دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی

پیاده‌سازی ارتباط باس CAN FD بین میکروکنترلر و FPGA مبتنی بر پروتکل CANOpen

نگارش
محمد چمن مطلق

استاد راهنما
دکتر محمدمهدی همایونپور

بهمن ۱۴۰۰

چکیده

ارتباطات بین اجزا سیستم‌های نفهته همواره موضوع بسیار مهمی بوده است و تلاش‌های فراوانی در جهت طراحی ارتباطات استاندارد وجود دارد. برخی از این استانداردها صرفا یک ارتباط فیزیکی را مشخصی می‌کنند، برخی یک پروتکل سطح بالا هستند و برخی نیز شامل هر دو قسم می‌شوند. در این میان می‌توان به باس‌هایی نظیر I^2C، SPI و CAN اشاره نمود. باس CAN یک باس استاندارد شناخته شده در صنایع خودروسازی، اتوماسیون، صنایع هوایی، فضایی و دیگر صنایع مرتبط است و ویرایش‌های مختلفی از این باس توسعه یافته است. CAN FD یک ویرایش خاص از باس CAN می‌باشد که توانایی افزایش نرخ انتقال بدون تغییر لایه فیزیکی CAN را فراهم می‌آورد. با توجه به نوین بودن CAN FD، تلاش‌های کمتری در راستای پیاده‌سازی و استفاده از این باس صورت گرفته است. هدف نهایی این پروژه، پیاده‌سازی یک ارتباط دو طرفه با بهره‌وری همزمان از باس‌های CAN و CAN FD به کمک یک میکروکنترلر و یک FPGA است. بنابراین در راستای انجام این پروژه، دو نوع پیاده‌سازی (میکروکنترلر و FPGA) توسعه یافته است که ارتباطی بین FPGA و میکروکنترلر از طریق CAN ایجاد کرده و در ادامه ارتباطی مبتنی بر باس CAN FD توسط IP-Core این باس در FPGA پیاده‌سازی شده است. این پیاده‌سازی در قالب دو پیکربندی متفاوت انجام شده‌اند و در نهایت پس از برقراری ارتباط، صحت انتقال مورد بررسی قرار گرفته و ارتباطات موجود از جنبه‌های مختلفی نظیر میزان ترافیک، نرخ بیت، تحمل نویز و درصد خطا مورد ارزیابی قرار گرفته‌اند. همچنین، تلاش شده است که انتقال داده میان قطعات مبتنی بر پروتکل CANOpen صورت گیرد.

واژه‌های کلیدی:

واسط ارتباطی، باس داده، سیستم نهفته، میکروکنترلر، FPGA

فهرست مطالب

صفحه

فهرست اشکال

صفحه

فهرست جداول

صفحه

فصل اول

مقدمه

مقدمه

۱-۱- تعریف مساله

بی‌شک یکی از مهم‌ترین کارکردهای هر روزه‌ی مهندسی، ارتباطات بین اجزاء مختلف است. درحالی که امروزه کامپیوترها فعالیت‌های بیشتر و بیشتری را به عهده می‌گیرند، اهمیت تبادل داده در حال افزایش می‌باشد. تبادل داده بین کاربران (چه عوامل انسانی و چه عوامل غیر انسانی) اجازه بازیابی یا ضبط داده، تکثیر و نسخه‌برداری داده و البته اجرای عملیات‌ها به شکل غیر متمرکز را می‌دهد [۱]. در راستای به‌کارگیری همین مزایا، واسط‌های ارتباطی مختلفی نظیر SPI، I^2C، UART و... پیاده‌سازی و استاندارد شده‌اند که البته هرکدام ویژگی‌های منحصر به فرد خود و مزایا و معایبی دارند که آن‌ها را مناسب کاربردهایی خاص ساخته است.

باس CAN[1] که نخستین بار در سال ۱۹۸۶ توسط شرکت آلمانی بوش[2] در کنگره‌ی جامعه مهندسان خودرویی[3] ارائه شد [۲]، تاکنون توجهات زیادی را به خود جلب نموده است. با وجود اینکه این باس نخستین بار به منظور استفاده در صنایع خودرویی معرفی شد [۲]، ویژگی‌های منحصر به فرد این باس، آن را برای کاربردهای زیادی در صنایع مختلف به خصوص صنایع هوا و فضا مناسب کرده است [۳]. میزان کاربرد و اهمیت این باس در طول سال به قدری افزایش یافت که جزئیات پیاده‌سازی و ارزیابی این باس، در طول سال‌های ۲۰۰۳ تا ۲۰۱۵ میلادی در قالب ۶ استاندارد سازمان بین‌المللی ارائه گردید

[1] Controlled Area Network
[2] Robert Bosch GmbH
[3] Society of Automotive Engineers (SAE) congress

[۴]. در پی افزایش کاربردهای باس CAN، نسخه‌های دیگری از این باس ارائه شدند که در این میان می‌توان به CAN FD[1] اشاره نمود [۲].

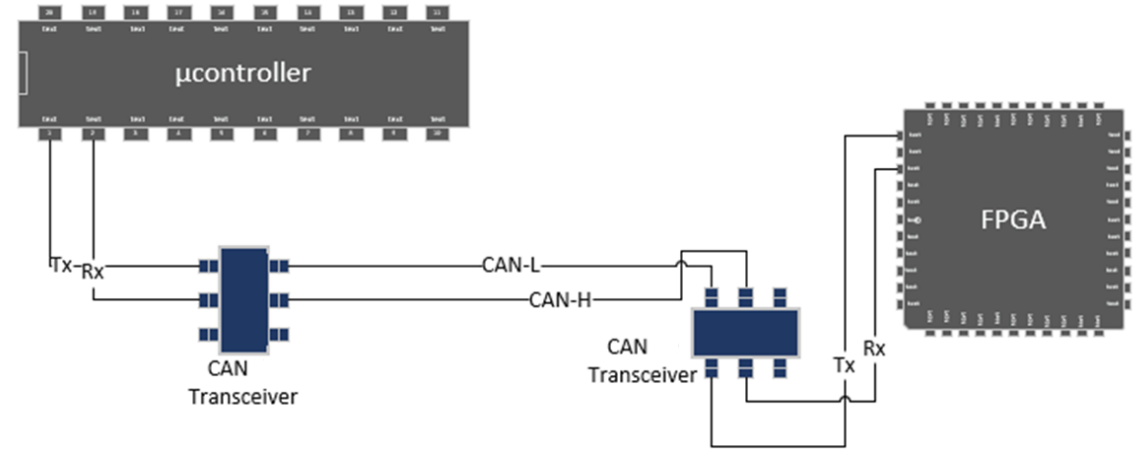
باس CAN FD نخستین بار در سال ۲۰۱۲ عرضه شد و هدف اصلی معرفی این ویرایش خاص از باس CAN، پشتیبانی از نرخ‌های انتقال بالا همزمان با افزایش اطمینان‌پذیری این باس بود [۱۰]. باس CAN FD با قیمتی مشابه به CAN استاندارد، سرعتی چند برابر آن را ارائه می‌کند [۱۱]. در راه طراحی CAN FD، تلاش بر این بود که لایه فیزیکی CAN حفظ شده و تنها لایه‌های بالاتر تغییر کنند، بنابراین CAN FD را می‌توان یک پروتکل برای لایه فیزیکی CAN استاندارد نیز دانست.

با وجود تمام خواص ارزشمند باس CAN و همچنین نسخه بهبود یافته آن یعنی CAN FD، این باس تنها ارتباط در لایه‌های پایین‌تر را فراهم می‌کند و روشی برای ارتباط در لایه‌های بالاتر و به ویژه لایه کاربرد[2] فراهم نمی‌کند [7]. بنابراین برای استفاده موثر از این باس در کاربردهایی با مقیاس بالا، نیاز به پروتکل‌هایی می‌باشد که ویژگی‌های لایه کاربرد را فراهم کنند. یکی از این پروتکل‌ها، پروتکل CANOpen می‌باشد که امروزه به طور گسترده در صنایع اتوماسیون مورد استفاده قرار می‌گیرد. این پروتکل قابلیت‌های نظیر تقطیع بسته، همگام‌سازی شبکه و مدیریت گره‌ها را فراهم می‌آورد [8].

هدف این پروژه، پیاده‌سازی یک ارتباط از طریق CAN بین یک میکروکنترلر و یک FPGA و همچنین پیاده‌سازی باس CAN FD به صورت حلقه بازگشتی[3] درون FPGA می‌باشد. همانطور که پیش‌تر اشاره شد، باس CAN استاندارد دارای سه لایه شی، انتقال و فیزیکی می‌باشد و CAN FD با استفاده از لایه فیزیکی باس CAN، به پیاده‌سازی دو لایه بالاتر می‌پردازند. این لایه فیزیکی از طریق یک جفت سیم

[1] Flexible Date rate CAN
[2] Application layer
[3] Loopback

درهم تنیده (رابط باس CAN) و دستگاه‌های فرستنده-گیرنده[1] انجام می‌شود. در نهایت نیز ارتباط بین اجزاء از طریق پروتکل CANOpen انجام خواهد شد. نمودار بلوکی اجزاء این پروژه، مطابق شکل ۱-۱ می‌باشد.



شکل ۱-۱: نمودار بلوکی اجزاء پروژه.

۱-۲- ضرورت و اهداف پروژه

همانطور که ذکر شد، باس CAN FD تاریخچه کوتاه‌تری از CAN استاندارد دارد و به طبع در تحقیقات و صنایع مختلف، توجه کمتری به آن شده است. با توجه به اینکه CAN FD به صورت بالقوه می‌تواند با قیمتی یکسان با CAN استاندارد، کارایی بالاتری ارائه دهد [٦] و عملا بدون ایجاد هزینه اضافی جایگزین باس CAN شود، وجود یک روش ساختارمند برای کار با CAN FD بسیار ارزشمند خواهد بود.

خروجی نهایی این پروژه، می‌تواند به صورت یک کتابخانه نرم‌افزاری (برای میکرکنترلر) و یک IP-Core (برای FPGA) درآید که هر کدام از این موارد به خودی خود می‌توانند استفاده از CAN FD را در

[1] Transceiver

آینده تسهیل نمایند. این فرایند می‌تواند به رشد استفاده از CAN FD در صنایع مختلف و در نتیجه بهبود کیفیت ارتباطات سریال و در نتیجه افزایش کارایی ادوات مختلف، کمک بسزایی نماید.

۱-۳- ساختار گزارش

در فصل ابتدایی این گزارش، مقدمه‌ای اجمالی بر روی طرح انجام گرفته، اهداف دنبال شده و ضرورت اجرای این طرح ارائه شد. در فصل دوم، به طور تفضیلی به ویژگی‌های باس CAN، کاربردهای آن و مقایسه آن با سایر باس‌های مطرح در سیستم‌های نفهته پرداخته شده است. همچنین به طور دقیق‌تر تفاوت‌های برخی نسخه‌های دیگر باس CAN و به ویژه CAN FD با باس CAN بررسی شده است و در پایان نیز پروتکل‌های لایه بالاتر به خصوص CANOpen مورد بررسی قرار گرفته است. در فصل سوم، پیاده‌سازی انجام شده در میکروکنترلر و FPGA بیان شده است. این موضوع شامل بررسی قطعات و کتابخانه‌های نرم‌افزاری مورد استفاده و همچنین ساختار سخت‌افزاری و ساختار نرم‌افزاری پیاده‌سازی شده می‌شود. در فصل چهارم، پیاده‌سازی انجام شده مورد ارزیابی قرار گرفته است و عملکرد طرح بررسی شده است. فصل پایانی نیز به بیان نتایج و پیشنهادات حاصل از انجام این طرح پرداخته است.

فصل دوم

باس CAN

باس CAN

باس سریال CAN نخستین بار در سال ۱۹۸۶ توسط شرکت بوش به صورت یک سیستم همه‌پخشی[1]، با قابلیت پشتیانی از چندین راهبر[2] و به صورت آسنکرون[3] توسعه پیدا کرد. هدف اولیه این باس که بعدها در قالب استاندارد ISO 11898 قرار گرفت، جایگزینی این باس در صنایع خودروسازی به دلیل مشکلات پیچیدگی و سیم‌کشی باس‌های دو-سیمه موجود بود [۹]. در ادامه، این باس محبوبیت زیادی در بین واحدهای کنترل، حسگرها، سیستم‌های ضد لرزش و سایر واحدهای کنترل الکتریکی[4] پیدا کرد [۱۰]. حداکثر نرخ ارسال این باس برابر ۱ Mbps می‌باشد که مقاومت آن در برابر اختلالات الکتریکی و قابلیت شناسایی و تصحیح خطاها به صورت خودکار، این باس را مناسب برای کاربردهای ذکر شده نموده است. طبق استاندارد ISO 11898، مشخصات این باس در دو بخش A و B و دو ویرایش یک و دو منتشر شده است [۹]. به طور کلی، می‌توان مزایای کلیدی باس CAN را در چند مورد اساسی خلاصه نمود:

- سادگی و پیاده‌سازی ارزان قیمت
- متمرکز[5] بودن
- اطمینان‌پذیری بالا
- کارایی مناسب [۷]

علل وجود هر یک از این مزیت‌ها در ادامه مشخص خواهد شد.

با وجود اینکه این باس نخستین بار به منظور استفاده در صنایع خودرویی معرفی شد [۲]، ویژگی‌های منحصر به فرد این باس، آن را برای کاربردهای زیادی در صنایع مختلف به خصوص صنایع اتوماسیون هوا و فضا مناسب کرده است [۲]، [۳] و [۱۶]. پس از توسعه ویرایش اولیه این باس برای خودروهای

[1] Broadcast
[2] Master
[3] Asynchronous
[4] Electronic Control Unit (ECU)
[5] Centralized

دارای سرنشین، از این باس به عنوان پروتکل ارتباطی برای کاربردهایی نظیر سیستم‌های کنترل آسانسور، مدیریت ماشین آلات نساجی و همچنین دستگاه‌های تولید اشعه X-Ray مورد استفاده قرار گرفت [۲].

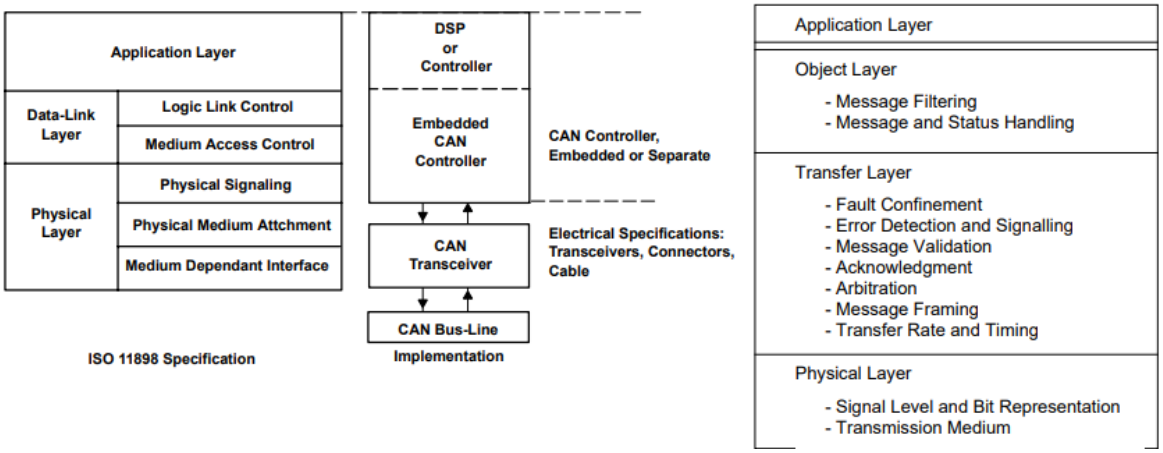
۲-۱- ویژگی‌های فنی

معماری یک باس CAN از سه لایه تشکیل می‌شود که عبارتند از:

- لایه شی[1] CAN
- لایه انتقال[2] CAN
- لایه فیزیکی CAN

در بین این موارد، مجموعه دو لایه شی و انتقال، معادل لایه لینک داده در مدل مرجع ISO/OSI هستند. لایه شی مسئول انتخاب پیام، انتخاب زمان ارسال آن و در نهایت ارتباط با لایه‌های بالاتر است. لایه انتقال نیز مسئوال اجرای پروتکل انتقال است که شامل موارد نظیر داوری، تشخیص خطا و کنترل قالب‌های ارسالی می‌شود. لایه فیزیکی نیز نمایانگر بستر حقیقی انتقال بیت‌های داده است [۱۰]. شکل ۲-۱، ساختار معماری این باس را بهتر نمایان می‌کنند.

[1] Object Layer
[2] Transfer layer



شکل ۲-۱: مدل لایه‌ای باس CAN طبق استاندارد پیاده‌سازی [۹] – [۱۰].

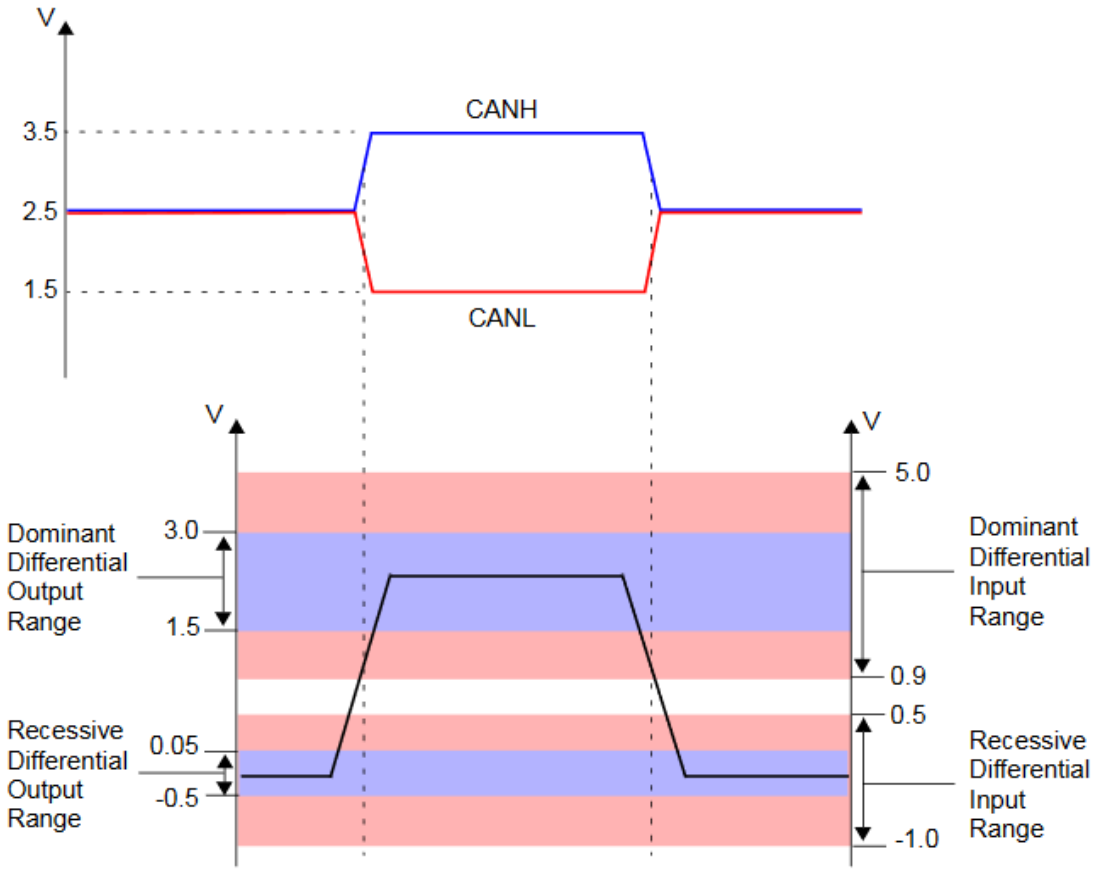
۲-۱-۱- لایه فیزیکی

لایه فیزیکی باس، رابط انتقال به صورت یک جفت سیم درهم تنیده[1] می‌باشد. سیم‌های موجود در این جفت که CANH و CANL نامیده می‌شوند، مسئول انتقال داده باس CAN هستند. سیگنال‌ها به صورت تفاضلی[2] و متعادل[3] روی این سیم‌ها جابجا می‌شوند، که این موضوع عامل اصلی خنثی‌سازی اثر نویزهای محیطی می‌باشد [۱۰]. روش انتقال سیگنال تفاضلی، روشی متداول است که در باس‌های پرکاربرد دیگری مانند USB و مورد استفاده قرار گرفته است [۱۱].

طبق استاندارد، در لایه فیزیکی باس CAN، دو نوع وضعیت سیگنال متفاوت تعریف شده است: وضعیت غالب[4] (سیگنال صفر منطقی) و وضعیت مغلوب[5] (سیگنال یک منطقی). همانطور که ذکر شد، این سیگنال‌ها توسط دو رشته سیم در هم تنیده ارسال می‌شوند و تفاوت ولتاژ این دو سیم (V_{DIFF})، وضعیت سیگنال انتقالی را مشخص می‌نماید. برابری ولتاژ CAN_L و CAN_H به معنی وضعیت مغلوب و اختلافی بیش از مقدار از پیش تعیین شده (عموما برابر ۱.۵V) نشان‌دهنده وضعیت غالب می‌باشد [۱۲]. شکل ۲-۲، این مساله را به تصویر می‌کشد.

[1] twisted-pair wires
[2] Differential
[3] Balanced
[4] Dominant
[5] Recessive

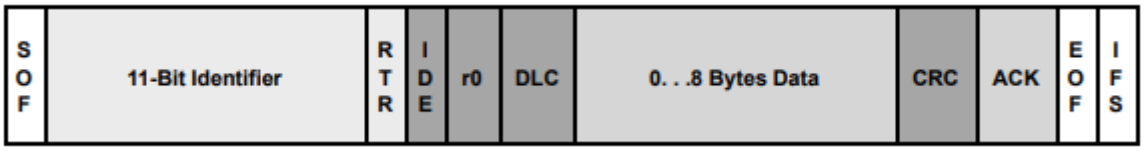
استاندارد ISO-11898، حداکثر طول کابل مناسب باس CAN برای ارسال با نرخ بیتی ۱Mbps را برابر ۴۰ متر اعلام کرده است [٤]. البته که استفاده از کابل‌های طولانی‌تر نیز برای این باس امکان پذیر است، ولی در صورت استفاده از کابل‌های طولانی‌تر، نرخ بیتی به شکل قابل ملاحظه‌ای کاهش می‌یابد. به عنوان یک مثال، یک باس CAN با طول کابل برابر ۵۰۰ متر، حداکثر نرخ بیتی ۱۲۵Kbps را خواهد داشت [۱۳].



شکل ۲-۲: تشخیص وضعیت سیگنال منتقل شده از روی اختلاف ولتاژ CAN_L و CAN_H [۱۳].

۲-۱-۲- لایه لینک داده

لایه لینک داده در باس CAN، وظایفی نظیر قالب‌بندی داده‌ها، داوری[1]، تشخیص خطا وکنترل دسترسی به باس را بر عهده دارد [۱۰]. همانطور که در شکل ۲-۱ نمایش داده شد، لایه لینک داده از دو زیرلایه به نام‌ها کنترل لینک منطقی[2] (LLC) و کنترل دسترسی به واسط[3] (MAC) تشکیل شده است. زیر لایه LLC مسئول ارائه خدمات تصدیق داده، ایجاد اطلاعیه‌ها و مدیریت بازیابی می‌باشد. زیر لایه MAC نیز وظیفه کپسوله‌سازی داده‌ها، کدگذاری قاب‌ها، مدیریت دسترسی به واسط، تشخیص و اطلاع خطا و همچنین مدیریت ACKها می‌باشد [۱٤]. قاب داده[4] باس CAN در شکل ۲-۳ قابل مشاهده است و توضیحات مربوط به هر یک از بخش‌های این قاب در ادامه بیان شده است.



شکل ۲-۳: قاب داده باس CAN 1.0 [۷].

SOF: بخشی یک بیتی است که شروع انتقال بسته را مشخص می‌کند و عموما به صورت بیت صفر غالب است.

Identifier: یک شناساگر منحصر به فرد پیام است که علاوه بر مشخص کردن نوع پیام، وظیفه اصلی آن مشخص کردن اولویت پیام است. کمتر بودن مقدار این بخش به معنای اولویت بالاتر پیام است و در مرحله داوری، پیام‌هایی که اولویت بالاتری دارند پیش از سایر پیام‌ها ارسال می‌شوند. تفاوت اصلی استاندارد CAN 2.0 با CAN 1.0 در طول همین بخش است که برای CAN 1.0 برابر ۱۱ بیت و برای CAN 2.0 برابر ۲۹ بیت می‌باشد.

[1] Arbitration
[2] Logical Link Control
[3] Medium Access Control
[4] Data Frame

RTR (Remote Transmission Request): بخشی یک بیتی است که نشانگر این است که پیام ارسالی درخواست ارسال داده است (بیت یک منطقی) و یا خود داده می‌باشد (بیت صفر منطقی).

IDE (Identifier extension bit): مقداری یک بیتی که مشخص می‌کند شناساگر ۱۱ بیتی است (صفر منطقی) و یا ۲۹ بیتی (یک منطقی).

r0: مقدار یک بیتی رزرو شده است که می‌تواند هر مقداری بپذیرد، ولی به صورت متداول برابر صفر منطقی قرار می‌گیرد.

DLC (Data length code): مقداری چهار بیتی که تعداد بایت‌های داده را نشان می‌دهد.

Data: محموله داده اصلی هر بسته که حداکثر برابر هشت بایت می‌باشد.

CRC: کد افزونگی چرخشی داده ارسالی که طول آن ۱۵ بیت می‌باشد (به همراه یک بیت اضافه برای مشخص شدن پایان CRC).

ACK: مقداری ۱ بیتی برای ارسال و دریافت ACK می‌باشد. فرستنده پیام مقدار یک مغلوب را درون این بخش می‌نویسد و گیرنده نیز پس از دریافت پیام، مقدار صفر غالب را درون این بخش قرار می دهد.

EOF: نشانگر پایان یک بسته می‌باشد و همواره مقداری یک بیتی برابر یک منطقی می‌باشد.

