รายงานสหกิจศึกษาฉบับสมบูรณ์

การศึกษาและพัฒนาอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA
Study and Development of Transceiver Product based on TETRA Standard

นายมารุฒ กรอิ่ม
นายสุธินันท์ สิงหานุวัฒน์

ภาควิชาวิศวกรรมโทรคมนาคม
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2560

รายงานสหกิจศึกษาฉบับสมบูรณ์

การศึกษาและพัฒนาอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA

Study and Development of Transceiver Product based on TETRA Standard

นายมารุฒ กรอิ่ม

นายสุธินันท์ สิงหานุวัฒน์

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2560

ชื่อโครงงานสหกิจศึกษา การศึกษาและพัฒนาอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA
ชื่อ-สกุล นักศึกษา นายมารุฒ กรอิ่ม
นายสุธินันท์ สิงหานุวัฒน์
คณะ วิศวกรรมศาสตร์ **ภาควิชา** วิศวกรรมโทรคมนาคม
ชื่อ-สกุล อาจารย์นิเทศ ผศ.ดร.ศรวัฒน์ ชิวปรีชา
ชื่อ-สกุล ผู้นิเทศงาน ดร.สมรักษ์ เพชรชาตรี
ชื่อสถานประกอบการ บริษัท กสท โทรคมนาคม จำกัด (มหาชน)

บทคัดย่อ

เนื่องจากทางบริษัท กสท โทรคมนาคม จำกัด (มหาชน) ได้ทำการสร้าง Advance Digital Radio Baseband Board ซึ่งเป็นบอร์ดที่ใช้งานในด้านการสื่อสาร สร้างขึ้นเพื่อใช้งานภายในองค์กร รวมไปถึงเพื่อใช้งานในเชิงพาณิชย์ ในปัจจุบันอยู่ในขั้นตอนพัฒนาให้สามารถใช้งานได้จริง โดยตัวบอร์ดมีหลักการสื่อสารตั้งอยู่บนมาตรฐานของ Terrestrial Trunked Radio (TETRA) ของยุโรป และถือเป็นหนึ่งในผลิตภัณฑ์ที่ใช้เทคโนโลยี Software Defined Radio (SDR) ซึ่งเป็นเทคโนโลยีที่ถูกใช้งานอย่างแพร่หลายในปัจจุบัน โครงงานฉบับนี้จึงมีการนำเสนอข้อมูลเกี่ยวกับเทคโนโลยี SDR ที่ตั้งอยู่บนมาตรฐาน TETRA, การศึกษามาตรฐาน TETRA โดยการนำมาตรฐาน TETRA มาประยุกต์ใช้กับ Universal Software Radio Peripheral (USRP) ซึ่งเป็นหนึ่งในหลากหลายรูปแบบของเทคโนโลยี SDR รวมไปถึงการศึกษา Processor Chip จากชุด Evaluation kit ของบริษัท CML Microchip ที่เป็นส่วนประกอบหลักในการสร้างเป็น Advance Digital Radio Baseband Board และสุดท้ายเป็นกระบวนการในการนำความรู้ที่ได้ไปพัฒนาตัวบอร์ดของทางบริษัท ให้สามารถใช้งานได้จริงต่อไป

คำสำคัญ : Software Defined Radio (SDR), Terrestrial Trunked Radio (TETRA), Universal Software Radio Peripheral (USRP), Advance Digital Radio Baseband Board

Co-operative Title: Study and Development of Transceiver Product based on TETRA Standard

Student Intern Name: Mr.Marut Korn-Im

Mr.Suthinan Singhanuwat

Faculty: Engineering **Department:** Telecommunication Engineering

Advisor Name: Assf. Prof. Dr. Sorawat Chivapreecha

Mentor Name: Dr.Somrak Petchartee

Company: CAT TELECOM PUBLIC COMPANY LIMITED

ABSTRACT

Because of the Advance Digital Radio Baseband Board developed by CAT Telecom Public Company Limited for the purpose of use within the Organization and for commercial use. Currently in the development stage, this board uses Terrestrial Trunked Radio (TETRA) communication standard of European and this board is one of many platform that uses Software Defined Radio (SDR) technology, which is widely used nowadays. Therefore, this project will present information about SDR technology based on TETRA standard, including the study of TETRA standard by applying TETRA standard for use on Universal Software Radio Peripheral (USRP) which is one of many platform of SDR technology and also including the study of processor chip on Evaluation kit from CML Microsystems Plc that is the main part of Advance Digital Radio Baseband Board. All these studies are done for bring the knowledge to develop Advance Digital Radio Baseband Board.

Keyword : Software Defined Radio (SDR), Terrestrial Trunked Radio (TETRA), Universal Software Radio Peripheral (USRP), Advance Digital Radio Baseband Board

กิตติกรรมประกาศ

โครงงานฉบับนี้ สำเร็จลุล่วงไปได้ด้วยดี เพราะได้รับคำปรึกษาตลอดจนคำแนะนำจากหลายๆภาคส่วน ทางคณะผู้จัดทำขอขอบพระคุณ ดร.สมรักษ์ เพชรชาตรี ผู้จัดการฝ่ายวิจัย บริษัท กสท โทรคมนาคม จำกัด (มหาชน) ผู้เป็นพี่เลี้ยงประจำโครงงานสหกิจศึกษาฉบับนี้ ที่คอยดูแลเอาใจใส่ ให้ความรู้ รวมถึงให้ประสบการณ์ในการทำงานตลอดช่วงระยะเวลาการปฏิบัติงานอันมีค่า

ขอขอบพระคุณ ผศ.ดร.ศรวัฒน์ ชิวปรีชา ผู้เป็นอาจารย์นิเทศของโครงงานสหกิจศึกษาฉบับนี้ ที่คอยสนับสนุนให้แนวทางรวมไปถึงให้ความรู้ที่เกี่ยวข้องกับการปฏิบัติงานโครงงานสหกิจศึกษา ส่งผลให้การปฏิบัติงานมีความสะดวกราบรื่น

สุดท้ายนี้ ทางคณะผู้จัดทำขอขอบคุณเพื่อนนักศึกษาที่ร่วมงานและครอบครัว ที่คอยช่วยเป็นกำลังใจสำคัญในการฝ่าฟันอุปสรรคต่าง ๆ ตลอดมา ขอขอบพระคุณอย่างสูง ไว้ ณ ที่นี้

มารุฒ กรอิ่ม

สุธินันท์ สิงหานุวัฒน์

สารบัญ

หน้า

บทคัดย่อภาษาไทย I

บทคัดย่อภาษาอังกฤษ II

กิตติกรรมประกาศ III

สารบัญ IV

สารบัญตาราง VI

สารบัญภาพ VII

บทที่ 1 บทนำ 1

1.1 ความเป็นมาและความสำคัญ 1

1.2 วัตถุประสงค์ของการวิจัย 2

1.3 ขอบเขตของการวิจัย 2

1.4 ประโยชน์ที่คาดว่าจะได้รับ 2

บทที่ 2 แนวคิด ทฤษฎีและงานวิจัยที่เกี่ยวข้อง 3

2.1 ทฤษฎีที่เกี่ยวข้อง 3

2.2 งานวิจัยที่เกี่ยวข้อง 14

บทที่ 3 วิธีดำนินการวิจัย 36

3.1 การเลือกอุปกรณ์ที่ใช้ในการทดลอง 36

3.2 การใช้งานอุปกรณ์ 36

บทที่ 4 ผลการวิจัย 60

4.1 ผลจากการใช้งานอุปกรณ์ USRP 60

4.2 ผลการใช้งาน Advance Digital Radio Baseband Board 62

4.3 ผลการเปรียบเทียบสัญญาณเอาต์พุตที่ขา TxI 63

บทที่ 5 สรุปผลการวิจัยและข้อเสนอแนะ 70

5.1 สรุปผล 70

5.2 ข้อเสนอแนะ 70

เอกสารอ้างอิง 71

ภาคผนวก 73

ภาคผนวก ก 74

สารบัญ (ต่อ)

หน้า

ภาคผนวก ข.....95
ภาคผนวก ค.....101

สารบัญตาราง

ตารางที่ **หน้า**

3.1 ความสัมพันธ์ของข้อมูลเลขฐาน 16 หกชุดกับความถี่ที่ใช้งาน.....55
3.2 ลักษณะการเชื่อมต่อระหว่างบอร์ด EV9810 กับบอร์ด PE0002.....58

สารบัญภาพ

ภาพที่ หน้า

2.1 ตัวอย่างฮาร์ดแวร์ที่ใช้ระบบSDR (SDR Dongle และ USRP)....3
2.2 บล็อกไดอะแกรมสำหรับระบบวิทยุควบคุมโดยซอฟต์แวร์....4
2.3 หลักการพื้นฐานของ Digital Up Conversion (DUC)....4
2.4 หลักการพื้นฐานของ Digital Down Conversion (DDC)....5
2.5 แสดงตัวอย่างสัญญาณ FSK....6
2.6 แสดงตัวอย่างสัญญาณ ASK....7
2.7 ตัวอย่าง constellation diagram ของ π/4-DQPSK....7
2.8 เชื่อมต่อการสื่อสารแบบ SPI ระหว่างอุปกรณ์ Master และ Slave....8
2.9 สัญญาณขา SDA และ SCL....10
2.10 การเชื่อมต่อโดยทั่วไปสำหรับหนึ่งหรือมากกว่าหนึ่ง C – BUS slaves....10
2.11 การทำงานของ C–BUS....11
2.12 การทำงานของ Fast Serial Bus....12
2.13 บล็อกไดอะแกรมของ CMX981....15
2.14 IC CMX981....15
2.15 บล็อกไดอะแกรมของ Evaluation Kit 9810....19
2.16 บอร์ด Evaluation Kit 9810....19
2.17 บล็อกไดอะแกรมของ CMX998....20
2.18 IC CMX998....20
2.19 บล็อกไดอะแกรมของ Evaluation Kit 9980....22
2.20 บอร์ด Evaluation Kit 9980....22
2.21 บล็อกไดอะแกรมของ CMX994....23
2.22 IC CMX994....24
2.23 บล็อกไดอะแกรมของ Evaluation Kit 9942....25
2.24 บอร์ด Evaluation Kit 9942....25
2.25 บล็อกไดอะแกรมของ PE0002....26
2.26 PE0002 Evaluation Kit Interface Card....26
2.27 กล่องข้อความเริ่มต้นโปรแกรม....27

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

2.28 C – BUS Control Tab....28
2.29 C - BUS Control Extended Tab....30
2.30 The Script Handler Tab....31
2.31 บล็อกไดอะแกรมของ Si570....32
2.32 IC Si570....32
2.33 FPGA Zynq7000....34
2.34 แสดงส่วนประกอบหลักของระบบ SCADA....35
3.1 การทำงานของภาคส่ง....37
3.2 การทำงานของภาครับ....37
3.3 Block Simulink ของภาคส่ง....37
3.4 subsystem ของบล็อก Bits Generation....38
3.5 Block parameter ของ Matrix Concatenate....38
3.6 Subsystem ของ Unipolar Barker Code....39
3.7 Block parameter ของ Constant Barker Code....39
3.8 Block parameter ของ Data Type Conversion....40
3.9 Subsystem ของบล็อก Message Source....40
3.10 Block parameter ของ Bernoulli Binary Generator....41
3.11 Block parameter ของ DQPSK Modulator Baseband....41
3.12 Block parameter ของ Raised Cosine Transmit Filter....42
3.13 Block parameter ของ SDRu Transmitter....43
3.14 Simulink Block ของภาครับ....43
3.15 Subsystem ของบล็อก Receiver....44
3.16 Block parameter ของบล็อก SDRu Receiver....44
3.17 Block parameter ของ Frame Conversion....45
3.18 Block parameter ของ Raised Cosine Receive Filter....45
3.19 Block parameter ของ Unbuffer....46
3.20 Subsystem ของบล็อก Data decoding....46

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

3.21 Subsystem ของ Barker code Symbol Generation....46
3.22 Block parameter ของ Constant Barker Code....47
3.23 Subsystem ของ Phase Ambiguity Correction & Demodulation....47
3.24 Block parameter ของบล็อก Descrambler....48
3.25 Advance Digital Radio Baseband Board....49
3.26 บล็อกไดอะแกรมการรับส่งข้อมูลของ CMX981....50
3.27 บล็อกไดอะแกรมของ CMX998....51
3.28 บล็อกไดอะแกรมการทำงานของ SDR Dongle....51
3.29 หน้าต่าง PuTTY Configuration....52
3.30 หน้าต่างซอฟแวร์ PuTTY....52
3.31 ซอฟแวร์ SDRSharp....53
3.32 เมนูทั้งหมดหลังจากรันไฟล์ cmx....53
3.33 การใช้งาน press 1....54
3.34 บอร์ด Evaluation Kit 9810 (EV9810)....56
3.35 PE0002 Evaluation Kit Interface Card....56
3.36 The Script Handler Tab....57
3.37 ลักษณะการเชื่อมต่อระหว่างบอร์ด EV9810 กับบอร์ด PE0002....58
3.38 schematic ขา TxI ของบอร์ด EV9810 และ Advance Digital Radio Baseband Board....59
4.1 code ที่ใช้ในการ Generate Text ข้อมูลในภาคส่ง....60
4.2 constellation ของข้อมูลหลังจากผ่านบล็อก DQPSK Modulator Baseband....61
4.3 บล็อก Probe ที่นำมาตรวจสอบหาอัตราการส่งข้อมูล....62
4.4 Simulink Block แสดงจุดวัดสัญญาณในภาคส่งโดยใช้ Spectrum Analyzer....62
4.5 Spectrum ของข้อมูลในภาคส่ง....63
4.6 constellation ของข้อมูลก่อนผ่านบล็อก DQPSK Demodulator Baseband....63
4.7 Simulink Block แสดงจุดวัดสัญญาณในภาครับโดยใช้ Spectrum Analyzer....64
4.8 Spectrum ของข้อมูลในภาครับ....64

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

4.9 ข้อมูล Text ที่รับได้.....65
4.10 สเปกตรัมของบอร์ด Advance Digital Radio Baseband Board เมื่อรันคำสั่งส่งข้อมูล.....65
4.11 สัญญาณข้อมูลเอาต์พุตที่ขา TxI ก่อนทำการรันคำสั่งส่งข้อมูลของบอร์ด Advance Digital.....66
4.12 สัญญาณข้อมูลเอาต์พุตที่ขา TxI ก่อนเริ่มต้นคำสั่งส่งข้อมูลของบอร์ด Evaluation Kit 9810.....67
4.13 สัญญาณข้อมูลเอาต์พุตที่ขา TxI หลังจากรันคำสั่งส่งข้อมูลของบอร์ด TETRA.....68
4.14 สัญญาณข้อมูลเอาต์พุตที่ขา TxI หลังจากรันคำสั่งส่งข้อมูลของบอร์ด Evaluation Kit 9810.....68

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญ

ปัจจุบันเทคโนโลยีทางด้านการสื่อสารถูกพัฒนาขึ้นอย่างรวดเร็วและกว้างขวาง บุคลากรด้านการสื่อสารควรมีการเตรียมพร้อมสำหรับเทคโนโลยีใหม่ๆอยู่เสมอ จากการที่เทคโนโลยีเกี่ยวกับ Software Defined Radio (SDR) ซึ่งเป็นเทคโนโลยีที่มีความยืดหยุ่นและสะดวกในการนำมาใช้งานบนเครื่องรับ-ส่งวิทยุ ถูกนำมาใช้งานอย่างแพร่หลาย ส่งผลให้เกิดการประยุกต์ใช้งานเทคโนโลยี SDR ในหลากหลาย Platform ทางองค์กรเล็งเห็นถึงช่องทางในการนำเทคโนโลยี SDR มาใช้งานให้เกิดประโยชน์ต่อองค์กร จึงได้มีการนำชุด Evaluation kit ของบริษัท CML Microcircuit ซึ่งเป็นบริษัทที่ทำธุรกิจ Micro Chip เกี่ยวกับการสื่อสารมาศึกษาจนนำไปสู่การประยุกต์ใช้งานจนเกิดเป็น Advance Digital Radio Baseband Board โดยมีวัตถุประสงค์เพื่อใช้งานด้านการสื่อสารภายในองค์กรและใช้งานในเชิงพาณิชย์ โดยตัวบอร์ดยังไม่สามารถใช้งานได้จริง การที่จะทำให้ตัวบอร์ดสามารถใช้งานได้จริง ตัวบุคลากรผู้พัฒนาจำเป็นต้องมีความรู้ความสามารถในด้านการสื่อสาร

เนื่องจากการพัฒนา Advance Digital Radio Baseband Board มีความจำเป็นที่จะต้องใช้ความรู้ทางด้านการสื่อสารแบบดิจิตอล ประกอบกับตัวบอร์ดเป็นบอร์ดสำหรับใช้งานด้านการสื่อสารโดยอยู่บนมาตรฐานของ Terrestrial Trunked Radio (TETRA) มี FPGA ซึ่งทำหน้าที่เป็น Processor มีระบบปฏิบัติการเป็น Linux System ถือเป็นหนึ่งใน Platform ที่ใช้เทคโนโลยีของ SDR ดังนั้น จึงจำเป็นต้องมีการศึกษาเรียนรู้ รวมไปถึงประยุกต์ใช้เทคโนโลยี SDR จาก Platform อื่นๆ องค์กรจึงได้มีการศึกษาเทคโนโลยี SDR เพื่อเป็นความรู้ในการพัฒนา Advance Digital Radio Baseband Board จาก Platform อื่น ซึ่งก็คือ Universal Software Radio Peripheral (USRP) ซึ่งเป็น Platform ที่เป็นที่นิยมในด้านการศึกษาในปัจจุบัน โดยทางผู้จัดทำได้ทำการศึกษาโดยการประยุกต์ใช้งานมาตรฐาน TETRA บน Platform ของ USRP โดยประยุกต์ใช้งานผ่านเทคโนโลยี SDR ซอฟท์แวร์ที่ใช้งานก็คือ MATLAB @Simulink เพื่อที่จะนำความรู้ที่ได้ไปพัฒนาตัว Advance Digital Radio Baseband Board ต่อไป

1.2 วัตถุประสงค์ของการวิจัย

1) เพื่อศึกษาเรียนรู้เกี่ยวกับระบบการสื่อสารแบบดิจิตอล

2) สามารถประยุกต์ใช้งานอุปกรณ์รับส่งข้อมูลโดยอยู่บนมาตรฐาน TETRA ได้

3) เพื่อพัฒนาอุปกรณ์รับส่งข้อมูลให้กับทางบริษัท

1.3 ขอบเขตของการวิจัย

1) มีความรู้เกี่ยวกับระบบการสื่อสารแบบดิจิตอล

2) สามารถรับส่งข้อมูลบนมาตรฐาน TETRA โดยใช้อุปกรณ์รับส่งข้อมูลในระบบ SDR ได้

3) มีความรู้ความสามารถที่เกี่ยวข้องกับการพัฒนาอุปกรณ์รับส่งข้อมูลให้กับสถานประกอบการ

1.4 วิธีดำเนินการวิจัย

1) ศึกษากระบวนการในระบบการสื่อสารแบบดิจิตอล

2) ศึกษาการใช้งานอุปกรณ์ที่ใช้ในระบบสื่อสารแบบดิจิตอลบนเทคโนโลยี SDR

3) ทดลองรับส่งข้อมูลโดยใช้อุปกรณ์ที่ศึกษา

4) ประยุกต์ใช้มาตรฐาน TETRA ในการรับส่งข้อมูล

5) นำความรู้จากการที่ได้ศึกษามาพัฒนาอุปกรณ์รับส่งข้อมูลให้กับทางสถานประกอบการ

1.5 ประโยชน์ที่คาดว่าจะได้รับ

- ได้นำความรู้ทฤษฎีที่ได้จากห้องเรียนมาประยุกต์ใช้งานจริง
- ได้รับประสบการณ์การทำงานจริงจากสถานประกอบการ
- มีความตรงต่อเวลาในการทำงาน
- มีความรับผิดชอบต่อการทำงาน
- มีสัมพันธไมตรีที่ดีต่อบุคลากรในสถานประกอบการ
- ได้ฝึกความอดทนต่อแรงกดดันจากการทำงาน

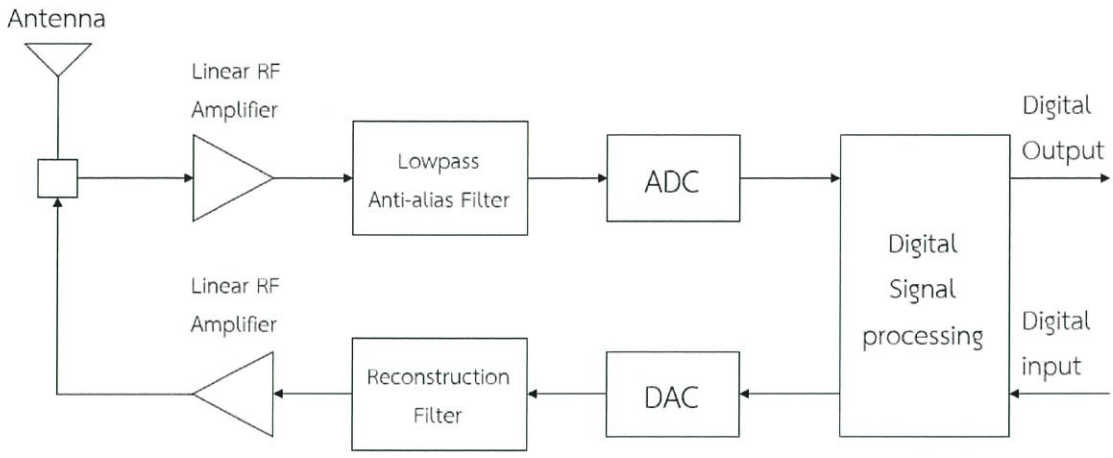
บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 วิทยุควบคุมโดยซอฟต์แวร์ (software-Defined Radio)

วิทยุควบคุมโดยซอฟต์แวร์ (Software-Defined Radio) หรือเรียกโดยย่อว่า SDR เป็นระบบสื่อสารวิทยุที่สามารถเปลี่ยนแปลงรูปแบบของสัญญาณหรือการสื่อสารได้โดยซอฟแวร์ (Software) ระบบ SDR อาจประกอบด้วยฮาร์ดแวร์ (Hardware) ที่ทำการรับส่งสัญญาณที่สามารถทำการเปลี่ยนแปลงหรือแก้ไข (Reconfigurable) ซ้ำๆได้ โดยการโปรแกรมลงบนซอฟแวร์ (Software) ตัวอย่างของฮาร์ดแวร์ที่ใช้ระบบ SDR คือ USRP (Universal Software Radio Peripheral) โดยตัวเครื่อง USRP จะใช้ Field Gate Programmable Array (FPGA) ในการประมวลผล ทำให้มีความยืดหยุ่นสูงสามารถนำมาปรับแต่งให้เป็นระบบที่ต้องการจะทดสอบได้ง่าย

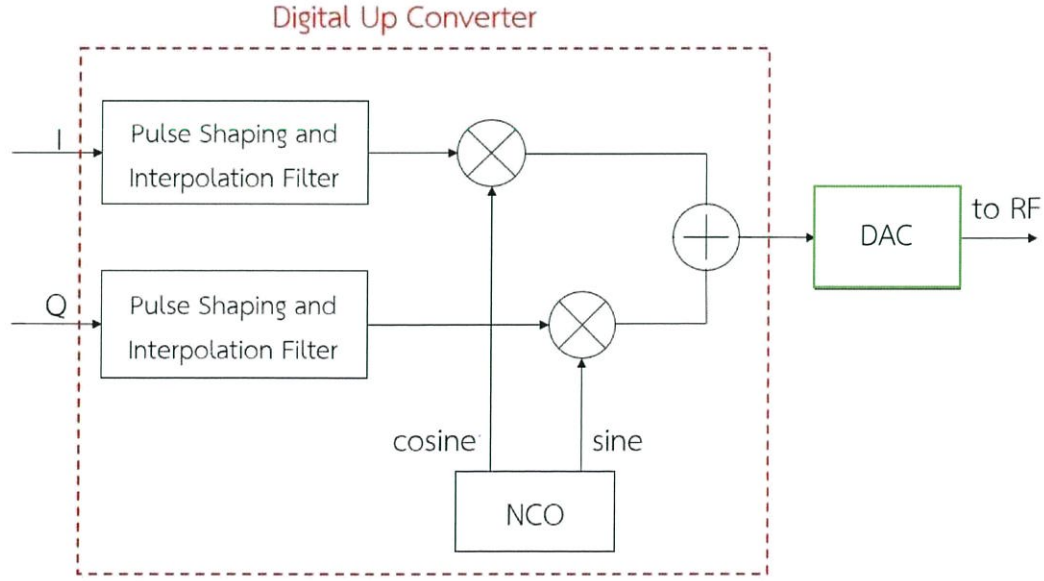
รูปที่ 2.1 ตัวอย่างฮาร์ดแวร์ที่ใช้ระบบ SDR (SDR Dongle และ USRP)



รูปที่ 2.2 บล็อกไดอะแกรมสำหรับระบบวิทยุควบคุมโดยซอฟต์แวร์

2.1.1.2 Digital Up Converter (DUC)

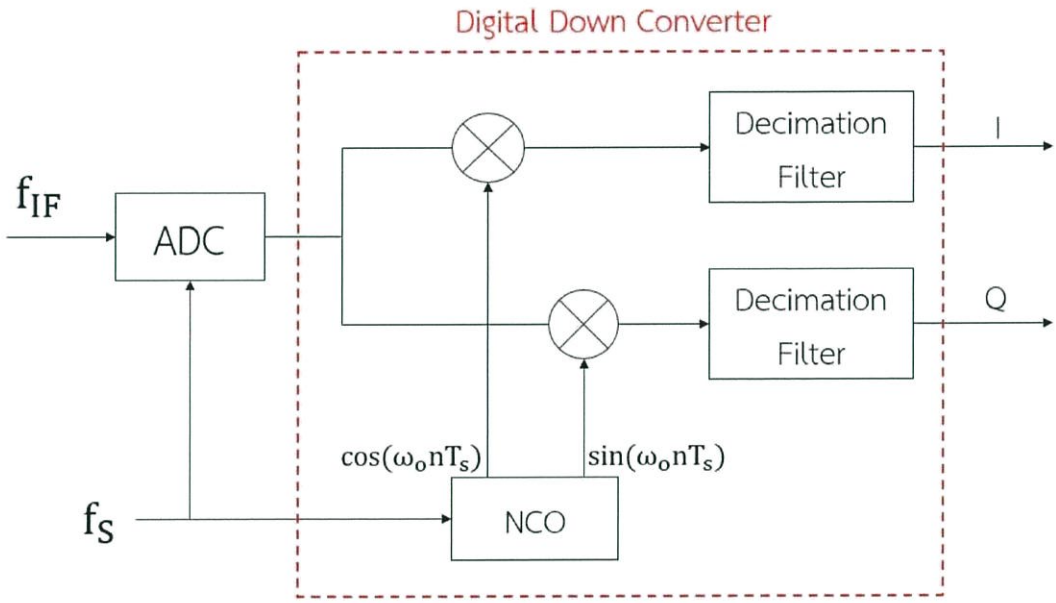
ในการใช้งานที่ต้องการควบคุมสัญญาณคลื่นความถี่วิทยุ (Radio Frequency : RF) จำเป็นต้องแปลงสัญญาณดิจิตอลเบสแบนด์ (digital baseband signal) ให้เป็นสัญญาณพาสแบนด์ (real passband signal) โดยในอุดมคตินั้น DAC (Digital-to-Analog Converter) สามารถสร้างสัญญาณคลื่นความถี่วิทยุได้โดยตรง อย่างไรก็ตาม DAC ในปัจจุบันยังไม่สามารถตอบสนองความต้องการในเรื่องของความแม่นยำและความเร็วสำหรับสัญญาณคลื่นความถี่วิทยุในช่วงหลายร้อยเมกะเฮิรตซ์ได้ ดังนั้นจึงจำเป็นต้องมี DUC เพื่อให้สามารถตอบสนองความต้องการเหล่านี้ได้



รูปที่ 2.3 หลักการพื้นฐานของ Digital Up Conversion (DUC)

2.1.1.3 Digital Down Converter (DDC)

เป็นเทคนิคในการลดอัตราการส่งข้อมูลโดยยังรักษาข้อมูลที่ถูกต้องเอาไว้ บางครั้งเรียกว่า Digital Drop Receiver (DDR) โดยแถบความถี่ที่เราสนใจจะอยู่รอบๆ ความถี่กลาง (intermediate frequency) จากทฤษฎีของไนควิสต์ (Nyquist theorem) กล่าวว่าอัตราการสุ่มตัวอย่าง (Sampling rate) ของสัญญาณที่มีแบนด์จำกัด (band-limited signal) อย่างน้อย ต้องเป็นสองเท่าของความถี่ข้อมูลเพื่อให้สามารถสร้างสัญญาณกลับมาได้ ทำให้นำไปสู่อัตราการส่งข้อมูลที่สูงและต้องมีการประมวลผลขั้นสูง โดย DDC จะทำให้สามารถลดอัตราการส่งข้อมูล ทำให้เราสามารถประมวลผลได้ง่ายขึ้น



รูปที่ 2.4 หลักการพื้นฐานของ Digital Down Conversion (DDC)

2.1.1.4 NCO (Numerically Controlled Oscillator)

เป็นเครื่องกำเนิดสัญญาณแบบดิจิตอลที่สร้างการซิงโครนัส (synchronous) เช่นสัญญาณคล็อก (clock signal) โดยส่วนใหญ่ NCO จะถูกนำมาใช้ในระบบสื่อสารจำนวนมากรวมทั้ง Digital Up/Down Converter ที่ใช้ในระบบวิทยุไร้สายและซอร์ฟแวร์ระบบดิจิตอล

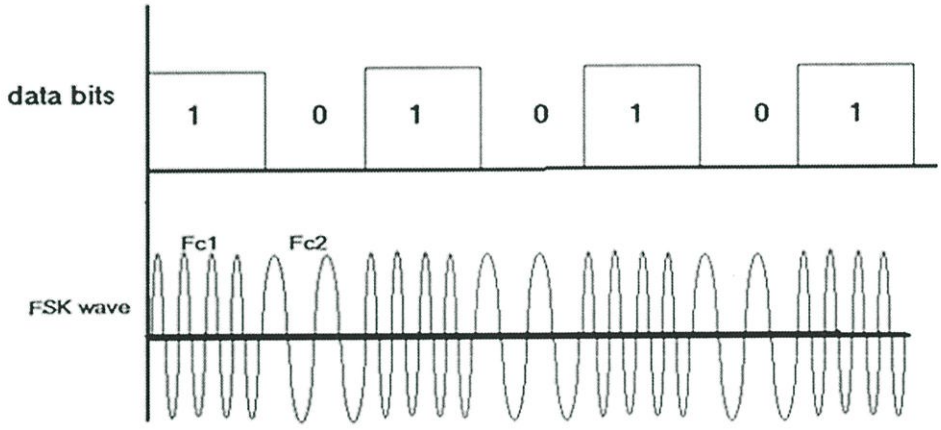
2.1.2 การมอดูเลตสัญญาณดิจิตอล (Digital Modulation)

ในการมอดูเลตสัญญาณดิจิตอล คลื่นพาห์แบบอนาล็อกจะถูกมอดูเลตโดยสัญญาณที่ไม่ต่อเนื่อง วิธีการมอดูเลตแบบดิจิตอลถือได้ว่าเป็นการแปลงดิจิตอลให้เป็นอนาล็อกและการดีมอดูเลต (demodulation) ที่สอดคล้องก็คือการแปลงอนาล็อกเป็นดิจิตอล การเปลี่ยนแปลงของสัญญาณ

