Design Document

Morse Code

Transmitter and Translator

เครื่องสื่อสารรหัสมอร์สพร้อมแปลความหมาย

จัดทำโดย

64010956 นายอติกันต์ พยุงทอง

64011011 นายอัฟนันน์ ลีวัน

เสนอ

รศ.ดร.เจริญ วงษ์ชุ่มเย็น

รายงานชิ้นนี้เป็นส่วนหนึ่งของวิชา Digital System Fundamentals รหัสวิชา 01076112

ภาคเรียนที่ 1 ปีการศึกษา 2565

คำนำ

การสื่อสารเป็นอาวุธที่สำคัญที่สุดในสงคราม รหัสมอร์สจึงถูกนำมาใช้ในการทหาร เพราะเป็นข้อความที่ประกอบด้วยเสียงสั้น-ยาวเท่านั้น ทำให้การส่งนั้น ง่ายและรวดเร็ว แต่ทว่าการแปลความหมายปลายทางนั้น ต้องอาศัยความชำนาญของผู้ฟังสูง หากผิดพลาดแม้เพียงเล็กน้อยอาจทำให้ข้อความที่ส่งมาสื่อความหมายผิดได้

จากปัญหาดังกล่าว คณะผู้จัดทำจึงเลือกพัฒนาเครื่องสื่อสารรหัสมอร์สพร้อมแปลความหมาย เพื่อเพิ่มความสะดวกและความแม่นยำในการแปลงสัญญาณรหัสมอร์สให้มากที่สุด

คณะผู้จัดทำ

15 ธันวาตม 2565

สารบัญ

บทนำ

ที่มาและความสำคัญ

รหัสมอร์ส (Morse Code) เป็นการส่งข้อความรูปแบบหนึ่ง ด้วยสัญญาณสั้นยาวโดยใช้สัญลักษณ์จุดและขีด แทนตัวอักษรต่างๆที่กำหนดไว้เป็นสากล เป็นการสื่อสารที่เก่า แต่ในปัจจุบันก็ยังมีการใช้อยู่ ข้อดีของรหัสมอร์สใช้แถบความถี่น้อยมากเมื่อเทียบกับการสื่อสารระบบอื่น ๆ เป็นการประหยัดความถี่ สามารถใช้งานพร้อม ๆ กันได้โดยไม่มีการรบกวนกัน ในปัจจุบันการติดต่อบางรูปแบบยังจำเป็นต้องใช้รหัสมอร์ส เช่นการติดต่อ สะท้อนออโรรา และเนื่องจากการสื่อสารด้วยรหัสมอร์สจำเป็นต้องอาศัยความแม่นยำในการกดสัญญาณและแปลสัญญาณในแต่ละครั้ง ผู้ส่งและผู้รับจึงจำเป็นต้องมีความชำนาญในการแปลตัวอักษรเป็นสัญญาณรหัสมอร์ส ดังนั้นผู้จัดทำจึงประยุกต์ใช้ความรู้ที่มีสู่การพัฒนาเครื่องสื่อสารรหัสมอร์สพร้อมแปลความหมาย เพื่อเพิ่มความสะดวกและเพื่อฝึกฝนการแปลสัญญาณรหัสมอสมอร์ส ด้วย FPGA 2 บอร์ด โดยทั้ง 2 บอร์ดจะสามารถรับ - ส่งสัญญาณรหัสมอร์ส ด้วยการกดปุ่ม Input บนบอร์ด FPGA บอร์ดหนึ่ง เป็นสัญญาณรหัสมอร์สที่ต้องการจะส่ง ในขณะนั้นลำโพงของบอร์ด FPGA อีกตัวหนึ่งจะรับสัญญาณและส่งเสียงผ่านลำโพงเป็นสัญญานตามที่ Input เข้ามาซึ่งในระหว่างส่งสัญญาณจะแสดงตัวอักษรที่งไปบน led 7-segment บนบอร์ดที่ทำการส่ง และแสดงตัวอักษรบน LCD บนบอร์ดที่ทำการรับสัญญาณ ซึ่งจะทำให้ทั้งผู่ส่งและผู้รับสัญญาณรู้ว่าสัญญาณที่ส่งหรือรับคือตัวอักษรใด

จุดประสงค์

1. เพื่อศึกษาพื้นฐานการสื่อสารของอุปกรณ์ดิจิตอล
2. เพื่อเพิ่มความสะดวกและความแม่นยำในการแปลงสัญญาณรหัสมอร์ส
3. เพื่อประยุกต์ใช้ความรู้ที่ได้เรียนมา ในการสร้างอุปกรณ์ที่ใช้งานได้จริง จาก FPGA
4. เพื่อศึกษาความรู้เพิ่มเติมนอกเหนือจากที่ได้เรียน โดยผ่านการทำโครงงาน

ประโยชน์ที่คาดว่าจะได้รับ

1. สามารถใช้เป็นเครื่องมือสื่อสารได้
2. สามารถแปลความหมายของรหัสมอร์สได้

ขอบเขตของโครงงาน

1. สามารถส่งสัญญาณในรูปแบบของรหัสมอร์สระหว่างบอร์ด FPGA ได้
2. สามารถแปลงสัญญาณรหัสมอร์สเป็นตัวอักษร และแสดงผลบนจอ LCD ได้
3. สามารถแปลงสัญญาณรหัสมอร์สเป็นตัวอักษร และแสดงผลบนจอ LED 7-Segment ได้

กระบวนการหาข้อมูล

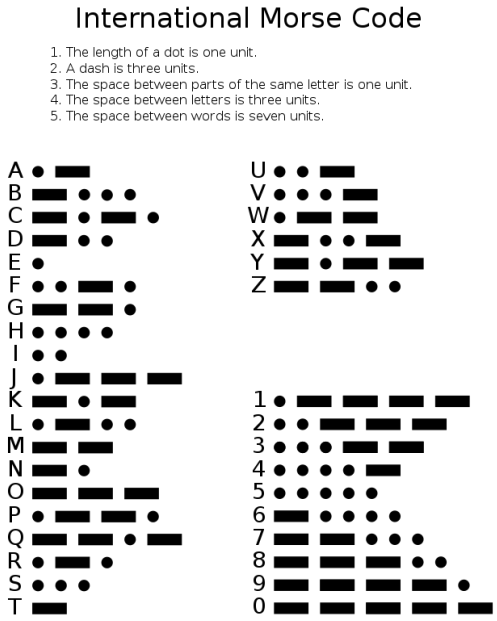
ข้อมูลและทฤษฎีที่เกี่ยวข้องที่ผู้จัดทำได้ศึกษาเพิ่มเติมจากแหล่งข้อมูลที่มีความน่าเชื่อถือเพื่อการทำโครงงานนี้ประกอบไปด้วย