IFS (Inter Frame Spacing): بخشی سه بیتی است که به منظور ایجاد فاصله بین پیام‌های متوالی قرار داده می‌شود و تمامی بیت‌های آن برابر یک منطقی می‌باشند [۱۰].

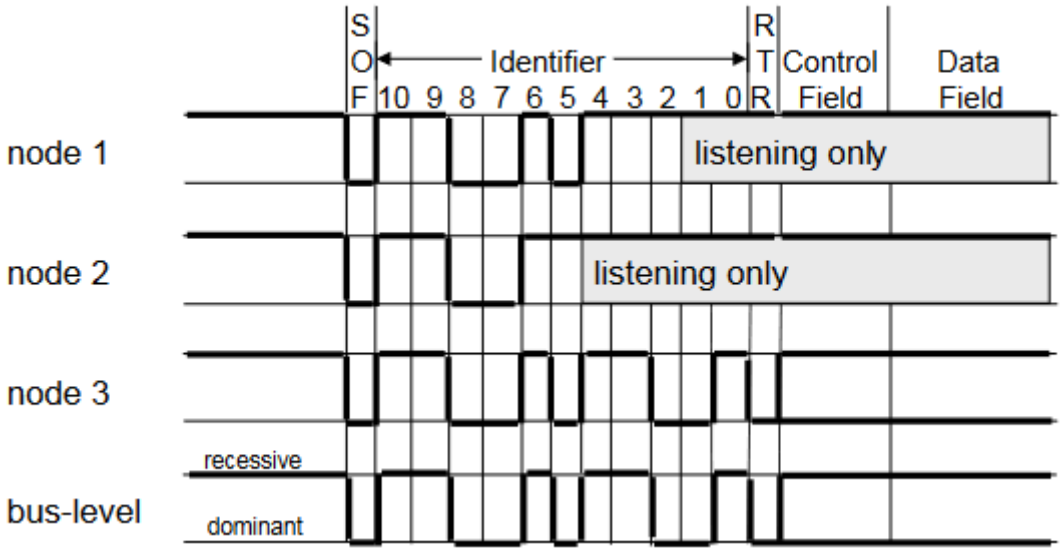
البته که در یک ارتباط CAN لزوما تمامی بخش‌های ذکر شده مورد استفاده قرار نمی‌گیرند، بلکه بسته به کاربری بسته، ممکن است هر یک از بخش‌ها به صورت انتخابی استفاده شوند. بسته‌های CAN به طور کلی می‌توانند چهار کاربرد متفاوت داشته باشند:

۱. بسته داده: این نوع بسته به منظور ارسال یک محموله از داده مورد استفاده قرار می‌گیرد.

۲. بسته راه دور[1]: هدف از ارسال این نوع بسته‌ها، درخواست داده‌ای خاص می‌باشد.

۳. بسته خطا: در صورتی که خطایی در ارسال و یا دریافت داده ایجاد شود، گره مربوطه این نوع پیام را روی باس مخابره می‌کند.

۴. بسته اضافه بار[2]: هر گره می‌تواند با ارسال این نوع بسته روی باس، از گره‌های دیگر تقاضا کند که برای ارسال بسته‌های بعدی مدتی صبر کنند [۷].

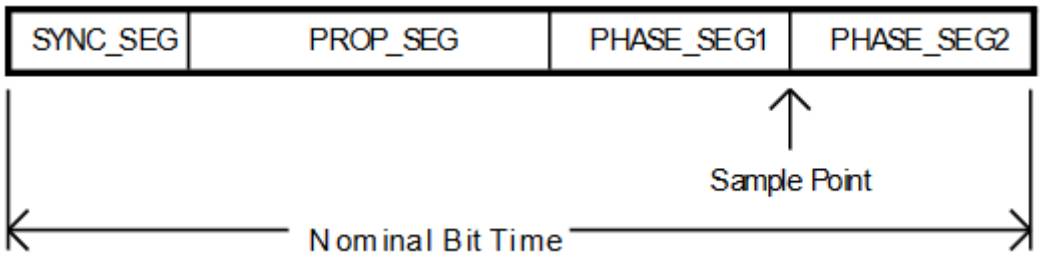


شکل ۲-۴: نمونه انجام داوری بین سه گره روی باس CAN. در اینجا سه گره به طور همزمان شروع به ارسال بسته می‌کنند، ولی از آنجایی که مقدار شناساگر گره سوم کمتر می‌باشد، مقدار غالب باس شده و برنده‌ی این داوری می‌باشد [۱٤].

علاوه بر ویژگی‌های ذکر شده برای قاب داده، عملیات پر کردن (لاگذاری) بیتی[3] نیز در هنگام ارسال قاب داده انجام می شود. این عملیات از طریق قراردادن (لاگذاری) یک بیت مخالف بعد از هر پنج بیت یکسان انجام می‌شود (بجز بخش‌های CRC و ACK) و هدف اصلی آن، کاهش خطاهای باس می‌باشد [۷].

[1] Remote
[2] Overload
[3] Bit Stuffing

ویژگی مهم دیگری که به هنگام بررسی باس CAN باید مد نظر قرار داده شود، مفهوم زمان بیتی[1] است. از آنجایی که باس CAN یک باس آسنکرون می‌باشد، باید ساز و کاری وجود داشته باشد که باعث اطمینان از صحت همگامی ارتباط شود. به همین منظور، زمان ارسال هر بیت در باس CAN به چهار قسمت تقسیم می‌شود. نام این قسمت‌ها عبارت است از: SYNC_SEG، PROP_SEG، PHASE_SEG 1 و PHASE_SEG 2. مقدار زمان بیتی اسمی (T_{NBT})، برابر مجموع این زمان‌ها می‌باشد [۱۰].



شکل ۲-۵: بخش‌های زمانی هر بیت در باس CAN [۱۵].

کوچکترین قطعه زمان در دامنه مفاهیم باس CAN، با نام کوانتوم زمان[2] یا T_Q شناخته می‌شود. این مقدار را می‌توان برابر زمان هر کلاک CAN در نظر گرفت. کلاک CAN نیز از تقسیم کلاک اصلی سیستم بر مقداری که با نام Prescaler شناخته می‌شود محاسبه می‌گردد. با توجه به مقدار T_Q و البته نیازمندی‌های نرخ داده هر کاربرد، می‌تواند مقادیر متغیرهای زمانی را محاسبه نمود [۱۵]. جدول ۲-۱ توضیحات محاسبه هر یک از این بخش‌ها را ارائه می‌دهد.

[1] Bit Time
[2] Time Quantum

جدول ۲-۱: مقادیر بخش‌های هر بیت زمانی [۱۶].

نام بخش	مقدار زمانی	توضیحات
SYNC_SEG	یک کوانتوم زمانی	همواره مقداری ثابت دارد.
PROP_SEG	بین یک تا هشت کوانتوم زمانی	مقدار آن وابسته به طول باس و جنس کابل‌ها و اتصالات می‌باشد.
PHASE_SEG 1	بین یک تا هشت کوانتوم زمانی	هدف اصلی آن، ایجاد تحمل کافی برای خطاها و ایجاد همگامی است.
PHASE_SEG 2	مقدار بزرگتر بین دو مقدار PHASE_SEG 1 و IPT[1]	هدفی مشابه PHASE_SEG 1 دارد.

۲-۲- مقایسه باس CAN با باس‌های دیگر

در مقابل باس CAN، باس‌های سریال زیادی وجود دارند که به صورت گسترده در صنایع مختلف مورد استفاده قرار می‌گیرند. در این میان می‌توان به باس‌های SPI، I2C و USART اشاره نمود. جدول ۲-۲ مقایسه‌ای اجمالی بین این باس‌ها را به نمایش می‌کشد. معماری Multi-Master، مختص به پروتکل‌های ارتباطی می‌باشد که هر یک از گره‌های متصل به شبکه می‌توانند اقدام به شروع مخابره کنند، در مقابل این موضوع معماری Single-Master قرار دارد که بدان معنی است که تنها یک گره می‌تواند اقدام به شروع مخابره نماید [۱۹]. انتقال سنکرون نیز به معنی وجود سیگنال کلاک مشترک بین گره‌های متصل به باس بوده و انتقال آسنکرون نیز عدم وجود چنین سیگنالی را نشان می‌دهد [۱۹].

[1] Information Proccessing Time: زمان پردازش اطلاعات که عموما برابر دو کوانتوم زمانی است.

جدول ۲-۱: مقایسه باس‌های سریال رایج و باس CAN [۱۹]–[۱۷] ,[۱۰].

	CAN	USART	I2C	SPI	USART
حداکثر نرخ داده	1 Mbps	وابسته به طرفین ارتباط	3.4 Mbps	60 Mbps	وابسته به طرفین ارتباط
توپولوژی	باس خطی	نقطه به نقطه	باس خطی	نقطه به نقطه	نقطه به نقطه
معماری	Multi-master	Multi-Master	Multi-Master	Single-Master	Multi-Master
نوع انتقال پیام	آسنکرون	آسنکرون	سنکرون	آسنکرون	آسنکرون
دارای پروتکل مشخص	بله	-	بله	-	-
تشخیص خطا	CRC	-	-	-	-
مقاومت در برابر نویز	بله	–	–	–	–

علاوه بر باس‌های سریال، باس‌های دیگری نیز وجود دارند که با اهدافی مشابه CAN و عموما به هدف استفاده در صنایع خودروسازی توسعه یافته‌اند. از جمله این باس‌ها می‌توان به LIN، MOST و FlexRay اشاره نمود [۲۰]-[۲۲]. در جدول ۲-۳، مقایسه‌ای اجمالی بین این باس‌ها انجا شده است. عملکرد پروتکل دسترسی CSMA به این صورت می‌باشد که که هر گره، پیش از ارسال هر پیام وضعیت باس را بررسی کرده و از عدم وجود ترافیک روی باس اطمینان می‌یابد [۲۰]. پروتکل TDMA نیز زمان را به بخش‌های مساوی تقسیم کرده و هر گره مجاز است در بخش‌های زمانی مشخصی اقدام به ارسال و یا دریافت نماید. کاربرد بلادرنگ سخت، به این معناست که عدم توانایی سیستم در انجام حتی یک عملکرد پیش از ضرب‌العجل مشخص، به معنای خرابی کل سیستم است، در حالی که در کاربردهای بلادرنگ نرم، تعداد کمی از دیرکردها قابل پذیرش هستند [۲۲].

جدول ۲-۲: مقایسه اجمالی بین باس‌های CAN، LIN و MOST [۲۳].

	CAN	LIN	FlexRay	MOST
حداکثر نرخ داده	1 Mbps	20 Kbps	10 Mbps	24 Mbps
پروتکل دسترسی	CSMA/CA	Polling	TDMA	TDM CSMA/CA
لایه فیزیکی	دو سیم	یک سیم	دو سیم و فیبر نوری	دو سیم مبتنی بر فیبر نوری
معماری	Multi-master	Single master	Multi-master	Multi-master
نوع انتقال پیام	آسنکرون	سنکرون	سنکرون	آسنکرون و سنکرون
کاربرد	بلادرنگ نرم	زیر شبکه‌ها	بلادرنگ سخت	چندرسانه‌ای
تاخیر	وابسته به میزان ترافیک شبکه	ثابت	ثابت	وابسته به جریان داده

۲-۳- ویرایش‌های باس CAN

به منظور رفع برخی مشکلات باس CAN و یا افزودن برخی قابلیت‌های خاص، به طور مثال افزایش نرخ انتقال داده و یا افزایش اطمینان‌پذیری، ویرایش‌های مختلفی از این باس ارائه شده است. از میان این ویرایش‌های متفاوت این باس، می‌توان به TTCAN (Time Triggered CAN) و CAN FD (CAN Flexible Data-Rate) اشاره نمود.

۲-۳-۱- TTCAN

TTCAN ویرایشی از باس CAN می‌باشد که برای کاربردهایی با درجه بلادرنگ بودن بالا توسعه یافته است و در استاندارد ISO-11898 نیز شرح داده شده‌است. یکی از مشکلاتی که امکان وجود آن در باس

CAN استاندارد وجود دارد، استفاده بی‌رویه یکی از گره‌ها از باس مشترک است. بدین صورت که یک گره با ارسال پیام‌های متوالی با مقدار شناساگر کم (اولویت بالا) باس را در اختیار گرفته و سایر گره‌ها توان استفاده درست از باس را نخواهند داشت. و این موضوع می‌تواند برای کاربردهای بلادرنگ مشکل‌ساز باشد [۲٤].

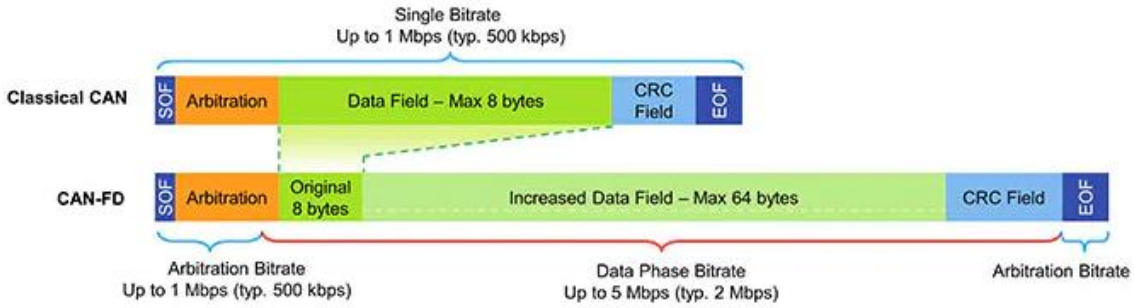
به منظور رفع مشکل ذکر شده، TTCAN رویکردی جدید برای استفاده از باس مشترک در نظر می‌گیرد. در این رویکرد، یکی از گره‌های شبکه نقش راهبر[1] شبکه را در اختیار گرفته و دو وظیفه اساسی را بر عهده دارد: ۱. همگام‌سازی زمانی گره‌ها از طریق مخابره همگانی یک شکاف زمانی. ۲. برنامه‌ریزی زمانی ارسال پیام از طریق یک ماتریس زمان‌بندی[2]. هر ستون این ماتریس نشان دهنده یک سیکل زمانی است و سه نوع سیکل زمانی مختلف وجود دارد: ۱. سیکل‌های زمانی که مخصوص ارسال پیامی خاص می‌باشد. ۲. سیکل‌های زمانی که مشابه CAN استاندارد، داوری بر اساس اولویت شناساگر انجام می‌شود. ۳. سیکل‌های زمانی که هیچ گره‌ای در آن نمی‌تواند به مخابره پیام بپردازد و برای استفاده‌های دیگر رزرو شده است [۲٤].

CAN FD - ۲-۳-۲

مهندسان شرکت بوش در سال ۲۰۱۲ باس CAN FD را معرفی کردند. هدف اصلی معرفی این ویرایش خاص از باعث CAN، پشتیبانی از نرخ‌های انتقال بالاتر از 1 Mbps در کنار افزایش اندازه محموله[3] هر قاب به مقادیر بیشتر از هشت بایت بود [٥]. باس CAN FD با قیمتی مشابه به CAN استاندارد، سرعتی چند برابر آن را ارائه می‌کند، البته موضوع به همین‌جا ختم نمی‌شود و CAN FD ساز و کار پیشرفته‌تری برای تشخیص خطا نیز دارد [٦]. در راه طراحی CAN FD، تلاش بر این بود که لایه فیزیکی CAN حفظ شده و تنها لایه‌های بالاتر تغییر کنند، بنابراین CAN FD را می‌توان یک پروتکل برای لایه فیزیکی CAN استاندارد نیز دانست. به طور کلی افزایش نرخ داده ذکر شده در CAN FD از

[1] Master
[2] Schedule matrix
[3] Payload

دو طریق تغییر صورت گرفته است: ۱. افزایش نسبت محموله به سرآیند در هر قاب داده، ۲. کوتاه کردن زمان بیتی[1] [۵]. شکل ۲-۶، مقایسه‌ای بین یک قاب CAN استاندارد و قاب CAN FD را نشان می‌دهد.



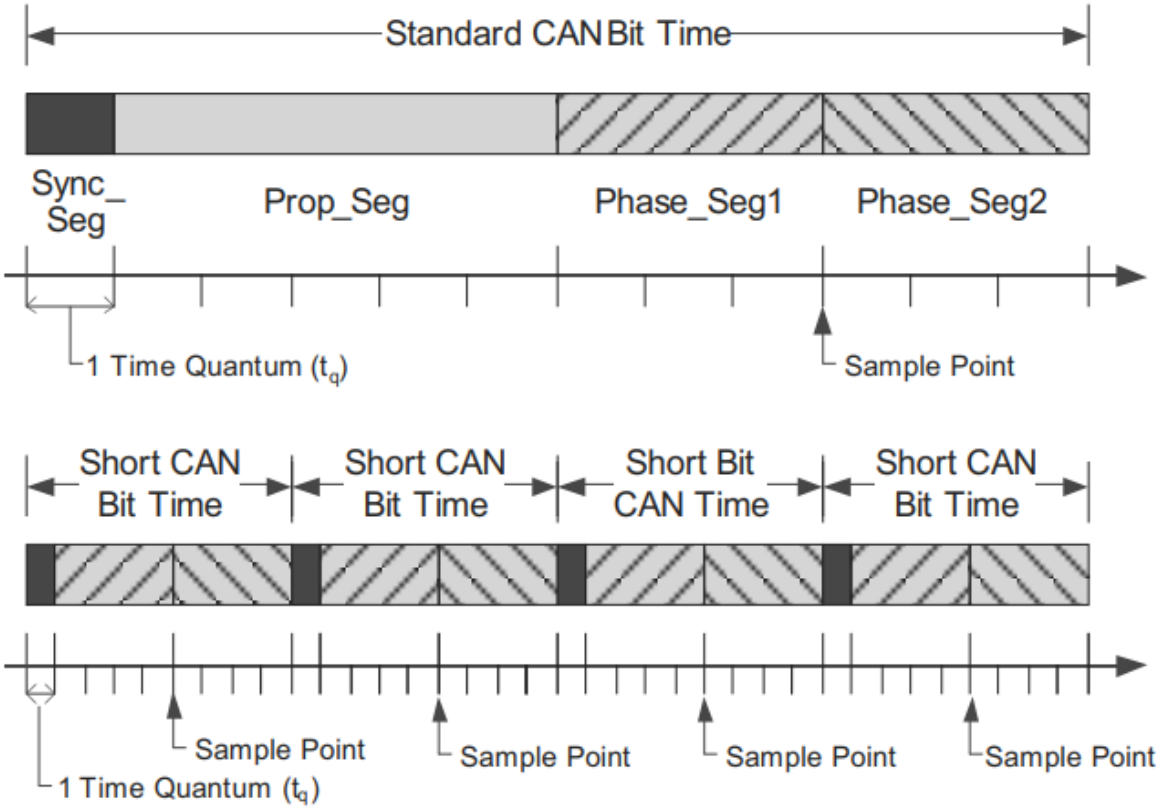
شکل ۲-۶: مقایسه قالب داده CAN استاندارد و CAN FD [۵].

همانطور که از شکل ۲-۶ بر می‌آید، تفاوت عمده قاب داده CAN و CAN FD در دو بخش Data و CRC دو باس خلاصه می‌شود. محموله داده یک پیام CAN استاندارد می‌تواند حداکثر هشت بایت باشد، در حالی که محموله داده پیام CAN FD می‌تواند طولی برابر ۶۴ بایت داشته باشد. همچنین طول CRC در CAN FD می‌تواند ۱۷ و یا ۲۱ بایت باشد (در مقابل ۱۵ بایت در CAN استاندارد) [۹]-[۲۵].

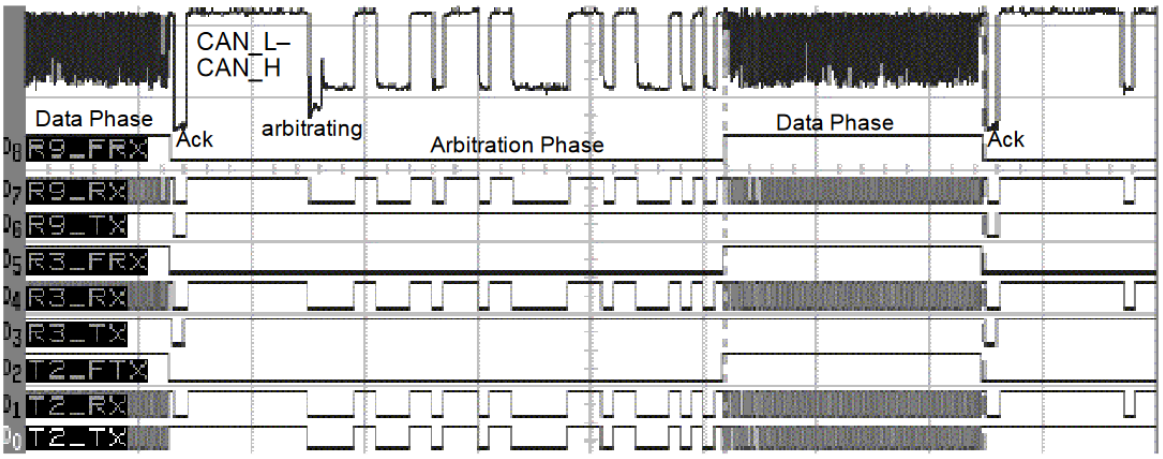
البته که افزایش نرخ داده CAN FD تنها توسط افزایش طول محموله داده هر پیام محقق نمی‌شود، بلکه ساز و کاری برای کاهش زمان بیتی نیز به کار گرفته شده است. ایده اصلی استفاده از این ساز و کار این است که هنگامی که عملیات داوری انجام شده و یک پیام برنده داوری شد، پیام مورد نظر می‌تواند بدون توجه به محدودیت‌های زمانی و بدون همگام‌سازی با همه گره‌های شبکه، به ارسال بخش محموله پیام بپردازد [۲۵]. وجود این ساز و کار در کنار افزایش طول محموله هر پیام، می‌تواند نرخ ارسال داده را تا مقادیری بیش از ۱Mbps افزایش داده که مقادیری تا حدود ۵Mbps در ارتباط CAN FD متداول است، هرچند که مقادیر بزرگ‌تر از این مقدار نیز بسته به شرایط ممکن است [۵].

[1] Bit Time

در واقع کنترل‌کننده باس CAN FD دارای دو پیکربندی زمانی است. پیکربندی زمانی اول، مشابه جدول ۲-۱ بوده و در مرحله داوری یک انتقال مورد استفاده قرار می‌گیرد. بنابراین در این مرحله، نرخ ارسال داده همچنان حداکثر برابر ۱Mbps می‌باشد. پیکربندی زمانی دوم (زمان بیتی کوتاه) نیز مختص زمان ارسال محموله است. این پیکربندی زمانی، دارای کوانتوم‌های زمانی با طول کوتاه‌تر بوده که باعث ارسال هر بیت با سرعتی بیشتر می‌شود. البته که ممکن است دو پیکربندی زمانی ذکر شده کاملا یکسان باشند و این موضوع وابسته به نیازمندی‌های موجود در هر پیاده‌سازی می‌باشد. علاوه بر مرحله داوری، در مرحله ارسال داده CRC نیز از پیکربندی زمانی اول استفاده می‌شود تا اطمینان‌پذیری انتقال بیشتر شود [۲۵]. شکل ۲-۷ مقایسه‌ای بین زمان بیتی کوتاه و معمولی را نشان می‌دهد. شکل ۲-۷ نیز یک شکل موج از ارسال داده مبتنی بر باس CAN FD را به نمایش می‌گذارد که در این ارتباط، پیکربندی زمانی دوم دارای کوانتوم‌های زمانی به مراتب کوتاه‌تر از پیکربندی اول است. بنابراین به وضوح بیت‌های محموله با نرخ بسیار بالاتری روی خطوط باس منتقل شده و نرخ ارسال کلی افزایش چشم‌گیری خواهد داشت.

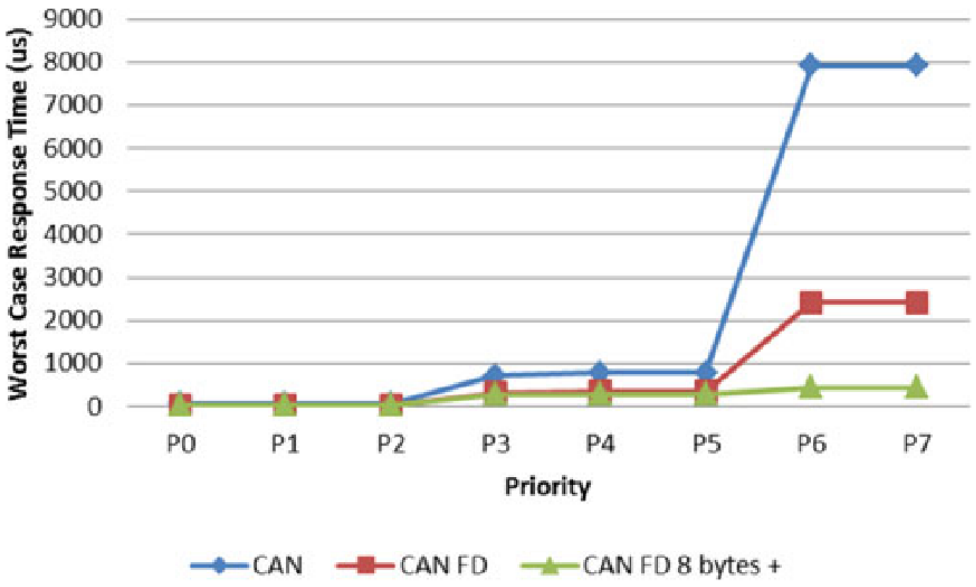


شکل ۲-۷: مقایسه دو پیکربندی زمانی موجود در CAN FD [۲۵].



شکل ۲-۸: شکل موج انتقال داده مبتنی بر CAN FD با نرخ ۱۲Mbps [۲۵].

