

[30] Dennis G Drescher, Dakshnamurthy Selvakumar, and Marian J Drescher. “Analysis of protein interactions by surface plasmon resonance”. In: *Protein-Protein Interactions in Human Disease, Part A*. Advances in protein chemistry and structural biology. Elsevier, 2018, pp. 1–30.

[31] Hoang Hiep Nguyen et al. “Surface plasmon resonance: a versatile technique for biosensor applications”. en. In: *Sensors (Basel)* 15.5 (May 2015), pp. 10481–10510.

[32] Douglas Silva de Vasconcellos. “Monolithic Integration of Multiple Porous Silicon Membranes for Lab-on-a-chip Applications”. Theses. Université Paul Sabatier - Toulouse III, Sept. 2020. URL: https://laas.hal.science/tel-03131243.

[33] Vesna Hodnik and Gregor Anderluh. “Toxin detection by surface plasmon resonance”. en. In: *Sensors (Basel)* 9.3 (Feb. 2009), pp. 1339–1354.

[34] Yijun Tang, Xiangqun Zeng, and Jennifer Liang. “Surface plasmon resonance: An introduction to a surface spectroscopy technique”. en. In: *J. Chem. Educ.* 87.7 (July 2010), pp. 742–746.

[35] C C Price et al. “Subcortical vascular dementia: integrating neuropsychological and neuroradiologic data”. en. In: *Neurology* 65.3 (Aug. 2005), pp. 376–382.

[36] Maria Yu Konoshenko et al. “Isolation of extracellular vesicles: General methodologies and latest trends”. en. In: *Biomed Res. Int.* 2018 (Jan. 2018), p. 8545347.

[37] Jonathan M Carnino, Heedoo Lee, and Yang Jin. “Isolation and characterization of extracellular vesicles from Broncho-alveolar lavage fluid: a review and comparison of different methods”. en. In: *Respir. Res.* 20.1 (Oct. 2019), p. 240.

[38] Farshid Jaberi Ansari et al. “Comparison of the efficiency of ultrafiltration, precipitation, and ultracentrifugation methods for exosome isolation”. en. In: *Biochem. Biophys. Rep.* 38.101668 (July 2024), p. 101668.

Ferroelectric Gated Diode Spiking Neural Network With Tunable Membrane Potential

Nicholas Haehn, *MSc, Elec. Eng.*, Sam Burkhard, *MSc, Comp. Eng.*, Andrew Jones, *Ph.D., Elec. Sci.*

***Abstract*—The Leaky Integrate-and-Fire (LIF) circuit is meant to model the performance of a biological neural network within a spiking neural network. Typically, the LIF circuit is modeled with a fixed capacitor and resistor. However, true biological networks utilize a tunable membrane potential. In this work, a ferroelectric gated diode (FeGD) is proposed to replace the capacitor in the LIF circuit to allow for a tunable membrane potential through the FeGD's variable capacitance. This circuit was demonstrated to have a different firing rate based on polarization. Additionally, the circuit architecture and crossbars were designed and simulated to show how a full implementation can be designed. Finally, the tuneable membrane potential was modeled within SNNTorch, demonstrating improved performance with a tunable membrane potential.**

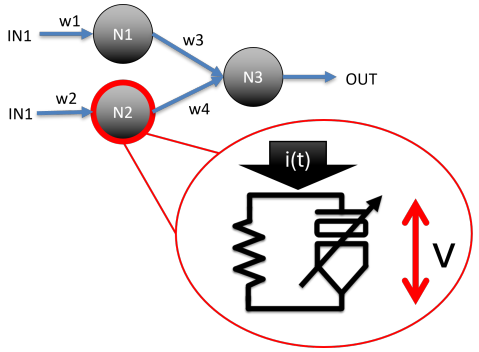


Fig. 1: Ferroelectric Gated Diode Leaky Integrate and Fire Diagram

I. Introduction

Deep Neural Networks (DNNs) are continually being developed to solve complex problems in numerous fields. However, as complexity and scale increase, traditional Artificial Neural Network (ANN) architectures suffer from issues in efficiency and power [1]. Spiking Neural Networks (SNNs) utilize a more energy-efficient approach through being event-driven. SNNs are biologically inspired, where spikes are used to encode and communicate information between neurons. In this way, SNN architectures are only active when information is being processed on that node [1, 2].

In biological neural networks, biological neurons utilize action potentials to transmit information to other neurons. When a neuron receives enough external stimuli, the membrane potential of the neuron is met, and the neuron "fires". Once fired, the neuron undergoes a refractory period in which it transitions back to a resting state before being ready to fire again [2]. Similarly, SNNs send spikes between neurons until a given membrane potential is met before the post-synaptic neuron fires [3].

Based on neuron dynamics discovered in the early 1900s, neurons sustain their membrane potential over a period of time after receiving temporal data [3]. Therefore, the neuron can be represented as a parallel resistor and capacitor similar to a low-pass filter circuit labeled as a leaky integrate-and-fire (LIF) neuron [3]. The capacitor integrates the charge of the incoming spikes, which leaks through the resistor [4]. The firing element can be implemented in various ways, but generally relies on the membrane threshold being met before turning on the firing element. This rapidly discharges the capacitor in the LIF circuit until almost fully discharged, generating an output spike as a result [4].

However, the membrane time constant τ is adaptive in biological neural networks, which has been observed in the visual cortex [1]. τ is distinct in different datasets even in the same architecture, demonstrating that τ should be tunable in any practical application. However, most applications fix τ [1]. [1] proposes a brain-inspired adaptive LIF neuron that can adjust the membrane time constant according to its inputs, allowing the prioritization of information based on the inputs and reduced manual initialization error in τ. The time constant is determined via a map that is the same size as the input layer [1]. [2] proposes achieving the tunable membrane potential via a memristor in place of the resistor in the LIF model, allowing the leakage to be adjusted [2].

The Ferroelectric Field Effect Transistor (FeFET) is a device that utilizes a ferroelectric material in the gate stack of the transistor [5]. This ferroelectric material allows the threshold voltage of the FeFET to be modified [5]. The ferroelectric gated diode (FeGD) utilizes a base FeFET structure to modify the MOSFET's capacitance curves [6]. Although demonstrated for memory applications, this device also shows promise in reducing the capacitance for LIF functions.

This work proposes expanding the membrane potential tunability by introducing the FeGD into the LIF circuit in place of the capacitor, as shown in Figure 1. The polarization in the ferroelectric allows the CV curve to be shifted, changing the effective capacitance in the LIF circuit. This offers enhanced capabilities, as both the capacitor and resistor values have different effects on the charging of the LIF.

This work discusses the development of the single neuron model of the FeGD LIF. In addition, the circuit implementation of an array of these neurons is discussed, as well as a system-based demonstration of the benefits of the tunable membrane. First, the methods are covered, then the results, and finally some discussion.

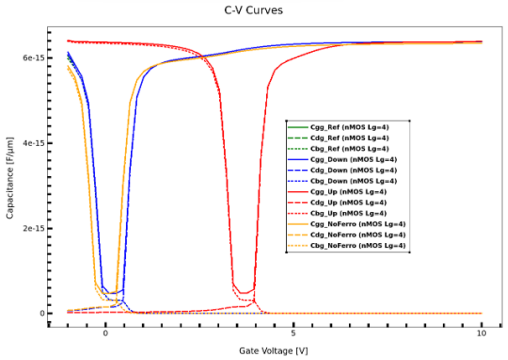


Fig. 2: Ferroelectric Gated Diode Capacitance versus Voltage (CV) curves for various polarization states. While the Reference and Down Polarized devices and the device with no ferroelectric have similar CV curves, the Up Polarization CV curve is shifted to the right, leading to a large capacitance at 0 V compared to the other polarization states.

II. Methodology

A. Device

The FeGD device was designed and simulated within Sentaurus TCAD. The device used for the majority of simulations was a lateral FeFET structure with a gate length of 4 µm and oxide thickness of 0.05 µm. The device was pre-polarized at various states, with reference polarization at 0 V before any bias, Down polarization at 5 V, and Up polarization at -5 V.

Figure 2 shows the capacitance shift for different polarization states in the FeGD. This graph demonstrates that at 0 V in the Down and Up polarizations, the capacitances are roughly 5.77×10^{-16} F and 6.37×10^{15} F, respectively. The LIF model and input currents were adjusted to give the best response near these two capacitances.

With these bulk capacitances, LT Spice was first used to model the current input to determine the charging and firing characteristics. Bulk capacitors and resistors were used to model the LIF circuit, while a pulsed current input of 4×10^{-6} A was added as the input. A voltage-controlled switch was used as the firing mechanism. The voltage-controlled switch turns on at $V_t + V_h$ and turns off at $V_t - V_h$. For most simulations, the switch was set to turn on at 0.7 V and off around 0.1 V, but this was adjusted throughout testing. The switch was configured to an OFF resistance of 1 MΩ and ON resistance of 5 kΩ. The LIF resistor value was set to 1 MΩ.