การส่งสัญญาณรหัสมอร์ส

รหัสมอร์ส (Morse Code) คือ วิธีการส่งสัญญาณเป็นชุดด้วยการเคาะเป็นเสียง หรือส่งสัญญาณไฟ ในลักษณะเป็นขีด หรือเป็นจุด ได้รับการพัฒนาเป็นภาษาที่ใช้กันทั่วโลกคิดค้นโดย ซามูเอล เอฟ. บี. มอร์ส (Samuel F. B. Morse) และผู้ช่วยของเขา อัลเฟรต เวล (Alfred Vail) ในปี ค.ศ. 1830 และพัฒนาอย่างต่อเนื่องจนถึงปี ค.ศ. 1840 ข้อความแรกถูกส่งทางโทรเลขในวันที่ 24 พฤษภาคม ค.ศ. 1844 ระหว่างเมืองวอชิงตันไปยังบอสทิมอร์ รหัสมอร์สเมื่อแปลออกมาแล้วจะเป็นตัวอักษรภาษาอังกฤษ ตัวเลข และเครื่องหมายวรรคตอนต่างๆ เป็นภาษาที่ใช้กันทั่วโลก ในกิจการโทรเลข วิทยุ เพื่อส่งสัญญาณสื่อสารกันในพื้นที่ห่างไกล รวมถึงใช้เป็นข้อความขอความช่วยเหลือได้อีกด้วย วิธีอ่านรหัสมอร์ส

. จุด อ่านออกเสียงว่า ดิท “Dits” หรือ ดอท “Dot”

_ ขีด อ่านออกเสียงว่า ดะห์ “Dahs” หรือ แดช “Dash” นำมาประกอบกันเป็นชุด ดังนี้

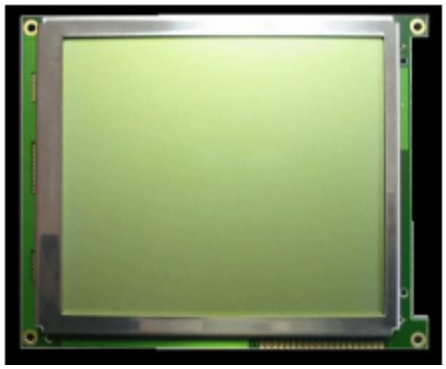


การใช้FPGA ร่วมกับ LCD

โมดูลแอลซีดี (LCD Module) เป็นอุปกรณ์แผงแสดงผลที่สามารถแสดงได้ทางข้อความตัวเลขและรูปภาพ เหมาะสำหรับงานแสดงผลที่ต้องการให้สิ้นเปลืองพลังงานต่ำโดยทั่วๆไปสามารถแบ่งได้เป็น 3 ประเภท

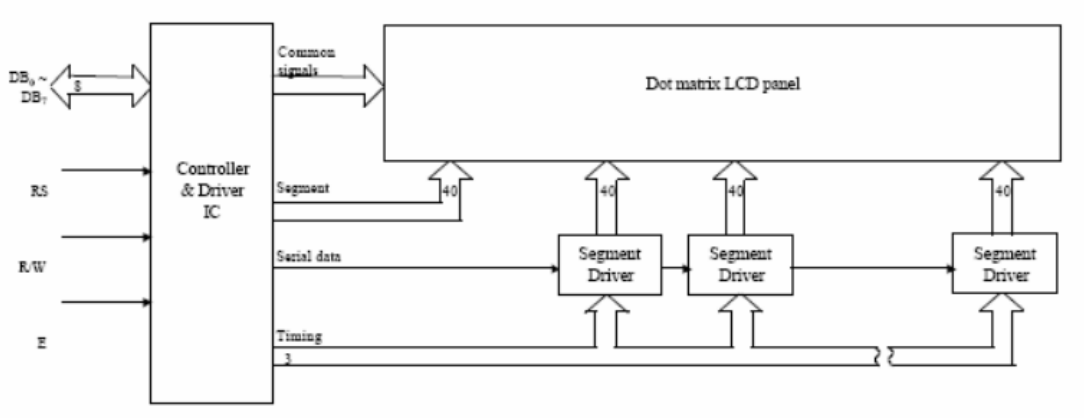
character LCD module แบบนี้สามารถแสดงผลได้ทั้งตัวอักษรและตัวเลขโดยจัดแบ่งเป็นแถวๆเช่นแบบ 1 แถว แถวละ 16 ตัวอักษรหรือแบบ 2 แถวแถวละ 16 ตัวอักษรดังที่ใช้ในโครงงานนี้เป็นต้น

Graphic LCD module แบบนี้สามารถแสดงผลเป็นรูปภาพได้และสามารถทำเป็นรูปภาพของอักษรแบบต่างๆได้



Segment Display LCD module เป็นแบบรูปที่เป็นการแสดงผลคล้าย 7 segment แบบนี้จะมีรูปแบบการแสดงที่จำกัดและไม่กี่แบบ

ส่วนประกอบของโมดูล LCD



ส่วนประกอบของโมดูล LCDประกอบด้วยกัน 3 ส่วนคือ

Dot Matrix LCD เป็นตัวแสดงผล ทำงานในลักษณะของการปิดหรือเปิด ตัวเองกับแสง

Driver เป็นตัวขับ LCD รับสัญญาณมาจากส่วนควบคุม เบอร์ที่นิยมใช้ได้แก่ HID44100H และ MSM5259

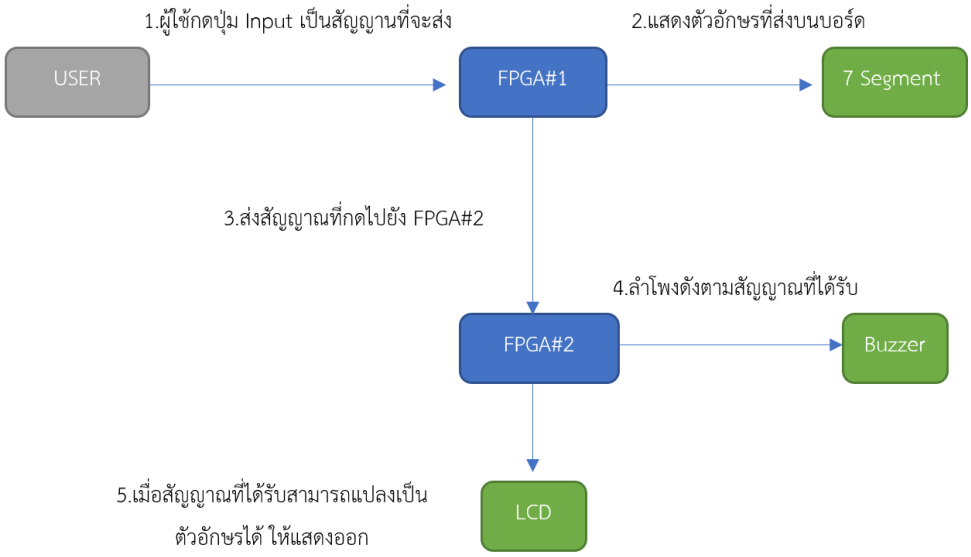
Controller เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอก แล้วควบคุมการทำงานของ LCD เบอร์ที่นิยมใช้ สำหรับแบบ Character ได้แก่ HD4478 ส่วนแบบ Graphic ได้แก่ HD61830

