بسمه تعالی

دانشکده مهندسی کامپیوتر

مدارهای واسط (نیم‌سال اول ۰۳-۰۴)

استاد فصحتی

مقایسه performance چند واسط ارتباطی حوزه‌ی سیستم‌های نهفته

مقدمه

در این پروژه، هدف ما طراحی و پیاده‌سازی یک دستگاه رمزنگار و رمزگشای پیام مبتنی بر الگوریتم AES256-CBC-PAD است که روی بورد نهفته Raspberry Pi Pico W اجرا می‌شود. دستگاه موردنظر پیام ۱۲۸ کاراکتری را از طریق شش واسط ارتباطی مختلف (I2C، SPI، UART، Wi-Fi، CAN، One-Wire) دریافت کرده و به عملیات رمزنگاری یا رمزگشایی می‌پردازد.

این پروژه با تمرکز بر مقایسه عملکرد زمانی این واسط‌های ارتباطی، به بررسی نقاط قوت و ضعف هر کدام در انتقال داده‌های حساس پرداخته و بهینه‌ترین واسط را برای کاربردهای مشابه پیشنهاد می‌دهد.

الگوریتم AES256-CBC-PAD

با توجه به اینکه ما از الگوریتم رمزنگاری AES256-CBC-PAD برای رمزنگاری پیام‌هامان استفاده می‌کنیم، ابتدا به توضیح این الگوریتم و نحوه‌ی کارکرد آن می‌پردازیم. الگوریتم AES (Advanced Encryption Standard) یک استاندارد رمزگذاری متقارن است که برای محافظت از داده‌ها استفاده می‌شود. AES در سال ۲۰۰۱ توسط NIST به عنوان جایگزینی برای DES معرفی شد. AES در سه نسخه ۱۲۸، ۱۹۲ و ۲۵۶ بیتی ارائه می‌شود که AES-۲۵۶ قوی‌ترین نسخه آن است.

ساختار کلی

AES-۲۵۶ داده‌ها را به بلوک‌های **۱۲۸ بیتی** تقسیم کرده و از **کلید ۲۵۶ بیتی** برای انجام **۱۴ دور رمزگذاری** استفاده می‌کند.

مراحل رمزگذاری

مراحل اصلی رمزگذاری در AES-۲۵۶ به شرح زیر است:

۱. توسعه کلید (Key Expansion)

کلید اولیه ۲۵۶ بیتی به چندین کلید فرعی (Keys Round) گسترش داده می‌شود. این کلیدها در هر دور مورد استفاده قرار می‌گیرند.

۲. جایگزینی بایت‌ها (SubBytes)

در این مرحله، هر بایت داده با مقدار جدیدی از جدول **S-Box** جایگزین می‌شود:

$$S(x) = \begin{bmatrix} S_{00} & S_{01} & \dots & S_{0n} \\ S_{10} & S_{11} & \dots & S_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ S_{m0} & S_{m1} & \dots & S_{mn} \end{bmatrix} \quad (۱)$$

۳. جابجایی سطرها (ShiftRows)

در این مرحله، هر سطر ماتریس داده به‌طور چرخشی جابه‌جا می‌شود تا انتشار اطلاعات بهبود یابد.

۴. ترکیب ستون‌ها (MixColumns)

ترکیب ستون‌ها با استفاده از یک ماتریس خاص در **میدان گالوا** به شکل زیر انجام می‌شود:

$$C(x) = A(x) \times M \quad (۲)$$

که در آن:

$$M = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \quad (۳)$$

۵. اضافه کردن کلید دور (AddRoundKey)

در این مرحله، داده‌ها با استفاده از عملیات XOR با کلید دور مربوطه ترکیب می‌شوند:

$$State = State \oplus RoundKey \quad (۴)$$

۶. تکرار مراحل برای ۱۴ دور

تمامی این مراحل برای **۱۴ دور** تکرار شده و در نهایت داده رمزگذاری‌شده تولید می‌شود.

حالت CBC (Cipher Block Chaining)

در این حالت، هر بلوک قبل از رمزگذاری با خروجی بلوک قبلی **XOR** می‌شود تا وابستگی ایجاد شده و امنیت افزایش یابد. معادلات آن به شکل زیر است:

رمزگذاری

$$C_0 = AES_{Encrypt}(P_0 \oplus IV) \tag{۵}$$

$$C_i = AES_{Encrypt}(P_i \oplus C_{i-1}), \quad i \geq 1 \tag{۶}$$

که در آن:

- P_i بلوک iـام از پیام اصلی است.
- C_i بلوک رمزگذاری‌شده‌ی iـام است.
- IV بردار مقداردهی اولیه (Initialization Vector) است.

رمزگشایی

$$P_0 = AES_{Decrypt}(C_0) \oplus IV \tag{۷}$$

$$P_i = AES_{Decrypt}(C_i) \oplus C_{i-1}, \quad i \geq 1 \tag{۸}$$

در CBC، مقدار IV باید **تصادفی و یکتا** باشد تا حملات قابل پیش‌بینی نباشد.

Padding (پدینگ)

چون AES فقط روی بلوک‌های ۱۲۸ بیتی (۱۶ بایتی) کار می‌کند، اگر اندازه‌ی پیام مضربی از ۱۶ نباشد، نیاز به **پدینگ** داریم. رایج‌ترین روش، **PKCS۷** است که مقدار N بایت اضافه می‌کند که مقدار آن برابر با تعداد بایت‌های اضافه‌شده است.

مثال پدینگ

فرض کنید پیام اصلی ۵ بایت باشد (hello). برای رسیدن به ۱۶ بایت، باید ۱۱ بایت 0x0B اضافه شود:

$$\text{hello } 0B\ 0B\ 0B\ 0B\ 0B\ 0B\ 0B\ 0B\ 0B\ 0B\ 0B$$

در هنگام رمزگشایی، این بایت‌های اضافی حذف می‌شوند.

پروتکل I2C

پروتکل I2C (Inter-Integrated Circuit) یک روش ارتباطی سریال و همزمان است که برای ارتباط بین میکروکنترلرها و دستگاه‌های جانبی مانند حسگرها، نمایشگرها و حافظه‌های EEPROM به کار می‌رود. این پروتکل از یک معماری Master-Slave بهره می‌برد و امکان اتصال چندین دستگاه روی یک باس مشترک را با استفاده از آدرس‌های یکتا فراهم می‌کند.

