รายงานสหกิจศึกษาฉบับสมบูรณ์

การสร้างต้นแบบอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA

TETRA RTU Product Prototyping

นางสาวปารย์วรดา สุนทร

นางสาววฤนดา ปฐมสกุลสุนทร

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2560

รายงานสหกิจศึกษาฉบับสมบูรณ์

การสร้างต้นแบบอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA

TETRA RTU Product Prototyping

นางสาวปารย์วรดา สุนทร

นางสาววฤนดา ปฐมสกุลสุนทร

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2560

ชื่อโครงงานสหกิจศึกษา การสร้างต้นแบบอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA

ชื่อ-สกุล นักศึกษา นางสาวปารย์วรดา สุนทร

นางสาววฤนดา ปฐมสกุลสุนทร

คณะ วิศวกรรมศาสตร์ **ภาควิชา** วิศวกรรมโทรคมนาคม

ชื่อ-สกุล อาจารย์นิเทศ รศ.ดร.ยุทธพงษ์ รังสรรค์เสรี

ดร.เวธิต ภาคย์พิสุทธิ์

ชื่อ-สกุล ผู้นิเทศงาน ดร.สมรักษ์ เพชรชาตรี

ชื่อสถานประกอบการ บริษัท กสท โทรคมนาคม จำกัด (มหาชน)

บทคัดย่อ

โครงงานนี้เป็นการศึกษาและออกแบบอุปกรณ์สำหรับรับส่งข้อมูล ตามมาตรฐาน TETRA (Terrestrial Trunked Radio System) เพื่อเป็นแนวทางการสื่อสารในอนาคตในรูปแบบการรับส่งข้อมูลแบบดิจิทัล เนื่องจากเทคโนโลยีด้านสื่อสารโทรคมนาคมมีการเปลี่ยนแปลงอย่างรวดเร็ว และอุปกรณ์ด้านสื่อสารโทรคมนาคมได้มีการพัฒนาให้สามารถใช้งานได้อย่างมีประสิทธิภาพมากขึ้น ระบบวิทยุสื่อสารในรูปแบบของดิจิทัลจึงเป็นเรื่องน่าสนใจในการนำมาพัฒนาและปรับใช้ให้เกิดประโยชน์ โดยอุปกรณ์ดังกล่าวประกอบด้วยวงจรรวม (Integrated Circuit) ซึ่งมีหน้าที่ประมวลผลข้อมูล รับข้อมูล ส่งข้อมูล และวงจรรวมสำหรับสร้างสัญญาณคลื่นพาห์เพื่อเป็นตัวกลางในการรับส่งข้อมูล

Co-operative Title: TETRA RTU Product Prototyping

Student Intern Name: Miss Parnwarrada Sunton

Miss Warunada Patomsakunsunthorn

Faculty: Engineering **Department:** Telecommunication Engineering

Advisor Name: Assoc.Prof.Dr.Yuttapong Rangsanseri

Dr.Watid Phakpisut

Mentor Name: Dr. Somrak Petchartee

Company: CAT Telecom Public Company Limited

ABSTRACT

This project is to study and design transceiver product based on TETRA standard and for guide about digital communication in the future. Due to Telecommunication Technology always changes and communication device developed for more performance. The digital trunked radio is so interested for development. This Integrated Circuit for processing, sending, receiving and to create carrier.

กิตติกรรมประกาศ

การจัดทำโครงงานการสร้างต้นแบบอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA นี้เป็นโครงงานของบริษัท กสท โทรคมนาคม จำกัด (มหาชน) เกิดขึ้นจากการเริ่มต้นนำเทคโนโลยีใหม่ๆ มาพัฒนาประสิทธิภาพการทำงานในองค์กรให้ดียิ่งขึ้น โดยผลงานดังกล่าวสำเร็จลุล่วงไปด้วยดี เพราะได้รับคำแนะนำและการให้คำปรึกษาตลอดจนติดตามผลความคืบหน้าจากหลายๆ ส่วน ทางคณะผู้จัดทำจึงขอขอบคุณทุกท่านที่ได้ให้ความช่วยเหลือตลอดระยะเวลาที่ได้มีโอกาสเข้าไปดำเนินงานโครงการสหกิจศึกษาและเรียนรู้ประสบการณ์การทำงานต่างๆ ตลอดจนสิ้นสุดโครงการ ตั้งแต่วันที่ 7 มิถุนายน 2560 จนถึง 24 พฤศจิกายน 2560

โครงงานนี้ไม่อาจสำเร็จลุล่วงได้หากขาดความกรุณาของ ดร.สมรักษ์ เพชรชาตรี หัวหน้าแผนกวิจัย บริษัท กสท โทรคมนาคม จำกัด (มหาชน) ผู้เป็นพี่เลี้ยงที่คอยช่วยเหลือ ดูแลเอาใจใส่ และให้คำปรึกษาต่างๆ มอบความทรงจำและประสบการณ์อันมีค่าตลอดระยะเวลาที่ผ่านมา

ขอขอบพระคุณ รศ.ดร.ยุทธพงษ์ รังสรรค์เสรี และ ดร.เวธิต ภาคย์พิสุทธิ์ ผู้เป็นอาจารย์นิเทศโครงการสหกิจศึกษาที่คอยให้การสนับสนุน ติดตามความคืบหน้าของโครงการ ให้คำปรึกษาและแนวทางแก้ไขปัญหาต่างๆ

สุดท้ายนี้ขอขอบคุณเพื่อนนักศึกษาและครอบครัว ที่คอยช่วยเหลือ ผลักดัน และเป็นกำลังใจสำคัญในการฟันฝ่าอุปสรรคต่างๆ ตลอดมา รวมถึงผู้มีพระคุณทุกท่านที่มิได้เอ่ยนามไว้ ณ ที่นี้ที่เป็นส่วนหนึ่งของความสำเร็จทั้งหมด จึงขอขอบคุณไว้ ณ โอกาสนี้

ปารย์วรดา สุนทร

วฤนดา ปฐมสกุลสุนทร

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญภาพ

ภาพที่ หน้า

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญ

เทคโนโลยีด้านสื่อสารโทรคมนาคมมีการเปลี่ยนแปลงอย่างรวดเร็ว โดยระบบวิทยุสื่อสารเป็นเครื่องมือสื่อสารที่สำคัญ ซึ่งในปัจจุบันระบบวิทยุสื่อสารเข้าสู่ยุคดิจิทัล อุปกรณ์ด้านสื่อสารโทรคมนาคมได้มีการพัฒนาให้สามารถใช้งานได้โดยมีประสิทธิภาพมากขึ้น ระบบวิทยุสื่อสารในรูปแบบของดิจิทัลจึงเป็นเรื่องน่าสนใจในการนำมาพัฒนาและปรับใช้ให้เกิดประโยชน์ บริษัท กสท โทรคมนาคม จำกัด (มหาชน) จึงได้ศึกษาและออกแบบอุปกรณ์สำหรับรับส่งข้อมูล Terrestrial Trunked Radio System Remote Terminal Unit (TETRA RTU) ตามมาตรฐาน TETRA (Terrestrial Trunked Radio System) ซึ่งถูกกำหนดโดยหน่วยงาน ETSI โดยมุ่งหวังให้เกิดทางเลือกใหม่ในการนำเทคโนโลยีสื่อสารแบบดิจิทัลมาใช้พัฒนาระบบการรับส่งข้อมูลแทนระบบสื่อสารผ่านเครือข่ายผู้ให้บริการโทรศัพท์มือถือ ให้สามารถรับส่งข้อมูลได้โดยไม่ต้องเสียค่าใช้จ่าย นอกจากนี้ ระบบ trunk radio มาตรฐาน TETRA ที่นำมาศึกษานี้ยังมีการใช้งานแพร่หลายในหลายหน่วยงาน เช่น บริษัทวิทยุการบิน การไฟฟ้านครหลวง การสื่อสารแห่งประเทศไทย กรมการปกครอง และรถเมล์ BRT ของกรุงเทพมหานคร

ซึ่งการศึกษาและออกแบบอุปกรณ์สำหรับรับส่งข้อมูลนี้ จะช่วยให้ทราบถึงแนวทางการสื่อสารในอนาคตในรูปแบบการรับส่งข้อมูลแบบดิจิทัล และอาจใช้งานเป็นการสื่อสารช่องทางสำรองอย่างมีประสิทธิภาพต่อไป

1.2 วัตถุประสงค์ของการวิจัย

1) เพื่อศึกษาอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA
2) เพื่อศึกษาการทำงานและการตั้งค่าวงจรรวม (Integrated Circuit)
3) เพื่อศึกษาการเขียน Script ในการควบคุมการทำงานของวงจรรวม CMX981
4) เพื่อศึกษาการเขียนโปรแกรมคำสั่งบน FPGA (Field Programmable Gate Array) โดยใช้ระบบปฏิบัติการ Linux

1.3 ขอบเขตของการวิจัย

ศึกษาอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA เพื่อเป็นแนวทางการสื่อสารในอนาคตในรูปแบบการรับส่งข้อมูลแบบดิจิทัล โดยแบ่งการทำงานออกเป็น 3 ส่วน คือ 1) ส่วนการประมวลผลข้อมูล 2) ส่วนการรับข้อมูล 3) ส่วนการส่งข้อมูล โดยทั้ง 3 ส่วน ถูกควบคุมโดยไมโครโพรเซสเซอร์ (Microprocessor) บนระบบปฏิบัติการลินุกซ์ (Linux) ซึ่งการทำงานจะเริ่มต้นจากการสั่งการผ่านไมโครโพรเซสเซอร์ เพื่อไปตั้งค่าวงจรรวม (Integrated Circuit) ทั้ง 3 ตัว รวมถึงการสั่งการเพื่อส่งข้อมูล โดยข้อมูลจะถูกส่งจากวงจรรวม CMX981 ไปยังวงจรรวม CMX998 เพื่อส่งออกไปในตัวกลาง จากนั้นวงจรรวม CMX994 จะเป็นตัวรับข้อมูลและส่งไปยังวงจรรวม CMX981 เพื่อประมวลผลและแสดงผลต่อไป

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1) ได้รับความรู้เกี่ยวกับมาตรฐาน TETRA
2) ได้รับความรู้เกี่ยวกับการใช้งานบนระบบปฏิบัติการ Linux
3) ได้รับความรู้เกี่ยวกับอุปกรณ์และ IC ต่างๆ ที่ใช้ในการรับส่งข้อมูล
4) ได้รับความรู้เรื่องการเขียนภาษา C สำหรับโปรแกรมคำสั่งบน FPGA
5) ได้รับความรู้ในการเขียน Script เพื่อควบคุม CMX981
6) มีความเข้าใจขั้นตอนการทำงานและสามารถปรับตัวเข้ากับสภาพแวดล้อมการทำงานของบริษัท กสท โทรคมนาคม จำกัด (มหาชน)
7) เรียนรู้อุปสรรคและการแก้ไขปัญหาในเหตุการณ์ต่างๆ ในการทำงาน ทั้งจากการปรึกษาเพื่อนร่วมงาน ปรึกษาอาจารย์ที่ปรึกษา ปรึกษาพี่เลี้ยง และศึกษาเรียนรู้ด้วยตนเอง

บทที่ 2
แนวคิด ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

โครงงาน "ต้นแบบการสื่อสารบนมาตรฐาน TETRA RTU" ได้ทำการพัฒนาการสื่อสารบนมาตรฐาน TETRA เพื่อให้สามารถใช้งานได้เต็มประสิทธิภาพมากขึ้น

2.1 มาตรฐาน TETRA (Terrestrial Trunked Radio) [1]

เป็นชุดของมาตรฐานที่ได้รับการพัฒนาโดย European Telecommunications Standardization Institute (ETSI) ที่อธิบายถึงโครงสร้างพื้นฐานการสื่อสารของวิทยุสื่อสารเคลื่อนที่ โครงสร้างพื้นฐานนี้มีเป้าหมายเป็นหลักสำหรับความต้องการของกลุ่มความปลอดภัยสาธารณะ ที่ให้บริการด้านการสื่อสารด้วยเสียงและข้อมูล ซึ่งเป็นมาตรฐานของเครือข่ายวิทยุสื่อสารเฉพาะกลุ่ม (trunked radio) โดยใช้ Time Division Multiple Access (TDMA) ในความถี่ 1 ความถี่จะแบ่งออกเป็น 4 Time Slot ซึ่งจะสามารถรองรับการสื่อสารแบบกลุ่มได้ถึง 4 กลุ่มพร้อมกัน รองรับการสื่อสารทางเสียงได้ทั้งแบบกลุ่ม (Group Call) แบบเดี่ยว (Private Call) โทรศัพท์พื้นฐาน และโทรศัพท์มือถือ (Telephone Interconnect) การสื่อสารกับวิทยุ Walkie Talkie และวิทยุสื่อสารแบบ Conventional ต่างๆ รองรับการสื่อสารข้อมูลทั้งแบบข้อความสั้น (Short Message Service, SMS) ข้อมูลสั้น (Short Data Service, SDS) และการสื่อสารแบบ Packet Data ซึ่งสามารถส่งข้อมูลได้ความเร็ว 7.2 – 28.8 kbps ซึ่งมีความปลอดภัยเพราะเป็นการสื่อแบบ Point-to-Point และเป็นการเข้ารหัสแบบ end-to–end โดยใช้ความถี่ทั้งหมด 5 ย่าน [2]

1) ย่านความถี่วิทยุ 380 - 399.9 MHz
2) ย่านความถี่วิทยุ 421.8 - 422.95 / 433.8 - 434.95 MHz
3) ย่านความถี่วิทยุ 484 - 489 / 494 - 499 MHz
4) ย่านความถี่วิทยุ 806 - 821 / 851 - 866MHz
5) ย่านความถี่วิทยุ 821 - 824 / 866 - 869MHz

2.1.1 ข้อดีของมาตรฐาน TETRA

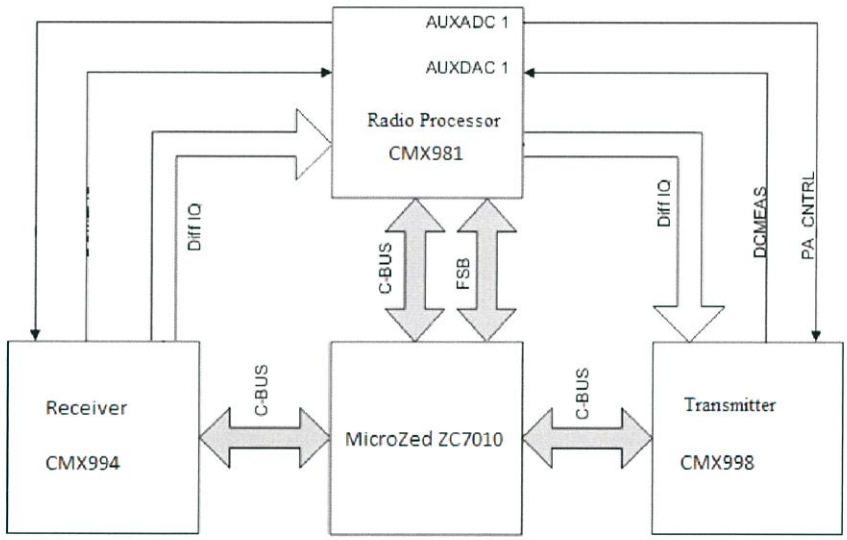
1) ในความถี่ต่ำจะครอบคลุมพื้นที่มากกว่า ซึ่งจะใช้เครื่องส่งสัญญาณน้อย จึงทำให้ลดต้นทุนด้านโครงสร้างพื้นฐาน
2) ระหว่างการสื่อสารข้อมูลผ่านเสียงจะไม่โดนการรบกวนหรือขัดจังหวะ
3) สามารถสื่อสารโดยตรงในโหมด Walkie talkie
4) เป็นการเชื่อมต่อแบบ Point-to-Point โดยไม่ต้องมีส่วนเกี่ยวข้องกับ Operator หรือ Dispatcher
5) เข้ารหัสแบบ End-to-End ซึ่งทำให้มีความปลอดภัยสูง

2.1.2 ข้อเสียของมาตรฐาน TETRA

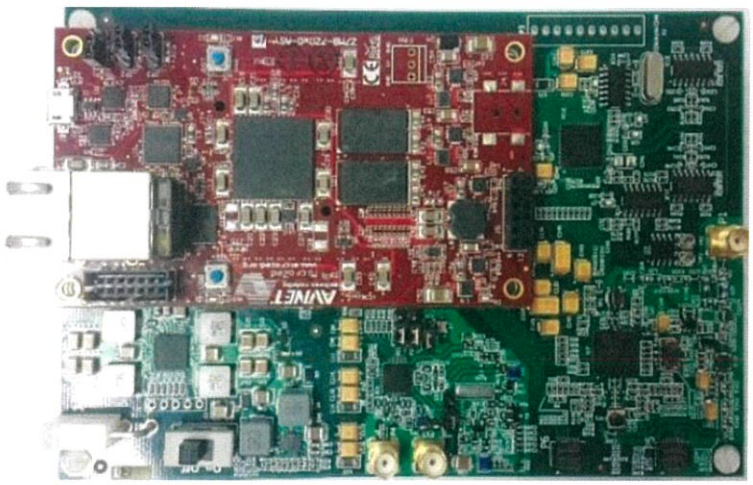
1) ต้องใช้เครื่องขยายเชิงเส้นเพื่อให้เป็นไปตามข้อกำหนดของ RF
2) ความเร็วในการรับส่งข้อมูลต่ำกว่ามาตรฐานยุคใหม่อื่นๆ

2.2 บอร์ด TETRA RTU

อุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA โดยแบ่งการทำงานออกเป็น 3 ส่วน คือ 1) ส่วนการประมวลผลข้อมูล 2) ส่วนการรับข้อมูล 3) ส่วนการส่งข้อมูล โดยทั้ง 3 ส่วน ถูกสั่งการผ่านไมโครโพรเซสเซอร์ (Microprocessor) บนระบบปฏิบัติการลินุกซ์ (Linux) ซึ่งการทำงานจะเริ่มต้นจากการสั่งการผ่านไมโครโพรเซสเซอร์เพื่อไปตั้งค่าวงจรรวม (Integrated Circuit) ทั้ง 3 ตัว รวมถึงการสั่งการเพื่อส่งข้อมูล โดยข้อมูลจะถูกส่งจากวงจรรวม CMX981 ไปยังวงจรรวม CMX998 ซึ่งจะนำข้อมูลไป มอดูเลตกับสัญญาณคลื่นพาห์เพื่อส่งออกไปในตัวกลาง จากนั้นวงจรรวม CMX994 จะเป็นตัวรับข้อมูลและส่งไปยังวงจรรวม CMX981 เพื่อประมวลผลและแสดงผลต่อไป โดยบล็อกไดอะแกรมของระบบแสดงดังภาพที่ 2.1 และบอร์ด TETRA RTU แสดงดังภาพที่ 2.2



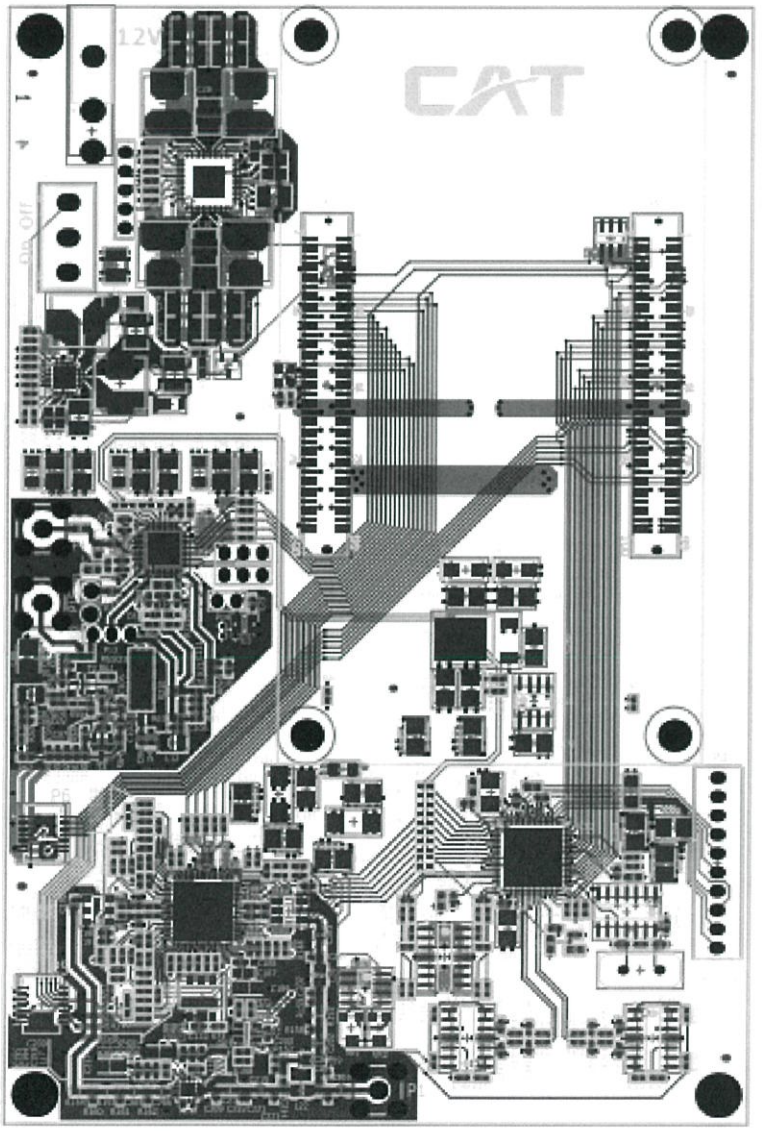
ภาพที่ 2.1 บล็อกไดอะแกรมของอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA



ภาพที่ 2.2 บอร์ด TETRA RTU

การออกแบบโครงสร้างของอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA ได้มีการทดลองจากบอร์ด EV9810, EV9980 และ EV9942 ซึ่งเป็นบอร์ด Evaluation Kit ที่มีการใช้วงจรรวม CMX981, CMX998 และ CMX994 เพื่อทดสอบว่าสามารถใช้วงจรรวมดังกล่าวได้หรือไม่ จากนั้นจึงได้นำโครงสร้างวงจรของทั้งสามบอร์ดมาเป็นต้นแบบในการออกแบบอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA โดยแบ่งโครงสร้างภายในออกเป็น 3 ส่วน ส่วนแรกเป็นส่วนประมวลข้อมูล ซึ่งใช้วงจรรวม CMX981 เป็นตัวประมวลผลและส่งไปยังส่วนที่สองซึ่งมีหน้าที่ในการส่งข้อมูล และใช้วงจรรวม CMX998 ในส่วนสุดท้ายเป็นส่วนการรับข้อมูล จะใช้วงจรรวม CMX994 ในการรับข้อมูลและส่งกลับไปยังส่วนการประมวลผล โดยมี MicroZed ZYNQ7000 เป็นตัวกลางในการสั่งการและควบคุมต่างๆ

จากวงจรต้นแบบของบอร์ดทั้งสามบอร์ด ได้มีการนำวงจรดังกล่าวมาเชื่อมต่อและออกแบบเป็นวงจรใหม่ของอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA แสดงดังภาพที่ 2.3



ภาพที่ 2.3 วงจรของอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA

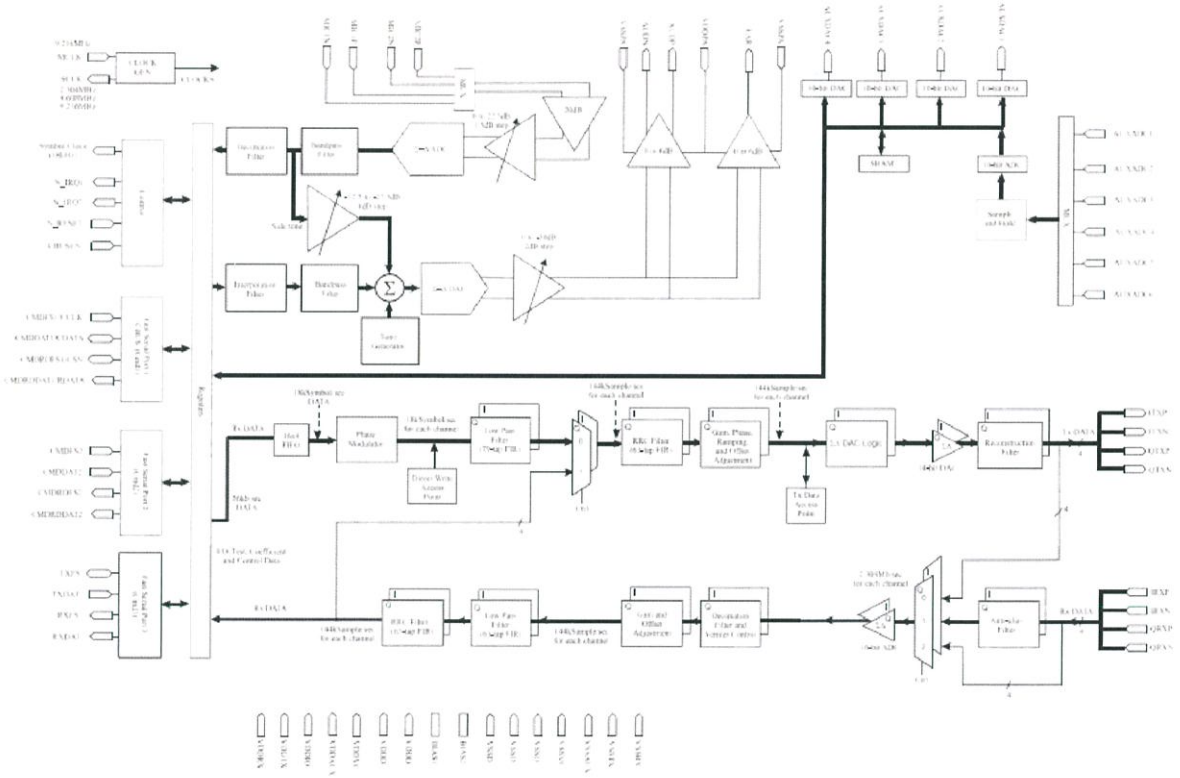
2.3 CMX981 Advanced Digital Radio Baseband Processor [5]