In TCAD, the FeGD component was placed into a mixed-mode spice simulation. The TCAD devices were defined in the mixed-mode model and instantiated, with the Up Polarization device eventually getting individually defined to better match initial conditions and improve convergence. The FeGD replaced the capacitor via connecting the gate to the storage node and the substrate and source to ground, leaving the drain floating. The resistor, voltage-controlled switch, and current source were the same in the TCAD simulation as in the LT Spice simulation. In early simulations, the Blocked method solver was utilized, while later tests utilized the ParDiso method with Blocked sub-method.

B. System

The SNN was built and trained using SNNTorch, a PyTorch library that extends it with spiking neuron models and other SNN utilities [3].

The MNIST dataset of 28×28 handwritten digits was used for all experiments [7]. Images were normalized to the $[0,1]$ range and converted to spike trains with SNNTorch's rate encoding tool. SNNTorch's built-in LIF neuron model was used, which allows for configuration of the decay rate, or β.

The β parameter is the ratio of membrane potential kept between time steps [3]. This variable approximates the behavior of the membrane time constant τ in the LIF neuron. SNNTorch defines the β value from the τ as shown in Equation 1, where Δt is a time step [3]. Comparing two plausible τ values from the FeGD on a real SNN by deriving their β values would demonstrate the efficacy of differing τ values.

$$\beta = \exp\left(-\frac{\Delta t}{\tau}\right) \tag{1}$$

Using plausible τ values from the FeGD gives two example β values used for network performance comparisons: $\beta_1 = 0.4999531419$ and $\beta_2 = 0.9391365887$.

The SNN consisted of three fully-connected layers:

1) **Input Layer:** 784 units for a flattened 28×28 input image.
2) **Hidden Layer:** 1000 LIF neurons.
3) **Output Layer:** 10 LIF neurons.

Weights were clipped between $[-1,1]$ and quantized into 11 evenly spaced values in intervals of 0.2 to match the hardware constraints of the memristor crossbar array.

To maximize weight assignments across the 11 possible values, weights were allowed to train freely for a full epoch without quantization, and then rescaled by the maximum value across the $[-1,1]$ range. The quantization was performed as follows:

$$X_{quant} = \frac{\lfloor X\times5\rceil}{5}$$

Where X is the weight before quantization and X_{quant} is the quantized weight.

The model was trained for at least 7 epochs with Adam optimizer learning rate set to 2×10^{-3}. The arc-tangent $(\arctan)$ function was employed as the surrogate gradient during backpropagation to address the non-differentiable nature of spiking neuron activations alongside the Cross Entropy Spike Count Loss function [3].

C. Memristor Crossbar Array

The architecture consisted of two primary components or "blocks" — the memristor crossbar array and the LIF neuron block. The function of the memristor CBA was to implement the weights obtained during training, taking advantage of the tunable nature of memristors to adjust the conductance of each device, allowing weights to be updated as needed. The method of assigning weights begins with a weight matrix containing 11 possible quantized values ranging from 1 to -1, obtained from training as shown below:

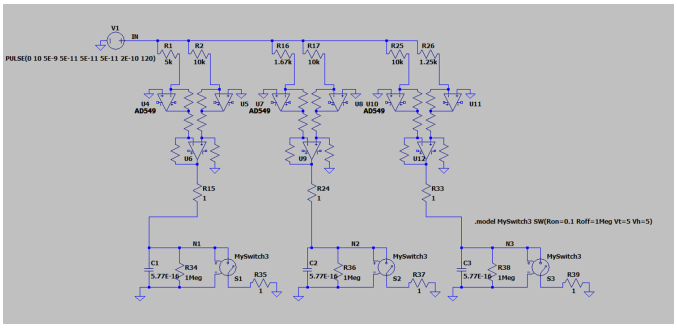


Fig. 3: Example implementation of cross bar array for assigning weights to differential memristors, TIA circuit, and FeGD neuron connections

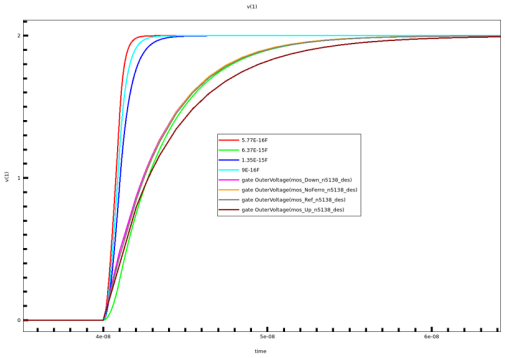


Fig. 4: Constant current response for bulk capacitors and for various polarization states of the Ferroelectric gated diode.

$$W_M = \begin{bmatrix} 0.2 & 1 & -0.8 & \dots & 1 \\ 0.4 & -0.6 & -1 & \dots & 0 \\ 1 & -1 & 0 & \dots & 0.2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -0.2 & 0.6 & -0.4 & \dots & -1 \end{bmatrix}$$

This weight matrix can be expressed as the difference between a matrix containing the positive weights and a matrix containing the negative weights, as shown below:

$$W_M^+ - W_M^- = \begin{bmatrix} 0.2 & 1 & 0 & \dots & 1 \\ 0.4 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0.2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0.6 & 0 & \dots & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0.8 & \dots & 0 \\ 0 & 0.6 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.2 & 0 & 0.4 & \dots & 1 \end{bmatrix}$$

Finally, a conductance value can be assigned to each weight, between G_{LRS} and G_{HRS} of the memristor. Because we are using 11 weights, there must be 5 conductances besides G_{HRS}. For our implementation, we use $0.2{\cdot}G_{LRS}$, $0.4{\cdot}G_{LRS}$, $0.6{\cdot}G_{LRS}$, $0.8{\cdot}G_{LRS}$, and G_{LRS}. The conductance matrix for the previous weight matrix examples would appear as follows:

$$G_M^+ - G_M^- = \begin{bmatrix} 0.2 \cdot G_{LRS} & G_{LRS} & G_{HRS} & \dots & G_{LRS} \\ 0.4 \cdot G_{LRS} & G_{HRS} & G_{HRS} & \dots & G_{HRS} \\ G_{LRS} & G_{HRS} & G_{HRS} & \dots & 0.2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ G_{HRS} & 0.6 \cdot G_{LRS} & G_{HRS} & \dots & G_{HRS} \end{bmatrix} -$$

$$\begin{bmatrix} G_{HRS} & G_{HRS} & 0.8 \cdot G_{LRS} & \dots & G_{HRS} \\ G_{HRS} & 0.6 \cdot G_{LRS} & G_{LRS} & \dots & G_{HRS} \\ G_{HRS} & G_{LRS} & G_{HRS} & \dots & G_{HRS} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.2 \cdot G_{LRS} & G_{HRS} & 0.4 \cdot G_{LRS} & \dots & G_{LRS} \end{bmatrix}$$

The memristors would be programmed accordingly and set up in a modified crossbar array structure with two memristors per input per neuron: [8] one to implement the positive weight and one to implement the negative. The output of these memristors is fed into a transimpedence amplifier implemented using op-amps, and sent to a certain neuron in the neuron block.

III. Results and Discussion

A. Device

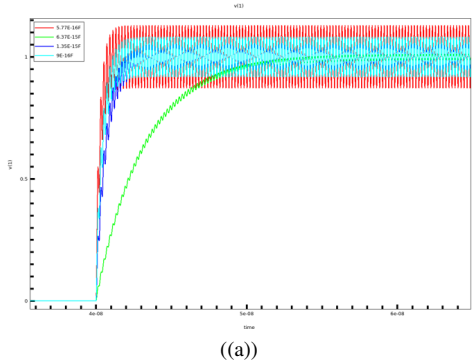
On the device side, first a constant current was fed into the LIF circuit to determine the effect of the FeGD on standard charging. The mathematical model for the standard LIF charging can be represented by $V(t) = V_0 \exp\left(-\frac{t}{RC}\right) + IR(1 - \exp\left(-\frac{t}{RC}\right))$, or simply $V(t) = IR(1 - \exp\left(-\frac{t}{RC}\right))$ if the capacitor is initially discharged. Therefore, charging will stabilize at $V = IR$. This is observed in all devices, both bulk capacitors and the FeGD, shown in Figure 4. However, the FeGD response is closest to the bulk 6.37×10^{-15} F capacitor when polarized in the down or reference. The Up polarization appeared to have an even greater equivalent capacitance than 6.37×10^{-15} F. This demonstrates that the CV characterization shown in Figure 2 does not match the capacitance values completely in practice. However, the difference in charging rate between the two polarizations still exists, showing the tunability in membrane potential.

The Up Polarization device was modified in this testing to better solve for its initial conditions to improve convergence. This involved creating a separate device definition and changing it to match the initial voltage conditions from polarization. Additionally, a longer rest period before the input pulse was added to the simulation to remove transient conditions from loading the device polarizations.