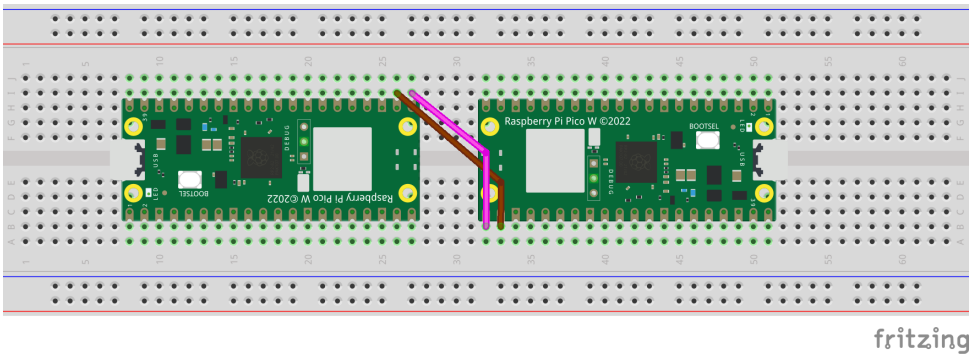
اتصال I2C شامل دو خط اصلی است:

- SDA (Serial Data Line): برای ارسال و دریافت داده بین Master و Slave.
- SCL (Serial Clock Line): برای تولید سیگنال کلاک که انتقال داده را همگام‌سازی می‌کند.

در ارتباط I2C، دستگاه Master کلاک را تولید کرده و ارسال و دریافت داده را کنترل می‌کند، درحالی‌که دستگاه‌های Slave با توجه به آدرس‌دهی منحصربه‌فرد خود، داده‌ها را دریافت یا ارسال می‌کنند. داده‌ها به‌صورت فریم‌های ۸ بیتی منتقل شده و پس از هر بایت، یک بیت تأیید (ACK) یا عدم تأیید (NACK) برای اطمینان از صحت انتقال ارسال می‌شود. از مزایای این پروتکل می‌توان به نیاز به تنها دو خط برای ارتباط چندین دستگاه و مصرف توان کم اشاره کرد، اما سرعت کمتر آن نسبت به SPI و پیچیدگی بیشتر در پیاده‌سازی می‌تواند از معایب آن باشد.

نحوه اتصالات داخلی

نحوه اتصالات داخلی در پروتکل I2C به صورت زیر خواهد بود.

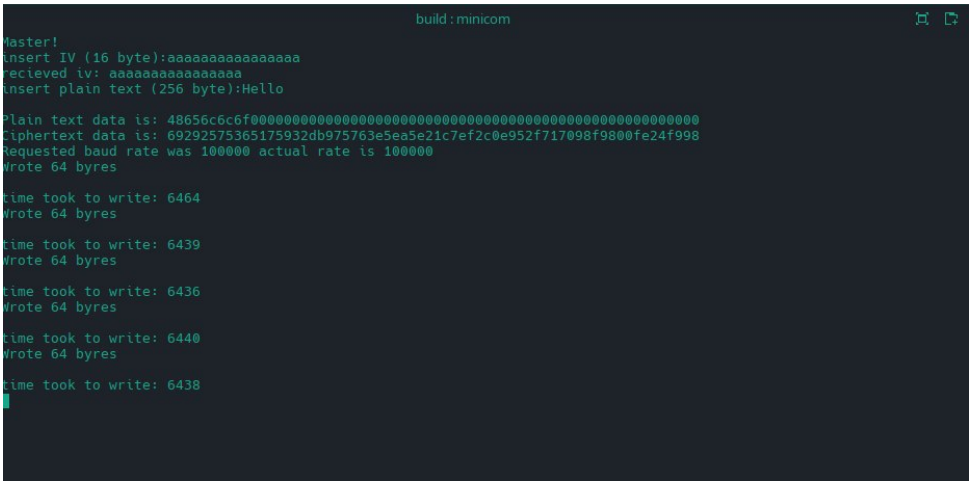


شکل ۱: اتصالات در پروتکل I2C

نتایج

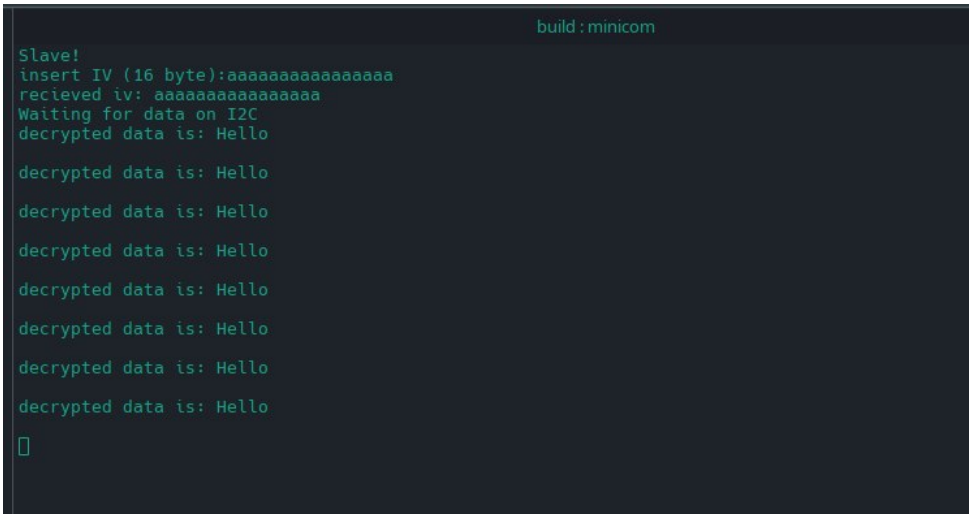
در تصاویر زیر به ترتیب، خروجی سریال Master و Slave نشان داده شده است.