CMX981 เป็นโปรเซสเซอร์แปลงสัญญาณและตัวประมวลผลแบบผสมผสานที่ใช้กับระบบแอนาล็อกและดิจิทัลของระบบวิทยุดิจิทัลและทำหน้าที่สำคัญอย่างยิ่งสำหรับ DSP (Digital signal processing) รองรับการทำงานพื้นฐานมากมาย เช่น TETRA, APCO25, RCR-39 (Japan) เป็นต้น

CMX981 ประกอบด้วยฟังก์ชันทั้งหมดที่จำเป็นในการแปลงข้อมูลสัญลักษณ์ดิจิทัลเป็นสัญญาณแอนาล็อก I และ Q ซึ่งรวมถึงการควบคุมเอาท์พุตของดิจิทัล การชดเชยเอาท์พุตและตัวกรองแบบดิจิทัลที่สามารถตั้งค่าได้ มีการมอดูเลตแบบ Differential Quadrature Phase Shift Keying (DQPSK)

การรับสัญญาณอินพุต I และ Q โดยการกรองข้อมูลในช่องสัญญาณดิจิทัลเพื่อลดความยุ่งยากในการประมวลผลและการดึงข้อมูล สามารถตั้งค่าการแก้ไขชดเชยดิจิทัลและตัวกรองแบบดิจิทัล

Auxiliary DAC และ ADC ใช้ควบคุมและวัดสัญญาณคลื่นวิทยุ RF ซึ่งรวมถึง AFC, AGC และ RSSI สามารถรับ แปลงและเข้ารหัสสัญญาณเสียงได้โดยมีบล็อกไดอะแกรมของ CMX981 ดังภาพที่ 2.4



ภาพที่ 2.4 บล็อกไดอะแกรมของ CMX981 [5]

ภาพที่ 2.5 วงจรรวม CMX981 [7]

ภาพที่ 2.5 แสดงวงจรรวม CMX981 โดยรายละเอียดของ วงจรรวม CMX981 ดังนี้

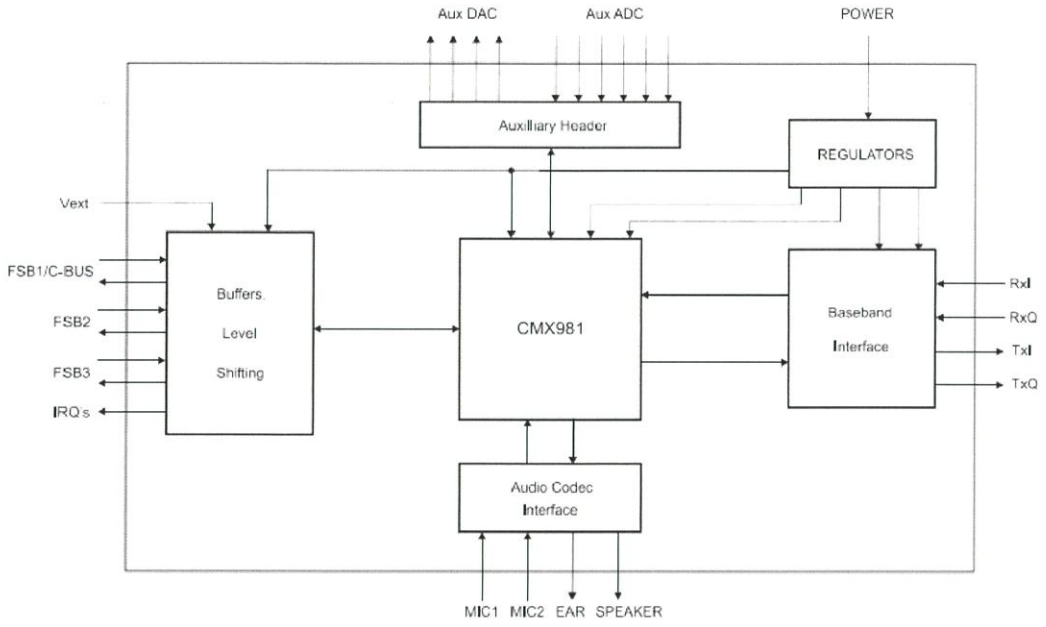
- ไฟเลี้ยง 2.5 โวลต์ ไม่เกิน 3.3 โวลต์
- Radio Rx: 2 x 16-Bit Sigma Delta ADC
- Radio Tx: 2 x 14-Bit Sigma Delta DAC
- Auxiliary: 6 x 10-Bit ADC
- Auxiliary: 4 x 10-Bit DAC
- Voice : 14-Bit Linear with Digital Filter
- มอดูเลตแบบ $\pi/4$ Differential Quadrature Phase Shift Keying (DQPSK)
- C-BUS and 3 Fast Serial Bus Interface
- 130mW Speaker Amplifier (8 Ω load)
- 16.5mW Earpiece Amplifier (32 Ω load)

การติดต่อสื่อสารกับ CMX981 ติดต่อสื่อสารได้ 2 โหมด 1) โหมด FSB ใช้ในการรับส่งข้อมูล 2) โหมด C-BUS ใช้ตั้งค่ารีจิสเตอร์ โดยการติดต่อสื่อสารแสดงดังตารางที่ 2.1

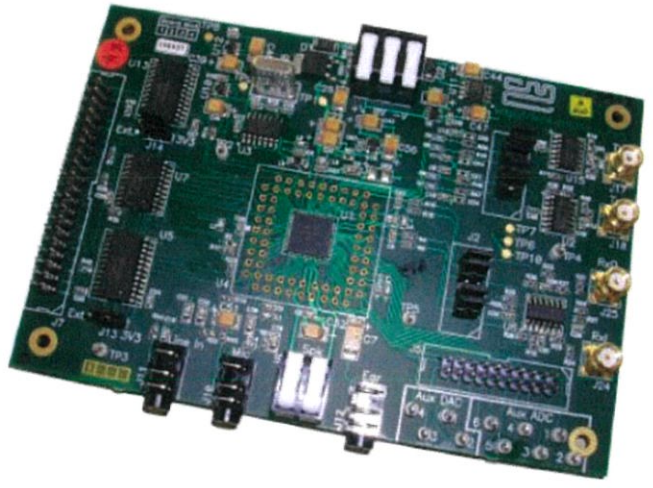
ตารางที่ 2.1 การติดต่อสื่อสารกับ CMX981

ประเภทของสัญญาณ	พอร์ตที่ใช้
รีจิสเตอร์และข้อมูล	Cmd1 (FSB1)
สัมประสิทธิ์ FIR	Cmd1 (FSB1)
การเข้ารหัสเสียง	Cmd2 (FSB2)
ข้อมูล ADC/DAC	Cmd2 (FSB2)
ข้อมูล FIR Rx/Tx	TxRx (FSB3)
ฟิลเตอร์ Direct Write to 79-tap	TxRx (FSB3)
ทดสอบอินพุต DAC	TxRx (FSB3)

Evaluation Kit 9810 (EV9810) สามารถใช้ในการประเมินผลการทดลองและการออกแบบ CMX981 Advanced Digital Radio Baseband Processor โดยจะประกอบด้วยอุปกรณ์ CMX981 บนบอร์ด เพื่อใช้ในการเข้าถึงสัญญาณข้อมูลเบสแบนด์ สัญญาณควบคุมและอินเทอร์เฟซข้อมูลของ CMX981 ตลอดจนฟังก์ชัน ADC และ DAC โดยมีบล็อกไดอะแกรมของ Evaluation Kit 9810 ดังภาพที่ 2.6 และ Evaluation Kit 9810 ดังภาพที่ 2.7 [8]

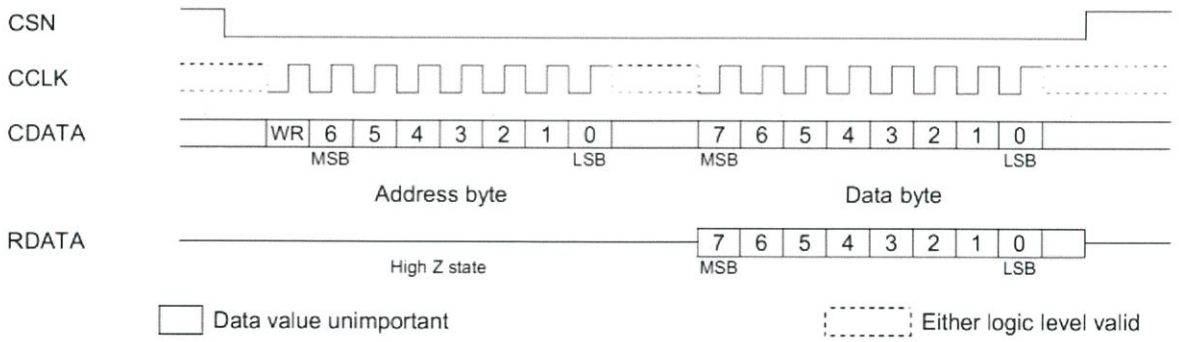


ภาพที่ 2.6 บล็อกไดอะแกรมของ Evaluation Kit 9810 [8]



ภาพที่ 2.7 Evaluation Kit 9810 [8]

EV9810 ประกอบด้วย CMX981 (Advanced Digital Radio Baseband Processor) ซึ่งเป็นส่วนประกอบที่ถูกควบคุมโดยรีจิสเตอร์ที่สามารถเข้าถึงได้ผ่านทาง CBUS กระบวนการในการอ่านและเขียนรีจิสเตอร์ของ CMX981 ผ่านทาง CBUS แสดงดังภาพที่ 2.8 สำหรับการอ่าน PE0002 จะส่งค่าแอดเดรสที่ CCLK 8 บิตแรก ซึ่งบิตที่มีนัยสำคัญสูงสุด (Most Significant Bit) มีค่าเป็น 0 และข้อมูลจะอยู่ที่ 8 บิตถัดไป สำหรับการเขียน บิตที่มีนัยสำคัญสูงสุด (บิต 7) จะต้องมีค่าเป็น 1

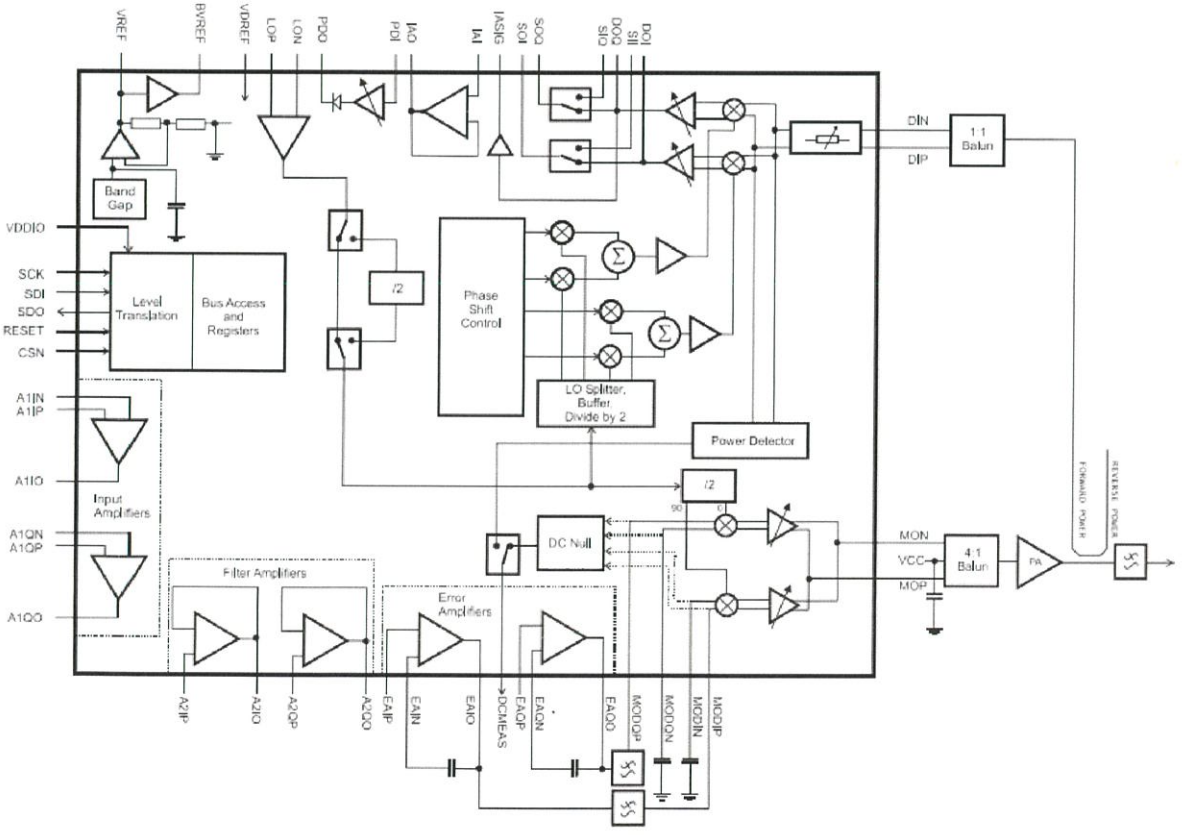


ภาพที่ 2.8 C-BUS [5]

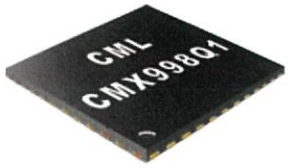
2.4 CMX998 Cartesian Feed-back Loop Transmitter [9]

เครื่องส่งสัญญาณแบบ Cartesian Feedback Loop (CFBL) ทำหน้าที่เป็นตัวแปลงสัญญาณโดยตรงจาก I และ Q ไปยังเอาท์พุต RF เป็น Linearize Power Amplifier (PA) โดย Cartesian Loop ช่วยเพิ่มประสิทธิภาพและความเป็น Linearity ของตัวส่งสัญญาณ

ความถูกต้องในการมอดูเลตของการส่งผ่าน CMX998 จะถูกกำหนดโดยตัวแปลงสัญญาณ down – converter แอมปลิจูดและเฟสของ I และ Q จะสมดุลกัน ซึ่งเป็นปัจจัยที่ทำให้การมอดูเลตมีความถูกต้อง โดยมีการรับประกันค่าที่แย่ที่สุด –30 dB สามารถใช้บนมาตรฐาน TETRA โดยมีบล็อกไดอะแกรมของ CMX998 ดังภาพที่ 2.9



ภาพที่ 2.9 บล็อกไดอะแกรมของ CMX998 [9]

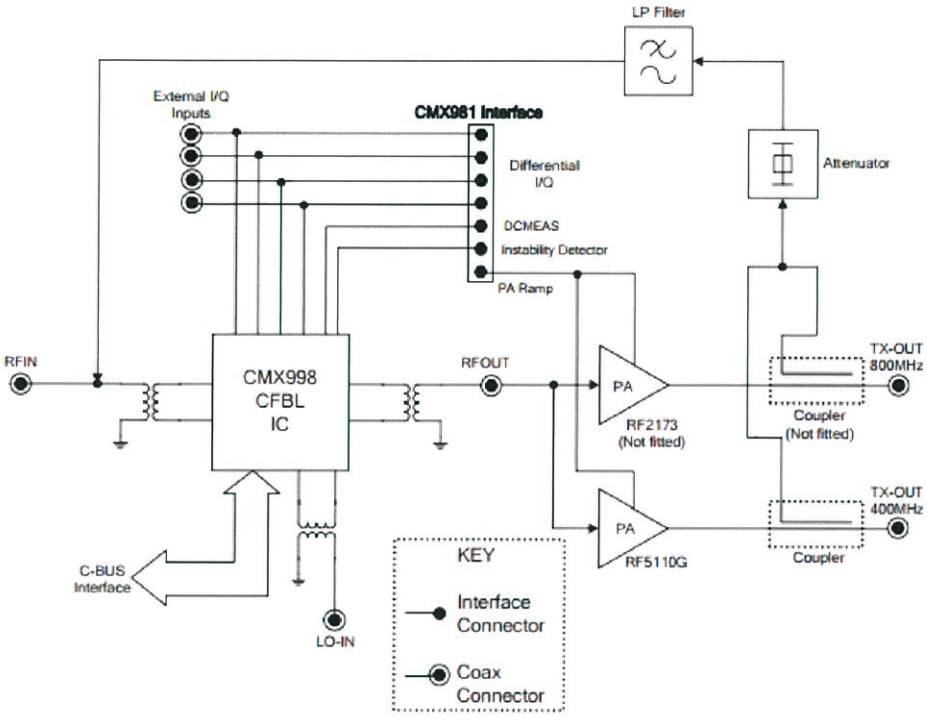


ภาพที่ 2.10 วงจรรวม CMX998 [9]

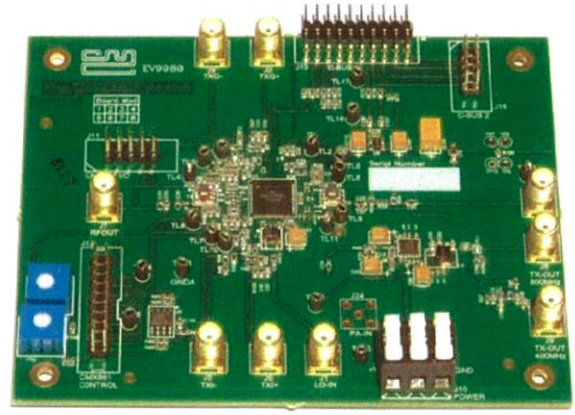
ภาพที่ 2.10 แสดงวงจรรวม CMX998 โดยรายละเอียดของ วงจรรวม CMX998 ดังนี้

- ความถี่ช่วง 30 MHz ถึง 1 GHz
- Wide Band Noise –148dBc/Hz
- C – BUS Serial Interface
- Gain Control
- Error Amplifier
- Up Converter Forward Path
- Down Converter for Feedback Linearisation
- 360° Loop Phase Shift Control
- DC Offset Measurement Output
- Flexible Digital Interface
- Linear Modulation Schemes เช่น QPSK, π/4-DQPSK, 8PSK, QAM, OFDM, F4FM เป็นต้น

Evaluation Kit 9980 (EV9980) สามารถใช้ในการประเมินผลการทดลองและการออกแบบ CMX998 Cartesian Feed-back Loop Transmitter โดยจะประกอบด้วย PCB ที่มีแพลตฟอร์มแบบยืดหยุ่น ทำให้สามารถใช้อินเทอร์เฟซควบคุมเพื่อกำหนดค่าและคลื่นความถี่ต่างๆ สามารถเข้าถึงสัญญาณทั้ง RF และ baseband โดยมีกำลังขยาย 450 MHz หรือ 800 MHz โดย EV9980 สามารถใช้ความถี่ระหว่าง 100 MHz ถึง 1 GHz สามารถทำงานร่วมกับ CMX981 โดยมีบล็อกไดอะแกรมของ Evaluation Kit 9980 ดังภาพที่ 2.11 และ Evaluation Kit 9980 ดังภาพที่ 2.12 [10]



ภาพที่ 2.11 บล็อกไดอะแกรมของ Evaluation Kit 9980 [10]

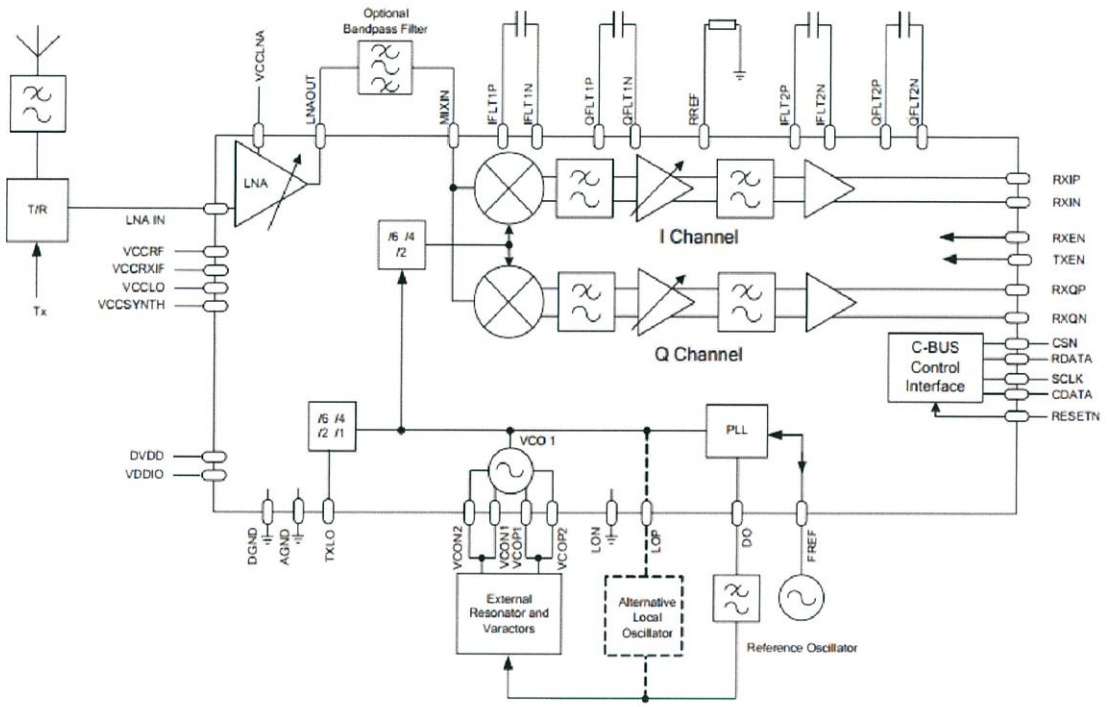


ภาพที่ 2.12 Evaluation Kit 9980 [10]

2.5 CMX994 Direct Conversion Receiver [12]

CMX994 เป็นตัวรับสัญญาณโดยแปลงสัญญาณแบบตรง ประกอบด้วย LAN (Local Area Network) แบบบรอดแบนด์ ตัวควบคุมกำลังขยายแบบไดนามิกสูงในการดีมอดูเลต I และ Q และ IIP2 ในส่วนตัวรับสัญญาณประกอบด้วยตัวขยายสัญญาณและตัวกรองแบบเบสแบนด์ที่แม่นยำ และ VCO (Votage – Controlled Oscillator) โดยโหมดพลังงานต่ำนั้นจะช่วยปิดการปรับเฟสของ LO ทำงานจากแหล่งจ่ายไฟ 3.3V เดียวในช่วงอุณหภูมิตั้งแต่ -40 ถึง +85 องศาเซลเซียสและมีอยู่ในชุด VQFN โดยมีบล็อกไดอะแกรมของ CMX994 ดังภาพที่ 2.13

- CMX994 โหมดมาตรฐานและโหมดพลังงานต่ำ
- CMX994A โหมดมาตรฐานและเพิ่มโหมดพลังงานต่ำ
- CMX994E โหมด Enhanced โหมดมาตรฐานและโหมดพลังงานต่ำ



ภาพที่ 2.13 บล็อกไดอะแกรมของ CMX994 [12]

ภาพที่ 2.14 วงจรรวม CMX994 [11]

ภาพที่ 2.14 แสดงวงจรรวม CMX994 โดยรายละเอียดของ IC CMX994 ดังนี้

1) Rx Direct Conversion Receiver

- LNA with gain control
- 100MHz to 940MHz I/Q demodulator
- Extended low frequency operation – down to 50MHz
- Precise filtering with bandwidth setting and 1 : 2 : 4 bandwidth modes