شکل ۲-۹، مقایسه‌ای زمانی بین CAN و CAN FD را به نمایش می‌کشد. در این مطالعه، در نظر گرفته شده است که هشت سطح اولویت متفاوت در یک سیستم وجود دارد و به طور تقریبا همزمان، درخواستی برای داده‌های با اولویت‌های متفاوت ایجاد می‌شود. در پایان، بیشترین زمان پاسخگویی گره‌ها برای پیام‌های با اولویت‌های مختلف اندازه‌گیری و مقایسه شده است [۲۶]. نتایج بدست آمده، حاکی از برتری چند برابری زمان پاسخگویی در CAN FD نسبت به CAN استاندارد می‌باشد.



شکل ۲-۹: مقایسه عملکرد زمانی باس CAN استاندارد، باس CAN FD با محموله ۸ بایتی و باس CAN FD با محموله بیش از ۸ بایت [۲۶].

تغییری دیگری که برای بکارگیری CAN FD نیاز است، وجود بخش‌هایی برای تمیز پیام CAN FD از پیام CAN استاندارد و همچنین نوع ارسال پیام می‌باشد. به همین منظور، دو بخشی که در ادامه معرفی می‌شوند، در قاب CAN FD گنجانده شده‌اند:

FDF (FD frame): این بخش یک بیتی جایگزین بخش رزرو r0 در CAN استاندارد شده است و اگر مقدار آن برابر یک مغلوب باشد، نشان دهنده‌ی این است که پیام ارسالی از نوع CAN FD است.

res: این بخش یکی بیتی به قاب CAN FD به عنوان بیت رزرو افزوده شده است و مقدار آن باید برابر یک مغلوب باشد.

BRS (Bit Rate Switch): این بخش یک بیتی، برای فعال یا غیرفعال‌سازی استفاده از ساز و کار کاهش زمان‌بیتی است. در صورتی که مقدار صفر غالب را پذیرفته باشد، بسته CAN FD با نرخ حداکثر ۱Mbps ارسال شده و در غیر این صورت، با کاهش زمان‌بیتی، دستیابی به نرخ‌های ارسال بالاتر نیز ممکن می‌شود.

ESI (Error Status Indicator): در صورت که مقدار یک غالب داشته باشد، ارتباط موجود نسبت به وجود خطاها حساس است و در غیر این صورت، ارتباط از خطاها صرف نظر می‌کند.

SBC (Stuff Bit Count): بخشی چهار بیتی است که شامل بیت‌های parity بوده و برای افزایش اطمینان‌پذیری به قاب CAN FD افزوده شده است [۲۷].

CAN FD چهار ساز و کار مختلف به منظور تشخیص و اصلاح خطا ارائه می‌دهد که دو مورد از آن‌ها در لایه فیزیکی صورت گرفته و دو مورد باقی‌مانده در لایه لینک داده اعمال می‌شوند. این ساز و کارها عبارتند از:

نظارت بر بیت[1]: کنترل‌کننده باس، مقداری که قصد ارسال آن را داشته با مقداری که روی باس قرار گرفته است مقایسه کرده از این طریق صحت ارسال را بررسی می‌کند.

پر کردن بیتی: استفاده از یک بیت با قطبیت مخالف پس از پنج بیت با قطبیت یکسان که به منظور تشخیص دقیق‌تر بیت ارسالی می‌باشد.

CRC: استفاده از کد افزونگی چرخشی داده ارسالی که طول آن ۱۷ بیت (برای محموله‌هایی با اندازه کمتر از ۱۷ بایت) و یا ۲۱ بیت می‌باشد.

بررسی قاب پیام: در سطح پیام‌های دریافتی، تطابق قاب دریافتی با استاندارد پروتکل CAN FD بررسی می‌شود.

[1] Bit Monitoring

علاوه بر ساز و کارهای فوق به منظور تشخیص خطا، امکان استفاده از پیام‌های ACK نیز وجود دارد [۲۸].

جدول ۲-۳: مقایسه اجمالی باس CAN و ویرایش CAN FD [۹]- [۱۰]-[۲۵]- [۲۸].

مشخصه	CAN	CAN FD
لایه فیزیکی	سیگنال تفاضلی روی جفت سیم در هم تنیده	سیگنال تفاضلی روی جفت سیم در هم تنیده
حداکثر طول باس	۴۰ متر (طبق استاندارد)	۴۰ متر (طبق استاندارد)
نرخ انتقال داده	حداکثر ۱Mbps	حداکثر ۵Mbps (طبق استاندارد)
طول محموله هر پیام	حداکثر ۸ بایت	حداکثر ۶۴ بایت
طول شناساگر پیام (ID)	۲۹ بیت (یا ۱۱ بیت)	۲۹ بیت (یا ۱۱ بیت)
طول کد بررسی CRC	۱۵ بیت	۱۷ یا ۲۱ بیت
زمان بیتی	یک پیکربندی به کوانتوم زمانی طولانی	دو پیکربندی متفاوت، یکی از آن‌ها کوتاه است و دیگری مشابه CAN استاندارد
سازگاری	با CAN FD سازگار نمی‌باشد	می‌تواند طوری پیکربندی شود که مشابه CAN عمل کند.

۲-٤- پروتکل‌های لایه‌های بالاتر سازگار با CAN

همانطور که پیش‌تر ذکر شد، در استاندارد باس CAN تنها به پیاده‌سازی لایه فیزیکی و لایه لینک داده پرداخته شده است [٤]. بنابراین به منظور استفاده بهینه از باس CAN در مقیاس بزرگ، نیاز به پروتکل‌هایی وجود دارد که در لایه کاربرد[1] پیاده‌سازی شده باشند و سطح تجرید مناسبی برای استفاده کاربران و طراحان سیستم ایجاد کنند. پروتکل‌های مختلفی در راستای همین نیاز ارائه شده‌اند که در ادامه به اختصار به برخی از آن‌ها پرداخته شده است.

۲-۴-۱- پروتکل OBD2

پروتکل OBD (On Board Diagnostics) به هدف تسهیل عیب‌یابی خودروها و کامیون‌های سبک طراحی شده است. از سال ۱۹۹۶ میلادی به بعد، قانونا باید درون همه خودروهای آمریکای شمالی پیاده‌سازی شده باشد. وجود این پروتکل به کاربران اجازه می‌دهد که با در اختیار داشتن یک پوینده OBD، به راحتی همه اطلاعات حسگرها و عملگردهای متصل به سیستم الکترونیکی خودرو را مشاهده نمایند [۲۹].

۲-۴-۲- پروتکل SAE J1939

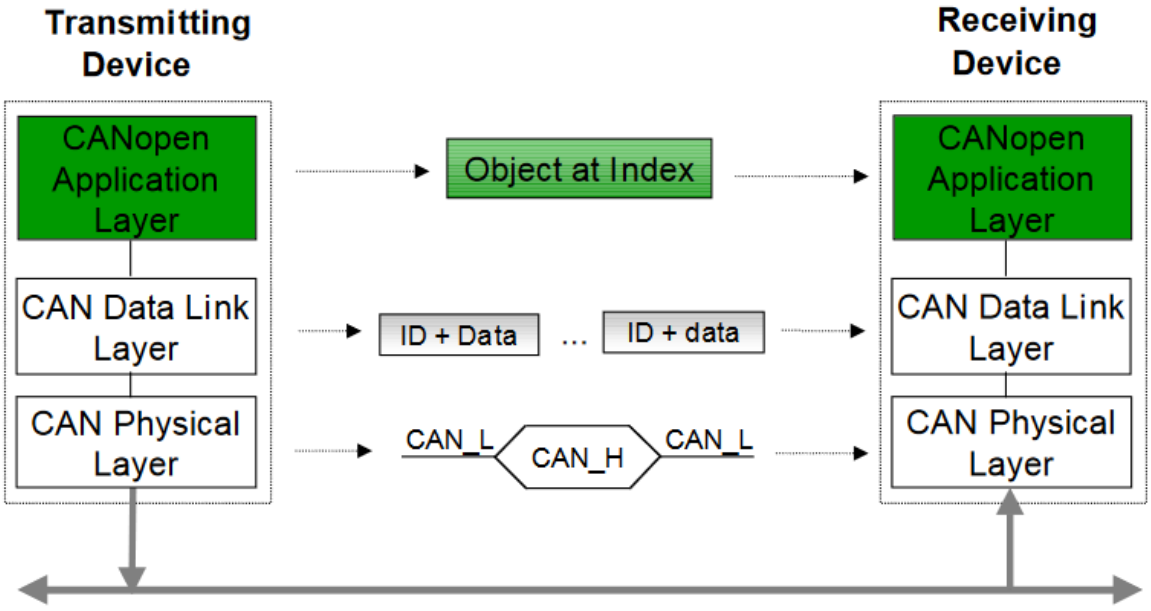
این پروتکل نیز مشابه پروتکل OBD2 برای استفاده در خودروها توسعه یافته است ولی با پیاده‌سازی امکانات اضافی در راستای اطمینان‌پذیری بالاتر، برای محیط‌های مشکل خشن مناسب می‌باشد [۳۰].

۲-۴-۳- پروتکل CANOpen

پروتکل CANOpen توسط کارگروهی شامل تولیدکنندگان و کاربران موسوم به CiA[2] پشتیبانی می‌شود که البته هر کاربر می‌تواند بسته به نیازش، مستقلا تغییراتی در پیاده‌سازی نهایی خود انجام دهد

[1] Application Layer
[2] Can in Automation

[۹]. همانطور که در شکل ۲-۱ نیز مشخص است، لایه‌های فیزیکی و لینک داده (مطابق مدل مرجع OSI) توسط استاندارد باس CAN مشخص می‌شود و هدف CANOpen، پیاده‌سازی لایه‌های بالایی است. به همین منظور، CANOpen توانایی مدیریت یک شبکه، ارتباط معنی‌دار بین گره‌ها و نظارت بر دستگاه‌ها را به یک مجموعه دستگاه متصل از طریق باس CAN می‌افزاید [۳۱]. بدین صورت، شکل ۲-۱۰ نمایشی از شیوه استقرار این پروتکل در یک ارتباط باس CAN را به نمایش می‌گذارد.



شکل ۲-۱۰: نمایش مدل لایه‌ای شبکه‌ای مبتنی بر پروتکل CANOpen [۳۲].

به منظور ارسال و دریافت داده معنی‌دار، CANOpen دو مفهوم نوع داده[1] و توالی بیتی[2] را شرح می‌دهد. نوع داده‌ها می‌توانند اولیه یا ایستا (مانند اعداد یا متغیر بولین) و یا پیچیده (مانند زمان) باشند. سپس بر اساس نوع داده، کدگذاری آن انجام شده و در توالی‌های بیتی هشتی (یک بایتی) به مقصد ارسال می‌شوند. این انتقال از طریق آدرس‌دهی به هر گره انجام می‌شود که در حالت استفاده از CAN 1.0A، حداکثر تعداد گره قابل آدرس‌دهی برابر ۱۲۷ می‌باشد و هر گره دارای یک آدرس منحصر به فرد می‌باشد. هر بسته ارسالی نیز یک شی ارتباطی[3] نام دارد که بسته به کاربرد آن در شبکه CANOpen،

[1] Data Type
[2] Bit sequences
[3] Communication Object

انواع مختلفی شی ارتباطی وجود دارد که در جدول ۲-۵ مشخص شده است [۳۳]. همچنین قاب داده پیام‌های پروتکل CANOpen در جدول ۲-۶ قابل مشاهده می‌باشد. از آنجایی که قاب داده CANOpen می‌تواند حداکثر ۸ بایت داده را منتقل کند، عملکرد قطعه قطعه کردن یک محموله ارسالی به چندین قاب مختلف برای ارسال روی باس CAN (و یا CAN FD) نیز پیاده‌سازی شده است [۳۲].

جدول ۲-۴: اشیا ارتباطی پروتکل CANOpen [۳۳].

شی ارتباطی	وظیفه
PDO	ارسال داده مورد نظر کاربر.
SDO	ارسال اطلاعات پیکربندی شبکه و دستگاه‌ها.
NMT node monitoring	بررسی صحت کارکرد یک دستگاه در شبکه از طریق پروتکل ضربان قلب[1].
LSS	پیکربندی شبکه، مانند تغییر نرخ داده.
NMT node control	تغییر وظعیت شبکه برای دستگاه‌ها، مانند اتصال یا قطع شدن از شبکه.
Global failsafe command	ارسال پیام‌هایی با اولویت بسیار بالا به منظور حفظ امنیت شبکه.
Sync	همگام‌سازی شبکه
Emergency	ارسال پیام‌هایی با اولویت بسیار بالا مانند خرابی یک دستگاه.
TimeStamp	ارسال زمان.

جدول ۲-۵: اجزا یک بسته ارسالی توسط CANOpen روی بستر CAN 1.0 [۳۳].

	Function code	**Node ID**	**Remote Transmission Request**	**Data length**	**Data**
طول	۴ بیت	۷ بیت	۱ بیت	۴ بیت	۰ تا ۸ بایت

[1] Heartbeat protocol

مفهوم کلیدی دیگری که در پروتکل CANOpen تعریف می‌شود، دیکشنری اشیاء[1] است که در حقیقت هسته مرکزی این پروتکل می‌باشد. در دیکشنری اشیاء، تمام داده‌های قابل ارسال و یا دریافت (اشیاء) شبکه ثبت می‌شوند و از طریق یک اندیس ۱۶ بیتی قابل دسترسی هستند. این اشیاء ثبت شده می‌توانند ثابت و یا متغیر باشند، ولی همواره به صورت استاندارد، برخی اشیاء خاص باید در این دیکشنری ثبت شده باشند. در نهایت، می‌توان از این دیکشنری برای درخواست و یا ارسال داده‌ای مشخص تنها از طریق دانستن مقدار اندیس آن اقدام نمود [۳۲]. جدول ۲-۷، نوع داده‌های موجود در دیکشنری اشیاء را نشان می‌دهد.

جدول ۲-۶: اندیس و نوع اشیاء موجود در دیکشنری اشیاء [۳۲].

اندیس شی (بر مبنای ۸)	نوع داده
0000	رزرو شده
0001-001F	داده‌های ایستا
0020-003F	داده‌هایی با نوع پیچیده
0040-005F	داده‌های مختص تولیدکننده
0060-007F	داده‌های ایستا مربوط به وضعیت دستگاه
0080-009F	داده‌های پیچیده مربوط به وضعیت دستگاه
00A0-0FFF	رزرو شده
1000-1FFF	داده‌های مختص شبکه
2000-5FFF	داده‌های مختص تولیدکننده دستگاه
6000-9FFF	داده‌های استاندارد دستگاه
A000-FFFF	داده‌های رزرو شده

[1] Object Dictionary

فصل سوم

طراحی و پیاده‌سازی ارتباطات

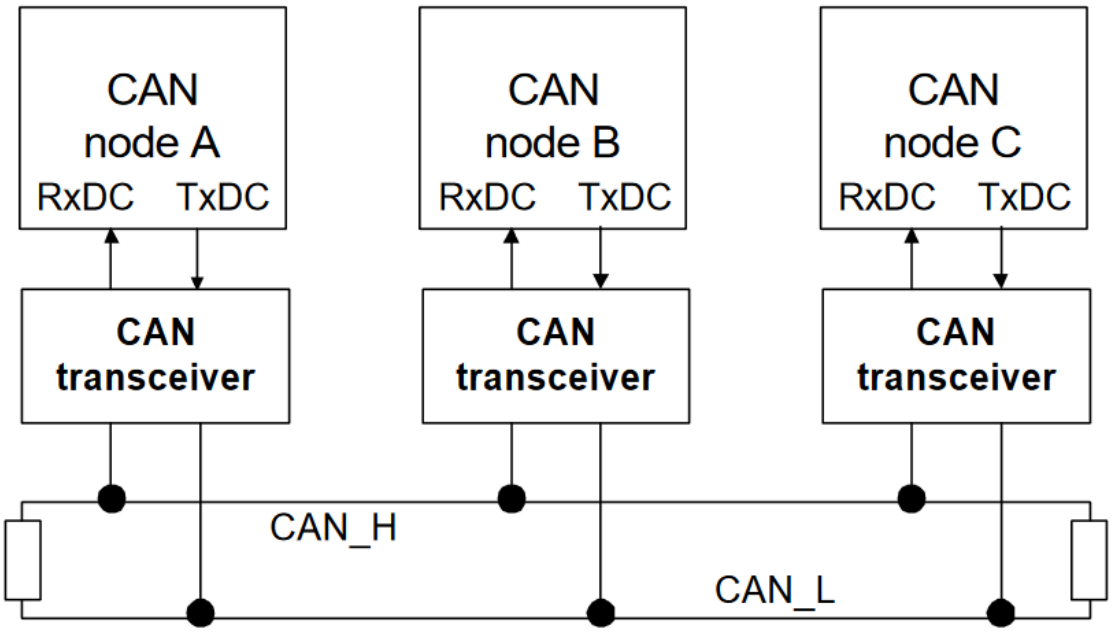
طراحی و پیاده‌سازی ارتباطات

۳-۱- اهداف و ساختار کلی

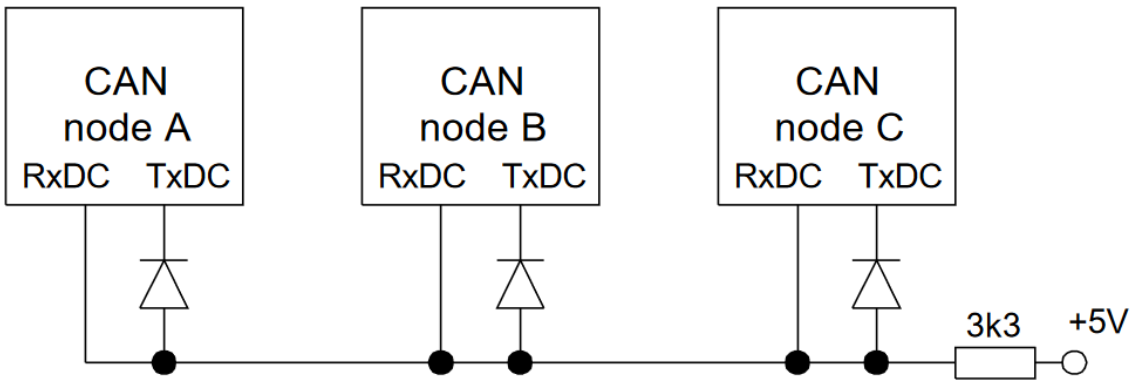
هدف این پروژه، پیاده‌سازی یک ارتباط مبتنی بر باس‌های CAN و CAN FD است که از طریق یک میکروکنترلر و یک FPGA محقق می‌شود. در گام اول، یک ارتباط دو طرفه بین دو دستگاه از طریق CAN ایجاد شده و علاوه بر این ارتباط دو طرفه، یک ارتباط از طریق باس CAN FD درون FPGA نیز پیاده‌سازی خواهد شد و تلاش شده است که این ارتباطات با پروتکل لایه کاربرد CANOpen سازگاری داشته باشند. خروجی نهایی این پیاده‌سازی، یک سیستم ارتباطی مناسب برای سیستم‌های نهفته در کاربردهای متفاوت است و از آن‌جایی دو پیاده‌سازی مجزا انجام می‌شوند (پیاده‌سازی نرم‌افزاری در سطح میکروکنترلر و پیاده‌سازی سخت‌افزاری در سطح FPGA)، نتایج این طرح می‌تواند برای کاربردهای زیادی مفید واقع شوند. تصویر کلی سیستم پیاده شده در شکل ۱-۱ قابل مشاهده می‌باشد.

همانطور که در فصل ۲ اشاره شد، لایه فیزیکی باس CAN از دو سیم درهم تنیده تشکیل شده است که سیگنال انتقالی را به صورت ولتاژ تفاضلی مخابره می‌کنند [۱۰]. با توجه به ماهیت دیجیتال قطعات مورد استفاده مانند میکروکنترلر، امکان تولید سیگنال تفاضلی آنالوگ مورد نظر بدون استفاده از ادوات جانبی وجود ندارد. بنابراین، به منظور تولید دو سیگنال CAN_H و CAN_L نیاز به دستگاه فرستنده-گیرنده‌ای هست که این عملیات را انجام دهد. شکل ۳-۱ نمودار بلوکی کلی یک باس CAN با استفاده از فرستنده-گیرنده را نشان می‌دهد. البته که استفاده از فرستنده-گیرنده تنها راه استفاده از باس CAN در قطعات دیجیتال نمی‌باشد، بلکه راهکار دیگری نیز پیشنهاد شده است که در شکل ۳-۲ مشاهده می‌شود. استفاده از این روش پیشنهاد نمی‌شود چرا که در عمل برخی از خواص باس CAN را زیر سوال

برده و همچنین حداکثر طول موثر باس را به شدت کاهش داده به طوری که تنها برای کاربردهای روی برد مناسب می‌باشد [۳٤].



شکل ۳-۱: باس CAN با استفاده از فرستنده-گیرنده [۳٤].



شکل ۳-۲: باس CAN بدون استفاده از فرستنده-گیرنده [۳٤].

۳-۲- قطعات مورد استفاده

در راستای پیاده‌سازی این پروژه، قطعاتی برای راه‌اندازی سیستم ارتباطی نیاز است که برای این طرح، از قطعات زیر استفاده شده است:

- برد Arduino Due به عنوان میکروکنترلر
- برد Ava3S400 به عنوان FPGA
- برد فرستنده-گیرنده TJA1050
- سیم جامپر متداول برای اتصالات

در ادامه مشخصات اجمالی هر یک از این قطعات بیان خواهد شد.

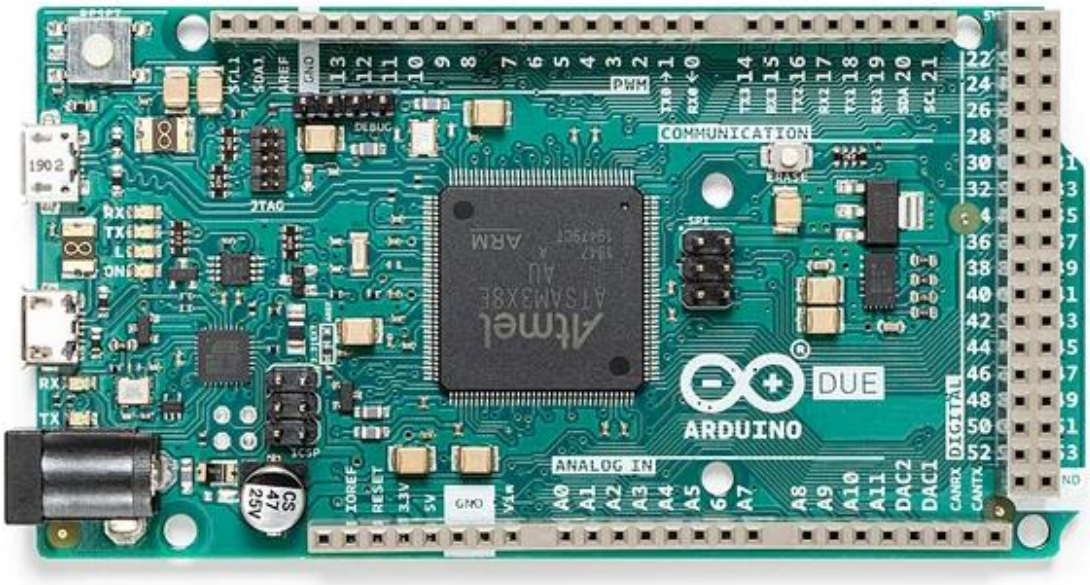
۳-۲-۱- برد Arduino Due

برد Arduino Due یک برد میکروکنترلر و ساخت شرکت Arduino می‌باشد. پردازنده موجود روی این برد Atmel SAM3X8E ARM Cortex-M3 بوده و نخستین برد Arduino است که از معماری ۳۲ بیتی ARM بهره برده و فرکانس کاری آن ۸۴MHz می‌باشد. بر خلاف دیگر بردهای Arduino (که عموما دارای معماری AVR هستند) ولتاژ کاری این برد برابر ۳.۳V بوده و توان لازم از طریق کابل Micro-USB تامین می‌شود. سایر ویژگی‌های کلیدی این برد عبارتند از:

- حافظه Flash با ظرفیت ۵۱۲ کیلوبایت
- حافظه SRAM با ظرفیت ۹۶ کیلوبایت
- ۵۴ پین ورودی-خروجی عمومی[1] دیجیتال
- ۱۲ پین ورودی-خروجی آنالوگ (PWM)

[1] GPIO

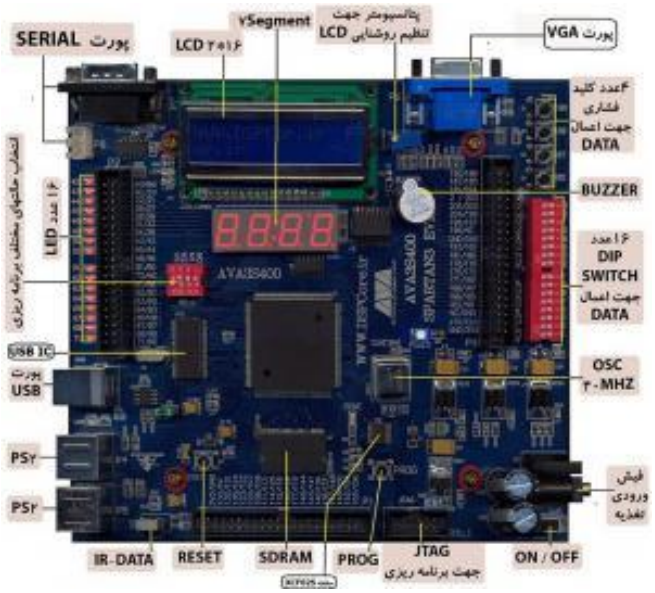
- ۲ مبدل دیجیتال به آنالوگ ۱۲ بیتی
- کنترل‌کننده واسط‌های SPI، I2C و CAN [۳۵]



شکل ۳-۳: برد Arduino Due از نمای بالا [۳۶].