Master:



شکل ۲: خروجی سریال دریافت شده از I2C Master

Slave:



شکل ۳: خروجی سریال دریافت شده از I2C Slave

پروتکل SPI

پروتکل SPI (Serial Peripheral Interface) یک استاندارد ارتباطی همزمان و پرسرعت است که برای انتقال داده بین میکروکنترلرها و دستگاه‌های جانبی مانند سنسورها، حافظه‌های Flash و مبدل‌های آنالوگ به دیجیتال استفاده می‌شود. این پروتکل از یک معماری Master-Slave پیروی می‌کند، به این صورت که یک دستگاه Master وظیفه تولید سیگنال کلاک و مدیریت ارتباط را بر عهده دارد، درحالی‌که یک یا چند دستگاه Slave داده را دریافت یا ارسال می‌کنند.

اتصال SPI معمولاً شامل چهار خط اصلی است:

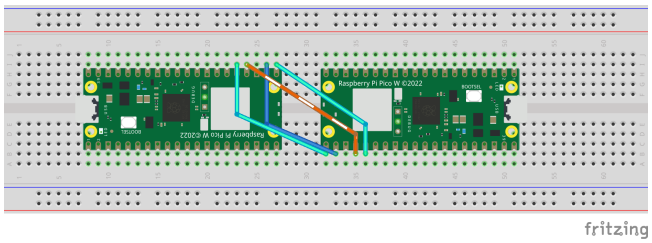
- MOSI (Master Out Slave In): برای ارسال داده از Master به Slave.
- MISO (Master In Slave Out): برای ارسال داده از Slave به Master.

- SCLK (Serial Clock): سیگنال کلاک که توسط Master تولید شده و انتقال داده‌ها را همگام‌سازی می‌کند.
- SS (Slave Select): سیگنالی که Master برای انتخاب یک Slave خاص فعال می‌کند.

در یک ارتباط SPI، داده‌ها به‌صورت همزمان در هر سیکل کلاک بین Master و Slave جابه‌جا می‌شوند، که باعث افزایش سرعت انتقال در مقایسه با پروتکل‌هایی مانند I2C می‌شود. با این حال، یکی از معایب آن، نیاز به خطوط مجزا برای هر Slave است که در ارتباط با چندین دستگاه می‌تواند منجر به استفاده زیاد از پین‌های GPIO شود.

نحوه اتصالات داخلی

نحوه اتصالات داخلی در پروتکل SPI به صورت زیر خواهد بود.

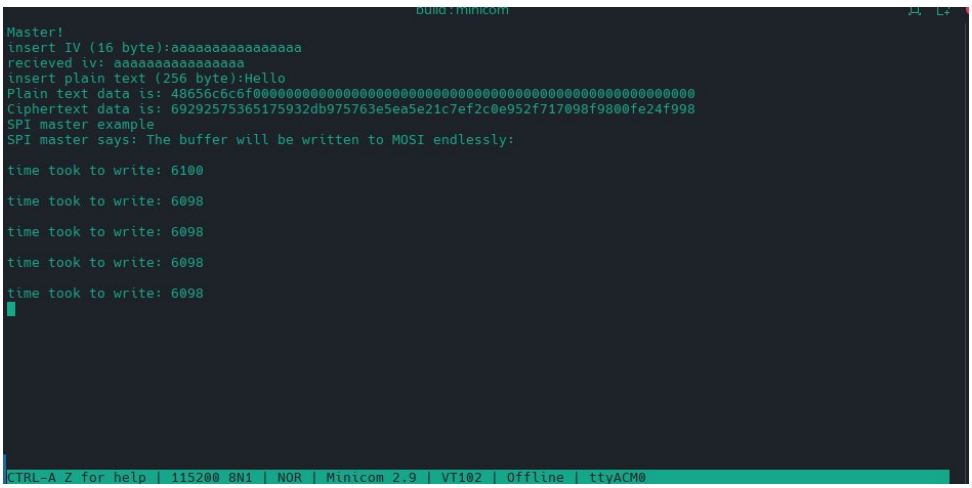


شکل ۴: اتصالات در پروتکل SPI

نتایج

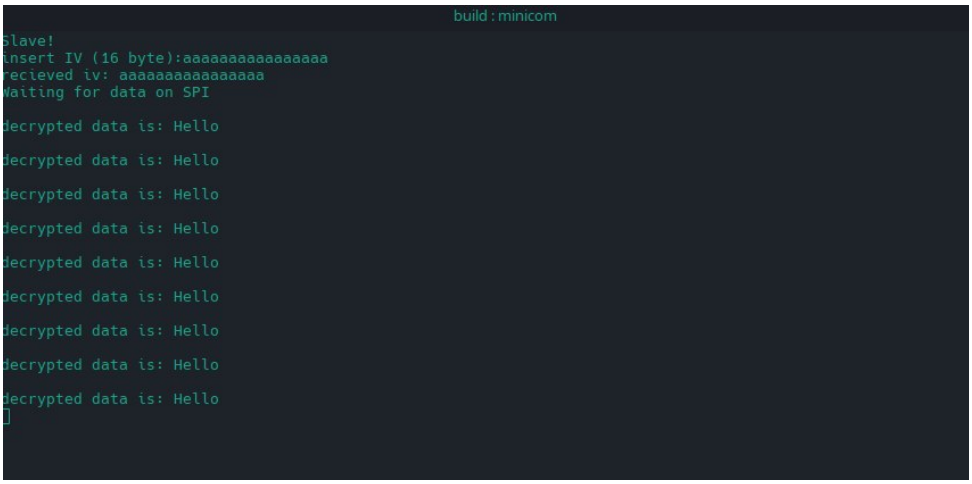
در تصاویر زیر به ترتیب، خروجی سریال Master و Slave نشان داده شده است.

Master:



شکل ۵: خروجی سریال دریافت شده از SPI Master

Slave:



شکل ۶: خروجی سریال دریافت شده از SPI Slave

پروتکل UART

پروتکل UART (Universal Asynchronous Receiver-Transmitter) یک روش ارتباطی سریال و ناهمزمان است که برای تبادل داده بین دو دستگاه، مانند میکروکنترلرها و ماژول‌های ارتباطی، استفاده می‌شود. این پروتکل برخلاف SPI و I2C نیازی به سیگنال کلاک مشترک ندارد و از تنظیمات نرخ انتقال داده (Baud Rate) برای همگام‌سازی داده‌ها بین فرستنده و گیرنده استفاده می‌کند.