2) Local Oscillator

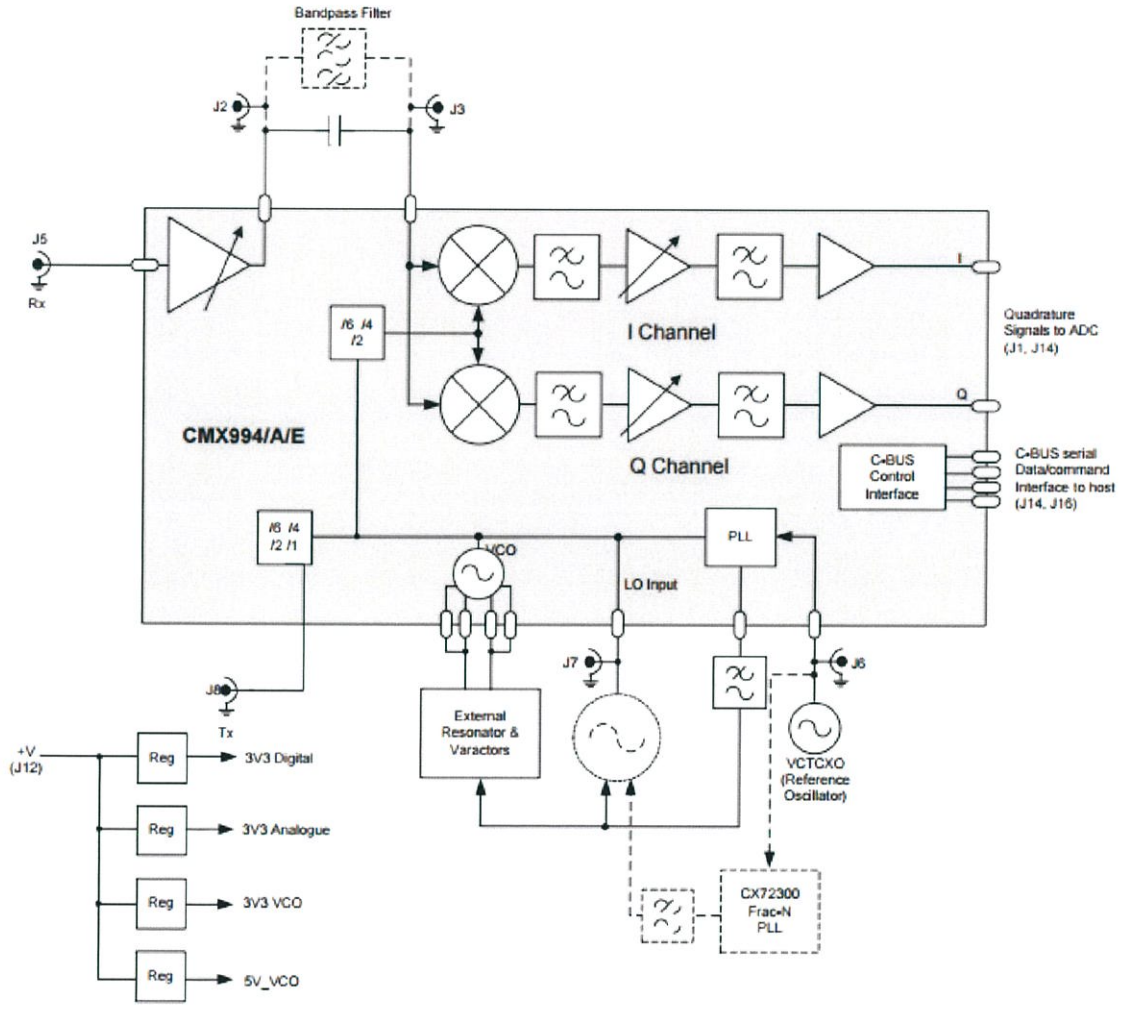
- LO synthesizer
- VCO negative resistance amplifier
- LO divide by 2, 4 or 6 modes
- Tx LO Output

3) ไฟเลี้ยง 3.0 V – 3.6 V

4) Small size 40-pin VQFN Package

5) Low - power Mode

Evaluation Kit 9942 (EV9942) สามารถใช้ในการประเมินผลการทดลองและการออกแบบ CMX994 Direct Conversion Receiver ประกอบด้วย VCO tank, Fractional-N PLL และ VCO ความถี่สูง อินเทอร์เฟซที่ใช้กันทั่วไปช่วยให้สามารถเชื่อมต่ออย่างรวดเร็วสำหรับการประเมิน CML baseband processor IC Evaluation Kit เช่นตระกูล PE0601 และตระกูล PE0403 โดยใช้ PE0003 Universal Interface Card ช่วงความถี่ RF ของชุดประเมินอยู่ที่ 100MHz ถึง 940MHz ซึ่งมีการทำงานเริ่มต้นที่ศูนย์กลางที่ 448MHz บล็อกไดอะแกรมของ Evaluation Kit 9942 ดังภาพที่ 2.15 และ Evaluation Kit 9942 ดังภาพที่ 2.16 [13]

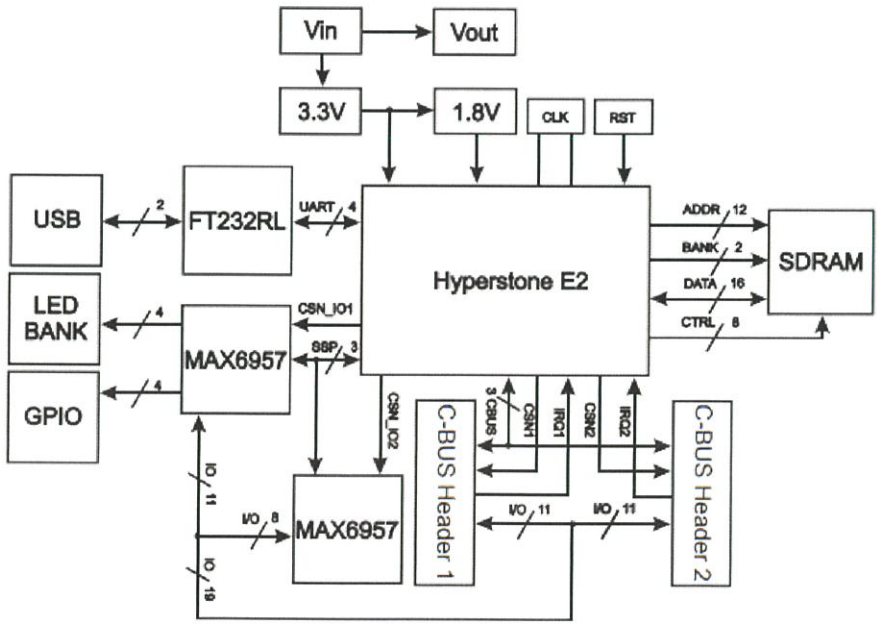


ภาพที่ 2.15 บล็อกไดอะแกรมของ Evaluation Kit 9942 [13]

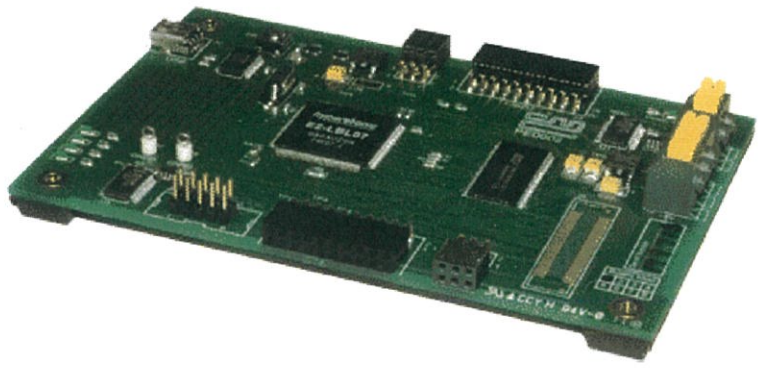
ภาพที่ 2.16 Evaluation Kit 9942 [13]

2.6 PE0002 Evaluation Kit Interface Card [14]

PE0002 เป็นระบบอินเทอร์เฟซสำหรับใช้กับชุดประเมินของบริษัท CML รวมทั้งผลิตภัณฑ์จาก FirmASIC ซึ่งจะช่วยลดความซับซ้อนในการประเมินผลและออกแบบขั้นตอนในการทดลอง จากการใช้ไมโครโพรเซสเซอร์ RISC / DSP 32 บิต ของ Hyperstone และใช้ PC GUI ข้อมูลที่สร้างจะอยู่ในรูปแบบเวลา และส่งไปยัง IC เป้าหมาย โดยใช้อินเทอร์เฟซแบบอนุกรมของ C–BUS สามารถดำเนินการอ่านและเขียนโดยการเขียนสคริปต์ ติดต่อสื่อสารกับคอมพิวเตอร์ผ่านทางพอร์ต USB (Universal Serial Bus) โดยมีบล็อกไดอะแกรมของ PE0002 ดังภาพที่ 2.17



ภาพที่ 2.17 บล็อกไดอะแกรมของ PE0002 [14]



ภาพที่ 2.18 PE0002 Evaluation Kit Interface Card [14]

ภาพที่ 2.18 แสดง PE0002 Evaluation Kit Interface Card มีรายละเอียดดังนี้

- Global interface for new generation IC evaluation kits
- Target IC C-BUS read and write operations
- 32-Bit RISC/DSP Microprocessor based operation
- Mating interface for wide range of target evkit boards
- PC GUI and hardware provided
- High-speed real-time script execution
- PC control/communications via USB
- Software configurable GPIO lines and LED bank

โดย PE0002 มีขาตามตารางที่ 2.2 ดังนี้

ตารางที่ 2.2 ขาของ PE0002

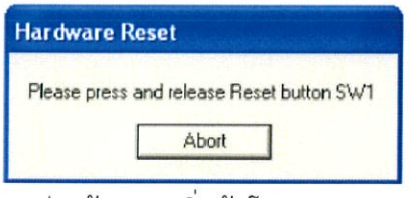
Connector Ref.	Connector Pin No.	Signal Name	Description
J2	1	VBUS	USB Mini B type connector – USB power supply
	2	D-	USB Mini B type connector – USB data signal minus
	3	D+	USB Mini B type connector – USB data signal plus
	4	n/c	
	5	GNDD	USB Mini B type connector – Digital ground
	6	SHELL	USB Mini B type connector – USB connector shell
	7	n/c	Spare I/O
J3	1	IO8	Spare I/O

	2	CSN2	C-Bus chip select
	3	IO9	Spare I/O
	4	CDATA	C-Bus command data
	5	IO10	Spare I/O
	6	SCLKCBUS	C-BUS serial clock
	7	IO11	Spare I/O
	8	RDATA	C-BUS reply data
	9	IO12	Spare I/O
	10	IRQN2	C-BUS interrupt request
	11,12	GNDD	Digital ground
	13	BOOTEN1	Hardware boot control
	14	BOOTEN2	Hardware boot control
	15	RS232/C-BUS	Hardware boot control
	16	IO13	Spare I/O
	17	IO14	Spare I/O
	18	IO15	Spare I/O
	19,20	n/c	
J5	1	IO0	Spare I/O
	2	CSN1	C-BUS chip select
	3	IO1	Spare I/O
	4	CDATA	C-BUS command data
	5	IO2	Spare I/O
	6	SCLKCBUS	C-BUS serial clock
	7	IO3	Spare I/O
	8	RDATA	C-BUS reply data
	9	IO4	Spare I/O
	10	IRQN1	C-BUS interrupt request
	11,12	GNDD	Digital ground
	13	BOOTEN1	Hardware boot control
	14	BOOTEN2	Hardware boot control
	15	RS232/C-BUS	Hardware boot control
	16	IO5	Spare I/O
	17	IO6	Spare I/O
	18	IO7	Spare I/O
	19,20	n/c	
J6	1	GPIO0	Spare I/O
	2	n/c	

	3	GPIO1	Spare I/O
	4	n/c	
	5	GPIO2	Spare I/O
	6	n/c	
	7	GPIO3	Spare I/O
	8	n/c	
	9	GNDD	Digital ground
	10	VCC3V3	+3.3V power supply
J7	1,2	GNDD	Digital ground
	3,4,5,6	VCCIN	+5.0V power supply
J8	1	GNDD	Digital ground
	2	VCCIN	+5.0 power supply
J9	1,2	GNDD	Digital ground
	3,4,5,6	VCCIN	+5.0V power supply
J10	1	GNDD	Digital ground
	2,3	VCCIN	+5.0V power supply

2.6.1 ES000243 [14]

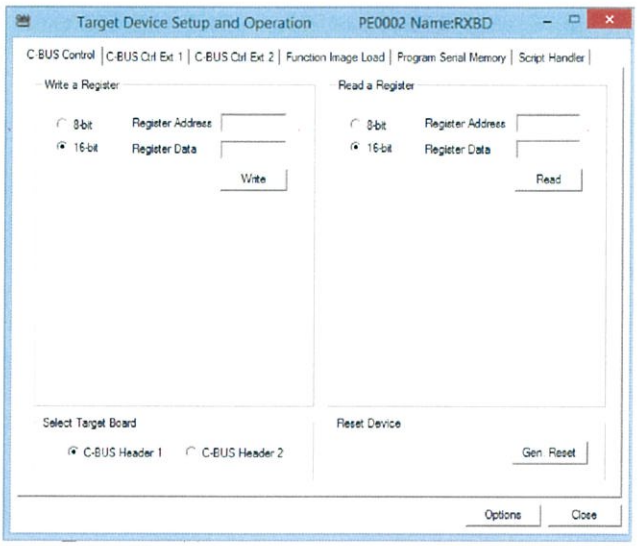
ES000243 เป็นโปรแกรมที่ใช้ในการสื่อสารกับ PE0002 บนระบบปฏิบัติการ Window โดยการใช้งานโปรแกรม ES000243.EXE หากไม่สามารถติดต่อสื่อสารจะปรากฏข้อความว่าล้มเหลว แต่ถ้าหากสามารถสื่อสารได้จะปรากฏกล่องข้อความตามภาพที่ 2.19 โดยจะให้กดสวิตช์รีเซ็ต SW1 บน PE0002 เฟิร์มแวร์จะดาวน์โหลดโดยอัตโนมัติไปยังบอร์ด PE0002 และกล่องตอบโต้จะเข้าสู่โปรแกรมหลัก โดยสามารถเชื่อมต่อ อัตโนมัติกับคอมพิวเตอร์ได้เมื่อต่อ PE0002



ภาพที่ 2.19 กล่องข้อความเริ่มต้นโปรแกรม ES000243 [14]

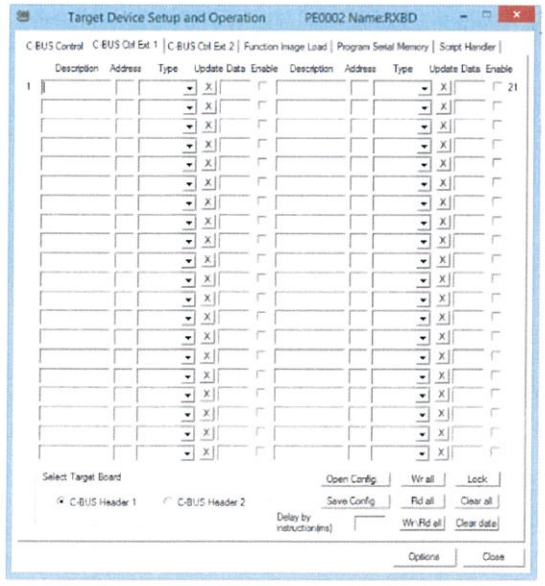
โดยโปรแกรม ES000243 จะมีทั้งหมด 5 ส่วน ดังนี้

1) C–BUS Control Tab โดยจะสามารถอ่าน เขียนและเซ็ตค่าทั่วไปของรีจิสเตอร์ โดยเมื่อป้อนอักษรลงในช่องแก้ไขจะทำการตรวจสอบเพื่อให้แน่ใจว่าเป็นเลขฐานสิบหก สามารถเลือกอ่านและเขียน 8 บิตหรือ 16 บิต โดยความยาวของค่าที่ถูกป้อนจำกัดไว้ที่ 2 อักษร (1 ไบต์) ในการอ่านหรือเขียน และความยาวของที่อยู่จำกัดไว้ที่ 2 อักษรหรือ 4 อักษร (1 ไบต์หรือ 2 ไบต์) โดย COM 1 จะเชื่อมต่อที่ J5 ของ PE0002 และ COM 2 จะเชื่อมต่อที่ J3 ซึ่ง C–BUS Control Tab แสดงในภาพที่ 2.20



ภาพที่ 2.20 C–BUS Control Tab

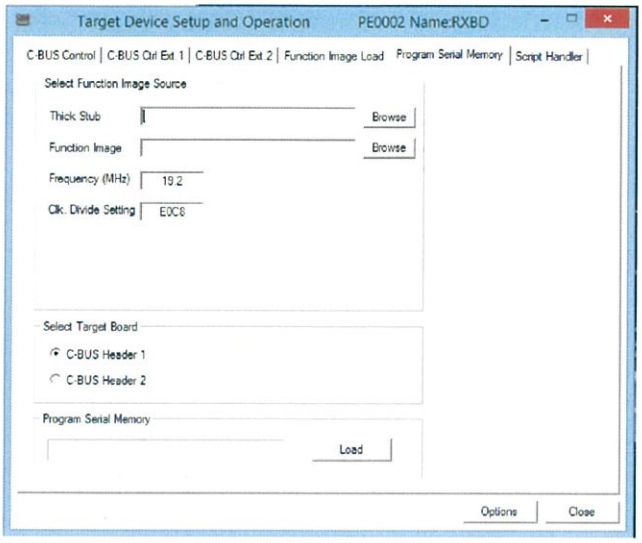
2) C-BUS Control Extended Tab (C-BUS Ctrl Ext. 1 and 2) มีฟังก์ชันการอ่านและเขียน C–BUS หลายชุด ในแต่ละแถวแสดงถึงรีจิสเตอร์ของ C–BUS โดยจะมีการตรวจสอบข้อมูลที่ป้อนลงในช่องข้อมูลเพื่อตรวจสอบว่าเป็นเลขฐานสิบหกหรือไม่ โดยความยาวที่อยู่ที่ป้อนจะถูกจำกัดที่ 2 อักษร (1 ไบต์) และสำหรับข้อมูลรีจิสเตอร์จำกัดที่ 2 อักษรหรือ 4 อักษร (1 ไบต์หรือ 2 ไบต์) กดปุ่มอัพเดทเพื่ออ่านหรือเขียนผ่านรีจิสเตอร์ C-BUS หากต้องการอ่านหรือเขียนหลายรายการให้กดปุ่ม “Wr all” หรือ “Rd all” กดปุ่ม “Clear all” เพื่อรีเซ็ตตาราง กดปุ่ม “Clear data” เพื่อรีเซ็ตช่องแก้ไขข้อมูล กดปุ่ม “Lock” เพื่อป้องการการแก้ไขโดยไม่ตั้งใจ กดปุ่ม “Save Config...” เพื่อบันทึกตารางปัจจุบัน “Open Config...” เพื่อโหลดตารางที่บันทึกไว้ก่อนหน้า ซึ่ง C - BUS Control Extended Tab แสดงในภาพที่ 2.21



ภาพที่ 2.21 C-BUS Control Extended Tab

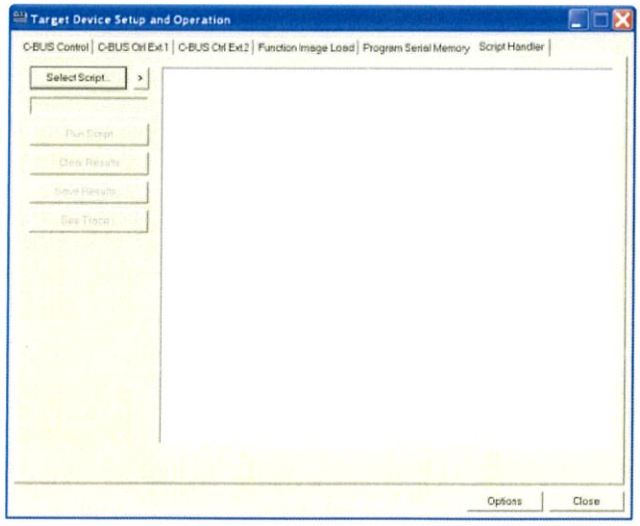
3) Function Image™ Load Tab มีฟังก์ชันโหลดและเปิดใช้งานฟังก์ชันสำหรับ FirmASIC

4) Serial Memory Programming Tab สำหรับเขียนโปรแกรมหน่วยความจำแบบอนุกรมสำหรับ FirmASIC เพื่อจัดเก็บข้อมูล Function Image แสดงในภาพที่ 2.22



ภาพที่ 2.22 Serial Memory Programming Tab

5) Script Handler Tab สามารถใช้สคริปต์ได้ โดยไฟล์เหล่านี้เป็นไฟล์ข้อความธรรมดาหรือไฟล์ .pes แต่จะทำงานบนไมโครโปรเซสเซอร์ โดยใช้ภาษาสคริปต์เฉพาะ ให้เลือกสคริปต์โดยกดปุ่ม “Select Script” โดยหน้าต่างผลลัพธ์แสดงค่าที่ส่งกลับโดยสคริปต์เมื่อกดปุ่ม “Run Script” ผลลัพธ์เหล่านี้สามารถบันทึกลงในไฟล์ข้อความโดยกดปุ่ม “Save Results” หรือต้องการล้างผลลัพธ์โดยกดปุ่ม “Clear Results” แสดงดังภาพที่ 2.23



ภาพที่ 2.23 Script Handler Tab

2.6.2 Script Language Reference [14]

ภาษาสคริปต์นี้ได้รับการพัฒนาขึ้นสำหรับการประเมิน ICs รุ่นใหม่ของ CML ซึ่งรวมถึง FirmASIC®-based ซึ่งช่วยลดความซับซ้อนในการประมวลผล สคริปต์มีความเรียบง่ายเพื่อให้ผู้ใช้สามารถอ่านและเขียนพอร์ต C-BUS ได้ถึง 2 พอร์ตบน PE0002 ซึ่งมีประสิทธิภาพในการทำงานเรียลไทม์ที่ดีขึ้นกว่าเมื่อรันสคริปต์บนพีซีโฮสต์ หน่วยความจำบน PE0002 แบ่งออกเป็นหลายส่วน ในขณะดำเนินการสคริปต์ทั้งหมดจะถูกรวบรวมและดาวน์โหลดลงใน "Command Buffer" ของ PE0002 “Data Buffer” ขนาด 61440 word ถ้าบัฟเฟอร์คำสั่งหรือบัฟเฟอร์ข้อมูลเกินกำหนดจะยกเลิกสคริปต์และแสดงข้อความ “Data out of range” ในหน้าต่างบันทึกเวลารันไทม์ โดยมีรูปแบบคำสั่งดังนี้

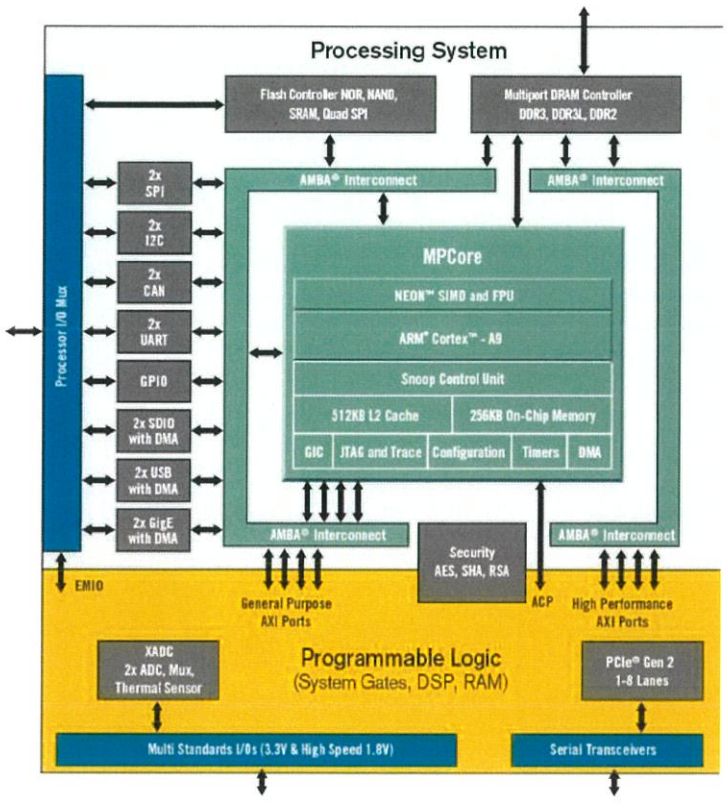
- คำสั่ง filew ข้อมูลจะถูกโอนย้ายจาก PE0002 ไปยังเครื่องโฮสต์ผ่านบัฟเฟอร์ถาวร (FIFO
- คำสั่ง filer ข้อมูลจะถูกโอนย้ายจากเครื่องโฮสต์ไปยัง PE0002 ผ่านบัฟเฟอร์ (FIFO) "File Buffer" ขนาด 294512 word
- คำสั่ง fopenw เปิดหรือสร้างไฟล์บนเครื่องโฮสต์ที่จะใช้เพื่อรับข้อมูลจากสคริปต์ PE0002 ทุกครั้งที่พบคำสั่ง filew จะมีการถ่ายโอนข้อมูล 16 บิตจากสคริปต์ไปยังบัฟเฟอร์
- คำสั่ง fopenr เปิดไฟล์บนเครื่องโฮสต์ที่จะใช้ในการส่งข้อมูลไปยังสคริปต์ PE0002