คลื่นพาห์จะถูกเลือกจากจำนวนที่แน่นอนของสัญลักษณ์ (symbol) M (อักษรที่ถูกมอดูเลต) โดยที่ตัวอักษรประกอบด้วย $M = 2^N$ สัญลักษณ์ทางเลือก แต่ละสัญลักษณ์แทนหนึ่งข้อความที่ประกอบด้วย N บิต ถ้าอัตราสัญลักษณ์หรืออัตราบอด (baud rate) เป็น f_S สัญลักษณ์/วินาที ทำให้อัตราข้อมูลหรืออัตราบิต (bit rate) เป็น Nf_S บิต/วินาที ตัวอย่างเช่น ตัวอักษรที่ประกอบด้วย 16 สัญลักษณ์ แต่ละสัญลักษณ์ถูกแทนค่าด้วย 4 บิต ดังนั้นอัตราข้อมูลจึงเป็นสี่เท่าของอัตราสัญลักษณ์ โดยเทคนิคของการมอดูเลตแบบดิจิตอลที่ใช้กันทั่วไป ได้แก่ FSK (Frequency-shift keying) , ASK (Amplitude-shift keying) , PSK (Phase-shift keying) และ QAM (Quadrature amplitude modulation)

2.1.2.1 การมอดูเลตทางความถี่ (Frequency -shift keying: FSK)

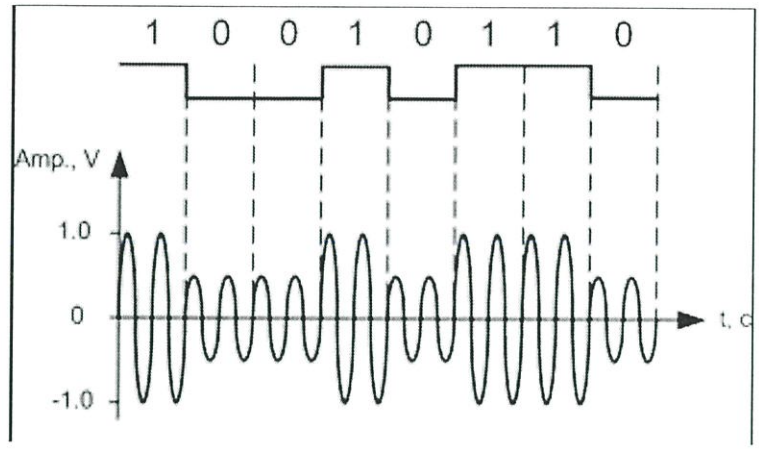
พื้นฐานการมอดูเลตสัญญาณแบบ FSK (Frequency Shift Keying) นั้นได้มาจากการมอดูเลตแบบ FM ซึ่งเป็นการเปลี่ยนเฉพาะความถี่ของสัญญาณคลื่นพาห์ตามการเปลี่ยนของบิตข้อมูล โดยทั่วไปสามารถแบ่งย่อยได้อีก ได้แก่ Audio frequency-shift keying (AFSK) , Multi-frequency shift keying (M-ary FSK or MFSK)



รูปที่ 2.5 แสดงตัวอย่างสัญญาณ FSK

2.1.2.2 การมอดูเลตทางแอมพลิจูด (Amplitude -shift keying: ASK)

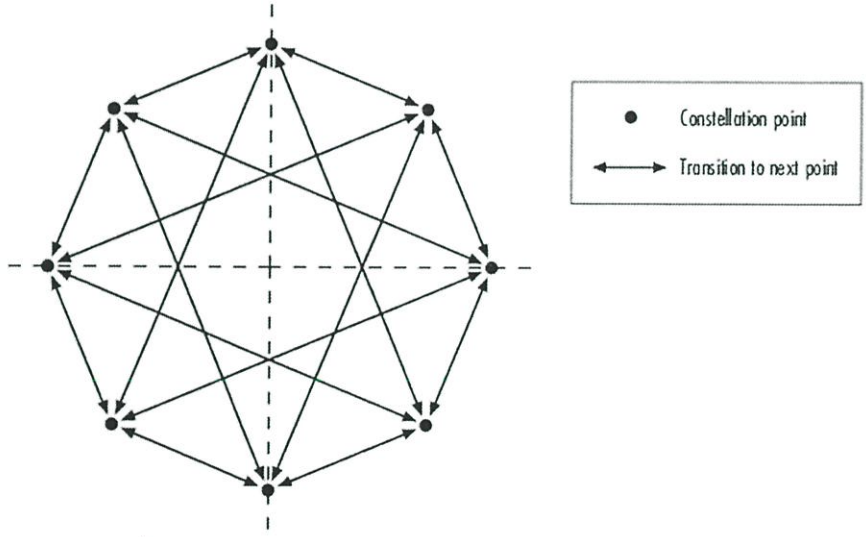
การมอดูเลตแบบ ASK นั้นมีพื้นฐานมาจากการมอดูเลตแบบ AM แต่อาศัยสัญญาณดิจิตอล (0,1) เข้ามาควบคุมในการมอดูเลต หรืออาจกล่าวได้ว่าสัญญาณมี 2 ระดับ คือ high และ low โดยเป็นการเปลี่ยนเฉพาะแอมพลิจูดของสัญญาณคลื่นพาห์ ซึ่งถ้ากำหนดให้ high เท่ากับ 1 และ low เท่ากับ 0 จะเรียกว่า ON/OFF keying



รูปที่ 2.6 แสดงตัวอย่างสัญญาณ ASK

2.1.2.3 การมอดูเลตทางเฟส (Phase-shift keying: PSK)

เป็นการเปลี่ยนเฉพาะเฟสของสัญญาณคลื่นพาห์ตามการเปลี่ยนของบิตข้อมูล โดยทั่วไปสามารถแบ่งย่อยได้อีก ได้แก่ Binary PSK (BPSK : M=2) , Quadrature PSK (QPSK : M=4) , 8PSK (M=8) , 16PSK (M=16) , Differential PSK (DPSK) , Differential QPSK (DQPSK) , Offset QPSK (OQPSK) , π/4-QPSK , π/4-DQPSK

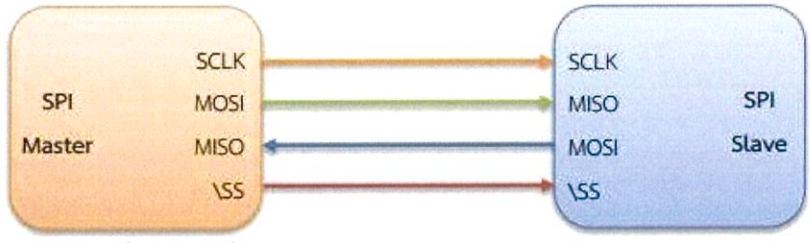


รูปที่ 2.7 ตัวอย่าง constellation diagram ของ π/4-DQPSK

โดยการมอดูเลตแบบ π/4-DQPSK จะแตกต่างกับการมอดูเลตแบบ DQPSK โดยที่การมอดูเลตแบบ π/4-DQPSK จะมีสัญลักษณ์ (symbol) ที่ถูกกำหนดโดยการเปลี่ยนเฟสจากสัญลักษณ์ก่อนหน้าไป π /4 ดังนั้นจำนวนจุดบน constellation diagram ของการมอดูเลตแบบ π/4-DQPSK จะมีทั้งหมดเท่ากับ 8 จุด ดังรูปที่ 2.5 และยังเป็นสัญญาณ Differential อีกด้วย

2.1.3 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 กับอุปกรณ์ต่อพ่วงอื่นๆเช่น Serial EEPROMs, Shift Register, Sensor, Display Driver, A/D Converters และ SD CARD เป็นต้น



รูปที่ 2.8 เชื่อมต่อการสื่อสารแบบ SPI ระหว่างอุปกรณ์ Master และ Slave

การเชื่อมต่อการสื่อสารแบบ SPI ระหว่างอุปกรณ์ Master และ Slave โดยมีสายสัญญาณทั้งหมด 4 เส้น หรือ Four Wire ประกอบด้วย

- 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

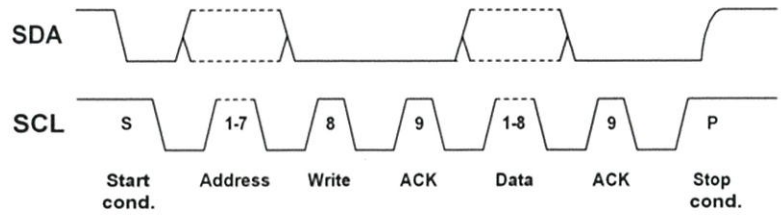
อุปกรณ์ Master ทำหน้าที่เป็นตัวควบคุมการสื่อสารทั้งหมด โดยควบคุมการสื่อสารตามสัญญาณนาฬิกา และสายสัญญาณ SS ตัวมาสเตอร์จะเป็นตัวที่ตัดสินใจเลือก รับ หรือ ส่งข้อมูลไปยังอุปกรณ์ Slave สัญญาณเส้น SS หรือ Slave select ในกรณี ที่มีตัว Slave มากกว่า 1 ตัว โดยการทำให้เส้น SS มีระดับสัญญาณเป็น Low เมื่อต้องการติดต่อกับ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

ตัวใด ก็เพียงทำให้สัญญาณ SS ของ Slave ตัวนั้น มีระดับสัญญาณเป็น Low 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 หลายตัว การสื่อสารแบบ SPI ต้องใช้สายสัญญาณมากกว่า I2C

2.1.4 Inter-Integrated Circuit (I2C)

การติดต่อสื่อสารระหว่างไอซีโดยบัส I2C ได้รับการพัฒนาโดยฟิลิปส์ (Philips) ด้วยจุดมุ่งหมายหลักคือ ต้องการให้ไอซีหรือโมดูลสามารถติดต่อ สั่งงานและควบคุมภายใต้สายสัญญาณเพียง 2 เส้น โดยเส้นหนึ่งคือสายสัญญาณนาฬิกาที่ใช้กำหนดจังหวะการทำงาน การต่อร่วมกันของอุปกรณ์บนบัส I2C ทำได้ง่ายมากเพียงต่อสายข้อมูลและสายสัญญาณนาฬิกาของอุปกรณ์แต่ละตัวขนานหรือพ่วงกันไป ส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่ออุปกรณ์แต่ละตัว จะใช้รหัสข้อมูลและการกำหนดสภาวะโลจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว สายข้อมูลบนบัส I2C มีชื่อเรียกอย่างเป็นทางการว่าสายข้อมูลอนุกรมหรือ SDA (Serial Data line) ส่วนสายสัญญาณนาฬิกามีชื่อเรียกว่าสายสัญญาณนาฬิกาอนุกรมหรือ SCL (Serial Clock line) คุณสมบัติโดยทั่วไปของบัส I2C คือสาย SDA และ SCL เป็นสายสัญญาณ 2 ทิศทาง (bi-directional line) ต้องมีการต่อตัวต้านทานพลูอัพกับแรงดัน +5V ไว้ตลอดเวลาเพื่อให้สายมีสถานะลอจิกสูงในขณะที่ไม่มีการติดต่อใช้งาน ทั้งยังช่วยป้องกันสัญญาณรบกวนที่อาจมีเข้ามาในสายสัญญาณทั้งสอง วงจรเอาต์พุตของอุปกรณ์ที่ต่ออยู่บนบัส I2C ต้องมีลักษณะเป็นวงจรเดรนเปิด (Open-drain) หรือคอลเล็กเตอร์เปิด(Open-collector)

หลักการของบัส I2C ประกอบด้วยสายสัญญาณ 2 เส้นคือ SDA และ SCL อุปกรณ์ที่ต่อพ่วงบนบัสสามารถมีได้มากมาย ดังนั้นจึงต้องมีการกำหนดรูปแบบของการติดต่อบนบัสเพื่อให้ผู้ใช้งานทราบว่าขณะนี้อุปกรณ์ใดติดต่อกันอยู่และอุปกรณ์ใดเป็นตัวรับหรือส่ง โดยอุปกรณ์ที่เป็นผู้สร้างข้อมูลหรือส่งข้อมูล เรียกว่าตัวส่ง (transmitter) อุปกรณ์ที่เป็นผู้รับข้อมูลเรียกว่าตัวรับ (receiver) อุปกรณ์บนบัส I2C สามารถเป็นได้ทั้งตัวรับและส่ง บางอุปกรณ์ทำหน้าที่เป็นตัวรับอย่างเดียว จะไม่มีอุปกรณ์ใดบนบัส I2C ที่ทำหน้าที่เป็นตัวส่งอย่างเดียว อุปกรณ์ที่ทำหน้าที่ควบคุมจังหวะการติดต่อบนบัส I2C เรียกว่ามาสเตอร์ (master) อุปกรณ์ที่ถูกควบคุมหรืออุปกรณ์ที่ต่อพ่วงเข้าไปบนบัส I2C เรียกว่า สเลฟ (slave) สัญญาณ SDA และ SCL แสดงในรูปที่ 2.9

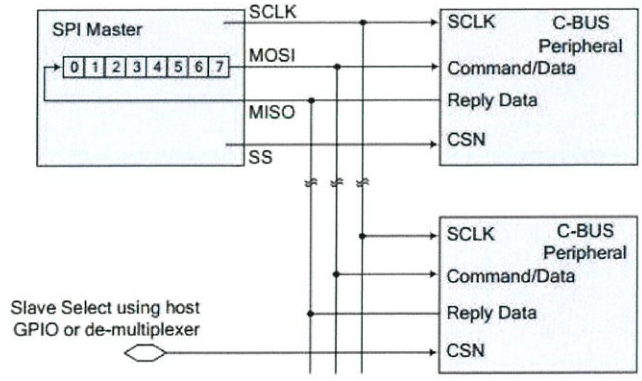


รูปที่ 2.9 สัญญาณขา SDA และ SCL

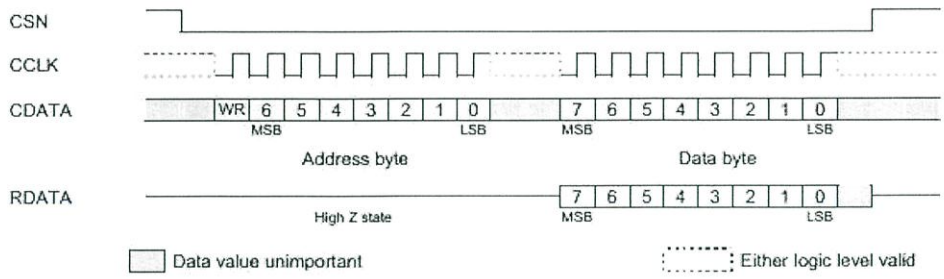
2.1.5 C – BUS

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



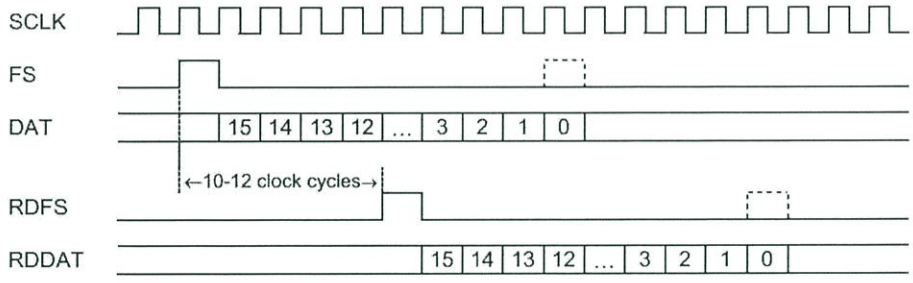
รูปที่ 2.10 การเชื่อมต่อโดยทั่วไปสำหรับหนึ่งหรือมากกว่าหนึ่ง C – BUS slaves



รูปที่ 2.11 การทำงานของ C–BUS

2.1.6 Fast Serial Bus (FSB)

เป็นการสื่อสารรูปแบบหนึ่ง ซึ่งทางบริษัท CML ที่ผลิต IC CMX981 ได้สร้างขึ้นเพื่อใช้ในการอ่านและเขียนค่ารีจีสเตอร์ภายใน IC CMX981 ซึ่งจะสามารถอ่านและเขียนได้เร็วกว่า C-BUS โดยมีรูปแบบการทำงานดังรูปที่ 2.12



รูปที่ 2.12 การทำงานของ Fast Serial Bus

2.1.7 ลีนุกซ์ (Linux)

ลีนุกซ์ถือกำเนิดขึ้นในฟินแลนด์ ปี ค.ศ. 1980 โดยลีนุส โทรวัลด์ส (Linus Trovalds) นักศึกษาภาควิชาวิทยาการคอมพิวเตอร์ (Computer Science) ในมหาวิทยาลัยเฮลซิงกิ ลีนุส เห็นว่าระบบมินิกซ์ (Minix) ที่เป็นระบบยูนิกซ์บนพีซีในขณะนั้น ซึ่งทำการพัฒนาโดย ศ.แอนดรูว์ ทาเนนบาวม์ (Andrew S. Tanenbaum) ยังมีความสามารถไม่เพียงพอแก่ความต้องการ จึงได้เริ่มต้นทำการพัฒนาระบบยูนิกซ์ของตนเองขึ้นมา โดยจุดประสงค์อีกประการ คือต้องการทำความเข้าใจในวิชาระบบปฏิบัติการคอมพิวเตอร์ด้วยเมื่อเขาเริ่มพัฒนาลีนุกซ์ไปช่วงหนึ่งแล้ว เขาก็ได้ทำการชักชวนให้นักพัฒนาโปรแกรมอื่นๆมาช่วยทำการพัฒนาลีนุกซ์ ซึ่งความร่วมมือส่วนใหญ่ก็จะเป็นความร่วมมือผ่านทางอินเทอร์เนต ลีนุสจะเป็นคนรวบรวมโปรแกรมที่ผู้พัฒนาต่างๆได้ร่วมกันทำการพัฒนาขึ้นมาและแจกจ่ายให้ทดลองใช้เพื่อทดสอบหาข้อบกพร่อง ที่น่าสนใจก็คืองานต่างๆเหล่านี้ผู้คนทั้งหมดต่างก็ทำงานโดยไม่คิดค่าตอบแทน และทำงานผ่านอินเทอร์เนตทั้งหมด ปัจจุบันเวอร์ชันล่าสุดของระบบลีนุกซ์ที่ได้

ประกาศออกมาคือเวอร์ชั่น 2.0.13 ข้อสังเกตในเรื่องเลขรหัสเวอร์ชันนี้ก็คือ ถ้ารหัสเวอร์ชันหลังทศนิยมตัวแรกเป็นเลขคู่เช่น 1.0.x,1.2.x เวอร์ชันเหล่านี้จะถือว่าเป็นเวอร์ชันที่เสถียรแล้วและมีความมั่นคงในระดับหนึ่ง แต่ถ้าเป็นเลขคี่เช่น 1.1.x, 1.3.x จะถือว่าเป็นเวอร์ชันทดสอบ ซึ่งในเวอร์ชันเหล่านี้จะมีการเพิ่มเติมความสามารถใหม่ๆลงไป และยังต้องทำการทดสอบหาข้อผิดพลาดต่างๆอยู่

ลีนุกซ์ เป็นระบบปฏิบัติการเช่นเดียวกับ ดอส ไมโครซอฟต์วินโดวส์ หรือยูนิกซ์ โดยลีนุกซ์นั้นจัดว่าเป็นระบบปฏิบัติการยูนิกซ์ประเภทหนึ่ง โดยลีนุกซ์ได้รับความนิยมจากความสามารถของตัวระบบปฏิบัติการและโปรแกรมประยุกต์ที่ทำงานบนระบบลีนุกซ์ โดยเฉพาะอย่างยิ่งโปรแกรมในตระกูลของ GNU (GNU's Not UNIX) และสิ่งที่สำคัญที่สุดก็คือระบบลีนุกซ์เป็นระบบปฏิบัติการประเภทฟรีแวร์ (Free Ware) คือไม่เสียค่าใช้จ่ายในการซื้อโปรแกรม

ระบบลีนุกซ์ตั้งแต่เวอร์ชั่น 4 นั้น สามารถทำงานได้บนซีพียูทั้ง 3 ตระกูล คือบนซีพียูของอิลเทล (PC Intel) ดิจิตอลอัลฟาคอมพิวเตอร์ (Digital Alpha Computer) และซันสปาร์ค (SUN SPARC) เนื่องจากใช้เทคโนโลยีที่เรียกว่า RPM (Red Hat Package Management) ถึงแม้ว่าในขณะนี้ลีนุกซ์ยังไม่สามารถแทนที่ไมโครซอฟต์ วินโดวส์ บนพีซีหรือแมคโอเอส (Mac OS) ได้ทั้งหมดก็ตาม แต่ผู้ใช้จำนวนไม่น้อยที่หันมาใช้และช่วยพัฒนาโปรแกรมประยุกต์บนลีนุกซ์กัน และเรื่องของการดูแลระบบลีนุกซ์นั้น ภายในระบบลีนุกซ์เองมีเครื่องมือช่วยสำหรับดำเนินการให้สะดวกยิ่งขึ้น

ปัจจุบันได้มีการนำระบบปฏิบัติการลีนุกซ์ไปประยุกต์เป็นระบบปฏิบัติการสำหรับงานด้านต่างๆเช่นงานด้านการคำนวณทางวิทยาศาสตร์ใช้เป็นสถานีงาน สถานีบริการ อินเทอร์เน็ต อินทราเน็ต หรือใช้ใน การเรียนการสอนและการทำวิจัยทางคอมพิวเตอร์ใช้พัฒนาโปรแกรมเนื่องจาก 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

2.1.8 ภาษาซี (C Language)

Programming Language C หรือ C Language (ภาษาซี) เป็นภาษาคอมพิวเตอร์ที่ใช้สำหรับพัฒนาโปรแกรมทั่วไป ถูกพัฒนาโดยเดนนิส ริสชี่ (Dennis Ritchie) เมื่อประมาณต้นปี ค.ศ. 1970 เพื่อใช้งานบนระบบปฏิบัติการยูนิกส์ แทนภาษาแอสเซมบลี ซึ่งเป็นภาษาระดับต่ำที่สามารถกระทำในระบบฮาร์ดแวร์ได้ด้วยความรวดเร็ว แต่จุดอ่อนของภาษาแอซเซมบลีก็คือความยุ่งยากในการโปรแกรมความเป็นเฉพาะตัว และความแตกต่างกันไปในแต่ละเครื่อง ต่อมาถูกนำไปใช้ในระบบปฏิบัติการต่าง ๆ จนถูกใช้เป็นภาษาพื้นฐานสำหรับภาษาอื่น เช่น ภาษาจาวา (Java) ภาษาพีเอชพี(PHP) ภาษาซีชาร์ป (C#) ภาษาซีพลัสพลัส (C++) ภาษาเพิร์ล (Perl) ภาษาไพธอน (Python) หรือภาษารูบี้ (Ruby) ภาษาซีเป็นภาษาเขียนโปรแกรมระบบเชิงคำสั่ง (หรือเชิงกระบวนงาน) ถูกออกแบบขึ้นเพื่อใช้แปลด้วย

ตัวแปลโปรแกรมแบบการเชื่อมโยงที่ตรงไปตรงมา สามารถเข้าถึงหน่วยความจำในระดับล่าง ภาษา C แม้จะเป็นภาษาระดับสูง แต่ก็สามารถใช้เป็นภาษาเครื่องได้เป็นอย่างดี

2.1.8.1 โครงสร้างของโปรแกรมภาษาซี

โปรแกรมในภาษาซีทุกโปรแกรมจะประกอบด้วยฟังก์ชันอย่างน้อย หนึ่งฟังก์ชัน คือ ฟังก์ชัน main โดยโปรแกรมภาษาซีจะเริ่มทำงานที่ฟังก์ชัน main ก่อน ในแต่ละฟังก์ชันจะประกอบด้วย

1) Function Heading ประกอบด้วยชื่อฟังก์ชัน และอาจมีรายการของ argument (บางคนเรียก parameter) อยู่ในวงเล็บ

2) Variable Declaration ส่วนประกาศตัวแปร สำหรับภาษาซี ตัวแปรหรือค่าคงที่ทุกตัว ที่ใช้ในโปรแกรมจะต้องมีการประกาศก่อนว่าจะใช้งานอย่างไร จะเก็บค่าในรูปแบบใดเช่น integer หรือ real number

3) Compound Statements ส่วนของประโยคคำสั่งต่างๆ ซึ่งแบ่งเป็นประโยคเชิงซ้อน (compound statement) กับ ประโยคนิพจน์ (expression statement) โดยประโยคเชิงซ้อนจะอยู่ภายในวงเล็บปีกกาคู่หนึ่ง { และ } โดยในหนึ่งประโยคเชิงซ้อน จะมีประโยคนิพจน์ที่แยกจากกันด้วยเครื่องหมาย semicolon (;) หลายๆ ประโยครวมกัน และ อาจมีวงเล็บปีกกาใส่ประโยคเชิงซ้อนย่อย

2.1.8.2 ข้อดีของภาษาซี

1) ภาษาซี สามารถนำไปใช้ได้บนเครื่องทุก platform ไม่ว่าจะเป็น Intel PC ที่ run Windows 95 หรือ Windows NT, Windows XP, Windows 7 หรือ แม้แต่ลีนุกซ์ ทั้งเครื่อง Macintosh และ เครื่องเวอร์คสเตชัน ตลอดจนเมนเฟรม เนื่องจากมี compiler ของภาษาซี อยู่ทั่วไป

2) ภาษาซี เป็นภาษาที่ง่ายๆ คือมีแต่ข้อกำหนดในการใช้งาน หรือ Syntax แต่ไม่มีฟังก์ชันสำเร็จรูป (Built-in Function) ใดๆ ดังนั้นหากผู้ใช้ต้องการทำอะไรก็ตาม ต้องเขียนทุกอย่างขึ้นเอง หรือ อาจเรียก Library Functions มาใช้งาน โดย ฟังก์ชันที่เป็นงานที่ใช้บ่อยๆ จะถูกรวบรวมไว้ใน Library Functions เช่น การจัดการข้อความ การดำเนินการเกี่ยวกับ Input / Output (I/O) การจองหน่วยความจำ (Memory Allocation) จะไม่มีใน Standard Library เช่น ฟังก์ชันที่จัดการ Graphics ทั้งนี้จะขึ้นกับระบบที่ใช้ (เช่น เป็นระบบ UNIX หรือ Windows 95) และ สิ่งแวดล้อมในการทำงาน (เช่น GUI เป็น X-Windows หรือ Direct X) การทำเช่นนี้จะทำให้ภาษาซี เป็นภาษาที่เคลื่อนย้ายได้ง่าย (portable)

2.2 งานวิจัยที่เกี่ยวข้อง

2.2.1 มาตรฐาน TETRA (Terrestrial Trunked Radio)

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 โดยมีการมอดูเลตแบบ $\pi/4$ differential quadrature phase-shift keying (π/4-DQPSK) ซึ่งมี symbol (baud) rate เท่ากับ 18 kSymbol/sec โดยแต่ละ symbol แทนด้วย 2 บิต เพราะฉะนั้นมีอัตราการรับส่งข้อมูลด้วยความเร็ว 36 kbps ซึ่งมีความปลอดภัยเพราะเป็นการสื่อแบบ Point-to-Point และเป็นการเข้ารหัสแบบ end-to-end

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

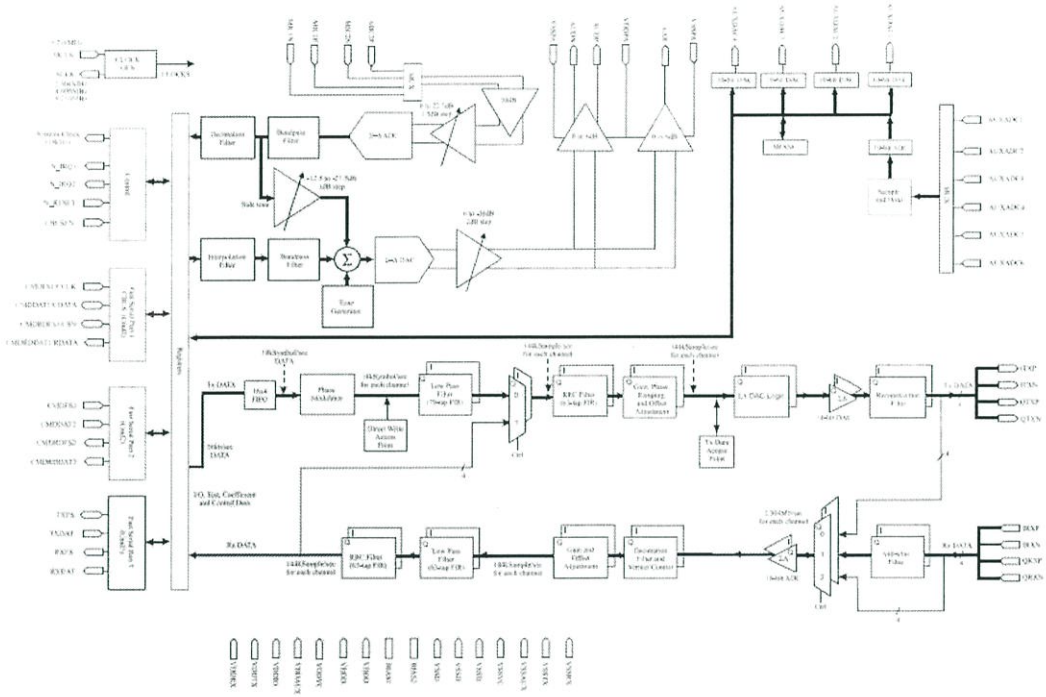
ข้อเสียของมาตรฐาน TETRA

1. ต้องใช้เครื่องขยายเชิงเส้นเพื่อให้เป็นไปตามข้อกำหนดของ RF
2. การถ่ายโอนข้อมูลได้ช้ากว่ามาตรฐานปัจจุบัน

2.2.2 CMX981 Advanced Digital Radio Baseband Processor

โปรเซสเซอร์เป็นตัวแปลงสัญญาณและตัวประมวลผลแบบผสมผสานที่ใช้เชื่อมต่อส่วนระบบอนาล็อกและดิจิตอลของระบบวิทยุดิจิตอลและทำหน้าที่สำคัญอย่างยิ่งสำหรับ DSP (Digital signal processing) รองรับการทำงานพื้นฐานมากมาย เช่น TETRA, APCO25, RCR-39 (Japan) เป็นต้น CMX981 ประกอบด้วยฟังก์ชันทั้งหมดที่จำเป็นในการแปลงข้อมูลสัญลักษณ์ดิจิทัลเป็นสัญญาณอนาล็อก I และ Q ซึ่งรวมถึงการควบคุมเอาท์พุตของดิจิทัล การชดเชยเอาท์พุตและตัวกรองแบบดิจิทัลที่สามารถตั้งค่าได้ มีการมอดูเลตแบบ Differential Quadrature Phase Shift Keying (DQPSK) มีการรับสัญญาณอินพุต I

และ Q โดยการกรองข้อมูลในช่องสัญญาณดิจิทัลเพื่อลดความยุ่งยากในการประมวลผลและการดึงข้อมูล สามารถตั้งค่าการแก้ไขชดเชยดิจิทัลและตัวกรองแบบดิจิทัล และยังมี Auxiliary DAC และ ADC ใช้ควบคุมและวัดสัญญาณคลื่นวิทยุ RF ซึ่งรวมถึง AFC, AGC และ RSSI สามารถรับ แปลงและเข้ารหัสสัญญาณเสียงได้โดยแสดงบล็อกไดอะแกรมของ CMX981 ดังรูปที่ 2.13 และแสดง IC CMX981 ดังรูปที่ 2.14