ขาของโมดูล LCD

ขาที่	สัญญาณ	I/O	ต่อกับ	รายละเอียด
1	VSS		แหล่งจ่าย	GND
2	VCC		แหล่งจ่าย	5V
3	VEE		แหล่งจ่าย	ใช้ปรับความสว่างของ LCD โดยต่อกับ VR ถ้าต่อ GND จะสว่างที่สุด
4	RS	Input	CPU	ใช้เลือก Register ควบคุมหรือหน่วยความจำแสดงผล
5	R/W	Input	CPU	ใช้ความคุมการอ่านเขียนโมดูล 0 เขียนข้อมูล 1 อ่านข้อมูล
6	E	Input	CPU	สัญญาณเริ่มต้นการทำงาน สำหรับการอ่านเขียนข้อมูล การรับส่งข้อมูลจะเกิดขึ้นเมื่อเป็น 1 และขอบขาลง
7 - 14	D0-D7	I/O	CPU	เป็นบัสแบบ 2 ทิศทางใช้สำหรับส่งถ่ายข้อมูลระหว่าง CPU กับโมดูล
15	A		แหล่งจ่าย	ต่อกับ 5V เพื่อเปิด Background
16	K		แหล่งจ่าย	ต่อกับ GND

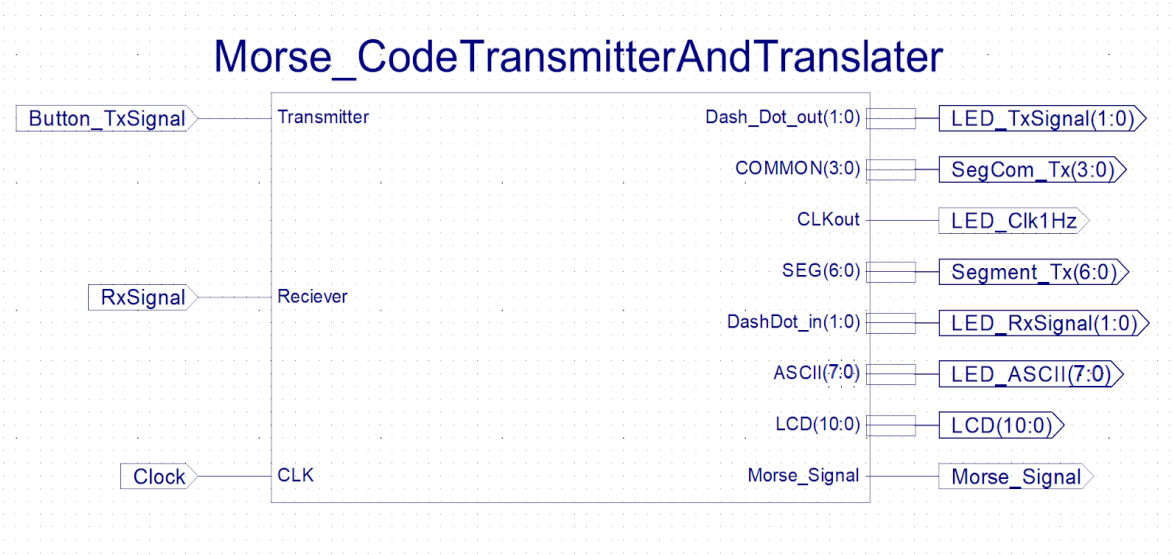
กระบวนการออกแบบ

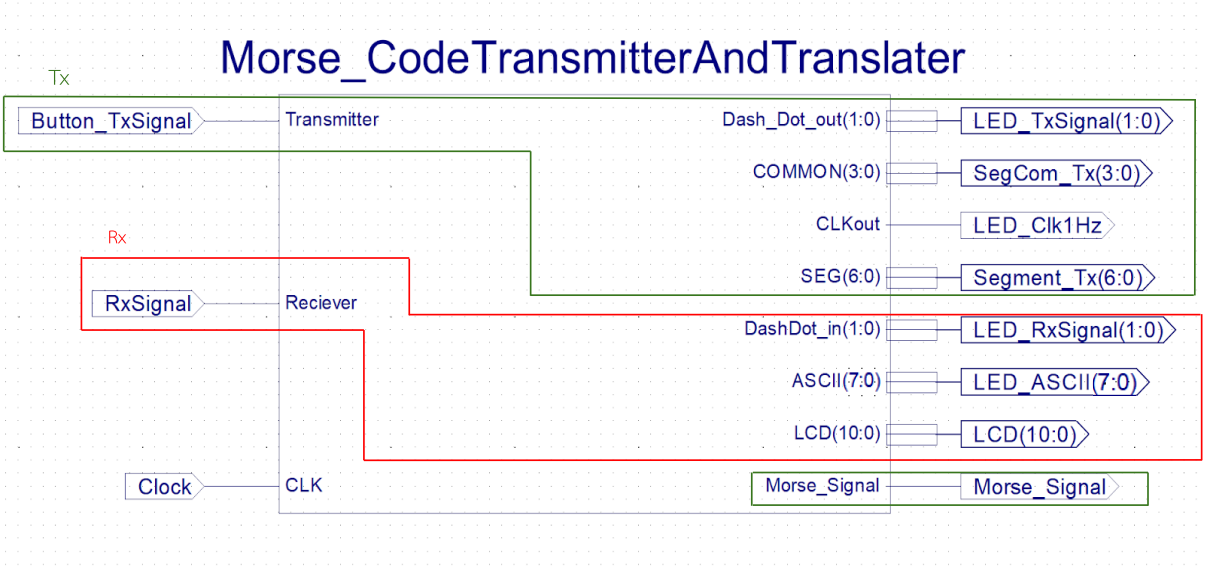
เริ่มจากออกแบบ Flowchart การทำงานของผู้ใช้ เมื่อผู้ใช้กดส่งสัญญาณ จะมีอะไรแสดงผลบ้าง