2.7 FPGA Zynq7000

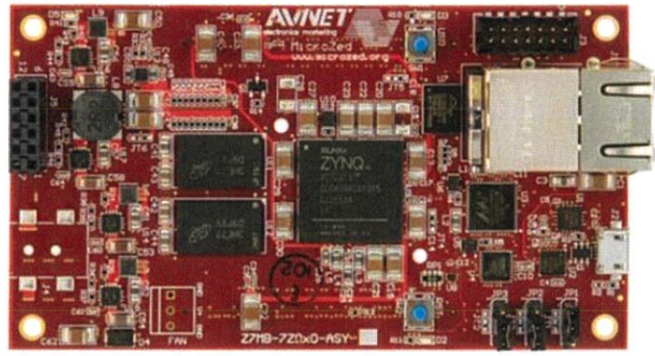
FPGA (Field Programmable Gate Array) คือ อุปกรณ์สารกึ่งตัวนำชนิดโปรแกรมได้ที่มีโครงข่ายการเชื่อมต่อภายในแบบแมตริกซ์ โครงสร้างภายในของ FPGA นั้นสามารถโปรแกรมให้มีหน้าที่การทำงานเหมือนลอจิกเกตพื้นฐาน เช่น AND, OR, XOR, NOT หรือรวมกันหลายๆ ชนิด (combinational logic) เพื่อให้ทำหน้าที่ที่มีความซับซ้อนเพิ่มขึ้น เช่น decoders หรือฟังก์ชันทางคณิตศาสตร์ ใน FPGAs ทั่วไปนอกจากจะประกอบด้วยส่วนของวงจรลอจิกแบบโปรแกรมได้แล้ว จะยังมีบล็อกของหน่วยความจำ ซึ่งอาจจะสร้างด้วยฟลิบฟลอปอย่างง่าย หรือใช้พื้นที่ของสารกึ่งตัวนำสร้างเป็นหน่วยความจำจริงๆ อยู่ภายในก็ได้ [15]

ตระกูล Zynq®-7000 ใช้สถาปัตยกรรม Xilinx All Programmable SoC ผลิตภัณฑ์เหล่านี้รวมระบบประมวลผลแบบ ARM® Cortex ™ -A9 (PS) และลินุกซ์ ซีพียู ARM Cortex-A9 เป็นหัวใจสำคัญของ PS รวมถึงหน่วยความจำบนชิพอินเทอร์เฟสหน่วยความจำภายนอกและชุดอินเทอร์เฟสการเชื่อมต่ออุปกรณ์ต่อพ่วงมากมาย [16]

MicroZed สนับสนุนรูปแบบการใช้งานหลายรูปแบบสำหรับการสำรวจ, การสร้างต้นแบบและการใช้ระบบบนโมดูล (SOM) ช่วยให้นักออกแบบสามารถเริ่มต้นด้วยแพลตฟอร์มเดียวและเติบโตไปพร้อมๆ กับความคืบหน้าในกระบวนการพัฒนาและเรียนรู้ SoC โดยมีบล็อกไดอะแกรมของ FPGA Zynq7000 ดังภาพที่ 2.24



ภาพที่ 2.24 บล็อกไดอะแกรมของ FPGA Zynq7000 [18]



ภาพที่ 2.25 FPGA Zynq7000 [16]

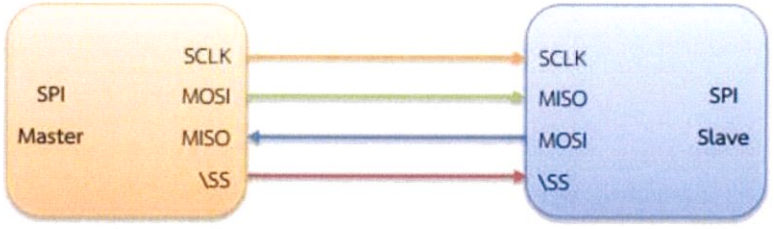
ภาพที่ 2.25 แสดง FPGA Zynq7000 มีรายละเอียดดังนี้ [16]

- Zynq-7000 XC7Z010-1CLG400C All Programmable SoC
- 1 GB of DDR3 SDRAM
- 128 Mb of QSPI Flash
- Micro SD card interface
- 10/100/1000 Ethernet
- USB 2.0
- USB-UART
- User I/O (via dual board-to-board connectors)
- 100 User I/O (50 per connector)

- Configurable as up to 48 LVDS pairs or 100 single-ended I/O
- 2×6 Digilent Pmod compatible interface providing 8 PS MIO connections
- Xilinx PC4 JTAG configuration port
- PS JTAG pins accessible via Pmod
- 33.33 MHz oscillator
- User LED and push switch

2.8 Serial Peripheral Interface (SPI)

เป็นวิธีการสื่อสารอนุกรมแบบ Synchronous อีกรูปแบบหนึ่ง ซึ่งทำงานในรูปแบบที่ให้อุปกรณ์ตัวหนึ่งทำหน้าที่เป็น Master ในขณะที่อีกตัวหนึ่งทำหน้าที่เป็น Slave และสามารถส่งข้อมูลในโหมด Full-duplex นั่นหมายความว่า สัญญาณสามารถส่งหากันได้ระหว่าง Master และ Slave ได้อย่างต่อเนื่อง รูปแบบข้อมูลการสื่อสารหรือ Protocol ของแบบ SPI นี้ ไม่ได้มาตรฐานกำหนดตายตัว ว่าข้อมูลที่ส่งหากันต้องอยู่ในรูปแบบหรือ Format แบบไหน เป็นการคิด Protocol การสื่อสารกันเอาเอง หรือดูจาก Datasheet ของอุปกรณ์ สามารถรับส่งข้อมูลได้ถูกต้องแม่นยำกว่า UART (Universal Asynchronous Receiver Transmitter) นิยมใช้ในการรับส่งข้อมูลระหว่าง Microcontroller กับอุปกรณ์ต่อพ่วงอื่นๆ เช่น Serial EEPROMs, Shift Register, Sensor, Display Driver, A/D Converters และ SD CARD เป็นต้น [19]

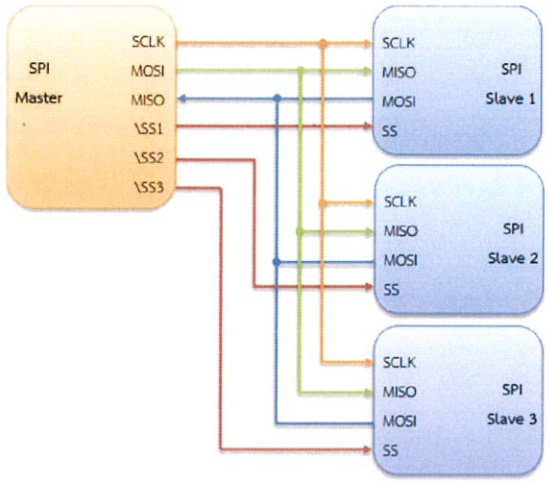


ภาพที่ 2.26 เชื่อมต่อการสื่อสารแบบ SPI ระหว่างอุปกรณ์ Master และ Slave [19]

ภาพที่ 2.26 แสดงการเชื่อมต่อการสื่อสารแบบ SPI ระหว่างอุปกรณ์ Master และ Slave โดยมีสายสัญญาณทั้งหมด 4 เส้น ประกอบด้วย [20]

- SCLK (Serial Clock) ใช้ส่งสัญญาณนาฬิกาจากอุปกรณ์ Master ไปยังอุปกรณ์ Slave เพื่อกำหนดจังหวะการรับส่งข้อมูล
- MOSI (Master Out Slave In) ใช้ส่งข้อมูลจากอุปกรณ์ Master ไปยังอุปกรณ์ Slave
- MISO (Master In Slave Out) ใช้รับข้อมูลจากอุปกรณ์ Slave
- SS (Slave Select) หรือ ขา CS (Chip Select) ใช้ส่งสัญญาณ Low ไปยังอุปกรณ์ Slave ที่ต้องการรับส่งข้อมูล

อุปกรณ์ Master ทำหน้าที่เป็นตัวควบคุมการสื่อสารทั้งหมด โดยควบคุมการสื่อสารตามสัญญาณนาฬิกา และสายสัญญาณ SS ตัวมาสเตอร์จะเป็นตัวที่ตัดสินใจเลือก รับ หรือ ส่งข้อมูลไปยังอุปกรณ์ Slave สัญญาณเส้น SS หรือ Slave select ในกรณี ที่มีตัว Slave มากกว่า 1 ตัว โดยการทำให้เส้น SS มีระดับสัญญาณเป็น Low เมื่อต้องการติดต่อกับ Slave ตัวใด หากต้องการติดต่อสื่อสารกับอุปกรณ์ Slave ตัวใด ก็เพียงทำให้สัญญาณ SS ของ Slave ตัวนั้น มีระดับสัญญาณเป็น Low ซึ่งแสดงในภาพที่ 2.27



ภาพที่ 2.27 เชื่อมต่อการสื่อสารแบบ SPI ระหว่างอุปกรณ์ Master และ Slave หลายตัว [19]

ข้อดีของการสื่อสารแบบ SPI คือ สามารถสื่อสารแบบ Full Duplex กล่าวคือสามารถรับและส่งข้อมูลได้พร้อมๆ กัน เพราะมีสายสัญญาณรับและส่งข้อมูลโดยเฉพาะ รูปแบบการสื่อสารของ SPI ไม่ต้องกำหนด Address เพื่อระบุอุปกรณ์ที่ต้องการสื่อสารเหมือน I2C เนื่องจากใช้สายสัญญาณ SS เป็นตัวควบคุม จึงมีอัตราการรับส่งข้อมูลสูงกว่า I2C และเหมาะสำหรับการรับส่งข้อมูลแบบต่อเนื่อง หรือ Streaming อย่างไรก็ตาม หากมีอุปกรณ์ Slave หลายตัวดังรูป การสื่อสารแบบ SPI ต้องใช้สายสัญญาณมากกว่า I2C

2.9 Inter-Integrated Circuit (I2C)

การติดต่อสื่อสารระหว่างไอซีโดยบัส I2C ได้รับการพัฒนาโดยฟิลิปส์ (Philips) ด้วยจุดมุ่งหมายหลักคือ ต้องการให้ไอซีหรือโมดูลสามารถติดต่อ สั่งงานและควบคุมภายใต้สายสัญญาณเพียง 2 เส้น โดยเส้นหนึ่งคือสายสัญญาณนาฬิกาที่ใช้กำหนดจังหวะการทำงาน การต่อร่วมกันของอุปกรณ์บนบัส I2C ทำได้ง่ายมาก เพียงต่อสายข้อมูลและสายสัญญาณนาฬิกาของอุปกรณ์แต่ละตัวขนานหรือพ่วงกันไป ส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่ออุปกรณ์แต่ละตัว จะใช้รหัสข้อมูลและการกำหนดสภาวะโลจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว

สายข้อมูลบนบัส I2C มีชื่อเรียกอย่างเป็นทางการว่าสายข้อมูลอนุกรมหรือ SDA (Serial Data Line) ส่วนสายสัญญาณนาฬิกามีชื่อเรียกว่าสายสัญญาณนาฬิกาอนุกรมหรือ SCL (Serial Clock Line)

2.9.1 คุณสมบัติโดยทั่วไปของบัส I2C

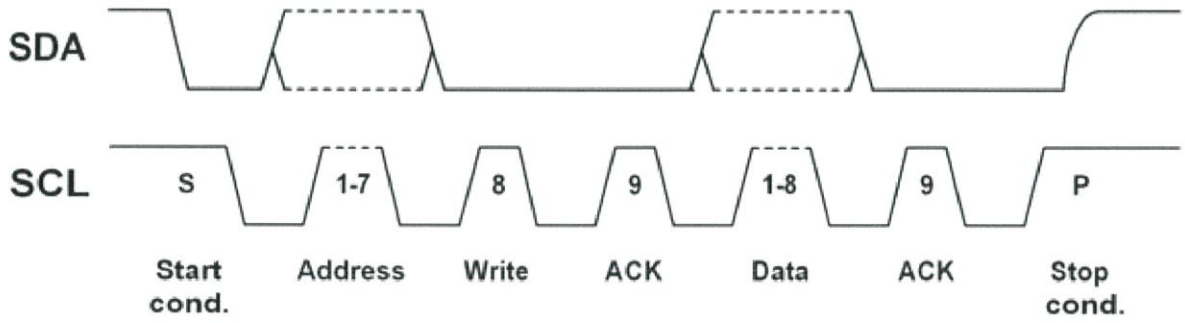
สาย SDA และ SCL เป็นสายสัญญาณ 2 ทิศทาง (bi-directional line) ต้องมีการต่อตัวต้านทานพูลอัพกับแรงดัน +5V ไว้ตลอดเวลาเพื่อให้สายมีสถานะลอจิกสูงในขณะที่ไม่มีการติดต่อใช้งาน ทั้งยังช่วยป้องกันสัญญาณรบกวนที่อาจมีเข้ามาในสายสัญญาณทั้งสอง วงจรเอาต์พุตของอุปกรณ์ที่ต่ออยู่บนบัส I2C ต้องมีลักษณะเป็นวงจรเดรนเปิด (Open-drain) หรือคอลเล็กเตอร์เปิด (Open-collector)

2.9.2 อัตราการถ่ายทอดข้อมูลบนบัส I2C

อัตราการถ่ายทอดข้อมูลบนบัส I2C สูงถึง 100 กิโลบิตต่อวินาทีในโหมดปกติและสูงถึง 400 กิโลบิตต่อวินาทีในโหมดความเร็วสูง อุปกรณ์ที่ต่อร่วมบนบัส I2C จะต้องมีความจุไฟฟ้ารวมที่เกิดขึ้นระหว่างสาย SDA และ SCL ไม่เกิน 400 pF การเข้าถึงอุปกรณ์บนบัส I2C ใช้ข้อมูลสำหรับการเข้าถึงสองค่าคือ 7 บิต (7-bit addressing) หรือ 10 บิต (10-bit addressing)

2.9.3 หลักการของบัส I2C [21]

บัส I2C ประกอบด้วยสายสัญญาณ 2 เส้นคือ SDA และ SCL อุปกรณ์ที่ต่อพ่วงบนบัสสามารถมีได้มากมาย ดังนั้นจึงต้องมีการกำหนดรูปแบบของการติดต่อบนบัสเพื่อให้ผู้ใช้งานทราบว่าขณะนี้อุปกรณ์ใดติดต่อกันอยู่และอุปกรณ์ใดเป็นตัวรับหรือส่ง โดยอุปกรณ์ที่เป็นผู้สร้างข้อมูลหรือส่งข้อมูล เรียกว่าตัวส่ง (transmitter) อุปกรณ์ที่เป็นผู้รับข้อมูลเรียกว่าตัวรับ (receiver) อุปกรณ์บนบัส I2C สามารถเป็นได้ทั้งตัวรับและส่ง บางอุปกรณ์ทำหน้าที่เป็นตัวรับอย่างเดียว จะไม่มีอุปกรณ์ใดบนบัส I2C ที่ทำหน้าที่เป็นตัวส่งอย่างเดียว อุปกรณ์ที่ทำหน้าที่ควบคุมจังหวะการติดต่อบนบัส I2C เรียกว่า มาสเตอร์ (master) อุปกรณ์ที่ถูกควบคุมหรืออุปกรณ์ที่ต่อพ่วงเข้าไปบนบัส I2C เรียกว่า สเลฟ (slave) สัญญาณ SDA และ SCL แสดงในภาพที่ 2.28



ภาพที่ 2.28 สัญญาณขา SDA และ SCL [21]

2.9.4 ข้อกำหนดสำคัญของการติดต่อบนบัส I2C

1) การถ่ายทอดข้อมูลจะเกิดขึ้นได้เมื่อบัสว่างเท่านั้น

2) ในระหว่างการถ่ายทอดข้อมูล เมื่อใดก็ตามที่สาย SCL มีสถานะลอจิกสูง สายข้อมูลต้องรักษาข้อมูลไว้ อย่าให้เกิดความเปลี่ยนแปลงเด็ดขาด มิฉะนั้นสัญญาณที่เกิดขึ้นจะได้รับการแปลความหมายเป็นสัญญาณควบคุมแทน

2.9.5 สภาวะที่เกิดขึ้นบนบัส I2C

1) บัสว่าง (Bus not busy) สภาวะนี้เกิดขึ้นเมื่อ สถานะลอจิกบนสาย SDA และ SCL มีลอจิกสูงทั้งคู่ นั้นหมายความว่า การถ่ายทอดข้อมูลสามารถเริ่มต้นขึ้นได้

2) เริ่มต้นการถ่ายทอดข้อมูล (start data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงลอจิกจากสูงไปต่ำ ในขณะที่สาย SCL มีสถานะลอจิกสูงเรียกสภาวะนี้ว่า สภาวะเริ่มต้น (START)

3) ข้อมูลดำรงอยู่บนบัส (data valid) สภาวะนี้เกิดขึ้นถัดจากสภาวะเริ่มต้น โดยสถานะลอจิกที่เกิดขึ้นบนสาย SDA ก็คือข้อมูลที่ทำการถ่ายทอด เมื่อสาย SCL มีลอจิกสูง สถานะที่สาย SDA ต้องคงที่ เพื่อให้อุปกรณ์รับข้อมูลในจังหวะนั้นว่าเป็น "0" หรือ "1" ข้อมูลอาจเกิดความเปลี่ยนแปลงได้ในขณะที่สาย SCL เป็นลอจิกต่ำ แต่เมื่อใดก็ตามที่ต้องการให้เกิดการถ่ายทอดข้อมูลอย่างสมบูรณ์สถานะลอจิกที่ขา SDA ต้องคงที่ตลอดช่วงเวลาที่สาย SCL มีสถานะลอจิกสูง หากเกิดการเปลี่ยนแปลงสถานะลอจิกในขณะที่สาย SCL มีลอจิกสูงอยู่นั้น อุปกรณ์มาสเตอร์ที่ควบคุมการถ่ายทอดข้อมูลจะแปลความหมายเป็นสภาวะหยุดหรือสภาวะเริ่มต้นก็ได้ ทำให้ข้อมูลที่ทำการถ่ายทอดเกิดความผิดพลาดเกิดขึ้น

4) รับรู้ข้อมูล (acknowledge) เกิดขึ้นหลังจากการถ่ายทอดข้อมูลจากตัวส่งมายังตัวรับเกิดขึ้นอย่างสมบูรณ์ โดยตัวส่งจะทำการส่งข้อมูลมา 1 บิตเรียกว่า บิตรับรู้ (acknowledge bit) มีสถานะเป็นลอจิกสูง หลังการส่งข้อมูลมาครบถ้วน ส่วนอุปกรณ์มาสเตอร์จะทำการส่งสัญญาณรับรู้พิเศษซึ่งสัมพันธ์กับสัญญาณนาฬิกา อุปกรณ์สเลฟที่ถูกอ้างอิงในการติดต่อหรือกำลังติดต่ออยู่ในขณะนั้นก็จะกำเนิดบิตรับรู้ที่มีสถานะลอจิกต่ำเพื่อตอบสนองให้ทราบว่า ได้รับข้อมูลเรียบร้อยแล้ว

5) หยุดการถ่ายทอดข้อมูล (stop data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากต่ำไปสูง ในขณะที่สาย SCL มีสถานะลอจิกสูงเรียกสภาวะที่เกิดขึ้นนี้ว่าสภาวะหยุด (STOP)

2.9.6 การทำงานบนบัส I2C

เริ่มต้นด้วยการเข้าถึงอุปกรณ์เสียก่อน โดยการเข้าถึงอุปกรณ์บนบัส I2C นั้นจะใช้การเข้าถึงแบบ 7 หรือ 10 บิต หลังจากที่ติดต่ออุปกรณ์แต่ละตัวเรียบร้อยแล้ว จะเริ่มต้นการถ่ายทอดข้อมูลต่อไป

2.9.6.1 การเข้าถึงแบบ 7 บิต (7-bit addressing)

ข้อมูลไบต์แรกที่เกิดขึ้นหลังจากสภาวะเริ่มต้นคือข้อมูลที่ใช้อ้างอุปกรณ์ที่ต้องการติดต่อโดยมีรูปแบบแสดงในรูป ใน 7 บิตบนรวมทั้งบิต LSB ด้วยจะเป็นข้อมูลแอดเดรสของอุปกรณ์สเลฟที่ต้องการติดต่อ โดยแบ่งเป็นบิตกำหนดแอดเดรสคงที่ (fix address bit) จำนวน 4 บิต ซึ่งข้อมูลนี้อุปกรณ์แต่ละตัวจะถูกกำหนดมาจากผู้ผลิตไม่สามารถเปลี่ยนแปลงแก้ไขได้ถัดมาอีก 3 บิตเป็นบิตกำหนดแอดเดรสที่สามารถโปรแกรมได้ (programmable address bit) โดยผู้ใช้งานต้องกำหนดสถานะลอจิกให้แก่ขา A0-A2 ของอุปกรณ์ที่มีการเชื่อมต่อแบบบัส I2C ส่วนในบิต LSB ที่ใช้กำหนดการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟตัวนั้นๆ หากบิต LSB เป็น "0" หมายถึงต้องการเขียนข้อมูลไปยังอุปกรณ์นั้น ถ้าเป็น "1" จะเป็นการอ่านข้อมูลจากอุปกรณ์สเลฟ

ข้อมูลในไบต์ต่อมาคือข้อมูลควบคุม (control byte) ในอุปกรณ์แต่ละตัวจะมีการกำหนดข้อมูลควบคุมที่แตกต่างกันไป ยกตัวอย่างเช่นไอซีเมมโมรีของทีวีตระกูล 24Cxx จะต้องส่งข้อมูลแอดเดรสของหน่วยความจำก่อนที่จะทำการส่งข้อมูลไป

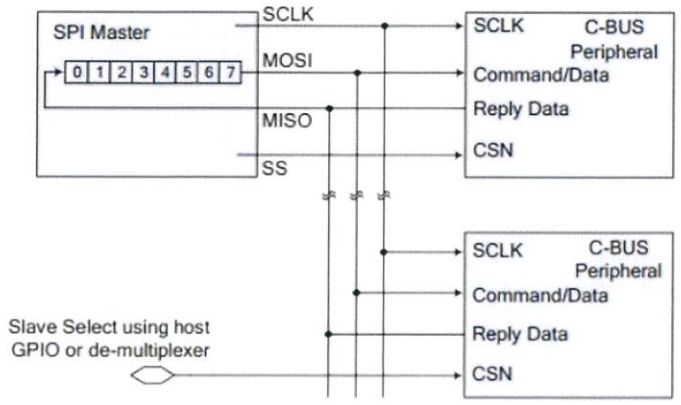
ข้อมูลในไบต์ต่อมาคือ ข้อมูลที่ทำการถ่ายทอดจริง (data) หลังจากการถ่ายทอดข้อมูลในแต่ละไบต์ อุปกรณ์สเลฟที่ได้รับการติดต่อต้องส่งสัญญาณรับรู้ตอบกลับมาด้วยทุกครั้ง

2.9.6.2 การเข้าถึงแบบ 10 บิต (10-bit addressing)

จะมีข้อมูลเพิ่มเติมขึ้นมาเล็กน้อย โดยในไบต์แรกหลักจากสภาวะเริ่มต้น ต้องกำหนดให้ 5 บิตบนมีข้อมูลเป็น 11110 ส่วนอีก 2 บิตถัดมาเป็นบิตแอดเดรสของอุปกรณ์ที่ต้องการติดต่อ ในบิต LSB ของข้อมูลไบต์แรกยังคงเป็นการกำหนดว่า ต้องการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟตัวที่ต้องการติดต่อด้วย ข้อมูลไบต์ต่อมาเป็นข้อมูลแอดเดรสในไบต์ที่ 2 ของอุปกรณ์ที่ต้องการติดต่อด้วย ข้อมูลไบต์ถัดไปจึงเป็นข้อมูลควบคุม ข้อมูลหลังจากนี้ก็จะเป็นข้อมูลจริงที่ใช้ในการติดต่อเช่นเดียวกันกับการเข้าถึงแบบ 7 บิต หลังจากถ่ายทอดข้อมูลครบทุกไบต์ ต้องมีสภาวะรับรู้เกิดขึ้น เพื่อให้กระบวนการถ่ายทอดข้อมูลสามารถดำเนินต่อไปได้

2.10 C–BUS [22]

C–BUS คล้ายกับ SPI ซึ่งเป็นการสื่อสารอนุกรมแบบ Synchronous รูปแบบหนึ่ง สามารถเชื่อมต่อกับ SPI Master ได้โดยตรง โดยวงจรรวม (Integrated Circuit) จะเป็น Slaves โดยสามารถเชื่อมต่อกับ SS เนื่องจากการส่งข้อมูลเหมือนกับ MISO และข้อมูลจะตอบกลับบน C–BUS เป็นสามสถานะ แสดงการเชื่อมต่อดังภาพที่ 2.29