รูปที่ 2.13 บล็อกไดอะแกรมของ CMX981

รูปที่ 2.14 IC CMX981

รายละเอียดของ IC CMX981 มีดังนี้

- 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)

รีจีสเตอร์ (Registers) ทั้งหมดของ IC CMX981 มีดังนี้

Control and Set-up Registers

- รีจิสเตอร์ 00 Configuration control register 1	Read/write
- รีจีสเตอร์ 01 Configuration control register 2	Read/write
- รีจีสเตอร์ 02 Interrupt control register	Read/write
- รีจีสเตอร์ 03 Transmit set-up register	Read/write
- รีจีสเตอร์ 04-07 Transmit data FIFO register	write only
- รีจีสเตอร์ 08 Receive set-up register 1	Read/write
- รีจีสเตอร์ 09 Receive set-up register 2	Read/write

Status and Interrupt Registers

- รีจีสเตอร์ 0A Status register 1	Read only
- รีจีสเตอร์ 0B Interrupt mask register 1	Read/write
- รีจีสเตอร์ 0C Status register 2	Read only
- รีจีสเตอร์ 0D Interrupt mask register 2	Read/write
- รีจีสเตอร์ 0E Status register 3	Read only
- รีจีสเตอร์ 0F Interrupt mask register 3	Read/write

Miscellaneous Registers

- รีจีสเตอร์ 10 Symbol clock phase adjustment register Read/write
- รีจีสเตอร์ 11 Clock stop control register Read/write
- รีจีสเตอร์ 12 Power down control register Read/write
- รีจีสเตอร์ 13 Loop back control register Read/write
- รีจีสเตอร์ 14 Auxiliary RAM ADC control register Read/write
- รีจีสเตอร์ 15 Auxiliary ADC control register 1 Read/write
- รีจีสเตอร์ 16 Auxiliary ADC control register 2 Read/write
- รีจีสเตอร์ 17 Transmit path ramping delay register Read/write
- รีจีสเตอร์ 18-1F Coefficient memory I/O access addresses Read/write

Transmit Path Set-up Registers

- รีจีสเตอร์ 20-21 Transmit I channel phase register Read/write
- รีจีสเตอร์ 22-23 Transmit I channel gain register Read/write
- รีจีสเตอร์ 24-25 Transmit I channel offset register Read/write
- รีจีสเตอร์ 26-27 Transmit Q channel phase register Read/write
- รีจีสเตอร์ 28-29 Transmit Q channel gain register Read/write
- รีจีสเตอร์ 2A-2B Transmit Q channel offset register Read/write
- รีจีสเตอร์ 2C-2D Transmit ramp up increment register Read/write
- รีจีสเตอร์ 2E-2F Transmit ramp down increment register Read/write

Receive Path Set-up Registers

- รีจีสเตอร์ 30-31 Receive I channel gain register Read/write
- รีจีสเตอร์ 32-33 Receive I channel offset register Read/write
- รีจีสเตอร์ 34-35 Receive Q channel gain register Read/write
- รีจีสเตอร์ 36-37 Receive Q channel offset register Read/write

Data Access Points

- รีจีสเตอร์ 38-39 Receive I channel data access point Read/write
- รีจีสเตอร์ 3A-3B Receive Q channel data access point Read/write
- รีจีสเตอร์ 3C-3D Transmit I channel data access point Read/write

- รีจีสเตอร์ 3E-3F Transmit Q channel data access point Read/write

Auxiliary Data Registers

- รีจีสเตอร์ 40-4B Auxiliary ADC data registers Read only
- รีจีสเตอร์ 4C-4F Auxiliary DAC memory I/O access addresses Read/write
- รีจีสเตอร์ 50-57 Auxiliary DAC data registers write only

Voice Codec Registers

- รีจีสเตอร์ 58 Voice codec set-up register 1 Read/write
- รีจีสเตอร์ 59 Voice codec set-up register 2 Read/write
- รีจีสเตอร์ 5A Voice codec gain register 1 Read/write
- รีจีสเตอร์ 5B Voice codec gain register 2 Read/write
- รีจีสเตอร์ 5C-5D Voice codec tone frequency register Read/write
- รีจีสเตอร์ 5E-5F Voice codec transmit/receive data register Read/write

Clock Division Registers

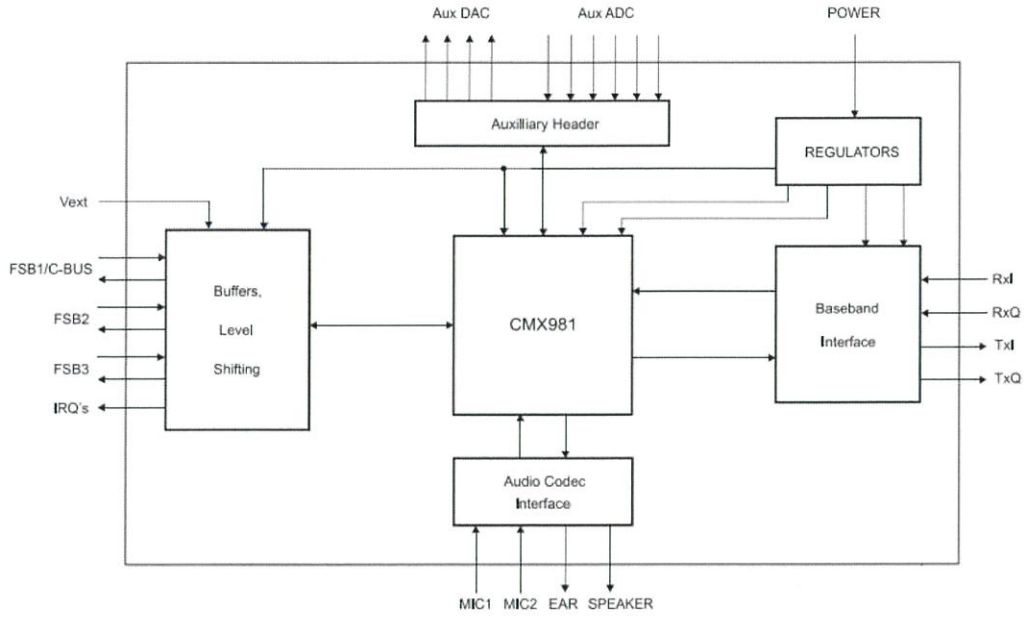
- รีจีสเตอร์ 60 Clock division control register 1 Read/write
- รีจีสเตอร์ 61 Clock division control register 2 Read/write

Direct Write Data Registers

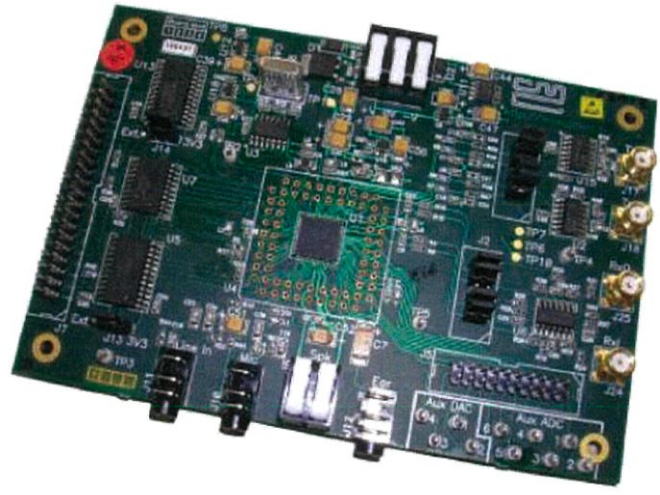
- รีจีสเตอร์ 60-6F Transmit I channel direct write data register write only
- รีจีสเตอร์ 70-7F Transmit Q channel direct write data register write only

2.2.3 Evaluation Kit 9810 (EV9810)

เป็นบอร์ดที่สามารถใช้ในการประเมินผลการทดลองและการออกแบบ CMX981 Advanced Digital Radio Baseband Processor โดยจะประกอบด้วย PCB (*printed circuit board)* แบบดิจิทัลและแอนาล็อก พร้อม IC CMX981 บนบอร์ด เพื่อใช้ในการเข้าถึงสัญญาณข้อมูลเบสแบนด์ สัญญาณควบคุมและอินเทอร์เฟซข้อมูลของ CMX981 ตลอดจนฟังก์ชัน ADC และ DAC บล็อกไดอะแกรม ของ Evaluation Kit 9810 แสดงดังรูปที่ 2.15 และบอร์ด Evaluation Kit 9810 แสดงดังรูปที่ 2.16



รูปที่ 2.15 บล็อกไดอะแกรมของ Evaluation Kit 9810

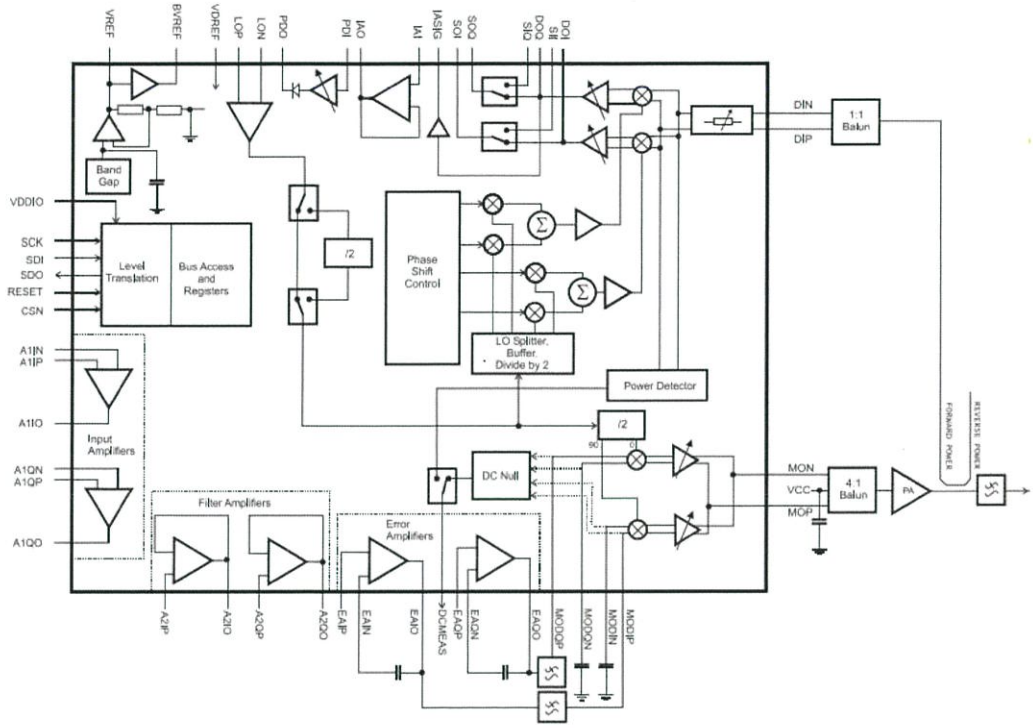


รูปที่ 2.16 บอร์ด Evaluation Kit 9810

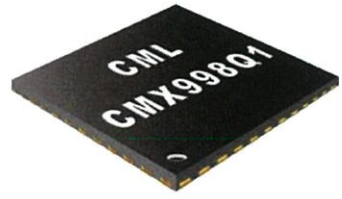
2.2.4 CMX998 Cartesian Feed-back Loop Transmitter

เป็นโซลูชั่นแบบครบวงจรสำหรับเครื่องส่งสัญญาณแบบ Cartesian Feedback Loop (CFBL) ทำหน้าที่เป็นตัวแปลงสัญญาณโดยตรงจาก I และ Q ไปยังเอาท์พุต RF เป็น Linearise 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 โดยแสดงบล็อกไดอะแกรมของ CMX998 ดังรูปที่ 2.17 และ IC CMX998 แสดงดังรูปที่ 2.18



รูปที่ 2.17 บล็อกไดอะแกรมของ CMX998



รูปที่ 2.18 IC CMX998

รายละเอียดของ IC 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

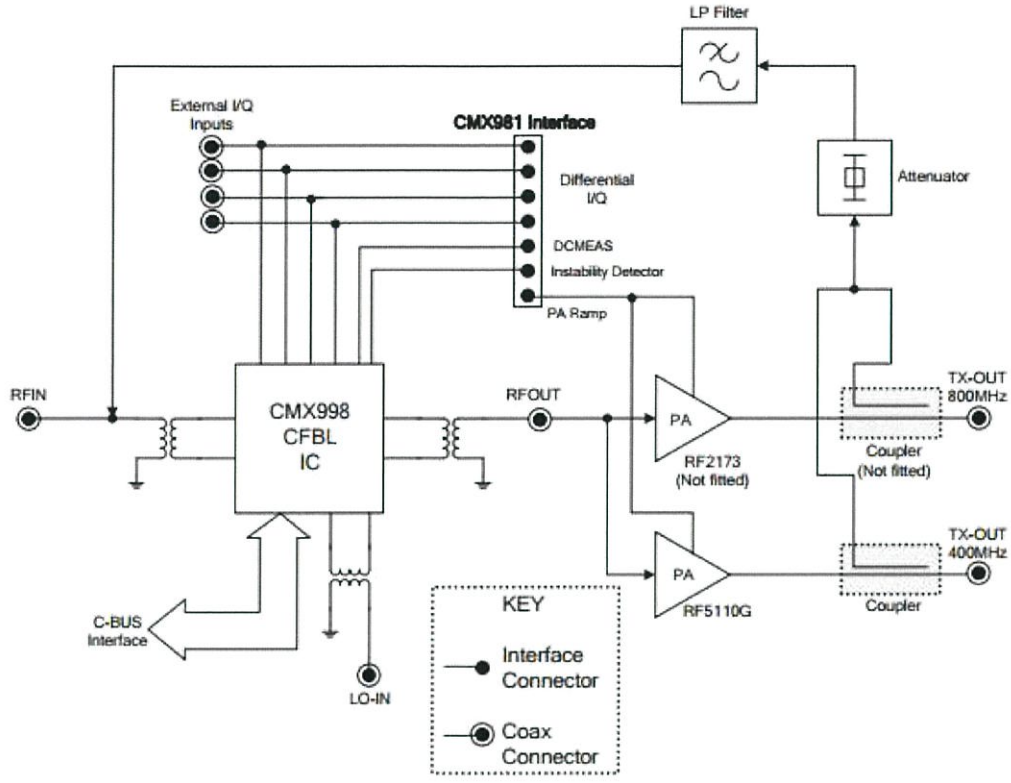
- Down Converter for Feedback Linearization
- 360° Loop Phase Shift Control
- DC Offset Measurement Output
- Flexible Digital Interface
- Linear Modulation Schemes เช่น QPSK, π/4-DQPSK, 8PSK, QAM, OFDM เป็นต้น

รีจิสเตอร์ (Registers) ทั้งหมดของ IC CMX998 มีดังนี้

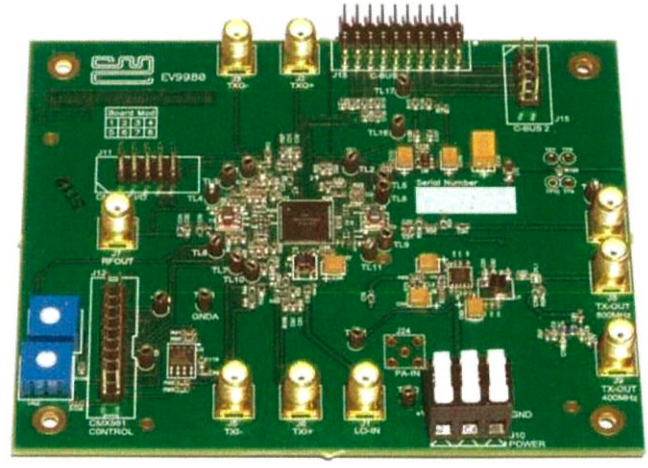
- รีจีสเตอร์ 01 General Reset Register (Address only, no data)
- รีจีสเตอร์ 02 General Command, 8-bit write only
- รีจีสเตอร์ 03 Phase Control Register, 8-bit write only
- รีจีสเตอร์ 04 Gain control, 16-bit write only
- รีจีสเตอร์ 05 Forward Path Gain Control, 8-bit write only
- รีจีสเตอร์ 06 Feedback Path Gain Control, 8-bit write only
- รีจีสเตอร์ 07 Aux Control, 8-bit write only
- รีจีสเตอร์ 08 Frequency Control Register, 8-bit write only
- รีจีสเตอร์ F2 General Command, 8-bit read only
- รีจีสเตอร์ F3 Phase Control Register, 8-bit read only
- รีจีสเตอร์ F4 Gain control, 16-bit read only
- รีจีสเตอร์ F5 Forward Path Gain Control, 8-bit read only
- รีจีสเตอร์ F6 Feedback Path Gain Control, 8-bit read only
- รีจีสเตอร์ F7 Aux Control, 8-bit read only
- รีจีสเตอร์ F8 Frequency Control Register, 8-bit read only

2.2.5 Evaluation Kit 9980 (EV9980)

เป็นบอร์ดที่สามารถใช้ในการประเมินผลการทดลองและการออกแบบ CMX998 Cartesian Feed-back Loop Transmitter โดยจะประกอบด้วย PCB ที่มีแพลตฟอร์มแบบยืดหยุ่นและมี IC CMX998 บนบอร์ด ทำให้สามารถใช้อินเทอร์เฟซควบคุมเพื่อกำหนดค่าและคลื่นความถี่ต่างๆ สามารถเข้าถึงสัญญาณทั้ง RF และ baseband โดยมีกำลังขยายที่ความถี่ 450 MHz หรือ 800 MHz โดย EV9980 สามารถใช้ความถี่ระหว่าง 100 MHz ถึง 1 GHz สามารถกำหนดค่าให้ทำงานร่วมกับ CMX981 สามารถกำหนดระบบฐานข้อมูลเองได้โดยแสดงบล็อกไดอะแกรมของ Evaluation Kit 9980 ดังรูปที่ 2.19 และ Evaluation Kit 9980 แสดงดังรูปที่ 2.20



รูปที่ 2.19 บล็อกไดอะแกรมของ Evaluation Kit 9980

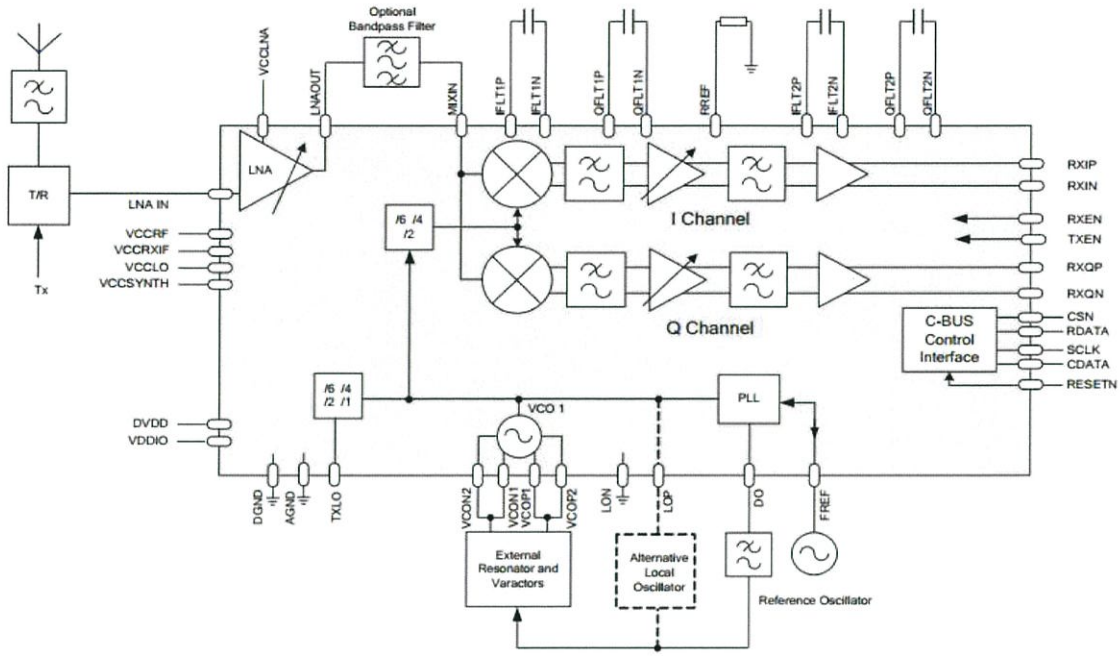


รูปที่ 2.20 บอร์ด Evaluation Kit 9980

2.2.6 CMX994 Direct Conversion Receiver

CMX994 เป็นตัวรับสัญญาณโดยแปลงสัญญาณแบบตรง ประกอบด้วย LAN (Local Area Network) แบบบรอดแบนด์ ตัวควบคุมกำลังขยายแบบไดนามิกสูงในการดีมอดูเลต I และ Q และในส่วนตัวรับสัญญาณประกอบด้วยตัวขยายสัญญาณและตัวกรองแบบเบสแบนด์ที่แม่นยำ และ VCO (Voltage – Controlled Oscillator โดยโหมดพลังงานต่ำนั้นจะช่วยปิดการปรับเฟสของ LO ทำงานจาก

แหล่งจ่ายไฟ 3.3 V เดียวในช่วงอุณหภูมิตั้งแต่ -40 ถึง +85 องศาเซลเซียสและมีอยู่ในชุด VQFN โดย บล็อกไดอะแกรมของ CMX994 แสดงดังรูปที่ 2.21 และ IC CMX 994 แสดงดังรูปที่ 2.22



รูปที่ 2.21 บล็อกไดอะแกรมของ CMX994

รูปที่ 2.22 IC CMX994

รายละเอียดของ IC CMX994 ดังนี้

1) Rx Direct Conversion Receiver
   - Direct conversion eliminates image responses
   - 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

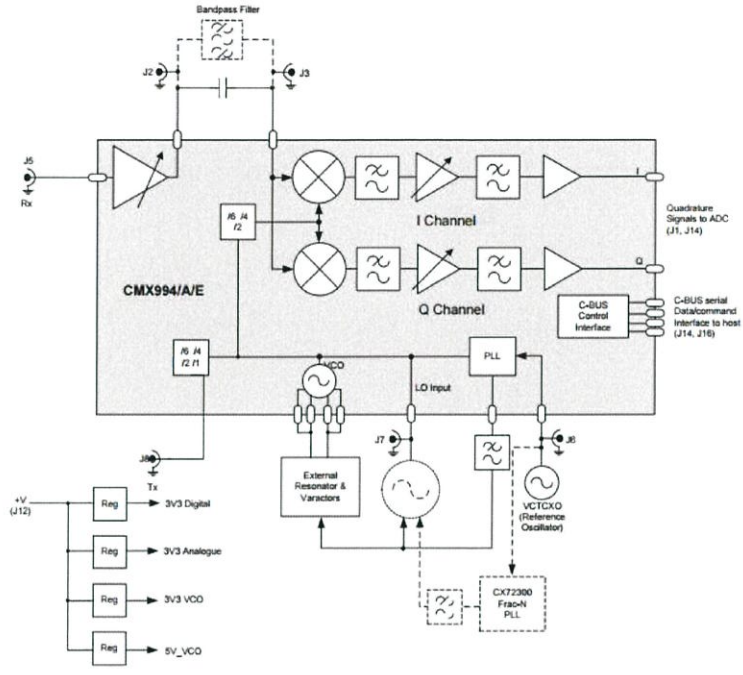
รีจิสเตอร์ (Registers) ทั้งหมดของ IC CMX998 มีดังนี้

- รีจีสเตอร์ 10 General Reset Register (Address only, no data)
- รีจีสเตอร์ 11 General Control Register, 8-bit write only
- รีจีสเตอร์ 12 Rx Control Register, 8-bit write only
- รีจีสเตอร์ 13 Rx Offset Register, 8-bit write only
- รีจีสเตอร์ 14 IM Control Register, 8-bit write only
- รีจีสเตอร์ 16 Rx Gain Register, 8-bit write only
- รีจีสเตอร์ 20-22 PLL M Divider Register, 8-bit write only
- รีจีสเตอร์ 23-24 PLL R Register, 8-bit write only
- รีจีสเตอร์ 25 VCO Control Register, 8-bit write only
- รีจีสเตอร์ E1 General Control Register, 8-bit read only
- รีจีสเตอร์ E2 Rx Control Register, 8-bit read only
- รีจีสเตอร์ E3 Rx Offset Register, 8-bit read only
- รีจีสเตอร์ E4 IM Control Register, 8-bit read only
- รีจีสเตอร์ E6 Rx Gain Register, 8-bit read only
- รีจีสเตอร์ D0-D2 PLL M Divider Register, 8-bit read only
- รีจีสเตอร์ D3-D4 PLL R Register, 8-bit read only
- รีจีสเตอร์ D5 VCO Control Register, 8-bit read only

2.2.7 Evaluation Kit 9942 (EV9942)

เป็นบอร์ดที่สามารถใช้ในการประเมินผลการทดลองและการออกแบบ CMX994 Direct Conversion Receiver โดยจะประกอบด้วย 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 100 MHz ถึง 940 MHz โดยมีการทำงานเริ่มต้นที่ศูนย์กลางที่ 448 MHz โดยแสดงบล็อกไดอะแกรมของ Evaluation Kit 9942 ดังรูปที่ 2.23 และบอร์ด Evaluation Kit 9942 แสดงดังรูปที่ 2.24

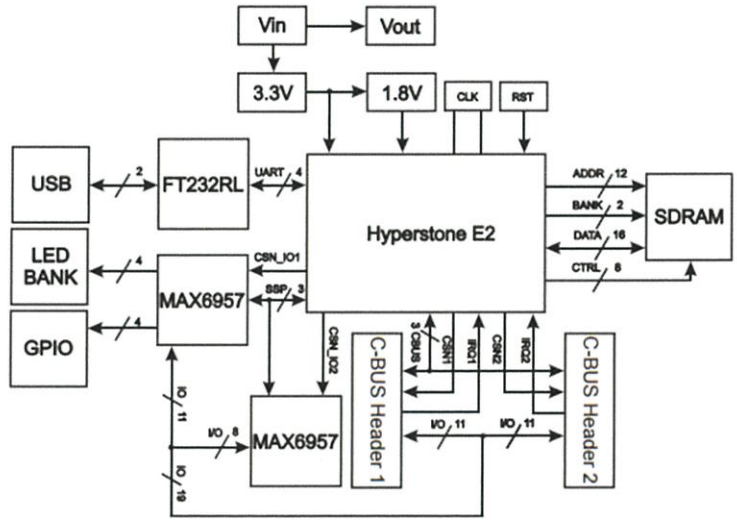


รูปที่ 2.23 บล็อกไดอะแกรมของ Evaluation Kit 9942

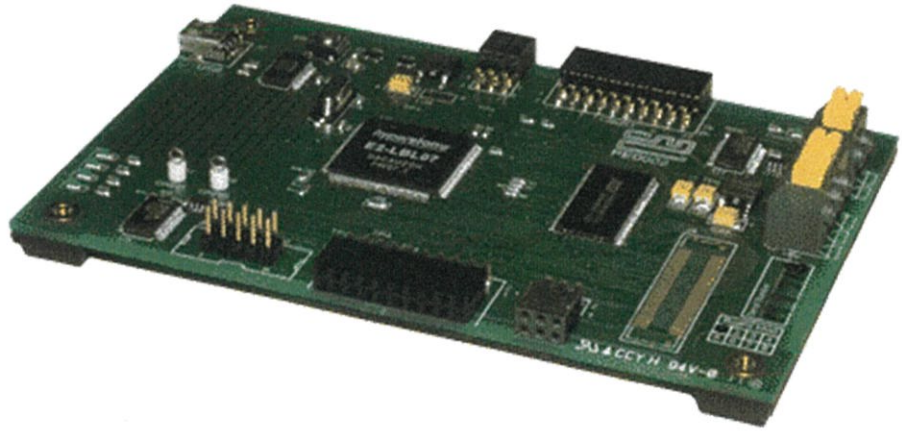
รูปที่ 2.24 บอร์ด Evaluation Kit 9942

2.2.8 PE0002 Evaluation Kit Interface Card

เป็นระบบอินเทอร์เฟซสำหรับใช้กับชุดประเมินของบริษัท CML รวมทั้งผลิตภัณฑ์จาก FirmASIC ซึ่งจะช่วยลดความซับซ้อนในการประเมินผลและออกแบบขั้นตอนในการทดลอง จากการใช้ไมโครโปรเซสเซอร์ RISC / DSP 32 บิต ของ Hyperstone และใช้ PC GUI ข้อมูลที่สร้างจะอยู่ในรูปแบบเวลา และส่งไปยัง IC เป้าหมาย โดยใช้อินเทอร์เฟซแบบอนุกรมของ C – BUS สามารถดำเนินการอ่านและเขียนโดยการเขียนสคริปต์ ติดต่อสื่อสารกับคอมพิวเตอร์ผ่านทางพอร์ต USB (Universal Serial Bus) โดยแสดงบล็อกไดอะแกรมของ PE0002 ดังรูปที่ 2.25 และ PE0002 Evaluation Kit Interface Card แสดงดังรูปที่ 2.26



รูปที่ 2.25 บล็อกไดอะแกรมของ PE0002



รูปที่ 2.26 PE0002 Evaluation Kit Interface Card

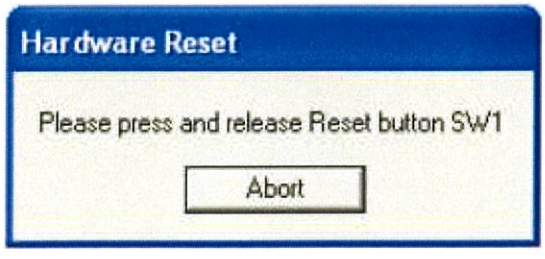
PE0002 Evaluation Kit Interface Card มีรายละเอียดดังนี้