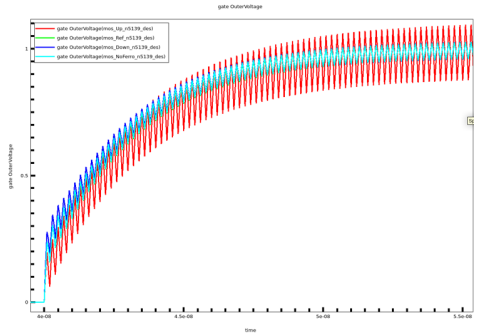
Next, a consistent pulse was applied to the bulk devices and to the FeGD LIF circuits without firing activated as shown in Figure 5. The curves with the pulsed input follow similar shapes to the constant current inputs, but are at a smaller magnitude due to the discharging. Discharging is governed by $V(t) = V_0 \exp\left(-\frac{t}{RC}\right)$. Therefore, the steady state voltage once discharging and charging match can be calculated.

$$V_{cc} = IR\left(\frac{1 - \exp\left(-\frac{t_1}{RC}\right)}{1 - \exp\left(-\frac{(t_1 + t_2)}{RC}\right)}\right)$$

Where V_{cc} is the voltage of the capacitor charging and t_1 is the time spent charging and t_2 is the time spent discharging. The FeGD is observed to have different charging rates and different magnitudes of discharging between pulses based on



((a))



((b))

Fig. 5: A consistent, pulsed current applied to (a) bulk capacitors in the LIF circuit and (b) the FeGD in various polarization states in the LIF circuit. The pulsed current has a period of 2×10^{-10} seconds with a rise, fall, and on time of 5×10^{-11} seconds.

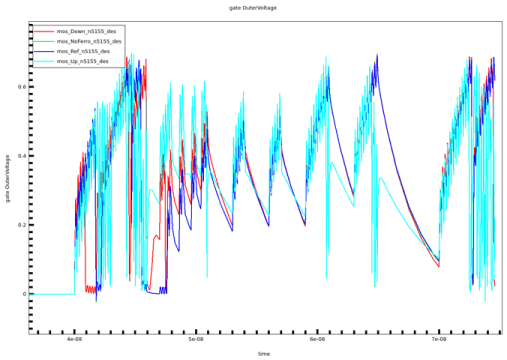


Fig. 6: Encoded current pulses on the ferroelectric gated diode at various polarization states. Different firing frequencies between the polarization states are observed, changing the behavior of the LIF circuit.

the different polarization states. Therefore, with the addition of the firing circuit, the different membrane conditions will lead to different firing rates.

Finally, once firing is added, the different polarization states are observed to have different firing rates. Figure 6 shows a sample of what an encoded input pulse sequence might look like on the FeGD circuit. The different polarization states lead to different firing between the Up polarization and the other polarization states. Therefore, more important data can have the polarization in the FeGD modified to increase that node's amount of firing to improve the SNN's results.

The FeGD can also be observed to change the amount of power consumption based on the polarization states. Based on a single firing sequence, a MOSFET of the same structure uses approximately 2.294×10^{-15} J/spike, with the reference and down polarizations consuming similar power at $2.342E-15$ and 2.424×10^{-15} J/spike, respectively. Meanwhile, the Up polarization consumed $2.074E-15$ J/spike. In dimensions, the FeGD tested in this work is $4\mu m$ long and $0.4\mu m$ thick, taking up a significant amount of area. However, device sizing can be improved with device tuning, especially with vertical and GAA structures that the FeGD can utilize [6].

B. System

Figure 9 shows accuracy results for training runs of the SNN for β_1 and β_2 on the same architecture. After training for 7 epochs, β_1 accurately classified 81.95% of the test set, while β_2 classified 86.68% correctly.

Achieving ¿80% accuracy with only 11 weight levels highlights SNNs' robustness to extreme quantization—crucial for implementation in the CBA described earlier in II-C.

Both accuracy graphs in Figure 9 show a clear drop in performance around iteration 500, where the stretching and quantization begins to take effect. Both accuracies eventually build up their accuracy, but β_2 recovers much faster since it better preserves learned representations by keeping more of the membrane potential between time steps.

The ideal β for a network is largely dependent on the input spike source. These results demonstrate the utility of a configurable β and how changing τ for an LIF-based SNN can produce different results with the same data set and architecture.

IV. Conclusions

In this work, a tunable membrane potential for the Leaky Integrate-and-Fire circuit was proposed and demonstrated utilizing a ferroelectric gated diode to create a tunable capacitor. The FeGD LIF demonstrated a different charging rate based on polarization, which changes the firing rate based on learned data to improve SNN performance. Furthermore, a memristive crossbar array block and LIF array block were modeled in LT Spice, demonstrating how the FeGD LIF circuit could be implemented in a full neuromorphic architecture. Finally, the effects of the FeGD tunable capacitance were demonstrated in an SNN based on SNN Torch. These simulations demonstrated improved performance based on a tunable membrane potential, supporting the addition of the FeGD to LIF circuits.

References

[1] Jiqing Zhang et al. "Spiking Neural Networks With Adaptive Membrane Time Constant for Event-Based Tracking". In: *Trans. Img. Proc.* 34 (Jan. 2025), pp. 1009–1021. ISSN: 1057-7149. DOI: 10.1109/TIP.2025.3533213. URL: https://doi.org/10.1109/TIP.2025.3533213.

0.7

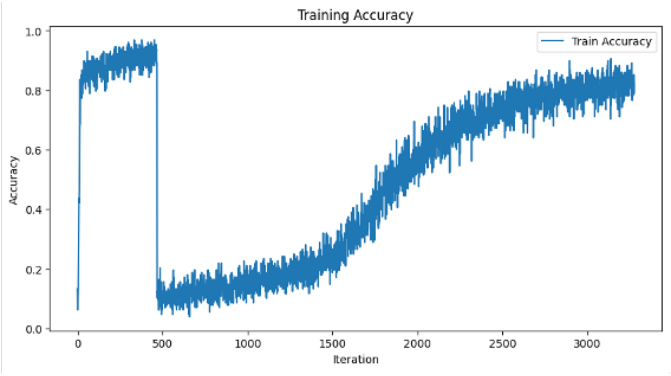


Fig. 7

0.7

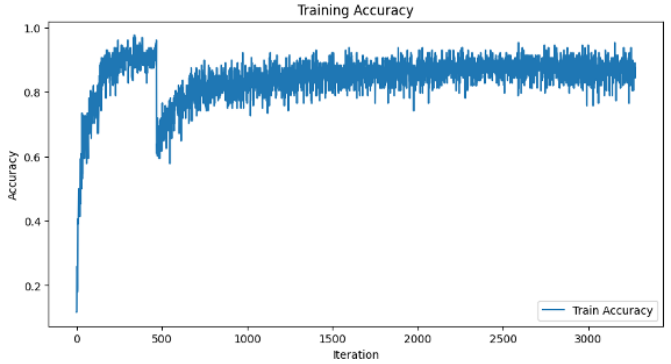


Fig. 8

Fig. 9: Accuracy results after 7 epochs at different decay rates (a) $\beta_1 = 0.4999531419$ and (b) $\beta_2 = 0.9391365887$.

[6] Nicholas Haehn. "Simulation and Testing of Ferroelectric Capacitors and Gated Diodes for Novel Memory Applications". PhD thesis. University of Cincinnati, 2025, p. 146.

[2] Tao Chen et al. "Memristive leaky integrate-and-fire neuron and learnable straight-through estimator in spiking neural networks". In: *Cognitive Neurodynamics* 18.5 (2024), pp. 3075–3091. ISSN: 1871-4099. DOI: 10.1007/s11571-024-10133-w. URL: https://doi.org/10.1007/s11571-024-10133-w.

[3] Jason K Eshraghian et al. "Training Spiking Neural Networks Using Lessons From Deep Learning". In: *Proceedings of the IEEE* 111.9 (2023), pp. 1016–1054. DOI: 10.1109/JPROC.2023.3308088.

[4] M J Rozenberg, O Schneegans, and P Stoliar. "An ultra-compact leaky-integrate-and-fire model for building spiking neural networks". In: *Scientific Reports* 9.1 (2019), p. 11123. ISSN: 2045-2322. DOI: 10.1038/s41598-019-47348-5. URL: https://doi.org/10.1038/s41598-019-47348-5.

[5] T Mikolajick, U Schroeder, and S Slesazeck. "The Past, the Present, and the Future of Ferroelectric Memories". In: *IEEE Transactions on Electron Devices* 67.4 (2020), pp. 1434–1443. DOI: 10.1109/TED.2020.2976148.

[7] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[8] F. Aguirre. "Hardware implementation of memristor-based artificial neural networks". In: *Nature Communications* 15.1 (2024), p. 1974.

On the Structure of the Linux Kernel

Ansh Mendiratta, *BSc, Comp. Sci.*

***Abstract*—The free and open-source Unix-like kernel is a notoriously daunting code base. In an effort to unveil the inner workings of and to lend ideas for possible starting points for learning inside the kernel, this article outlines and very briefly describes its structure, but does not specify how these parts unify.**

I. Introduction

THERE are many old discussion forums scrutinizing the kernel's design, some of which are even by Andrew S. Tanenbaum [1]. Whether or not Linus Torvalds envisioned the perfect kernel is beyond the scope of this article. However, what is not is what he and some 16,000 contributors have envisioned since then [2]. Although the kernel[1] has undergone massive change since its 0.02 release in 1991 [3]. Today, it can roughly be divided into subsystems: device drivers, memory management, process management, a network stack, a virtual filesystem, and architecture-specific code.