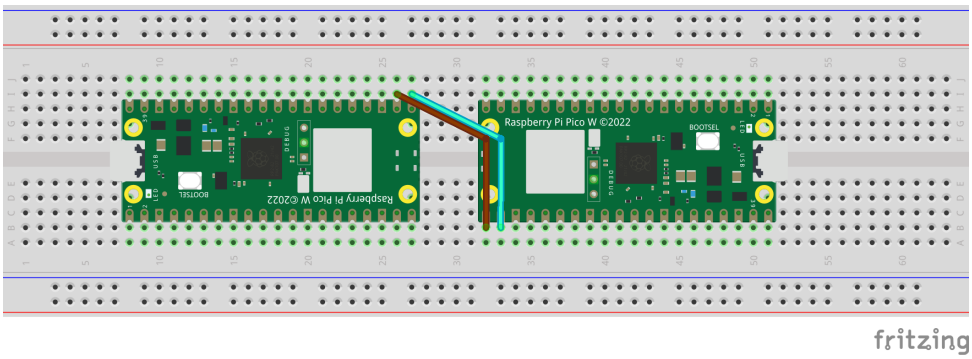
اتصال UART معمولاً شامل دو خط اصلی است:

- TX (Transmit): برای ارسال داده از دستگاه فرستنده.
- RX (Receive): برای دریافت داده در دستگاه گیرنده.

در ارتباط UART، داده‌ها به‌صورت سریالی و بیت‌به‌بیت ارسال می‌شوند. هر فریم داده معمولاً شامل یک بیت شروع (Start Bit)، ۸ بیت داده، یک بیت توازن اختیاری (Parity Bit) برای بررسی صحت داده و یک یا دو بیت توقف (Stop Bit) برای تشخیص پایان داده است. یکی از مزایای این پروتکل، سادگی و نیاز به حداقل تعداد پین‌ها است، اما عدم وجود سیگنال کلاک می‌تواند در صورت عدم تنظیم صحیح نرخ انتقال داده، منجر به بروز خطا در دریافت داده شود.

نحوه اتصالات داخلی

نحوه اتصالات داخلی در پروتکل UART به صورت زیر خواهد بود.

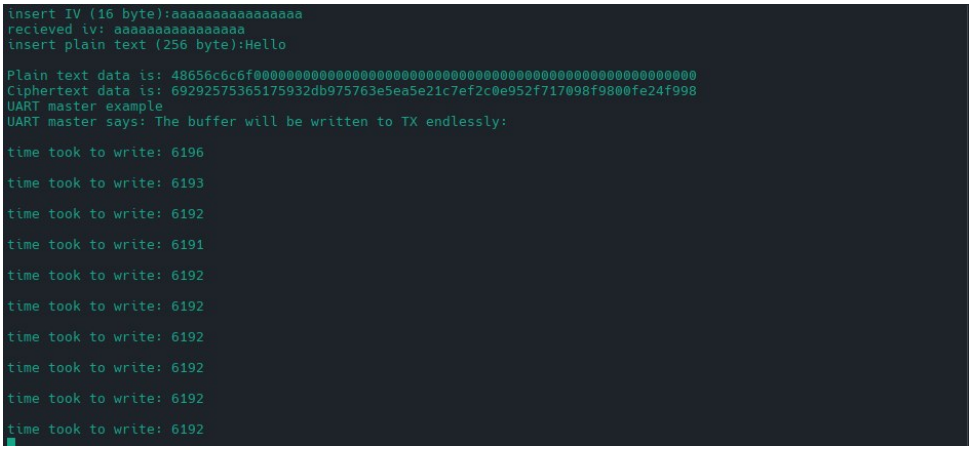


شکل ۷: اتصالات در پروتکل UART

نتایج

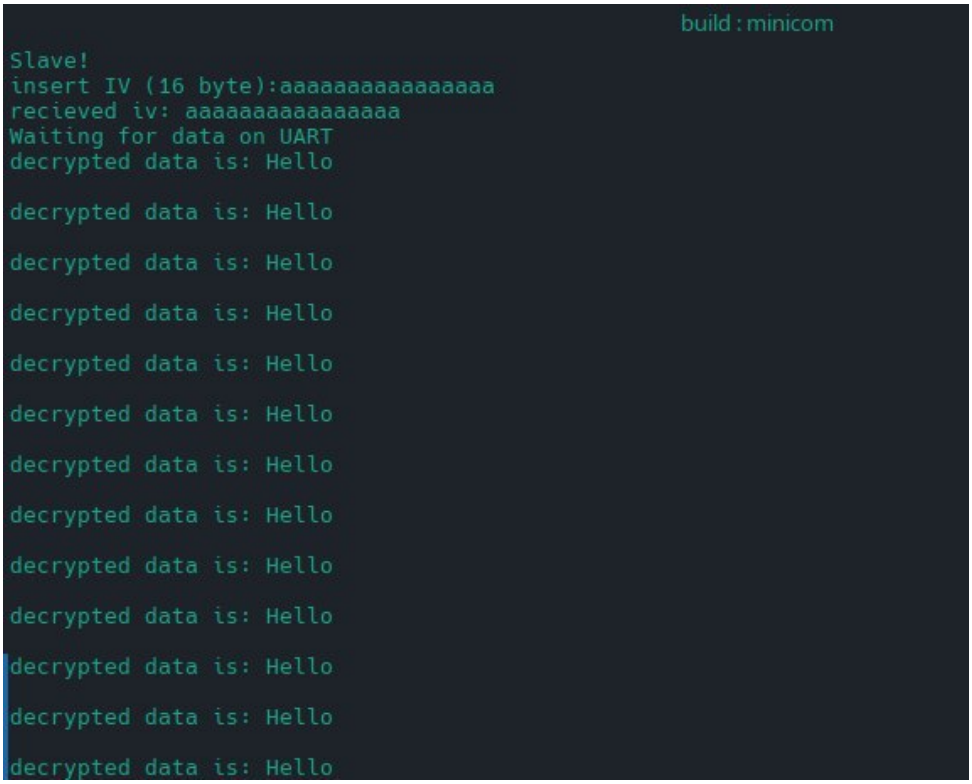
در تصاویر زیر به ترتیب، خروجی سریال Master و Slave نشان داده شده است.