- 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

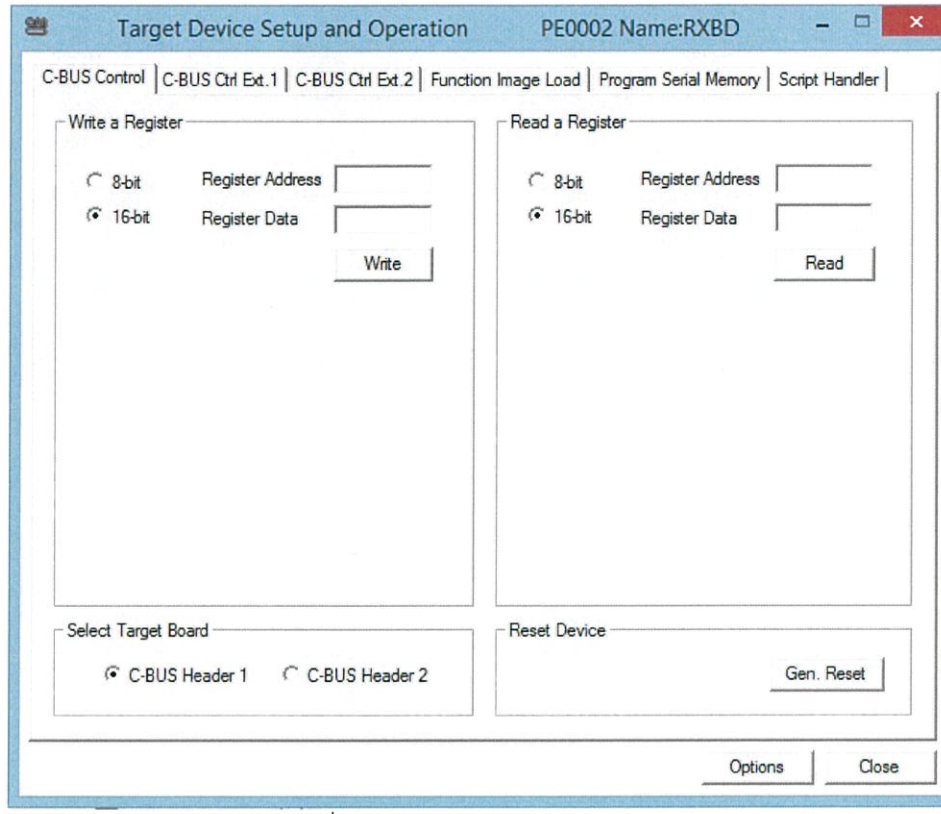
2.2.8.1 โปรแกรม ES000243

เป็นโปรแกรมที่ใช้ในการสื่อสารกับ PE0002 บนระบบปฏิบัติการ Window โดยการใช้งานโปรแกรม ES000243.EXE หากไม่สามารถติดต่อสื่อสารจะปรากฏข้อความว่าล้มเหลว แต่ถ้าหากสามารถสื่อสารได้จะปรากฏกล่องข้อความตามรูปที่ 2.27 โดยจะให้กดสวิตช์รีเซ็ต SW1 บน PE0002 เฟิร์มแวร์จะดาวน์โหลดโดยอัตโนมัติไปยังบอร์ด PE0002 และกล่องตอบโต้จะเข้าสู่โปรแกรมหลัก โดยสามารถเชื่อมต่ออัตโนมัติกับคอมพิวเตอร์ได้เมื่อต่อ PE0002 โดยโปรแกรม ES000243 จะมีทั้งหมด 5 ส่วน ดังนี้

1) C – BUS Control Tab โดยจะสามารถอ่าน เขียนและเซ็ตค่าทั่วไปของรีจิสเตอร์ โดยเมื่อป้อนอักษรลงในช่องแก้ไขจะทำการตรวจสอบเพื่อให้แน่ใจว่าเป็นเลขฐานสิบหก สามารถเลือกอ่านและเขียน 8 บิต หรือ 16 บิต โดยความยาวของค่าที่ถูกป้อนจำกัดไว้ที่ 2 อักษร (1 ไบต์) ในการอ่านหรือเขียน และความยาวของที่อยู่จำกัดไว้ที่ 2 อักษร หรือ 4 อักษร (1 ไบต์ หรือ 2 ไบต์) โดย COM 1 จะเชื่อมต่อที่ J5 ของ PE0002 และ COM 2 จะเชื่อมต่อที่ J3 ซึ่ง C – BUS Control Tab แสดงดังรูปที่ 2.28

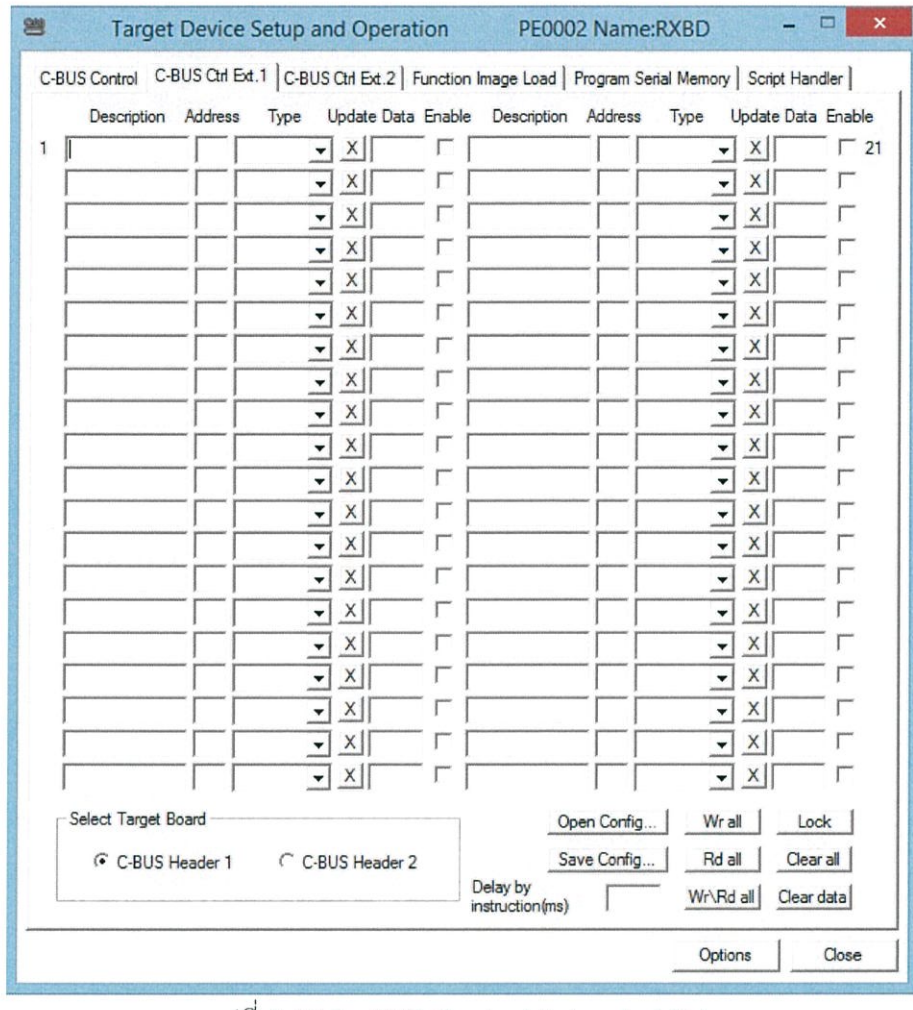


รูปที่ 2.27 กล่องข้อความเริ่มต้นโปรแกรม



รูปที่ 2.28 C – BUS Control Tab

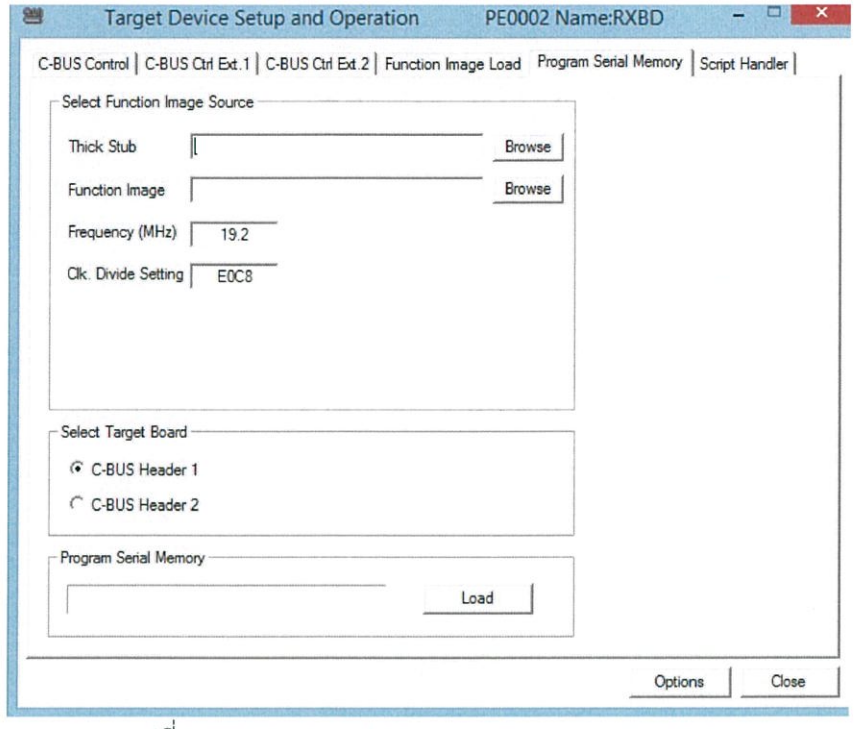
2) C - BUS Control Extended Tab (C - BUS Ctrl Ext. 1 and 2) มีฟังก์ชันการอ่านและเขียน C – BUS หลายชุด ในแต่ละแถวแสดงถึงรีจิสเตอร์ของ C – BUS โดยจะมีการตรวจสอบข้อมูลที่ป้อนลงในช่องข้อมูลเพื่อตรวจสอบว่าเป็นเลขฐานสิบหกหรือไม่ โดยความยาวที่อยู่ที่ป้อนจะถูกจำกัดที่ 2 อักษร (1 ไบต์) และสำหรับข้อมูลรีจิสเตอร์จำกัดที่ 2 อักษร หรือ 4 อักษร (1 ไบต์ หรือ 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 แสดงดังรูป ที่ 2.29



รูปที่ 2.29 C - BUS Control Extended Tab

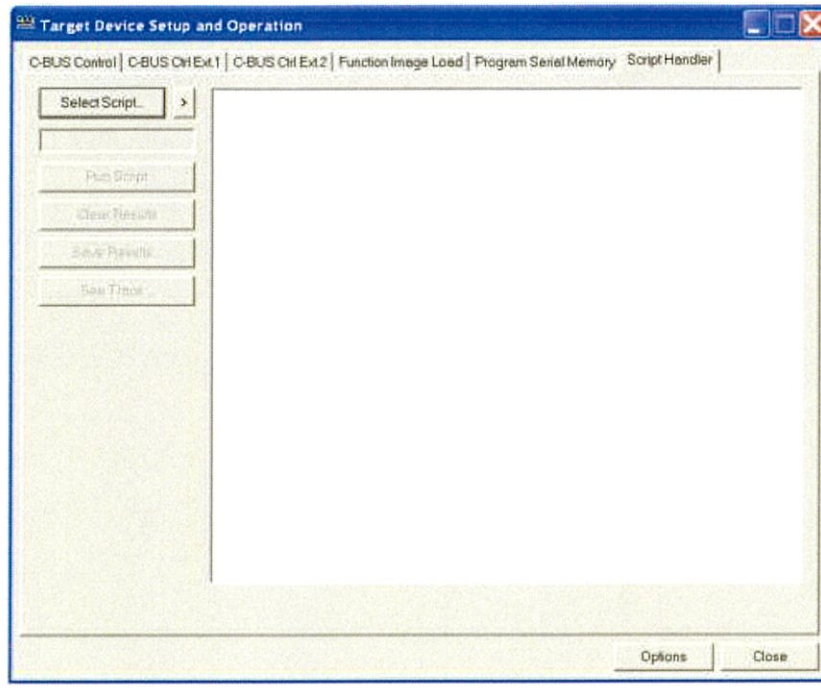
3) The Function Image™ Load Tab มีฟังก์ชันโหลดและเปิดใช้งานฟังก์ชันสำหรับ FirmASIC

4) The Serial Memory Programming Tab สำหรับเขียนโปรแกรมหน่วยความจำแบบอนุกรมสำหรับ FirmASIC เพื่อจัดเก็บข้อมูล Function Image แสดงในรูปที่ 2.30



รูปที่ 2.30 The Serial Memory Programming Tab

5) The Script Handler Tab สามารถใช้สคริปต์ได้ โดยไฟล์เหล่านี้เป็นไฟล์ข้อความธรรมดาหรือไฟล์ .pes แต่จะทำงานบนไมโครโปรเซสเซอร์ โดยใช้ภาษาสคริปต์เฉพาะ ให้เลือกสคริปต์โดยกดปุ่ม “Select Script” โดยหน้าต่างผลลัพธ์แสดงค่าที่ส่งกลับโดยสคริปต์เมื่อกดปุ่ม “Run Script” ผลลัพธ์เหล่านี้สามารถบันทึกลงในไฟล์ข้อความโดยกดปุ่ม “Save Results” หรือต้องการล้างผลลัพธ์โดยกดปุ่ม “Clear Results” โดยแสดงดังรูปที่ 2.31



รูปที่ 2.31 The Script Handler Tab

2.2.9 Script Language Reference

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

- คำสั่ง filew ข้อมูลจะถูกโอนย้ายจาก PE0002 ไปยังเครื่องโฮสต์ผ่านบัฟเฟอร์ถาวร (FIFO

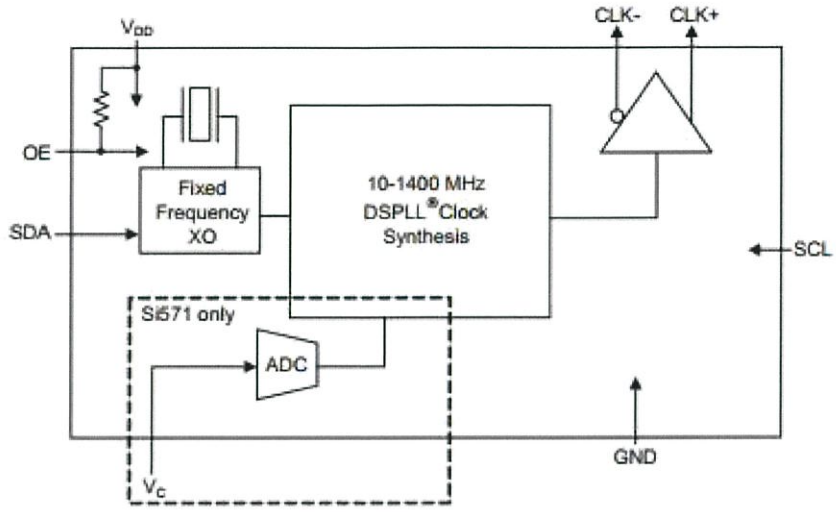
- คำสั่ง filer ข้อมูลจะถูกโอนย้ายจากเครื่องโฮสต์ไปยัง PE0002 ผ่านบัฟเฟอร์ถาวรอื่น(FIFO) "FileBuffer" ขนาด 294512 word

- คำสั่ง fopenw เปิดหรือสร้างไฟล์บนเครื่องโฮสต์ที่จะใช้เพื่อรับข้อมูลจากสคริปต์ PE0002 ทุกครั้งที่พบคำสั่ง filew จะมีการถ่ายโอนข้อมูล 16 บิตจากสคริปต์ไปยังบัฟเฟอร์

-คำสั่ง fopenr เปิดไฟล์บนเครื่องโฮสต์ที่จะใช้ในการส่งข้อมูลไปยังสคริปต์ PE0002

2.2.10 Si570

Si570 XO / Si571 VCXO เป็น IC ที่สามารถ generate ความถี่ โดยใช้วงจร DSPLL ขั้นสูงของ Silicon Laboratories เพื่อให้สัญญาณนาฬิกามีการตอบสนองในทุกความถี่ Si570 / Si571 สามารถใช้งานได้กับทุกความถี่ขาออกตั้งแต่ 10 ถึง 945 MHz สามารถเลือกความถี่ที่ 1400 MHz ด้วยความละเอียด <1 ppm โดยเชื่อมต่อผ่านอินเทอร์เฟซแบบอนุกรม I2C โดย IC นี้มีความเสถียร ความเชื่อถือสูง โดยแสดงบล็อกไดอะแกรมของ Si570 ดังรูปที่ 2.32 และแสดง IC Si570 ดังรูปที่ 2.33



รูปที่ 2.32 บล็อกไดอะแกรมของ Si570

รูปที่ 2.33 IC Si570

IC 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

รีจิสเตอร์ (Registers) ทั้งหมดของ IC Si570 มีดังนี้

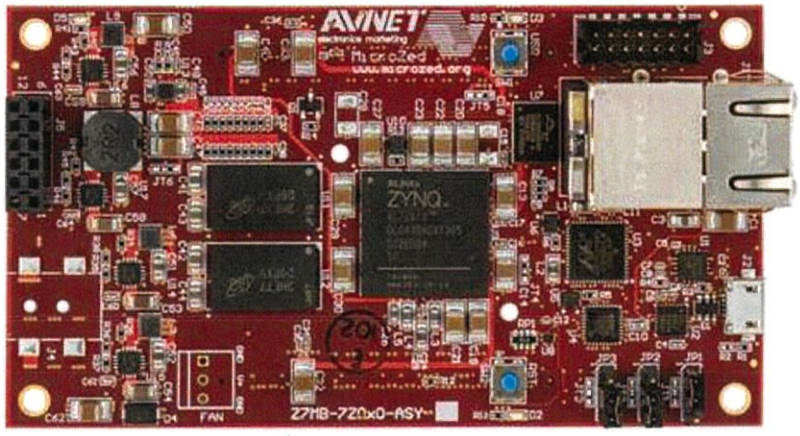
- รีจีสเตอร์ 7 High Speed/N1 Dividers
- รีจีสเตอร์ 8 Reference Frequency
- รีจีสเตอร์ 9 Reference Frequency
- รีจีสเตอร์ 10 Reference Frequency
- รีจีสเตอร์ 11 Reference Frequency
- รีจีสเตอร์ 12 Reference Frequency
- รีจีสเตอร์ 13 High Speed/N1 Dividers
- รีจีสเตอร์ 14 Reference Frequency
- รีจีสเตอร์ 15 Reference Frequency
- รีจีสเตอร์ 16 Reference Frequency
- รีจีสเตอร์ 17 Reference Frequency
- รีจีสเตอร์ 18 Reference Frequency
- รีจีสเตอร์ 135 Reset/Freeze/Memory Control
- รีจีสเตอร์ 137 Freeze DCO

Si570 XO และ Si571 VCXO เป็นเครื่องกำเนิดสัญญาณ สามารถตั้งโปรแกรมเพื่อกำหนดความถี่เอาท์พุท 10 MHz ถึง 1.4 GHz โดยสามารถกำหนดความถี่เอาท์พุท (f_{out}) จากการคำนวณโดยกำหนด HSDIV มีค่าเท่ากับ 4, 5, 6, 7, 9 หรือ 11, N1 มีค่าเท่ากับ 1 หรือเลขคู่จนถึง 128, f_0 มีค่าเท่ากับ 156.28 MHz และ RFREQ = 0x2BBD25236 (อ่านค่าจากรีจิสเตอร์ของ Si570) = 43.7502734363

2.2.11 Field Programmable Gate Array (FPGA)

FPGA คือ อุปกรณ์สารกึ่งตัวนำชนิดโปรแกรมได้ที่มีโครงข่ายการเชื่อมต่อภายในแบบเมตริกซ์ โครงสร้างภายในของ FPGA นั้นสามารถโปรแกรมให้มีหน้าที่การทำงานเหมือนลอจิกเกตพื้นฐาน เช่น AND, OR, XOR, NOT หรือรวมกันหลายๆ ชนิด (combinational logic) เพื่อให้ทำหน้าที่ที่มีความซับซ้อนเพิ่มขึ้น เช่น decoders หรือฟังก์ชันทางคณิตศาสตร์ ใน FPGAs ทั่วไป นอกจากจะประกอบด้วยส่วนของวงจรลอจิกแบบโปรแกรมได้แล้ว จะยังมีบล็อกของหน่วยความจำ ซึ่งอาจจะสร้างด้วยฟลิบฟลอปอย่างง่าย หรือใช้พื้นที่ของสารกึ่งตัวนำสร้างเป็นหน่วยความจำจริงๆ อยู่ภายในก็ได้ ในการออกแบบวงจรดิจิตอลอิเล็กทรอนิกส์ ที่มี FPGA อยู่บนแผงวงจรด้วยนั้น จะช่วยให้ผู้ออกแบบสามารถลดขนาดของแผงวงจร รวมทั้งสามารถออกแบบได้รวดเร็ว ไม่ต้องทดสอบรายละเอียดภายในให้เสร็จสมบูรณ์ 100 % ก็สามารถออกแบบแผงวงจรได้ เมื่อได้รับแผงวงจรและประกอบอุปกรณ์ต่างๆ เสร็จแล้ว จึงค่อยกำหนดหน้าที่การทำงานของ FPGA ได้ในภายหลัง ต่างจากการออกแบบด้วยลอจิกเกตขนาดเล็ก ที่ต้องออกแบบ

ทางเดินของลายทองแดงให้เสร็จสมบูรณ์ก่อน และไม่สามารถแก้ไขได้ในภายหลัง นอกจากนี้ การใช้งาน FPGA สามารถโปรแกรมการทำงานได้ในทุกขณะแม้แต่ขณะที่ส่งมอบงานแล้ว ก็ยังสามารถเข้าไปแก้ไขวงจรได้โดยง่ายดาย จึงเป็นที่มาของคำว่า "field programmable" ซึ่งก็หมายถึงโปรแกรมได้ในภาคสนามหรือที่หน้างานนั่นเอง อย่างไรก็ตามข้อกำหนด (Configuration) ของ FPGA จะหายไปหลังจากปิดไฟเลี้ยง ดังนั้น จะต้องมีหน่วยความจำภายนอก (Flash) มาคอยรักษาข้อกำหนดของ FPGA ไว้ ซึ่ง FPGA จะมีกระบวนการอ่านข้อกำหนดนั้นโดยอัตโนมัติหลังจากได้รับไฟเลี้ยง



รูปที่ 2.34 FPGA Zynq7000

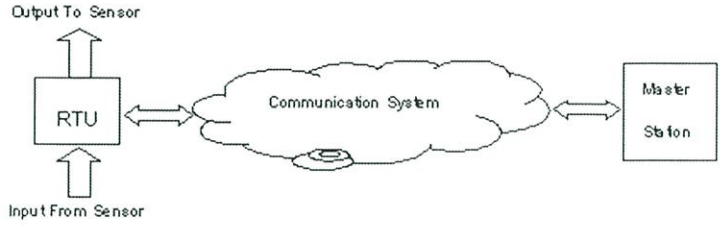
2.2.12 Remote Terminal Unit (RTU)

เป็นหน่วยควบคุมระยะไกลของระบบ SCADA (Supervisory Control And Data Acquisition) ระบบ SCADA คือระบบที่ใช้ในการรวบรวมและจัดการข้อมูล แสดงผลของการทำงานการตรวจวัดรับ-ส่งข้อมูล และควบคุมการทำงานของอุปกรณ์ โดยเฉพาะกับอุปกรณ์ที่อยู่ห่างไกลออกไปจากศูนย์ควบคุมและไม่มีเจ้าหน้าที่ปฏิบัติงานที่สถานีนั้นๆ (UnMan Station) โดย RTU ทำหน้าที่ในการความคุมการทำงานของอุปกรณ์และประมวลผลข้อมูลก่อนที่จะส่งรายงานไปยังศูนย์ควบคุม นอกจากนั้น RTU ยังจะต้องรับคำสั่งในการควบคุมอุปกรณ์จากศูนย์ควบคุม ซึ่งแสดงในรูปที่ 2.35

2.2.12.1 การทำงานที่จะต้อง ติดต่อกับส่วนอื่นๆ ในระบบคือ

1) การติดต่อกับ MTU (Maximum transmission unit) เป็นการติดต่อในลักษณะของการส่งข้อมูลกลับหรือรับคำสั่งจาก MTU ผ่านระบบสื่อสาร

2) การติดต่อกับส่วนควบคุมอื่นๆ ภายใน RTU เอง เช่น เซ็นเซอร์ PLC หรือมอเตอร์ เป็นการติดต่อในลักษณะ ของการส่งคำสั่งควบคุมไปควบคุมอุปกรณ์ต่างๆ หรือรับเอาค่าต่างๆ ที่ได้ จากอุปกรณ์มาประมวลผล ผ่านสัญญาณ ทางไฟฟ้าแบบต่างๆ



รูปที่ 2.35 แสดงส่วนประกอบหลักของระบบ SCADA

2.2.12.2 ส่วนประกอบหลักของ RTU ที่สำคัญมีอยู่ 3 ส่วนดังต่อไปนี้

1) Central Processing Unit (CPU) ของ RTU หน้าที่หลักคือ ทำหน้าที่ในการประมวลผลสัญญาณที่รับมาจาก Filed Instrument โดยสัญญาณที่ได้รับมาจะถูกต่อเข้ากับ I/O Module ตามมาตรฐานสัญญาณต่างๆ จะทำการการประมวลผลข้อมูลต่างๆที่ได้รับจาก I/O Module เพื่อที่จะส่งข้อมูลให้กับศูนย์ควบคุม และทำหน้าที่แปลงคำสั่งจากศูนย์ควบคุมเพื่อใช้ในการควบคุมอุปกรณ์ต่างๆ ที่ติดตั้งอยู่ที่สถานีสนามและควบคุมระบบการสื่อสารระหว่าง RTU กับศูนย์ควบคุม โดยผ่าน Port ในการสื่อสาร ซึ่ง Port ที่ใช้ในการสื่อสารนั้นจะขึ้นอยู่กับสื่อที่ใช้

2) Input / Output Module (I/O Module) ทำหน้าที่ในการรับส่งสัญญาณจาก CPU เพื่อส่งไปควบคุม หรืออ่านค่าจากอุปกรณ์เครื่องมือวัดต่างๆ

3) Communication Port ทำหน้าที่เป็นช่องทางในการสื่อสารข้อมูลระหว่าง RTU กับศูนย์ควบคุมหรือ RTU ด้วยกัน และสามารถใช้ช่องทาง (Port) ในการสื่อสารมากกว่า 1 Port โดยจะต้องกำหนดชนิดของช่องทางในการสื่อสารและรวมถึงการกำหนดสื่อที่ใช้ด้วย เช่น วิทยุสื่อสาร ดาวเทียม Microwave เป็นต้น

โดยที่ในโครงงานนี้มีการใช้ RTU ทั้งในส่วนของ USRP และ บอร์ดของทางสถานประกอบการ ทั้งสองส่วนจะมี Central Processing Unit คือ FPGA เหมือนกันแตกต่างกันที่รุ่นของ FPGA ที่ใช้ มี Input / Output Module ที่รับคำสั่งจาก FPGA ไปควบคุมส่วนต่างๆคล้ายๆกัน และมี Communication Port คือสายอากาศ สื่อสารข้อมูลระหว่าง RTU ด้วยกัน

บทที่ 3
วิธีการดำเนินงาน

3.1 การเลือกอุปกรณ์ที่ใช้ในการทดลอง

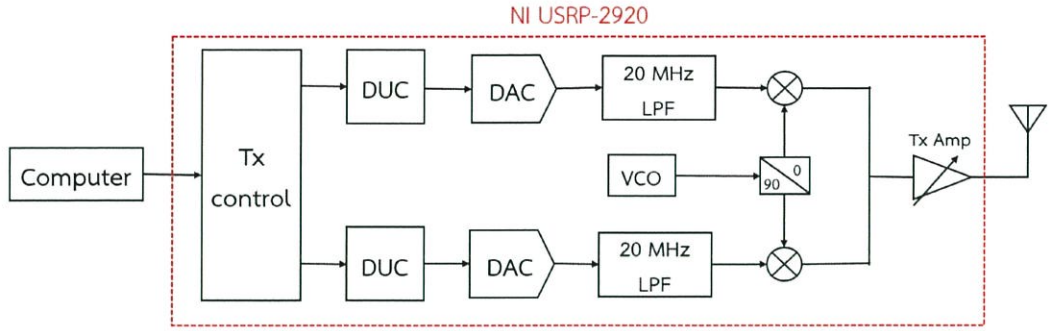
บริษัท กสท. โทรคมนาคม จำกัด มหาชน ต้องการใช้งานบอร์ด Advance Digital Radio Baseband Board ให้สามารถรับส่งข้อมูลบนมาตรฐาน TETRA (Terrestrial Trunked Radio) ได้ โดยทางบริษัทได้ศึกษาและพัฒนา Advance Digital Radio Baseband Board (บอร์ด TETRA RTU ของบริษัท กสท. โทรคมนาคม จำกัด มหาชน) มาระยะเวลาหนึ่งแล้วโดยต้นแบบของบอร์ดนี้คือบอร์ด Evaluation Kit 9810 (EV9810), Evaluation Kit 9980 (EV9980) และ Evaluation Kit 9942 (EV9942) ซึ่งเป็นบอร์ดที่มี IC CMX981, IC CMX998 และ IC CMX994 อยู่บนบอร์ด โดยไอซีทั้งหมดนี้ได้สร้างขึ้นบนมาตรฐาน TETRA

โดยบริษัท กสท. โทรคมนาคม จำกัด มหาชน ได้เล็งเห็นว่าอุปกรณ์ในระบบ SDR (software-Defined Radio) เป็นอุปกรณ์ที่มีความยืดหยุ่นและง่ายต่อการใช้งาน ซึ่งบางอุปกรณ์ในระบบ SDR สามารถนำไปประยุกต์ใช้งานบนมาตรฐาน TETRA ได้ ดังนั้นจึงได้นำอุปกรณ์ที่เรียกว่า USRP (Universal Software Radio Peripheral) มาศึกษาและทดลองให้สามารถรับส่งข้อมูลบนมาตรฐาน TETRA ได้ก่อนก่อนที่จะนำบอร์ด Advance Digital Radio Baseband Board มาทดลองและพัฒนา เพราะตัวเครื่อง USRP จะใช้ Field Gate Programmable Array (FPGA) ในการประมวลผล ทำให้สามารถนำมาปรับแต่งให้เป็นระบบที่ต้องการจะทดสอบได้ง่ายและในปัจจุบัน USRP เป็นอุปกรณ์ที่สามารถใช้งานบนซอฟแวร์ MATLAB ที่เป็น Simulink ได้ ซึ่งง่ายต่อการประยุกต์ใช้งาน เพราะสามารถปรับเปลี่ยนพารามิเตอร์ (parameter) ในบล็อก (block) ต่างๆ ใน ซอฟแวร์ MATLAB ที่เป็น Simulink ให้สามารถใช้งานบนมาตรฐาน TETRA ได้

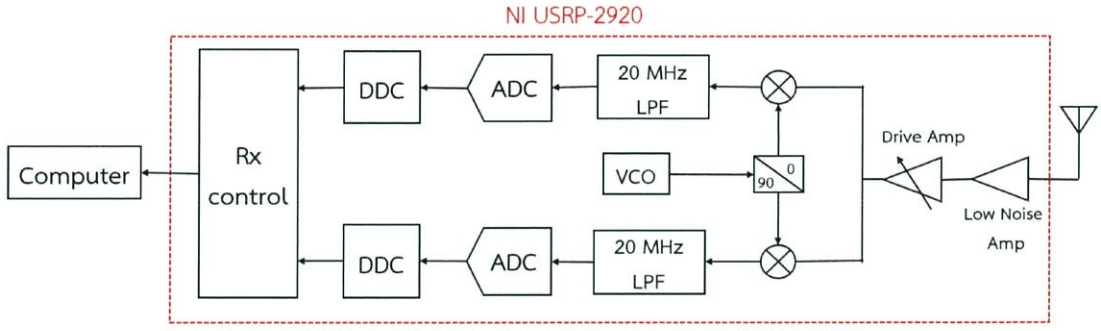
3.2 การใช้งานอุปกรณ์

3.2.1 การใช้งาน USRP (NI USRP-2920)

เป็นการใช้งาน USRP บนมาตรฐาน TETRA ที่ความถี่ 450 MHz ด้วยอัตราการรับส่งข้อมูล 144 ksample/sec และมอดูเลตแบบ $\pi/4$-Differential Quadrature Phase Shift Keying ($\pi/4$-DQPSK) โดยใช้ซอฟแวร์ MATLAB ที่เป็น Simulink โดยภาคส่งมีการทำงานดังรูปที่ 3.1 และ ภาครับมีการทำงานดังรูปที่ 3.2