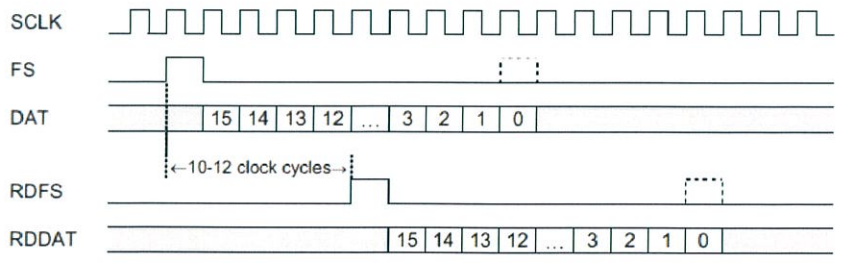
ภาพที่ 2.29 การเชื่อมต่อโดยทั่วไปสำหรับหนึ่งหรือมากกว่าหนึ่ง C–BUS slaves [11]

ภาพที่ 2.29 แสดงการเชื่อมต่อการสื่อสารแบบ C-BUS ระหว่างอุปกรณ์ Master และ Slave โดยมีสายสัญญาณทั้งหมด 4 เส้น หรือ Four Wire ประกอบด้วย

- SCLK (Serial Clock) ใช้ส่งสัญญาณนาฬิกาจากอุปกรณ์ Master ไปยังอุปกรณ์ Slave เพื่อกำหนดจังหวะการรับส่งข้อมูล
- Command / Data ใช้ส่งข้อมูลจากอุปกรณ์ Slave ไปยังอุปกรณ์ Master
- Reply Data ใช้รับข้อมูลจากอุปกรณ์ Master
- CSN ใช้ส่งสัญญาณ Low ไปยังอุปกรณ์ Master เพื่อเปิดใช้งาน SPI

2.11 Fast Serial Bus (FSB) [8]

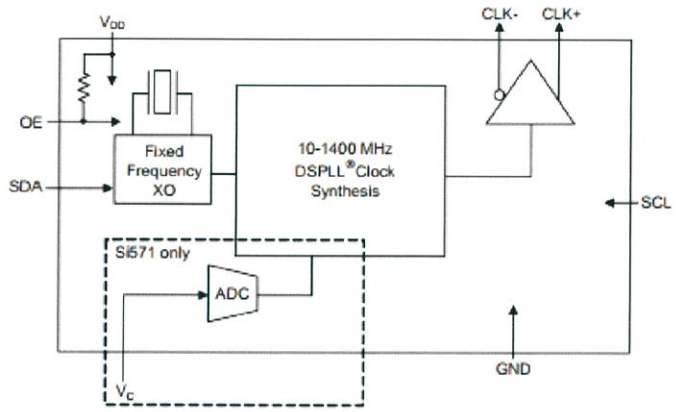
FSB เป็นการสื่อสารแบบอนุกรมซึ่งคิดค้นโดยบริษัท CML เพื่อใช้ในอุปกรณ์เฉพาะ จะคล้ายกับ C-BUS แต่สามารถรับส่งข้อมูลได้เร็วกว่า โดยการทำงานของ FSB แสดงดังภาพที่ 2.28



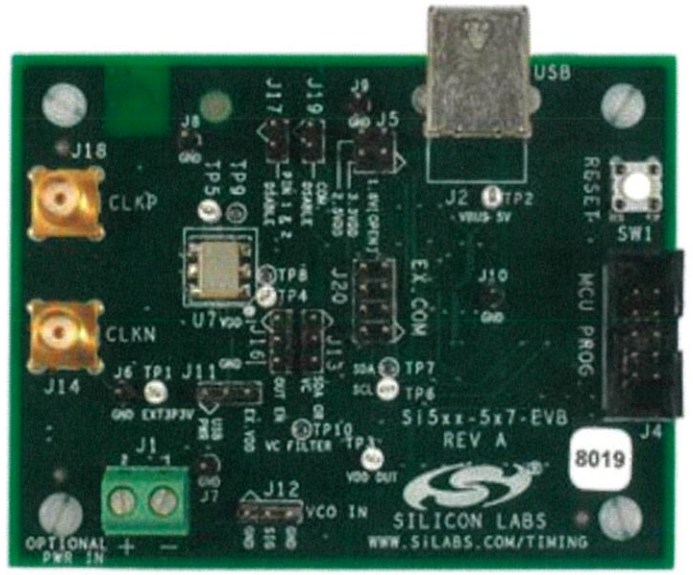
ภาพที่ 2.30 การทำงานของ FSB [8]

2.12 Si570 [23]

Si570 XO / Si571 VCXO ใช้วงจร DSPLL® ขั้นสูงของ Silicon Laboratories เพื่อให้นาฬิกามีการตอบสนองในทุกความถี่ Si570 / Si571 สามารถใช้งานได้กับความถี่เอาต์พุตตั้งแต่ 10 ถึง 945 MHz ด้วยความละเอียด <1 ppb โดยเชื่อมต่อผ่านอินเทอร์เฟซแบบอนุกรม I2C โดย IC นี้มีความเสถียร ความเชื่อถือสูง โดยมีบล็อกไดอะแกรมของ IC Si570 ดังภาพที่ 2.31 และภาพที่ 2.32 บอร์ด Si570-PROG-EV



ภาพที่ 2.31 บล็อกไดอะแกรมของ IC Si570 [23]



ภาพที่ 2.32 Si570-PROG-EVB [23]

ภาพที่ 2.33 วงจรรวม Si570 [23]

ภาพที่ 2.33 แสดงวงจรรวม Si570 มีรายละเอียดดังนี้

- เอาท์พุทความถี่ตั้งแต่ 10 ถึง 945 MHz และเลือกความถี่เป็น 1.4 GHz
- เชื่อมต่อผ่าน I2C
- มีความเสถียรของความถี่ที่ดีกว่า 3 เท่าของ oscillator SAW
- LVPECL, CMOS, LVDS และ CML outputs
- ไฟเลี้ยง 1.8, 2.5 หรือ 3.3 โวลต์

Si570 XO และ Si571 VCXO เป็นเครื่องกำเนิดสัญญาณ สามารถตั้งโปรแกรมเพื่อกำหนดความถี่เอาท์พุท 10 MHz ถึง 1.4 GHz โดยสามารถกำหนดความถี่เอาท์พุท (f_{out})

โดย HSDIV มีค่าเท่ากับ 4, 5, 6, 7, 9 หรือ 11

N1 มีค่าเท่ากับ 1 หรือเลขคู่จนถึง 128

f_0 มีค่าเท่ากับ 156.28 MHz

RFREQ = 0x2BBD25236 = 43.7502734363

สูตรการคำนวณความถี่เอาท์พุท fxtal = $\frac{f_0 \, x \, HSDIV \, x \, N1}{RFREQ}$

$$Fdco = f_1 \, x \, HSDIV \, x \, N1$$

$$RFREQ = \frac{fdco}{fxtal}$$

2.13 ลินุกซ์ [24]

ลินุกซ์ถือกำเนิดขึ้นในฟินแลนด์ ปี คศ. 1980 โดยลีนุส โทรวัลด์ส (Linus Trovalds) นักศึกษาภาควิชาวิทยาการคอมพิวเตอร์ (Computer Science) ในมหาวิทยาลัยเฮลซิงกิ ลีนุส เห็นว่าระบบมินิกซ์ (Minix) ที่เป็นระบบยูนิกซ์บนพีซีในขณะนั้น ซึ่งทำการพัฒนาโดย ศ.แอนดรูว์ ทาเนนบาวม์ (Andrew S. Tanenbaum) ได้พัฒนาระบบยูนิกซ์ขึ้นมา โดยจุดประสงค์อีกประการ คือต้องการทำความเข้าใจในวิชาระบบปฏิบัติการคอมพิวเตอร์ด้วยเมื่อเขาเริ่มพัฒนาลินุกซ์ไปช่วงหนึ่งแล้ว เขาก็ได้ทำการชักชวนให้นักพัฒนาโปรแกรมอื่นๆ มาช่วยทำการพัฒนาลินุกซ์ ซึ่งความร่วมมือส่วนใหญ่ก็จะเป็นความร่วมมือผ่านทางอินเทอร์เน็ต ลีนุซจะเป็นคนรวบรวมโปรแกรมที่ผู้พัฒนาต่างๆ ได้ร่วมกันทำการพัฒนาขึ้นมาและแจกจ่ายให้ทดลองใช้เพื่อทดสอบหาข้อบกพร่อง ที่น่าสนใจก็คืองานต่างๆ เหล่านี้ผู้คนทั้งหมดต่างก็ทำงานโดยไม่คิดค่าตอบแทน และทำงานผ่านอินเทอร์เน็ตทั้งหมด

ลินุกซ์ เป็นระบบปฏิบัติการเช่นเดียวกับ ดอส ไมโครซอฟต์วินโดวส์ หรือยูนิกซ์ โดยลินุกซ์นั้นจัดว่าเป็นระบบปฏิบัติการยูนิกซ์ประเภทหนึ่ง โดยลินุกซ์ได้รับความนิยมจากความสามารถของตัวระบบปฏิบัติการและโปรแกรมประยุกต์ที่ทำงานบนระบบลินุกซ์ โดยเฉพาะอย่างยิ่งโปรแกรมในตระกูลของ GNU (GNU's Not UNIX) และสิ่งที่สำคัญที่สุดก็คือระบบลินุกซ์เป็นระบบปฏิบัติการประเภทฟรีแวร์ (Free Ware) คือไม่เสียค่าใช้จ่ายในการซื้อโปรแกรม

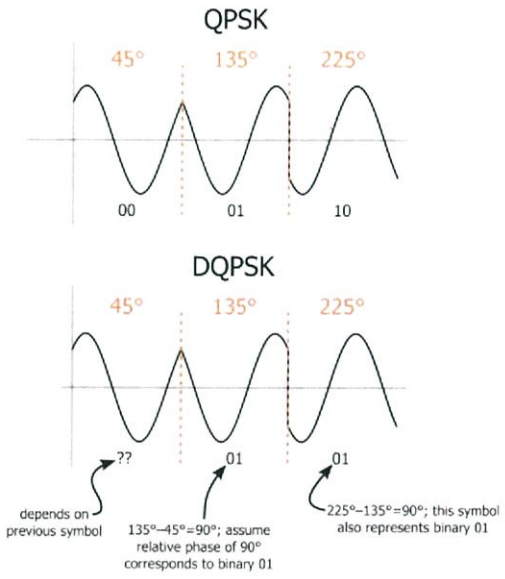
ระบบลินุกซ์ตั้งแต่เวอร์ชั่น 4 นั้น สามารถทำงานได้บนซีพียูทั้ง 3 ตระกูล คือบนซีพียูของอิลเทล (PC Intel) ดิจิทัลอัลฟาคอมพิวเตอร์ (Digital Alpha Computer) และซันสปาร์ค (SUN SPARC) เนื่องจากใช้เทคโนโลยีที่เรียกว่า RPM (Red Hat Package Management) ถึงแม้ว่าในขณะนี้ลินุกซ์ยังไม่สามารถแทนที่ไมโครซอฟต์ วินโดวส์ บนพีซีหรือแมคโอเอส (Mac OS) ได้ทั้งหมดก็ตาม แต่ผู้ใช้จำนวนไม่น้อยที่หันมาใช้และช่วยพัฒนาโปรแกรมประยุกต์บนลินุกซ์กัน และเรื่องของการดูแลระบบลินุกซ์นั้น ภายในระบบลินุกซ์เองมีเครื่องมือช่วยสำหรับดำเนินการให้สะดวกยิ่งขึ้น

ปัจจุบันได้มีการนำระบบปฏิบัติการลินุกซ์ไปประยุกต์เป็นระบบปฏิบัติการสำหรับงานด้านต่างๆ เช่นงานด้านการคำนวณทางวิทยาศาสตร์ใช้เป็นสถานีงาน สถานีบริการ อินเทอร์เน็ต อินทราเน็ต หรือใช้ในการเรียนการสอนและการทำวิจัยทางคอมพิวเตอร์ใช้พัฒนาโปรแกรมเนื่องจาก มีเครื่องมือมากมาย เช่น โปรแกรมภาษาซี (C) ซีพลัสพลัส (C++) ปาสคาล (Pascal) ฟอร์แทรน (Fortran) ลิสป์ (Lisp) โปรล็อก (Prolog) เอดา (ADA) มีภาษาสคริปต์ เช่น เชลล์ (Shell) บาสช์เชลล์ (Bash Shell) ซีเชลล์ (C Shell) คอร์นเชลล์ (Korn Shell) เพิร์ล (Perl) ไพธอน (python) TCL/TK

2.14 การมอดูเลตแบบ Differential Quadrature Phase Shift Keying (DQPSK) [25]

การมอดูเลตสัญญาณ คือการส่งสัญญาณเสียงหรือข้อมูลผ่านช่องทางการสื่อสารโดยอาศัยพลังงานไฟฟ้าช่วยพาสัญญาณเหล่านั้นให้ย้ายจากที่หนึ่งไปยังอีกที่หนึ่ง กระบวนการหรือขั้นตอนในการเพิ่มพลังงานไฟฟ้าดังกล่าวเราเรียกว่าการมอดูเลต (Modulation) พลังงานไฟฟ้าซึ่งมีความถี่สูงและคงที่รวมทั้งมีแอมปลิจูด (ขนาด) สูงด้วยนั้นจะเรียกว่าสัญญาณคลื่นพาห์ (Signal Carrier) อุปกรณ์สำหรับมอดูเลตสัญญาณ (Modulator) จะสร้างสัญญาณคลื่นพาห์และรวมเข้ากับสัญญาณข้อมูลเพื่อให้สัญญาณมีความแรงพอที่จะส่งผ่านสื่อกลางไปยังอีกจุดหนึ่งที่อยู่ไกลออกไปได้

Differential phase shift keying (DPSK) เป็นรูปแบบทั่วไปของการปรับเฟสที่ถ่ายทอดข้อมูลโดยการเปลี่ยนเฟสของคลื่นผู้ให้บริการ เช่นเดียวกับ BPSK และ QPSK คือ ค่าของขนาดและความถี่ของคลื่นพาห์จะไม่มีการเปลี่ยนแปลง แต่ที่เปลี่ยนคือ เฟสของสัญญาณ เมื่อมีการเปลี่ยนสภาวะจากบิต 1 เป็น 0 หรือเปลี่ยนจาก 0 เป็น 1 เฟสของคลื่นจะเปลี่ยนไป 180° ค่าเฟส QPSK สี่แบบมาตรฐานคือ 45°, 135°, 225°และ 315° ตัวเลือกเฟส DQPSK จะกลายเป็น 0°, 90°, -90° และ 180° (หรือเทียบเท่า -180°)



ภาพที่ 2.34 การมอดูเลตแบบ DQPSK [27]

บทที่ 3

วิธีดำเนินการวิจัย

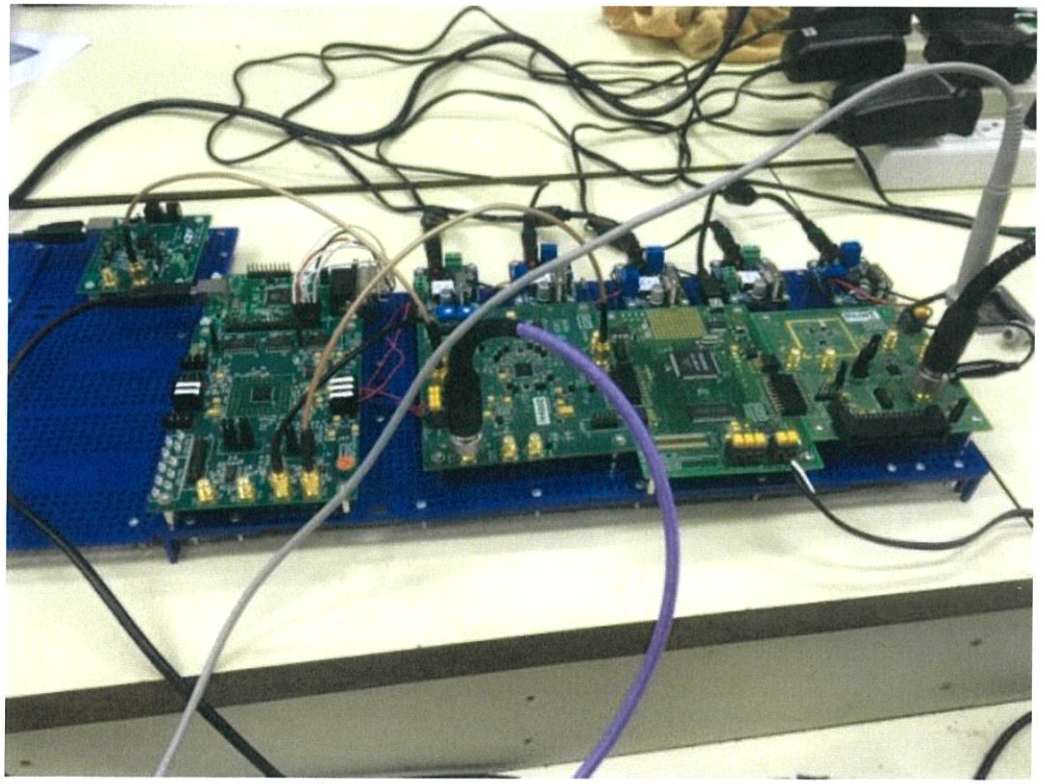
อุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA มีจุดประสงค์ คือ เพื่อช่วยให้ทราบถึงแนวทางการสื่อสารในอนาคตในรูปแบบการรับส่งข้อมูลแบบดิจิทัล โดยการศึกษาและออกแบบอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA นี้ ได้ถูกออกแบบการทำงานของระบบมีดังต่อไปนี้

3.1 ศึกษาและทดลองอุปกรณ์ต้นแบบ

อุปกรณ์ต้นแบบนี้เป็นอุปกรณ์เพื่อการรับส่งข้อมูลบนมาตรฐาน TETRA โดยใช้ Evaluation board ของ CML Microcircuit (CML) ประกอบด้วย 4 ชุดอุปกรณ์ดังตารางที่ 3.1 และมีการเชื่อมต่อกันดังภาพที่ 3.1

ตารางที่ 3.1 ชุดอุปกรณ์ทดลอง

บอร์ด	จุดประสงค์
PE0002	เพื่อสั่งการและควบคุมบอร์ด EV9810 ผ่านทาง C-BUS
EV9810	ประมวลผลสัญญาณเบสแบนด์
EV9980	วงจรส่ง
EV9942	วงจรรับ



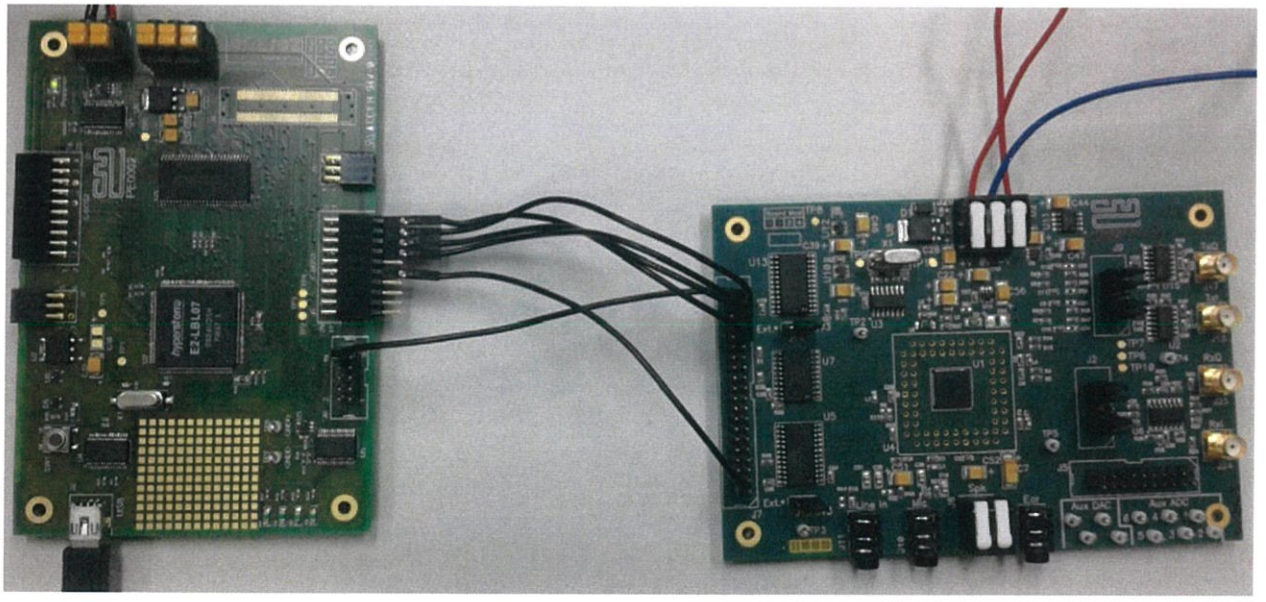
ภาพที่ 3.1 การเชื่อมต่อของชุด Evaluation Kit

3.1.1 การเชื่อมต่อ PE0002 และ EV9810

บอร์ด EV9810 เป็นบอร์ดสำหรับประมวลผลสัญญาณเบสแบนด์โดยจะสั่งการและควบคุมบอร์ดดังกล่าวด้วย PE0002 ผ่าน C-BUS ซึ่งก่อนการเชื่อมต่อบอร์ดทั้งสองจะต้องมีการตั้งค่ากำลังไฟที่จ่ายให้กับบอร์ดอย่างถูกต้องก่อน บอร์ด PE0002 ใช้ไฟฟ้ากระแสตรง 5 V และบอร์ด EV9810 ใช้ไฟฟ้ากระแสตรง 8 V ในการเชื่อมต่อบอร์ด PE0002 และบอร์ด EV9810 จะเชื่อมต่อกันโดยตรงผ่าน 4 ช่อง C-BUS Pins ภายหลังการเชื่อมต่อจะต้องมีการตั้งค่า C-BUS clock เป็น 2 MHz และเปิดการใช้งาน CBUSEN ของ EV9810 โดยรายละเอียดการเชื่อมต่อ C-BUS Pins ของ PE0002 และ EV9810 แสดงดังตารางที่ 3.2

ตารางที่ 3.2 รายละเอียดการเชื่อมต่อ C-BUS Pins ของ PE0002 และ EV9810

EV9810 (connector/pin)	รายละเอียด	PE0002 (connector/pin)	รายละเอียด
J7/40	GND	J3/12	GND
J7/39	CBUSEN	J6/10	3.3V power supply
J7/37	CDATA	J3/4	CDATA
J7/35	RDATA	J3/8	RDATA
J7/33	CCLK	J3/6	SCLKBUS
J7/31	CSN	J3/2	CSN



ภาพที่ 3.2 การเชื่อมต่อระหว่าง PE0002 และ EV9810

3.1.2 การควบคุม EV9810 ร่วมกับ PE0002

PE0002 ถูกควบคุมด้วยคอมพิวเตอร์ เชื่อมต่อผ่าน USB และสั่งการผ่านโปรแกรมซึ่งสามารถสั่งการด้วยส่วนต่อประสานกราฟิกผู้ใช้ สำหรับถ่ายโอนข้อมูลรีจิสเตอร์ไปยังอุปกรณ์ที่เชื่อมต่อผ่านทาง C-BUS และสามารถสั่งการด้วยการปฏิบัติตามสคริปต์ที่ถูกเขียนขึ้นตามภาษาสคริปต์โดย CML ด้วยภาษาสคริปต์นี้ทำให้ผู้ใช้สามารถควบคุมบอร์ด PE0002 และทำการถ่ายโอนข้อมูลรีจิสเตอร์ผ่านทาง C-BUS ได้

3.1.3 การทดสอบ Digital Analog Converter ของ EV9810

ตัวแปลงดิจิทัลเป็นแอนาล็อกของ CMX981 สามารถควบคุมโดยการเขียนลงรีจิสเตอร์ DAC โดยตรงหรือใส่ Aux RAM ด้วยข้อมูลและทำให้สามารถอ่านข้อมูลจากแรมได้ ในการทดลองนี้ ข้อมูลจะถูกเขียนโดยตรงไปยังรีจิสเตอร์ DAC และถูกแปลงเป็นสัญญาณแอนาล็อกโดยตรง

3.1.4 การทดลองส่งข้อมูลบน EV9810

ทดสอบการส่งข้อมูลบน EV9810 ส่วนประมวลผล โดยการเชื่อมต่อกับ PE0002 ซึ่งเป็นตัวควบคุมที่มีการเชื่อมต่อกับคอมพิวเตอร์ผ่าน USB และสั่งการผ่านส่วนต่อประสานกราฟิกผู้ใช้ โดย EV9810 จะมีการส่งข้อมูลได้ทั้งหมด 2 แบบ 1) โหมด C-BUS อ่านและเขียนคำสั่งโดยใช้ขา CCLK, CDATA และ CSN ข้อมูลที่อ่านจะถูกส่งกลับมาที่ขา RDATA 2) โหมด FSB อ่านและเขียนคำสั่งโดยใช้ขา CmdFS1 และ CmdDat1 ข้อมูลจะถูกส่งกลับมาที่ขา CmdRdFS1 และ CmdRdDat1