II. Kernel Modules

Monolithic kernels such as the Linux kernel, although faster than microkernels, suffer from the lack of modularity. More modern monolithic kernels solve this problem by using *kernel Modules* [4].

Kernel modules are pieces of code that can be dynamically loaded and unloaded upon request into the kernel without requiring a system reboot.

There are some key differences between kernel modules and user-land programs [5].

Kernel modules:

- are in a separate address space from user programs.
- have higher execution privileges.
- are not sequential.
- can be interrupted.
- must be preemptable. (To be preempted is for the CPU to be interrupted in the middle of executing kernel code.)
- can share data across threads (like drivers).

Beyond execution, there are also key structural differences. Kernel modules:

- do not define a `main()`. Instead, they're just a collection of subroutines and data.
- are linked ONLY to the kernel.
- use different header files from the ones user programs use.
- avoid global variables even more aggressively than user programs.
- can be customized for specific hardware.

[1]The Linux kernel is written in a special C language supported by GCC that the allows inlining of ASM into high-level code that has been compiled.

1) A note on address spaces: Some architectures, such as SPARC and x86, behave differently when the kernel attempts to directly access user address space. On SPARC, the system panics. On x86, the system does not; direct access is allowed but discouraged for the sake of portability. There are several routines to help with the accessing of user address space, notably `ddi_copyin(9F)` and `ddi_copyout(9F` but they will not be covered here.

The following subsections outline the major subsystems of the Linux kernel. Some of these are implemented as kernel modules, such as device drivers, while the others are not and always loaded into the kernel (because they form a part of it).

III. Device Drivers

A driver is a loadable *kernel module* [5] that acts as an interface and manages data transfers between the operating system and a device [5], organized as a collection of C routines (which must use standard interfaces called "entry points") and data structures. It is through these "entry points" abstraction that the calling kernel modules are shielded from the internal details of the driver. These drivers are written using the kernel's "API" that also generalizes over the implementation details of the filesystem.

Device drivers do not directly talk to each component of a device. Instead, they communicate with the hardware controller that manages the hardware. The driver shields the user application layer from the details of the specific device so that application level or system calls can be generic or device-independent.

The drivers can be thought of as assuming the same role as generics in a programming language, wherein you pass in something that fits some vague criteria, with the specifics being left to the implementation.

A. Data Transfer

Because data transfer between a device and the system is slower than data transfers internal to the CPU, when in a data transfer, the device typically suspends the execution of its own thread until complete. During this time, the CPU is free to work on other threads. Once the data transfer is complete, the driver sends an interrupt to the CPU and instructs it to resume execution.

B. Entry Points

A device driver declares its general entry points in its `dev_ops(9S)` structure. A driver declares entry points for routines that are related to character or block data in its `cb_ops(9S)` structure.

There are a number of different driver entry points. Different devices would require different entry points. See figure 6 for a list.

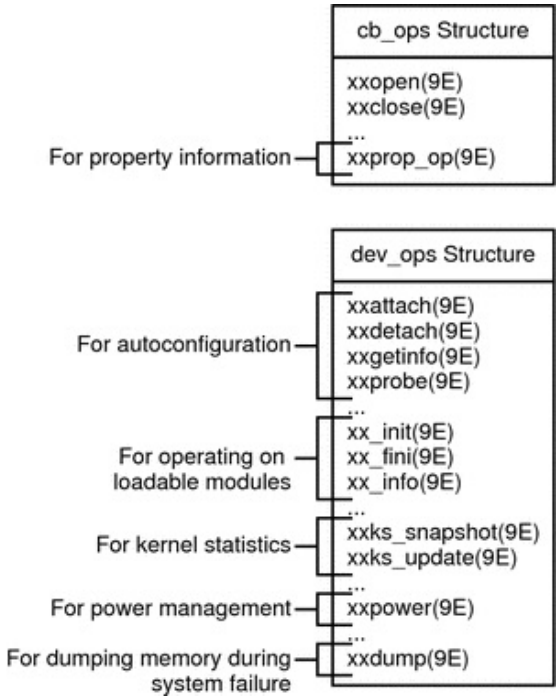


Fig. 1: Typical Device Driver Entry Points

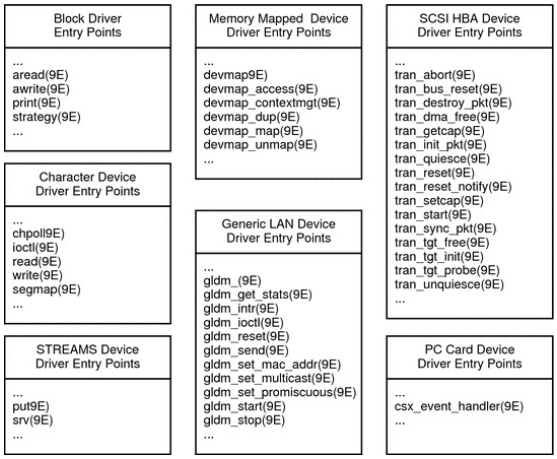


Fig. 2: Entry points for different drivers

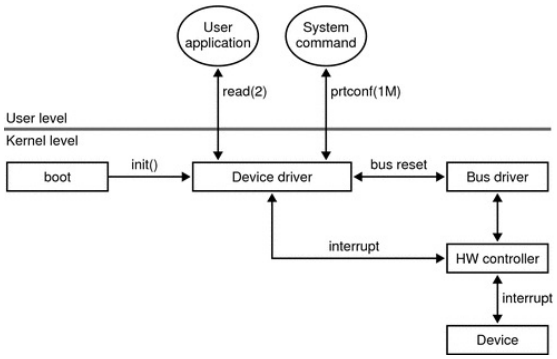


Fig. 3: Device driver interactions

IV. Memory Management

Memory management [6] is a vast topic that deserves an article on its own. In this article, we will cover some fundamentals.

The kernel memory management implements, among others, virtual memory, demand paging, memory allocation for both kernel internal structures and user space programs, and the mappings of files into processes' address space.

Definition IV.1 (Virtual Memory)**.** An abstraction over physical memory that, similarly to drivers, attempts to hide the hardware-specific complexities behind a veil.

Physical memory is abstracted away behind virtual memory, an abstraction meant to simplify the reference of memory in the kernel. To strip away this abstraction, every memory access uses a virtual memory address that, when read by the CPU, is translated to a physical memory address that the memory controller can understand.

A. Virtual Address Translation

Definition IV.2 (Pages)**.** Physical system memory is divided into page frames, or pages, whose size is architecture-specific. Each page can be mapped as one or more virtual pages, described in page tables that allow translations from virtual addresses used by programs to physical memory addresses. These tables are organized hierarchically.

Definition IV.3 (Page Cache)**.** A cache of the most recent reads/writes of files. For writes, the data are placed in the page cache which is marked as *dirty*. When Linux decides to reuse the page cache for other purposes, it makes sure to synchronize the file contents on the device with the updated data.

Physical system memory is divided into page frames, or pages, whose size is architecture-specific. Each page can be mapped as one or more virtual pages, described in page tables that allow translations from virtual addresses used by programs to physical memory addresses. These tables are organized hierarchically.

In this hierarchy:

- The tables at the lowest levels contains physical addresses of actual pages used by the software.
- The tables at higher levels contain physical address of the pages belonging to the lower levels.
- The pointer to the top-level page resides in a register.

The CPU uses a register to access the pointer pointing to the top-level page. It uses the high bits of the virtual address to index an entry in the top-level page table. This entry is now used to access the next table with the next bits of the virtual address as an offset, and so on, until the actual page is reached where the final bits of the virtual address are used as the offset inside the page.

B. Huge Pages

Definition IV.4 (Huge Pages)**.** A mechanism that allows programs to ask for pages larger than the default size (4 KB).

The address translation from virtual to physical requires several memory addresses (memory page tables) and these accesses are slow relative to CPU speed. To avoid spending as many processor cycles, the CPU maintains a cache of such translations called a Translation Lookaside Buffer (TLB). Usually, the TLB is a pretty scarce resource and applications with a large working memory set will experience performance hits because of frequent cache misses.

Many modern CPU architectures allow mapping of the memory pages directory by the higher levels in the page table. On x86, it is possible to map 2M and even 1G pages using entries in the second and third level page tables. In Linux, these pages are called *huge*. Using huge pages significantly reduces pressure on the TLB, improves TLB hit-rate, and thus improves performance.

There exist two mechanisms in Linux that enable mapping of the physical memory with the huge pages:

- HugeTLB filesystem (`hugetlbfs`): A pseudo-filesystem that uses RAM as its backing store.
- Transparent Huge Pages (`THP`): the system memory mapped by the huge pages is is managed transparently to the user, in contrast to `hugetlbfs`.

C. Zones

Definition IV.5 (Zones)**.** Linux groups memory pages into *zones* based on their possible usage. For instance, `ZONE_DMA` will contain memory that can be used by devices for DMA (direct memory access); `ZONE_HIGHMEM` will contain memory that is not permanently mapped into kernel's address space and `ZONE_NORMAL` will contain normally addressed pages.