۳-۲-۲- برد Ava3S400

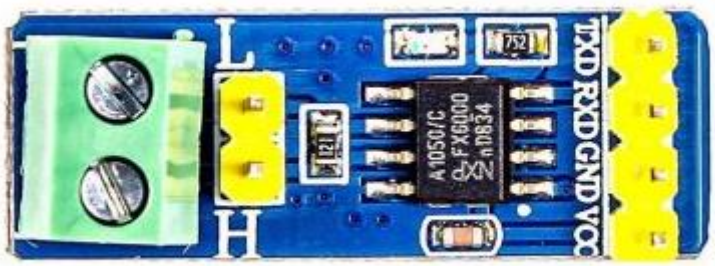
برد Ava3S400 یک برد FPGA مبتنی بر Xilinx Spartan XC3S400 ساخت شرکت رهپویان علم و صنعت آوا می‌باشد. این FPGA دارای ۸۰۶۴ سلول منطقی، ۵۶ کیلوبایت حافظه RAM توضیع شده، ۲۸۸ کیلوبایت حافظه BRAM و حداکثر ۲۶۴ ورودی-خروجی می‌باشد. برنامه‌ریزی این FPGA از طریق واسط JTAG امکان‌پذیر می‌باشد [۳۷]. برد مورد نظر نیز امکانات متنوعی برای مدیریت ورودی-خروجی‌ها نظیر ۱۶ سوئیچ، ۱۶ عدد LED، درگاه USB و همچنین یک اسیلاتور با فرکانس کاری ۴۰MHz تدارک دیده است که همگی از طریق محیط توسعه FPGA قابل برنامه‌ریزی و استفاده می‌باشند [۳۸].



شکل ۳-۴: تصویر برد Ava3S400 و ادوات جانبی آن [۳۸].

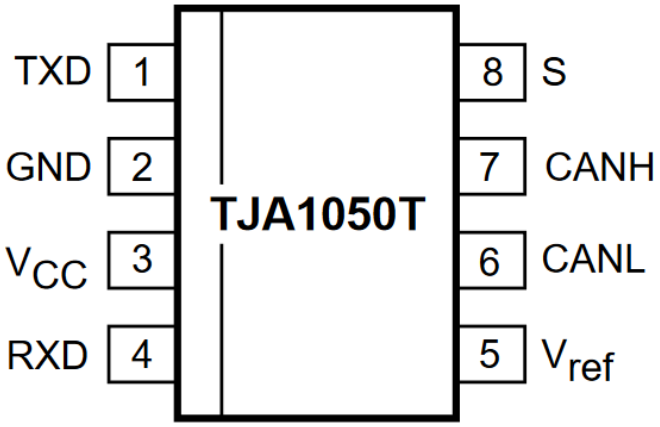
۳-۲-۳- برد فرستنده-گیرنده TJA1050

TJA 1050 یک فرستنده-گیرنده باس CAN است که توسط شرکت NXP تولید شده است. این فرستنده گیرنده کاملا با استاندارد ISO1898 سازگار بوده و وظیفه‌ی اصلی آن این است که سیگنال‌های تفاضلی CAN_H و CAN_L را دریافت کرده و طبق مشخصات پروتکل CAN، این دو یک سیگنال سریال تولید کرده که داده منتقل شده را نمایش می‌دهد. حداکثر نرخ بیتی انتقال برای این فرستنده-گیرنده طبق مشخصات تولیدکننده برابر ۱Mbps می‌باشد [۳۹].



شکل ۳-۵: برد شامل فرستنده-گیرنده TJA 1050.

همانطور که ذکر شد، عملکرد این برد مبتنی بر تراشه فرستنده-گیرنده TJA 1050 می‌باشد. شکل ۳-۶، شماتیک این تراشه و جدول ۳-۱ وظایف پایه‌های این تراشه را نشان می‌دهد.



شکل ۳-۶: شماتیک تراشه TJA 1050 از نمای بالا [۳۹].

جدول ۳-۱: توضیحات پایه‌های تراشه TJA 1050 [۳۹].

شماره پایه	نماد پایه	عملکرد	توضیحات
۱	TXD	خروجی ارسال داده	مقادیر دریافتی از کنترل‌کننده CAN را دریافت کرده و روی باس ارسال می‌نماید.
۲	GND	مرجع زمین تراشه	-
۳	V_{CC}	ولتاژ تغذیه تراشه	باید بین ۴.۷۵ تا ۵.۲۵ ولت باشد.
۴	RXD	دریافت داده ورودی	داده موجود روی باس را دریافت کرده و برای کنترل‌کننده CAN ارسال می‌کند.
۵	V_{ref}	ولتاژ مرجع خروجی	-
۶	CANL	خط CAN_L باس	مقداری بین صفر ولت تا V_{CC} را می‌پذیرد.
۷	CANH	خط CAN_H باس	مقداری بین صفر ولت تا V_{CC} را می‌پذیرد.
۸	S	انتخاب حالت عملکرد	انتخاب بین دو حالت فعال و غیر فعال از طریق این

			پایه ممکن می‌شود.

۳-۳- نرم‌افزارهای مورد استفاده

عمده توسعه این طرح توسط دو نرم‌افزار صورت گرفته است:

- Arduino IDE: این نرم‌افزار محیطی یکپارچه برای برنامه‌نویسی Arduino و بردهای مشابه به زبان‌های C و C++ می‌باشد. علاوه بر تسهیل نوشتن و کامپایل نمودن کد، این نرم‌افزار در زمینه برنامه‌ریزی بردها نیز کمک به سزایی می‌کند. یکی از مهم‌ترین خواص این برنامه، مجرد ساختن کار کردن با ورودی-خروجی‌های برد می‌باشد. به دلیل سازگاری بالا با بردهای Arduino، تا کنون تعداد زیادی کتابخانه برای این محیط برنامه‌نویسی تولید و ارائه شده است [۴۰].
- Xilinx ISE: نرم‌افزار ISE محیطی جامع برای توسعه و پیاده‌سازی سخت‌افزارهای برنامه‌پذیر (FPGA، CPLD و حتی ASIC) می‌باشد. این نرم‌افزار علاوه بر فراهم آوردن محیط برنامه‌نویسی برای زبان‌های توصیف سخت‌افزار[1]، قابلیت‌های شبیه‌سازی نرم‌افزاری، سنتر و پیاده‌سازی، برنامه‌ریزی سخت‌افزار و همچنین تجزیه و تحلیل طرح‌ها را ارائه می‌کند [۴۱]. البته در چند سال اخیر، نرم‌افزار Vivado شرکت Xilinx تقریبا جایگزین این نرم‌افزار شده است ویژگی‌های بیشتری نیز برای استفاده طراحان فراهم ساخته است [۴۲]. بنابراین از ISE برای کار با FPGAها قدیمی‌تر شرکت Xilinx استفاده می‌شود.

[1] Hardware Description Language (HDL)

۳-٤- کتابخانه‌های مورد استفاده

در مراحل توسعه این طرح، از تعدادی کتابخانه نرم‌افزاری برای برنامه‌ریزی میکروکنترلر و همچنین تعدادی IP Core برای پیاده‌سازی ساخت‌افزاری FPGA استفاده شده است که ویژگی‌ها و توانایی‌های موارد قابل توجه بیان شده است.

۳-۴-۱- کتابخانه Due CAN

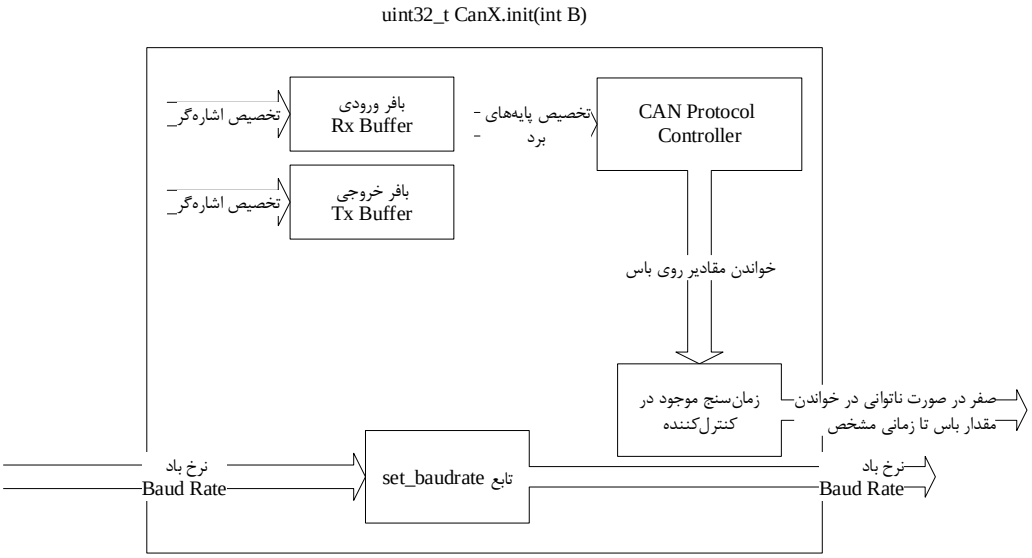
کتابخانه Due CAN به صورت یک مجموعه متن‌باز است که توسط تعدادی توسعه‌دهنده به صورت عمومی طراحی شده و در اختیار همگان قرار گرفته است. هدف اصلی این کتابخانه، پیاده‌سازی عملکرد عمومی باس CAN روی معماری پردازنده‌های SAM می‌باشد [٤٣].

پیاده‌سازی کلی این کتابخانه در یک لایه و به صورت شی‌گرایانه انجام شده است. در ادامه، توضیحات مختصر توابع کلیدی و انواع داده لازم به منظور کار با این کتابخانه ارائه شده است.

تابع uint32_t CanX.init(int B): وظیفه این تابع فعال‌سازی باس CAN شماره X با نرخ باد[1] برابر مقدار B می‌باشد. خروجی این تابع در صورت موفقیت مقدار B بوده و در غیر این صورت عدد صفر می‌باشد. اساسی‌ترین عمل این تابع، محاسبه بخش‌های زمانی (زمان بیتی) باس CAN بر اساس مقدار نرخ باد ورودی می‌باشد. به عنوان مثال، اجرای تابع به صورت ()Can0.init مقدار 125000 را بازگردانی کرده و اجرای آن به صورت (1250000)Can0.init نیز مقدار صفر را باز می‌گرداند. شکل ۳-۷ نمودارد بلوکی عملکردی این تابع را نمایش می‌دهد.

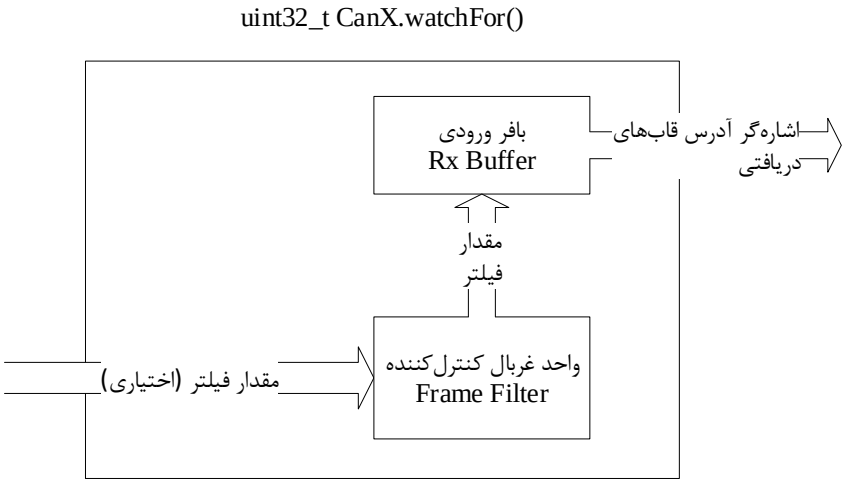
نوع داده CAN_FRAME: این نوع داده یک ساختار[2] برنامه‌نویسی بوده که شامل بخش‌های قابل تغییر یک قاب CAN نظیر شناساگر (اولویت)، محموله داده و چندین بخش دیگر می‌باشد. بخش‌های نظیر کد CRC و اندازه داده (بر اساس DLC) در هنگام ارسال بسته محاسبه می‌شود.

[1] Baud Rate
[2] Struct



شکل ۳-۷: نمودار بلوکی عملکردی تابع init.

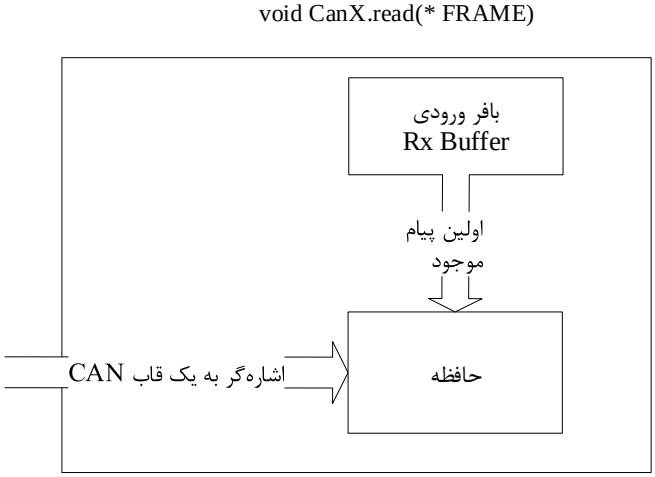
تابع uint32_t CanX.watchFor(): این تابع پین‌های باس CAN شماره X را فعال کرده و در نتیجه، در ادامه می‌توان تمامی ترافیک انتقالی روی باس را مشاهده نمود. مقدار بازگشتی این تابع، آدرس محل شروع ذخیره پیام‌های دریافتی روی کنترل‌کننده X می‌باشد. در صورت فرخوانی این تابع، هر پیام موجود روی باس CAN درون بافر ورودی[1] قرار گرفته و از طریق تابع read قابل مشاهده خواهد بود. شکل ۳-۸ نمودار بلوکی عملکردی این تابع را نمایش می‌دهد.



شکل ۳-۸: نمودار بلوکی عملکردی تابع watchFor.

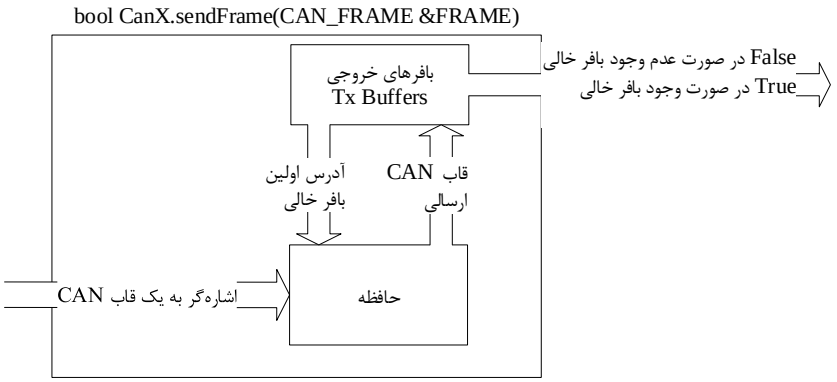
[1] Buffer

تابع void CanX.read(* FRAME): این تابع آخرین قاب پیام موجود روی بافر ورودی CAN را بازیابی کرده و در ادامه می‌توان محتوای موجود را مورد بررسی قرار داد. قاب پیام دریافت، از طریق اشاره‌گر FRAME قابل دسترسی می‌باشد. شکل ۳-۹ نمودار بلوکی عملکردی این تابع را نمایش می‌دهد.



شکل ۳-۹: نمودار بلوکی عملکردی تابع read.

تابع bool CanX.sendFrame(CAN_FRAME &FRAME): این تابع تلاش می‌کند که قاب FRAME را از طریق CAN شماره X ارسال نماید. طبیعتا ممکن است به دلیل اولویت پیام و داوری باس CAN، این قاب در لحظه ارسال نشود. بنابراین قاب پیام در بافر خروجی ذخیره شده و در اولین زمان ممکن ارسال خواهد شد. خروجی این تابع، موفقیت آمیز بودن یا نبودن ارسال قاب پیام را مشخص می‌کند. شکل ۳-۱۰ نمودار بلوکی عملکردی این تابع را نمایش می‌دهد.



شکل ۳-۱۰: نمودار بلوکی عملکردی تابع sendFrame.

تابع uint16_t CanX.setRXFilter(int id, int mask, bool extended): این تابع امکان غربال کردن داده‌های دریافتی را فراهم می‌کند. این غربال می‌تواند بر اساس شناساگر (ID)، محتوا (mask) و یا نوع پیام (extended) و یا هر سه باشد. بنابراین از طریق این عملکرد، می‌توان از قاب‌های داده فاقد اهمیت صرف نظر کرده و تنها به پردازش داده‌های مهم پرداخت. خروجی این تابع، نشان می‌دهد که پیام دریافتی با غربال مشخص در چه آدرسی ذخیره شده است. عملکرد نهایی مانند تابع WathcFor() می‌باشد، با این تفاوت که در هر زمانی قابل فراخوانی بوده و از طریق آن می‌توان مقدار غربال‌ها را تغییر داد.

تابع uint16_t CanX.available(): با فراخوانی این تابع، ثبات FIFO ورودی کنترل‌کننده CAN بررسی شده و در صورت وجود پیام، تعداد پیام‌های موجود بازگردانده می‌شود.

تابع uint32_t CanX.beginAutoSpeed(): باس را با نرخ‌های بیتی مختلف شنود کرده (از طریق تابع init) و در صورتی که انتقالی با یکی از سرعت‌های مشخص شده تشخیص دهد، مقدار آن را بازگردانی می‌کند [۴۳].

۳-۴-۲- liteCAN IP Core

هسته liteCAN توسط جامعه متن‌باز توسعه یافته است و توسعه آن در دانشگاه USTC کشور چین آغاز شده است. هدف این IP Core پیاده‌سازی جامعه تمامی ویژگی‌های باس CAN می‌باشد. بنابراین،

محدودیت‌های روی ارسال قاب‌های پیام CAN اعمال شده است، به طور مثال، محموله ارسالی حداکثر می‌تواند برابر ۴ بایت باشد. این IP Core به صورت Soft می‌باشد و در سطح انتقال ثبات[1] و با استفاده از زبان System Verilog نوشته شده است [٤٤]. به همین علت، به منظور استفاده از آن در این پروژه، کدهای موجود این هسته به زبان Verilog بازنویسی شده‌اند.

پیاده‌سازی این IP Core در چهار لایه انجام شده است. توضیحات و عملکرد هر لایه از بالا به پایین در ادامه بیان شده است:

لایه اول (لایه TOP): این لایه مسئول بر قراری اتصالات میان لایه فیزیکی FPGA و لایه‌های پایین‌تر می‌باشد. عملکردهایی مانند مدیریت کلاک، بررسی صحت ورودی و خروجی‌ها در این لایه به انجام می‌رسند.

لایه دوم (لایه FIFO): این لایه مسئول مدیریت بافرهای FIFO ورودی و خروجی می‌باشد.

لایه سوم (لایه Packet): وظیفه این لایه، تولید قاب‌های خروجی و تفسیر قاب‌های ورودی می‌باشد و همچنین ارسال و دریافت قاب‌ها از بافرهای FIFO می‌باشد.

لایه چهارم (لایه BIT): این لایه، پایین‌ترین لایه هسته ذکر شده می‌باشد و وظیفه‌ی آن، ارسال و دریافت پیام‌های لایه‌ی بالاتر در سطح بیت می‌باشد [٤٤].

به منظور استفاده از قابلیت‌های این هسته، نیاز است که با واسطهای ارائه شده در بالاترین سطح کار شود. به همین منظور، درگاه[2]های مهم این لایه در ادامه شرح داده شده‌اند:

LOCAL_ID: مقدار شناساگر گره را مشخص می‌کند.

default_c_PTS: تعداد کوانتوم زمان بخش PROP_SEG را تعیین می‌کند.

default_c_PBS1: تعداد کوانتوم زمان بخش PHASE_SEG 1 را تعیین می‌کند.

[1] Register Transfer Level

[2] Port

default_c_PBS2: تعداد کوانتوم زمان بخش PHASE_SEG 2 را تعیین می‌کند.

rstn: سیگنال بازنشانی سیستمی را دریافت می‌کند.

CLK: سیگنال کلاک سیستمی را دریافت می‌کند.

can_rx: سیگنال دریافت داده باس CAN می‌باشد.

can_tx: سیگنال ارسال داده باس CAN می‌باشد.

tx_valid: مشخص می‌کند که گره می‌خواهد داده‌ای ارسال کند یا خیر.

tx_data: داده ارسالی روی باس CAN را مشخص می‌نماید.

rx_valid: مشخص می‌کند که آیا گره در حال دریافت داده‌ای صحیح است یا خیر.

rx_data: داده‌ی دریافت شده از باس روی این درگاه قرار می‌گیرد [٤٤].

۳-۴-۳- CTU CAN FD IP Core

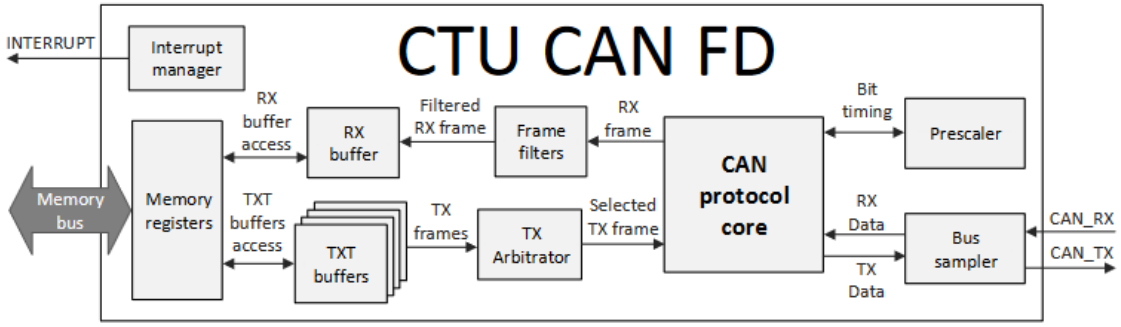
این IP Core نیز یک هسته مالکیت معنوی متن باز بوده که توسط متخصصان دانشگاه Czech Technical University توسعه یافته است. این هسته یک هسته Soft است که تماما توسط زبان VHDL نوشته شده است و هدف آن، مطابقت کامل با استاندارد CAN FD می‌باشد. تعداد بایت داده ارسالی در هر فریم طبق استاندارد می‌تواند تا ۶۴ بایت باشد. علاوه بر قابلیت‌های CAN FD، این هسته از باس APB[1] نیز پشتیبانی می‌کند؛ بنابراین به منظور استفاده از این هسته، باید از طریق APB با آن ارتباط برقرار نموده و طبیعتا از قابلیت‌های نظیر وقفه بهره برد [٤٥].

شکل ۳-۱۱ نمودار بلوکی این هسته را نشان می‌دهد. همانطور که در این شکل پیداست، اصلی‌ترین بخش این IP Core هسته پروتکل CAN می‌باشد که مسئول پیاده‌سازی تمامی عملکردهای ذکر شده

[1] Advanced Peripheral Bus

در لایه لینک داده CAN FD می‌باشد. ارتباط این بلوک با لایه فیزیکی از طریق نمونه‌گیر باس[1] انجام می‌شود که مسئول خواندن و یا نوشتن اطلاعات سریال دریافتی از باس می‌باشد [٤٦].

در سمت دیگر هسته پروتکل CAN FD، اجزاء پردازش پیام‌های ارسالی و یا دریافتی قرار دارد. در صورت تشخیص رسیدن پیام در هسته پروتکل، نخست از غربال‌های قاب[2] عبور کرده و در صورتی که با آن‌ها مطابقت داشت، به بافر ورودی ارسال می‌شود که توسط حافظه قابل خواندن است. در صورت ایجاد دستور ارسال پیام نیز پیام ارسالی نخست در بافرهای خروجی قرار می‌گیرد. سپس با استفاده از عملکرد بلوک داور، تشخیص داده می‌شود که پیام قابل ارسال است یا خیر. در ادامه نیز پیام‌ها به صورت FIFO از بافر خارج شده و روی باس قرار می‌گیرند [٤٦].



شکل ۳-۱۱: نمودار بلوکی هسته CTU CAN FD [٤٦].

به منظور استفاده از قابلیت‌های این هسته، نیاز است که با واسط‌های ارائه شده در بالاترین سطح (ماژول CAN_APB_TOP) کار شود. به همین منظور، درگاه‌های مهم این لایه در ادامه شرح داده شده‌اند:

aclk: سیگنال کلاک سیستمی را دریافت می‌کند.

arstn: سیگنال بازنشانی سیستمی را دریافت می‌کند.

CAN_rx: سیگنال دریافت داده باس CAN FD می‌باشد.

[1] Bus Sampler

[2] Frame Filter

CAN_tx: سیگنال ارسال داده باس CAN FD می‌باشد.

همچنین به منظور کنترل‌کننده با خارج، از واسط AMBA APB استفاده می‌شود. بنابراین علاوه بر درگاه‌های فوق، درگاه‌های واسط APB نظیر paddr، pwdata و prdata نیز درون این هسته تدارک دیده شده‌اند [45].

۳-۵- پیاده‌سازی انجام شده

در این بخش، پیاده‌سازی انجام شده مورد بررسی قرار می‌گیرد. به دلیل ساختار طبیعی پروژه، این بررسی در دو بخش سخت‌افزاری و نرم‌افزاری انجام خواهد شد. بخش سخت‌افزاری، شامل توضیحات منابع، قطعات و اتصالات سخت‌افزاری بود و بخش نرم‌افزاری نیز شامل توضیحات شیوه برنامه‌نویسی روی میکروکنترلر (از طریق زبان‌های برنامه‌نویسی) و FPGA (از طریق زبان‌های توصیف سخت‌افزار) می‌باشد.

۳-۵-۱- ساختار سخت‌افزاری

همانطور که ذکر شد، لایه فیزیکی باس CAN و همچنین CAN FD از دو رشته سیم تشکیل شده است که سیگنال مورد انتقال را به صورت تفاضلی مخابره می‌کنند. بنابراین برای ایجاد این باس خطی، دو رشته سیم در نظر گرفته می‌شود که البته مشابه با اغلب باس‌های تفاضلی، نیازمند مقاومت خاتمه[1] نیز می‌باشد. در ادامه به منظور ایجاد اتصال بین باس و بردهای پردازشی، یک فرستنده-گیرنده TJA 1050 استفاده می‌شود که وظیفه آن، تبدیل سیگنال‌های تفاضلی به سیگنال سریال قابل دریافت و پردازش روی بردها می‌باشد.