จากนั้นทำการออกแบบ Top Down Design

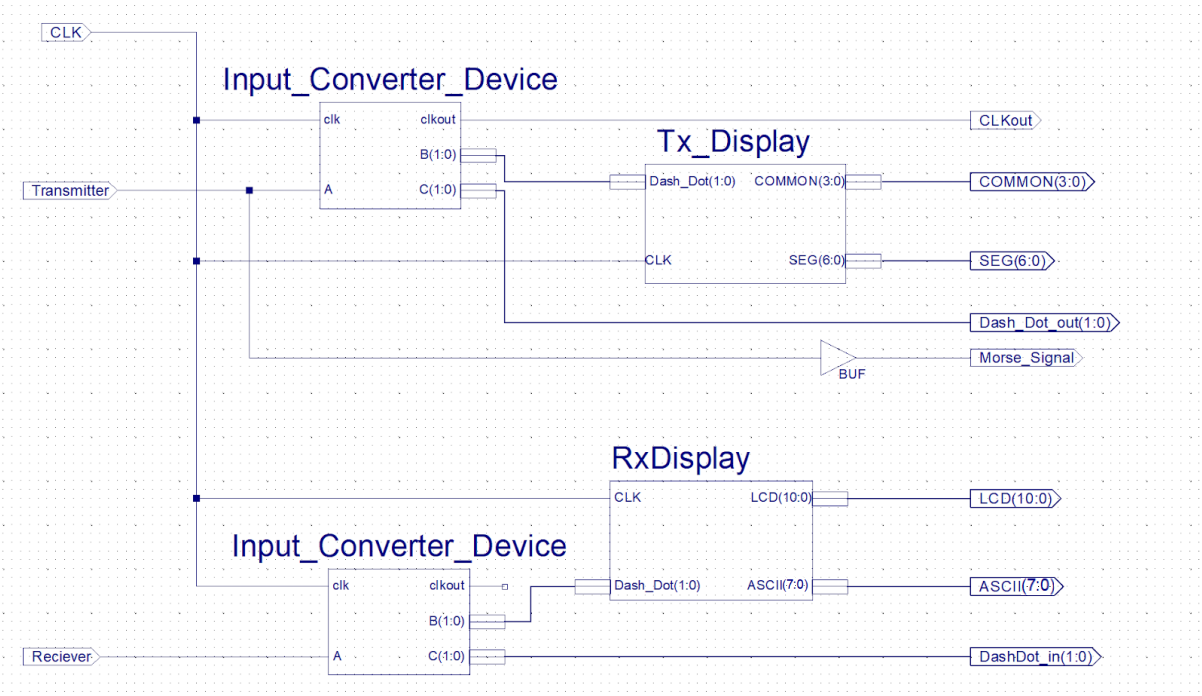
Top layer





เนื่องจาก การทำงานของเราเป็นแบบ Full Duplex (การสื่อสารแบบ 2 ทาง) คือ เราสามารถส่งและรับสัญญานพร้อมกันได้ จึงกำหนด Top layer ให้มีทั้ง Input และ Output ของ เครื่องส่งสัญญาณ (Tx) และเครื่องรับสัญญาณ (Rx) และจะเป็น Design เดียวที่อัพโหลดขึ้นไปบนทั้ง 2 บอร์ด

2nd Layer

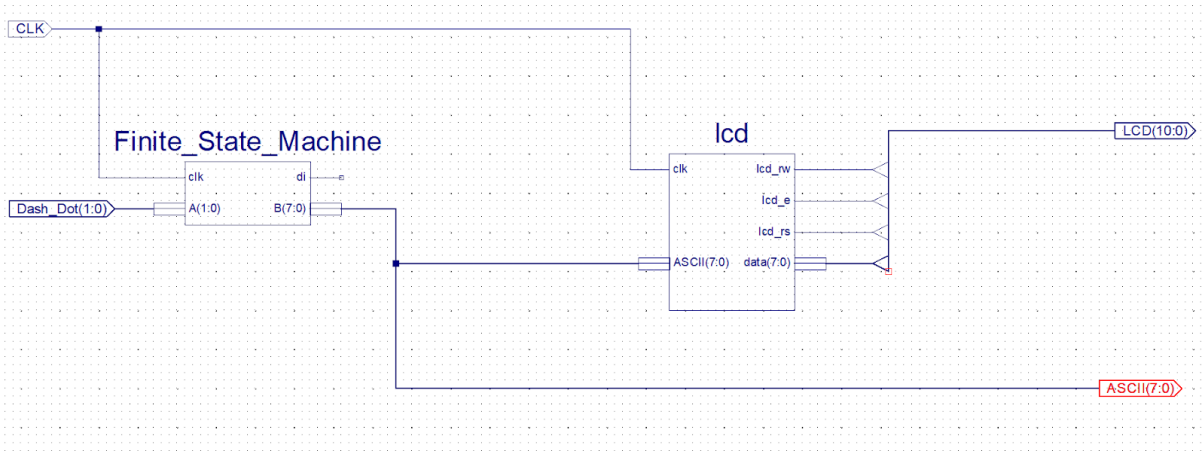


กระบวนการทำงานใน 2nd Layer นั้นจะแบ่งเป็นสองสองส่วนคือส่วนที่ทำหน้าที่รับ Input จากปุ่ม เพื่อส่งไปยังอีกบอร์ดพร้อมแสดงบน Led 7-Segment และส่วนที่รับ Input จากอีกบอร์ดเพื่อแสดงผลบนจอ LCD

ส่วนที่รับ Input จากปุ่มกด (Transmitter) จะส่งไปยังบอร์ดที่รับสัญญาณ ในขณะเดียวกันจะนำสัญญาณไปแปลงเป็นสัญญาณมอร์ส 2 bit ด้วย Input_Converter_Devbice โดยสัญญาณที่ออกมาจะมีสามสถานะ คือ ไม่ได้รับสัญญาณ รับสัญญาณ Dash และ รับสัญญาณ Dot และส่งสัญญาณไปยัง Tx_Display เพื่อแปลงสัญญาณ Dash และ Dot เป็นสัญญาณ 7-Segment เพื่อแสดงผลบน LED 7-Segment

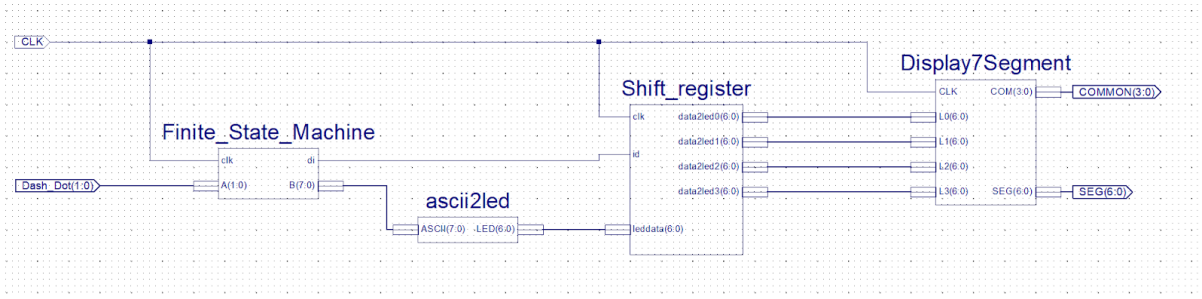
ส่วนที่รับ Input จากอีกบอร์ดจะแปลงสัญญาณเป็นสัญญาณมอร์ส 2 bit ด้วย Input_Converter_Devbice และส่งสัญญาณ Dash และ Dot ไปยัง RxDisplay เพื่อแปลงสัญญาณไปแสดงผลบนจอ LCD พร้อมกับแสดงผลตัวอักษรในรูปแบบของ ASCII ผ่าน LED