3.2 ศึกษาและทดลองอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA

3.2.1 การทดสอบการสื่อสาร

ทดสอบการสื่อสารกับวงจรรวมผ่านโปรแกรม PuTTY ด้วยภาษา C โดยจะทดลองเขียนค่าไปที่รีจิสเตอร์ของวงจรรวมแต่ละตัว และจะเรียกอ่านค่าที่รีจิสเตอร์ตัวนั้น เพื่อตรวจสอบว่าค่าที่อ่านได้ตรงกับค่าที่ส่งไปหรือไม่

3.2.2 ทดสอบการสื่อสารกับ Si570

ในการส่งข้อมูลออกไปยังตัวกลาง จำเป็นต้องนำข้อมูลไปคูณกับสัญญาณคลื่นพาห์ซึ่งถูกสร้างโดย Si570 เพื่อให้สามารถส่งข้อมูลไปในช่องความถี่ที่ต้องการได้ การสร้างสัญญาณคลื่นพาห์ที่ความถี่ต่างๆ สามารถทำได้โดยการเปิดโปรแกรมบนบอร์ด MicroZed และใส่ค่า Crystal Frequency Multiplication Ratio ซึ่งถูกคำนวณตามความถี่ที่ต้องการสร้างสัญญาณ เป็นอินพุตของโปรแกรม จากนั้น Si570 จะสร้างสัญญาณที่ความถี่นั้นๆ

3.3 เครื่องมือที่ใช้ในการทดลอง

3.3.1 CMX981 Advanced Digital Radio Baseband Processor

3.3.2 CMX998 Cartesian Feed-back Loop Transmitter

3.3.3 CMX994 Direct Conversion Receiver

3.3.4 PE0002 Evaluation Kit Interface Card

บทที่ 4

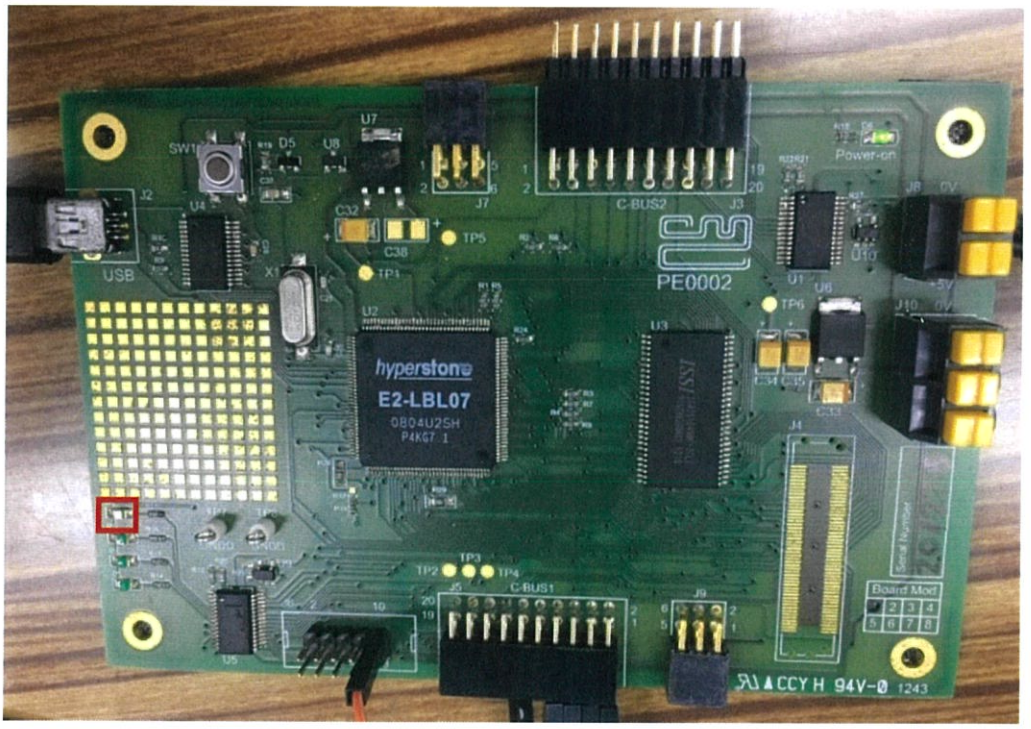
ผลการวิจัย

สำหรับการทดสอบอุปกรณ์ชุดทดลองและอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA ได้ทำการจัดเก็บผลการทำงานของระบบ โดยแบ่งการทดลองและจัดเก็บผลการทดลองออกเป็นส่วนต่างๆ ดังต่อไปนี้

4.1 การทดสอบการทำงานของ PE0002 Evaluation Kit Interface Card

4.1.1 การทดสอบเปิดปิดไฟ LED บนบอร์ด PE0002

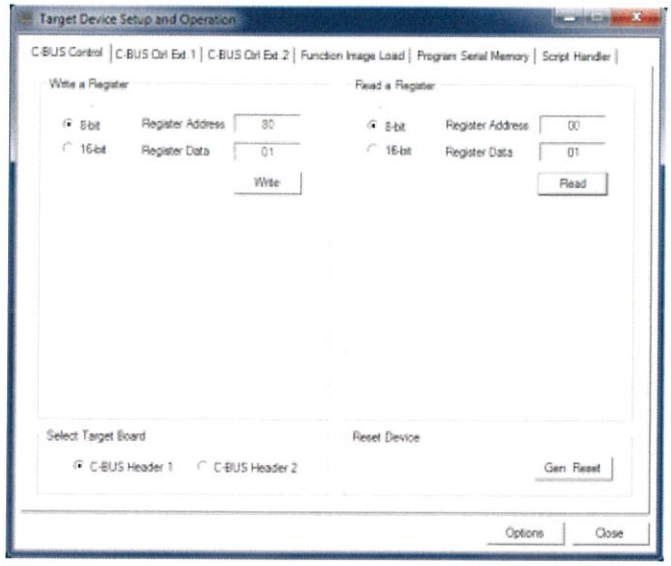
ในการทดลองนี้เป็นการเปิดปิดไฟ LED1 บนบอร์ด PE0002 ด้วยสคริปต์เปิดปิดไฟ (ภาคผนวก ก) เริ่มจากการเชื่อมต่อ PE0002 เข้ากับคอมพิวเตอร์ผ่าน USB โดยบอร์ด PE0002 ต้องการไฟเลี้ยง 5V จากนั้นอัปโหลดสคริปต์เพื่อสั่งการบอร์ด PE0002 ให้เปิด-ปิดไฟ LED1 ไฟ LED1 จะกระพริบเปิดและปิดเรื่อยๆ เมื่อมีการสั่งหยุดสคริปต์หรืออัปโหลดสคริปต์อื่นไฟ LED1 จะดับลง ผลลัพธ์แสดงดังภาพที่ 4.1



ภาพที่ 4.1 ผลลัพธ์จากการทดสอบเปิด-ปิดไฟ LED1

4.1.2 การทดสอบการเชื่อมต่อกับ EV9810

ในการทดลองนี้เป็นการใช้บอร์ด PE0002 เป็นตัวควบคุมเพื่อสื่อสารกับ EV9810 ผ่านโหมด C-BUS โดยใช้โปรแกรม ES000243 เขียนค่าลงรีจิสเตอร์ Configuration control register 1 ด้วยค่า 00 และอ่านค่าได้ 00 แสดงดังภาพที่ 4.2



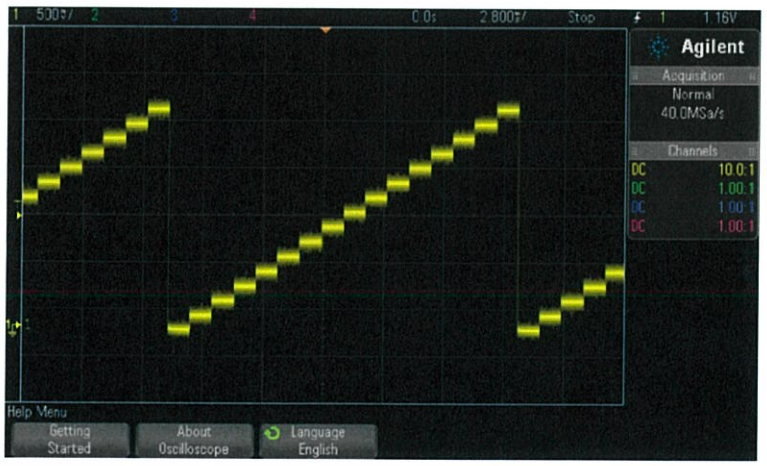
ภาพที่ 4.2 การเขียนและอ่านค่าที่รีจิสเตอร์ Configuration control register 1

4.2 การทดสอบการทำงานของ Evaluation Kit 9810 (EV9810)

4.2.1 การทดสอบ Digital Analog Converter ของ EV9810

ตัวแปลงดิจิทัลเป็นแอนาล็อกของ CMX981 สามารถควบคุมโดยการเขียนลงรีจิสเตอร์ DAC โดยตรงหรือใส่ Aux RAM ด้วยข้อมูลและทำให้สามารถอ่านข้อมูลจากแรมได้ ในการทดลองนี้ ข้อมูลจะถูกเขียนโดยตรงไปยังรีจิสเตอร์ DAC และถูกแปลงเป็นสัญญาณแอนาล็อกโดยตรง

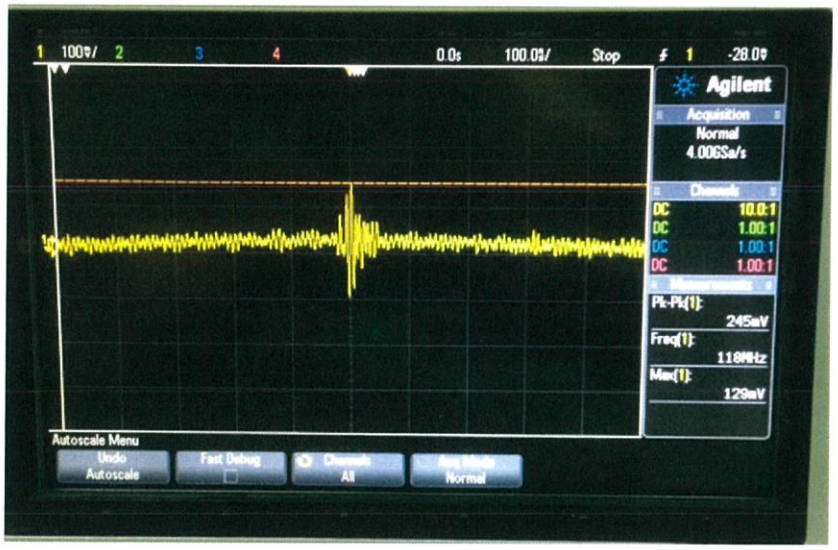
ในการเขียนข้อมูลลงรีจิสเตอร์ DAC จะต้องปิดใช้งาน Aux RAM ก่อน โดยการตั้งค่าบิต 3 ของรีจิสเตอร์ ConfigCtrl2 เป็น 0 และต้องเปิดการใช้งานเอาต์พุตของ DAC เช่น DAC1 โดยการตั้งค่าบิต 0 ของรีจิสเตอร์ PowerDownCtrl เป็น 1 ซึ่งข้อมูลที่เขียนลงรีจิสเตอร์ AuxDacData1 จะถูกแปลงเป็นสัญญาณแอนาล็อกโดยตรงที่ขา DAC1 แสดงดังภาพที่ 4.3 สัญญาณฟันเลื่อยที่สร้างจากการเขียนข้อมูลลงรีจิสเตอร์ AuxDacData1 (ภาคผนวก ข)



ภาพที่ 4.3 สัญญาณฟันเลื่อยที่สร้างจากการเขียนข้อมูลลงรีจิสเตอร์ AuxDacData1

4.2.2 การทดลองส่งข้อมูลบน EV9810

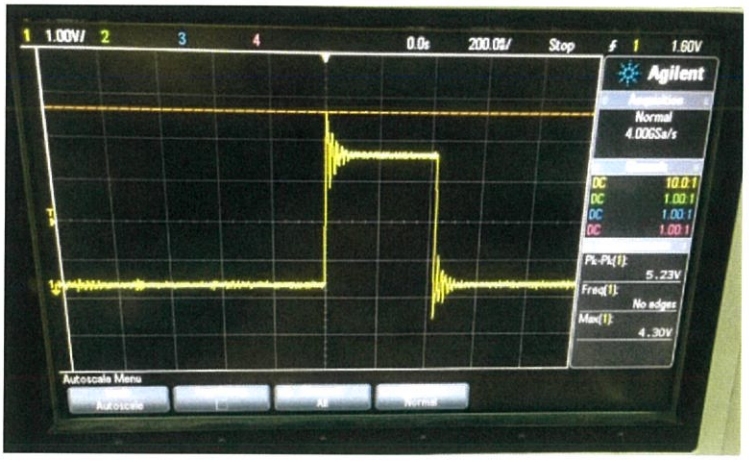
จากบล็อกไดอะแกรมการทำงานของ CMX981 พบว่ามีรีจิสเตอร์ที่สามารถเขียนข้อมูลได้สองตัวคือ 1) Direct Write Access Point 2) Tx Data Access Point โดยในการทดลองนี้ จะเลือกเขียนข้อมูลเข้าที่ Tx Data Access Point ซึ่งในการเขียนข้อมูลชนิด I ต้องเขียนเข้าที่แอดเดรส 3C 8 บิต (Least Significant Bit) แอดเดรส 3D 6 บิต (Most Significant Bit) และในการเขียนข้อมูลชนิด Q ต้องเขียนเข้าที่แอดเดรส 3E 8 บิต (Least Significant Bit) แอดเดรส 3F 6 บิต (Most Significant Bit) จากนั้นทำการเขียนโปรแกรมผ่าน PE0002 (ภาคผนวก ค) เพื่อเขียนข้อมูลชนิด I ที่ Tx Data Access Point และวัดสัญญาณที่ขา TxI บนบอร์ด EV9810 จะสังเกตได้ว่าสัญญาณที่ขา TxI หลังส่งข้อมูลจะถูกขยายขึ้น โดยลักษณะของสัญญาณที่วัดได้ก่อนส่งข้อมูลและหลังส่งข้อมูลแสดงดังภาพที่ 4.4 และ 4.5 ตามลำดับ



ภาพที่ 4.4 สัญญาณที่ขา TxI ก่อนส่งข้อมูล

ภาพที่ 4.5 สัญญาณที่ขา TxI หลังส่งข้อมูล

เขียนข้อมูลที่ Tx Data Access Point ซึ่งในการเขียนข้อมูลชนิด I ต้องเขียนเข้าที่รีจิสเตอร์ Transmit I channel data access point จากนั้นทำการเขียนโปรแกรมผ่าน PE0002 (ภาคผนวก ง) ส่งข้อมูลผ่าน C-BUS ให้ขา CBUSEN เป็นโลจิกต่ำ วัดสัญญาณที่ขา RDATA แสดงดังภาพที่ 4.6 และส่งข้อมูลผ่าน FSB ให้ขา CBUSEN เป็นโลจิกสูง วัดสัญญาณที่ขา RxDAT แสดงดังภาพที่ 4.7



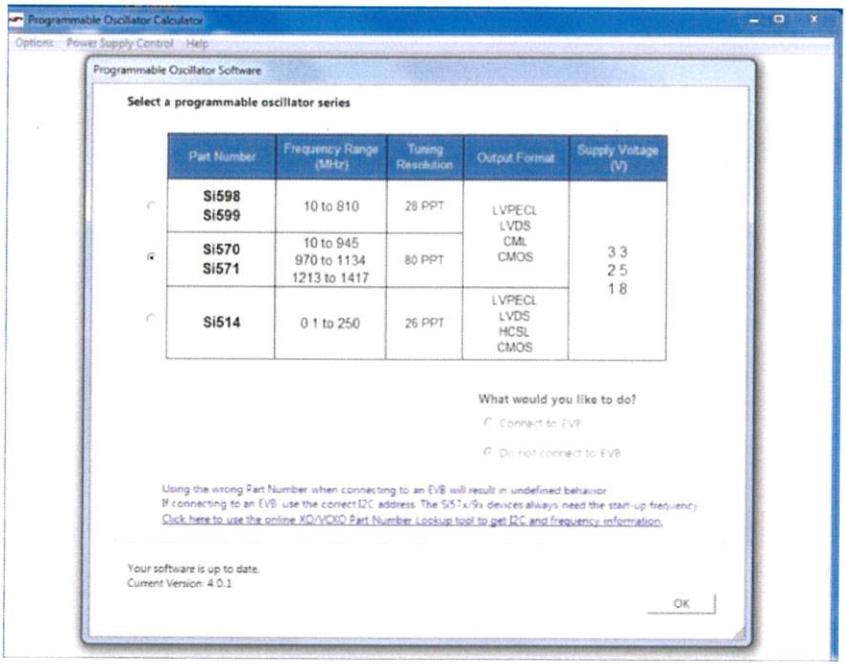
ภาพที่ 4.6 สัญญาณที่ขา RDATA

ภาพที่ 4.7 สัญญาณที่ RxDAT

ซึ่งวงจรรวม CMX981 สามารถเขียนข้อมูลได้ทั้งโหมด C-BUS และโหมด FSB แต่การอ่านข้อมูลนั้นต้องอ่านผ่านโหมด FSB เท่านั้น ซึ่งในบอร์ด EV9810 มีเพียงโหมด C-BUS เท่านั้น เราจึงไม่สามารถอ่านข้อมูลได้ สามารถมองเห็นได้ผ่านออสซิลโลสโคปเท่านั้น จึงได้มีการพัฒนาขึ้นมาเป็นบอร์ด TETRA RTU

4.3 การทดสอบการทำงานของ Si570 Programmable Oscillator Development Kit

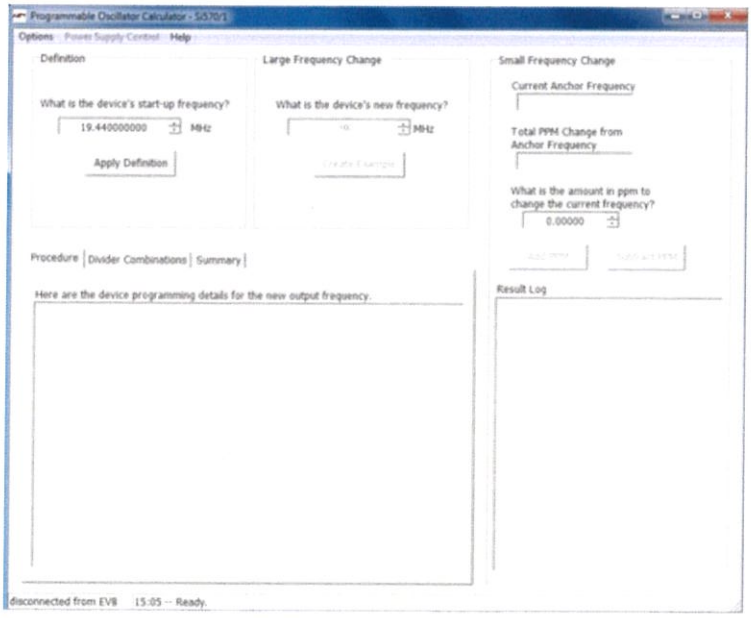
ทดสอบการทำงานของ Si570 โดยเชื่อมต่อ Si570-PROG-EVB เข้ากับคอมพิวเตอร์ผ่านสาย USB และใช้โปรแกรม Programmable Oscillator Software ซึ่งเป็นซอฟแวร์ของ Si570-PROG-EVB มีหน้าต่างการใช้งานแสดงดังภาพที่ 4.8



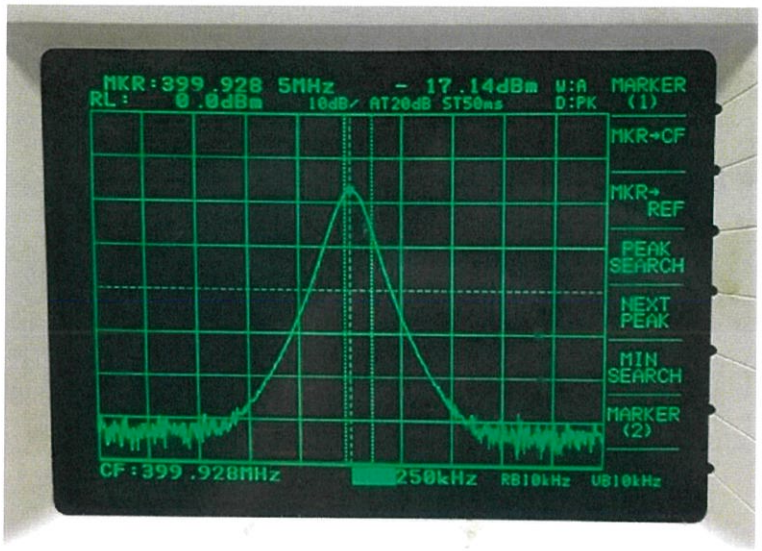
ภาพที่ 4.8 หน้าแรกของโปรแกรม Programmable Oscillator Software

เมื่อเปิดโปรแกรม Programmable Oscillator Software แล้ว เลือกอุปกรณ์ Si570 จากนั้นเลือก Connect to EVB แล้วกด OK โปรแกรมจะทำการเชื่อมต่อระหว่างบอร์ดและโปรแกรมในคอมพิวเตอร์เพื่อให้สามารถควบคุมการทำงานของบอร์ด Si570-PROG-EVB ได้

เมื่อทำการเชื่อมต่อแล้ว โปรแกรมจะแสดงหน้าต่างดังภาพที่ 4.9 ซึ่งเป็นหน้าต่างสำหรับกำหนดความถี่ที่ต้องการสร้างสัญญาณ โดยจะใส่ค่าความถี่ที่ต้องการลงในช่อง “What the device’s new frequency” จากนั้นกด Create Example และใช้เครื่อง Spectrum Analyzer วัดสัญญาณที่ขา CLKP ได้ผลของสัญญาณแสดงดังภาพที่ 4.10



ภาพที่ 4.9 หน้าต่างการใช้งานของโปรแกรม Programmable Oscillator Software



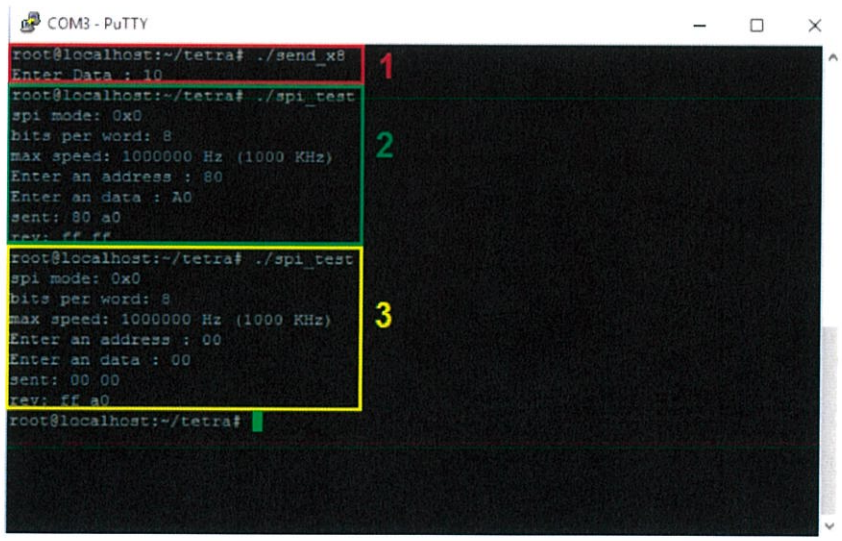
ภาพที่ 4.10 สเปกตรัมที่วัดได้

4.4 การทดสอบการทำงานของอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA

อุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA RTU ถูกออกแบบวงจรตามการเชื่อมต่อของบอร์ด EV9810, EV9980 และ EV9942 และถูกควบคุมด้วยบอร์ด MicroZed บนระบบปฏิบัติการลินุกซ์ ซึ่งสามารถรับ-ส่งข้อมูลผ่านทาง C-BUS และ Fast Serial Bus ได้