اتصالات در لایه ورودی-خروجی بردها نیز ساختار خود را دارد. در سمت برد AVS3S400، می‌توان از هریک از پین‌های موجود استفاده نمود، البته به این شرط که در زمان سنتز و پیاده‌سازی کد توصیف سخت‌افزار روی این برد، پین‌های مورد نظر به عنوان مراجع اتصال CAN در نظر گرفته شوند. برد

[1] Termination Resistor

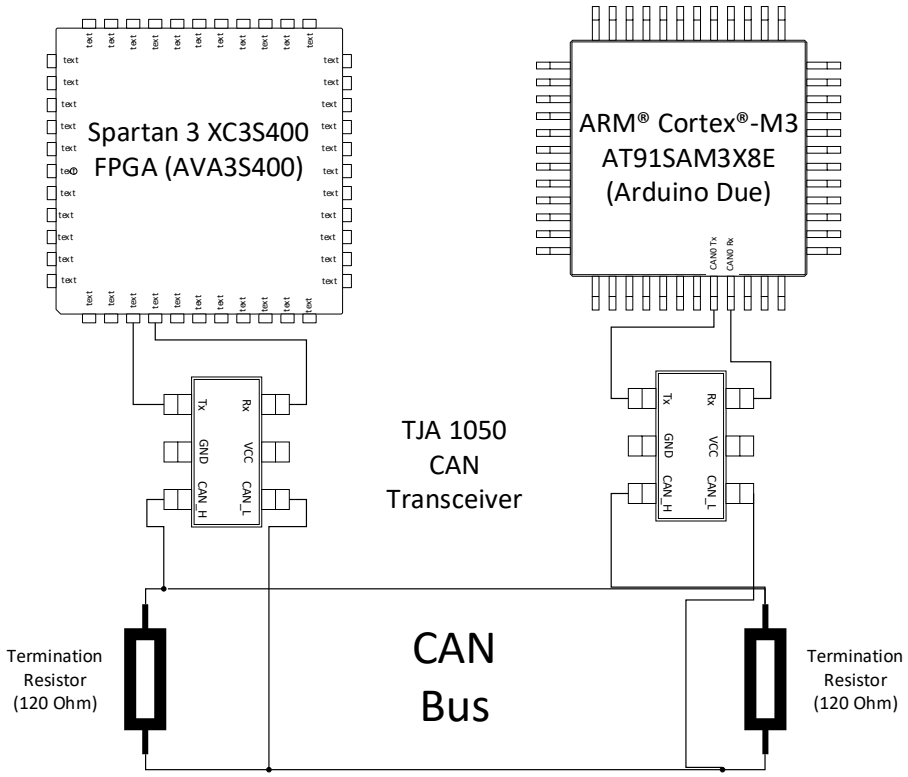
Arduino Due نیز طبیعتا انعطاف‌پذیری به مراتب پایین‌تری داشته و اتصال باس CAN روی این برد باید از طریق یکی از دو درگاه CAN0 (پین‌های ۶۸ و ۶۹) و یا CAN1 (پین‌های ۵۳ و ۶۶) انجام شود. دو پین اتصال مورد استفاده روی بردهای پردازشی، در قالب پین‌های اتصال سریال Tx و Rx بوده که به ترتیب وظیفه ارسال و دریافت سیگنال به فرستنده-گیرنده را بر عهده دارند.

هدف اصلی در این پروژه برقراری ارتباط از طریق یک باس CAN-FD بین یک میکروکنترلر و یک FPGA بوده است که به دلیل عدم امکان تهیه برد میکروکنترلر با CAN-FD سناریوهای دیگری در نظر گرفته شده است که در ادامه تشریح خواهد شد.

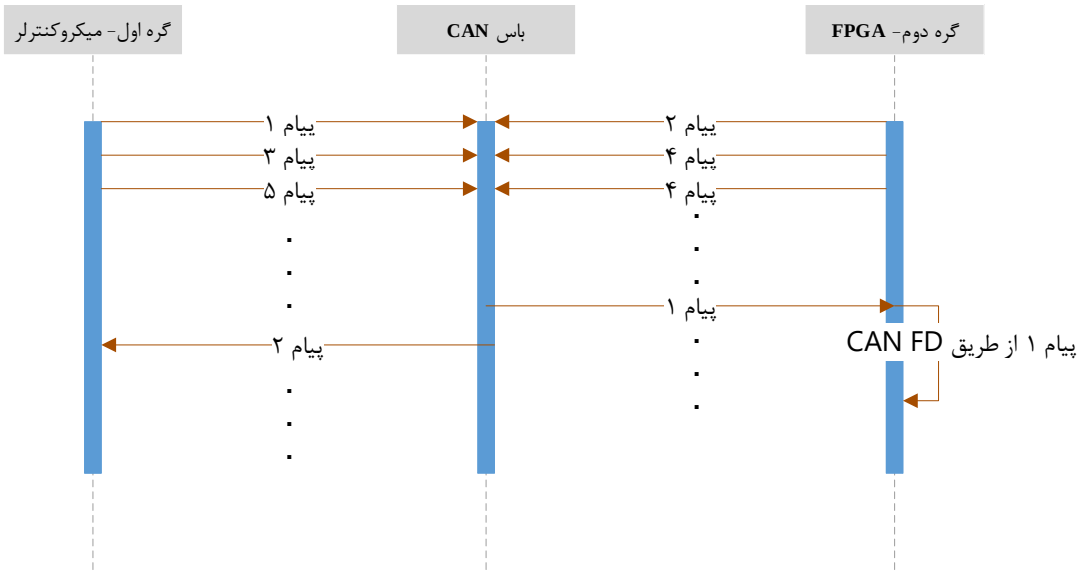
در این پروژه، دو پیکربندی متفاوت پیاده‌سازی شده است و هر یک مورد بررسی و ارزیابی قرار گرفته‌اند. در پیکربندی اول، ارتباطی بین میکروکنترلر و برد FPGA از طریق باس CAN ایجاد شده است و درون FPGA نیز دو هسته باس CAN FD قرار گرفته است که داده دریافتی از واسط CAN بین آن‌ها منتقل می‌شود. شکل ۳-۱۲ شماتیک سخت‌افزار و اتصالات انجام شده در پیکربندی اول را نشان داده و شکل ۳-۱۳ نیز نمودار توالی[1] این پیکربندی را نشان می‌دهد.

پیکربندی دوم، با هدف ارزیابی جامع‌تر باس CAN FD پیاده‌سازی شده است. به همین منظور، دو کنترل‌کننده باس CAN FD روی FPGA قرار گرفته‌اند و درگاه‌های انتقال و دریافت هریک روی پایه‌های ورودی-خروجی برد FPGA قرار گرفته است. در پایان نیز ارتباط دو هسته CAN FD از طریق فرستنده-گیرنده TJA1050 برقرار شده است و بررسی‌های لازم روی این معماری ارتباطی انجام شده‌اند. شکل ۳-۱۴ شماتیک سخت‌افزار و اتصالات انجام شده در پیکربندی دوم را به تصویر می‌کشد. همچنین شکل ۳-۱۵ نیز نمودار توالی این پیکربندی را نشان می‌دهد.

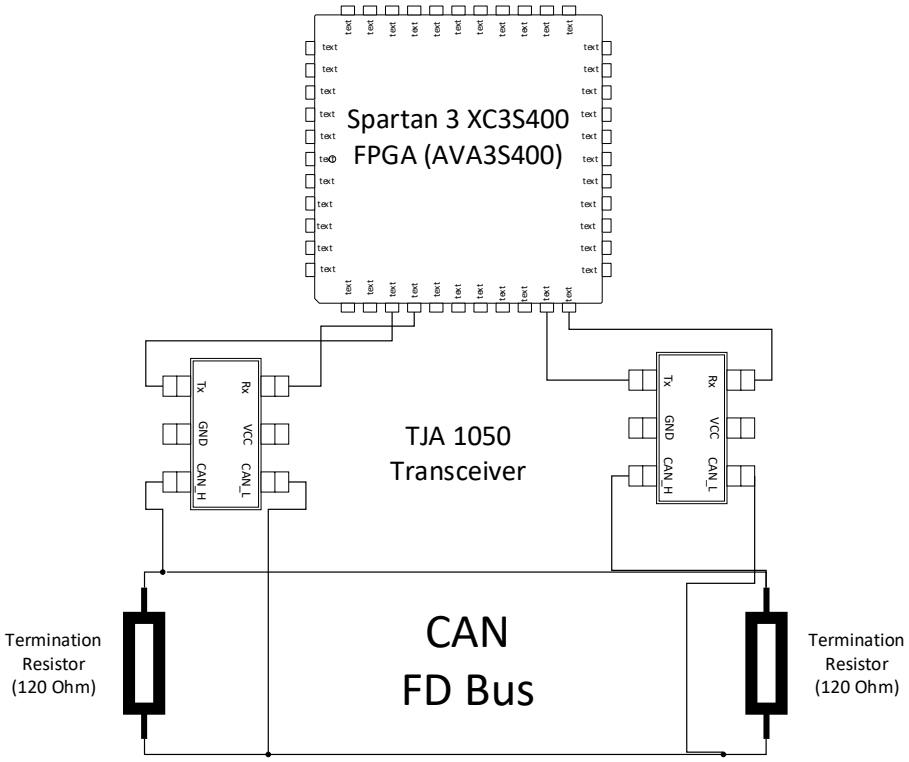
[1] Sequence Diagram



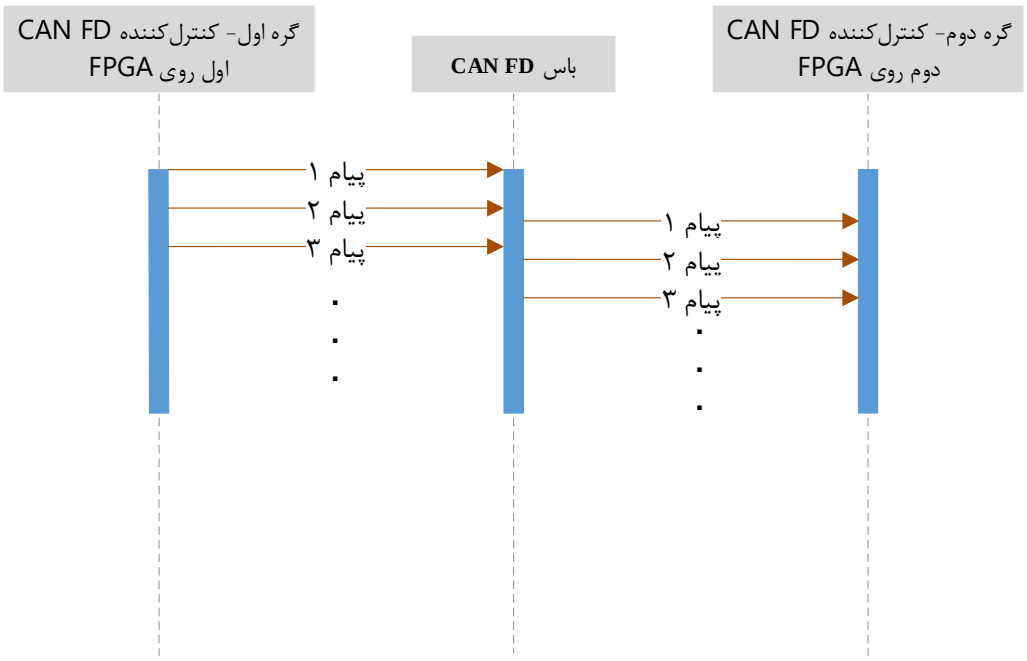
شکل ۳-۱۲: شماتیک ساختار سخت‌افزاری پیکربندی اول.



شکل ۳-۱۳: نمودار توالی پیکربندی اول.



شکل ۳-۱۴: شماتیک ساختار سخت‌افزاری پیکربندی دوم.



شکل ۳-۱۵: نمودار توالی پیکربندی دوم.

۳-۵-۲- ساختار نرم‌افزاری

پیاده‌سازی نرم‌افزاری این طرح از دو بخش مجزا ولی مرتبط تشکیل می‌شود که هر یک بخشی از ارتباط را تشکیل می‌دهند: پیاده‌سازی با زبان توصیف سخت‌افزار به منظور راه‌اندازی FPGA و همچنین پیاده‌سازی با زبان‌های نرم‌افزاری به منظور راه‌اندازی میکروکنترلر. با وجود تشابه عملکرد این دو بخش، طراحی و کد کاملا متفاوتی دارند. در FPGA، کد مورد نظر به صورت ترکیبی از دو زبان توصیف سخت‌افزار VHDL و Verilog و در میکروکنترلر، کد از طریق زبان C/C++ نوشته شده است. همانطور که ذکر شد، پیاده‌سازی میکروکنترلر تنها مختص پیکربندی اول است و پیکربندی دوم تنها بر روی FPGA پیاده‌سازی شده است.

طراحی هر دو بخش پیاده‌سازی به صورت معماری لایه‌ای بوده است که این معماری به طور کلی از دو لایه کاربرد و لایه اتصال تشکیل شده است. لایه کاربرد، لایه بالاتر و محل پیاده‌سازی قابلیت‌های CANOpen می‌باشد و لایه اتصال، مسئول ارسال و دریافت قاب‌های پروتکل CANOpen از طریق ارتباط فیزیکی موجود و به کمک کتاب‌خانه‌های نرم‌افزاری می‌باشد.

لایه کاربرد در میکروکنترلر در قالب یک کتابخانه با نام CANOpen.h طراحی شده است. واسط‌های اصلی ارائه شده توسط این کتابخانه عبارتند از:

CO_FRAME* send_co_frame(char *): این تابع رشته پیام دلخواه را از طریق ورودی دریافت کرده و پیام را به صورت قاب‌های پروتکل CANOpen در می‌آورد. در پایان اجرا نیز پیام‌های تولیدی را باز می‌گرداند.

char * receive_co_frame(CO_FRAME*): این تابع نیز در صورت فراخوانی، در انتظار یک پیام می‌نشیند و اگر پیامی مطابق پروتکل CANOpen دریافت کند، محموله آن محاسبه کرده و بازگردانی می‌کند.

در بطن کتابخانه طراحی شده CANOpen.h، عملکردهای لازم برای تولید یک پیام مطابق پروتکل CANOpen از روی محموله دریافتی طراحی شده است. مهم‌ترین عملکردهای انجام شده شامل

قطعه‌بندی محموله، محاسبه بخش‌های قاب محموله، قراردادن هر قطعه داخل قاب مربوطه و در پایان ارسال از طریق توابع لایه اتصال می‌شود. لایه اتصال، به کمک کتابخانه‌های موجود توسعه داده شده است که توضیحات مربوطه در بند ۴-۳- ارائه شد. لازم به ذکر است که این پیاده‌سازی CANOpen به هیچ وجه پیاده‌سازی کاملی از CANOpen و تمامی ویژگی‌های آن نبوده و تنها قابلیت‌های اساسی CANOpen یعنی ارسال و دریافت داده پیاده‌سازی شده‌اند.

در سمت دیگر پیاده‌سازی یعنی راه‌اندازی FPGA، دو پیکربندی پیاده‌سازی شده است. در پیکربندی اول، عملکردی مشابه میکروکنترلر پیاده‌سازی شده است؛ با این تفاوت که در اینجا بجای تولید کتابخانه نرم‌افزاری، ماژول‌های سخت‌افزاری به منظور ایجاد ارتباط CAN و همچنین CAN FD طراحی شده‌اند. تلاش شده است که کد توصیف سخت‌افزار تا حد امکان مشابه کد نرم‌افزاری بوده و دارای اسامی و عملکرد یکسانی باشد. در لایه اتصال FPGA نیز از طریق هسته‌های معرفی شده در بند ۴-۳- ارتباط CAN و CAN FD برقرار شده و تبادل داده صورت می‌پذیرد. CAN FD نیز به صورت داخلی درون FPGA پیاده‌سازی شده است و ارتباط آن داخلی است.

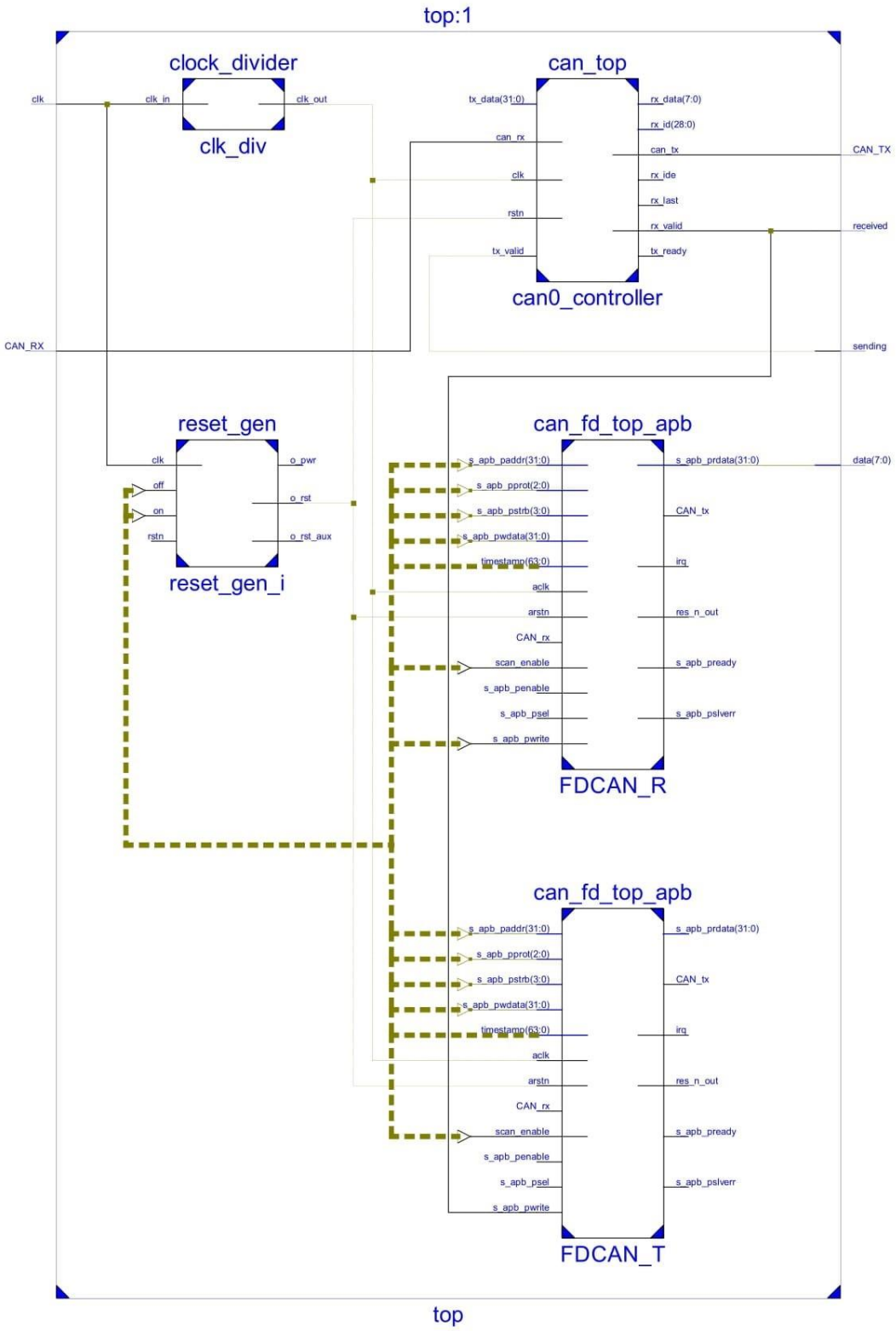
در پیکربندی دوم، تنها ارتباط CAN FD پیاده‌سازی شده است و تفاوت آن با پیکربندی اول، ایجاد درگاه‌های کنترل‌کننده‌های CAN FD روی پایه‌های FPGA است که امکان بررسی دقیق‌تر باس CAN FD را ایجاد می‌کند. با توجه به انعطاف‌پذیری طراحی FPGA، می‌توان از هر پایه دلخواهی برای درگاه‌های ورودی و خروجی دو کنترل‌کننده استفاده نمود. در ادامه نیز این ارتباط با متغیرهای زمانی متفاوت مورد بررسی‌های مختلف قرار گرفته است.

مقدار نرخ بیت در حالت کلی پیکربندی اول (باس CAN بین دو برد پردازشی) نیز با توجه به کاربردهای متداول پروتکل CANOpen، برابر مقدار ۱۲۵ Kbps قرار داده شده است. البته که در هردو پیکربندی، ارتباط با سرعت‌های متداول بالاتر و همچنین نرخ‌های بیتی بالاتر از ۱ Mbps نیز پیاده‌سازی شده و مورد بررسی و ارزیابی قرار گرفته است. به همین منظور، مقادیر زمانی مطابق جدول ۲-۳ محاسبه

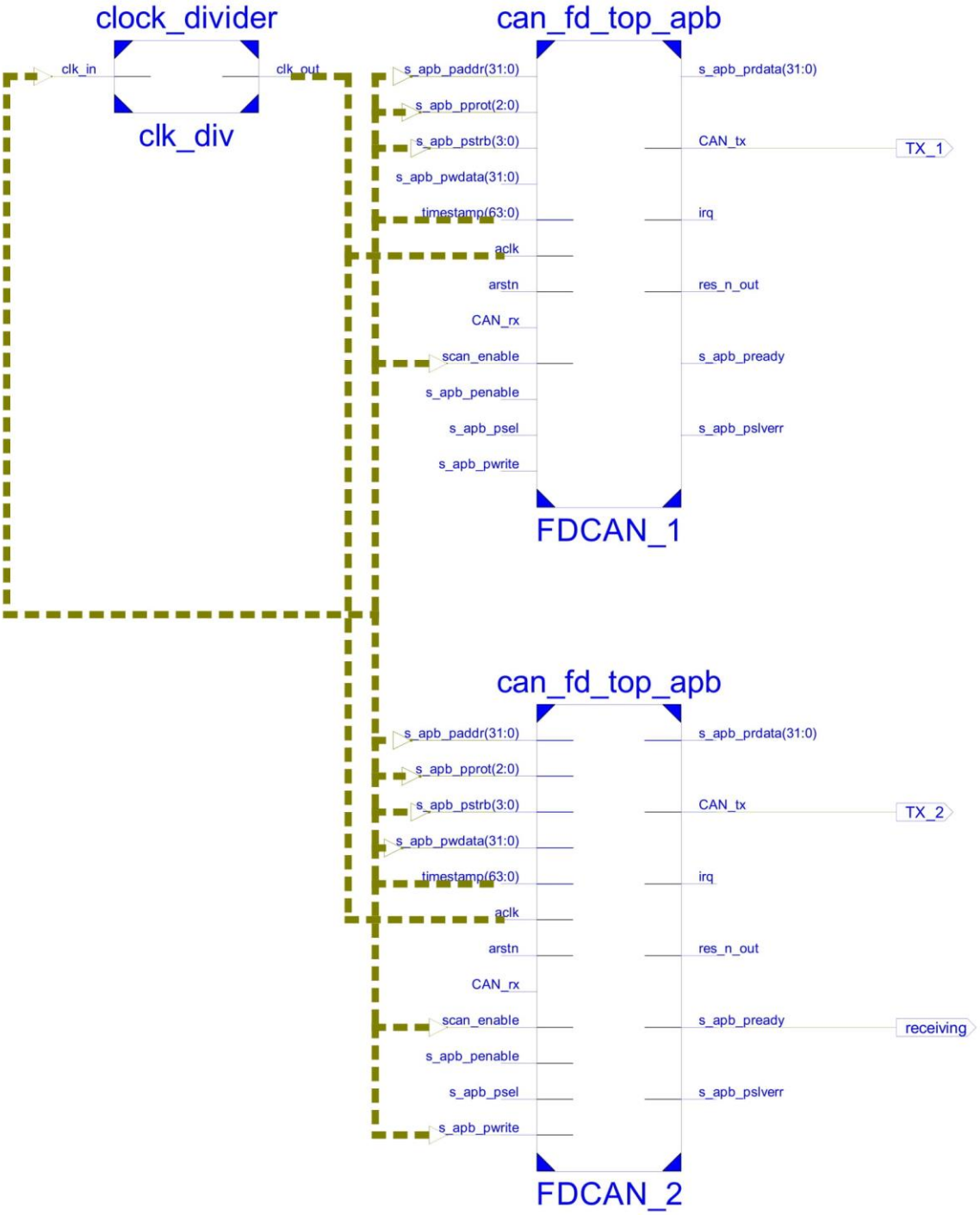
و انتخاب شده‌اند. لازم به ذکر است که فرکانس کلاک پایه برد FPGA برابر ۴۰ MHz و برای میکروکنترلر برابر ۸۴ MHz می‌باشد.

جدول ۳-۲: مقادیر زمانی محاسبه شده برای ارتباط صحیح دو برد تحت باس CAN و CAN FD.

نرخ بیت	۱۲۵ Kbps	۵۰۰ Kbps	۱۰۰۰ Kbps	۲۰۰۰ Kbps (فقط باس CAN FD)	۳۰۰۰ Kbps (فقط باس CAN FD)
عامل Prescaler برای میکروکنترلر	۸۳	۲۰	۶	-	-
عامل Prescaler برای FPGA	۴۰	۱۰	۳	۲	۱
SYNC_SEG	یک کوانتوم زمانی	یک کوانتوم زمانی	یک کوانتوم زمانی	یک کوانتوم زمانی	یک کوانتوم زمانی
PROP_SEG	دو کوانتوم زمانی	دو کوانتوم زمانی	دو کوانتوم زمانی	سه کوانتوم زمانی	سه کوانتوم زمانی
PHASE_SEG 1	یک کوانتوم زمانی	یک کوانتوم زمانی	سه کوانتوم زمانی	سه کوانتوم زمانی	سه کوانتوم زمانی
PHASE_SEG 2	دو کوانتوم زمانی	دو کوانتوم زمانی	سه کوانتوم زمانی	چهار کوانتوم زمانی	چهار کوانتوم زمانی



شکل ۳-۱۶: شماتیک RTL ماژول سطح بالا در FPGA (پیکربندی اول).



شکل ۳-۱۷: شماتیک RTL ماژول سطح بالا در FPGA (پیکربندی دوم).