3rd Later (RxDisplay)



RxDisplay ทำหน้าที่แปลงสัญญาณและแสดงผลเมื่อเมื่อได้รับ Input จากอีกบอร์ด โดยเมื่อได้รับสัญญาณแล้ว Finite_State_Machine จะแปลงสัญญาณจากสัญญาณ Dot และ Dash เป็นสัญญาณ ASCII ของแต่ละตัวอักษร และแสดงผลผ่าน LED ในขณะเดียวกันจะส่งสัญญาณ ASCII ไปยัง lcd เพื่อใช้แสดงผลบน จอ LCD

3rd Layer (Tx_Display)



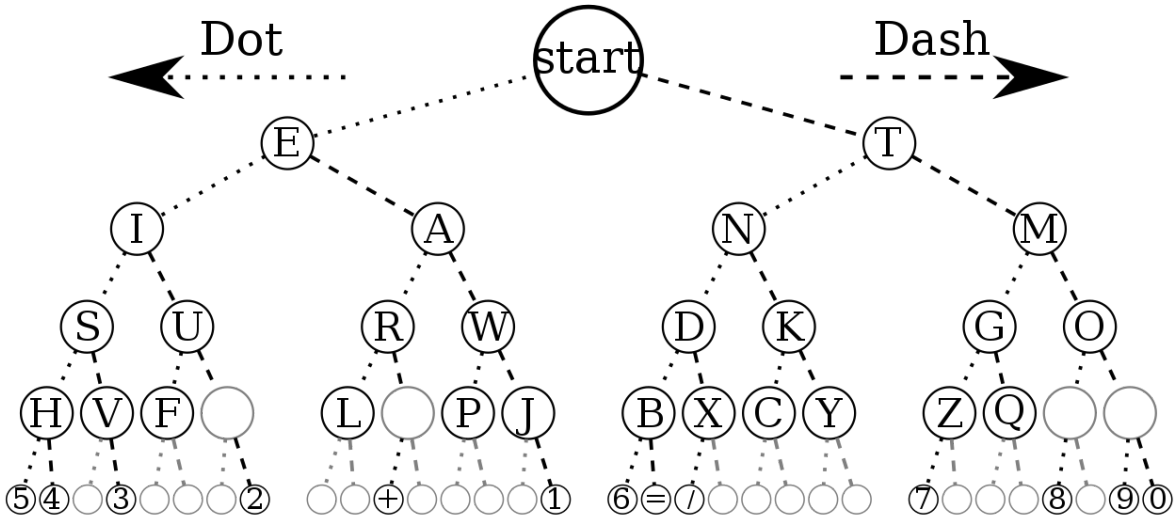
Tx_Display ทำหน้าที่แปลงสัญญาณและแสดงผลเมื่อเมื่อได้รับ Input โดยตรงจากปุ่มโดยเมื่อได้รับสัญญาณแล้ว Finite_State_Machine จะแปลงสัญญาณจากสัญญาณ Dot และ Dash เป็นสัญญาณ ASCII ของแต่ละตัวอักษร หลังจากนั้นจะส่งสัญญาณไปยัง Shift_register เพื่อบันทึกข้อมูลครั้งละ 4 ตัวอักษร และนำแต่ละตัวอักษรไปแสดงผลบน LED 7-Segment โดยควบคุมผ่าน Display 7 Segment

3rd Layer (Input_Converter_Divice)

เป็นการเขียน VHDL สร้างโมดูลที่ทำหน้าที่การแปลงสัญญาณการกดปุ่ม (1 Bit) ให้เป็นสัญญาณ Dot, Dash หรือไม่กด (2 Bit) โดยการนับระยะเวลาที่กด หากกดปุ่มน้อยกว่า 0.5 วินาที จะถือเป็นสัญญาณ Dot (01) หากกดมากกว่า 0.5 วินาทีขึ้นไป จะถือเป็นสัญญาณ Dash (10) และหากไม่กดมากกว่า 0.5 วินาที จะถือเป็น ไม่ได้ส่งสัญญาณ

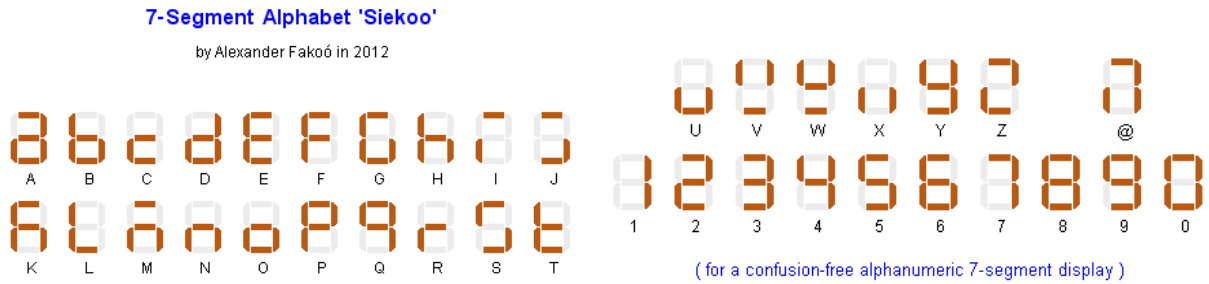
4th layer (Finite_Stage_Machine)

เป็นการเขียน VHDL สร้างโมดูลที่ทำหน้าที่แปลงสัญญาณ Dot สัญญาณ Dash และไม่ได้ส่งสัญญาณ ให้เป็น ASCII โดยใช้หลักการของ Finite Stage ในการดูว่าสัญญาณที่ได้รับตอนนี้เป็นสัญญาณใด และสัญญาณต่อไปเป็นสัญญาณใด โดยที่สัญญาณDot และ Dash จะมีจะมีสัญญาณถัดไปเป็นได้ทั้ง Dot และ Dash (เปรียบได้กับ Binary tree ที่ 1 Node จะมีลูกได้ 2 Node) และเมื่อปล่อยปุ่มกดส่งสัญญาณ จะนำออกสัญญาณที่แปลงได้ต่อไป



4th layer (ascii2led)