Master:



شکل ۸: خروجی سریال دریافت شده از UART Master

Slave:



شکل ۹: خروجی سریال دریافت شده از UART Slave

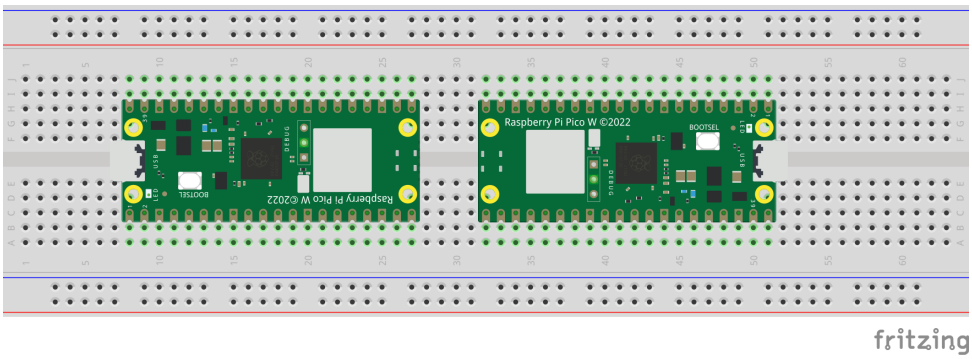
پروتکل WiFi

ماژول Wi-Fi یکی از مهم‌ترین قابلیت‌های Raspberry Pi Pico W است که امکان برقراری ارتباط بی‌سیم را برای این برد فراهم می‌کند. با استفاده از این فناوری، Pico W می‌تواند بدون نیاز به سیم، داده‌ها را ارسال و دریافت کرده و به شبکه‌های محلی یا اینترنت متصل شود. این قابلیت باعث می‌شود که از این برد در پروژه‌های اینترنت اشیا (IoT)، اتوماسیون و سیستم‌های ارتباطی بی‌سیم استفاده شود.

Wi-Fi در Pico W از طریق تراشه Infineon CYW43439 پیاده‌سازی شده است که امکان اتصال پایدار و کم‌مصرف را فراهم می‌کند. به کمک این ویژگی، می‌توان داده‌ها را از طریق پروتکل‌های شبکه به سایر دستگاه‌ها یا سرورها ارسال کرد. این قابلیت نسبت به روش‌های ارتباطی سیمی مانند UART، I2C و SPI، انعطاف‌پذیری بیشتری ارائه می‌دهد و امکان ارتباط در فواصل طولانی‌تر را فراهم می‌کند.

اتصالات داخلی

پروتکل WiFi به صورت بی‌سیم عمل می‌کند لذا اتصالات فیزیکی ندارد.



شکل ۱۰: اتصالات! در پروتکل WiFi

نتایج

در تصاویر زیر خروجی سریال Server و Client مشخص است.

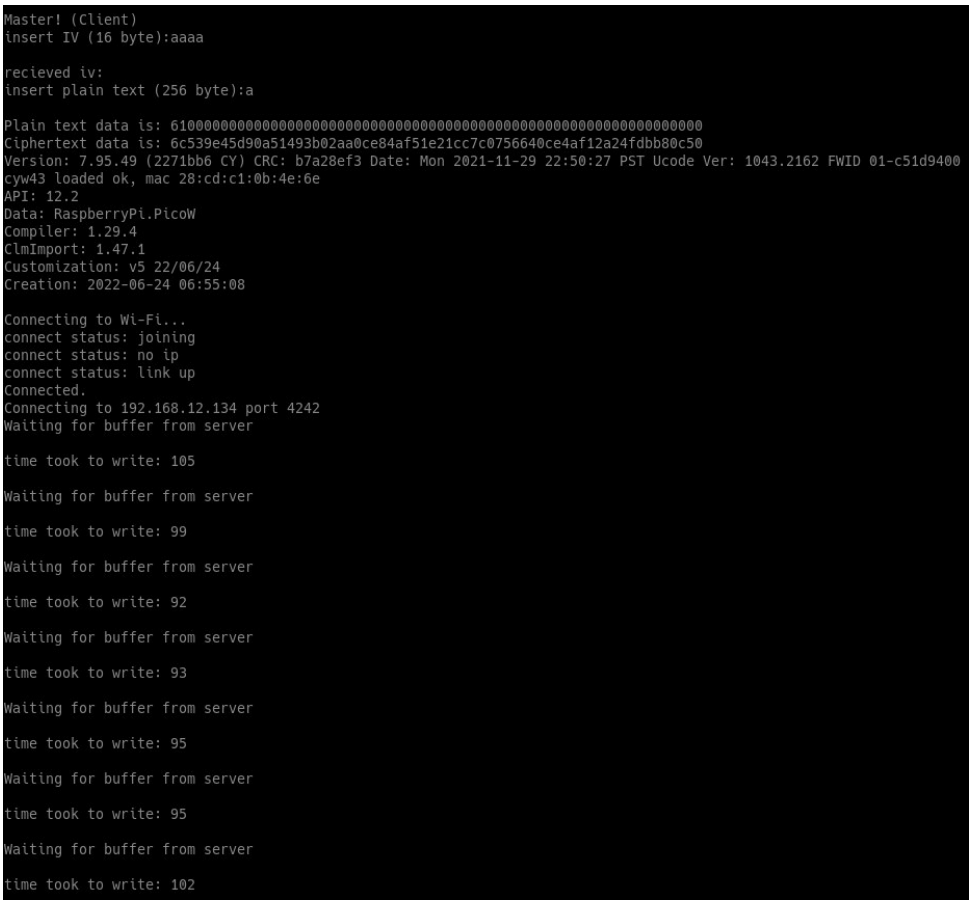
Server:

```
Slave (Server)!
insert IV (16 byte):aaaaaaaaaaaaaa
recieved iv:
Version: 7.95.49 (2271bb6 CY) CRC: b7a28ef3 Date: Mon 2021-11-29 22:50:27 PST Ucode Ver: 1043.2162 FWID0
cyw43 loaded ok, mac 28:cd:c1:05:ea:de
API: 12.2
Data: RaspberryPi.PicoW
Compiler: 1.29.4
ClmImport: 1.47.1
Customization: v5 22/06/24
Creation: 2022-06-24 06:55:08

Connecting to Wi-Fi...
connect status: joining
connect status: no ip
connect status: link up
Connected.
Starting server at 192.168.12.134 on port 4242
Client connected
Received text: az
Received text: az
Received text: az
Received text: az
Received text: az
Received text: az
Received text: az
Received text: az
Received text: az
Received text: az
test success
```