فصل چهارم

ارزیابی ارتباطات

ارزیابی ارتباطات

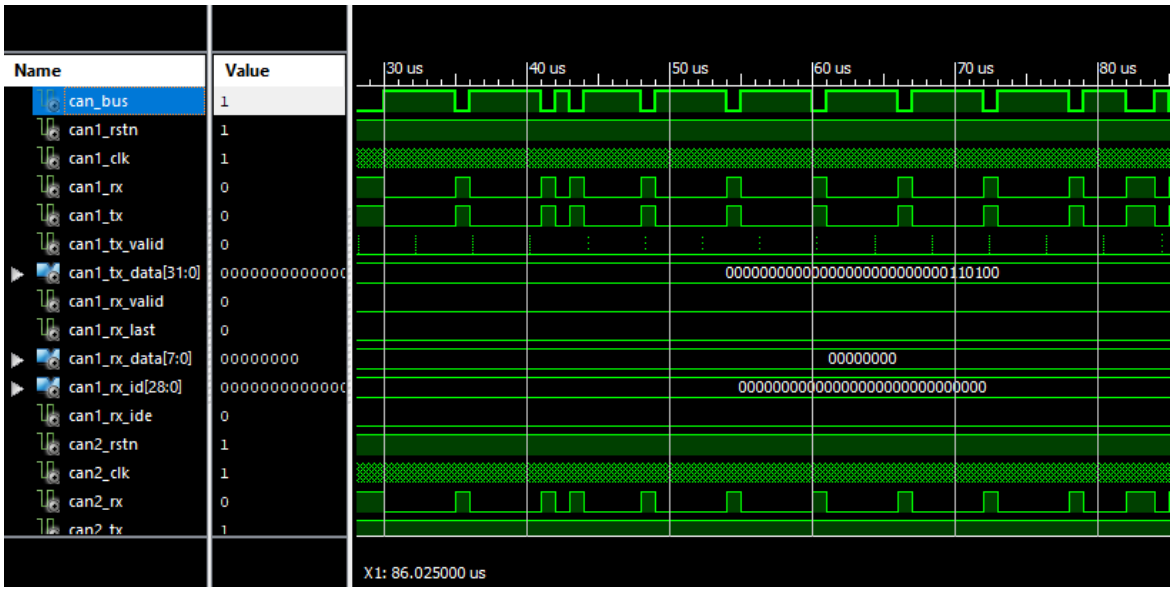
٤-١- صحت عملکرد

در نخستین گام ارزیابی پیاده‌سازی انجام شده، نیاز است که صحت عملکرد ارتباطات ایجاد شده مورد بررسی قرار گیرد. به همین منظور، نخست هر یک از بردها به صورت جداگانه مورد آزمایش قرار گرفته‌اند و سپس پیکربندی‌های پیاده‌سازی شده به صورت یکپارچه مورد آزمایش قرار گرفته‌اند.

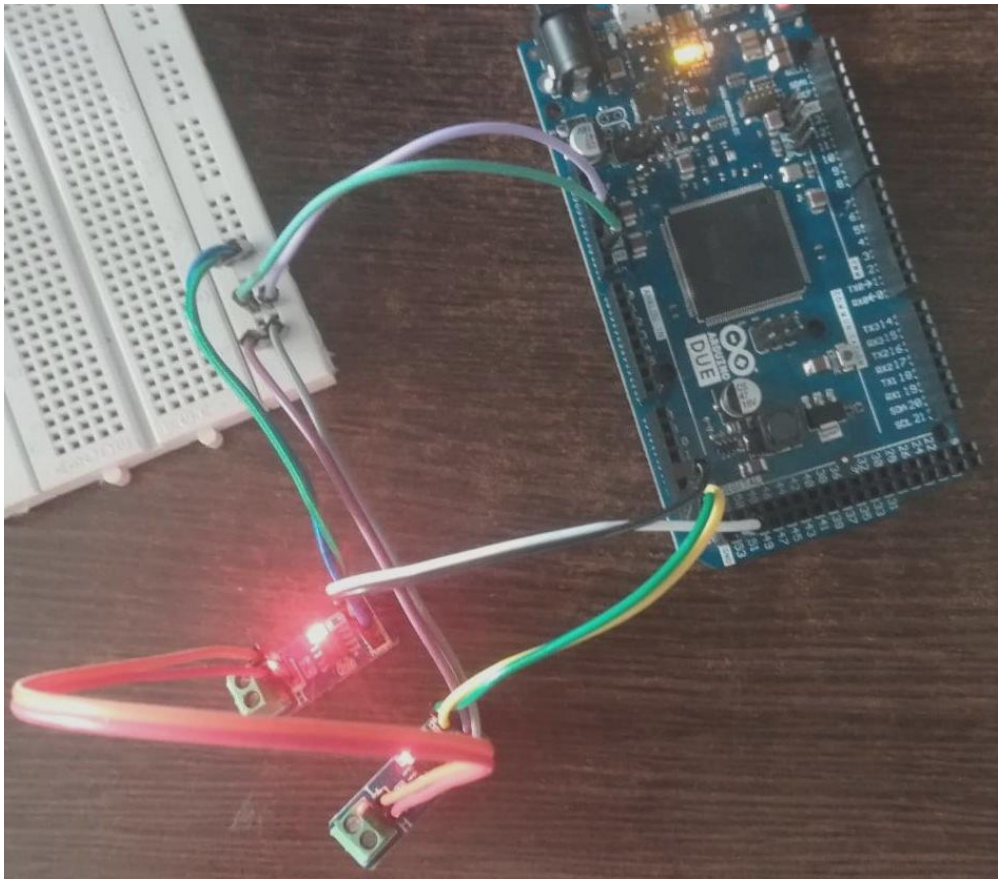
در راستای بررسی عملکرد پیکربندی‌های برد FPGA می‌توان از testbench نرم‌افزاری استفاده نمود. در این بخش، یک مورد testbench نوشتیم که روند آن بدین صورت است که دو مورد ماژول سطح بالای درون محیط آزمون نمونه‌سازی[1] شده و با ارسال و دریافت چندین بسته به صورت متناوب روی خطوط باس، صحت ارتباط بررسی می‌شود. با توجه به سادگی و به طبع انعطاف عملکرد این آزمون، می‌توان از آن برای هر دو پیکربندی پیاده‌سازی شده روی FPGA استفاده نمود. شکل ۴-۱ نمایی از بخش کوچکی از این آزمون نرم‌افزاری را به نمایش می‌گذارد.

در سمت دیگر ارتباط یعنی پیاده‌سازی میکروکنترلر نیز عملکرد کلی مورد ارزیابی قرار گرفته است. به همین منظور، یک آزمون نوشته شده است که بر طبق آن، داده‌ای یکسان از کنترل‌کننده باس CAN اول به دیگری ارسال می‌شود و همین عملکرد در کنترل‌کننده دوم نیز رخ می‌دهد. در پایان نیز صورت دریافت موفق داده در هر کنترل‌کننده، محتوای پیام دریافتی از طریق ارتباط سریال (ارتباط UART روی USB) برای سیستم برنامه‌نویسی ارسال شده و روی صفحه چاپ می‌شود. شکل ۴-۲، شمای مدار این آزمون و شکل ۴-۳ خروجی این آزمون را نشان می‌دهد.

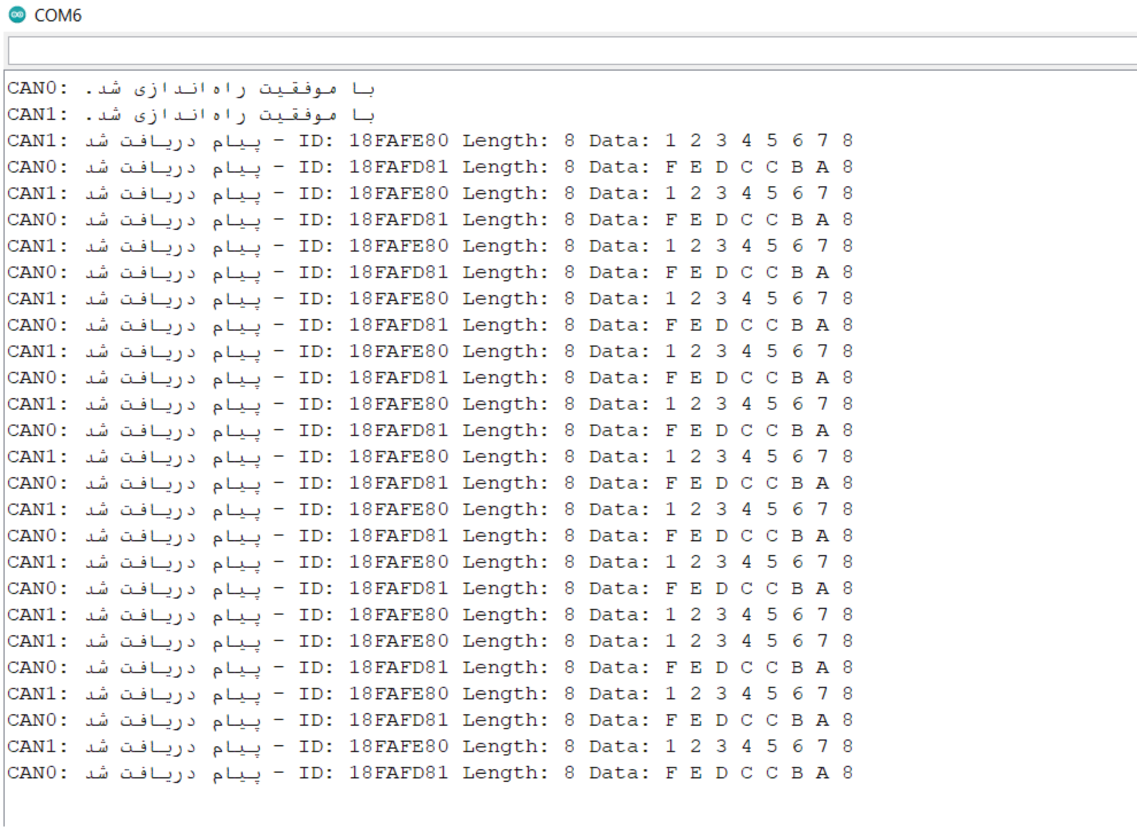
[1] Instantiate



شکل ۴-۱: شبیه‌سازی باس CAN روی FPGA. در این محدوده، کنترل‌کننده اول در حال نوشتن اطلاعات روی باس مشترک می‌باشد.

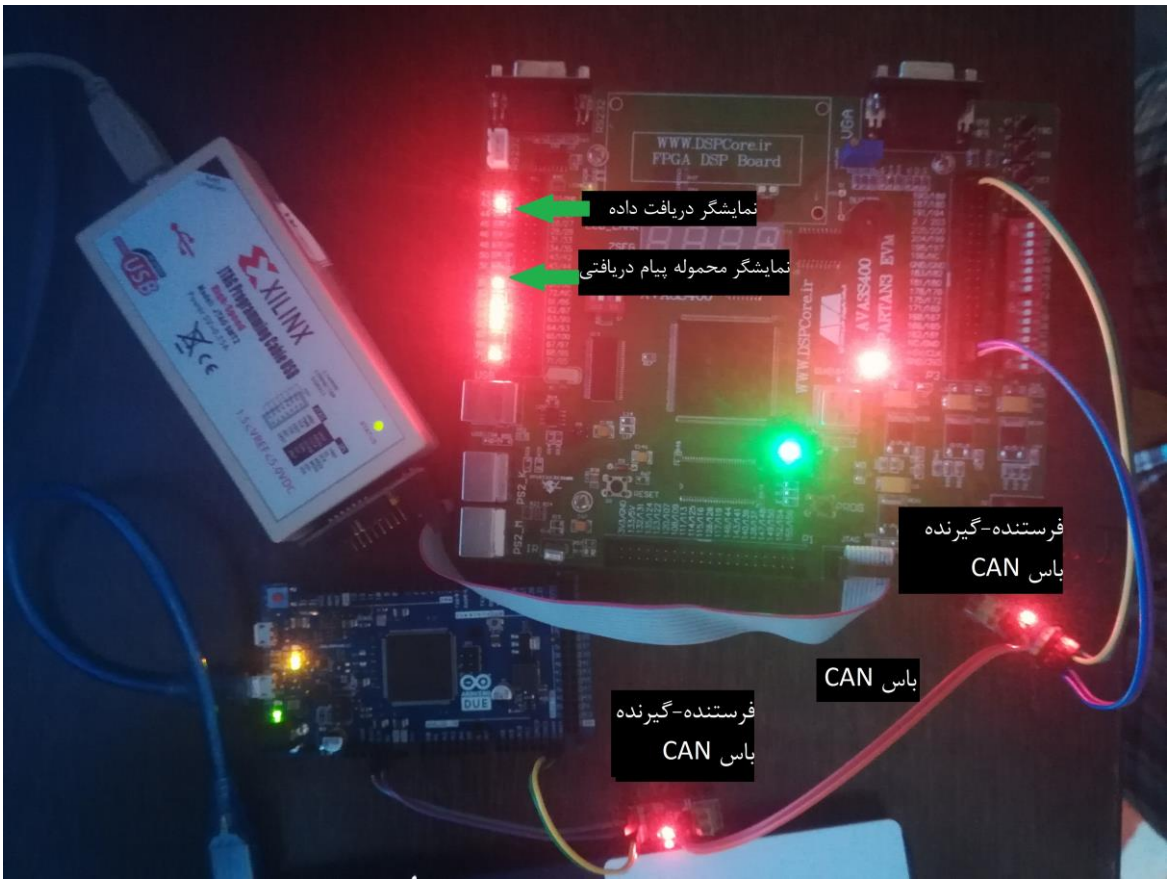


شکل ۴-۲: مدار آزمون کنترل‌کننده‌های CAN میکروکنترلر.



شکل ۴-۳: نتیجه آزمون باس CAN روی میکروکنترلر.

در گام آخر سنجش صحت عملکرد، کلیت پیاده‌سازی هر دو پیکربندی به صورت یکپارچه مورد آزمایش قرار گرفت. به همین منظور، آزمونی در دل کدهای پیاده‌سازی گنجانیده شد که طبق آن، داده‌ها به صورت متناوب توسط هر کنترل‌کننده CAN ارسال می‌شوند. در سمت میکروکنترلر، موفقیت این ارسال و دریافت از طریق چاپ رو کامپیوتر میزبان بررسی می‌شود. در سمت FPGA نیز این موضوع از طریق کنترل دیودهای نوری موجود روی برد FPGA صورت گرفت؛ بدین صورت که ارسال و یا دریافت پیام از طریق دیودهای مشخص اعلام شده و همچنین در صورت دریافت داده، مقدار عددی آن روی دیودهای نوری قابل مشاهده می‌باشد. شکل ۴-۴ نیز شمای کلی این مدار در حین کار را به نمایش می‌گذارد. برای سنجش عملکرد پیکربندی دوم پیاده‌سازی نیز از روشی مشابه استفاده شده است و تنها تفاوت آن با شکل ۴-۴، عدم وجود میکروکنترلر و بازگشتی بودن باس CAN از دو پایه FPGA به دو پایه دیگر می‌باشد.



شکل ۴-۴: مدار بررسی عملکرد ارتباط‌های CAN و CAN FD.

٤-٢- میزان بهره‌وری از منابع

مرحله دیگر ارزیابی، بررسی میزان منابع مصرفی طراحی در هر دو پیکربندی می‌باشد. به همین منظور، میزان استفاده از سخت‌افزار در میکروکنترلر و همچنین FPGA ارائه شده‌اند. در میکروکنترلر، مهم‌ترین منبع موجود، حافظه Flash می‌باشد که طبق اطلاعیه نرم‌افزار Arduino IDE، حدود ۲٪ از حافظه Flash مورد استفاده قرار گرفته است (۱۳.۹ کیلوبایت از ۵۲۵ کیلوبایت حافظه در دسترس). همانطور که مشخص می‌باشد، حافظه ناچیزی برای پیاده‌سازی این طرح استفاده شده است. در مورد FPGA نیز میزان استفاده از بلوک‌های منطقی موجود از طریق نرم‌افزار Xilinx ISE استخراج شده است و برای پیکربندی اول و دوم به ترتیب در جداول ۴-۱ و ۴-۲ قابل مشاهده هستند. در این زمینه نیز همانطور

که قابل مشاهده می‌باشد، پیاده‌سازی انجام شده بهینگی مناسبی داشته و درصد کمی از منابع سخت‌افزاری FPGA را اشغال می‌نماید.

جدول ۴-۱: میزان استفاده از منابع سخت‌افزاری در پیکربندی اول.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	178	7,168	2%	
Number of 4 input LUTs	307	7,168	4%	
Number of occupied Slices	163	3,584	4%	
Number of Slices containing only related logic	163	163	100%	
Number of Slices containing unrelated logic	0	163	0%	
Total Number of 4 input LUTs	307	7,168	4%	
Number used as logic	306			
Number used as a route-thru	301			
Number of bonded IOBs	20	141	14%	
Number of BUFGMUXs	2	8	25%	
Average Fanout of Non-Clock Nets	3.20			

جدول ۴-۲: میزان استفاده از منابع سخت‌افزاری در پیکربندی دوم.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	152	7,168	2%	
Number of 4 input LUTs	173	7,168	2%	
Number of occupied Slices	159	3,584	4%	
Number of Slices containing only related logic	159	159	100%	
Number of Slices containing unrelated logic	0	159	0%	
Total Number of 4 input LUTs	299	7,168	4%	
Number used as logic	173			
Number used as a route-thru	126			
Number of bonded IOBs	12	141	8%	
Number of BUFGMUXs	2	8	25%	
Average Fanout of Non-Clock Nets	3.10			

۴-۳- ارزیابی و مقاومت باس

باس CAN از ابتدا با هدف استفاده صنعتی و به خصوص در صنعت خودرو طراحی شده است؛ به همین علت، طراحی آن از ابتدا به گونه‌ای بوده است که تا حد امکان در مقابل نویزهای محیطی مقاوم باشد [۱۳]. باس CAN FD نیز به دلیل استفاده از لایه فیزیکی مشابه CAN، به صورت بالقوه دارای همین

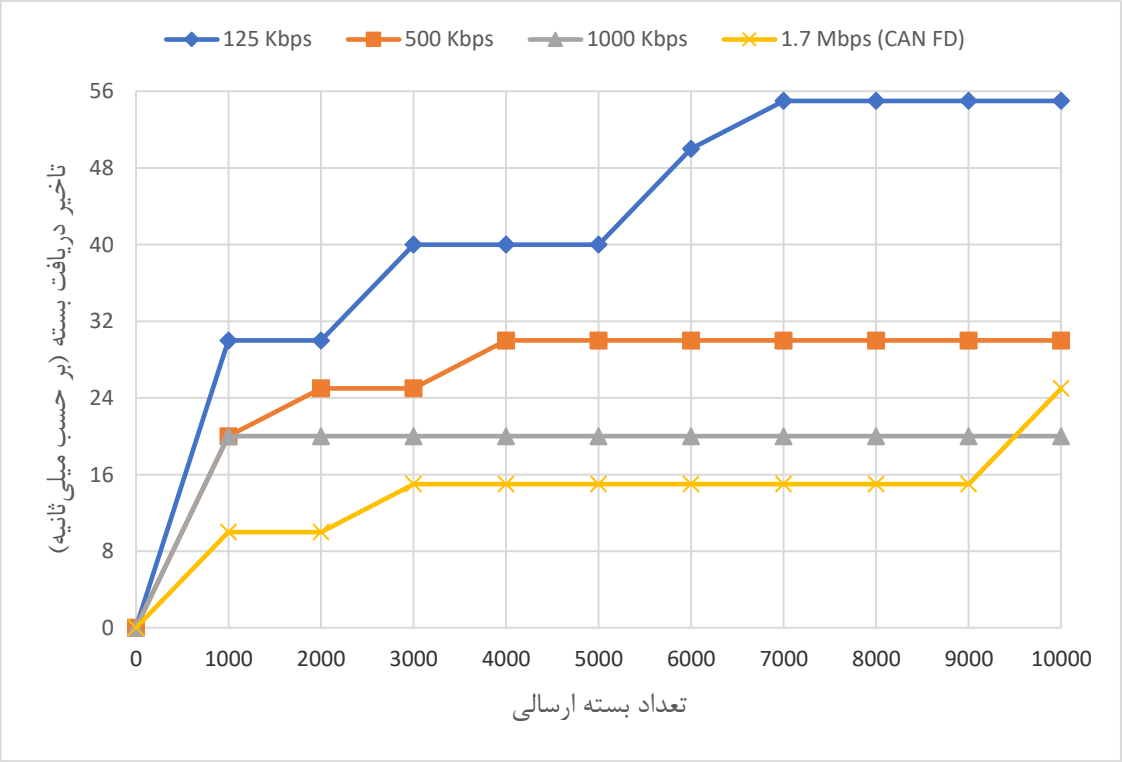
ویژگی‌ها می‌باشد [۲۸]، ولی به دلیل بهبود قابلیت‌های تشخیص خطا در لایه لینک داده باس CAN FD، توانایی این باس در تشخیص خطاهای احتمالی به مراتب بیشتر می‌باشد [۴۷].

به منظور ارزیابی طرح پیاده‌سازی شده، دو نوع آزمون برای هر دو پیکربندی ارائه شده است که هدف آن، بررسی میزان خطا و تاخیر باس‌های CAN و CAN FD در محیط‌های طبیعی و همچنین در مواجه با نویز می‌باشد. در این آزمون‌ها، هر یک از طرفین ارتباط به صورت مداوم و با فاصله زمانی حدود ۲۰ میلی‌ثانیه، بسته‌ای روی باس ارسال می‌کنند. این بسته‌های دارای محموله‌ای عددی بوده که مقدار آن با هر ارسال افزایش می‌یابد و همچنین شناساگر هر یک از بسته‌ها وابسته به گره ارسالی بوده و مقدار ثابت می‌باشد. این مساله باعث ایجاد ترافیکی بسیار بالا روی باس شده که می‌تواند تاخیر و همچنین امکان خطا را به صورت قابل توجهی افزایش دهد. در هر یک از این آزمون‌ها، تعداد ۱۰۰۰۰ هزار بسته با نرخ‌های بیتی متفاوت روی باس ارسال شده است. محموله هر یک از بسته‌ها دارای اندازه‌ای ثابت ۶۴ بیت که برابر حداکثر اندازه قابل پذیرش برای یک بسته CAN است می‌باشد.

شکل ۴-۵، متوسط تاخیر ارسال و دریافت داده در هر دو پیکربندی را بر حسب تعداد بسته‌های ارسالی را در محیط بدون نویز نشان می‌دهد. در این آزمون، حتی پس از ارسال ۱۰۰۰۰ بسته نیز خطایی روی باس‌ها ایجاد نشده بود. تاخیر محاسبه شده، از روی فاصله زمانی ارسال یک بسته تا دریافت آن در سمت گیرنده محاسبه شده است و بین مقادیر به دست آمده در دو سمت ارتباط، بیشترین آن‌ها انتخاب شده است.

همچنین میزان تاخیر پیکربندی دوم در باس CAN FD نیز در شکل ۴-۵ قابل مشاهده می‌باشد. در این آزمون، نخست تلاش شد که باس با سرعت‌های بیشتر از ۲ Mbps مورد آزمایش قرار بگیرد. ولی فرستنده-گیرنده در دسترس توانایی تفسیر سیگنال‌های دریافتی در این نرخ‌ها را نداشت. به همین عالت، تلاش شد که از طریق آزمون و خطا، حداکثر نرخ بیتی قابل استفاده در فرستنده-گیرنده یافت شده و بررسی انجام شده در آن نرخ انجام شود. بنابراین، نرخ بیتی ۱.۷ Mbps انتخاب شده و ارسال و دریافت بسته‌ها با این نرخ انجام گرفت. نتایج قابل مشاهده در شکل ۴-۵ نشان می‌دهند که باس CAN

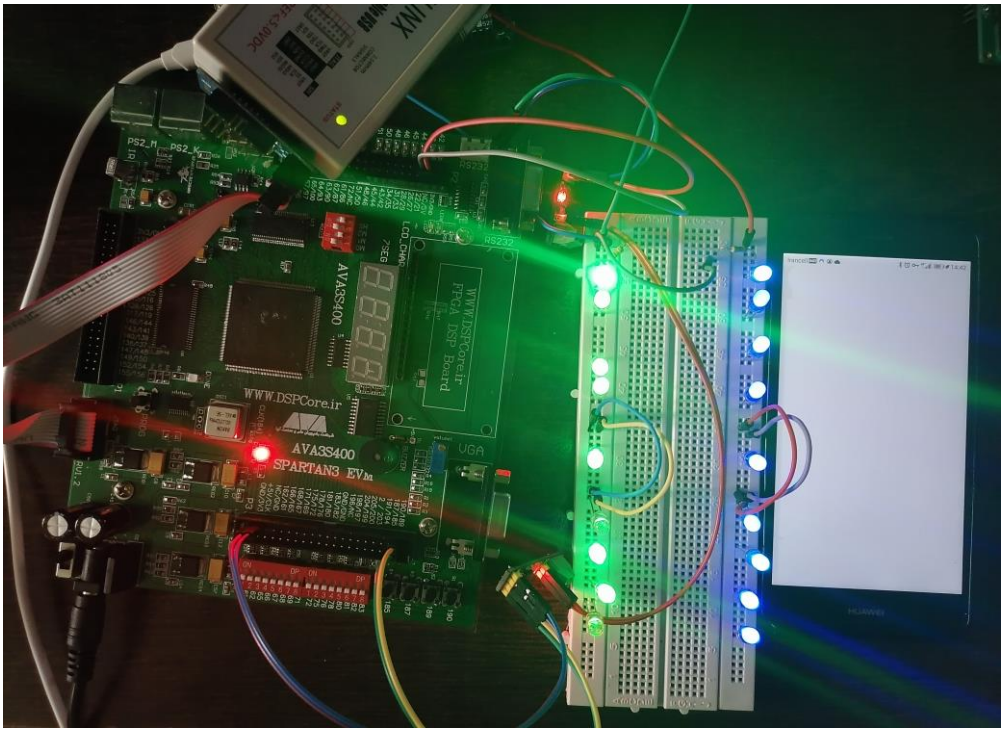
FD در محیط بدون نویز تا حدودی بهتر از CAN استاندارد عمل می‌کند، هر چند که اندکی افزایش تاخیر پس از ۱۰۰۰۰ بسته احتمالا به دلیل بیشتر بودن سربار CAN FD نسبت به CAN استاندارد است.