เป็นการเขียน VHDL สร้างโมดูลที่ทำหน้าที่แปลง ASCII (8 bit) ให้เป็นแต่ละ Pin ของ LED 7 Segment (7 bit) โดยการกำหนดตัวอักษรแต่ละตัวตามมาตรฐาน The Siekoo Alphabet ที่กำหนดโดย Alexander Fakoó ในปี 2012



4th layer (Shift_register)

เป็นการเขียน VHDL สร้างโมดูลที่ทำหน้าที่เก็บและเลื่อนสัญญาณของ LED 7 Segment (7 bit) ไปเรื่อย และนำออกเป็นบัส 4 ชุด

4th layer (Display7Segment)

เป็นการเขียน VHDL สร้างโมดูลที่ทำหน้าที่เลือกบัส 4 ชุดที่ Shift_register ส่งมา ไปออกแต่ละ Pin Common ของ 7 Segment แต่ละ Pin

กระบวนการพัฒนา

รวบรวมข้อมูล

-กระบวนการส่งและรับสัญญาณรหัสมอร์สและการแปลงสัญญาณจากรหัสมอร์สเป็นตัวอักษร

- การแปลงรหัสมอร์สเป็นตัวอักษรแต่ละตัวอักษร

- คู่มือการใช้งานจอ LCD ร่วมกับบอร์ด FPGA

เตรียมอุปกรณ์

-บอร์ด FPGA จำนวน 2 บอร์ด

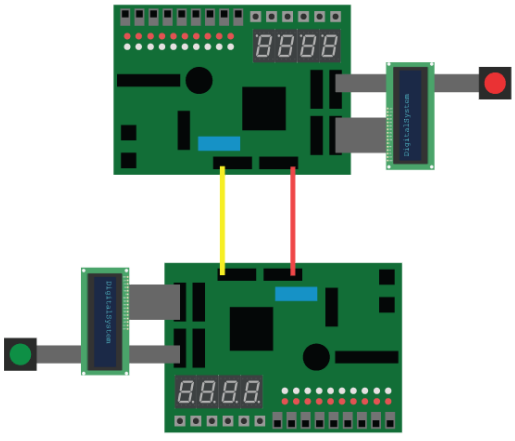
-จอแสดงผล LCD จำนวน 2 จอ

-สายไฟ

- โปรแกรม Xilinx ISE

การออกแบบการทำงาน

บอร์ด FPGA หนึ่งตัวจะสามารถส่งและรับสัญญาณรหัสมอร์สได้ โดยเมื่อมีบอร์ดที่ทำการกดส่งสัญญาณจะทำการแสดงตัวอักษรตามสัญญาณที่กดบน LED 7-Segment และส่งสัญญาณรหัสมอร์สไปยังอีกบอร์ด บอร์ดที่ทำหน้าที่รับสัญญาณจะทำการแปลงสัญญาณที่ได้รับและแสดงผลบนจอ LCD



รูปวงจรการเชื่อมต่ออุปกรณ์

การลงมือพัฒนา

การสร้างวงจรแปลงรหัสมอร์สเป็นรหัส ASCII

รูปผลลัพธ์การแสดงผลในรูปแบบของรหัส ASCII หลังจากการแปลงสัญญาณรหัสมอร์ โดย LED ที่แสดงสถานะสีเขียวคือ 1 และสีแดงคือ 0

การสร้างวงจรแปลง ASCII เป็นสัญญาณแสดงผลบน LED 7-Segment

รูปผลลัพธ์การแสดงผลบน LED 7-Segment หลังจากการแปลงสัญญาณรหัส ASCII จากรูปจะแสดงตัวอักษร S สองตัว

การสร้างวงจรแปลง ASCII เป็นสัญญาณแสดงผลบนจอ LCD

รูปผลลัพธ์การแสดงผลบนจอ LCD หลังจากการแปลงสัญญาณรหัส ASCII จากรูปจะแสดงตัวอักษร DIGITA

การทดสอบผลงานและแก้ไขข้อผิดพลาด

เมื่อดำเนินการเสร็จสักขั้นตอนหนึ่ง ก็ทำการทดสอบเพื่อหาข้อผิดพลาด โดยข้อผิดพลาดที่พบได้แก่

1. ไฟเลี้ยง LCD ไม่เพียงพอ

   เนื่องจาก LCD ที่เราใช้นั้น เป็น LCD ที่ต้องการไฟเลี้ยง 5V แต่บอร์ด FPGA มี Pin ที่จ่ายไฟได้เพียง 3.3V จึงแก้ปัญหาโดยการต่อไฟ 3.3V เข้ากับโมดูล Step up (วงจรเพิ่มแรงดันไฟ) เพื่อเพิ่มเป็น 5V ก่อนที่จะนำเข้า LCD

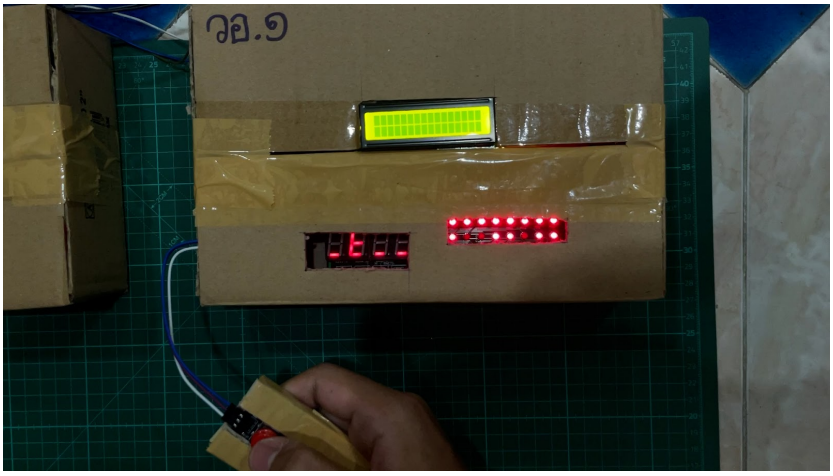
Step up Module (0.9-5V to 5V)