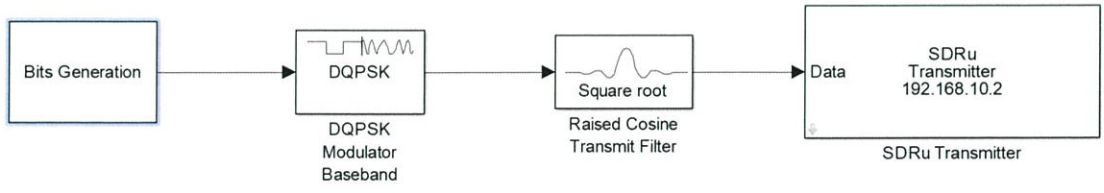
รูปที่ 3.1 การทำงานของภาคส่ง



รูปที่ 3.2 การทำงานของภาครับ

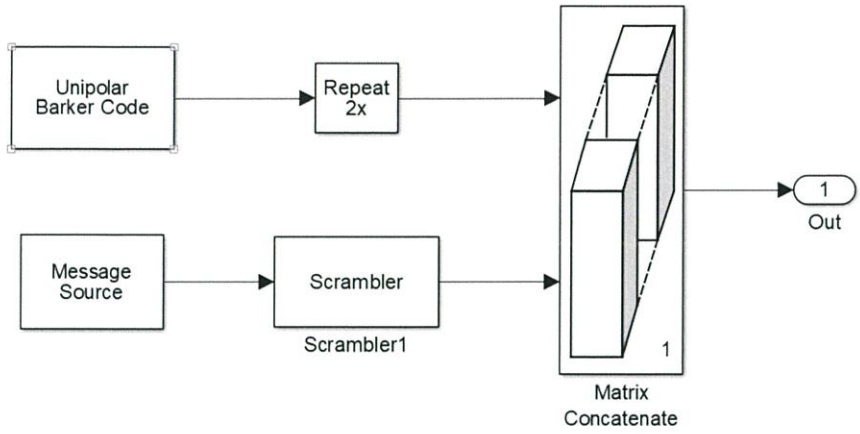
3.2.1.1 ภาคส่ง

ในส่วนของโปรแกรม MATLAB ที่เป็น Simulink ผู้จัดทำได้ทำการปรับพารามิเตอร์ในบล็อกต่างๆ ใน Communications System Toolbox Support Package for USRP Radio example ที่เป็นภาคส่งเพื่อให้สามารถส่งข้อมูลที่เป็น Text ได้ โดยใช้งานที่ความถี่ 450 MHz บนมาตรฐาน TETRA ได้ โดย Simulink Block ของภาคส่งแสดงดังรูปที่ 3.3



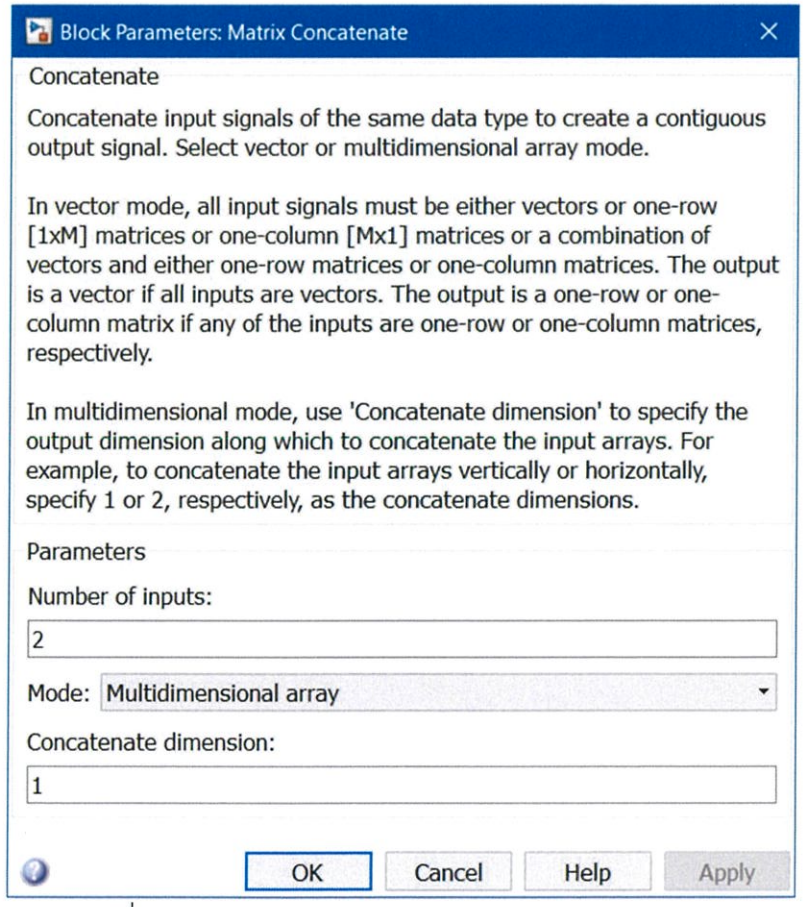
รูปที่ 3.3 Block Simulink ของภาคส่ง

จากรูปที่ 3.3 เป็น Simulink Block โดยรวมทั้งหมดของภาคส่ง โดยในบล็อก Bits generation เป็นกระบวนการ encode ของสัญญาณเบสแบนด์ (baseband) ซึ่งมีการมอดูเลตแบบ DQPSK โดยใช้บล็อก DQPSK Modulator Baseband และใช้บล็อก Raised Cosine Transmit Filter ในการ Upsample โดยภายในบล็อก Bits Generation มี subsystem แสดงดังรูปที่ 3.4



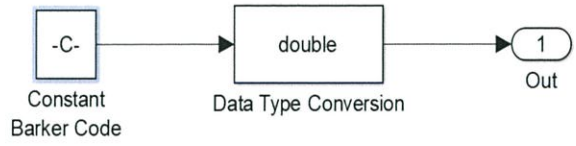
รูปที่ 3.4 subsystem ของบล็อก Bits Generation

บล็อก Matrix Concatenate ถูกนำมาใช้เพื่อเชื่อมต่อสัญญาณข้อมูลอินพุตประเภทเดียวกันเพื่อสร้างสัญญาณเอาต์พุตต่อเนื่องและสามารถเลือกโหมดอาร์เรย์แบบเวกเตอร์หรือหลายมิติ โดย Block parameter ของ Matrix Concatenate แสดงดังรูปที่ 3.5



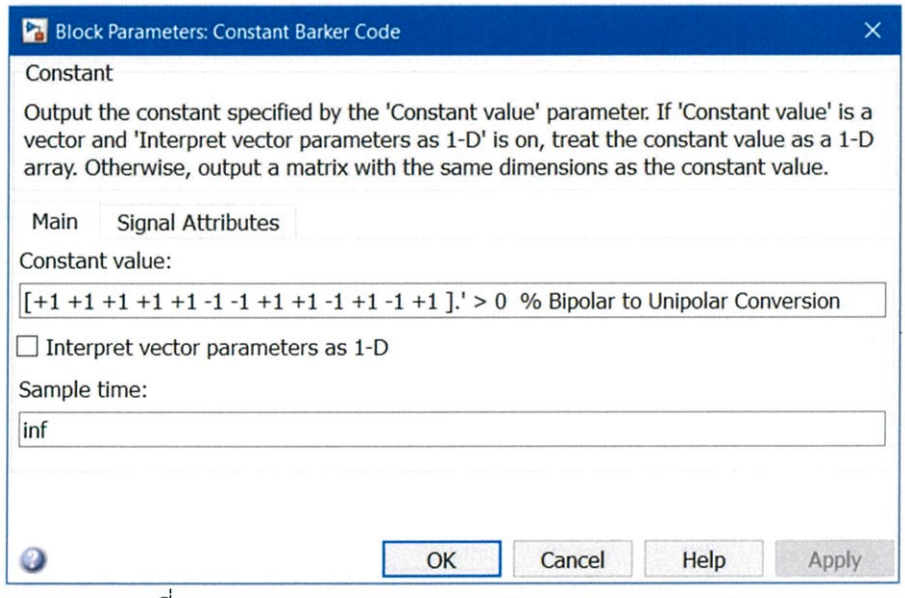
รูปที่ 3.5 Block parameter ของ Matrix Concatenate

โดยบล็อก Unipolar Barker Code จากรูปที่ 3.4 นำมาใช้เพื่อสร้าง header ให้ข้อมูลที่ต้องการส่ง โดย Subsystem ของบล็อก Unipolar Barker Code แสดงดังรูปที่ 3.6

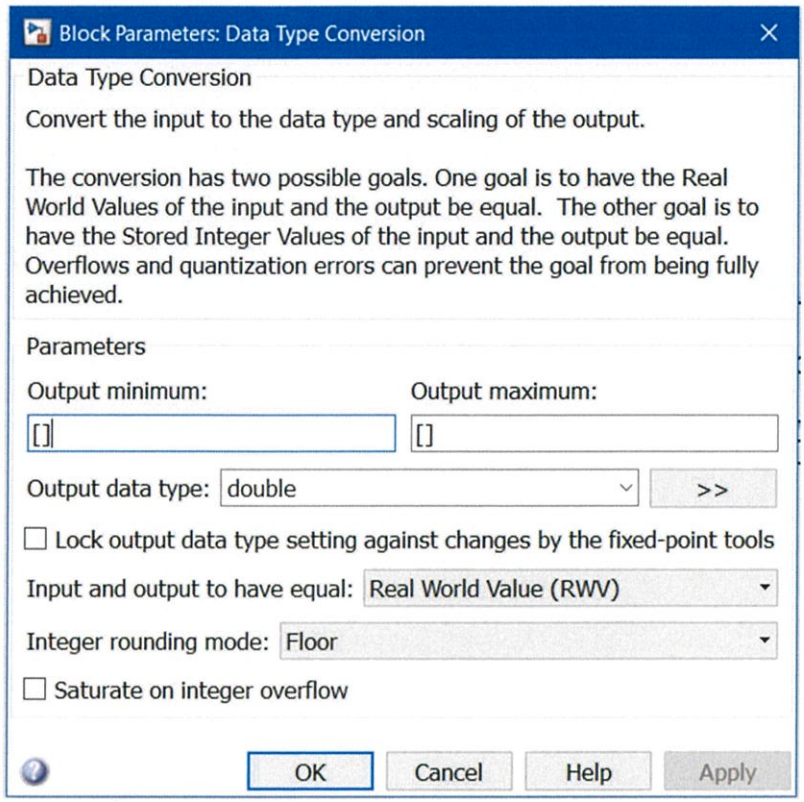


รูปที่ 3.6 Subsystem ของ Unipolar Barker Code

บล็อก Constant Barker Code ถูกนำมาใช้เพื่อส่งค่าคงที่ที่กำหนดโดยพารามิเตอร์ Constant value ถ้า Constant value เป็นเวกเตอร์ และช่อง Interpret vector parameters as 1-D เปิดอยู่ จะทำให้ Constant value ที่เป็นเวกเตอร์หลายมิติเปลี่ยนเป็นค่าคงที่เป็นอาร์เรย์หนึ่งมิติ บล็อก Data Type Conversion ถูกนำมาใช้เพื่อแปลงชนิดของข้อมูลและปรับขนาดของเอาต์พุต โดย Block parameter ของ Constant Barker Code และ Data Type Conversion แสดงดังรูปที่ 3.7 และ รูปที่ 3.8 ตามลำดับ

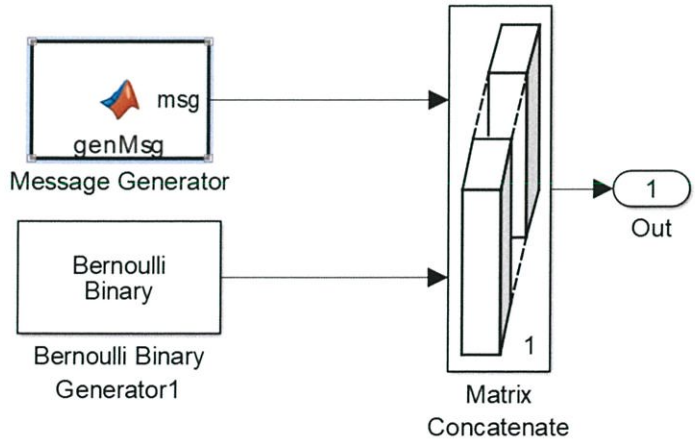


รูปที่ 3.7 Block parameter ของ Constant Barker Code



รูปที่ 3.8 Block parameter ของ Data Type Conversion

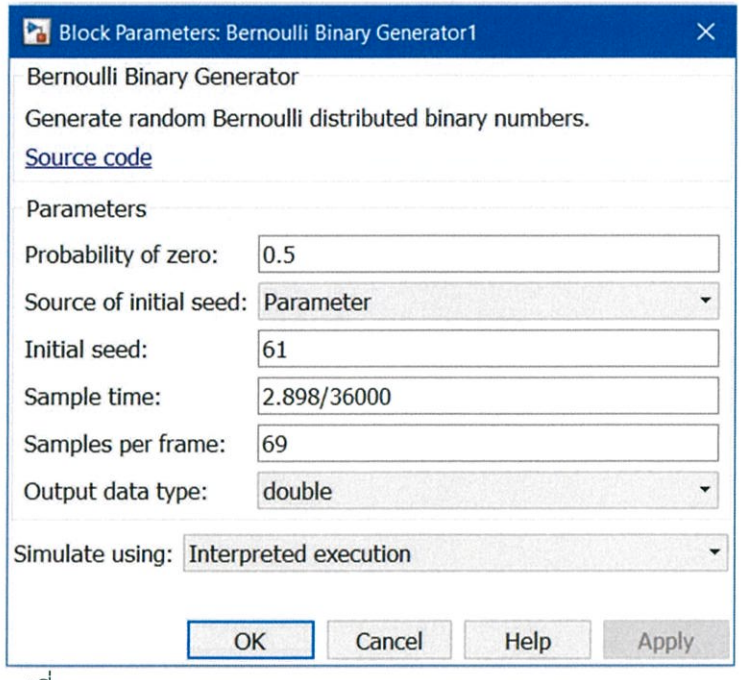
ในบล็อก Message Source จากรูปที่ 3.4 มี Subsystem แสดงดังรูปที่ 3.9



รูปที่ 3.9 Subsystem ของบล็อก Message Source

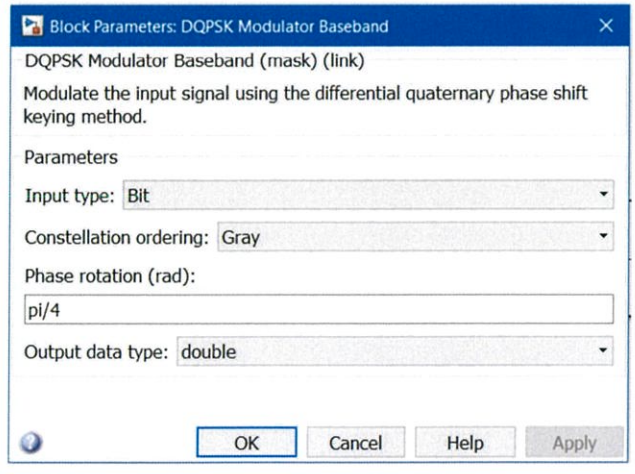
โดยข้อมูลที่เราจะส่งสามารถปรับได้จากบล็อก Message Generator และในการปรับพารามิเตอร์ให้สามารถส่งข้อมูลด้วยอัตรา 36 Kbps เพื่อให้ตรงตามมาตรฐาน TETRA สามารถ

ปรับได้จากค่า Sample time ใน Block parameter ของบล็อก Bernoulli Binary Generator โดย Block parameter ของ Bernoulli Binary Generator แสดงดังรูปที่ 3.10



รูปที่ 3.10 Block parameter ของ Bernoulli Binary Generator

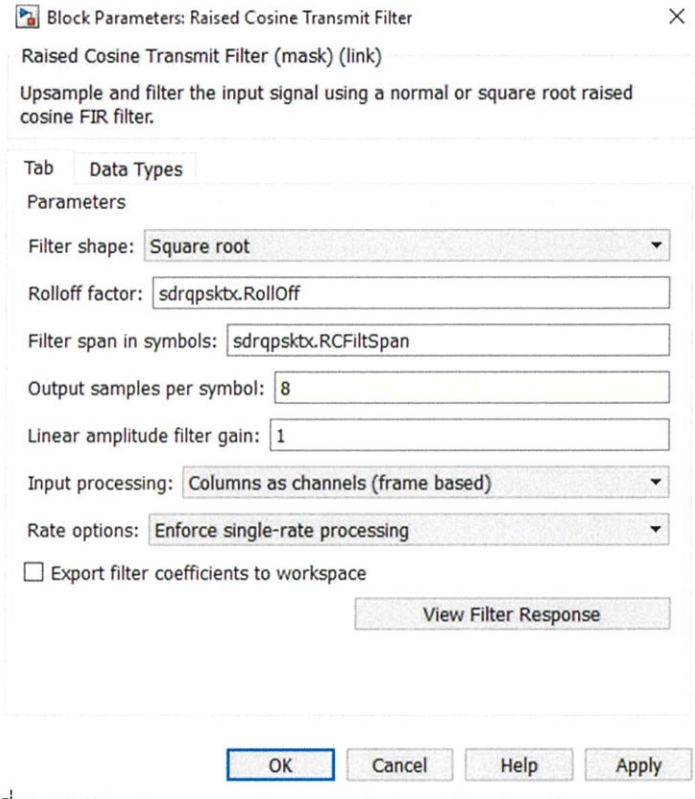
โดยบล็อก DQPSK Modulator Baseband จากรูปที่ 3.3 ที่นำมาใช้มีการปรับพารามิเตอร์ต่างๆ โดย Block parameter ของ DQPSK Modulator Baseband แสดงดังรูปที่ 3.11



รูปที่ 3.11 Block parameter ของ DQPSK Modulator Baseband

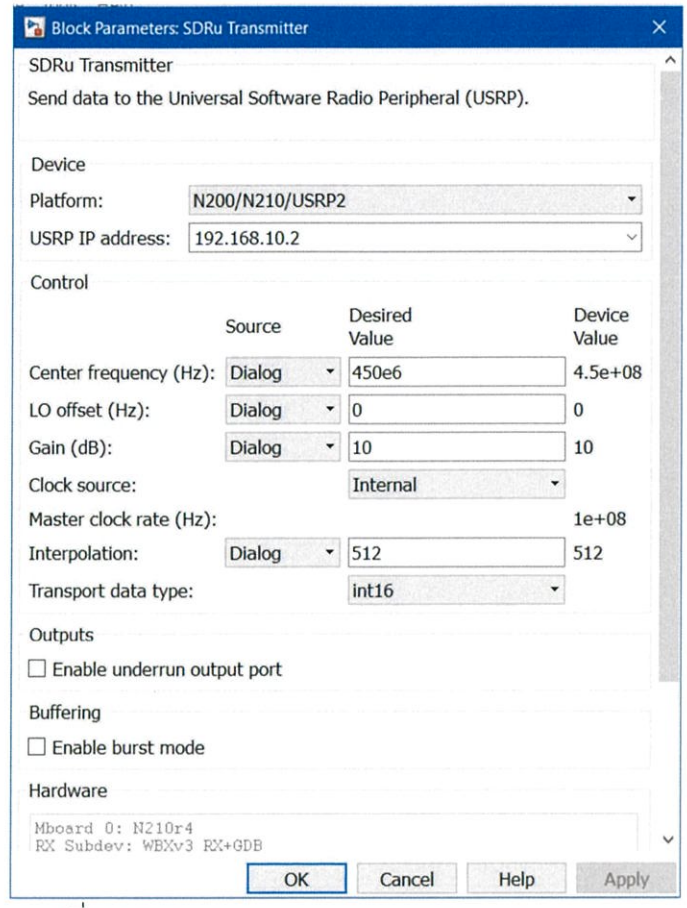
โดยผู้จัดทำได้ปรับค่า Phase rotation ให้เท่ากับ pi/4 โดยอัตราการส่งข้อมูลหลังจากข้อมูลผ่านบล็อก DQPSK Modulator Baseband จะเท่ากับ 18 kSymbol/sec ซึ่งบล็อก Raised Cosine Transmit Filter ถูกนำมาใช้ในการ Upsample และ จัดรูป Pulse เพื่อลดการเกิด

Inter Symbol Interference (ISI) โดยปรับค่า Output samples per symbol ให้มีค่าเท่ากับ 8 เมื่อข้อมูลผ่าน Raised Cosine Transmit Filter อัตราการส่งข้อมูลจะเท่ากับ 144 kSample/sec โดย Block parameter ของ Raised Cosine Transmit Filter ดังรูปที่ 3.12



รูปที่ 3.12 Block parameter ของ Raised Cosine Transmit Filter

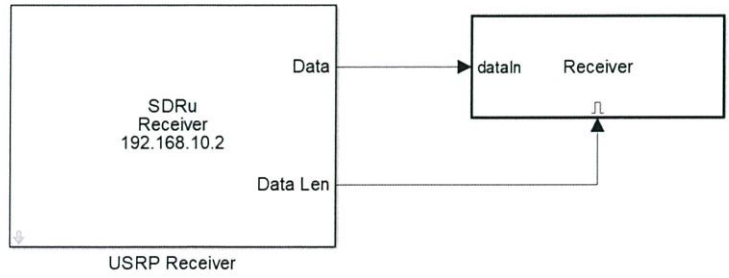
จากรูปที่ 3.3 บล็อก SDRu Transmitter ถูกนำมาใช้เพื่อส่งข้อมูลไปยังอุปกรณ์ USRP โดยปรับ Center frequency ให้เท่ากับ 450 MHz ซึ่ง Block parameter ของ SDRu Transmitter แสดงดังรูปที่ 3.13



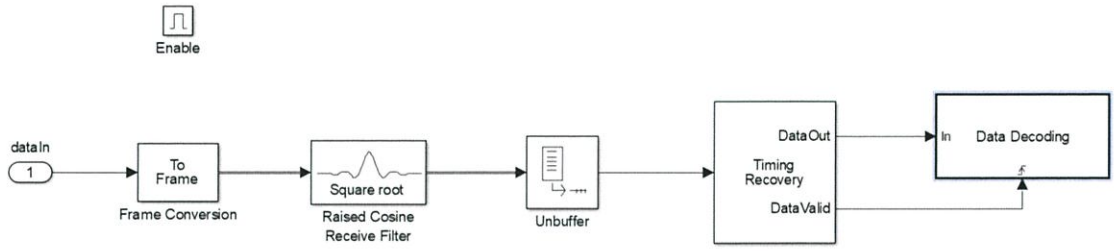
รูปที่ 3.13 Block parameter ของ SDRu Transmitter

3.2.1.2 ภาครับ

ผู้จัดทำได้ทำการปรับพารามิเตอร์ในบล็อกต่างๆ ใน Communications System Toolbox Support Package for USRP Radio example ที่เป็นภาครับเพื่อให้สามารถรับสัญญาณข้อมูลที่เป็น Text โดยใช้งานที่ความถี่ 450 MHz บนมาตรฐาน TETRA ได้ โดย Simulink Block ของภาครับและ Subsystem ของบล็อก Receiver แสดงดังรูปที่ 3.14 และรูปที่ 3.15 ตามลำดับ

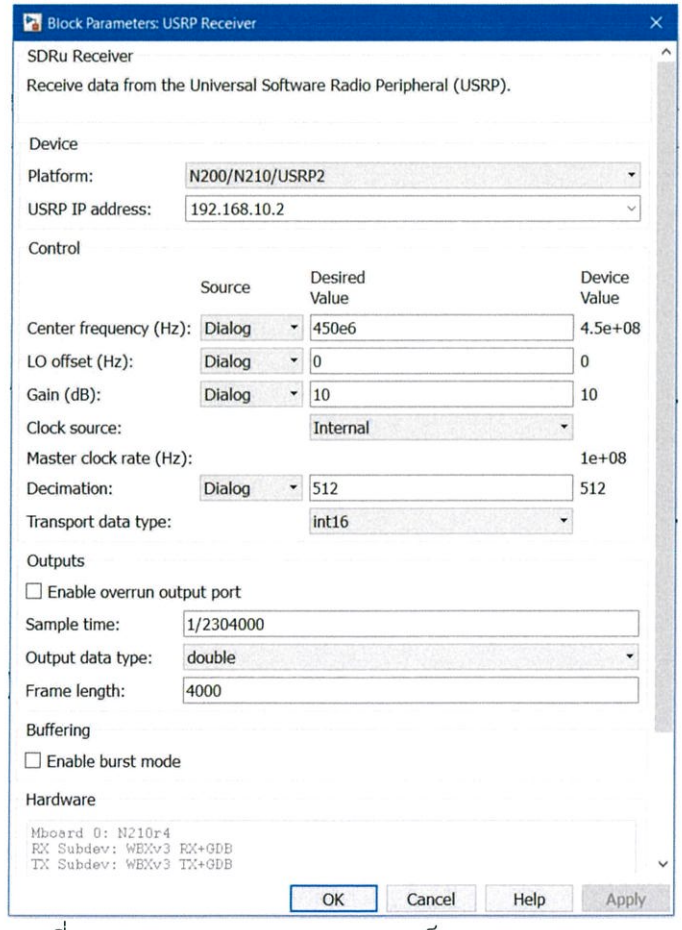


รูปที่ 3.14 Simulink Block ของภาครับ



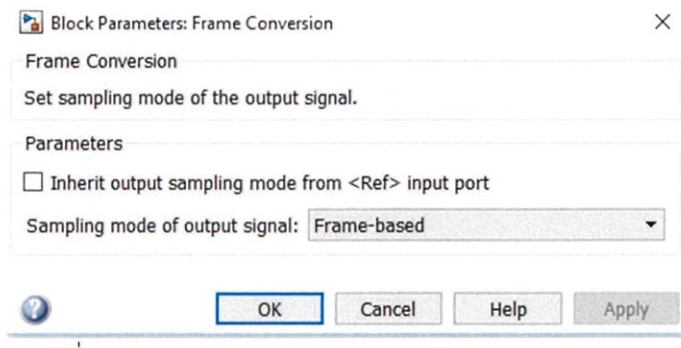
รูปที่ 3.15 Subsystem ของบล็อก Receiver

จากรูปที่ 3.14 บล็อก SDRu Receiver ถูกนำมาใช้เพื่อรับข้อมูลจาก USRP โดยปรับค่า Center frequency ให้มีค่าเท่ากับ 450 MHz และกำหนด Sample time เท่ากับ 1/2304000 เพราะฉะนั้นอัตราการรับข้อมูลจะเท่ากับ 2.304 MSample/sec โดย Block parameter ของบล็อก SDRu Receiver แสดงดังรูปที่ 3.16



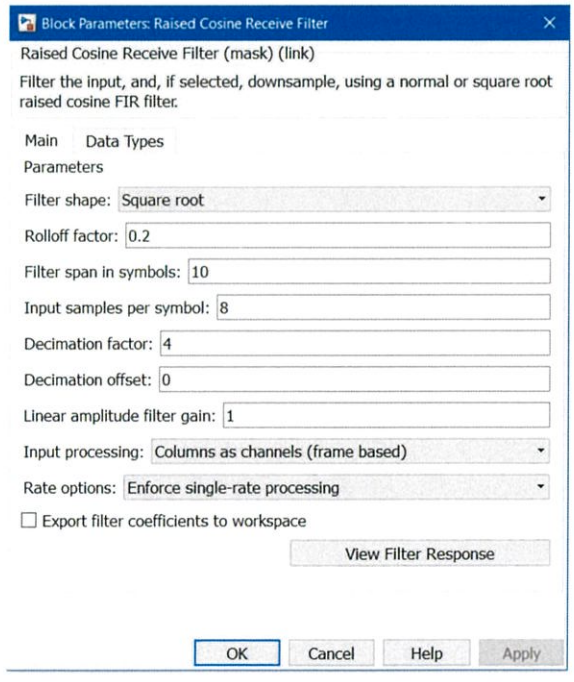
รูปที่ 3.16 Block parameter ของบล็อก SDRu Receiver

จากรูปที่ 3.15 บล็อก Frame Conversion ถูกนำมาใช้เพื่อตั้งค่า sampling mode ของสัญญาณเอาต์พุตโดยปรับ sampling mode of output signal ให้เป็น Frame-based โดย Block parameter ของ Frame Conversion แสดงดังรูปที่ 3.17



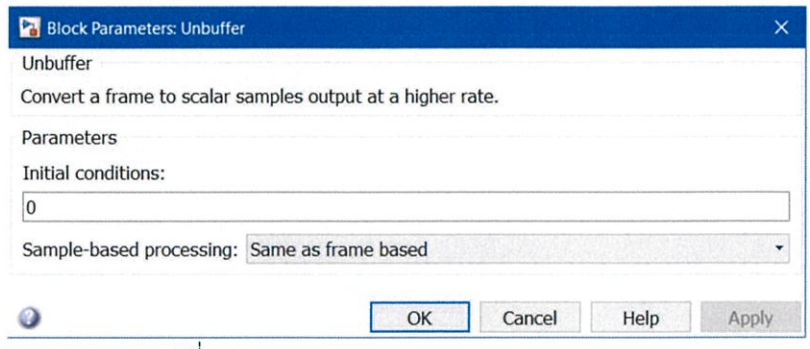
รูปที่ 3.17 Block parameter ของ Frame Conversion

จากรูปที่ 3.15 บล็อก Raised Cosine Receive Filter ถูกนำมาใช้เพื่อ Downsample โดย Block parameter ของ Raised Cosine Receive Filter แสดงดังรูปที่ 3.18



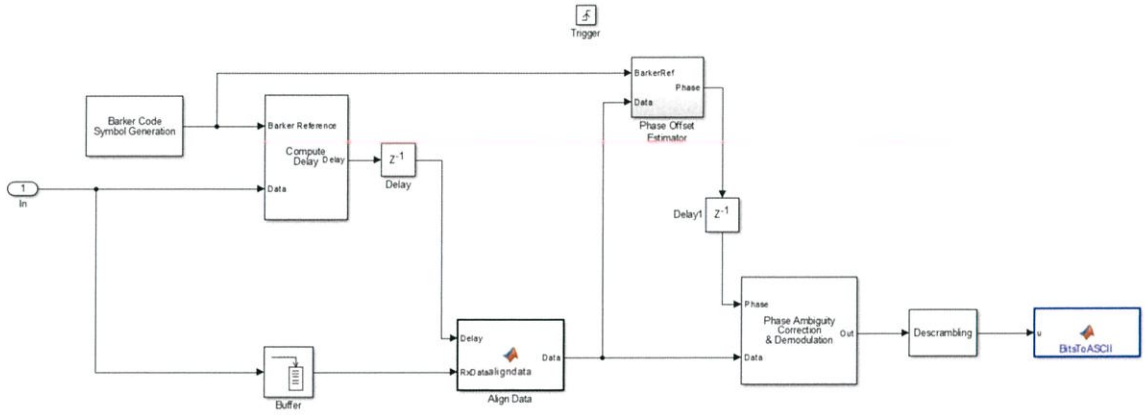
รูปที่ 3.18 Block parameter ของ Raised Cosine Receive Filter

จากรูปที่ 3.15 บล็อก Unbuffer ถูกนำมาใช้เพื่อแปลง frame ให้เป็น scalar samples ในอัตราที่สูงกว่า โดย Block parameter ของ Unbuffer แสดงดังรูปที่ 3.19