The actual layout of the memory zones is hardware dependent as not all architectures define all zones, and requirements for DMA are different for different platforms.

Hardware can impose restrictions on how different physical memory ranges can be accessed. Sometimes, devices cannot perform DMA to all addressable memory. Some other times, the size of the physical memory exceeds the maximal addressable size of virtual memory and special actions are needed to access portions of the memory.

D. Reclaim

In the lifetime of a system, a physical page can be used for storing different kinds of data: kernel internal data structures, DMA'able buffers for device drivers, data read from a filesystem, memory allocated by user space processes, etc.

Depending on the page usage, it is treated differently by the Linux memory management. Pages are called *reclaimable* if they can be freed at any time. This could occur if they cache data available elsewhere, or because they can be swapped out to the hard disk. The most notable categories of reclaimable pages are page caches and anonymous memory.

Definition IV.6 (Anonymous Memory)**.** Memory that is not backed by a filesystem. Such mappings are implicitly created for a program's stack and heap or by explicitly calls to `mmap(2)`. Usually, these mappings only define virtual memory that the program is allowed to access.

In most cases, pages holding internal kernel data and used as DMA buffers cannot be repurposed, and they remain pinned until freed by their user. Such pages are called un-reclaimable. However, under certain circumstances, such as when the system is under memory pressure, even pages occupied with kernel data structures can be reclaimed. Such an example may occur when memory caches of filesystem metadata can be re-read from the storage device and it is therefore possible to discard them from main memory at the cost of slow accesses.

This process of freeing reclaimable physical memory pages and re-purposing them is called *reclaim*. Linux can do this synchronously or asynchronously, depending on the state of the system.

When the system is not loaded, most memory is free and allocation requests will be satisfied immediately from the free pages supply. As the load increases, the amount of free pages goes down and when it reaches a certain threshold (low watermark), an allocation request will awaken the `kswapd` daemon (see IV-G). This daemon will asynchronously scan memory pages and either just free them if the data is available elsewhere, or evict them to the backing storage device (finish the dirty pages job). As memory usage increases even more and reaches another threshold (min watermark), an allocation will trigger *direct reclaim*. In this case, allocation is stalled until enough memory pages are reclaimed to satisfy the request.

E. Compaction

As the system runs, tasks allocate and free memory. Enough of these and the memory can become fragmented. While it is possible with virtual memory to present scattered physical pages as virtually contiguous, this does not sidestep the problem of large contiguous allocations. Such a need may arise when a device driver requires a large buffer for DMA, or when THP allocates a huge page.

Memory *compaction* addresses this fragmentation issue. This mechanism moves occupied pages from the lower part of a memory zone to free pages in the upper part of the zone. When a compaction scan is finished, free pages are grouped together at the beginning of the zone and allocations large physically contiguous areas become possible.

Like reclaim, compaction may occur asynchronously in the `kcompactd` daemon or synchronously as a result of a mem allocation request.

F. OOM Killer

In the event that all memory has been exhausted, the kernel will be unable to reclaim enough memory to continue its operation. In order to save the rest of the system, it invokes the *OOM killer*.

The *OOM killer* selects a task based on its `oom_score` [7] to sacrifice for the sake of the overall system health. The selected task is killed in hope that after it exits, enough memory will be freed to continue normal operations.

You can check a process's OOM score by looking at `/proc/$PID/oom_score` where `$PID` is the relevant process ID. This score is simply calculated as

`100 * mem_usage_percentage` where the factor of `100` is introduced because . Privileged processes get a small subtraction of 30 to their score because they are considered more valuable [8].

The actual killing of a process also depends on `oom_score_adj`. The final value used by the killer is `oom_score + oom_score_adj`. You can manually set `oom_score_adj` to be -1000, which effectively prevents the process from being targeted even if it uses 100% of the memory allocated to it [9].

G. Swap Memory

As an aside, swap memory is a part of permanent storage such as an SSD/HDD that acts as an extension of main memory. In particular, it stores more inactive data from the main memory. This memory can be used even if the main memory is not full, and this is done at the discretion of the kernel when the swap algorithm determines which memory pages have not been in used for a long time and would rather free up main memory than retain inactive processes/pages.

V. Process Management [10]

Definition V.1 (Process)**.** A process in the kernel is an abstraction that groups together several resources: an address space, shared memory regions, one or more threads, timers, signal handlers, opened files, sockets, semaphores, status information, and more. In the kernel, this is all in the struct `task_struct`.

Definition V.2 (Thread)**.** A thread is a basic unit that the kernel process scheduler uses to allow applications to run the CPU. A thread has the following characteristics:

- Each thread has its own stack and together with the register values, it determines the thread execution state.
- A thread runs in the context of a process and all threads in the same process share the same resources.
- The kernel schedules threads, not processes, and user-level threads (e.g., fibers, coroutines, etc) are not visible at the kernel level.

Typically, threads are implemented as a separate data structure which is linked to the process data structure. Linux, however, does things differently. The basic unit is called a task and it is used for both processes and threads. And instead of embedding resources in the task struct, it contains pointers to these resources. This allows you to determine if two threads are in the same process: if they point to the same resource structure instance, they are the same.

A. Clone

In Linux, a new thread or process is created with the `clone()` syscall. Both `fork()` and `pthread_create()` use `clone()`.

This call allows the caller to decide what resources should be shared with the parent and which should be isolated/copied:

- `CLONE_FILES`: shares the file descriptor table with the parent
- `CLONE_VM`: shares the address space with the parent
- `CLONE_FS`: shares the filesystem information (root directory, current directory) with the parent
- `CLONE_NEWNS`: does not share the mount namespace with the parent
- `CLONE_NEWIPC`: does not share the IPC namespace (System V IPC objects, POSIX message queues) with the parent
- `CLONE_NEWNET`: does not share the networking namespaces (network interfaces, routing table) with the parent

if `CLONE_FILES | CLONE_VM | CLONE_FS` is used, then effectively a new thread is created.

B. Access

Since fields of the `task` struct need to be frequently accessed, a per CPU variable (variable that the CPU stores) is used to store and retrieve the pointer to the current `task_struct`.

C. Context Switching

Every time the kernel must context switch from one task to another, the kernel saves the user registers on the kernel stack, and switches tasks. When returning, it pops off values from the stack to restore them and continue the operations of the task. To force a context switch, either a syscall or an interrupt must occur, which is what triggers the saving of the user registers. Then, the `schedule()` [11] syscall is called and tasks are switched. Finally, `context_switch()` will perform architecture-specific operations and will switch the address space if needed.

VI. Virtual File System

Definition VI.1 (Virtual File System)**.** The software layer in the kernel that provides the filesystem interface to user programs, as well as a switching layer between the SCI and the filesystems supported by the kernel (switch between different, coexisting filesystem implementations in the kernel).

- At the top: API abstraction of functions such as `open`, `close`, `read`, `write`, `stat`, and `chmod` called from a process context.
- At the bottom: abstractions defining how the upper-level functions are implemented. These are plug-ins for the filesystem, for which the sources are located in `./linux/fs`.

Below the filesystem is the buffer cache, that provides a different set of common functions for the filesystem (independent of any specific filesystem). Like the name suggests, this layer optimizes access to the physical devices by keeping data around for a short time (or speculatively read ahead so that the data is available when needed). Below even the buffer cache is the device drivers, which implement the interface for the particular physical device.

More in III.

A. Directory Entry Cache (dcache) [12]

The pathname passed to the above implemented system calls if used by the virtual filesystem to search through the directory entry cache (also known as dcache or dentry cache). This is a fast look-up mechanism to translate a pathname into a specific dentry. Dentries live in RAM and are never persistent — they exist only for storage.

Since the dentry cache is meant to be a view into your entire filespace, most computers cannot fit all the dentries in the RAM at the same time. To mitigate this, the virtual filesystem may create dentries along the way and then load the inode. This is done by looking up the inode.

B. Inode

Definition VI.2 (Inode)**.** In a Unix-like filesystem, an object such as a file, directory, and FIFOs [13].

A dentry usually have a pointer to an inode. They live either on the disk (block device filesystems) or in the memory (pseudo-filesystems). Inodes on the disk are copied into memory when required and changes to the inode are written back to the disk. A single inode can be pointed to by multiple dentries (think hard links).

To look up an inode requires that the virtual filesystem calls the `lookup()` method of the parent directory inode. This method is installed by the specific filesystem implementation that the inode lives in. Once the virtual filesystem has the dentry, it peeks at the inode data and passes some of it back to user space.

C. File Object

Opening a file requires another operation: allocation of a file structure (kernel-side implementation of file descriptors). The file structure is initialized with a pointer to the dentry and a set of file operation member functions. These are taken from the inode data. `open()` is called so the specific filesystem implementation can do its work.

D. Registering and Mounting a Filesystem[2]

```
#include <linux/fs.h>

register_filesystem(struct file_system_type *);
unregister_filesystem(struct file_system_type *);
```

The above is used to register/unregister a filesystem. The parameter struct is defined as:

```
struct file_system_type {
    const char *name;
    int fs_flags;
    struct dentry *(*mount)
                (struct file_system_type *, int,
                 const char *, void *);
    void (*kill_sb) (struct super_block *);
    struct module *owner;
    struct file_system_type * next;
    struct list_head fs_supers;
    struct lock_class_key s_lock_key;
    struct lock_class_key s_umount_key;
};
```

[2]The code blocks have been edited to fit the width of the column of the page they are in.