2. ตัวอักษรบน LCD ขึ้นช้าเกินไป

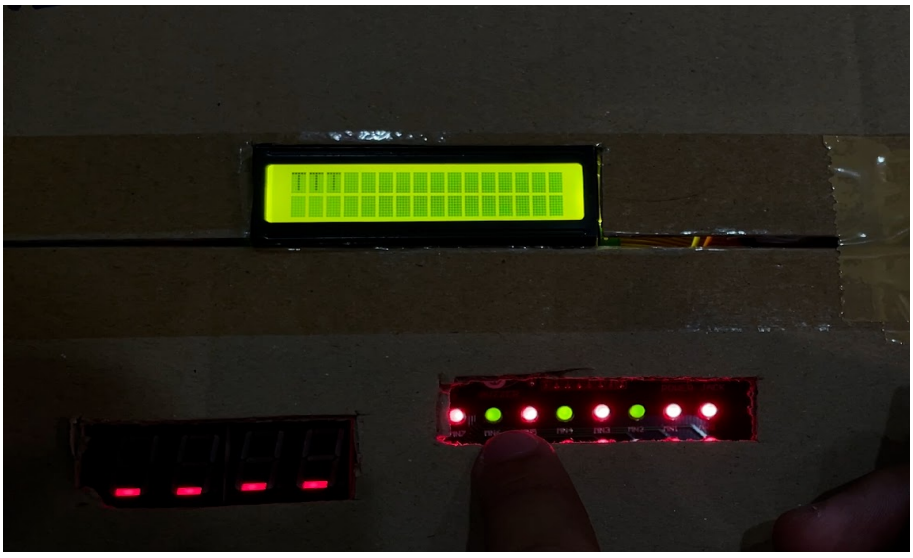
   ในขณะที่กดสัญญาณส่งไปแล้ว ข้อความบนจอ LCD ขึ้นช้าเกินไป จากการหาข้อผิดพลาดพบว่า เป็นการกำหนด Clock การอ่านเขียนของ LCD ช้าเกินไป ทำให้ตัวอักษรที่ขึ้นดีเลย์ออกไป จึงแก้ปัญหาโดยการคำนวณและปรับ Clock ใหม่

กระบวนการทดสอบ

เมื่อผู้ใช้ฝั่งส่งสัญญาณทำการกดปุ่มส่งสัญญาณรหัสมอร์สแล้วจะทำการส่งสัญญาณออกไปยังบอร์ดที่ทำการรับสัญญาณ พร้อมทั้งแสดงตัวอักษรบน LED 7-Segment



ในส่วนของบอร์ดที่ทำหน้าทีรับสัญญาณจะทำการรับสัญญาณรหัสมอร์สและแสดงผลตามสัญญาณที่ได้รับบนจอ LCD พร้อมทั้งแสดงในรูปแบบสัญญาณ ASCII บน LED แถวบนโดยสถานะ LED สีเขียวคือ 1 และสีแดงคือ 0



อ้างอิง

ไทยรัฐออนไลน์. (2022, 1 26). *รู้จัก “รหัสมอร์ส (Morse Code)” พร้อมวิธีอ่านความหมาย*. thairath. https://www.thairath.co.th/lifestyle/money/2296506

บวบทอง, ณ. (2016, 5 2). *การควบคุมโมดูล LCD ด้วย FPGA*. narong. http://narong.ece.engr.tu.ac.th/vhdl/document/LCD%20Module.pdf

แสงแก้ว, จ. (2017, 6 1). *การเรียนรู้และใช้งาน FPGA ด้วยภาษา VHDL*. วิทยาการข้อมูลและนวัตกรรมดิจิทัล. https://dsdi.msu.ac.th/?article=fpga

ahmetdenizyilmaz/Morse-Code-FPGA: fpga code convert morse code inputs to ascii. (n.d.). GitHub. Retrieved December 16, 2022, from https://github.com/ahmetdenizyilmaz/Morse-Code-FPGA

Braille, L., & Fakoó, A. (n.d.). *Siekoo Alphabet, 7-Segment Alphabet (seven-segment alphabet), reliefscript*. fakoo.de. Retrieved December 16, 2022, from https://fakoo.de/en/siekoo.html

ภาคผนวก

วีดิโอแนะนำชิ้นงาน

Youtube : https://youtu.be/pb_ZKF-AwDo

Google Drive : https://drive.google.com/drive/folders/1RVMemOhS7Yep8woWQYFQxu3xx5Y6MvHg?usp=share_link

4 layer VHDL Code

Input_Converter_Device.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity Clock_Divider is
port
( clk,reset: in std_logic;
clock_out: out std_logic
);
end Clock_Divider;

architecture bhv of Clock_Divider is

signal count: integer:=1;
signal tmp : std_logic := '0';

begin

process(clk,reset)
begin
if(reset='1') then
count<=1;
tmp<='0';
elsif(clk'event and clk='1') then
count <=count+1;
if (count = 10000000) then -- 20MHZ/20000000 = 2hz
tmp <= NOT tmp;
count <= 1;
end if;
end if;
clock_out <= tmp;
end process;

end bhv;
----------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity Input_Converter is
port (
clk: in std_logic;
A : in std_logic;
B:out std_logic_vector(1 downto 0);
C:out std_logic_vector(1 downto 0)
);
```

```
end Input_Converter;

architecture bhv of Input_Converter is
signal countt: integer:=1;
signal idlecount: integer:=1;
signal tmp : std_logic := '0';
signal pretmp : std_logic := '0';
signal outsig: std_logic_vector(1 downto 0);
signal outsigdata: std_logic_vector(1 downto 0);
constant  TIMEUNIT50m :integer := 4000000;

begin

process(A,clk)
begin
if(rising_edge(clk)) then
outsig<="00";
	if(A='1') then
		countt<=countt+1;
		idlecount<=0;
	else
		if(idlecount>TIMEUNIT50m) then
			outsig<="11";
			outsigdata<="11";
			idlecount<=0;
		else
			idlecount<=idlecount+1;
		end if;

		if(countt>TIMEUNIT50m) then --- more than 0.5 s
			outsig<="10";
			outsigdata<="10";
		elsif (countt>5000) then
			outsig<="01";
			outsigdata<="01";
		end if;
		countt<=0;
	end if;
end if;


end process;

B<=outsig;
C<=outsigdata;
end bhv;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
```

```
entity Input_Converter_Device is
port (
clk: in std_logic;
A : in std_logic;
B:out std_logic_vector(1 downto 0);
C:out std_logic_vector(1 downto 0);
clkout: out std_logic
);
end entity;

architecture rtl of Input_Converter_Device is

component Input_Converter is

port (
clk: in std_logic;
A : in std_logic;
B:out std_logic_vector(1 downto 0);
C:out std_logic_vector(1 downto 0)