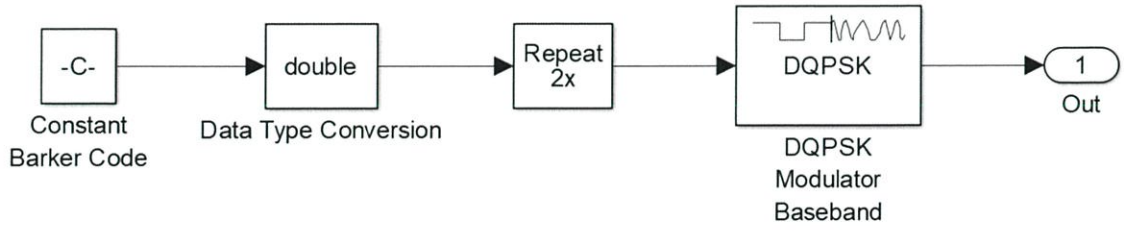
รูปที่ 3.19 Block parameter ของ Unbuffer

จากรูปที่ 3.15 บล็อก Data decoding มี Subsystem แสดงดังรูปที่ 3.20



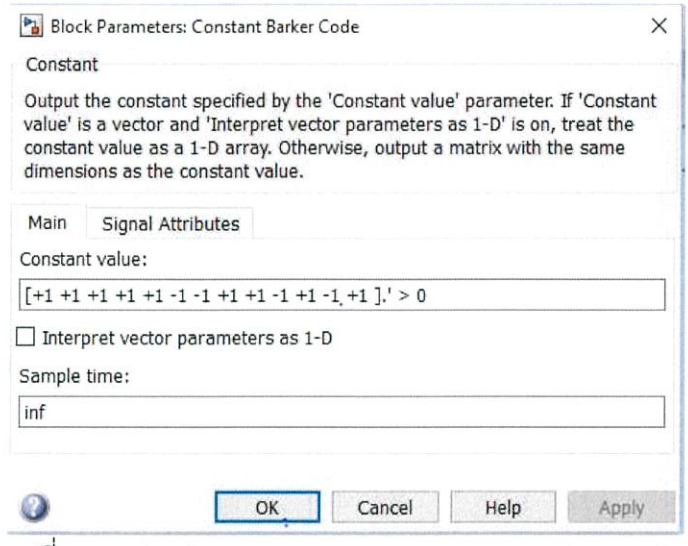
รูปที่ 3.20 Subsystem ของบล็อก Data decoding

โดยในบล็อก Barker code Symbol Generation มี Subsystem ดังรูปที่ 3.21



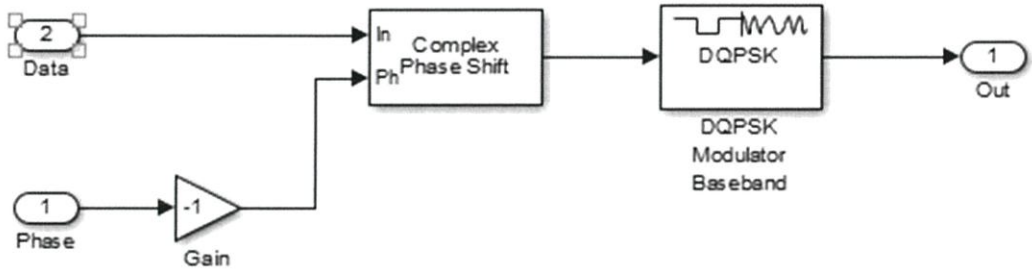
รูปที่ 3.21 Subsystem ของ Barker code Symbol Generation

จากภาคส่งได้มีการนำ Barker Code มาใช้เป็น Header ของข้อมูล ดังนั้น ภาครับจึงต้องสร้าง Barker code เพื่อใช้หา Header ของข้อมูลที่รับได้ ดังนั้นค่า Constant value ในบล็อก Constant Barker Code ในภาครับจึงต้องมีค่าเหมือนกับค่า Constant value ในบล็อก Constant Barker Code ในภาคส่ง โดย Block parameter ของ Constant Barker Code ในภาครับแสดงดังรูปที่ 3.22



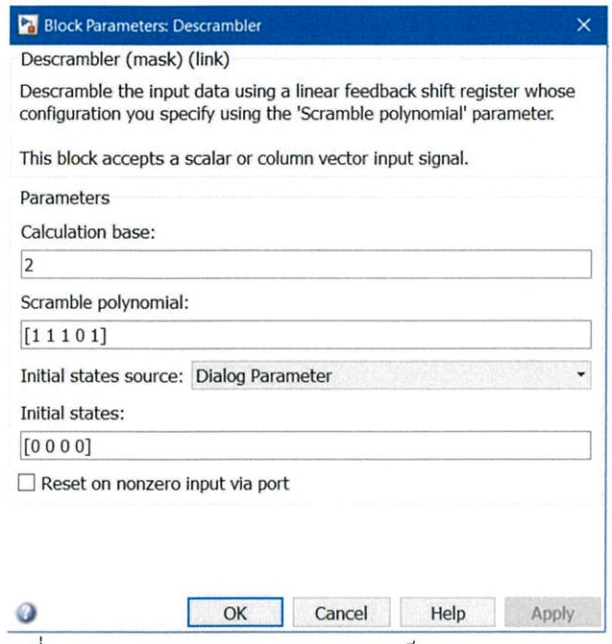
รูปที่ 3.22 Block parameter ของ Constant Barker Code

จากรูปที่ 3.20 ในบล็อก Phase Ambiguity Correction & Demodulation มี Subsystem แสดงดังรูปที่ 3.23



รูปที่ 3.23 Subsystem ของ Phase Ambiguity Correction & Demodulation

จากรูปที่ 3.20 บล็อก Descrambling ถูกนำมาใช้เพื่อไม่ให้ข้อมูลบิต 0 หรือบิต 1 อยู่ติดกันมากจนเกินไป โดย Block parameter ของบล็อก Descrambler อยู่ในบล็อก Descrambling แสดงดังรูปที่ 3.24

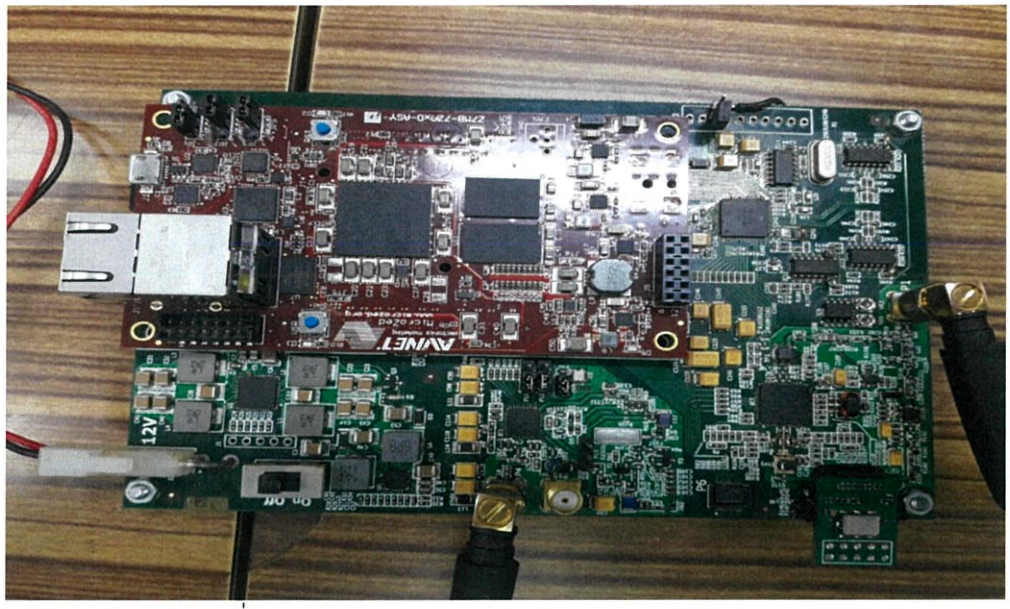


รูปที่ 3.24 Block parameter ของบล็อก Descrambler

จากรูปที่ 3.20 เมื่อสัญญาณข้อมูลผ่านบล็อก Descrambling แล้ว สัญญาณข้อมูลจะเข้าสู่บล็อก BitToASCII เพื่อสร้างสัญญาณข้อมูลที่เป็น Text กลับคืนมา และเมื่อทำการรัน Simulink block ทั้งภาคส่งและภาครับ ค่าที่รับได้จะแสดงในบล็อก BitToASCII ซึ่งปรากฏบน command window ของโปรแกรม MATLAB

3.2.2 การใช้งาน Advance Digital Radio Baseband Board และ SDR dongle

Advance Digital Radio Baseband Board เป็นบอร์ดที่ทาง บริษัท กสท. โทรคมนาคม จำกัด ได้จัดทำขึ้นซึ่งมีต้นแบบมาจากบอร์ด Evaluation Kit ทั้ง EV9810, EV9980 และ EV9942 โดยใช้ IC Si570 เป็น Oscillator เพื่อสร้างความถี่ RF โดยบอร์ด Advance Digital Radio Baseband Board แสดงดังรูปที่ 3.25



รูปที่ 3.25 Advance Digital Radio Baseband Board

โดยบอร์ด Advance Digital Radio Baseband Board จะใช้ FPGA Zynq7000 ในการอ่านเขียนค่ารีจีสเตอร์ (Register) ของ CMX 981, CMX 998, CMX 994 และ Si570 โดยรีจีสเตอร์ที่สำคัญ มีดังนี้

IC CMX981

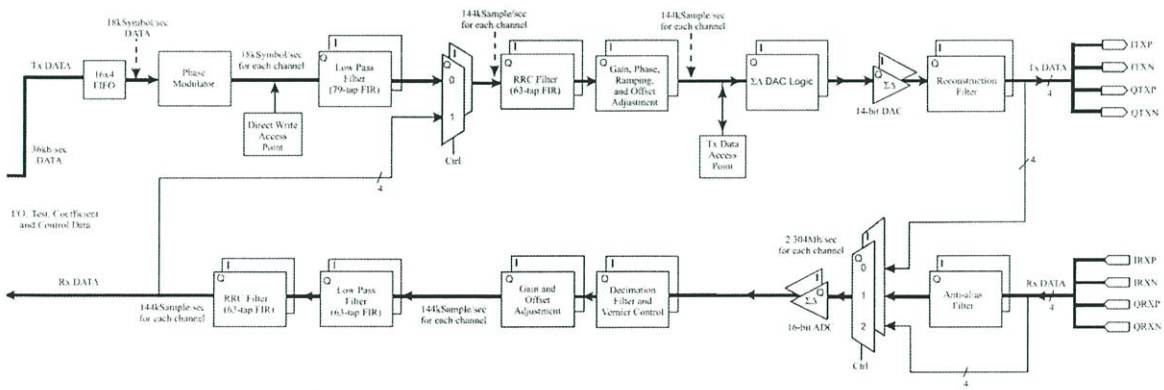
- รีจีสเตอร์ 00 Configuration control register 1
- รีจีสเตอร์ 01 Configuration control register 2
- รีจีสเตอร์ 03 Transmit set-up register
- รีจีสเตอร์ 04-07 Transmit data FIFO register
- รีจีสเตอร์ 08 Receive Set-up register 1
- รีจีสเตอร์ 09 Receive Set-up register 2
- รีจีสเตอร์ 11 Clock stop control register
- รีจีสเตอร์ 13 Loop back control register
- รีจีสเตอร์ 20-21 Transmit I channel phase register
- รีจีสเตอร์ 22-23 Transmit I channel gain register
- รีจีสเตอร์ 24-25 Transmit I channel offset register
- รีจีสเตอร์ 26-27 Transmit Q channel phase register
- รีจีสเตอร์ 28-29 Transmit Q channel gain register

- รีจีสเตอร์ 2A-2B Transmit Q channel offset register
- รีจีสเตอร์ 38-39 Receive I channel data access point
- รีจีสเตอร์ 3A-3B Receive Q channel data access point
- รีจีสเตอร์ 3C-3D Transmit I channel data access point
- รีจีสเตอร์ 3E-3F Transmit Q channel data access point

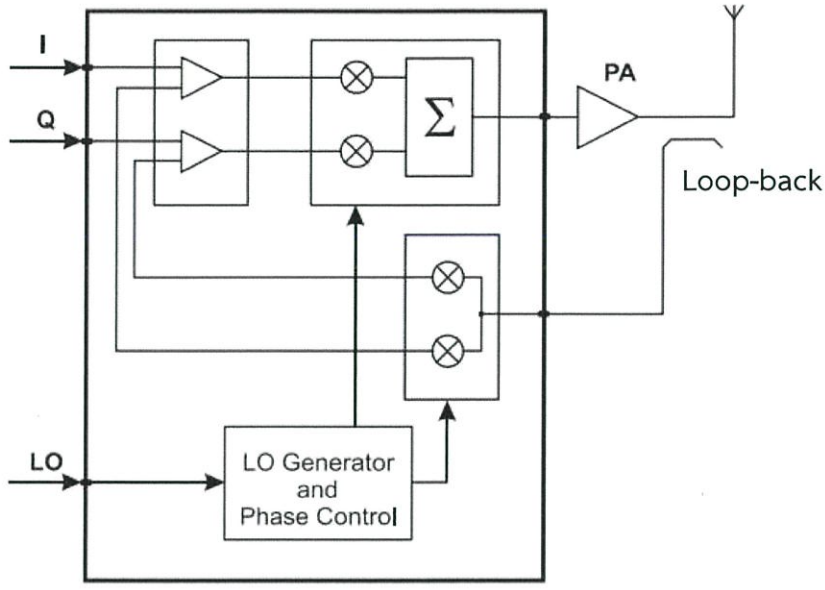
IC CMX998

- รีจีสเตอร์ 02 General Command, 8-bit write only
- รีจีสเตอร์ 03 Phase Control Register, 8-bit write only
- รีจีสเตอร์ 05 Forward Path Gain Control, 8-bit write only
- รีจีสเตอร์ 06 Feedback Path Gain Control, 8-bit write only
- รีจีสเตอร์ 08 Frequency Control Register, 8-bit write only

ในงานวิจัยนี้ IC CMX994 ซึ่งเป็นส่วนของภาครับยังไม่ถูกนำมาพิจารณาเนื่องจากทางผู้จัดทำต้องการให้ Advance Digital Radio Baseband Board สามารถใช้ส่งข้อมูลได้ก่อนโดยใช้ IC CMX981 เป็นตัว Processor หลัก, IC CMX998 เป็นส่วนของภาคส่งและ IC Si570 เป็นตัว Oscillator โดยบล็อกไดอะแกรมการรับส่งข้อมูลที่สำคัญของ CMX981 แสดงดังรูปที่ 3.26 และบล็อกไดอะแกรมอย่าง่ายของ CMX998 แสดงดังรูปที่ 3.27

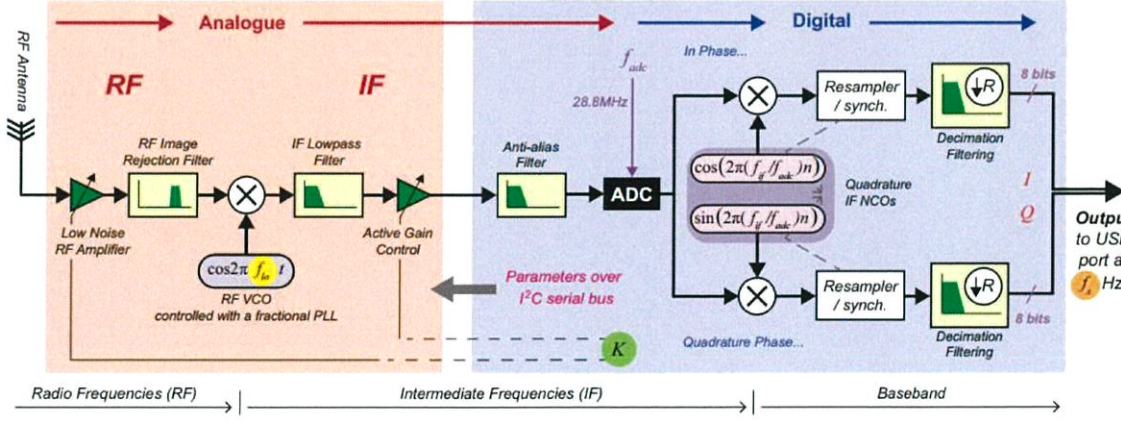


รูปที่ 3.26 บล็อกไดอะแกรมการรับส่งข้อมูลของ CMX981



รูปที่ 3.27 บล็อกไดอะแกรมของ CMX998

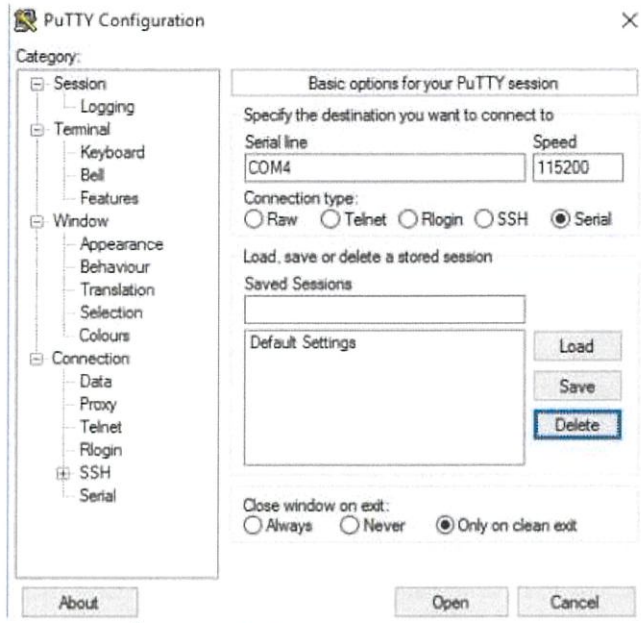
SDR Dongle ถูกนำมาใช้เพื่อดูสเปกตรัมของ Advance Digital Radio Baseband Board โดยใช้ซอฟแวร์ SDRSharp บล็อกไดอะแกรมการทำงานของ SDR Dongle แสดงดังรูปที่ 3.28



รูปที่ 3.28 บล็อกไดอะแกรมการทำงานของ SDR Dongle

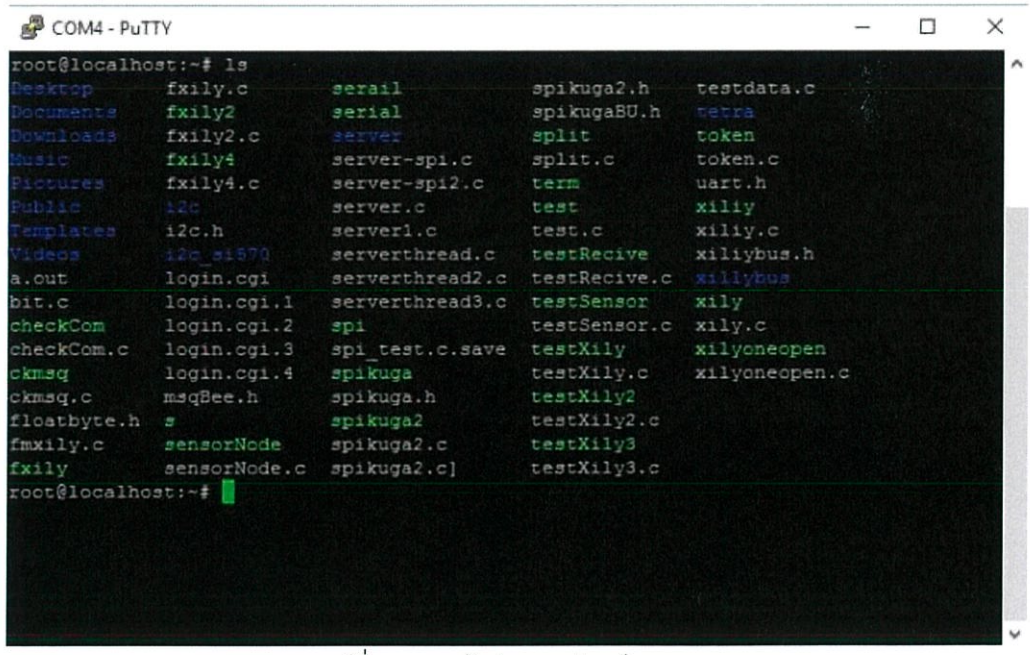
3.2.2.1 ซอฟแวร์ที่ใช้

1) ซอฟแวร์ที่ใช้ร่วมกับ Advance Digital Radio Baseband Board คือ PuTTY โดยเป็นโปรแกรม Remote Server หรือ SSH (Secure Shell) โดยโปรแกรมนี้สามารถสั่งงาน Server ด้วย command line โดยจะใช้เชื่อมต่อไปยัง Server ที่เป็น Linux เมื่อเปิดซอฟแวร์ PuTTY ขึ้นมาจะขึ้นหน้าต่าง PuTTY Configuration แสดงดังรูปที่ 3.29



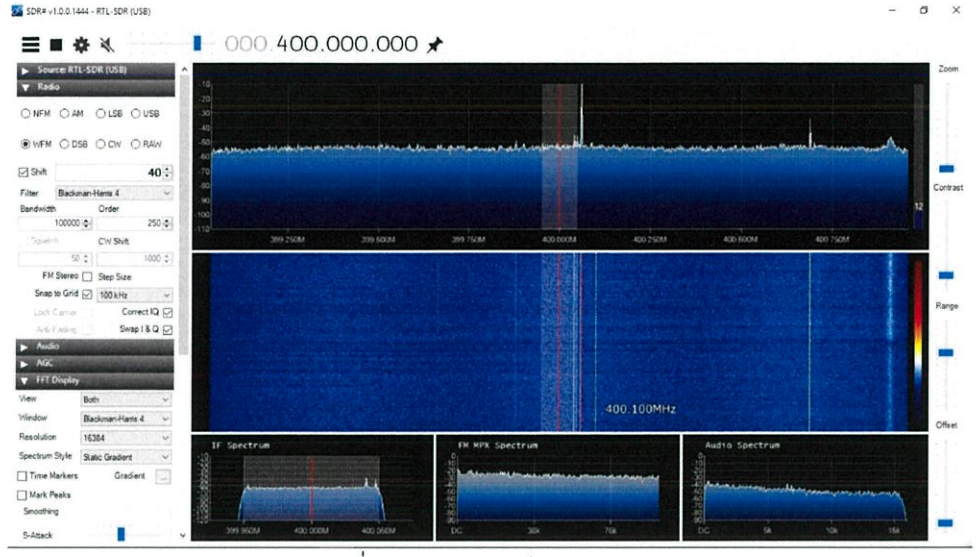
รูปที่ 3.29 หน้าต่าง PuTTY Configuration

เมื่อทำการกำหนด Serial line และ speed แล้วกด open จะขึ้นหน้าต่างซอฟแวร์ PuTTY ดังรูปที่ 3.30 โดยตัวอักษรสีน้ำเงินหมายถึง Directory file ตัวอักษรสีเขียวหมายถึง Execute file และตัวอักษรสีขาวหมายถึง Text file



รูปที่ 3.30 หน้าต่างซอฟแวร์ PuTTY

2) ซอฟแวร์ที่ใช้ร่วมกับ SDR Dongle คือ SDRSharp โดยนำมาใช้เพื่อตรวจสอบว่า Advance Digital Radio Baseband Board สามารถส่งสัญญาณที่ความถี่ต่างๆได้ โดยดูจากสเปกตรัมในซอฟแวร์ SDRSharp ซึ่งหน้าต่างซอฟแวร์ SDRSharp แสดงดังรูปที่ 3.31

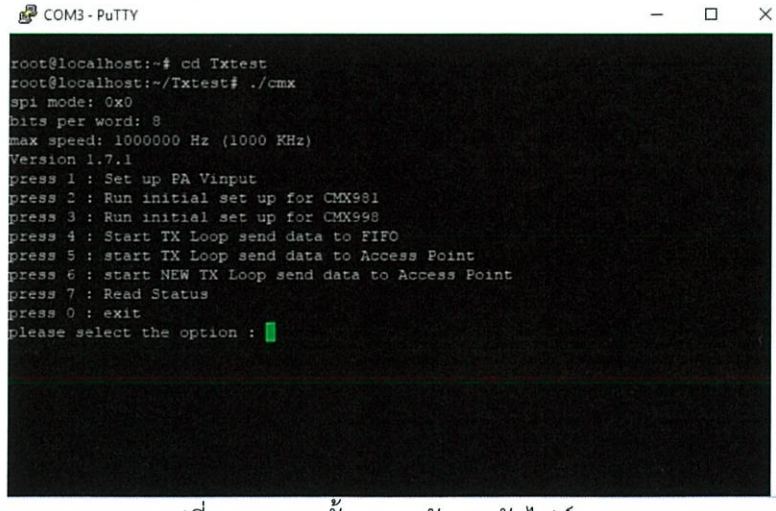


รูปที่ 3.31 ซอฟแวร์ SDRSharp

3.2.2.2 ขั้นตอนการใช้งาน

โดย list ทั้งหมดที่แสดงในรูปที่ 3.30 อยู่ใน directory ที่ชื่อว่า root ซึ่งlist ทั้งหมดได้มีผู้จัดทำไว้ก่อนแล้ว โดยไฟล์ที่ผู้จัดทำได้นำมาทดลองและพัฒนาคือ Directory: /root/Txtest ซึ่งมีขั้นตอนดังนี้

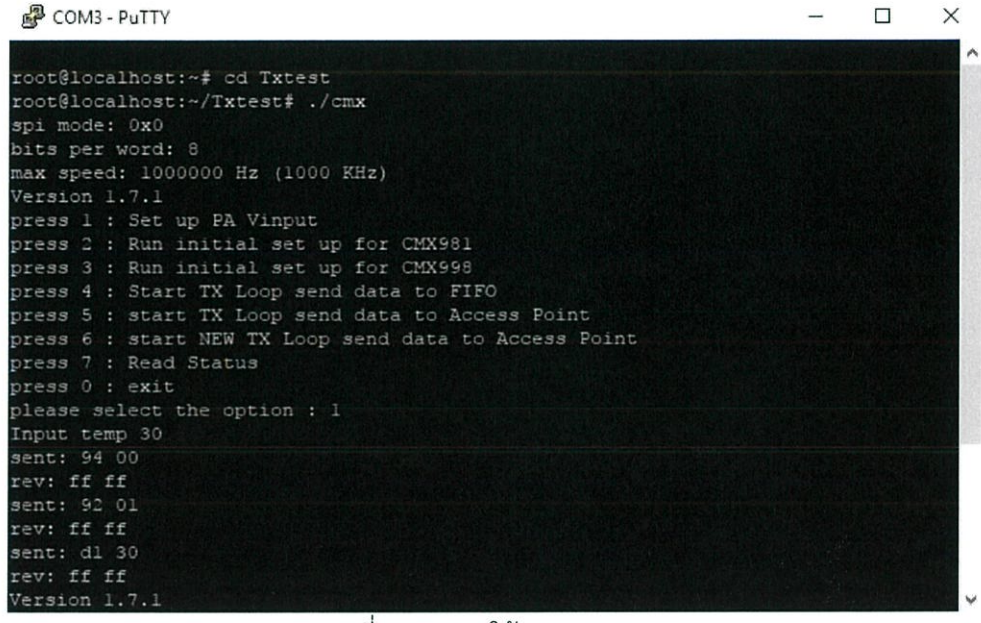
1) ทำการรันไฟล์ที่มีชื่อว่า cmx โดยใช้คำสั่ง ./cmx เมื่อทำการรันไฟล์ cmx แล้วจะแสดงฟังก์ชันการใช้งานดังรูปที่ 3.32



รูปที่ 3.32 เมนูทั้งหมดหลังจากรันไฟล์ cmx

โดยฟังก์ชันการใช้งานทั้งหมดมีดังนี้

- press 1 : Set up PA Vinput เป็นการกำหนดแรงดันให้ภาคขยายโดยโปรแกรมจะรอรับเลขฐาน 16 โดยที่แรงดันที่ต้องใช้คือ 1.1v โดยต้องป้อนค่า 30 ลงไป แสดงดังรูปที่ 3.33



รูปที่ 3.33 การใช้งาน press 1

- press 2 : Run initial set up for CMX981 เป็นการกำหนดค่าเริ่มต้นให้กับรีจีสเตอร์ของ IC CMX981
- press 3 : Run initial set up for CMX998 เป็นการกำหนดค่าเริ่มต้นให้กับรีจีสเตอร์ของ IC CMX998
- press 4 : Start Tx Loop send data to FIFO เป็นการกำหนดค่าให้กับรีจีสเตอร์ของ IC CMX981 ให้สามารถส่งข้อมูลแบบลูปโดยส่งข้อมูลไปยัง FIFO ได้
- press 5 : Start Tx Loop send data to Access point เป็นการกำหนดค่าให้กับรีจีสเตอร์ของ IC CMX981 ให้สามารถส่งข้อมูลแบบลูปโดยส่งข้อมูลไปยัง Access point ได้
- press 6 : Start Tx Loop send data to Access point เป็นการกำหนดค่าให้กับรีจีสเตอร์ของ IC CMX981 ให้สามารถส่งข้อมูลแบบลูปโดยส่งข้อมูลไปยัง Access point ได้ และมีการตรวจสอบว่าข้อมูลที่ถูกส่งไปแล้ว สำเร็จหรือไม่
- press 0 : exit เพื่อใช้ออกจากโปรแกรม cmx โดยใช้คำสั่ง exit(0)

โปรมแกรมที่กล่าวมาทั้งหมดสามารถแก้ไขและปรับปรุงได้จากไฟล์ CMX981.c ที่อยู่ใน directory:/root/Txtest

2) เมื่อทำการแก้ไขและปรับปรุงในไฟล์ CMX981.c และกำหนดฟังก์ชันในโปรแกรม cmx เพื่อให้สามารถส่งข้อมูลได้เสร็จเรียบร้อยแล้ว เราจะรันไฟล์ที่ชื่อว่า Si570 เพื่อกำหนดความถี่ให้กับ IC Si570 โดยมีรูปแบบคำสั่งดังนี้ ./si570 –P –d [เลขฐาน 16 หกชุด] –w โดยเลขฐาน 16 หกชุดในคำสั่งคือค่าที่เขียนให้รีจีสเตอร์ของ IC Si570 นั่นเอง โดยความสัมพันธ์ของเลขฐาน 16 หกชุดกับความถี่ที่ใช้งานแสดงดังตารางที่ 3.1 โดยผู้จัดทำได้ใช้งาน Advance Digital Radio Baseband Board ที่ความถี่ 450 MHz

ตารางที่ 3.1 ความสัมพันธ์ของข้อมูลเลขฐาน 16 หกชุดกับความถี่ที่ใช้งาน

ความถี่ที่ใช้งาน	เลขฐาน 16 หกชุด
400 MHz	60 43 10 01 41 20
450 MHz	40 02 F4 01 35 A8
800 MHz	60 03 10 01 41 20
940 MHz	40 03 15 9A DD 05
1000 MHz	20 02 BC 01 1E B8

4) เมื่อทำการรันคำสั่ง ./si570 –P –d 40 02 F4 01 35 A8 –w ซึ่งเป็นการใช้งานที่ความถี่ 450 MHz โดยเราจะใช้ SDR Dongle เพื่อดูสเปกตรัมโดยใช้ซอฟแวร์ SDRSharp