شکل ۴-۵: تاخیر باس ارتباطی بر حسب ترافیک شبکه در هر دو پیکربندی.

در گام بعدی، آزمون انجام شده را در محیطی با نویز کنترل شده تکرار می‌کنیم. به همین منظور، در مجاورت دو خط انتقال باس CAN، چندین دیود نوری به صورت پیاپی قرار گرفته است که هریک با فرکانسی بین فرکانس باس و فرکانس برد FPGA به صورت نوسانی روشن و خاموش می‌شوند. تعدادی از این دیودها مستقیما روی خطوط باس قرار گرفته و از این طریق روشن و یا خاموش می‌شوند و سایر دیودها از طریق یک برنامه خارجی کنترل شده و به صورت نوسانی روشن و یا خاموش می‌شوند. همچنین به صورت مداوم محدوده خطوط باس تحت تاثیر فرکانس رادیویی تلفن همراه قرار گرفته است که به طور مداوم در حالت مکالمه قرار داشته و در حال ارسال و دریافت همزمان داده می‌باشد. شکل ۴-۶ وضعیت این محیط را برای پیکربندی دوم نشان می‌دهد.

نتایج حاصل شده از ارزیابی در محیط نویزدار، مقاومت باس CAN و CAN FD را به خوبی نمایش می‌دهد. در تکرار مجدد آزمون‌های مرحله قبل در محیط نویزدار، تنها در نرخ داده ۱۲۵ Kbps تعدادی خطا ایجاد شده و برای انتقال با سایر نرخ‌ها روی باس CAN و همچنین نرخ ۱.۷ Mbps روی باس CAN FD، هیچ خطایی ایجاد نشده و انتقال مانند قبل انجام شد. البته که در مجموع انتقال ۱۰۰۰۰ بسته با نرخ ۱۲۵ Kbpsروی باس CAN، تنها ۹ بسته ۶۴ بیتی بر اثر نویزهای موجود در شبکه از دست رفته و به درستی به مقصد نرسیدند. این مساله حاکی از مقاومت بسیار مناسب لایه فیزیکی باس CAN در مقابل نویزهای محیطی است. این موضوع، معماری ارتباطی پیاده‌سازی شده را برای استفاده در محیط‌های صنعتی و همچین خشن[1]، تبدیل گزینه‌ای ایده‌آل می‌کند.



شکل ۴-۶: ارزیابی با نویز محیطی (پیکربندی دوم).

[1] Harsh Environment

فصل پنجم

نتیجه‌گیری و پیشنهادات

نتیجه‌گیری و پیشنهادات

اهمیت ارتباطات و مخابرات در دنیای کامپیوتر و الکترونیک بر هیچ کس پوشیده نیست. این ارتباطات ممکن است در سطوح بالا و میان خوشه‌های کامپیوتری[1] باشند، و یا در سطوح پایین و به منظور ارتباط اجزاء سیستم‌های نفهته طراحی شوند. تلاش‌های فراوانی در جهت ارائه استانداردهای مخابراتی در مبحث سیستم‌های نهفته شده است که هر یک تلاشی برای محقق کردند اهدافی نظیر امنیت، اطمینان‌پذیری، سرعت تبادل اطلاعات و یا دیگر عامل‌ها داشته‌اند. در میان این استانداردها، هدف برخی صرف ارتباط در لایه فیزیکی بوده در حالی که برخی دیگر پروتکل‌های سطح بالا هستند و در نهایت نیز تعدادی از این استانداردها یک پشته پروتکل کامل را شرح می‌دهند. این تلاش‌ها باعث ارائه باس‌ها پرکاربردی نظیر SPI، I2C و CAN شده‌است.

باس CAN نخستین بار در راستای ایجاد استاندارد در صنایع خودروسازی طراحی شده است ولی ویژگی‌ها و توانایی‌های آن، این باس را تبدیل به یکی از محبوب‌ترین باس‌های ارتباطی در صنایع هوافضا، اتوماسیون و بسیاری از سیستم‌های نهفته کرده‌است[۲]. ویرایش‌های متعددی از این باس در طول سالیان متوالی طراحی شده است که در این میان می‌توان به CAN FD اشاره نمود. هدف از توسعه استاندارد CAN FD، افزایش نرخ انتقال باس در کنار افزایش اطمینان‌پذیری البته بدون تغییر لایه فیزیکی CAN می‌باشد. با این وجود، باس CAN FD به علت نوین بودن کمتر مورد توجه کاربران و توسعه‌دهندگان قرار گرفته است. پیاده‌سازی یک ارتباط دو طرفه، با قرارگیری FPGA در یک سمت و میکروکنترلر در سمت دیگر به گونه‌ای که باس ارتباطی از نوع CAN FD باشد، هدف نهایی این پروژه می‌باشد. همچنین صحت ارتباط ایجاد شده بررسی شده و کیفیت ارتباط مورد ارزیابی قرار گرفته است.

[1] Cluster

طبق این توضیحات، راه‌اندازی این پروژه نیازمند دو پیاده‌سازی می‌باشد؛ یک پیاده‌سازی برای میکروکنترلر انجام شده و دیگری برای FPGA می‌باشد. علاوه بر این پیاده‌سازی، انتقال داده مبتنی برا پروتکل لایه بالا CANOpen نیز جزء اهداف این پروژه بوده است. متاسفانه به دلیل عدم دسترسی به کنترل‌کننده باس CAN FD، پیاده‌سازی این باس روی میکروکنترلر صورت نگرفته است و در طرح نهایی پیاده‌سازی شده، یک ارتباط میان میکروکنترلر و FPGA از طریق باس CAN استاندارد محقق شده است و در عوض، در قلب FPGA دو باس CAN FD قرار گرفته‌اند که اطلاعات ارسالی از کنترل‌کننده CAN موجود روی FPGA را دریافت کرده و به صورت حلقه بازگشتی به یکدیگر ارسال نمایند. پیکربندی دیگری نیز پیاده‌سازی شده است که تنها روی FPGA راه‌اندازی شده است و بدین صورت می‌باشد که دو کنترل‌کننده CAN FD روی FPGA قرار گرفته و از طریق پایه‌های برد FPGA با یکدیگر ارتباط برقرار کرده و به تبادل داده می‌پردازند.

ارزیابی نهایی طرح روی FPGA و البته میکروکنترلر، نشان‌دهنده بهینگی باس CAN و همچنین CAN FD از نظر منابع سخت‌افزاری مصرفی است. وجود چنین ویژگی‌ای در کنار خواصی مانند نرخ داده بالا، سادگی و هزینه پایین باس‌های CAN و CAN FD در کنار مقاومت بالا در مقابل نویز که در این پروژه مورد ارزیابی قرار گرفت، آن‌ها را تبدیل به یک گزینه ایده‌آل برای بسیاری از کاربردهای نهفته و به خصوص محیط‌های خشن می‌کند.

باس CAN FD و مفاهیم مرتبط با آن، می‌توانند محل بسیاری از تحقیقات آینده باشند. شاخص‌های متعددی که در طول سالیان به روی باس CAN مورد آزمایش قرار گرفته‌اند، مانند میزان توانایی و عملکرد آن در کاربردهای بلادرنگ [۴۸]، همچنان برای باس CAN FD در هاله‌ای از ابهام قرار دارند. همچنین علاوه بر ویژگی‌های مرتبط با سرعت و بهره‌وری CAN FD، عواملی نظیر امنیت نیز مورد توجه زیاد قرار نگرفته‌اند، عاملی که در سال‌های اخیر بسیار اهمیت یافته است [۴۹].

همچنین همانطور که باس CAN FD اولین ویرایش باس CAN نبوده است، آخرین ویرایش نیز نخواهد بود. هم اکنون نیز تلاش‌های متعددی برای توسعه ویرایش‌های جدیدتر باس CAN در حال

انجام است که در این میان می‌توان به دو مورد CAN XL و CAN FD Light اشاره نمود که استاندارد آن‌ها در دست توسعه بوده و در سال‌های آتی عرضه خواهند شد. CAN XL قابلیت ارسال محموله‌هایی تا ۲۰۴۸ بایت را خواهد داشت و ویژگی‌های متعددی از مدل مرجع OSI به آن افزوده خواهد شد. همچنین تلاش کارگروه توسعه CAN XL بر آن که لایه فیزیکی جدیدی را برای این باس توسعه دهند که از نرخ ارسال پیش‌فرض ۱۰ Mbps پشتیبانی کند. همچنین CAN FD Light ویرایشی از CAN FD است که معماری ارتباطی آن به صورت پیرو/راهبر بوده و در عوض نرخ ارسال داده آن محدود به مقدار ۱ Mbps می‌باشد [۵۰].

بررسی و تحلیل عمیق‌تر باس CAN FD در کنار استانداردهای آینده این مسیر یعنی CAN XL و CAN FD Light می‌تواند نتایج فوق‌العاده ارزشمندی را به همراه داشته باشد.

منابع و مراجع

[۱] C. S. Clifton, *What every engineer should know about data communications*. pp. 1-7, 1987. Available: https://www.taylorfrancis.com/books/9781003065586

[۲] CAN in Auromation, “History of CAN technology.” CAN in Auromation, 2019. Available: https://www.can-cia.org/can-knowledge/can/can-history/

[۳] A. Scholz, T.-H. Hsiao, J.-N. Juang, and C. Cherciu, “Open source implementation of ECSS CAN bus protocol for CubeSats,” *Adv. Space Res.*, vol. 62, no. 12, pp. 3438–3448, Dec. 2018, doi: 10.1016/j.asr.2017.10.015

[۴] International Organization for Standardization, “ISO 11898-1:2015,” *2015*. Available: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/36/63648.html

[۵] F. Hartwich, *CAN with flexible data-rate*, pp 3-11, 2012. Available: https://www.can-cia.org/fileadmin/resources/documents/proceedings/2012_hartwich.pdf

[۶] A. Mutter and Florian Hartwich, “Advantages of CAN FD Error detection mechanisms compared to Classical CAN,” *CAN Autom. ICC*, 2015. Available: https://s3.eu-central-1.amazonaws.com/cancia-de/documents/proceedings/icc_2015_mutter.pdf

[۷] S. Corrigan, “Introduction to the Controller Area Network (CAN),” pp, 2-16. Available: https://www.ti.com/lit/an/sloa101b/sloa101b.pdf?ts=1643829417705&ref_url=https%253A%252F%252Fwww.google.com%252F

[۸] CAN in Automation, “CAN in Automation (CiA): CANopen.” Available: https://www.can-cia.org/canopen/.

[۹] S. Corrigan and S. Corrigan, “Introduction to the controller area network (CAN),” *Tex. Instuments*, 2002.

[۱۰] Robert Bosch GmbH, “CAN Specification,” pp. 1, 4-18, 23-30, 1991. Available: http://esd.cs.ucr.edu/webres/can20.pdf

[۱۱] USB Implementers Forum, “USB 2.0 Specification.” pp, 139-140. Available: https://www.usb.org/document-library/usb-20-specification

[۱۲] P. Richards, “A CAN Physical Layer Discussion,” pp. 1-10, 2002. Available: https://ww1.microchip.com/downloads/en/appnotes/00228a.pdf

[۱۳] Kvaser, “CAN Physical Layers,” Dec. 2018. Available: https://www.kvaser.com/lesson/can-physical-layers/

[۱۴] CAN in Automation, “CAN Data Link Layer.” pp. 1-26. Available: http://www.diakom.com.ru/el/communication/can/candll.pdf

[۱۵] NXP, “CAN Bit Timing Requirements,” pp 1-3, 2004. Available: https://www.renesas.com/us/en/document/whp/isolated-can-bus-small-satellites?language=en

[۱۶] Renesas, “Isolated CAN Bus for Small Satellite Applications,” pp 1, Feb. 2019. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[۱۷] NXP, “I2C-bus specification and user manual,” pp. 2-6, Oct. 2021. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[۱۸] Texas Instruments, “Universal Synchronous Asynchronous Receive/Transmit USART,” pp. 12_1-12_24. Available: https://www.ti.com/sc/docs/products/micro/msp430/userguid/ag_12.pdf

[۱۹] Texas Instruments, “Serial Peripheral Interface User Guide,” pp. 1_2-1_3, Mar. 2012. Available: https://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf

[۲۰] FRANZIS, “MOST: THE AUTOMOTIVE MULTIMEDIA NETWORK,” pp. 22-49, 2008. Available: http://www2.ciando.com/img/books/extract/3645250611_lp.pdf

[۲۱] AUTOSAR, “Specification of LIN Interface.” pp. 9-14. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_LINInterface.pdf

[۲۲] FlexRay Consortium, “FlexRay Communications System Protocol Specification,” pp. 14-31, Oct. 2010. Available: https://svn.ipd.kit.edu/nlrp/public/FlexRay/FlexRay%E2%84%A2%20Protocol%20Specification%20Version%203.0.1.pdf

[۲۳] RF Wireless World, “LIN, CAN, FlexRay, MOST-Difference.” Available: https://www.rfwireless-world.com/Terminology/LIN-vs-CAN-vs-FlexRay-vs-MOST.html

[۲۴] L.-B. Fredriksson, “TTCAN explained,” pp. 1-4, 2008 Available: https://www.kvaser.com/wp-content/uploads/2014/08/ttcan-explained.pdf.

[۲۵] Microchip, “Controller Area Network with Flexible Data-rate (CAN FD),” pp. 56_2-56_8,2018. Available: https://www.can-cia.org/fileadmin/resources/documents/proceedings/2012_hartwich.pdf

[۲۶] G. Marcon Zago and E. Pignaton de Freitas, “A Quantitative Performance Study on CAN and CAN FD Vehicular Networks,” *IEEE Trans. Ind. Electron.*, vol. 65, no. 5, pp. 4413–4422, May 2018, doi: 10.1109/TIE.2017.2762638.

[۲۷] CAN in Automation, “CAN in Automation (CiA): CAN FD - The basic idea.” Available: https://www.can-cia.org/can-knowledge/can/can-fd/

[۲۸] Robert Bosch GmbH, “CAN with Flexible Data-Rate,” pp. 3-34, Apr. 2012. Available: https://can-newsletter.org/assets/files/ttmedia/raw/e5740b7b5781b8960f55efcc2b93edf8.pdf

[۲۹] Esatinc, “OBD II Specifications and Connections.” Available: http://www.esatinc.ca/News_Letters/OBD_II_Specifications_and_Connections.pdf

[۳۰] MicroControl, “J1939 Protocol Stack,” pp. 9-11. Available: http://www.microcontrol.net/wp-content/uploads/2021/10/hb_j1939_v3r00_en.pdf

[۳۱] CAN in Automation, “CAN in Automation (CiA): CANopen.” CAN in Automation: https://www.can-cia.org/canopen/

[۳۲] CAN in Automation (CiA), “CANOpen Report,” 2016. Available: http://affon.narod.ru/CAN/CANopen.pdf

[۳۳] CAN in Automation (CiA), “CANopen application layer and communication profile.” CAN in Automation (CiA), Feb. 2011. Available: https://www.can-cia.org/index.php?eID=tx_nawsecuredl&u=25087&g=11&t=1630974428&hash=05e0474253578c1227a3c60c3cfeb8429588578a&file=fileadmin/resources/documents/groups/301v04020007_cor.pdf

[۳۴] Siemens, “On-Board Communication via CAN without Transceiver.” Available: https://www.mikrocontroller.net/attachment/28831/siemens_AP2921.pdf

[۳۵] Farnell, “Arduino Due.” Available: http://www.farnell.com/datasheets/1682211.pdf

[۳۶] “Arduino Due,” *Arduino Online Shop*. http://store-usa.arduino.cc/products/arduino-due.

[۳۷] Xilinx, “Spartan-3 FPGA Family: Introduction and Ordering Information,” pp. 1-16, Jun. 2013. Available: https://www.xilinx.com/support/documentation/data_sheets/ds099.pdf

[۳۸] رهپویان علم و صنعت آوا، «برد توسعه Spartan3 AVA3S400»، قابل دسترسی از طریق پیوند: http://revsa.ir/products/xilinx-boards-and-kits/spartan/spartan3-ava3s400

[۳۹] NXP, “TJA1050 High Speed CAN Transceiver Datasheet,” pp. 1-18, Oct. 2003. Available: https://www.nxp.com/docs/en/data-sheet/TJA1050.pdf

[۴۰] M. Fezari and A. Al Dahoud, “Integrated Development Environment ‘IDE’ For Arduino,” Oct. 2018. Available: https://www.researchgate.net/publication/328615543_Integrated_Development_Environment_IDE_For_Arduino

[۴۱] Xilinx, *ISE In-Depth Tutorial*, pp. 7-64 Oct. 2011. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx13_3/ise_tutorial_ug695.pdf.

[۴۲] Xilinx, “Vivado Design Suite User Guide,” pp. 5-9, Oct. 2017. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2021_1/ug904-vivado-implementation.pdf

[۴۳] C. Kidder, *due_can*. 2021. Available: https://github.com/collin80/due_can

[۴۴] X.Wang, *FPGA-liteCAN*. 2022. Available: https://github.com/WangXuan95/liteCAN

[۴۵] Czech Technical University, *CTU CAN FD IP Core*. Available: https://gitlab.fel.cvut.cz/canbus/ctucanfd_ip_core

[۴۶] Czech Technical University, “CTU CAN FD IP Core Datasheet,” pp. 2-69, Dec. 2021. Available: https://canbus.pages.fel.cvut.cz/ctucanfd_ip_core/doc/Datasheet.pdf

[۴۷] A. Mutter, R. Bosch, and F. Hartwich, “Advantages of CAN FD Error detection mechanisms compared to Classical CAN,” 2015. Available: https://www.semanticscholar.org/paper/Advantages-of-CAN-FD-Error-detection-mechanisms-to-Mutter-Bosch/745ad99d9619f3e682edd97d25585a34a203b803

[۴۸] J. Xia, C. Zhang, R. Bai, and L. Xue, “Real-time and reliability analysis of time-triggered CAN-bus,” *Chin. J. Aeronaut.*, vol. 26, no. 1, pp. 171–178, Feb. 2013, doi: 10.1016/j.cja.2012.12.017.

[۴۹] A. Taylor, N. Japkowicz, and S. Leblanc, “Frequency-based anomaly detection for the automotive CAN bus,” in *2015 World Congress on Industrial Control Systems Security (WCICSS)*, Dec. 2015, pp. 45–49. doi: 10.1109/WCICSS.2015.7420322.

[۵۰] CAN in Automation, “CAN XL and CAN FD light.” https://www.can-cia.org/news/cia-in-action/view/can-xl-and-can-fd-light/

پیوست‌ها

در ادامه، کد پیاده‌سازی این پروژه برای میکروکنترلر و FPGA برای هر دو پیکربندی طراحی شده ارائه شده است.

جدول پ-۱: شرح کد پیاده‌سازی پیکربندی اول در FPGA.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////
// Organization:        Amirkabir University of Technology
// Author: Mohamad      Chamanmotlagh
//////////////////////////////////////////////////////////////////////////////
```

این بخش از کد، به توصیف ماژول اصلی و درگاه‌های متصل به آن می‌پردازد.

```
module top (
      clk,
      CAN_RX,
      CAN_TX,
      sending,
      received,
      data
);
//    ====================IO Signals============================//
```

این بخش از کد، نوع هر یک از درگاه‌های ورودی و خروجی را مشخص می‌نماید. این درگاه‌ها شامل کلاک سیستمی، سیگنال‌های RX و TX باس CAN، سیگنال نشانگر ارسال و دریافت و سیگنال نمایشگر داده خروجی می‌باشد.

```
      input       wire  clk;              // System Clock
      input       wire  CAN_RX;           // CAN Receive signla
      output      wire  CAN_TX;           // CAN Transmit signal
      output      wire  sending;          // Send indicator
      output      wire  received;         // Receive indicator
      output      wire  [7:0]data;        // Received data visualization

//    ============================================================//
```

این بخش، تعدادی از سیگنال‌های میانی را توصیف می‌کند& سیگنال‌های نظیر بازنشانی داخلی، داده ارسالی و دریافتی از ورودی-خروجی، و همچنین مقدار سیگنال کلاک Prescale شده.

```
      wire  rstn;                   // Reset line
      reg   [31:0] can_tx_cnt;      // Sample sent data
      reg   can_tx_valid;           // Sent data is valid or not
      reg   [31:0] can_tx_data;     // Sent data
      wire  can_rx_valid;           // Received data is valid or not
      wire  [7:0] can_rx_data;      // Recived data
      wire  clock_out;              // output clock after dividing the
                                    // input clock by divisor
```

دو مقدار دهی سیگنال انجام شده زیر، مقادیر نشانگر ارسال و دریافت را مشخص می‌کنند.

```
	assign sending = can_tx_valid;
	assign received = can_rx_valid;

//	================Generate Reset Signal====================//
```

این بخش، به تولید سیگنال بازنشانی باس CAN به صورت متناوب می‌پردازد. وجود این بخش ضروری نیست و می‌توان بازنشانی را به یکی از پایه‌های ورودی FPGA متصل نمود.

```
	reset_gen #(
		.DEFAULT(1),
		.tP(25000),
		.tR(25000)
	) reset_gen_i(
		.rstn(1'b1),
		.clk(clk),
		.on(1'b0),
		.off(1'b0),
		.o_rst(rstn)
	);

//	================Apply Prescaler for CAN====================//
```

این بخش، عامل Prescaler را بر روی کلاک سیستمی اعمال کرده و مقدار خروجی را درون سیگنال Clock_out قرار می‌دهد. در حالت فعلی، مقدار عامل Prescaler برابر ۴۰ می‌باشد.

```
	clock_divider
	#(28'd40)
	clk_div(
		clk,
		clock_out
		);

//	=================Send Preiodic Data=======================//
```

وظیفه‌ی این بخش، تولید داده افزایشی به منظور ارسال روی خطوط باس و همچنین ارسال متناوب داده تولید شده می‌باشد.

```
	always @(posedge clock_out or negedge rstn)
		if (~rstn) begin
			can_tx_cnt <= 0;
			can_tx_valid <= 1'b0;
			can_tx_data <= 0;
		end
		else if (can_tx_cnt < 1000000) begin
			can_tx_cnt <= can_tx_cnt + 1;
			can_tx_valid <= 1'b0;
		end
		else begin
			can_tx_cnt <= 0;
			can_tx_valid <= 1'b1;
			can_tx_data <= can_tx_data + 1;
		end

//	================Instantiate CAN module====================//
```

در این بلوک از کد، یک نمونه از روی هسته liteCAN ساخته می‌شود. در راستای همین امر، نخست متغیرهای پیکربندی (نظیر تعداد کوانتوم زمانی بخش‌های مختلف) مشخص شده و سپس سیگنال‌های کنترلی و همچنین ورودی-خروجی این باس مقداردهی

می‌شوند.

```
	can_top #(
		.LOCAL_ID(11'h456),
		.default_c_PTS(16'd2),
		.default_c_PBS1(16'd1),
		.default_c_PBS2(16'd2)
	) can_controller(
		.rstn(rstn),
		.clk(clock_out),
		.can_rx(CAN_RX),
		.can_tx(CAN_TX),
		.tx_valid(can_tx_valid),
		.tx_ready(),
		.tx_data(can_tx_data),
		.rx_valid(can_rx_valid),
		.rx_last(),
		.rx_data(can_rx_data),
		.rx_id(),
		.rx_ide()
	);

//	===============Instantiate CAN-FD modules==================//
```

در این بخش، دو نمونه از کنترل‌کننده CAN FD ساخته شده و ارتباط میان آن‌ها برقرار می‌شود. در گام نخست، سیگنال‌های میانی دو کنترل‌کننده ساخته می‌شود؛ سپس نمونه هسته‌ها ساخته می‌شود.

```
	reg		CAN_FD_TX;		// CAN-FD module Transmit signal
	reg		CAN_FD_RX;		// CAN-FD module  Receive signal
	wire		[31:0] extended_date;	// Extended data for CAN-FD
	wire		[31:0] read_data;	// CAN-FD module Returned data
	assign		extended_data = {24'b0, can_rx_data};
```

در این مرحله، دو هسته CTU CAN FD نمونه‌سازی می‌شوند. این هسته از طریق سیگنال‌های TX و RX باس CAN FD با یکدیگر ارتباط دارند. ارتباط خارجی آن‌ها با سایر ماژول‌های کد نیز از طریق واسط AMBA APB طراحی شده درون هسته‌ها صورت می‌گیرد.

```
	can_fd_top_apb FDCAN_T(
		.CAN_tx(CAN_FD_TX),
		.CAN_rx(CAN_FD_RX),
		.aclk(clock_out),
		.arstn(rstn),
		.s_apb_pwdata(extended_date),
		.s_apb_pwrite(can_rx_valid),
		.s_apb_psel(can_rx_valid),
		.scan_enable(),
		.res_n_out(),
		.irq(),
		.timestamp(),
		.s_apb_paddr(),
		.s_apb_penable(1'b1),
		.s_apb_pprot(),
		.s_apb_pready(),
		.s_apb_pslverr(),
		.s_apb_pstrb(),
		.s_apb_prdata()
```

```
            );

            can_fd_top_apb FDCAN_R(
            .CAN_tx(CAN_FD_RX),
            .CAN_rx(CAN_FD_TX),
            .aclk(clock_out),
            .arstn(rstn),
            .s_apb_pwdata(),
            .s_apb_pwrite(),
            .s_apb_psel(can_rx_valid),
            .scan_enable(),
            .res_n_out(),
            .irq(),
            .timestamp(),
            .s_apb_paddr(),
            .s_apb_penable(1'b1),
            .s_apb_pprot(),
            .s_apb_pready(),
            .s_apb_pslverr(),
            .s_apb_pstrb(),
            .s_apb_prdata(read_data)
            );

//    =================Write Data to outputs==================//
```

در گام آخر، داده دریافت شده از کنترل‌کننده CAN FD دوم روی سیگنال خروجی نمایش‌دهنده داده FPGA نوشته می‌شود.