شکل ۱۱: خروجی سریال دریافت شده از WiFi Server

Client:



شکل ۱۲: خروجی سریال دریافت شده از WiFi Client

پروتکل CAN

در این پروژه، قصد داشتیم از پروتکل CAN (Controller Area Network) برای تبادل داده در Raspberry Pi Pico استفاده کنیم. با این حال، پس از بررسی‌های انجام‌شده و مطالعه کتابخانه‌های موجود، از جمله pico-mcp2515، مشخص شد که این برد به‌طور مستقیم از CAN پشتیبانی نمی‌کند و برای پیاده‌سازی آن نیاز به سخت‌افزار اضافی مانند کنترلر MCP2515 وجود دارد. این محدودیت نه‌تنها برای Pico بلکه برای بردهای مشابه مانند Arduino نیز صدق می‌کند.

دلیل اصلی این نیازمندی، نحوه عملکرد پروتکل CAN است که از سیگنالینگ تفاضلی (Differential Signaling) برای انتقال داده‌ها استفاده می‌کند. این پروتکل دارای دو خط CAN_H و CAN_L است که داده‌ها را به‌صورت اختلاف ولتاژ بین این دو خط ارسال می‌کنند. این روش باعث افزایش مقاومت در برابر نویز و بهبود عملکرد در محیط‌های صنعتی می‌شود. اما از آنجا که Raspberry Pi Pico تنها دارای واسط‌های استانداردی مانند UART، I2C و SPI است و قابلیت پردازش سیگنالینگ تفاضلی CAN را ندارد، برای راه‌اندازی آن نیاز به کنترلر خارجی مانند MCP2515 وجود دارد.

با توجه به این محدودیت سخت‌افزاری و نیاز به تجهیزات اضافی، پیاده‌سازی CAN در این پروژه امکان‌پذیر نبود.

پروتکل One-Wire

پروتکل 1-Wire یک روش ارتباطی سریال و ناهمزمان است که برای ارتباط بین میکروکنترلرها و دستگاه‌های جانبی مانند سنسورها، حافظه‌های دیجیتال و تراشه‌های شناسایی استفاده می‌شود. این پروتکل به دلیل استفاده از تنها یک سیم داده، علاوه بر زمین، نیاز به تعداد کمی از پین‌ها دارد و هزینه پیاده‌سازی را کاهش می‌دهد.

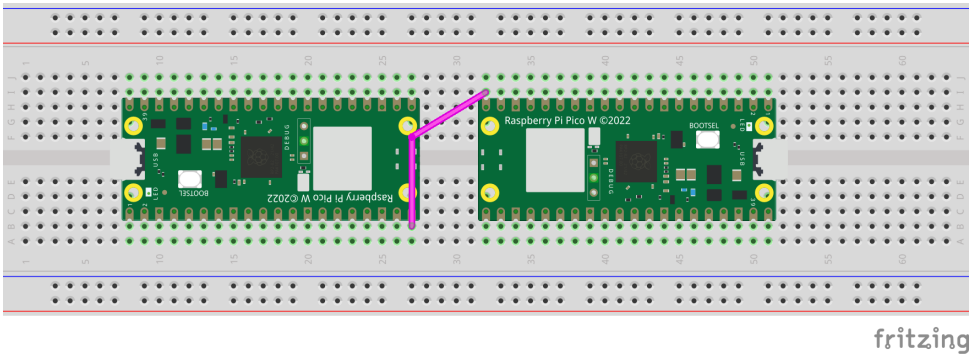
اتصال 1-Wire شامل دو خط است:

- DQ (Data Line): برای ارسال و دریافت داده بین دستگاه‌های متصل.
- GND (Ground): برای مرجع ولتاژ مشترک بین دستگاه‌ها.

در ارتباط 1-Wire، تمامی دستگاه‌ها روی یک باس مشترک متصل شده و هر دستگاه دارای یک شناسه یکتا ۶۴ بیتی است که برای آدرس‌دهی استفاده می‌شود. ارتباط در این پروتکل توسط یک دستگاه Master مدیریت شده و تبادل داده بر اساس ارسال پالس‌های ولتاژی انجام می‌شود. از ویژگی‌های مهم این پروتکل می‌توان به امکان تأمین توان برخی از دستگاه‌ها از طریق همان خط داده (Parasitic Power) اشاره کرد. در مقابل، از معایب آن می‌توان به سرعت پایین‌تر نسبت به پروتکل‌هایی مانند SPI و I2C و حساسیت بالاتر به نویز اشاره کرد.

اتصالات داخلی

نحوه اتصالات داخلی در پروتکل 1Wire به صورت زیر خواهد بود.



شکل ۱۳: اتصالات در پروتکل 1Wire

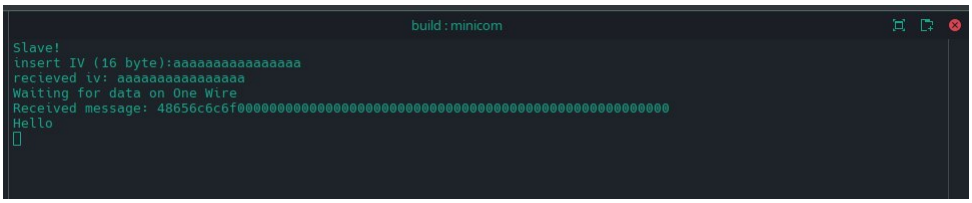
نتایج

در تصاویر زیر خروجی سریال Server و Client مشخص است.