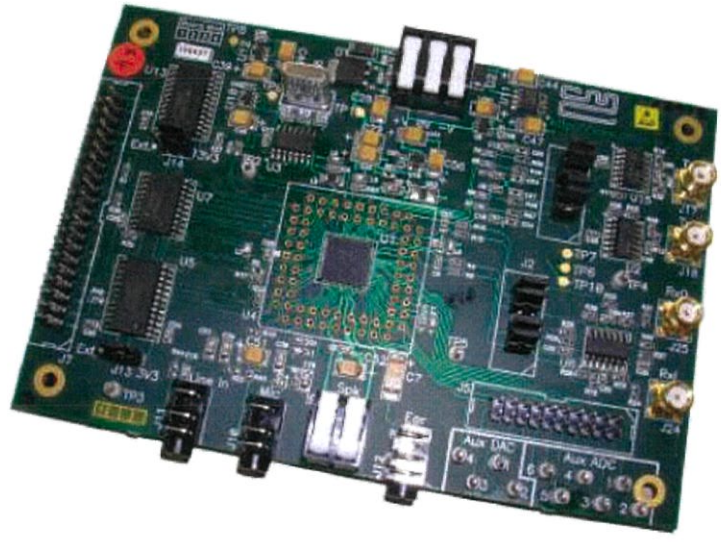
3.2.2.3 ปัญหาที่พบ

จากการดูสเปกตรัมของบอร์ด Advance Digital Radio Baseband Board โดยใช้ซอฟแวร์ SDRSharp ทำให้สรุปได้ว่าไม่สามารถทำให้บอร์ด Advance Digital Radio Baseband Board ใช้ส่งข้อมูลได้ ทางผู้จัดทำจึงได้นำ schematic ของบอร์ด Advance Digital Radio Baseband Board และบอร์ด Evaluation Kit ซึ่งเป็นต้นแบบของ Advance Digital Radio Baseband Board มาเปรียบเทียบเพื่อตรวจสอบ ซึ่งจากการตรวจสอบสรุปได้ว่าไม่มีข้อผิดพลาดทาง schematic ดังนั้นทางผู้จัดทำจึงได้นำบอร์ด Evaluation Kit 9810 (EV9810) มาศึกษาและทดลอง เพื่อดูรูปแบบของสัญญาณเอาต์พุตของทั้งสองบอร์ดว่ามีรูปแบบเหมือนกันหรือไม่ ดังนั้นผู้จัดทำได้เขียนคำสั่งโดยใช้ภาษาซี เพื่อกำหนดค่าในรีจีสเตอร์ให้สามารถตรวจสอบรูปแบบสัญญาณเอาต์พุตที่ขา TxI ของบอร์ด Advance Digital Radio Baseband Board ได้ โดยมีวิธีการคือ กำหนดค่าลงไปในรีจีสเตอร์ 3C-3D (Transmit I channel data access point) ซึ่งค่าที่เขียนลงไปเป็นเลขฐาน 16 เริ่มต้นมีค่าเป็น 0000 โดยเพิ่มทีละ 0001 จนถึง 3FFF เมื่อจนถึง 3FFF จะลดลงทีละ 0001 จนถึง 0000 วนเช่นนี้ไปเรื่อยๆ และเมื่อทำการ

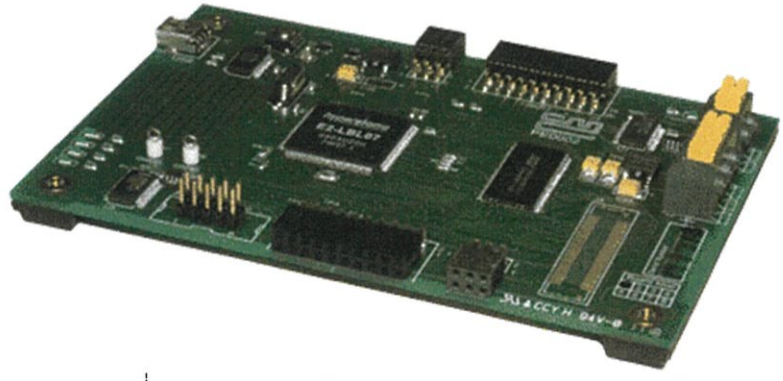
รันคำสั่งเราจะใช้ Oscilloscope วัดสัญญาณที่ขา TxI เพื่อดูรูปแบบของสัญญาณเอาต์พุต ของบอร์ด Advance Digital Radio Baseband Board เพื่อนำไปเปรียบเทียบกับบอร์ด EV9810

3.2.3 การใช้งาน Evaluation Kit 9810 และ PE0002 (Evaluation Kit Interface Card)

ทางผู้จัดทำได้นำบอร์ด Evaluation Kit 9810 (EV9810) และบอร์ด PE0002 (Evaluation Kit Interface Card) มาใช้งานเพื่อเปรียบเทียบรูปแบบของสัญญาณเอาต์พุต TxI กับบอร์ด Advance Digital Radio Baseband Board โดยบอร์ด PE0002 (Evaluation Kit Interface Card) เป็นระบบอินเทอร์เฟซสำหรับใช้กับชุดประเมินของบริษัท CML ซึ่งจะช่วยลดความซับซ้อนในการประเมินผลและออกแบบขั้นตอนในการทดลอง โดยบอร์ด Evaluation Kit 9810 (EV9810) แสดงดังรูปที่ 3.34 และบอร์ด PE0002 (Evaluation Kit Interface Card) แสดงดังรูปที่ 3.35



รูปที่ 3.34 บอร์ด Evaluation Kit 9810 (EV9810)



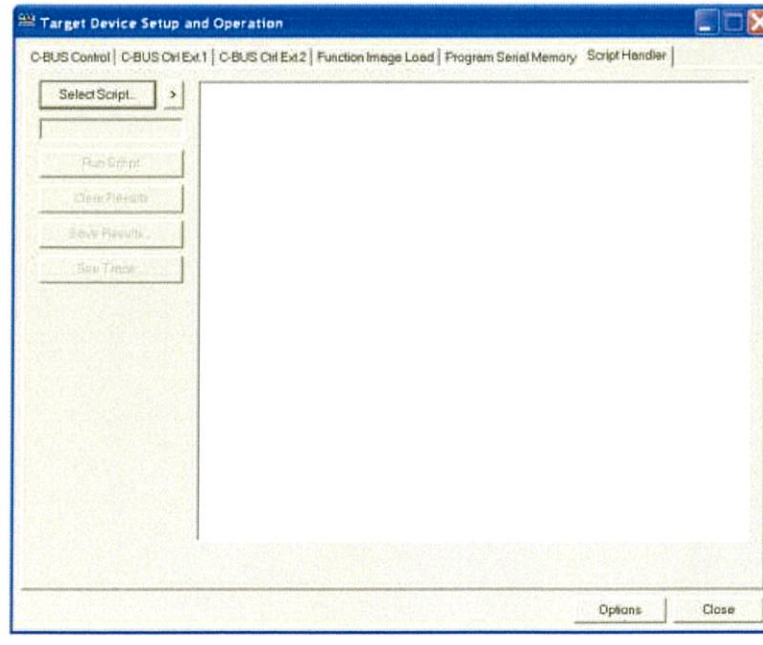
รูปที่ 3.35 PE0002 Evaluation Kit Interface Card

รีจีสเตอร์ของบอร์ด EV9810 ที่สำคัญในการใช้งานมีดังนี้

- รีจีสเตอร์ 03 Transmit set-up register
- รีจีสเตอร์ 13 Loop back control register
- รีจีสเตอร์ 22-23 Transmit I channel gain register
- รีจีสเตอร์ 38-39 Receive I channel data access point
- รีจีสเตอร์ 3C-3D Transmit I channel data access point

3.2.3.1 ซอฟแวร์ที่ใช้

ซอฟแวร์ที่ใช้ร่วมกับ Evaluation Kit 9810 (EV9810) คือ ES000243 เป็นโปรแกรมที่ใช้ในการสื่อสารกับ PE0002 บนระบบปฏิบัติการ Window โดยการใช้งานโปรแกรม ES000243.EXE หากไม่สามารถติดต่อสื่อสารจะปรากฏข้อความว่าล้มเหลว แต่ถ้าหากสามารถสื่อสารได้จะปรากฏกล่องข้อความตามรูปที่ 2.21 โดยจะให้กดสวิตช์รีเซ็ต SW1 บน PE0002 เฟิร์มแวร์จะดาวน์โหลดโดยอัตโนมัติไปยังบอร์ด PE0002 และกล่องตอบโต้จะเข้าสู่โปรแกรมหลัก โดยสามารถเชื่อมต่ออัตโนมัติกับคอมพิวเตอร์ได้เมื่อต่อ PE0002 โดยส่วนของโปรแกรม ES000243 ที่ใช้ในงานวิจัยนี้คือ The Script Handler Tab แสดงดงรูปที่ 3.36



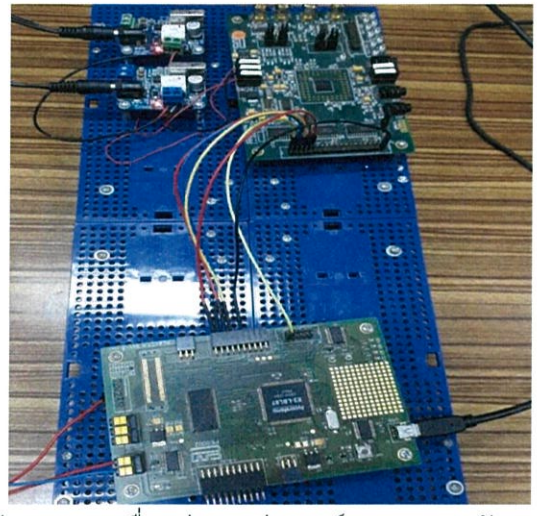
รูปที่ 3.36 The Script Handler Tab

3.2.3.2 ขั้นตอนการใช้งาน

1) ทำการเชื่อมต่อบอร์ด EV9810 กับบอร์ด PE0002 โดยมีลักษณะการเชื่อมต่อแบบ C-BUS แสดงดังตารางที่ 3.2 และรูปที่ 3.37

ตารางที่ 3.2 ลักษณะการเชื่อมต่อระหว่างบอร์ด EV9810 กับบอร์ด PE0002

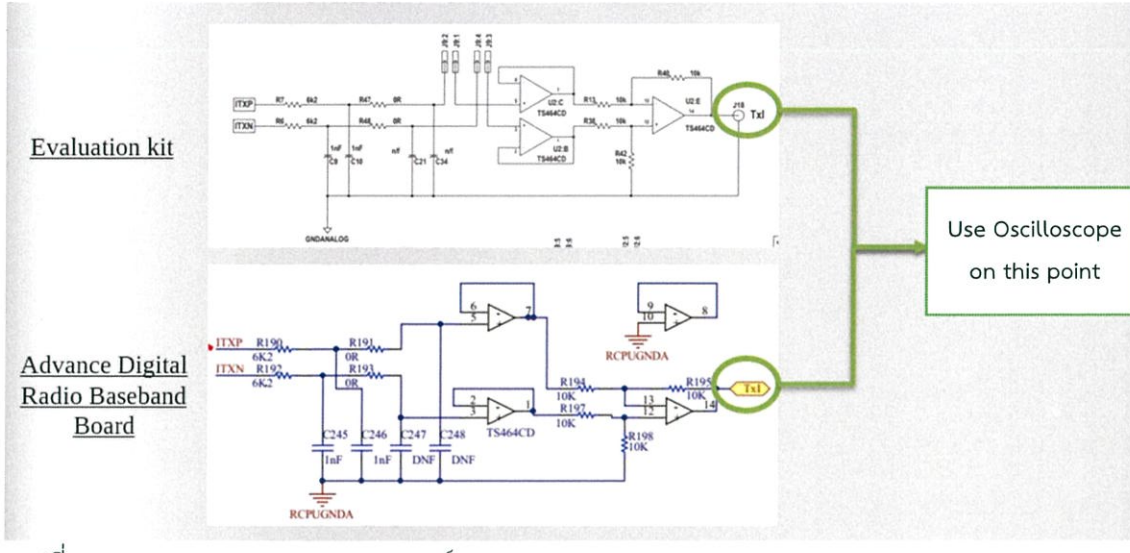
EV9810 (connector/pin)	description	PE0002 (connector/pin)	description
J7/40	GND	J3/12	GND
J7/39	CBUSEN	J6/10	3.3v power supply
J7/37	CDATA	J3/4	CDATA
J7/35	RDATA	J3/8	RDATA
J7/33	CCLK	J3/6	SCLKBUS
J7/31	CSN	J3/2	CSN



รูปที่ 3.37 ลักษณะการเชื่อมต่อระหว่างบอร์ด EV9810 กับบอร์ด PE0002

2) ทำการเขียนสคริปต์ (Script Language Reference) โดยภาษาสคริปต์นี้ได้รับการพัฒนาขึ้นสำหรับการประเมิน ICs รุ่นใหม่ของ CML ซึ่งช่วยลดความซับซ้อนในการประมวลผลสคริปต์มีความเรียบง่ายเพื่อให้ผู้ใช้สามารถอ่านและเขียนพอร์ต C-BUS โดยผู้จัดทำได้เขียนสคริปต์ขึ้นเพื่อกำหนดค่าในรีจีสเตอร์ให้สามารถตรวจสอบรูปแบบสัญญาณเอาต์พุตที่ขา TxI ของบอร์ด EV9810 ได้ โดยมีวิธีการคือกำหนดค่าเอาต์พุตลงในรีจีสเตอร์ 3C-3D (Transmit I channel data access point) ซึ่งค่าเอาต์พุตที่เขียนลงไปเป็นเลขฐาน 16 เริ่มต้นมีค่าเป็น 0000 แล้วเพิ่มทีละ 0001 จนถึง 3FFF เมื่อถึง 3FFF จะลดลงทีละ 0001 จนถึง 0000 วนเช่นนี้ไปเรื่อยๆ

3) เมื่อทำการรันสคริปต์ที่หน้าต่าง The Script Handler Tab ในโปรแกรม ES000243 แล้ว เราจะใช้ Oscilloscope วัดสัญญาณเพื่อดูรูปแบบของสัญญาณที่ขา TxI ของบอร์ด EV9810 เพื่อเปรียบเทียบกับบอร์ด Advance Digital Radio Baseband Board โดย schematic ของขา TxI ทั้งสองบอร์ดแสดงดังรูปที่ 3.38



รูปที่ 3.38 schematic ขา TxI ของบอร์ด EV9810 และ Advance Digital Radio Baseband Board

บทที่ 4
ผลการวิจัย

ในส่วนของผลการวิจัยจะถูกแบ่งออกเป็นสามส่วน ส่วนแรกคือผลจากการใช้งานอุปกรณ์ USRP โดยจะแสดงผลการดำเนินงานที่ได้จากทางภาคส่งและภาครับ ประกอบด้วยข้อมูลที่ส่งและรับได้ constellation ภาคส่งและภาครับ รวมไปถึงการคำนวณหา bit rate ของระบบ ในส่วนที่สองเป็นผลจากการใช้งาน Advance Digital Radio Baseband Board โดยจะแสดงผลการใช้งานตัวบอร์ด และในส่วนสุดท้ายเป็นผลการเปรียบเทียบสัญญาณเอาต์พุตที่ขา TxI ระหว่าง Advance Digital Radio Baseband Board กับ Evaluation Kit 9810 แสดงผลการเปรียบเทียบสัญญาณที่วัดได้จาก Oscilloscope ของอุปกรณ์ทั้งสองตัว

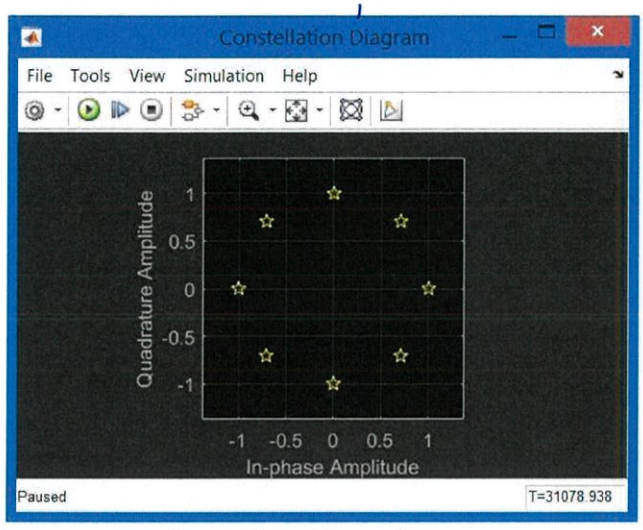
4.1 ผลจากการใช้งานอุปกรณ์ USRP

เนื่องจากโครงงานวิจัยในส่วนของการใช้งานอุปกรณ์ USRP เป็นการประยุกต์ใช้มาตรฐาน TETRA ซึ่งทางผู้จัดทำได้เลือกใช้ข้อมูลในภาคส่งเป็น Text คำว่า “I Love KMITL” ตามด้วยตัวเลข 00 ไปจนถึง 15 และวนกลับไปที่ 00 ใหม่ เพื่อที่ในฝั่งรับจะสามารถเห็นการเปลี่ยนแปลงของข้อมูลที่สามารถรับได้ โดยผลจากการเขียน Code ในการแปลง Text ที่ส่งเป็นบิตข้อมูล แสดงได้ดังรูปที่ 4.1

```
function msg = genMsg
%#codegen
persistent msgStrSet count;
if isempty(msgStrSet)
    count = 0;
    msgStrSet = ['I love KMITL 00';...
                 'I love KMITL 01';...
                 'I love KMITL 02';...
                 'I love KMITL 03';...
                 'I love KMITL 04';...
                 'I love KMITL 05';...
                 'I love KMITL 06';...
                 'I love KMITL 07';...
                 'I love KMITL 08';...
                 'I love KMITL 09';...
                 'I love KMITL 10';...
                 'I love KMITL 11';...
                 'I love KMITL 12';...
                 'I love KMITL 13';...
                 'I love KMITL 14';...
                 'I love KMITL 15';...
```

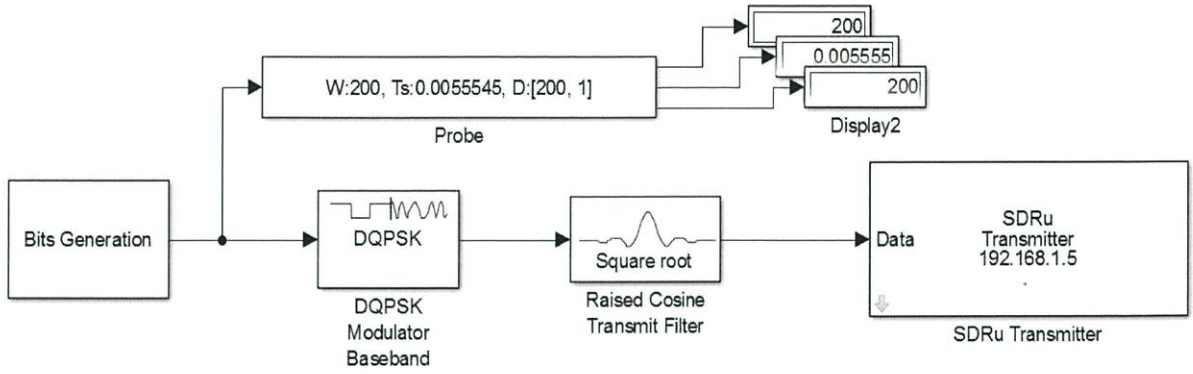
รูปที่ 4.1 code ที่ใช้ในการ Generate Text ข้อมูลในภาคส่ง

การที่ภาครับจะสามารถรับข้อมูลจากฝั่งส่งได้ จะต้องมีการถอดรหัสจากข้อมูลที่รับได้ โครงงานในส่วนนี้มีการใช้การเข้ารหัสข้อมูลโดยใน 1 เฟรม จะมี 200 บิท 26 บิตแรกจะเป็น Header ของข้อมูล และ 174 บิตถัดมาจะเป็นบิตของ Text ข้อมูลที่ส่ง บิตทั้งหมดจะถูกส่งเข้าสู่กระบวนการ Digital modulation ต่อไป และเมื่อทำการรัน Simulink Block ของภาคส่ง ที่ทำการส่ง Text ข้อมูลคำว่า "I Love KMITL" constellation ของข้อมูลหลังจากผ่านบล็อก DQPSK Modulator Baseband แสดงดังรูปที่ 4.2



รูปที่ 4.2 constellation ของข้อมูลหลังจากผ่านบล็อก DQPSK Modulator Baseband

ภายหลังจากที่บิตข้อมูลผ่านการ Modulate จะถูกนำเข้าสู่ Raised Cosine Transmit Filter เพื่อทำการ upsample และจัดรูป pulse ข้อมูล โดยมีสัมประสิทธิ์การ upsample เป็น 8 จะทำให้อัตราเร็วการข้อมูลจาก 18 ksymbol/s เป็น 144 ksymbol/s และสุดท้ายจะผ่านบล็อก SDRu Transmitter ที่ทำหน้าที่ในกระบวนการ DAC และส่งข้อมูลผ่านสายอากาศ มี Carrier Frequency เป็น 450 MHz ตรงตามมาตรฐาน TETRA ต่อไป โดยในภาคส่งจะมีการตรวจสอบหาอัตราเร็วการส่งข้อมูลระหว่างการ simulation โดยใช้บล็อก Probe ในการวัดความกว้างของข้อมูลใน 1 เฟรม ช่วงเวลาแซมปลิ้งของข้อมูลใน 1 เฟรม และไดเมนชั่นของข้อมูลใน 1 เฟรม สามารถแสดงได้ดังรูปที่ 4.3



รูปที่ 4.3 บล็อก Probe ที่นำมาตรวจสอบหาอัตราการส่งข้อมูล

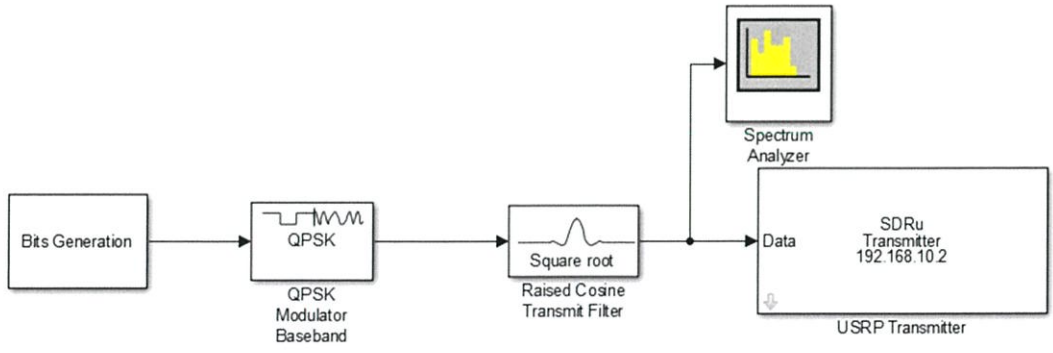
โดยที่วิธีการคำนวณหาอัตราการส่งข้อมูล (bit rate) จากบล็อก Probe สามารถแสดงได้ดังสมการ

$$\frac{\mathrm{W(Width\ of\ Data)}}{\mathrm{T_s(sample\ time)}} = \mathrm{bit\ rate}$$

จากสมการทำให้สามารถคำนวณอัตราการส่งข้อมูล (bit rate) ได้ดังนี้

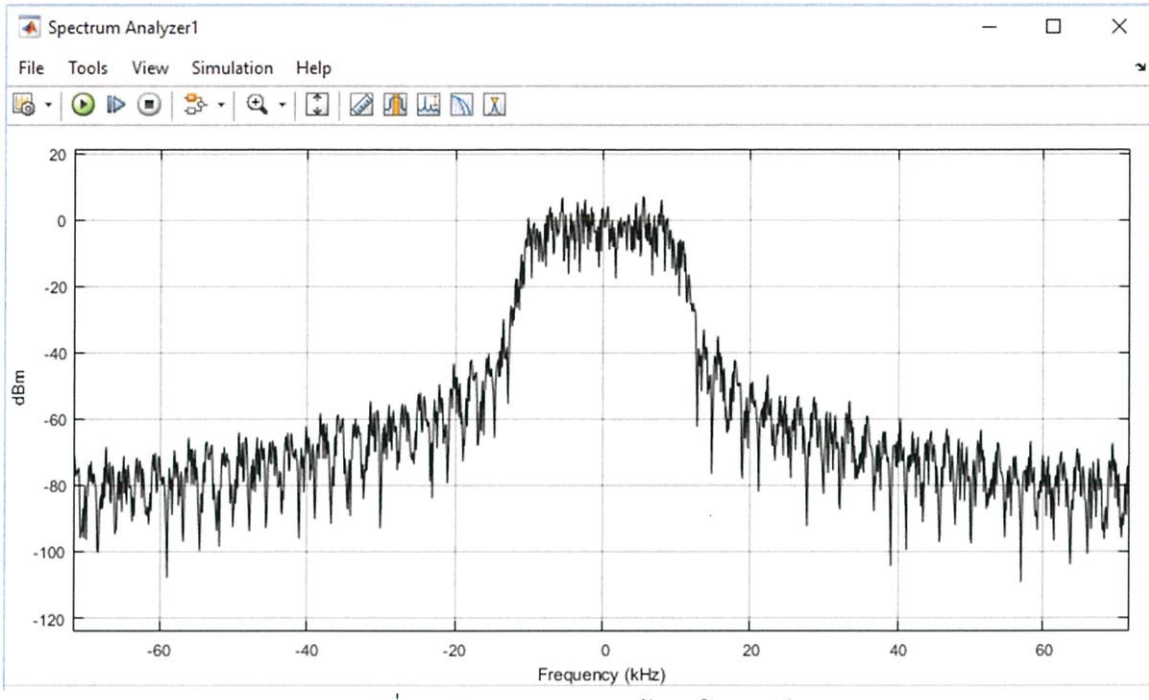
$$\frac{200}{0.0055545} \approx 36000\ bit/s$$

ในภาคส่ง สามารถแสดงผลสัญญาณในโดเมนของความถี่ได้โดยใช้ Spectrum Analyzer วัดสัญญาณข้อมูลภายหลังจากผ่าน Raised Cosine Transmit Filter ซึ่ง Simulink block แสดงได้ดังรูปที่ 4.4



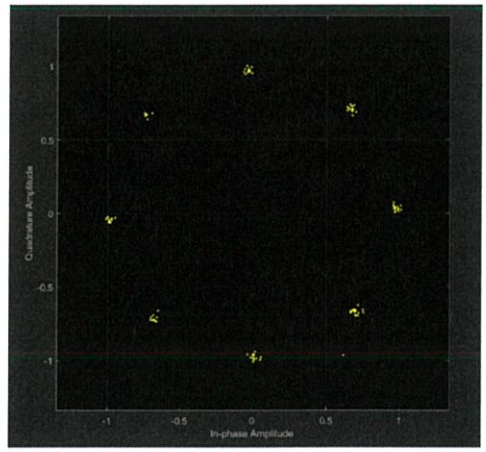
รูปที่ 4.4 Simulink Block แสดงจุดวัดสัญญาณในภาคส่งโดยใช้ Spectrum Analyzer

สัญญาณที่วัดได้จาก Spectrum Analyzer สามารถแสดงได้ดังรูปที่ 4.5



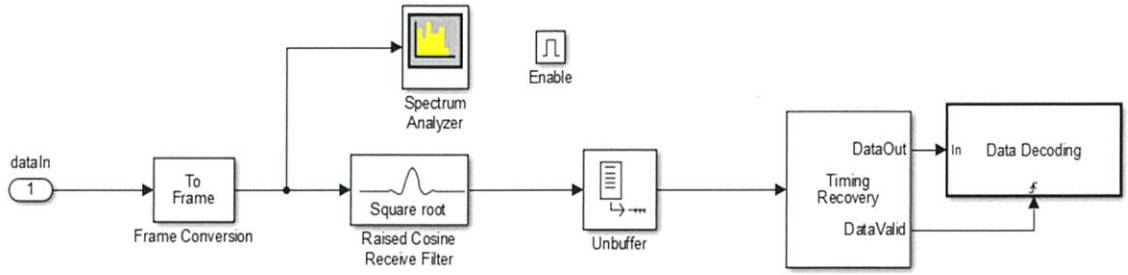
รูปที่ 4.5 Spectrum ของข้อมูลในภาคส่ง

ในส่วนของภาครับ สัญญาณจะถูกรับจากบล็อก SDRu Receiver รวมไปถึงกระบวนการ ADC จะถูกทำในส่วนนี้ด้วย โดยสัญญาณที่รับมาได้จะถูกทำให้เป็นเฟรมข้อมูลและผ่านกระบวนการ Down Sampling จาก Raised Cosine Receive Filter หลังจากนั้นจะสัญญาณจะถูกนำเข้าสู่กระบวนการในการหาจุดเริ่มต้นของบิตข้อมูลที่รับได้ และนำไปสู่กระบวนการในการถอดรหัส Header ของข้อมูล เพื่อแสดงผล Text ข้อมูลสุดท้ายต่อไป ในส่วนของภาคส่งโดย constellation ของข้อมูลก่อนผ่านบล็อก DQPSK Demodulator Baseband ในภาครับแสดงดังรูปที่ 4.6



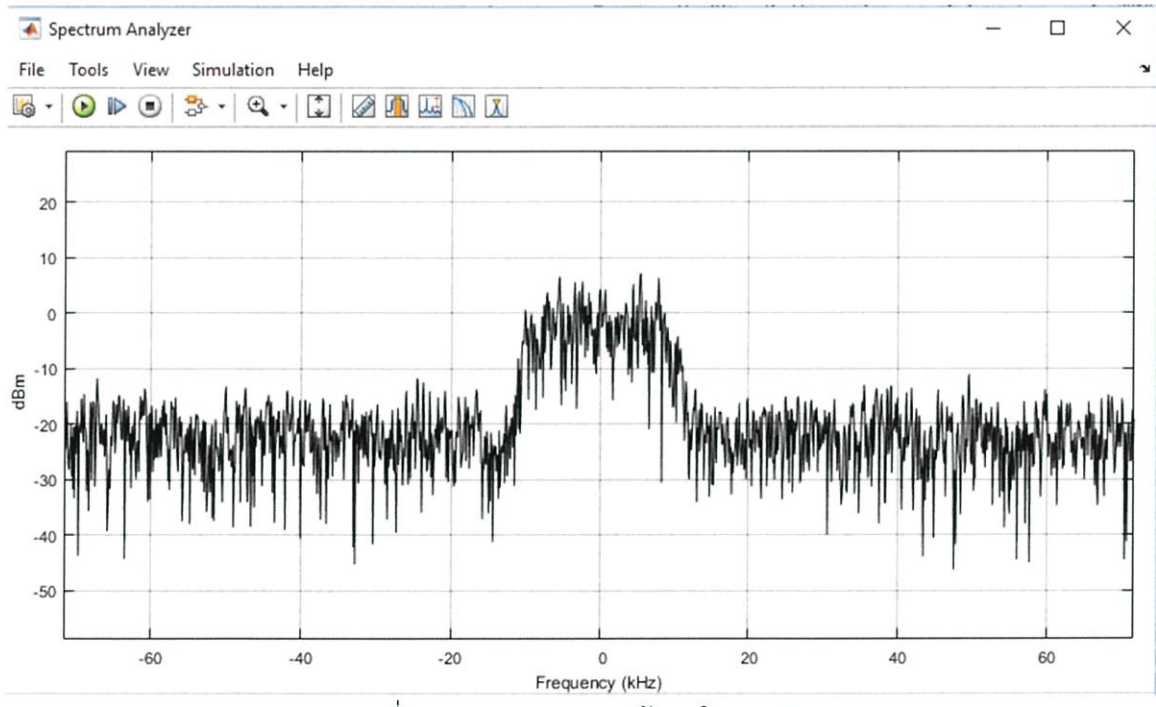
รูปที่ 4.6 constellation ของข้อมูลก่อนผ่านบล็อก DQPSK Demodulator Baseband