Once `register_filesystem` is called, the virtual filesystem will call the appropriate `mount()` method for the specific filesystem. A tree pointed to by a vfsmount will be returned by `->mount()` and attached to the mountpoint, so that when pathname resolution reaches the mountpoint it will jump to the root of that vfsmount.

VII. Network Stack

Both because of the sheer complexity of the networking stack and its poor documentation, we avoid discussing this part of the kernel in any useful detail. Instead, we provide a blanket discussion of common networking ideas and their implementation in the kernel.

The Network Stack [14] is an optional component of the kernel. With most uses of a system requiring some connectivity, this is rarely a fact that manifests usefully.

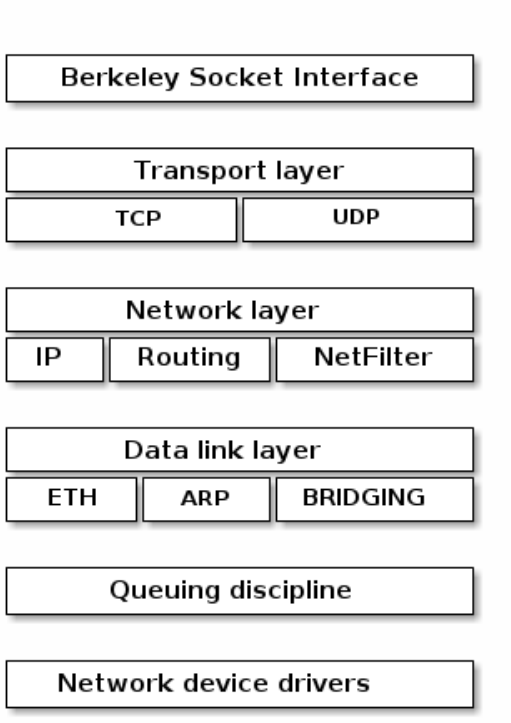


Fig. 4: An overview of the network management in the kernel

A. Networking in User Space

The abstraction of network communication in user space is the socket. Sockets abstract a communication channel and is the kernel-based TCP/IP stack interaction interface. An IP socket is associated with an IP address, the transport layer protocol used (TCP, UDP, etc) and a port.

Common functions that use sockets:

- creation (`socket`)
- initialization (`bind`)
- connecting (`connect`)
- waiting for a connection (`listen`, `accept`)
- closing (`close`)

Network communication is achieved via `read`/`write` or `recv`/`send` calls for TCP sockets and `recvfrom`/`sendto` for UDP sockets. Transmission and reception operations are transparent to the application, leaving encapsulation and transmission over network at the kernel's discretion. However, it is possible to implement the TCP/IP stack in user space using raw sockets (the `PF_PACKET` option when creating a socket), or implementing an application layer protocol in kernel (TUX web server).

B. Implementation in the kernel

The kernel provides three basic `structs` for working with packets: `socket`, `sock`, and `sk_buff`.

- `struct socket` is an abstraction very close to user space, i.e., BSD sockets used to program network applications.
- `struct sock` or *INET socket* in Linux terminology is the network representation of a socket.

The first two are related: `socket` contains an INET socket field, and every `sock` has an BSD socket that holds it.

`sk_buff` is the representation of a network packet and its status. The structure is created when a kernel packet is received, either from the user space or from the network interface.

The above `structs` have methods associated with them. `socket`'s methods are listed above and we omit the others.

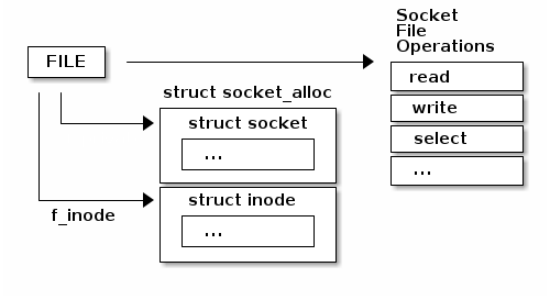


Fig. 5: An overview of the socket implementation

C. Conversions

Binary can be interpreted as either Little Endian (most significant bit to the left), or as Big Endian (most significant bit to the right). Different hosts may choose to interpret numerical data differently, and so the Internet has imposed a standard sequence for the storage of numerical data, called *network byte-order*. In contrast to this, the byte sequence for numerical data on a host is called *host byte-order*. When receiving/sending data, the byte-order format should be converted into the appropriate order.

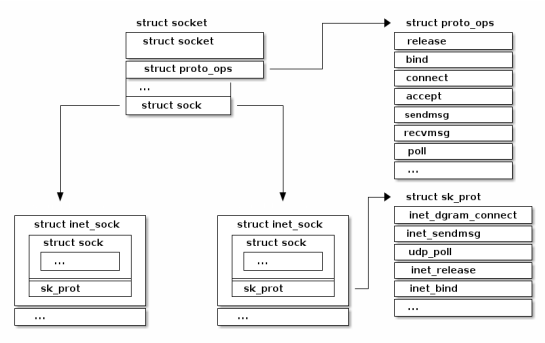


Fig. 6: Some socket families and protocols

VIII. Final Remarks

We acknowledge that this work leaves many gaps in how every notion mentioned fits together inside the kernel, and how some of the concepts mentioned work entirely (most obviously the networking stack). Fitting the long and complicated anatomy of the kernel would be entirely out of the scope of an IEEE@UC magazine article. However, we leave further knowledge and understanding as an exercise for the reader if they are so inclined. For anyone, we recommend the official kernel documentation archive at https://docs.kernel.org/. Few resources are similarly comprehensive, so any learning should occur by being cross-pollinated by several sources, and — most helpfully — by reading the kernel source.

All in all, the kernel is a complicated and yet relatively simple work of art. While most of its contributions these days are by corporate entities with possibly a team worth of people contributing, the language of choice, C, makes reading the kernel a much easier task than one may expect — at least compared to other monolithic projects. Even a cursory glance over the source code illuminates many ideas and clarifies how ideas in theory can be implemented in practice, starting at the lowest level; which can act as a epiphany for new learners and provide starting points when attempting to write not just functional, but performant and correct code.

References

[1] Andrew S. Tanenbaum. *LINUX is obsolete*. URL: https://groups.google.com/g/comp.os.minix/c/wlhw16QWltI?pli=1.

[2] *Linux Kernel GitHub Mirror*. URL: https://github.com/torvalds/linux/.

[3] *Linux Kernel Wikipedia*. URL: https://en.wikipedia.org/wiki/Linux_kernel.

[4] *Kernel Modules*. URL: https://linux-kernel-labs.github.io/refs/heads/master/labs/kernel_modules.html.

[5] *Oracle Kernel Overview*. URL: https://docs.oracle.com/cd/E19120-01/open.solaris/819-3159/6n5d9rbv7/index.html.

[6] *Linux Memory Management Overview*. URL: https://docs.kernel.org/admin-guide/mm/concepts.html#concepts-overview.

[7] *neo4j Linux Out of Memory Killer*. URL: https://neo4j.com/developer/kb/linux-out-of-memory-killer/.

[8] *adilrk Linux Out of Memory Killer*. URL: https://medium.com/@adilrk/linux-oom-out-of-memory-killer-74fbae6dc1b0.

[9] *man page choom*. URL: https://man7.org/linux/man-pages/man1/choom.1.html.

[10] *Kernel Processes Lecture*. URL: https://linux-kernel-labs.github.io/refs/heads/master/lectures/processes.html.

[11] *man7 sched*. URL: https://man7.org/linux/man-pages/man7/sched.7.html.

[12] *Kernel Filesystem Documentation*. URL: https://www.kernel.org/doc/html/v6.3/filesystems/vfs.html.

[13] *man7 fifo*. URL: https://man7.org/linux/man-pages/man7/fifo.7.html.

[14] *Kernel Networking Lecture*. URL: https://linux-kernel-labs.github.io/refs/heads/master/labs/networking.html.

MakeUC Hackathon 2024 Recap

Katy Hildebrant, *B.S., Elec. Eng. and Comp. Eng.*

***Abstract*—MakeUC is the hackathon hosted by the University of Cincinnati's student section of IEEE every fall. The event spans two days with 24 total hours of hacking in which individuals or teams create projects focused on software and/or hardware. Students attended workshops to learn new skills and tools, networked with sponsors and fellow hackers, and enjoyed fun mini-events throughout. This year's event took place online and in person at the 1819 Innovation Hub.**

I. Background

MakeUC is the hackathon hosted by IEEE at the University of Cincinnati every fall. It is a 24-hour event in which individuals or teams create projects centered around software and/or hardware, learn new skills, and network with each other and event sponsors. This year's competition took place on November 9th and 10th online and in-person at the University of Cincinnati's 1819 Innovation Hub. The main activity was "hacking", but a sponsor expo, multiple workshops, and a Capture The Flag (CTF) cybersecurity challenge hosted by Cyber@UC were among the other fun activities available at the event. In-person participants also had access to a variety of unique benefits at the 1819 Makerspace, which boasts a woodworking shop, soldering stations, 3D printers, laser cutters, and many other tools for professional prototyping and fabrication.