Master:

شکل ۱۴: خروجی سریال دریافت شده از 1Wire Master

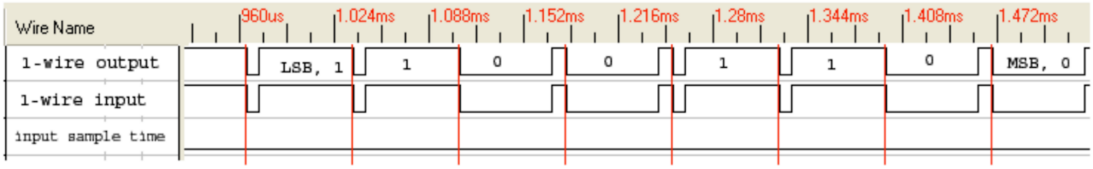
Slave:



شکل ۱۵: خروجی سریال دریافت شده از 1Wire Slave

توضیحی کوتاه در مورد نحوه کارکرد پروتکل One-Wire

برای ارسال بیت به مقدار ۱ روی خط، master باید خط را به مدت ۱ الی ۱۵ میکرو ثانیه صفر کند. همچنین برای ارسال بیت به مقدار ۰ روی خط، master باید خط را به مدت طولانی تر ۶۰ میکرو ثانیه صفر کند. لبه پایین رونده سیگنال باعث شروع یک multivibrator monostable در slave می شود. multivibrator در slave حدود ۳۰ میکرو ثانیه پس از لبه پایین رونده مقدار بیت را می خواند. تایمر داخل slave یک تایمر ارزان با قطعات ساده است که دقت خیلی خوبی ندارد. به خاطر همین است که باید بیت ۰ به مدت ۶۰ میکرو ثانیه باشد و بیت ۱ نباید بیشتر از ۱۵ میکرو ثانیه طول بکشد. تصویر زیر فرایند ارسال داده از master به slave را به تصویر می کشد.



شکل ۱۶: ارسال داده از master به slave در One-Wire

نحوه پیاده‌سازی One-Wire در این پروژه

پروتکل پروتکلیOne-Wire ساده و ارزان است که برای اتصال سنسورهای جانبی کوچک به میکروکنترلرها طراحی شده است. اصولا از این پروتکل برای ارتباط دو میکروکنترلر استفاده نمی‌شود. چراکه پروتکل‌های بهتر و سریع‌تری به عنوان جایگزین برای One-Wire وجود دارد. ما برای پیاده سازی ارتباط One-Wire میان دو میکروکنترلر (رزبری‌های پیکو) مجبور شدیم کد راه‌انداز One-Wire را از صفر بنویسیم. چون در اینترنت کتابخانه مناسبی برای پیاده‌سازی قسمت Slave پروتکل One-Wire نیافتیم. شیوه پیاده‌سازی پروتکل به این صورت است که در قسمت Slave یک تابع interrupt-handler روی لبه پایین رونده خط قرار دادیم. هنگامی که خط دیتا، صفر شود، این تابع اجرا خواهد شد. این تابع به اندازه ۳۰ میکروثانیه صبر می‌کند و سپس خط را می‌خواند. قسمت Master هم مطابق توضیحات قسمت قبل است. برای ارسال بیت یک، خط را به مدت ۱۰ میکروثانیه صفر می‌کنیم و برای ارسال بیت صفر، خط را به مدت ۶۰ میکرو ثانیه صفر می‌کنیم. بدین وسیله توانستیم ارتباط One-Wire را پیاده کنیم. در تصویر زیر می‌توانید کدهای Master و Slave را مشاهده کنید.

```
147     void onewire_master(){
148         gpio_init(DATA_PIN);
149         gpio_set_dir(DATA_PIN, GPIO_OUT);
150         gpio_put(DATA_PIN, true);
151         uint64_t start = time_us_64();
152         for (int i = 0; i < BUF_LEN; i++)
153             for (int j = 7; j >= 0; j--) {
154                 char data = (char)plain[i];
155                 char bitVal = (data >> j) & 0x01;
156                 if (bitVal == 0) {
157                     gpio_put(DATA_PIN, false);
158                     sleep_us(60);
159                     gpio_put(DATA_PIN, true);
160                     sleep_us(10);
161                 } else {
162                     gpio_put(DATA_PIN, false);
163                     sleep_us(10);
164                     gpio_put(DATA_PIN, true);
165                     sleep_us(60);
166                 }
167             }
168         uint64_t end = time_us_64();
169         printf("\n time took to write: %lld\n", end - start);
170     }
```