4.4.1 การทดสอบการสื่อสารกับ CMX981 ด้วยโหมด C-BUS

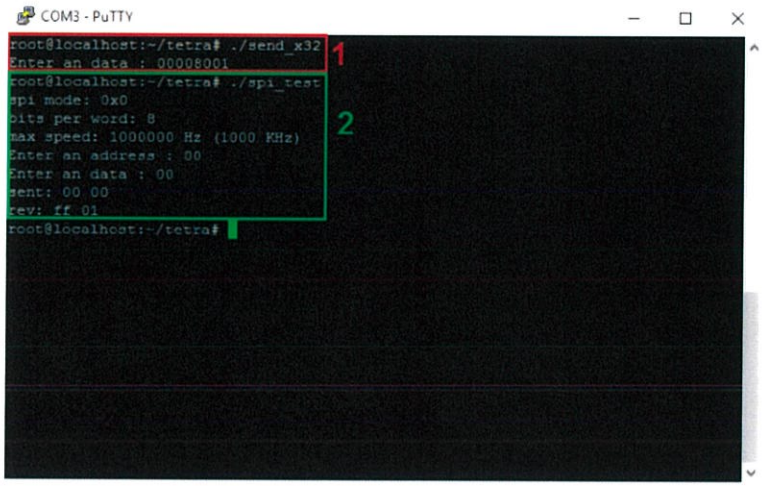
ทดสอบการสื่อสารกับ CMX981 ด้วยโหมด C-BUS ผ่านโปรแกรม PuTTY โดยในส่วนที่ 1) ตั้งค่าให้ใช้ C-BUS สื่อสารกับ CMX981 ส่วนที่ 2) เขียนข้อมูลไปยังรีจิสเตอร์ Configuration control register 1 ด้วยค่า A0 และส่วนที่ 3) อ่านข้อมูลจากรีจิสเตอร์ Configuration control register 1 ได้ A0 แสดงดังภาพที่ 4.11



ภาพที่ 4.11 การสื่อสารกับ CMX981 ด้วยโหมด C-BUS

4.4.2 ทดสอบการสื่อสารกับ CMX981 ด้วยโหมด FSB

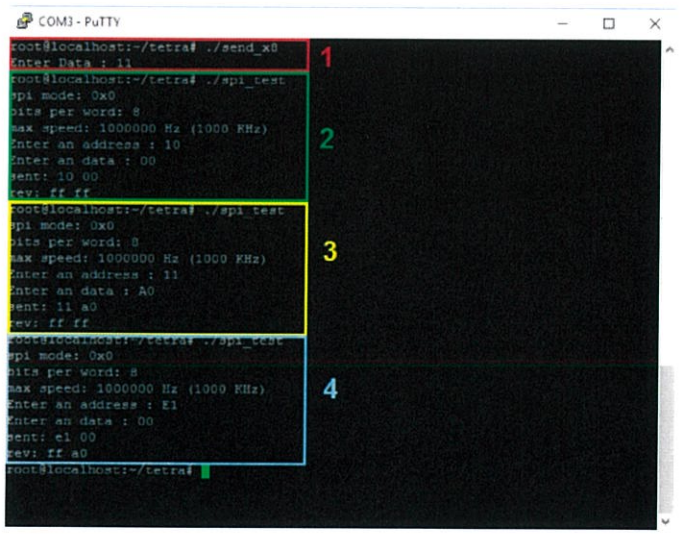
ทดสอบการสื่อสารกับ CMX981 ด้วยโหมด FSB ผ่านโปรแกรม PuTTY โดยในส่วนที่ 1) ตั้งค่าให้ใช้โหมด FSB สื่อสารกับ CMX981 และเขียนข้อมูลลงรีจิสเตอร์ Configuration control register 1 ด้วยค่า 01 และส่วนที่ 2) อ่านข้อมูลจากรีจิสเตอร์ Configuration control register 1 ได้ 01 แสดงดังภาพที่ 4.12



ภาพที่ 4.12 การสื่อสารกับ CMX981 ด้วยโหมด FSB

4.4.3 ทดสอบการสื่อสารกับ CMX994 ด้วยโหมด C-BUS

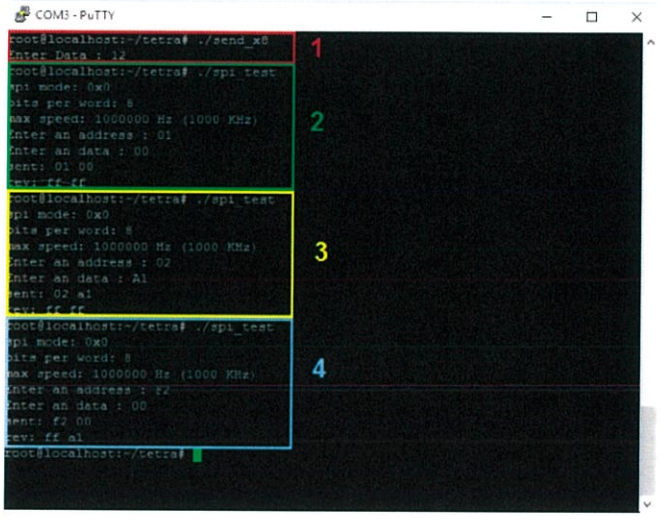
ทดสอบการสื่อสารกับ CMX994 ด้วยโหมด C-BUS ผ่านโปรแกรม PuTTY โดยในส่วนที่ 1) ตั้งค่าให้ใช้ C-BUS สื่อสารกับ CMX994 ส่วนที่ 2) รีเซ็ตรีจิสเตอร์ทั้งหมดด้วยการเขียนข้อมูลไปยังรีจิสเตอร์ Configuration control register 2 ส่วนที่ 3) เขียนข้อมูลไปยัง General register ด้วยค่า A1 และส่วนที่ 4) อ่านข้อมูลจาก General register เป็นค่า A1 แสดงดังภาพที่ 4.13



ภาพที่ 4.13 การสื่อสารกับ CMX994 ด้วยโหมด C-BUS

4.4.4 ทดสอบการสื่อสารกับ CMX998 ด้วยโหมด C-BUS

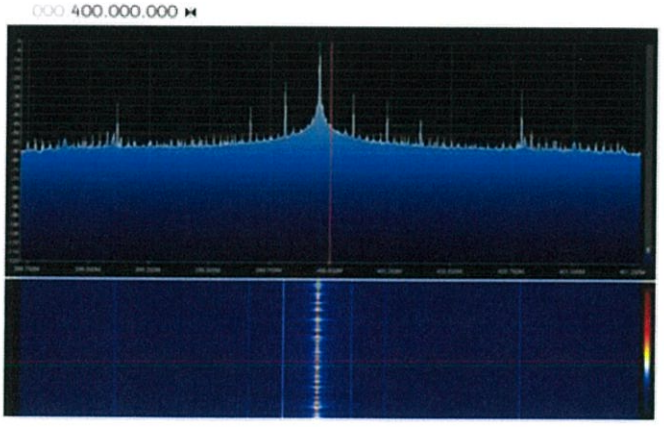
ทดสอบการสื่อสารกับ CMX998 ด้วยโหมด C-BUS ผ่านโปรแกรม PuTTY โดยในส่วนที่ 1) ตั้งค่าให้ใช้ C-BUS สื่อสารกับ CMX998 ส่วนที่ 2) รีเซ็ตรีจิสเตอร์ทั้งหมดด้วยการเขียนข้อมูลไปยังรีจิสเตอร์ Configuration control register 2 ส่วนที่ 3) เขียนข้อมูลไปยัง General register ด้วยค่า A1 และส่วนที่ 4) อ่านข้อมูลจาก General register เป็นค่า A1 แสดงดังภาพที่ 4.14



ภาพที่ 4.14 การสื่อสารกับ CMX998 ด้วยโหมด C-BUS

4.4.5 ทดสอบการสื่อสารกับ Si570

เมื่อรันโปรแกรมบนบอร์ด MicroZed และใส่ค่า Crystal Frequency Multiplication Ratio เป็นอินพุตของโปรแกรม จากนั้นใช้ RTL-SDR วัดสเปกตรัม จะเห็นแท่งของสัญญาณที่ความถี่ที่สร้างสัญญาณ โดยในการทดลองนี้เป็นการทดลองสร้างสัญญาณที่ความถี่ 400 MHz จึงมีแท่งของสัญญาณที่ความถี่ 400 MHz แสดงดังภาพที่ 4.15



ภาพที่ 4.15 สัญญาณที่ถูกสร้างโดย Si570

4.4.5 ทดลองวัดสัญญาณที่ขา Tx

เมื่อทำการวัดสัญญาณที่ขา TxI ซึ่งอยู่หลังวงจรขยาย พบว่าสัญญาณของ TETRA RTU ก่อนส่งข้อมูล (ภาพผนวก ซ) สัญญาณมีแรงดัน 840 mV หลังจากส่งข้อมูล สัญญาณถูกยกระดับขึ้นไปอยู่ที่ 2.20 V แสดงดังภาพที่ 4.16 และ 4.17 ตามลำดับ

ภาพที่ 4.16 สัญญาณที่ขา Tx ก่อนส่งข้อมูล

ภาพที่ 4.17 สัญญาณที่ขา Tx หลังส่งข้อมูล

พบว่าสัญญาณหลังผ่านวงจรขยายมีลักษณะแตกต่างกับชุดทดลอง ซึ่งในชุดทดลองสัญญาณจะถูกขยายขึ้นหลังจากผ่านวงจรขยาย ทำให้เกิดข้อสังเกตว่าวงจรขยายบน TETRA RTU นี้ยังมีส่วนที่ผิดพลาดอยู่ และต้องดำเนินการแก้ไขต่อไป

บทที่ 5

สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผล

โครงงานนี้เป็นการศึกษาและออกแบบอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA ระบบนี้ประกอบด้วยวงจรรวม CMX981 เป็นส่วนประมวลผล CMX998 เป็นส่วนส่งข้อมูล CMX994 เป็นส่วนรับข้อมูล และมี Si570 ทำหน้าที่สร้างสัญญาณคลื่นพาห์เพื่อส่งข้อมูลไปในย่านความถี่ที่เราต้องการ โดยวงจรรวมทั้งหมดนี้จะถูกควบคุมผ่านบอร์ด MicroZed บนระบบปฏิบัติการลินุกซ์ ซึ่งอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA ถูกออกแบบและพัฒนาจากอุปกรณ์ชุดทดลอง Evaluation Kit โดยอุปกรณ์ชุดทดลองนี้มีผลของสัญญาณข้อมูลที่สามารถวัดได้ แต่ยังไม่สามารถแสดงผลการรับส่งข้อมูลบนคอมพิวเตอร์ได้ เนื่องจากมีข้อจำกัดเกี่ยวกับความเร็วของอุปกรณ์ ทำให้ไม่สามารถเรียกอ่านค่าของข้อมูลได้ ในการออกแบบอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA จึงมีการพัฒนาให้สามารถถ่ายโอนข้อมูลที่ความเร็วสูงขึ้น

ผลการทดสอบการทำงานพบว่า สามารถสั่งการอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA เพื่อเขียนและอ่านค่ารีจิสเตอร์ต่างๆ ได้อย่างถูกต้อง โดยสามารถเขียนได้ทั้งแบบ Serial Peripheral Interface (SPI) และ Fast Serial Bus (FSB) แต่ในการทดสอบส่งข้อมูล สัญญาณหลังผ่านวงจรขยายยังมีลักษณะที่แตกต่างจากอุปกรณ์ชุดทดลอง ซึ่งอาจเป็นสาเหตุที่ทำให้อุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA ยังไม่สามารถส่งข้อมูลได้

5.2 ข้อเสนอแนะ

จากผลการทดสอบการทำงานของอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA นี้ จะเห็นได้ว่าสามารถควบคุมและสั่งการในเบื้องต้นได้ แต่ยังไม่สามารถส่งและรับข้อมูลได้จริง ซึ่งสาเหตุอาจเกิดจากความผิดพลาดของวงจรขยาย ในการปรับปรุงและพัฒนาต่อ ควรศึกษาข้อมูลและแก้ไขวงจรให้ทำงานได้ เพื่อให้สามารถส่งสัญญาณออกได้อย่างถูกต้อง

บรรณานุกรม

[1] "Terrestrial Trunked Radio", เข้าถึงได้จาก https://en.wikipedia.org/wiki/Terrestrial_Trunked_Radio

[2] "TETRA (Terrestrial Trunked Radio)", เข้าถึงได้จาก http://www.mobilecommstechnology.com/projects/tetra/

[3] "โครงการศึกษาระบบ trunked radio เพื่อใช้ในภารกิจสื่อสารกรมชลประทาน", เข้าถึงได้จาก http://kromchol.rid.go.th/ict/cmd/download/DigitalTrunkedRadio2013.pdf

[4] "วิทยุสื่อสาร ระบบ ทรั้งค์แบบ ดิจิทัล" โทรคมนาคมที่มีอนาคตดีอีกระบบหนึ่ง", เข้าถึงได้จาก http://www.manager.co.th/asp-bin/PrintNews.aspx?NewsID=9500000122937

[5] "CMX981", เข้าถึงได้จาก http://www.cmlmicro.com/products/CMX981_Advanced_Digital_Radio_ Baseband_Processor/

[6] "EV981", เข้าถึงได้จาก http://www.cmlmicro.com/searchresults/?q=evaluation%20981

[7] "CML", เข้าถึงได้จาก http://archive.eetasia.com/www.eetasia.com/ART_8800281072_1034362_NP_bc323bf1.HTM

[8] "Evaluation Kit for the CMX981", เข้าถึงได้จาก http://www.cmlmicro.com/products/ EV9810_Evaluation_Kit/

[9] "CMX998", เข้าถึงได้จาก http://mikrokontroler.pl/2010/11/22/cmx998-scalony-uklad-linearyzacji-nadajnikow-radiowych-firmy-cml-microcircuits/

[10] "Evaluation Kit for the CMX998", เข้าถึงได้จาก http://www.cmlmicro.com/products/ EV9980_Evaluation_ Kit/?q=ev9980&curr=69

[11] "Radio integrated circuit receiver", เข้าถึงได้จาก http://www.directindustry.com/prod/ cml-microcircuits/product-34037-782135.html

[12] "RF Direct Conversion Receiver ICs", เข้าถึงได้จาก http://www.cmlmicro.com/products/CMX_994_RF_Direct_Conversion_Receiver/?q=cmx994&curr=24

[13] "EV9942 User Manual (rev5)", เข้าถึงได้จาก http://www.cmlmicro.com/DesignSupport/resources/2013/03/25/EV9942UserManual

[14] "PE0002", เข้าถึงได้จาก http://www.cmlmicro.com/products/PE0002_Evaluation_Kit _Interface _Card/

[15] "เอฟพีจีเอ", เข้าถึงได้จาก https://th.wikipedia.org/wiki/เอฟพีจีเอ

[16] "Zynq-7000", เข้าถึงได้จาก https://www.xilinx.com/support/documentation/data_ sheets/ds190- Zynq-7000-Overview.pdf

[17] "Processing System FPGA", เข้าถึงได้จาก https://www.xilinx.com/content/dam/xilinx/imgs/block-diagrams/zynq-mp-core-single.png

[18] "FPGA Blog", เข้าถึงได้จาก http://fpgablog.com/posts/avnet-vivado/

[19] "SPI", เข้าถึงได้จาก http://aimagin.com/blog/spi/?lang=th

[20] "การติดต่อสื่อสารด้วย SPI", เข้าถึงได้จาก http://www.123microcontroller.com/Hardware-Interfacing/SPI-Serial-Peripheral-Interface-communication

[21] "I2C BUS", เข้าถึงได้จาก https://www.i2c-bus.org/addressing/

[22] "CBUS", เข้าถึงได้จาก http://www.cmlmicro.com/products/CMX981_Advanced_Digital_Radio_ Baseband_Processor/

[23] "Si570", เข้าถึงได้จาก https://www.silabs.com/documents/public/data-sheets/si570.pdf

[24] "Linux", เข้าถึงได้จาก https://web.ku.ac.th/schoolnet/snet1/software/linux/index.html

[25] "DQPSK", เข้าถึงได้จาก https://www.allaboutcircuits.com/technical-articles/differential-quadrature-phase-shift-keying-dqpsk-modulation/

ภาคผนวก ก
คำสั่งในการทดสอบเปิดปิดไฟ LED ของบอร์ด PE0002

การเปิดปิดไฟ LED บนบอร์ด PE0002 โดยแสดงดังนี้

```
Port1cpy        word
Main
        jsr Led1On
        delay 300
        jsr Led1Off
        delay 300
        jmp Main
Led1Off
        port 1                  ;Ensure Port1 is default
        portr Port1cpy          ;Read current setting
        and $FFFE Port1cpy      ;Turn off LED1 (b0 = 0)
        portw Port1cpy
        return
Led1On
        port 1                  ;Ensure Port1 is default
        portr Port1cpy          ;Read current setting
        or $0001 Port1cpy       ;Turn on LED1 (b0 = 1)
        portw Port1cpy
        return
```

ภาคผนวก ข
คำสั่งในเขียนข้อมูลลงรีจิสเตอร์ DAC ของบอร์ด EV9810

การเขียนข้อมูลลงรีจิสเตอร์เป็นสัญญาณฟันเลื่อย โดยแสดงดังนี้

```
; Registers
AuxDacData1_write const $D1
RamDacCtrl_read const $14
RamDacCtrl_write const $94
PowerDownCtrl_read const $12
PowerDownCtrl_write const $92
; main
temp word 1
        device 1
        cls
        copy □ RamDacCtrl_read , temp
        and $FD00 , temp
        copy temp , □ RamDacCtrl_write
        ; make sure that aux mem i s turned o f f
        copy 0 , temp
        copy $0100 , □ PowerDownCtrl_write ; turn on aux dac1
        delay 1
main_loop
        ; sawtooth 1
        copy temp , □ AuxDacData1_write
        add $1000 , temp
        delay 1
        jmp main_loop
```

ภาคผนวก ค
คำสั่งในเขียนข้อมูลผ่าน C-BUS

การเขียนข้อมูลชนิด I ด้วยค่าตั้งแต่ 0000 – 3FFF ผ่าน C-BUS โดยแสดงดังนี้

```
;*************************
; CMX981 register
;*************************

ConfigCtrl1_write           const $80
IRQCtrl_write               const $82
ClkStopCtrl_write           const $91
PowerDownCtrl_write         const $92
RxSetup1_write              const &88
RxSetup2_write              const &89
TxSetup_write               const $83
LoopBackCtrl_write          const $93

TxDPIData1_write            const $BC
TxDPIData2_write            const $BD
TxDPIData1_read             const $3C
TxDPIData2_read             const $3D
TxDPQData1_write            const $BE
TxDPQData2_write            const $BF
RxDPIData1_read             const &38
RxDPIData2_read             const &39
RxDPQData1_read             const &BA
RxDPQData2_read             const &BB
TxIGain1_write              const $A2
TxIGain2_write              const $A3
TxIOffset1_write            const $A4
TxIOffset2_write            const $A5
RxIGain1_write              const $B4
RxIGain2_write              const $B5
RxIOffset1_write            const $B6
RxIOffset2_write            const $B7

;************************
;declare register data size
;************************

     register   1, ConfigCtrl1_write, 1
     register   1, IRQCtrl_write, 1
     register   1, ClkStopCtrl_write, 1
     register   1, PowerDownCtrl_write, 1
     register   1, RxSetup1_write, 1
     register   1, RxSetup2_write, 1
     register   1, TxSetup_write, 1
     register   1, LoopBackCtrl_write, 1
     register   1, TxDPIData1_write, 1
     register   1, TxDPIData2_write, 1
     register   1, TxDPIData1_read, 1
     register   1, TxDPIData2_read, 1
     register   1, TxDPQData1_write, 1
     register   1, TxDPQData2_write, 1
     register   1, RxDPIData1_read, 1
     register   1, RxDPIData2_read, 1
     register   1, RxDPQData1_read, 1
```

```
      register   1, RxDPQData2_read, 1
      register   1, TxIGain1_write, 1
      register   1, TxIGain2_write, 1
      register   1, TxIOffset1_write, 1
      register   1, TxIOffset2_write, 1
      register   1, RxIGain1_write, 1
      register   1, RxIGain2_write, 1
      register   1, RxIOffset1_write, 1
      register   1, RxIOffset2_write, 1

; main

dataA           word 0
dataB           word 0
i               word 0
Iread1          word 0
Iread2          word 0
Qread1          word 0
Qread2          word 0
temp1           word 0
temp2           word 0
temp3           word 0
temp4           word 0
flag            word 0

      device     1  ; select CBUS1

      jsr  receive_loop
      lsl  8, dataB
      lsl  8, Iread2
      copy $0000, *ConfigCtrl1_write
      copy $9800, *TxSetup_write
      copy $0600, *LoopBackCtrl_write

receive_loop
      copy dataA, *TxDPIData1_write
      copy dataB, *TxDPIData2_write
      copy dataA, temp1
      copy dataB, temp2
      add temp2, temp1
      disp "%x        \c" , temp1
      if   flag =0
           add #1, dataA
           while (dataA =256)
                copy 0, dataA
                add 256, dataB
           endwhile
           while (temp1 =16384)
                copy 1, flag
                copy 16383, temp1
                copy 1, dataA
           disp "flag =%d", flag
           endwhile
     endif
```

```
if    flag ==1
            sub   #1, dataA
      while (dataA ==0)
            copy  255, dataA
            sub   256, dataB
      endwhile
      while (temp1 =1)
            copy  0, flag
            copy  0, temp1
            copy  0, dataA
            copy  0, dataB
            disp "temp1 =%d", temp1
      endwhile
endif
copy *TxDPIData1_read, Iread1
copy *TxDPIData2_read, Iread2
add Iread1, Iread2
disp "    %x \c", Iread1
disp "%x", Iread2
delay 2
jmp   receive_loop
```

ภาคผนวก ง
คำสั่งในเขียนข้อมูลผ่าน FSB

การเขียนข้อมูลชนิด I ด้วยค่า A3 ผ่าน FSB โดยแสดงดังนี้

```
ConfigCtrl1_read          const $00
ConfigCtrl1_write         const $80
TxDPIData1_write          const $BC
TxDPIData1_read           const $3C
RxDPIData1_write          const $B8
RxDPIData1_read           const $38
LoopBackCtrl_write        const $93
LoopBackCtrl_read         const $13
RxSetup1_write            const $88
RxSetup1_read             const $08
TxSetup_write             const $83
TxSetup_read              const $03

; main

rx      word  0
i       word  0

        copy  $0000, *ConfigCtrl1_write
        copy  $0200, *RxSetup1_write
        copy  $9800, *TxSetup_write
        copy  $0600, *LoopBackCtrl_write
        copy  $A300, *TxDPIData1_write
        jsr   read_loop

read_loop
        copy  $A300, *TxDPIData1_write
        copy  *TxSetup_read, rx
        disp  "%x", rx
        delay 200
        jmp   read_loop
```

ภาคผนวก จ
คำสั่งเปิดใช้งาน C-BUS บนบอร์ด TETRA RTU

ตรวจและเปิดการใช้งาน C-BUS บนบอร์ด TETRA RTU แสดงดังนี้

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdint.h>
static void send_x8(uint8_t data){
    int fdw;
    struct
    {
       uint8_t v1;
    } tologic;
    fdw = open("/dev/xillybus_write_8", O_WRONLY);
    if (fdw < 0) {
                perror("Failed to open Xillybus device file(s)");
                exit(1);
    }
    tologic.v1 = data;
    write(fdw, (void *) &tologic, sizeof(tologic));
    close(fdw);
}
int main(int argc, char *argv[]){
        uint8_t D;
        printf("Enter Data : ");
        scanf("%02x", &D);
        send_x8(D);

        return 0;
}
```

ภาคผนวก ฉ
คำสั่งเปิดใช้งาน FSB บนบอร์ด TETRA RTU

ตรวจและเปิดการใช้งาน FSB บนบอร์ด TETRA RTU แสดงดังนี้

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdint.h>

static void send_X32(uint32_t data){
    int fdw;
        struct
        {
                uint32_t v1;
        } tologic;

        fdw = open("/dev/xillybus_write_32", O_WRONLY); //open port write
        if (fdw < 0) {
        perror("Failed to open Xillybus device file(s)");
        exit(1);
        }
        tologic.v1 = data;
        write(fdw, (void *) &tologic, sizeof(tologic)); //send Data
        close(fdw);
}
int main(int argc, char *argv[]) {
        uint32_t D;

        printf("Enter an data : ");
        scanf("%08x", &D);
        send_X32(D);

        return 0;
}
```