ในภาครับ สามารถแสดงผลสัญญาณในโดเมนของความถี่ได้โดยใช้ Spectrum Analyzer วัดสัญญาณข้อมูลภายหลังจากผ่าน Raised Cosine Receive Filter ซึ่ง Simulink block แสดงได้ดังรูปที่ 4.7



รูปที่ 4.7 Simulink Block แสดงจุดวัดสัญญาณในภาครับโดยใช้ Spectrum Analyzer

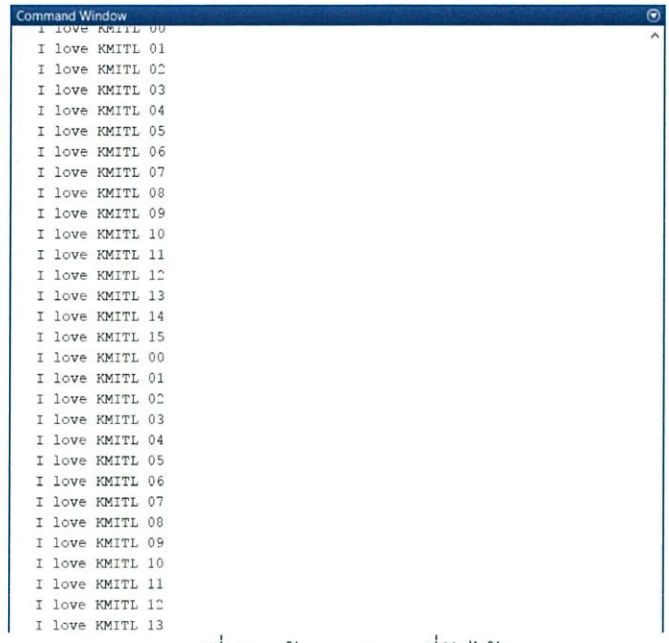
สัญญาณที่วัดได้จาก Spectrum Analyzer สามารถแสดงได้ดังรูปที่ 4.8



รูปที่ 4.8 Spectrum ของข้อมูลในภาครับ

จากรูปที่ 4.5 และ 4.8 แสดง Spectrum ของข้อมูลในฝั่งส่งและฝั่งรับ จะสังเกตเห็นได้ว่าช่วงความถี่ของสเปกตรัมมีความใกล้เคียงกัน ส่งผลให้สามารถยืนยันได้ว่าช่วงความถี่ของข้อมูลที่ส่งและรับ มีค่าใกล้เคียงกัน

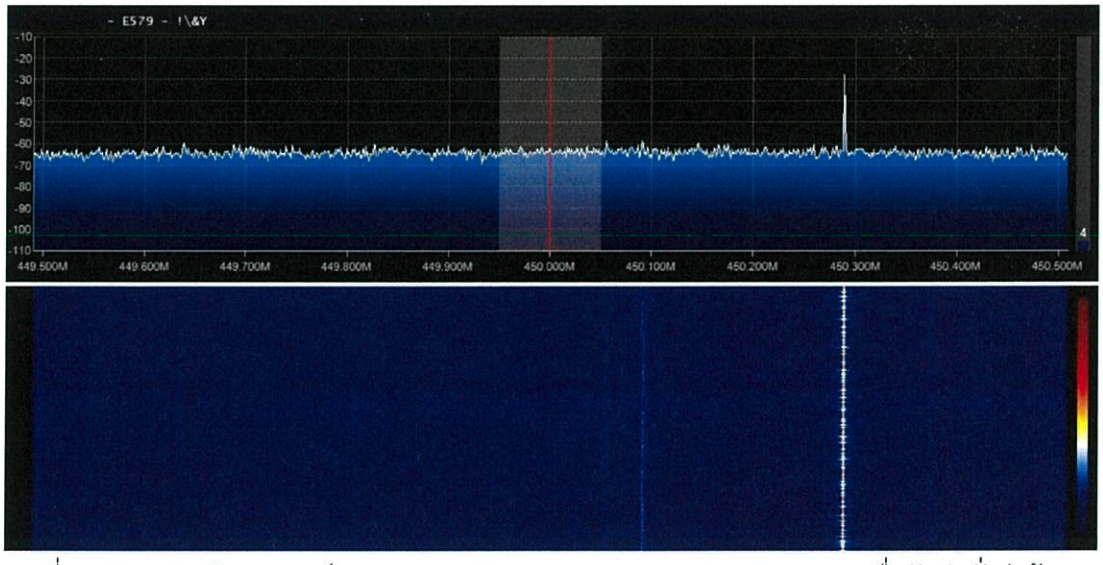
โดยข้อมูล Text ที่ภาครับสามารถถอดรหัสและแสดงผล แสดงดังรูปที่ 4.9



รูปที่ 4.9 ข้อมูล Text ที่รับได้

4.2 ผลการใช้งาน Advance Digital Radio Baseband Board

ผลจากการใช้ SDR Dongle เพื่อดูสเปกตรัมของบอร์ด Advance Digital Radio Baseband Board เมื่อมีการรันคำสั่งส่งข้อมูล โดยใช้ซอฟแวร์ SDRSharp แสดงดังรูปที่ 4.10

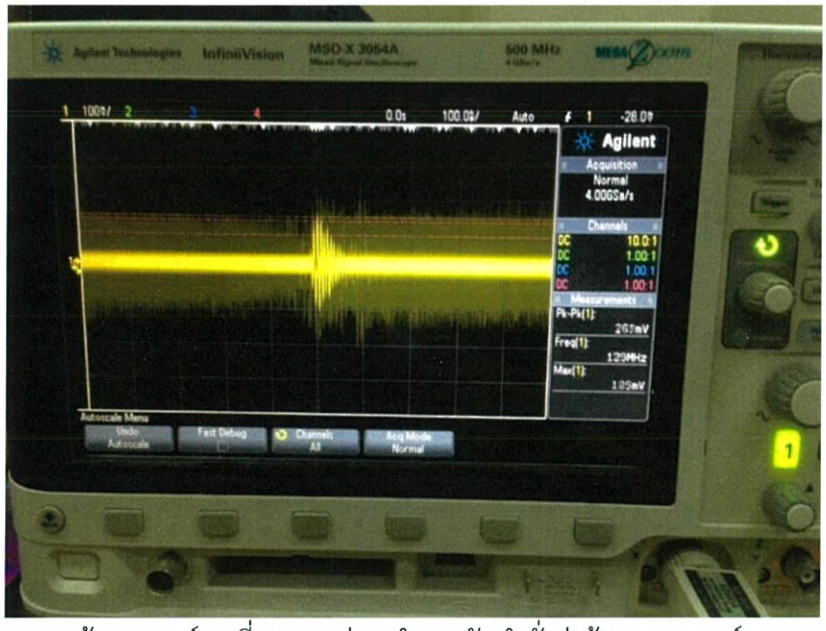


รูปที่ 4.10 สเปกตรัมของบอร์ด Advance Digital Radio Baseband Board เมื่อรันคำสั่งส่งข้อมูล

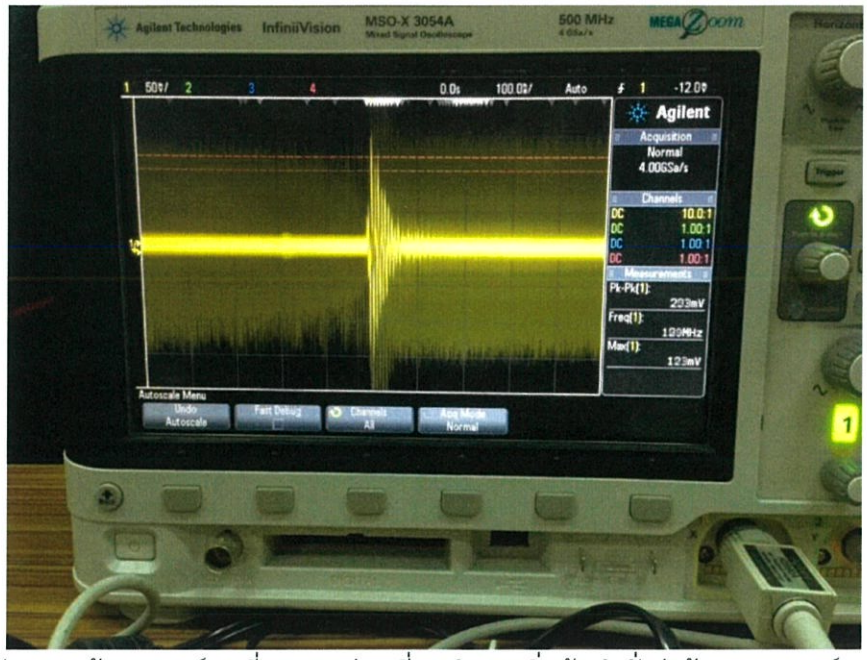
จากการสังเกตสเปกตรัมของบอร์ด Advance Digital Radio Baseband Board เมื่อรันคำสั่งส่งข้อมูล แสดงให้เห็นว่า สเปกตรัมที่วัดได้เป็นเพียงสัญญาณคลื่นพาห์จาก IC Si570 ของบอร์ด Advance Digital Radio Baseband Board ที่ความถี่ 450 MHz เท่านั้น ยังไม่สามารถยืนยันว่ามีการส่งข้อมูลจริงได้

4.3 ผลการเปรียบเทียบสัญญาณเอาต์พุตที่ขา TxI ระหว่าง Advance Digital Radio Baseband Board กับ Evaluation Kit 9810

ผลจากการวัดสัญญาณเปรียบเทียบข้อมูลเอาต์พุตที่ขา TxI ดังแสดงในรูป 3.38 ก่อนทำการรันคำสั่งส่งข้อมูลของบอร์ด Advance Digital Radio Baseband Board และบอร์ด Evaluation Kit 9810 แสดงดังรูปที่ 4.11 และ รูปที่ 4.12 ตามลำดับ

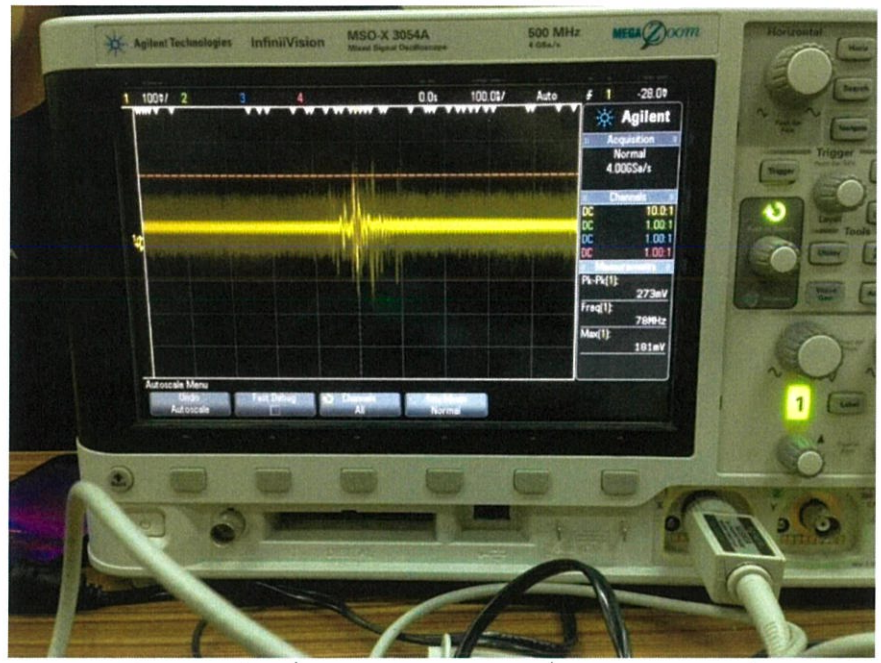


รูปที่ 4.11 สัญญาณข้อมูลเอาต์พุตที่ขา TxI ก่อนทำการรันคำสั่งส่งข้อมูลของบอร์ด Advance Digital Radio Baseband Board

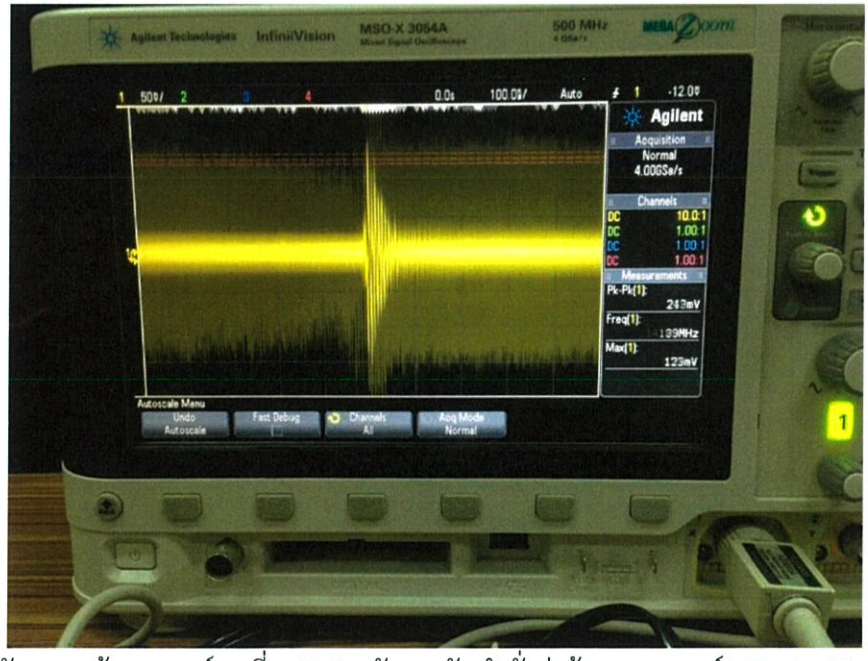


รูปที่ 4.12 สัญญาณข้อมูลเอาต์พุตที่ขา TxI ก่อนที่จะทำการเริ่มต้นคำสั่งส่งข้อมูลของบอร์ด Evaluation Kit 9810

ผลจากการเปรียบเทียบสัญญาณข้อมูลเอาต์พุตที่ขา TxI ก่อนรันคำสั่งส่งข้อมูลของบอร์ด Advance Digital Radio Baseband Board และบอร์ด Evaluation Kit 9810 พบว่าลักษณะของสัญญาณที่วัดได้มีความคล้ายคลึงกันกันทั้งในแง่ของแอมปลิจูดและความถี่ โดยผลจากการวัดสัญญาณข้อมูลเอาต์พุตที่ขา TxI หลังจากรันคำสั่งส่งข้อมูลของบอร์ด Advance Digital Radio Baseband Board และบอร์ด Evaluation Kit 9810 แสดงดังรูปที่ 4.13 และ รูปที่ 4.14 ตามลำดับ



รูปที่ 4.13 สัญญาณข้อมูลเอาต์พุตที่ขา TxI หลังจากรันคำสั่งส่งข้อมูลของบอร์ด Advance Digital Radio Baseband Board



รูปที่ 4.14 สัญญาณข้อมูลเอาต์พุตที่ขา TxI หลังจากรันคำสั่งส่งข้อมูลของบอร์ด Evaluation Kit 9810

ผลจากการเปรียบเทียบรูปแบบของสัญญาณข้อมูลเอาต์พุตที่ขา TxI หลังจากรันคำสั่งส่งข้อมูลของบอร์ด Advance Digital Radio Baseband Board และบอร์ด Evaluation Kit 9810 พบว่าลักษณะของสัญญาณแตกต่างกัน โดยรูปแบบสัญญาณของบอร์ด Evaluation Kit 9810 มีการขยายแอมปลิจูดแต่ไม่มีสัญญาณไฟตรงเกิดขึ้น แต่รูปแบบสัญญาณของบอร์ด Advance Digital Radio Baseband Board ไม่มีการขยายทางแอมปลิจูดและมีสัญญาณไฟตรงเกิดขึ้น ประกอบกับข้อผิดพลาดที่เกิดขึ้นจากการใช้งานตัวบอร์ดก่อนหน้านี้ ทำให้สามารถสรุปได้ว่าตัวบอร์ดของทางสถานประกอบการยังไม่สามารถใช้งานได้จริง

บทที่ 5
สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

ในงานวิจัยนี้เป็นการศึกษาและพัฒนาอุปกรณ์รับส่งข้อมูลบนมาตรฐาน TETRA โดยในช่วงแรกได้ศึกษาและพัฒนาบนอุปกรณ์ USRP (Universal Software Radio Peripheral) เพราะเป็นอุปกรณ์ในระบบ SDR (software-Defined Radio) ซึ่งเป็นอุปกรณ์ที่มีความยืดหยุ่นและง่ายต่อการใช้งานและสามารถนำไปประยุกต์ใช้งานบนมาตรฐาน TETRA ได้ ก่อนที่จะศึกษาและพัฒนาบอร์ด Advance Digital Radio Baseband Board (บอร์ด TETRA RTU ของ บริษัท กสท. โทรคมนาคม จำกัด มหาชน) โดยต้นแบบของบอร์ดนี้คือ Evaluation Kit 9810 (EV9810), Evaluation Kit 9980 (EV9980) และ Evaluation Kit 9942 (EV9942) ซึ่งเป็นบอร์ดที่มี IC CMX981, IC CMX998 และ IC CMX994 อยู่บนบอร์ด โดยไอซีทั้งหมดนี้ได้สร้างขึ้นบนมาตรฐาน TETRA

ผลที่ได้จากการศึกษาและประยุกต์ใช้มาตรฐาน TETRA บนเทคโนโลยี Software Defined Radio โดยใช้ NI USRP-2920 ผ่าน Software MATLAB @Simulink สามารถออกแบบระบบทั้งในภาคส่งและภาครับให้สามารถรับส่งข้อมูลที่เป็น Text คำว่า "I Love KMITL" ได้จริง จากผลสำเร็จที่ได้บน Platform ของ USRP ทางสถานประกอบการ จึงมอบหมายให้พัฒนาอุปกรณ์รับส่งข้อมูลของทางสถานประกอบการ (Advance Digital Radio Baseband Board) ที่ได้ถูกสร้างขึ้นบนมาตรฐาน TETRA เช่นเดียวกัน และถือเป็นอีกหนึ่ง Platform บนเทคโนโลยี SDR หลังจากที่ทางผู้จัดทำโครงงานได้พัฒนาอุปกรณ์รับส่งข้อมูลของบริษัทไปได้ระยะหนึ่ง บอร์ดของทางสถานประกอบการก็ยังไม่สามารถใช้งานได้ ทางผู้จัดทำจึงได้ทำการเปรียบเทียบสัญญาณระหว่างตัวบอร์ด กับตัวบอร์ดต้นแบบที่ถูกนำมาสร้างบอร์ด ณ จุดเดียวกันนั้น ผลลัพท์ที่ได้มีความแตกต่างกัน ประกอบกับข้อผิดพลาดที่พบเจอมาตลอดช่วงระยะเวลาที่พัฒนา จึงสามารถสรุปได้ว่าตัวบอร์ดมีข้อผิดพลาดทาง Scemetic ทางสถานประกอบการรับทราบปัญหาและหาทางแก้ต่อไป

5.2 ข้อเสนอแนะ

1. การสื่อสารระหว่างเครื่องคอมพิวเตอร์กับตัวบอร์ดของทางสถานประกอบการมีความผิดพลาดบ่อยครั้งเนื่องจากปัญหาในเรื่องการเชื่อมต่อ Hardware ผู้พัฒนาจึงควรมีความรอบคอบอยู่เสมอ

2. การใช้งานบอร์ด Advance Digital Radio Baseband Board มีความซับซ้อนในการใช้งานอยู่พอสมควร ผู้ใช้ควรศึกษาการใช้งาน รวมไปถึงทดลองใช้งานให้เกิดความชำนาญก่อนใช้งานจริง

เอกสารอ้างอิง

[1] "Terrestrial Trunked Radio" [ออนไลน์]. เข้าถึงได้จาก https://en.wikipedia.org/wiki/Terrestrial_Trunked_Radio/ (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[2] "CMX981" [ออนไลน์]. เข้าถึงได้จาก http://www.cmlmicro.com/products/CMX981_Advanced_Digital_Radio_Baseband_Processor/ (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[3] "EV981" [ออนไลน์]. เข้าถึงได้จาก http://www.cmlmicro.com/searchresults/?q=evaluation%20981 (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[4] "CMX998" [ออนไลน์]. เข้าถึงได้จาก http://mikrokontroler.pl/2010/11/22/cmx998-scalony-ukladlinearyzacji-adajnikow-radiowych-firmy-cml-microcircuits/ (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[5] "Radio integrated circuit receiver" [ออนไลน์]. เข้าถึงได้จาก http://www.directindustry.com/prod/cmlmicrocircuits/product-34037-82135.html (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[6] "RF Direct Conversion Receiver ICs" [ออนไลน์]. เข้าถึงได้จาก http://www.cmlmicro.com/products/CMX_994_RF_Direct_Conversion_Receiver/ (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[7] "PE0002" [ออนไลน์]. เข้าถึงได้จาก http://www.cmlmicro.com/products/PE0002_Evaluation_Kit_Interface_Card/ (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[8] "เอฟพีจีเอ" [ออนไลน์]. เข้าถึงได้จาก https://th.wikipedia.org/wiki/เอฟพีจีเอ (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[9] "Zynq-7000" [ออนไลน์]. เข้าถึงได้จาก https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[10] "SPI" [ออนไลน์]. เข้าถึงได้จาก http://aimagin.com/blog/spi/?lang=th (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[11] "I2C BUS" [ออนไลน์]. เข้าถึงได้จาก https://www.i2c-bus.org/addressing/ (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[12] "CBUS" [ออนไลน์]. เข้าถึงได้จาก http://www.cmlmicro.com/products/CMX981_Advanced_Digital_Radio_Baseband_Processor/ (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[13] "Si570" [ออนไลน์]. เข้าถึงได้จาก https://www.silabs.com/documents/public/datasheets/si570.pdf (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[14] "Linux" [ออนไลน์]. เข้าถึงได้จาก https://web.ku.ac.th/schoolnet/snet1/software/linux/index.html (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[15] "DQPSK" [ออนไลน์]. เข้าถึงได้จาก https://www.allaboutcircuits.com/technicalarticles/differentialquadrature-phase-shift-keying-dqpsk-modulation/ (วันที่สืบค้นข้อมูล 15 พฤศจิกายน 2560)

[16] Michael Rice. (2009). Digital Communications: A Discrete-time Approach. Pearson/Prentice Hall.

[17] ETSI. (2007). Terrestrial Trunked Radio (TETRA) : Voice plus Data (V+D). EN 300 392-2, 2007.

[18] Stefan Nagel, Dennis Epple, Friedrich K. Jondral. (2008). Implementing the TETRA Physical Layer on Lyrtech's SFF SDR Development Platform. Karlsruhe University, Germany.

ภาคผนวก ก

Code ของ CMX981.c

```
#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

/* ---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
/* 05,06 == 04*/
#define Aux_Control_read 0xF7
#define Frequency_Control_read 0xF8

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 uint16_t delay;

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);
}
static uint8_t reciver(int fd,uint8_t address,uint8_t data){
        int status;
//      memset(tx,0,ARRAY_SIZE(tx));
```

```
//        memset(rx,0,ARRAY_SIZE(rx));
          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,PowerDownCtrl_write,0x00);
        reciver(fd,ClkStopCtrl_write,0x00);
        reciver(fd,TxSetup_write,0x18);          // *** Original ***
        //reciver(fd,TxSetup_write,0x38);
        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,0x03);
        reciver(fd,TxIGain1_write,0x00);
        reciver(fd,TxIGain2_write,0x03);
        reciver(fd,TxQOffset1_write,0x00);
        reciver(fd,TxQOffset2_write,0x00);
        reciver(fd,TxIOffset1_write,0x00);
        reciver(fd,TxIOffset2_write,0x00);
        //reciver(fd,RampCtrl_write,0x00);
        //reciver(fd,LoopBackCtrl_write,0x01);
        //reciver(fd,LoopBackCtrl_write,0xBE); old
        reciver(fd,LoopBackCtrl_write,0x04);
        /*
        reciver(fd,TxSetup_write,0x18);
        reciver(fd,RxSetup1_write,0x01);
        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,0x03);
        reciver(fd,RxIGain1_write,0x00);
        reciver(fd,RxIGain2_write,0x03);
        reciver(fd,TxSetup_write,0x18);
        reciver(fd,RxSetup2_write,0x00);
        reciver(fd,RxSetup1_write ,0x18);
        temp = reciver(fd,RxSetup1_read,0x00);
        printf("temp : %04x\n",temp);*/
}

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,Aux_Control_write,0x00);
        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");
        int status;
        int fd = (intptr_t) param;
        //fd = open(device, O_RDWR);
        for(int j=0;j<1000;j++){
                for(int i=0; i<8 ; i++){
                uint8_t data[8]={0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7};
        //      memset(tx,0,ARRAY_SIZE(tx));
        //      memset(rx,0,ARRAY_SIZE(rx));
                tx[0]=0x87;
                tx[1]=data[i]; //          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[%d][%d]: %02x %02x \n",j,i, tx[0], tx[1]);
                printf("rev: %02x %02x \n", rx[0], rx[1]);
                }
```

```
        }
}
void* txloop2(void *param){
        printf("\n------- START TXLOOP2-------\n");
        int status;
        int fd = (intptr_t) param;
        uint16_t TxI = 0x0000;
        uint16_t TxQ = 0x0000;
//------------------------------------ I --------------------------------
        //fd = open(device, O_RDWR);
        for(int j=0;j<10000;j++){
        //      for(int i=0; i<8 ; i++){
                //uint16_t data[8]={0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7};
        //      memset(tx,0,ARRAY_SIZE(tx));
        //      memset(rx,0,ARRAY_SIZE(rx));
                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");
                }
```

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

  /*        long pause;
  clock_t now,then;
  pause = 1000*(CLOCKS_PER_SEC/1000);
  now = then = clock();
  while( (now-then) < pause )
  now = clock();
*/
            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");
            }
            printf("sent 0xBC [%d]: %02x %02x \n",j, tx[0], tx[1]);
            printf("rev: %02x %02x \n", rx[0], rx[1]);
  //}
            TxI += 0x0f;
            if(TxI == 0x3FFF)
```

```
                TxI = 0x00;
//------------------------------------ Q ----------------------------------//
        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);

        if(status < 0){
                pabort("SPI_IOC_MESSAGE");
        }
        //printf("sent[%d][%d]: %02x %02x \n",j,i, tx[0], tx[1]);
        //printf("rev: %02x %02x \n", rx[0], rx[1]);

/*      long pause;
clock_t now,then;
pause = 1000*(CLOCKS_PER_SEC/1000);
now = then = clock();
while( (now-then) < pause )
now = clock();
*/
        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");
		}
		printf("sent 0xBE [%d]: %02x %02x \n",j, tx[0], tx[1]);
		printf("rev: %02x %02x \n", rx[0], rx[1]);
	}
		TxQ += 0x0f;
		if(TxQ == 0x3FFF)
			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 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 += 0x0f;
        if(TxI == 0x3FFF)
                TxI = 0x0000;
/*  Write Q **/
        Address = 0xBE;
        Data    = (uint8_t)( TxQ & 0xFF);
        printf("Write I On 3C");
        reciver(fd,Address,Data);

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

```
        TxQ += 0x0f;
        if(TxQ == 0x3FFF)
                TxQ = 0x0000;

        TxReceive = reciver(fd,0x0c,0x00);
        TxStatusIRQ = (TxReceive & 0x01);
        TxStatusQOverflow = (TxReceive & 0x20);
        TxStatusIOverflow = (TxReceive & 0x10);

        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};
                //uint8_t data[5]={0x00,0x00,0x00,0x00,0x00};
        //      memset(tx,0,ARRAY_SIZE(tx));
        //      memset(rx,0,ARRAY_SIZE(rx));
                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]);
```

```
        /*      long pause;
        clock_t now,then;
        pause = 100*(CLOCKS_PER_SEC/1000);
        now = then = clock();
        while( (now-then) < pause )
        now = clock();*/
        }
}
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;
}

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 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) {
                    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{
                    exit(0);
            }
    }
        //delay(10000);
        return ret;
}
```

ภาคผนวก ข
สคริปต์ที่ใช้ใน EV9810

```
;***************************
; 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
        copy    $08, *TxSetup_write
        copy    $06, *LoopBackCtrl_write
        copy    $01, *RxSetup2_write
        copy    $00, *TxIOffset1_write
        copy    $00, *TxIOffset2_write
        copy    $00, *RxIOffset1_write
        copy    $00, *RxIOffset2_write
        copy    $FF, *TxIGain1_write
        copy    $FF, *TxIGain2_write
        copy    $FF, *RxIGain1_write
        copy    $FF, *RxIGain2_write
        copy    $06, *ConfigCtrl1_write
        jsr     receive_loop
        lsl     8, dataB
```

```
        lsl     8, lread2

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_write, Iread1
copy    *TxDPIData2_write, Iread2
;add Iread1, Iread2
disp     "              %x \c"  , Iread2
disp     "%x"  , Iread1
delay 2
jmp     receive_loop
```

ภาคผนวก ค

Code คำสั่งส่งข้อมูลของบอร์ด Advance Digital Radio Baseband Board

```
#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>

#define ConfigCtrl1_read        0x00
#define ConfigCtrl1_write       0x80
#define TxDPIData1_read         0x3C
#define TxDPIData1_write        0xBC
#define TxDPIData2_read         0x3D
#define TxDPIData2_write        0xBD
#define RxDPIData1_read         0x38
#define RxDPIData1_write        0xB8
#define LoopBackCtrl_read       0x13
#define LoopBackCtrl_write      0x93
#define RxSetup1_read           0x08
#define RxSetup1_write          0x88
#define TxSetup_read            0x03
#define TxSetup_write           0x83
```

```
struct spi_ioc_transfer msg[1];
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 pabort(const char *s) {
        perror(s);
        abort();
}

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);
}

static uint8_t reciver(int fd,uint8_t address,uint8_t data){
```

```
        int status;
//      memset(tx,0,ARRAY_SIZE(tx));
//      memset(rx,0,ARRAY_SIZE(rx));
        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];
}
```

```
int main(int argc, char *argv[]){
	int fd;
	uint8_t i;
	int ret = 0;
	uint8_t ilsb;
	uint8_t imsb = 0x01;
	int flag = 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);

reciver(fd,ConfigCtrl1_write,0x00);
reciver(fd,RxSetup1_write,0x02);
reciver(fd,TxSetup_write,0x98);
reciver(fd,LoopBackCtrl_write,0x06);
if (flag == 0){
        for (ilsb=0 ; ilsb<256 ; ilsb++){
                reciver(fd,TxDPIData1_write,ilsb);
                if (ilsb == 0xFF){
                        reciver(fd,TxDPIData2_write,imsb);
                        ilsb == 0;
                        imsb += 0x01;
                        if (ilsb == 0xFF && imsb == 0x3F){
                                flag == 1;
                        }
                }
        }
}
else {
        for (ilsb=255 ; ilsb>0 ; ilsb--){
                reciver(fd,TxDPIData1_write,ilsb);
```

```
                    if (ilsb == 0x00){
                        reciver(fd,TxDPIData2_write,imsb);
                        ilsb == 255;
                        imsb -= 0x01;
                        if (ilsb == 0x00 && imsb == 0x00){
                            flag == 0;
                        }
                    }
                }
            }
            return ret;
}
```