شکل ۱۷: کد Master برای ارسال داده

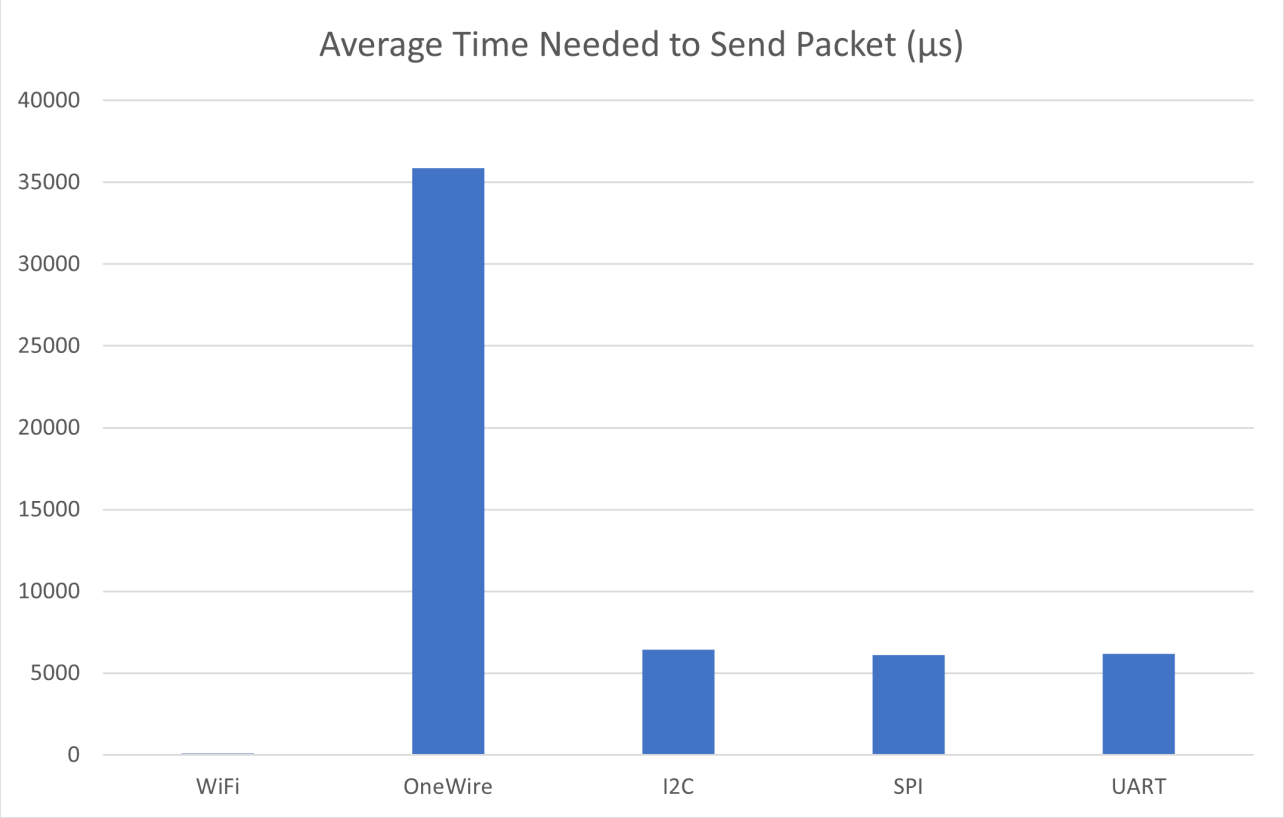
```
186     static void onewire_handler(uint gpio, uint32_t events){
187         busy_wait_us(30);
188         bool bitVal = gpio_get(DATA_PIN);
189         currentByte = (currentByte << 1) | bitVal; // Build up the byte from the incoming bits.
190         bitCount++;
191         if (bitCount % 8 == 0){
192             if (byteIndex < BUF_LEN)
193                 onewire_in_buf[byteIndex++] = currentByte;
194             currentByte = 0;
195         }
196         if (byteIndex >= BUF_LEN){
197             messageReceived = true;
198             gpio_set_irq_enabled_with_callback(DATA_PIN, GPIO_IRQ_EDGE_FALL, false, &onewire_handler);
199         }
200     }
201 
202     static void onewire_slave(){
203         printf("Waiting for data on One Wire\n");
204         gpio_init(DATA_PIN);
205         gpio_set_dir(DATA_PIN, GPIO_IN);
206         gpio_set_irq_enabled_with_callback(DATA_PIN, GPIO_IRQ_EDGE_FALL, true, &onewire_handler);
207         while(true){
208             if(messageReceived){
209                 messageReceived = false;
210                 AES_CBC_decrypt_buffer(&ctx, onewire_in_buf, BUF_LEN);
211                 printf("Received message: ");
212                 phex(onewire_in_buf, 32);
213                 printf("%s\n", onewire_in_buf);
214                 AES_ctx_set_iv(&ctx, iv);
215                 memset(onewire_in_buf, 0, BUF_LEN);
216                 gpio_set_irq_enabled_with_callback(DATA_PIN, GPIO_IRQ_EDGE_FALL, true, &onewire_handler);
217             }
218         }
219     }
```

شکل ۱۸: کد Slave برای دریافت داده

همچنین در گیت‌هاب پروژه برای راه‌اندازی One-Wire توسط دو پلتفرم Arduino و ESP۸۲۶۶ مثال آوردیم. می‌توانید آن‌ها را بررسی کنید.

مقایسه‌ی پروتکل‌ها

در نهایت برای ۵ پروتکل پیاده‌سازی شده داده‌ها را ثبت می‌کنیم و میانگین زمان را محاسبه می‌کنیم. نمودار زیر این مقایسه‌را نشان می‌دهد:



شکل ۱۹: مقایسه‌ی بین پروتکل‌ها

همان‌طور که نتایج نشان می‌دهند، پروتکل OneWire بیشترین زمان را برای انتقال داده نیاز دارد، در حالی که WiFi کمترین زمان را دارد. علت این تفاوت را می‌توان در عوامل زیر جستجو کرد:

- **WiFi**: این پروتکل از ارتباط بی‌سیم و نرخ انتقال داده بسیار بالاتر نسبت به سایر پروتکل‌ها بهره می‌برد، که دلیل اصلی زمان انتقال بسیار کم آن است.
- **OneWire**: این پروتکل به دلیل پایین بودن rate خواندن و نوشتن روی خط (هر ۷۰ میکروثانیه یک بیت روی خط می‌رود) دارای کمترین نرخ انتقال شد.
- **I۲C**: این پروتکل ارتباطی سریال با دو سیم دارای سرعت انتقال بالاتری نسبت به OneWire است، اما به دلیل داشتن فرآیندهای دسترسی به باس و تأیید دریافت داده، از SPI و UART کندتر است.
- **SPI**: این پروتکل از چهار سیم برای انتقال داده استفاده می‌کند که امکان ارسال همزمان داده را فراهم کرده و در نتیجه سرعت بالاتری نسبت به I2C دارد. در عین حال، نیاز به مدیریت چندین خط ارتباطی می‌تواند پیچیدگی آن را افزایش دهد.
- **UART**: این پروتکل از ارتباط سریال غیرهمگام استفاده می‌کند که باعث می‌شود سرعت آن نسبت به SPI کمتر باشد. با این حال، به دلیل سادگی در پیاده‌سازی و نیاز نداشتن به خطوط متعدد، کاربرد گسترده‌ای دارد.

نتایج به‌دست‌آمده نشان می‌دهند که برای کاربردهایی که به انتقال سریع داده نیاز دارند، WiFi یا SPI گزینه‌های بهتری هستند، در حالی که

OneWire تنها برای مواردی که تعداد سیم‌های محدود اهمیت دارد، مناسب است.