```
      assign data = read_data[7:0];
endmodule
```

انتهای ماژول بالاترین سطح.

```
//    =================Clock_divider module==================//
```

این ماژول، وظیفه تغییر طول کلاک و در حقیقت اعمال عامل Prescaler را بر عهده دارد.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.ALL;
```

در ابتدا، نام ماژول و درگاه‌های آن توصیف می‌شوند.

```
ENTITY Clock_Divider IS
        GENERIC (Prescaler : INTEGER);
        PORT (
                clk,
                clock_out : OUT std_logic
        );
END Clock_Divider;


ARCHITECTURE bhv OF Clock_Divider IS

        SIGNAL count : INTEGER := 1;
        SIGNAL tmp : std_logic := '0';
```

عملکرد اصلی این ماژول، از طریق یک حلقه حساس به کلاک اصلی انجام می‌شود.

```
BEGIN
        PROCESS (clk)
        IF (clk'EVENT AND clk = '1') THEN
                count <= count + 1;
                IF (count = Prescaler) THEN
                        tmp <= NOT tmp;
                        count <= 1;
                END IF;
        END IF;
        clock_out <= tmp;
END PROCESS;
```

جدول پ-۲: شرح کد پیاده‌سازی پیکربندی دوم در FPGA.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Organization:        Amirkabir University of Technology
// Author: Mohamad      Chamanmotlagh
//////////////////////////////////////////////////////////////////////////////////
```

این بخش از کد، به توصیف ماژول اصلی و درگاه‌های متصل به آن می‌پردازد.

```
module top_config_2(
    input  RX_1,
    output TX_1,
    input  RX_2,
    output TX_2,
    input  CLK,
    output sending,
    output receiving,
    output [7:0] data
    );

//    ================Apply Prescaler for CAN====================//
```

در این بخش، عامل Prescaler روی کلاک سیستمی اعمال شده و سیگنال خروجی درون clock_out قرار می‌گیرد.

```
      wire  clock_out;                    // output clock after dividing
the input clock by divisor
      clock_divider
      #(28'd1)                                  // Prescale factor
      clk_div(
            clk,
            clock_out
            );



//    ==================Send Preiodic Data=======================//
```

در این بخش، داده‌ی ارسالی روی باس CAN FD تولید شده و به صورت متناوب تغییر کرده و درخواست ارسال صادر می‌شود.

```
      reg         send;
      reg         [31:0] counter;
```

```
	reg		[31:0] sent_data = 32'b0;
	always @(posedge clock_out)
		if (counter < 1000000) begin
			counter <= counter + 1;
			send <= 1'b0;
		end
		else begin
			counter <= 0;
			send <= 1'b1;
			sent_data <= sent_data + 1;
		end
	assign	sending = send;


//	===============Instantiate CAN-FD modules==================//
```

در این بخش، دو عدد ماژول CAN FD نمونه‌سازی شده و درگاه‌های آن از طریق سیگنال‌های مربوطه مقداردهی می‌شوند. همچنین داده تولید شده در بلوک قبل، از طریق ماژول CAN FD اول ارسال شده و در ماژول CAN FD دوم دریافت می‌شود.

```
	wire		[31:0] read_data;		// CAN-FD module 1 Returned data

	can_fd_top_apb FDCAN_1(
		.CAN_tx(TX_1),
		.CAN_rx(RX_1),
		.aclk(clock_out),
		.arstn(1'b1),
		.s_apb_pwdata(sent_data),
		.s_apb_pwrite(send),
		.s_apb_psel(1'b1),
		.scan_enable(),
		.res_n_out(),
		.irq(),
		.timestamp(),
		.s_apb_paddr(),
		.s_apb_penable(1'b1),
		.s_apb_pprot(),
		.s_apb_pready(),
		.s_apb_pslverr(),
		.s_apb_pstrb(),
		.s_apb_prdata()
		);

		can_fd_top_apb FDCAN_2(
		.CAN_tx(TX_2),
		.CAN_rx(RX_2),
		.aclk(clock_out),
		.arstn(1'b1),
		.s_apb_pwdata(),
		.s_apb_pwrite(),
		.s_apb_psel(1'b1),
		.scan_enable(),
		.res_n_out(),
		.irq(),
		.timestamp(),
		.s_apb_paddr(),
```

```
            .s_apb_penable(1'b1),
            .s_apb_pprot(),
            .s_apb_pready(receiving),
            .s_apb_pslverr(),
            .s_apb_pstrb(),
            .s_apb_prdata(read_data)
            );

      //    ==============Write signals to outputs================//
```

در این بخش، داده دریافتی روی درگاه‌های خروجی ماژول نوشته می‌شود تا از طریق ادوات موجود روی برد FPGA قابل مشاهده باشد.

```
      assign data = read_data[7:0];

endmodule
```

جدول پ-۳: فایل ایجاد محدودیت برای مسیریابی ورودی و خروجی‌های FPGA.

```
# Geberal ports
```

این بخش از فایل محدودیت، در هر دو پیکربندی یکسان است.

```
NET "clk" LOC = P184;
NET "sending" LOC = P42;
NET "received" LOC = P44;

NET "data[7]" LOC = P61;
NET "data[6]" LOC = P62;
NET "data[5]" LOC = P63;
NET "data[4]" LOC = P64;
NET "data[3]" LOC = P65;
NET "data[2]" LOC = P67;
NET "data[1]" LOC = P68;
NET "data[0]" LOC = P71;

# Configuration 1 ports
```

این بخش از فایل محدودیت، مخصوص پیکربندی اول می‌باشد.

```
NET "CAN_TX" LOC = P189;
NET "CAN_RX" LOC = P190;

# Configuration 2 ports
```

این بخش از فایل محدودیت، مخصوص پیکربندی دوم می‌باشد.

```
NET "TX_1" LOC = P189;
NET "RX_1" LOC = P190;
NET "TX_2" LOC = P194;
NET "RX_2" LOC = P191;
```

جدول پ-۴: شرح کد پیاده‌سازی پیکربندی اول در میکروکنترلر.

```
/**
 * Author: Mohamad Chamanmotlagh
 * Organization: Amirkabir University of Technology
 */
```

این بخش از کد وظیفه ارسال و دریافت پیام باس CAN را بر عهده دارد.

```
// Required libraries
```

```
#include "variant.h"
#include <due_can.h>
```

در این بخش، متغیرهای جهانی[1] مورد نیاز تعریف و مقدار دهی می‌شوند.

```
//random ID
#define TEST1_CAN_TRANSFER_ID    0x456

// CAN frame max data length
#define MAX_CAN_FRAME_DATA_LEN   8
// Number of test frames
#define TEST_MAX_COUNT 10000
// CAN's bit rate
#define BIT_RATE 125000
```

این متغیرها مسئول شمارش تعداد پیام‌های ارسالی و دریافتی هستند.

```
Uint32_t sentFrames, receivedFrames;

//Leave this defined if you use the native port or comment it out if you
use the programming port
//#define Serial SerialUSB

// Used frames
CAN_FRAME frame, incoming;

void setup() {

// start serial port at 115200 bps:
  Serial.begin(115200);

// Wait unitl serial monitor is opened
  while(!Serial);
```

در این بخش، کنترل‌کننده CAN مقدار دهی اولیه شده و آماده ارسال و دریافت با نرخ‌بیتی متغیر BIT_RATE می‌شود.

```
  // Verify CAN0 initialization with BIT_RATE:
  if (Can0.begin(BIT_RATE)
    Serial.println("CAN0: با موفقیت راه‌اندازی شد.");
  else
    Serial.println("CAN initialization (sync) ERROR");
```

در این بخش، یک قاب پیام CAN ساخته می‌شود.

```
  //Initializing the sending frame.
  frame.id = TEST1_CAN_TRANSFER_ID;
  frame.length = MAX_CAN_FRAME_DATA_LEN;
  frame.data.value = 0;
  frame.extended = 1;
```

این قطعه کد، کنترل‌کننده CAN را به گونه‌ای مقداردهی می‌کند که آماده دریافت پیام‌هایی با شناساگر TEST1_CAN_TRANSFER_ID باشد.

```
  // Filter incoming IDs
```

[1] Global variable

```
  Can0.watchFor(TEST1_CAN_TRANSFER_ID);
```

تابع ارسال و دریافت پیام فراخوانی می‌شود.

```
  start_can();
}

/*     Sending and receiving pure CAN messages       */
```

این تابع، وظیفه ارسال قاب پیام ساخته شده و همچنین دریافت پیام‌های دارای شناساگری خاص را دارد.

```
static void start_can(void)
{
  uint32_t counter = 0;

  while (1==1) {
```

در یک حلقه نامتنهای، به صورت مداوم قاب ساخته شده روی باس ارسال می‌شود تا ترافیک مناسب را ایجاد کند. همچنین مقدار محموله پیام در هر ارسال افزایش می‌یابد.

```
    // Send data on bus lines
    if(sentFrames < TEST_MAX_COUNT){
      Can0.sendFrame(frame);
      sentFrames++;
      frame.data.value += 2;
    }

    Serial.print((int)frame.data.value);
    Serial.print("\t");
```

به صورت مداوم، بافر ورودی کنترل‌کننده CAN بررسی شده و در صورتی که پیامی روی آن موجود باشد، آن مقدار خوانده شده و قابل مشاهده می‌باشد.

```
    // Read incoming messages if any is available
    if (Can0.available() > 0) {
      Can0.read(incoming);
      Serial.print((int)incoming.data.value);
      Serial.print("\t");
      receivedFrames++;
      counter++;
    } else {
      Serial.print("-");
      Serial.print("\t");
    }
     Serial.print(sentFrames);
     Serial.print("\t");

     Serial.print(receivedFrames);
     Serial.print("\t");
```

در این بخش، ثبات خطای کنترل‌کننده CAN برای هر دو نوع پیام ارسالی و دریافتی خوانده شده و مقدار آن گزارش می‌شود.

```
    // Couting number of errors
    int total_error_1 = Can0.get_rx_error_cnt() +
Can0.get_tx_error_cnt();
```

```
    Serial.println(total_error_1);

    // Finish if all of messages hasbeen recieved
    if(receivedFrames == sentFrames)
      break;
  }
}

void loop()
{
}


/*     Initializing CAN controller       */
```

در این قطعه کد، کنترل‌کننده CAN مقداردهی اولیه می‌شود. این مقداردهی به معنای مشخص نمودن متغیرهای زمانی CAN می‌باشد.

```
uint32_t CANRaw::set_baudrate(uint32_t ul_baudrate)
{
```

در این بخش، متغیرها مقداردهی اولیه می‌شوند.

```
      uint8_t uc_tq;
      uint8_t uc_prescale;
      uint32_t ul_mod;
      uint32_t ul_cur_mod;
      can_bit_timing_t *p_bit_time;
      static uint32_t ul_mck = SystemCoreClock;
```

در صورتی که مقدار prescaler بیشتر از حداکثر قابل استفاده باشد، عملیات با خطا مواجه شده و به اتمام می‌رسد.

```
      if (((ul_mck + (ul_baudrate * CAN_MAX_TQ_NUM - 1)) /
            (ul_baudrate * CAN_MAX_TQ_NUM)) > CAN_BAUDRATE_MAX_DIV) {
            return 0;
      }
```

اگر اندازه کلاک سیستمی کمتر از اندازه قابل قبول باشد، تابع با خطا مواجه می‌شود.

```
      if (ul_mck  < ul_baudrate * CAN_MIN_TQ_NUM) {
            return 0;
      }
```

مقدار کوانتوم زمانی تقریبی با توجه به نرخ بیتی انتخاب می‌شود.

```
      uc_tq = CAN_MIN_TQ_NUM;
      ul_mod = 0xffffffff;
      for (uint8_t i = CAN_MIN_TQ_NUM; i <= CAN_MAX_TQ_NUM; i++) {
            if ((ul_mck / (ul_baudrate * i)) <= CAN_BAUDRATE_MAX_DIV) {
                  ul_cur_mod = ul_mck % (ul_baudrate * i);
                  if (ul_cur_mod < ul_mod){
                        ul_mod = ul_cur_mod;
                        uc_tq = i;
                        if (!ul_mod) {
                              break;
                        }
                  }
```

```
            }
        }
        uint32_t ul_status = m_pCan->CAN_SR;
```

مقدار عامل Prescaler انتخاب می‌شود.

```
        uc_prescale = ul_mck / (ul_baudrate * uc_tq);
```

بر اساس مقادیر محاسبه شده تاکنون (کوانتوم زمانی و عامل Prescaler)، سایر مقادیر زمانی از داخل لیستی که از پیش نوشته شده است انتخاب می‌شوند.

```
        p_bit_time = (can_bit_timing_t *)&can_bit_time[uc_tq -
CAN_MIN_TQ_NUM];
```

در این بخش، کنترل‌کننده باس CAN غیرفعال شده و سپس مقادیر زمانی روی ثبات‌های آن نوشته می‌شوند. در پایان نیز کنترل‌کننده مجددا آغاز به کار کرده و عملکرد تابع به اتمام می‌رسد.

```
    can_disable(m_pCan);
    uint32_t oldCANMR = m_pCan->CAN_MR;
    m_pCan->CAN_MR &= ~CAN_MR_CANEN;

    m_pCan->CAN_BR = CAN_BR_PHASE2(p_bit_time->uc_phase2 - 1) |
                             CAN_BR_PHASE1(p_bit_time->uc_phase1 - 1) |
                             CAN_BR_PROPAG(p_bit_time->uc_prog - 1) |
                             CAN_BR_SJW(p_bit_time->uc_sjw - 1) |
                             CAN_BR_BRP(uc_prescale - 1);

    m_pCan->CAN_MR = oldCANMR;
    ul_status = m_pCan->CAN_SR;
    (void)ul_status;
    numBusErrors = 0;
    numRxFrames = 0;
    busSpeed = ul_baudrate;

    return 1;
}

/*     CANOpen Functions       */
```

در این قطعه کد، کنترل‌کننده، عملکردهای مربوط به ارسال و دریافت پیام‌های مطابق پروتکل CANOpen انجام شده است.

ساختار زیر، ویژگی‌های یک گره شبکه CANOpen را مشخص می‌کند.

```
typedef struct {
    uint8_t         node_id;        // Node-Id
    uint32_t        Baudrate;      // Baudrate
    struct CO_OBJ_T *Dict;         // object dictionary
    uint16_t        dict_len;       // object dictionary (max) length
} CO_NODE;
```

ساختار زیر، یک قاب پیام CANOpen را مشخص می‌نماید.

```
typedef struct {
    unit8_t node_id;         // node_id
    unit8_t function_code;   // function of the message
    unit8_t RTR;             // request for data
    unit8_t length;          // length of the data
    unit8_t data;            // payload data
} CO_FRAME;
```

عملکرد این تابع بدین صورت می‌باشد که یک پیام متنی در قالب یک رشته را دریافت نموده محتوای آن را به صورت پیام‌های

CANOpen درآورده و بازگردانی می‌کند.

```
CO_FRAME* send_co_frame(char* msg){
```

در ابتدا، اندازه پیام و تعداد قطعات مورد نیاز محاسبه می‌شود.

```
    size_t msg_count = 0;
    while (msg[msg_count] != '\0') msg_count++;
    CO_FRAME *messages = malloc(sizeof(CO_FRAME) * msg_count);
    int i = 0;
```

در این قطعه از کد، پیام مورد نظر به قطعات کافی تقسیم شده و هر قطعه درون یک قاب پیام CANOpen قرار می‌گیرد.

```
    // Segmenting the whole message
    while (I <= msg_count) {
        CO_FRAME frame;
         // function-code for sending data messages is 0x3
        frame.function_code = 0x3;
         // node-id is assumed 1 for the microcontroller
        frame.node_id = 1;
        frame.RTR = 0;
        frame.length = msg_count - i;
        frame.data = (int)msg;
        messages[i] = frame;
        i++;
    }
```

در پایان نیز اشاره‌گری به آرایه پیام‌های CANOpen تولیدی بازگردانده می‌شود.

```
    return messages;
}
```

در این بلوک از کد، پیام‌های دریافتی پیاپی مطابق پروتکل CANOpen دریافت شده و محموله نهایی تولید و بازگردانی می‌شود.

```
char * receive_co_frame(CO_FRAME* messages){
```

در ابتدای دریافت پیام‌های CANOpen، تعداد پیام‌ها محاسبه می‌شود.

```
    CO_FRAME frame_1 = messages[0];
    if(frame_1.function_code != 0x3)
        return NULL;
    size_t len = messages[0].length;

    char * full_msg = malloc(sizeof(char) * len);
```

با در اختیار داشتن تعداد پیام‌ها، در این بخش پیام‌های متوالی خوانده شده و از طریق محموله هر یک، محموله کلی تولید می‌شود.

```
    for (int i = 0; i < len; i++) {
        char msg_char = messages[i].data +'0';
        full_msg[i] = msg_char;
    }
```

در پایان نیز محموله کامل پیام‌ها بازگردانی می‌شود.

```
    return full_msg;
}
```

Name: CANSendSegment

File: CANALP.C

Inputs: CanID, CanData

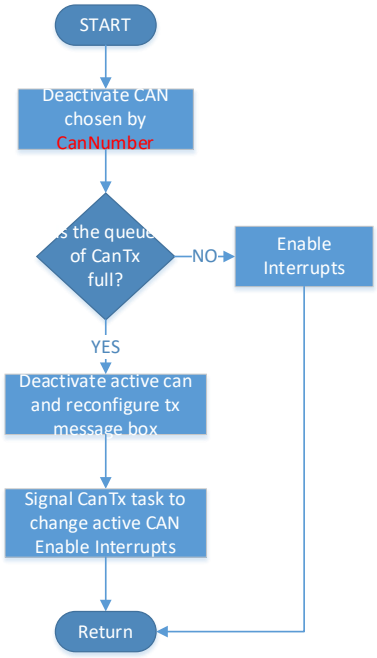
Outputs: -

Global Variables: ActiveCAN

Peripherals: -

Explanation

این تابع سگمنت CANی که از وروی می‌گیرد را برای ارسال به CAnTxTask می‌فرستد. اگر صف Task پر بود، متوجه اشکال باس CAN می‌شود، فلذا CanTxTAsk سیگنال عوض کردن CAN فعال را مخابره می‌کند و ارسال سگمنت CAN را صورت می‌دهد.



Name: ProcessPacket

File: CANALP.C

Inputs: DesTask, CanPacket

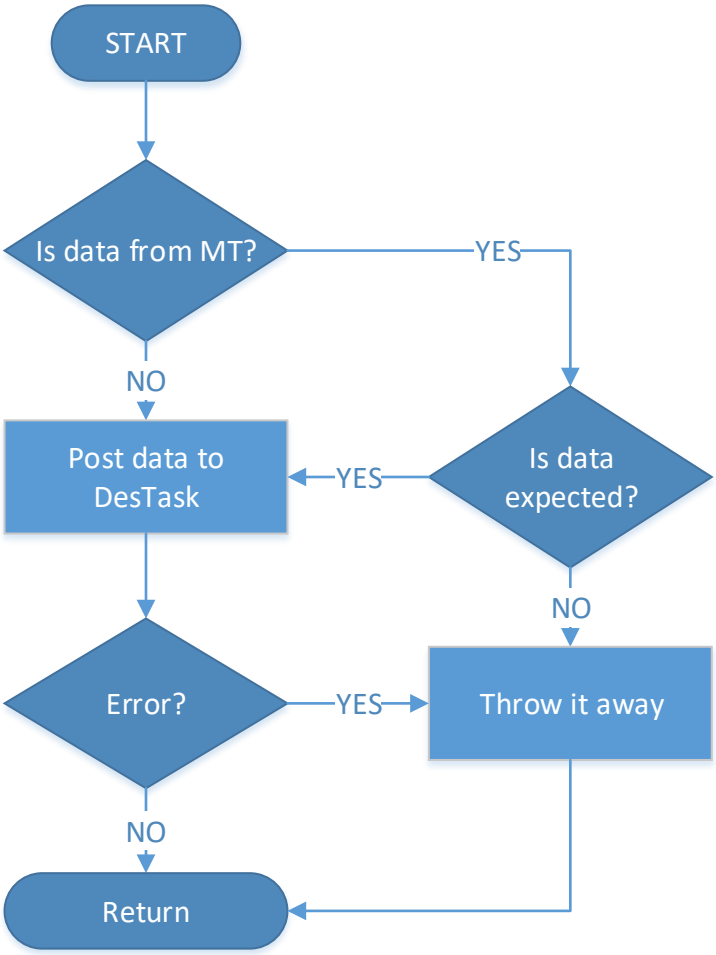
Outputs: -

Global Variables: -

Peripherals: -

Explanation:

این تابع بسته دریافت شده از CAN را به Task مربوطه می‌فرستد.



Name: ProcessCANError

File: CANALP.C

Inputs: DesTask, CANOpenRegs

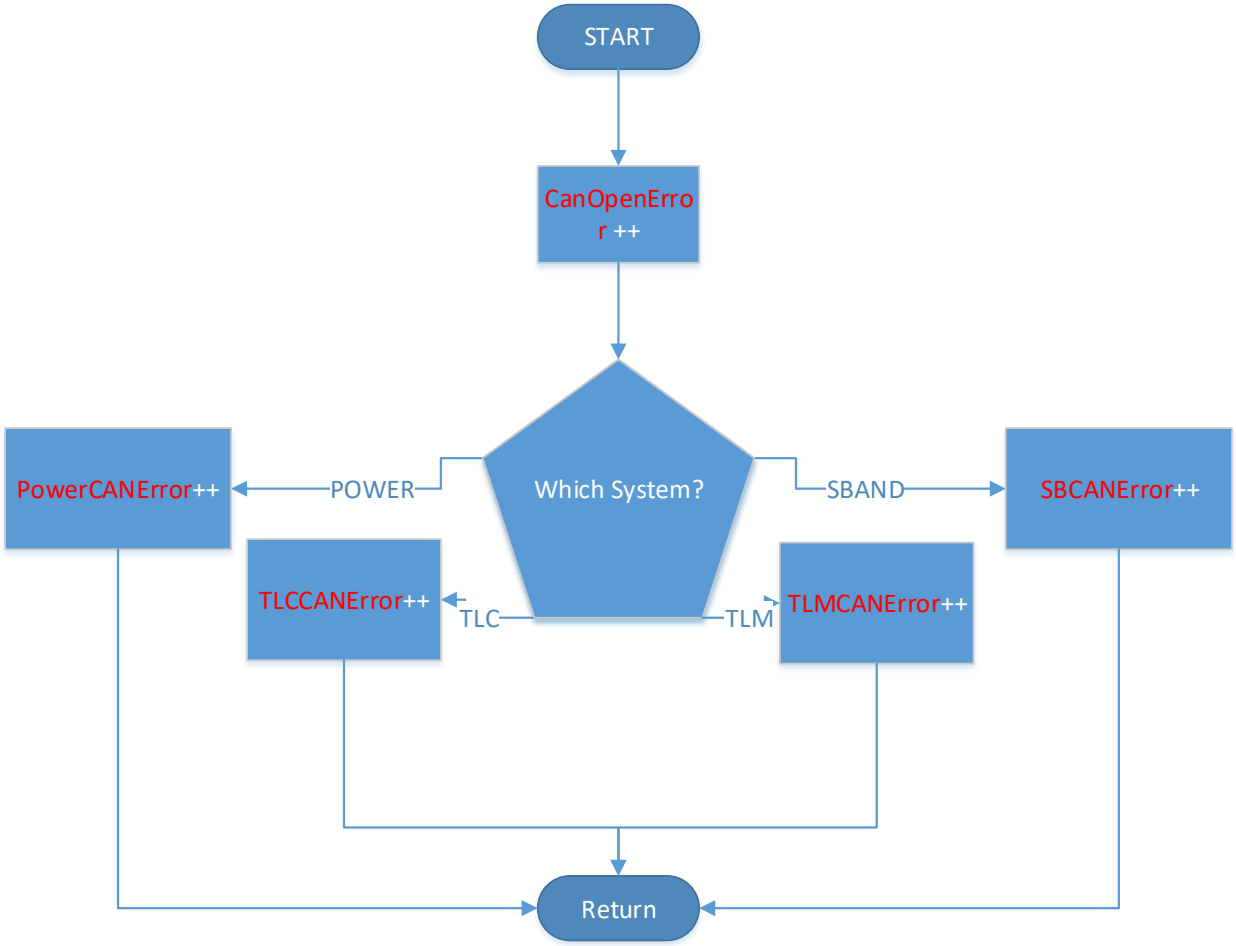
Outputs: -

Global Variables: CANOpenError, ProcessCAnError, TLMCanError, TLC CanError, SBCanError

Peripherals: -

Explanation:

این تابع در زمان به مشکل خوردن پروتکل CANOpen فراخوانی می‌شود و شمارنده‌های Error را زیاد می‌کند.



Abstract

Communication between embedded systems' components is a significant issue in engineering sciences. Numerous efforts have been made to offer and standardize these communications. Some of these proposed standards define merely the physical connection, while others define high-level protocols as well. These standards include, but are not limited to, SPI, I^2C, and CAN busses. The CAN bus has established itself as a well-known standard bus in the automotive, automation, and aerospace industries, among others, and different variations of this bus have been produced to satisfy various requirements across industries. CAN FD is a customized version of the CAN bus that accelerates transmission without altering the physical layer of the CAN bus. Due to the novelty of CAN FD, little effort has been expended on its implementation and use. The ultimate goal of this project is to implement two-way communication based on both CAN and CAN FD busses using a microcontroller and an FPGA using CAN FD bus utilization. Therefore, in the implementation of this project, two types of implementations (microcontroller and FGA) were developed, and finally, upon successful communication, the data rate, accuracy, and reliability of the transfer were evaluated. Also, efforts have been made to use the CANOpen protocol as a high-level protocol.

Keywords: Communication Interface, Data Bus, Embedded System, Microcontroller, FPGA

Amirkabir University of Technology
(Tehran Polytechnic)

Department of Computer Engineering

B.Sc. Thesis

Implementation of a data communication system between a microcontroller and an FPGA using CAN-FD bus interface and CANOpen Protocol

By
Mohamad Chamanmotlagh

Supervisor
Dr. M. Mehdi Homayounpour

February 2022