ภาคผนวก ช
คำสั่งการเขียนและอ่านด้วยบอร์ด TETRA RTU

การเขียนอ่านข้อมูลบอร์ด TETRA RTU แสดงดังนี้

```
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>
#include <time.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))
struct spi_ioc_transfer msg[1];

static void pabort(const char *s) {
        perror(s);
        abort();
}
static const char *device = "/dev/spidev1.0";
uint8_t tx[10];
uint8_t rx[10];
static uint8_t mode;
static uint8_t bits = 8;
static uint32_t speed = 1000000;
static void reciver(int fd,uint8_t address,uint8_t data){
        int status;
        tx[0]=address;
        tx[1]=data;
        msg[0].tx_buf = (unsigned long)tx;
        msg[0].rx_buf = (unsigned long)rx;
        msg[0].len =2;
        msg[0].cs_change =0;
```

```
        msg[1].delay_usecs = 0;
        msg[0].speed_hz=speed;
        msg[0].bits_per_word =8 ;
                status = ioctl(fd,SPI_IOC_MESSAGE(1),&msg);
        if(status < 0){
                pabort("SPI_IOC_MESSAGE");
        }
        printf("sent: %02x %02x \n", tx[0], tx[1]);
        printf("rev: %02x %02x \n", rx[0], rx[1]);
}
static void delay(int milliseconds) {
    long pause;
    clock_t now,then;
    pause = milliseconds*(CLOCKS_PER_SEC/1000);
    now = then = clock();
    while( (now-then) < pause )
        now = clock();
}
int main(){
        int ret = 0;
        int fd;
        uint8_t A,D;
        mode = 0x00;
        fd = open(device, O_RDWR);
        if (fd < 0)
                pabort("can't open device");
        /*
         * spi mode
         */
        ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);
        if (ret == -1)
                pabort("can't set spi mode");
        ret = ioctl(fd, SPI_IOC_RD_MODE, &mode);
        if (ret == -1)
                pabort("can't get spi mode");
```

```
	/*
	 * bits per word
	 */
	ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
	if (ret == -1)
		pabort("can't set bits per word");
	ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
	if (ret == -1)
		pabort("can't get bits per word");
	/*
	 * max speed hz
	 */
	ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
	if (ret == -1)
		pabort("can't set max speed hz");
	ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
	if (ret == -1)
		pabort("can't get max speed hz");
	printf("spi mode: 0x%x\n", mode);
	printf("bits per word: %d\n", bits);
	printf("max speed: %d Hz (%d KHz)\n", speed, speed/1000);
	printf("Enter an address : ");
	scanf("%02x", &A);
	printf("Enter an data : ");
	scanf("%02x", &D);
	reciver(fd,A,D);
	delay(100);
	return ret;
}
```

ภาคผนวก ซ
คำสั่งการเขียนด้วยบอร์ด TETRA RTU

การเขียนคำสั่งเซ็ตค่ารีจิสเตอร์และส่งข้อมูลบนบอร์ด TETRA RTU แสดงดังนี้

```
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>
#include <time.h>
#include <pthread.h>
#include <math.h>
//#include <windows.h>
#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))
/* ---READ (CMX981)--- */
#define ConfigCtrl1_read            0x00
#define ConfigCtrl2_read            0x01
#define IRQCtrl_read                0x02
#define Status1_read                0x0A
#define InterruptMask1_read         0x0B
#define Status2_read                0x0C
#define InterruptMask2_read         0x0D
#define Status3_read                0x0E
#define InterruptMask3_read         0x0F
#define SymClkPhase_read            0x10
#define ClkStopCtrl_read            0x11
#define PowerDownCtrl_read          0x12
#define LoopBackCtrl_read           0x13
#define RamDacCtrl_read             0x14
#define AuxAdcCtrl1_read            0x15
#define AuxAdcCtrl2_read            0x16
#define RampCtrl_read               0x17
```

```
#define ClkDiv1_read                0x60
#define ClkDiv2_read                0x61
/* ---WRITE (CMX981)--- */
#define ConfigCtrl1_write           0x80
#define ConfigCtrl2_write           0x81
#define IRQCtrl_write               0x82
#define Status1_write               0x8A
#define InterruptMask1_write 0x8B
#define Status2_write               0x8C
#define InterruptMask2_write 0x8D
#define Status3_write               0x8E
#define InterruptMask3_write 0x8F
#define SymClkPhase_write           0x90
#define ClkStopCtrl_write           0x91
#define PowerDownCtrl_write         0x92
#define LoopBackCtrl_write          0x93
#define RamDacCtrl_write            0x94
#define AuxAdcCtrl1_write           0x95
#define AuxAdcCtrl2_write           0x96
#define RampCtrl_write              0x97
#define ClkDiv1_write               0xE0
#define ClkDiv2_write               0xE1
#define AuxDacData1_write           0xD1
#define TxSetup_read                        0x03
#define TxSetup_write                       0x83
#define TxData_write                        0x84
#define TxData_multisymbol_write            0x85
#define TxData_rampup_write                 0x86
#define TxData_multisymbol_rampup_write     0x87
#define TxIPhase1_read              0x20
#define TxIPhase2_read              0x21
#define TxIPhase1_write             0xA0
#define TxIPhase2_write             0xA1
#define TxIGain1_read               0x22
#define TxIGain2_read               0x23
```

```
#define TxIGain1_write          0xA2
#define TxIGain2_write          0xA3
#define TxIOffset1_read         0x24
#define TxIOffset2_read         0x25
#define TxIOffset1_write        0xA4
#define TxIOffset2_write        0xA5
#define TxQPhase1_read          0x26
#define TxQPhase2_read          0x27
#define TxQPhase1_write         0xA6
#define TxQPhase2_write         0xA7
#define TxQGain1_read           0x28
#define TxQGain2_read           0x29
#define TxQGain1_write          0xA8
#define TxQGain2_write          0xA9
#define TxQOffset1_read         0x2A
#define TxQOffset2_read         0x2B
#define TxQOffset1_write        0xAA
#define TxQOffset2_write        0xAB
#define TxDPIData1_read         0x3C
#define TxDPIData2_read         0x3D
#define TxDPQData1_read         0x3E
#define TxDPQData2_read         0x3F
#define TxDPIData1_write        0xBC
#define TxDPIData2_write        0xBD
#define TxDPQData1_write        0xBE
#define TxDPQData2_write        0xBF
#define RxDPIData1_read         0x38
#define RxDPIData2_read         0x39
#define RxDPQData1_read         0x3A
#define RxDPQData2_read         0x3B
#define RxDPIData1_write        0xB8
#define RxDPIData2_write        0xB9
#define RxDPQData1_write        0xBA
#define RxDPQData2_write        0xBB
#define RxIGain1_read           0x31
```

```
#define RxIGain2_read                0x32
#define RxIGain1_write               0xB1
#define RxIGain2_write               0xB2
#define RxIOffset1_read              0x32
#define RxIOffset2_read              0x33
#define RxIOffset1_write             0xB2
#define RxIOffset2_write             0xB3
#define RxQGain1_read                0x34
#define RxQGain2_read                0x35
#define RxQGain1_write               0xB4
#define RxQGain2_write               0xB5
#define RxSetup1_read                0x08
#define RxSetup2_read                0x09
#define RxSetup1_write               0x88
#define RxSetup2_write               0x89
#define RxQOffset1_read              0x36
#define RxQOffset2_read              0x37
#define RxQOffset1_write             0xB6
#define RxQOffset2_write             0xB7
#define DirectWrite79tapI1           0x6F
#define DirectWrite79tapQ1           0x7F
#define IRQCtrl_read                 0x02
#define IRQCtrl_write                0x82
/* ---WRITE (CMX998)--- */
#define General_Reset_write          0x01
#define General_Control_write        0x02
#define Phase_Control_write          0x03
#define Gain_Control_write_16bit     0x04
#define Gain_Control_write_Forword 0x05
#define Gain_Control_write_Feedback 0x06
/* 05,06 == 04*/
#define Aux_Control_write            0x07
#define Frequency_Control_write      0x08
/* ---READ (CMX998)--- */
#define General_Control_read         0xF2
```

```
#define Phase_Control_read              0xF3
#define Gain_Control_read_16bit         0xF4
#define Gain_Control_read_Forword       0xF5
#define Gain_Control_read_Feedback      0xF6
#define Aux_Control_read 0xF7
#define Frequency_Control_read 0xF8
#define pi 3.14159265
static uint8_t  temp1 ;
static uint8_t temp ;
struct spi_ioc_transfer msg[1];
static void pabort(const char *s) {
        perror(s);
        abort();
}
static const char *device = "/dev/spidev1.0";
uint8_t tx[10];
uint8_t rx[10];
static uint8_t mode;
static uint8_t bits = 8;
static uint32_t speed = 1000000;
static void send_x8(uint8_t data){
   int fd;
   struct
   {
     uint8_t v1;
   } tologic;
   fd = open("/dev/xillybus_write_8", O_WRONLY);
   if (fd < 0) {
                perror("Failed to open Xillybus device file(s)");
                exit(1);
   }
   tologic.v1 = data;
   write(fd, (void *) &tologic, sizeof(tologic));
   close(fd);
}
```

```
static void send_x32(uint32_t data){
    int fd;
        struct
        {
                uint32_t v1;
        } tologic;

        fd = open("/dev/xillybus_write_32", O_WRONLY); //open port write
        if (fd < 0) {
        perror("Failed to open Xillybus device file(s)");
        exit(1);
        }
        tologic.v1 = data;
        write(fd, (void *) &tologic, sizeof(tologic)); //send Data
        close(fd);
}

static uint8_t reciver(int fd,uint8_t address,uint8_t data){
        int status;
        tx[0]=address;
        tx[1]=data; //  tx[2]=0x00;
        msg[0].tx_buf = (unsigned long)tx;
        msg[0].rx_buf = (unsigned long)rx;
        msg[0].len =2;
        msg[0].cs_change =0;
        msg[1].delay_usecs = 0;
        msg[0].speed_hz=speed;
        msg[0].bits_per_word =8 ;

                status = ioctl(fd,SPI_IOC_MESSAGE(1),&msg);

        if(status < 0){
                pabort("SPI_IOC_MESSAGE");
        }
        long pause;
```

```
    clock_t now,then;
    pause = 3*(CLOCKS_PER_SEC/1000);
    now = then = clock();
    while( (now-then) < pause )
    now = clock();
        printf("sent: %02x %02x \n", tx[0], tx[1]);
        printf("rev: %02x %02x \n", rx[0], rx[1]);
        return rx[1];
}
void initialCMX981(int fd){
   /*initial setup        for CMX981 register*/
        send_x8(0x10);
        reciver(fd,ConfigCtrl1_write,0x00);
        reciver(fd,ConfigCtrl2_write,0x00);
        reciver(fd,IRQCtrl_write,0x00);
        //disable interupt masking
        reciver(fd,InterruptMask1_write,0x00);
        reciver(fd,InterruptMask2_write,0x00);
        reciver(fd,InterruptMask3_write,0x00);
        reciver(fd,ClkStopCtrl_write,0x00);
        reciver(fd,TxSetup_write,0x00);
        reciver(fd,ClkDiv1_write,0x00);
        reciver(fd,ClkDiv2_write,0x0F);
        reciver(fd,TxQPhase1_write,0x00);
        reciver(fd,TxIPhase1_write,0x00);
        reciver(fd,TxQGain1_write ,0xFF);
        reciver(fd,TxQGain2_write,0x07);
        reciver(fd,TxIGain1_write,0xFF);
        reciver(fd,TxIGain2_write,0x07);
        reciver(fd,TxQOffset1_write,0x00);
        reciver(fd,TxQOffset2_write,0x00);
        reciver(fd,TxIOffset1_write,0x00);
        reciver(fd,TxIOffset2_write,0x00);
        reciver(fd,LoopBackCtrl_write,0x01);
}
```

```
void initialCMX998(int fd){
        /*initial setup   for CMX998 register*/
        send_x8(0x12);
        reciver(fd,General_Control_write ,0xFC);
        reciver(fd,Phase_Control_write,0x00);
        reciver(fd,Gain_Control_write_Forword,0x00);
        reciver(fd,Gain_Control_write_Feedback,0xFF);
        reciver(fd,Frequency_Control_write,0x00);
}
static void delay(int milliseconds) {
   long pause;
   clock_t now,then;
   pause = milliseconds*(CLOCKS_PER_SEC/1000);
   now = then = clock();
   while( (now-then) < pause )
      now = clock();
}
void* txloop(void *param){
        printf("\n------- START TXLOOP-------\n");
        send_x8(0x10);
        int status;
        int fd = (intptr_t) param;
        reciver(fd,ConfigCtrl1_write,0x08);
        /*reciver(fd,TxData_write,0x03);
        reciver(fd,TxData_rampup_write,0x02);
        reciver(fd,Status1_read,00);
        reciver(fd,Status2_read,00);
        reciver(fd,Status3_read,00);*/
        delay(1000);
        for(int j=0;j<100;j++){
        while(!(0x10&reciver(fd,Status1_read,00)));
        reciver(fd,TxData_write,0x03);
        reciver(fd,TxData_multisymbol_write,0xAA);
        reciver(fd,TxSetup_write,0x88);
        while(!(0x10&reciver(fd,Status1_read,00)));
```

```
        reciver(fd,TxData_write,0x01);
        reciver(fd,TxData_multisymbol_write,0xAA);
        reciver(fd,TxSetup_write,0x88);
        }

}
void* txloop2(void *param){
        printf("\n------- START TXLOOP2-------\n");
        int status;
        int DD;
        int fd = (intptr_t) param;
        send_x8(0x10);
        uint16_t TxI = 0x0000;
        uint16_t TxQ = 0x0000;

        for(int j=0;j<1000;j++){
                tx[0]=0xBC;
                tx[1]= (uint8_t)(TxI & 0xFF); // tx[2]=0x00;

                msg[0].tx_buf = (unsigned long)tx;
                msg[0].rx_buf = (unsigned long)rx;
                msg[0].len =2;
                msg[0].cs_change =0;
                msg[1].delay_usecs = 0;
                msg[0].speed_hz=speed;
                msg[0].bits_per_word =8 ;
                        status = ioctl(fd,SPI_IOC_MESSAGE(1),&msg);
                if(status < 0){
                        pabort("SPI_IOC_MESSAGE");
                        delay(1000);

                }

                printf("sent 0xBC [%d]: %02x %02x \n",j, tx[0], tx[1]);
                printf("rev 0xBC : %02x %02x \n", rx[0], rx[1]);
```

```
tx[0]=0xBD;
tx[1]=(TxI>>8)&0x3F; //          tx[2]=0x00;
msg[0].tx_buf = (unsigned long)tx;
msg[0].rx_buf = (unsigned long)rx;
msg[0].len =2;
msg[0].cs_change =0;
msg[1].delay_usecs = 0;
msg[0].speed_hz=speed;
msg[0].bits_per_word =8 ;
        status = ioctl(fd,SPI_IOC_MESSAGE(1),&msg);
if(status < 0){
        pabort("SPI_IOC_MESSAGE");
}
tx[0]=0xBE;
tx[1]= (uint8_t)( TxQ & 0xFF); //        tx[2]=0x00;
msg[0].tx_buf = (unsigned long)tx;
msg[0].rx_buf = (unsigned long)rx;
msg[0].len =2;
msg[0].cs_change =0;
msg[1].delay_usecs = 0;
msg[0].speed_hz=speed;
msg[0].bits_per_word =8;
status = ioctl(fd,SPI_IOC_MESSAGE(1),&msg); //การเขียนโครงสร้างให้ส่งผ่าน spi bus
if(status < 0){
        pabort("SPI_IOC_MESSAGE");
}
tx[0]=0xBF;
tx[1]=( TxQ>>8 )&0x3F; //      tx[2]=0x00;
msg[0].tx_buf = (unsigned long)tx;
msg[0].rx_buf = (unsigned long)rx;
msg[0].len =2;
msg[0].cs_change =0;
msg[1].delay_usecs = 0;
msg[0].speed_hz=speed;
msg[0].bits_per_word =8 ;
```

```
                status = ioctl(fd,SPI_IOC_MESSAGE(1),&msg);
        if(status < 0){
                pabort("SPI_IOC_MESSAGE");
        }
    }
        TxQ += 0x01;
        if(TxQ == 0xFFFF)
                TxQ = 0x00;
}

void Txloop2new(int fd){
    printf("\n------- START TXLOOP2-------\n");
    int status;
    uint16_t TxI = 0x0000;
    uint16_t TxQ = 0x0000;
    uint8_t Address;
    uint8_t Data;
    uint8_t TxReceive;
    uint8_t Tx;
    uint8_t TxStatusIRQ;
    uint8_t TxStatusQOverflow;
    uint8_t TxStatusIOverflow;
    for(int j=0;j<1000;j++){
    printf("---------------- Round %d ----------\n",j );
    /*  Write I **/
        Address=0xBC;
        Data = (uint8_t)(TxI & 0xFF);
        printf("Write I On 3C");
        reciver(fd,Address,Data);

        Address=0xBD;
        Data =(TxI>>8)&0x3F;
        printf("Write I On 3D");
        reciver(fd,Address,Data);
```

```
        TxI += 0x01;
        if(TxI == 0xFFFF)
                TxI = 0x0000;
/*  Write Q **/
        Address = 0xBE;
        Data    = (uint8_t)( TxQ & 0xFF);
        printf("Write Q On 3E");
        reciver(fd,Address,Data);

        Address=0xBF;
        Data=( TxQ>>8 )&0x3F; //      tx[2]=0x00;
        printf("Write Q On 3F");
        reciver(fd,Address,Data);

        TxQ += 0x01;
        if(TxQ == 0xFFFF)
                TxQ = 0x0000;

        TxReceive = reciver(fd,0x0c,0x00);
        Tx = reciver(fd,0x0a,00);
        TxStatusIRQ = (TxReceive & 0x01);
        TxStatusQOverflow = (TxReceive & 0x20);
        TxStatusIOverflow = (TxReceive & 0x10);
        reciver(fd,Status1_read,00);
        reciver(fd,0x0c,0x00);
        reciver(fd,0x0a,0x00);
        printf("TxReceive = %02x",TxReceive);
        printf("Tx = %02x",Tx);
        printf("fd= %02x \n",fd);

        if(TxStatusIRQ == 1){
                printf("Sent!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n");
                printf("%02x",TxReceive);
        }else{
```

```
                printf("Send Fail\n");
                printf("%02x",TxReceive);
            }
            if(TxStatusQOverflow == 1 || TxStatusIOverflow == 1){
                printf("Data Overflow Terminate TxLoop");
                exit(0);
            }
        }
        while(1){
            TxReceive = reciver(fd,0x0c,0x00);
            TxStatusIRQ = (TxReceive & 0x01);

            if(TxStatusIRQ == 1){
                printf("Sent! %02x \n",TxReceive);
            }else{
                printf("Send Fail %02x \n",TxReceive);
            }
            delay(500);
        }

}
void readstatus(int fd){
        int status;
        //int fd = *((int *)fd_ptr);
        for(int i=0; i<5 ; i++){
            uint8_t address[4]={0x3C,0x3D,0x3E,0x3F};
            tx[0]=address[i];
            tx[1]=0x00; //  tx[2]=0x00;
            msg[0].tx_buf = (unsigned long)tx;
            msg[0].rx_buf = (unsigned long)rx;
            msg[0].len =2;
            msg[0].cs_change =0;
            msg[1].delay_usecs = 0;
            msg[0].speed_hz=speed;
            msg[0].bits_per_word =8 ;
```

```
                status = ioctl(fd,SPI_IOC_MESSAGE(1),&msg);

                if(status < 0){
                        pabort("SPI_IOC_MESSAGE");
                }
                printf("sent: %02x %02x \n", tx[0], tx[1]);
                printf("rev: %02x %02x \n", rx[0], rx[1]);
        }
}
void vinput(int fd,uint8_t temp){
        send_x8(0x10);
        reciver(fd,RamDacCtrl_write,0x00);
        reciver(fd,PowerDownCtrl_write,0x01);
        reciver(fd,AuxDacData1_write,temp);
}
uint8_t map(uint8_t x,int in_min,int in_max,uint8_t out_min,uint8_t out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
void* loop3d(void *param){
        int status;
        int Tx;
        int fd = (intptr_t) param;
        send_x8(0x10);
        reciver(fd,TxSetup_write,0x98);
        reciver(fd,ConfigCtrl1_write,0x00);
        reciver(fd,RxSetup1_write,0x02);
        reciver(fd,LoopBackCtrl_write,0x06);
        reciver(fd,TxDPIData1_write,0xA3);
        while 1 {
                reciver(fd,TxDPIData1_write,0xA3);
                Tx = reciver(fd,TxDPIData1_read,00);
                printf("%x", Tx);
                }
```

```
}
void* clearloop(void *param){
        int fd = (intptr_t) param;
        send_x8(0x10);
 reciver(fd,ConfigCtrl1_write,0x08);
 reciver(fd,ConfigCtrl1_read,00);
 reciver(fd,ClkDiv2_write,0x00);
 reciver(fd,InterruptMask1_write,0x00);
 reciver(fd,TxQGain1_write,0x00);
 reciver(fd,TxQGain2_write,0x00);
 reciver(fd,TxIGain1_write,0x00);
 reciver(fd,TxIGain2_write,0x00);
 reciver(fd,TxSetup_write,0x00);
 reciver(fd,ClkStopCtrl_write,0x00);
 reciver(fd,LoopBackCtrl_write,0x00);
 reciver(fd,IRQCtrl_write,0x00);
 reciver(fd,ConfigCtrl1_write,0x00);
 reciver(fd,ConfigCtrl2_write,0x00);
 reciver(fd,IRQCtrl_write,0x00);
 reciver(fd,ClkStopCtrl_write,0x00);
 reciver(fd,TxSetup_write,0x00);     // *** Original ***
 reciver(fd,ClkDiv1_write,0x00);
 reciver(fd,ClkDiv2_write,0x00);
 reciver(fd,TxQPhase1_write,0x00);
 reciver(fd,TxIPhase1_write,0x00);
 reciver(fd,TxQGain1_write,0x00);
 reciver(fd,TxQGain2_write,0x00);
 reciver(fd,TxIGain1_write,0x00);
 reciver(fd,TxIGain2_write,0x00);
 reciver(fd,TxQOffset1_write,0x00);
 reciver(fd,TxQOffset2_write,0x00);
 reciver(fd,TxIOffset1_write,0x00);
 reciver(fd,TxIOffset2_write,0x00);
 reciver(fd,LoopBackCtrl_write,0x00);
 reciver(fd,TxSetup_write,0x00);
```

```
 reciver(fd,RxSetup1_write,0x00);
 reciver(fd,RxQOffset1_write,0x00);
 reciver(fd,RxQOffset2_write,0x00);
 reciver(fd,RxIOffset1_write,0x00);
 reciver(fd,RxIOffset2_write,0x00);
 reciver(fd,RxQGain1_write,0x00);
 reciver(fd,RxQGain2_write,0x00);
 reciver(fd,RxIGain1_write,0x00);
 reciver(fd,RxIGain2_write,0x00);
 reciver(fd,TxSetup_write,0x00);
 reciver(fd,RxSetup2_write,0x00);
 reciver(fd,RxSetup1_write,0x00);
 reciver(fd,0xBC,0x00);
 reciver(fd,0xBD,0x00);
 reciver(fd,0xBE,0x00);
 reciver(fd,0xBF,0x00);
}
int main(int argc, char *argv[]){
        pthread_t threads;
        int ret = 0;
        int fd;
        uint8_t A,D,xV;
        //float V;
        int status;
        uint8_t V;

        uint8_t temp2;
        uint8_t temp3;
        uint8_t temp4;
        //array   buffer  32768
        //index   word    0

        //mode = SPI_MODE_1 | SPI_CS_HIGH;
        mode = 0x00;
        fd = open(device, O_RDWR);
```

```
if (fd < 0)
        pabort("can't open device");

//spi mode
ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);
if (ret == -1)
        pabort("can't set spi mode");
ret = ioctl(fd, SPI_IOC_RD_MODE, &mode);
if (ret == -1)
        pabort("can't get spi mode");
// bits per word
ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
if (ret == -1)
        pabort("can't set bits per word");
ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
if (ret == -1)
        pabort("can't get bits per word");

//max speed hz
ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
if (ret == -1)
        pabort("can't set max speed hz");
ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
if (ret == -1)
        pabort("can't get max speed hz");
printf("spi mode: 0x%x\n", mode);
printf("bits per word: %d\n", bits);
printf("max speed: %d Hz (%d KHz)\n", speed, speed/1000);
while(1){
        printf("Version 1.7\n");
        printf("press 1 : Set up PA Vinput\n");
        printf("press 2 : Run initial set up for CMX981\n");
        printf("press 3 : Run initial set up for CMX998\n");
        printf("press 4 : Start TX Loop send data to FIFO\n");
        printf("press 5 : start TX Loop send data to Access Point\n");
```

```
printf("press 6 : start NEW TX Loop send data to Access Point\n");
printf("press 7 : Read Status\n");
printf("press 8 : Start Local Oscillator\n");
printf("press 9 : loop 3D\n");
printf("press 10 : clear register \n");
printf("press 0 : exit\n");
printf("please select the option : ");

scanf("%d", &status);

if(status==1){
        printf("Input temp ");
        scanf("%02x",&V);
        xV = map((V*1000),(0.15*1000),(3.245*9000),0x26,0x76);
        printf("%02x \n",xV);
        vinput(fd,V);

}
else if(status==2){
        initialCMX981(fd);
}
else if(status==3){
        initialCMX998(fd);

}
else if(status==4) {
        send_x8(0x10);
        pthread_create(&threads, NULL, txloop,(void *) (intptr_t)fd);
}
else if(status==5) {
        send_x8(0x10);
        pthread_create(&threads, NULL, txloop2,(void *) (intptr_t)fd);
}
else if(status==6) {
        send_x8(0x10);
```

```
                Txloop2new(fd);
            }
            else if(status==7) {
                readstatus(fd);
            }
            else if(status==9) {
                loop3d(fd);
            }
            else if(status==10) {
                clearloop(fd);
            }
            else{
                exit(0);
            }
    }
        return ret;

}
```