II. Projects

All hackers were encouraged to submit a project in teams of one to four people. A total of 60 projects were submitted by the deadline on November 5th. The event had track prizes (optional focus areas or themes for projects), sponsored prizes, and prizes for first, second, and third overall. Below are the top three projects overall.

1) *Emergency Exit* is a device aimed at enhancing user safety in active fire situations. The application takes the user's current location as well as a given destination and dynamically generates the safest route between the two points. On the app side, the team utilized various APIs to obtain real-time fire data and hotel availability as well as to provide audio directions. On the hardware side, the team utilized a Raspberry Pi to run the application and provide the user with basic radio functions. This allows the user to tune in for public broadcasts and to make their own transmissions over short distances. The members of this team were Andrew Zhang, Sebastian Alexis, and Priansh Mittra.
2) *Uni-Guesser* is inspired by GeoGuessr, a popular online game in which players must deduce a location based on random Google Street imagery. In Uni-Guessr, players receive a random image taken on campus at the University of Cincinnati. The closer they are to the actual location, the more points they get. This project's front-end was written in React and Tailwind CSS, while the back-end was written in Flask and Python. The members of this team were Logan Muhlen, Andy Au, and Cameran Beason.
3) *The Skilluminati* is a one-stop platform to empower both students and recruiters to quickly identify a resume's strengths and weaknesses. The tool compares the resume's listed skills and experience with a general career goal or a specific job posting, then highlights gaps and suggests improvements. This helps a job-seeker expand their skills and tailor their resume to a given role, while also helping recruiters make informed hiring decisions. The members of this team were Advait Vagerwal and Sahil Thakare.

Fig. 1: Hackers hard at work

Fig. 2: Hackers having fun

III. Workshops

Workshops are short, hands-on sessions hosted by sponsors where hackers can expand their skills and learn more about topics relating to careers, technology, and/or the hackathon themes. This year, workshops were given by Kinetic Vision, Cyber@UC, and MakeUC organizers.

1) *Development Environments* by Kinetic Vision
   In this workshop, hackers learned how to build efficient and productive development environments using tools such as Windows Subsystem for Linux, Visual Studio Code, and the Linux/Unix terminal. They also learned about version control with Git, containerization with Docker, and other tips and tricks to streamline their workflow and boost productivity for both hackathons and everyday use.
2) *Intro to CTF* by Cyber@UC
   In this workshop, Cyber@UC club members presented an overview of their beginner-friendly Capture the Flag competition. They explained the different categories of problems, which included open-source intelligence and reverse engineering, and how hackers can begin to tackle them. They also highlighted many helpful tools for hackers to try such as Kali Linux and CyberChef.
3) *Hackathon 101* by Sam Burkhard
   In this workshop, Sam provided hackers with helpful hackathon information. Topics included what a hackathon is, how to find a team, how to brainstorm for your idea, and how to navigate the judging process. He also shared general tips for success, such as how to best plan out your 24 hours of hacking and how to best finalize your project submission.

Fig. 3: Cyber@UC hosting the CTF Challenge

IV. Learn More

If you are interested in learning more about MakeUC, please visit our website at https://makeuc.io. This website is updated for each year's hackathon, so be sure to take a look for important updates as well as links to our other platforms. If you would like to get involved with the event or if you have any questions for us, please send an email to info@makeuc.io. If you would like to learn more about the IEEE@UC student branch, please visit our website at https://ieee.uc.edu.

V. Acknowledgements

We are very grateful to the following groups for supporting this event.

- The MakeUC sponsors, for dedicating time, money, and other resources to supporting valuable opportunities for students.
  - Swallowtail tier: HII
  - Birdwing tier: Kinetic Vision
  - Morpho tier: Seeed Studio, Siemens, Infinera, ICR
  - Monarch tier: .xyz, Axure
- The MakeUC judges and mentors, for providing valuable expertise and feedback to the participants.
- The MakeUC hackers, for sharing their creativity with the rest of the community.
- The MakeUC director, for making MakeUC possible!
  - Elaine Mansour
- The MakeUC logistics team, for coordinating all organizer teams and hackathon operations.
  - Sam Burkhard (lead), Pradnyesh More, Sai Vivek Kambam, Sucheta Ghosh, Ryan Sieber, Oriana Hernandez Alvarado, Joy Hernandez, Anay Sehgal, Quan Le, Katy Hildebrant
- The MakeUC marketing team, for maintaining our branding, promoting the event, and connecting with hackers.
  - Ziad Oweis (lead), Maanya Naveen (lead), Kristin Hildebrant, Alex Budnik, Jasmine Walls, Kaustubh Mathur, Arleen Monteiro, Anna Franchi, Maite Peña, Mayumi Chinchihualpa Paredes, Anas Khairy
- The MakeUC sponsorship team, for connecting with our sponsors.
  - Khaled Oweis (lead), Vedant Samarth, Sharanya Awasthi, Poorna Sasidhar Penujeevi Venkata, Sushant Padhye, Anton Hoelmer, Forrest Bushstone, Dhyan Patel, Asmita Yalamanchili, Shashank Sathiyanarayanan
- The MakeUC technology team, for developing and maintaining our website, Discord server, and other tools.
  - Rai Duong (lead), Jason Welsh, Metty Ukey, Aniruddhan Ramesh, Raihan Rafeek, Bao Huynh

Fig. 4: The MakeUC Organizer Team (not all pictured)

My Experience at Innovation Challenge

Mettika Ukey, *BS, Elec. Eng.*

***Abstract*—The Innovation Challenge is a semester-long competition in which students compete to sell the best product to a panel of judges. The CEAS Tribunal holds weekly meetings to guide students in completing their project. The final product is judged on the basis of various criteria such as marketability and originality. All students who complete the deliverables are rewarded with a stipend, with the top teams earning a bonus.**

I. Introduction

I HAD the privilege to participate in the Innovation Challenge (hosted by CEAS Tribunal) in the Fall 2024 semester. I worked alongside my partner, Niva Ranjit *BS, Elec. Eng 2026.* We wanted to create a product that would be useful to college students in their daily lives while also solving a nationwide issue. We decided to prototype a smart water bottle that tracks a user's water intake.

II. Ideating

A. Problem

All across the board, doctors agree that Americans do not drink enough water. According to Kettering Health "Nearly 75% drink at most 2.5 cups of water...suffice it to say: 2.5 cups aren't enough". This is problematic, as our body is mostly made up of water. From our brain to our heart to our hair, water is in every cell of our body and is necessary to perform all of our bodily functions [1].

Additionally, lack of water can cause dehydration. Some effects of dehydration include

- Headaches
- Low blood pressure
- Low energy
- Muscle Cramps
- Dull Skin
- Mood swings
- Loss of stamina
- Decreased motivation
- Lack of focus
- Weight gain

These effects are especially detrimental to college students, who already undergo immense amounts of stress.

B. Solution

To combat this epidemic, my partner and I wanted to create a product that effectively targets the issue and offers a solution. The primary reason why Americans don't drink enough water is due to the fact that they wait until they are thirsty to drink. Oftentimes, thirst is also confused with hunger [1]. We decided to make a smart water bottle that tracks the users water consumption and reminds the user when they need to drink more water.

III. Market Research

In order to effectively market this product, we need to gauge the water consumption habits of college students. We did this by conducting a survey and asking students of University of Cincinnati for their input. With 50+ responses, we found that most students carry a water bottle with them regularly and drink 4-6 cups of water a day.

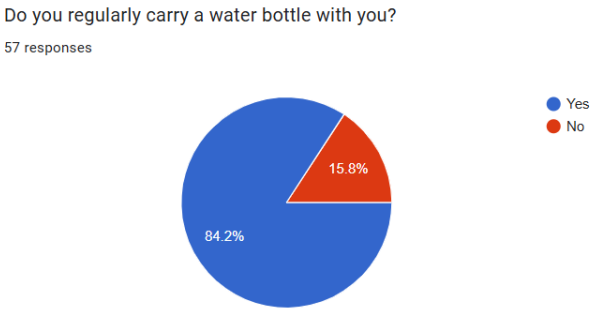


Fig. 1: Students Who Regularly Carry a Water Bottle

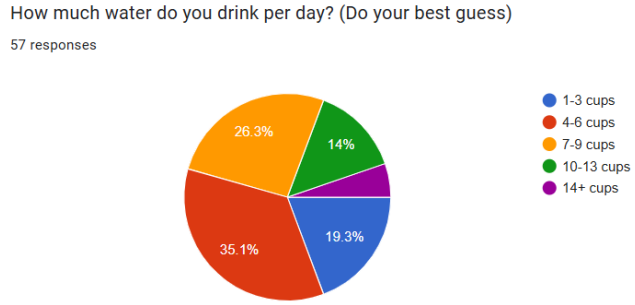


Fig. 2: Student Water Intake (Cups)

This proved to be concerning, because adequate water intake is 13 cups for men and 9 cups for women [2]. This is nowhere near how much students reported drinking. Additionally, the survey confirmed the issue of dehydration lies in human error (forgetting to drink water, waiting until you are thirsty) as opposed to inability to access water as most students regularly carried a water bottle with them.

IV. The product

Figure 3 shows the initial concept we had for our product. It featured a user-friendly screen on the side with all the electrical components located at the bottom. This design would later be changed for logistical reasons to be located at the top of the bottle with the electrical components inside the lid.

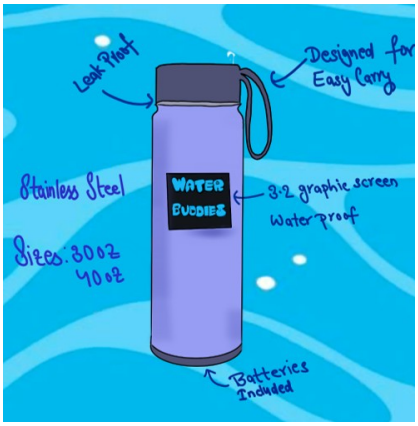


Fig. 3: Concept Design

Fig. 4: Final Prototype

A. Electrical Components

The following are the electronic components we used to build our prototype:

- Arduino Uno – Contains all the logic/code of the product
- HC-SRO4 Ultrasonic sensor – Detects water surface level every 5 seconds
- Waveshare 2.4 Inch LCD Screen – Displays data in real-time and tracks user progress to goal

B. Bottle Model

To make the body of the water bottle tailored to our needs, we used the 3D modeling software *Creality Print*. The model included 2 holes at the base of the lid so that the ultrasonic sensor could effectively read the water level without any additional noise. This was then printed using PLA plastic.

C. Logic

The primary function of the code was to gather data of the water surface level and convert it into volume so that the user knows how much water they drank. The following lines of code illustrate this idea

```
// Calculate volumes
volume_initial = PI * RADIUS * RADIUS *
    height_inital;
volume_current = PI * RADIUS * RADIUS *
    height_current;
volume_consumed = volume_initial -
    volume_current;
```

This converts distance from the water surface level to the ultrasonic sensor to volume, then calculates the change in volume and displays it onto the LCD screen.

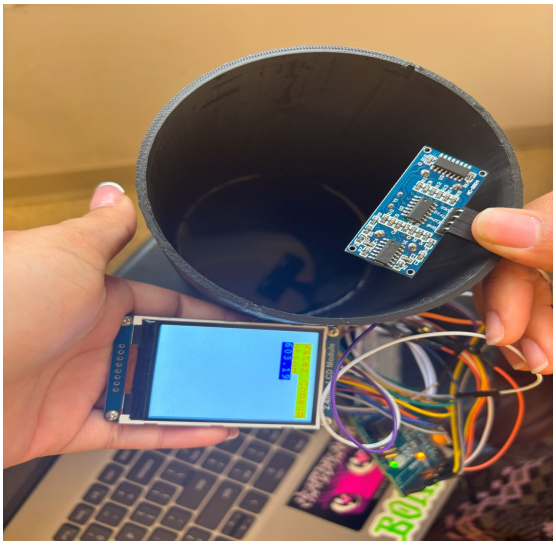


Fig. 5: Picture of Electrical Components

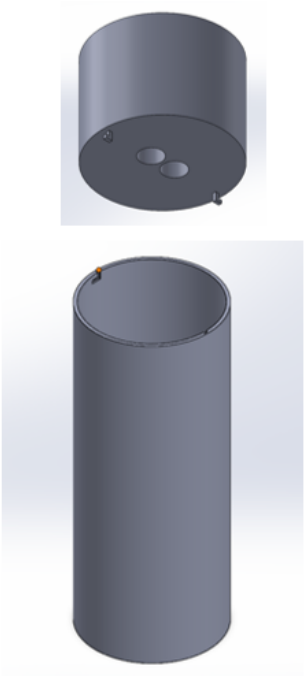


Fig. 6: 3D model of bottle

V. Market Trend

The market for a smart water bottle looks promising. It is no secret that high-quality water bottles are a massive trend

Fig. 7: LCD display example

among the youth. Brands like Stanley Cups, Hydroflasks and Owala have skyrocketed into success due do the demand for water-bottles. Adding a smart feature that helps users track their water intake will set us apart from these competitors. Additionally, the smart water bottle market is expected to grow 10.51% from 2025-2033 [3].

Fig. 8: Smart Water Bottle Market

VI. Conclusion

Through this project, we were made aware of the epidemic surrounding dehydration in college students. We believe that our creation will help combat this issue. Our prototype functions exactly as expected and we are very pleased with how it turned out. Our efforts were recognized with a second-place finish overall. This project reinforced my skills surrounding the engineering design process. From identifying a real-world problem to developing a functional solution, I'm grateful for the opportunity to contribute to the Innovation Challenge.

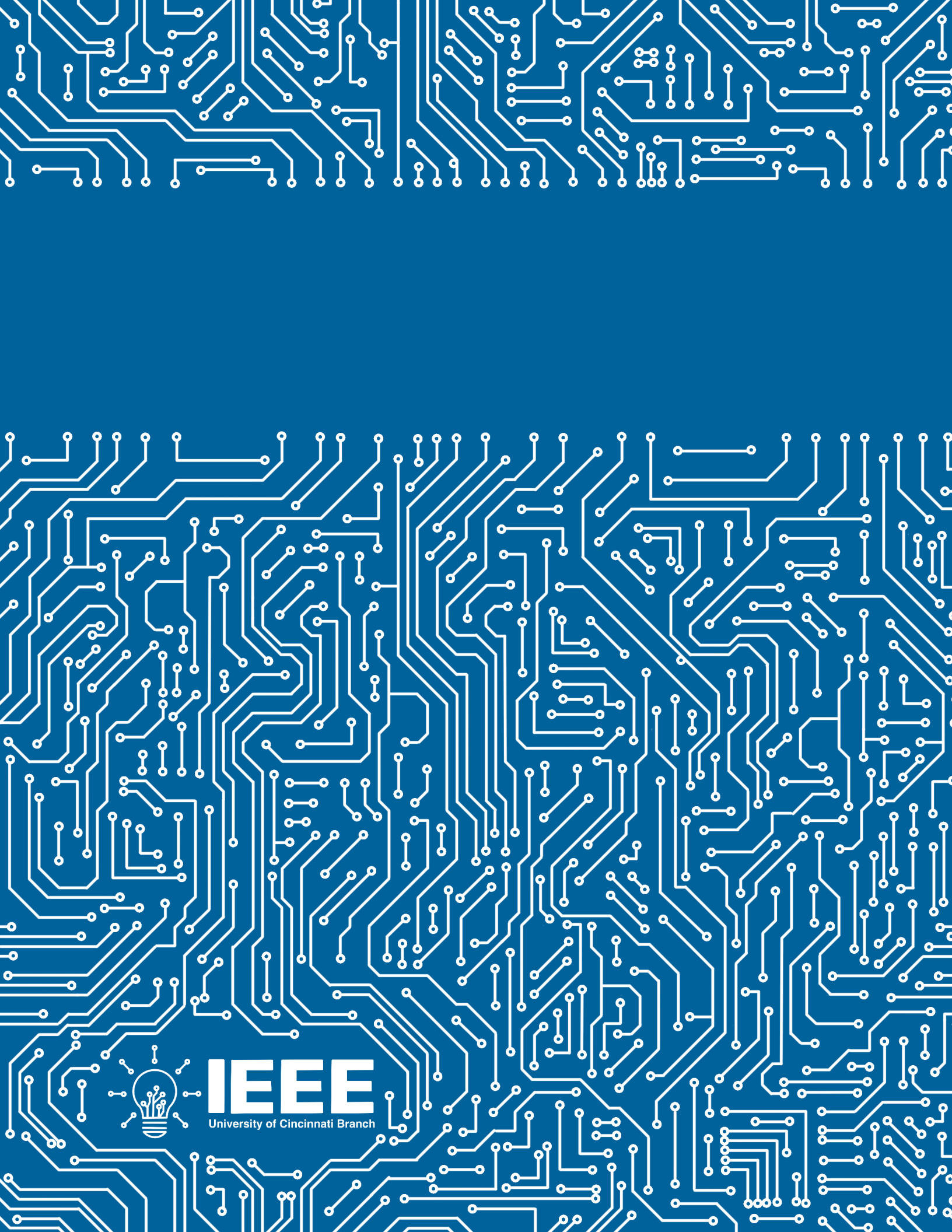
References

[1] *You're Probably Not Drinking Enough Water*. URL: https://ketteringhealth.org/youre-probably-not-drinking-enough-water/.

[2] Ashley Marcin. *How Much Water Do I Need to Drink?* URL: https://www.healthline.com/health/how-much-water-should-I-drink#tips.

[3] Vrushali Bothare. *Smart Water Bottle Market Size, Demand Growth Report by 2033*. URL: https://straitsresearch.com/report/smart-water-bottle-market.

NOTES


IEEE
University of Cincinnati Branch