);


end component;


component Clock_Divider is
 port
( clk,reset: in std_logic;
clock_out: out std_logic
);
end component;

begin
G1: Clock_Divider port map(clk,reset,clkout);
G2: Input_Converter port map(clk,A,B,C);


end rtl;
```

Finite_state_Machine.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
entity Finite_State_Machine is
port (
clk: in std_logic;
A :in std_logic_vector(1 downto 0);
B:out std_logic_vector(7 downto 0);
di: out std_logic

);
end Finite_State_Machine;
architecture bhv of Finite_State_Machine is
signal state: std_logic_vector(7 downto 0);
signal outsig: std_logic_vector(7 downto 0);
signal insig: std_logic_vector(1 downto 0);
signal send: std_logic:= '0';
begin
--insig<=A;
process(clk)
begin

if(rising_edge(clk)) then
send<='0';
       if(A/="00") then
       case state is
       when "00000000" =>--START
              case A is
              when "01" => --dot
        state<="01000101";--E
              when "10" => --dash
      state<="01010100";--T
              when others=>
       end case;
       when "01000101" => --E
       ------
              case A is
              when "01" => --dot
        state<="01001001";--I
              when "10" => --dash
      state<="01000001";--A

       when others=>
       end case;
       when "01001001" => --I
              case A is
              when "01" => --dot
        state<="01010011";--S
              when "10" => --dash
```

```
state<="01010101";--U

       when others=>
 end case;
 when "01010011" => --S
 case A is
       when "01" => --dot
  state<="01001000";--H
       when "10" => --dash
state<="01010110";--V

       when others=>

 end case;
 when "01001000" => --H
 case A is
       when "01" => --dot
  state<="00110101";--5
       when "10" => --dash
state<="00110100";--4
       when others=>
 end case;
 when "01010110" =>--V
 case A is
       when "01" => --dot
  state<="11111111";--NULL
       when "10" => --dash
state<="00110011";--3
       when others=>
 end case;
 when "01010101" =>--U
 case A is
       when "01" => --dot
  state<="01000110";--f
       when "10" => --dash
state<="11111110";--NULL2

       when others=>
 end case;
 when "11111110" =>--NULL2
 case A is
       when "01" => --dot
  state<="11111111";--NULL
       when "10" => --dash
state<="00110010";--2
       when others=>
 end case;
 when "01000001" =>--A
 case A is
       when "01" => --dot
  state<="01010010";--R
```

```
       when "10" => --dash
state<="01010111";--W
       when others=>
 end case;
 when "01010010" =>--R
 case A is
       when "01" => --dot
  state<="01001100";--L
       when "10" => --dash
state<="11111111";--NULL
       when others=>
 end case;
       when "01010111" =>--R
 case A is
       when "01" => --dot
  state<="01010000";--P
       when "10" => --dash
state<="01001010";--J
       when others=>
 end case;
       when "01001010" =>--J
 case A is
       when "01" => --dot
  state<="11111111";--NULL
       when "10" => --dash
state<="00110001";--J
       when others=>

 end case;
       when "01010100" =>--T
 case A is
       when "01" => --dot
  state<="01001110";--N
       when "10" => --dash
state<="01001101";--M
       when others=>
 end case;
       when "01001110" =>--N
 ------
 case A is
       when "01" => --dot
  state<="01000100";--D
       when "10" => --dash
state<="01001011";--K
       when others=>

 end case;
 ------

       when "01000100" =>--D
 ------
```

```
case A is
       when "01" => --dot
 state<="01000010";--B
       when "10" => --dash
state<="01011000";--X
       when others=>
end case;
       when "01000010" =>--B
case A is
       when "01" => --dot
 state<="00110110";--6
       when "10" => --dash
state<="11111111";--NULL
       when others=>
end case;
when "01001011" =>--K
case A is
       when "01" => --dot
 state<="01000011";--C
       when "10" => --dash
state<="01011001";--Y
       when others=>
end case;
when "01001101" =>--M
------
case A is
       when "01" => --dot
 state<="01000111";--G
       when "10" => --dash
state<="01001111";--O
       when others=>
end case;
when "01000111" =>--G
case A is
       when "01" => --dot
 state<="01011010";--Z
       when "10" => --dash
state<="01010001";--Q
       when others=>
end case;
------

when "01011010" =>--Z
------
case A is
       when "01" => --dot
 state<="00110111";--7
       when "10" => --dash
state<="11111111";--NULL
       when others=>
```

```
        end case;
        ------
        when "01001111" =>--O
        ------
        case A is
               when "01" => --dot
         state<="11111101";--NULL8
               when "10" => --dash
        state<="11111011";--NULL9
               when others=>
        end case;
               when "11111101" =>--NULL8
        case A is
               when "01" => --dot
         state<="00111000";--8
               when "10" => --dash
        state<="11111111";--NULL
               when others=>

        end case;
        when "11111011" =>--NULL9
        case A is
               when "01" => --dot
         state<="00111001";--9
               when "10" => --dash
        state<="00110000";--0
               when others=>
        end case;
        when others =>
         state<="11111111";--NULL

        end case;

               if (A="11")  then
                      outsig<=state;
                      state<="00000000";
                      send<='1';
               end if;

        end if;
end if;
insig<="00";
end process;
B<=outsig;
di<=send;
end bhv;
```

Ascii2led.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ascii2led is
Port ( ASCII : in STD_LOGIC_VECTOR (7 downto 0);
LED : out STD_LOGIC_VECTOR (6 downto 0));
end ascii2led;
architecture Behavioral of ascii2led is
BEGIN
WITH ASCII SELECT
LED <= "0000001" WHEN "00110000", --0
"1001111" WHEN "00110001", --1
"0010010" WHEN "00110010", --2
"0000110" WHEN "00110011", --3
"1001100" WHEN "00110100", --4
"0100100" WHEN "00110101", --5
"0100000" WHEN "00110110", --6
"0001111" WHEN "00110111", --7
"0000000" WHEN "00111000", --8
"0000100" WHEN "00111001", --9
"0000010" WHEN "01000001", --A
"1100000" WHEN "01000010", --B
"1110010" WHEN "01000011", --C
"1000010" WHEN "01000100", --D
"0110000" WHEN "01000101", --E
"0111000" WHEN "01000110", --F
"0100001" WHEN "01000111", --G
"1101000" WHEN "01001000", --H
"0111011" WHEN "01001001", --I
"0100111" WHEN "01001010", --J
"0101000" WHEN "01001011", --K
"1110001" WHEN "01001100", --L
"0101010" WHEN "01001101", --M
"1101010" WHEN "01001110", --N
"1100010" WHEN "01001111", --O
"0011000" WHEN "01010000", --P
"0001100" WHEN "01010001", --Q
"1111010" WHEN "01010010", --R
"0100101" WHEN "01010011", --S
"1110000" WHEN "01010100", --T
"1100011" WHEN "01010101", --U
"1010101" WHEN "01010110", --V
"1010100" WHEN "01010111", --W
"1101011" WHEN "01011000", --X
"1000100" WHEN "01011001", --Y
"0010011" WHEN "01011010", --Z
"1110111" WHEN OTHERS; -- UNDEFINED
end Behavioral;
```

Shift_register.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity Shift_register is
port (
clk: in std_logic;
id: in std_logic;
leddata :in std_logic_vector(6 downto 0);
data2led0 :out std_logic_vector(6 downto 0);
data2led1 :out std_logic_vector(6 downto 0);
data2led2 :out std_logic_vector(6 downto 0);
data2led3 :out std_logic_vector(6 downto 0)

);

end Shift_register;

architecture bhv of Shift_register is
signal Q0: std_logic_vector(6 downto 0):="0110110";
signal Q1: std_logic_vector(6 downto 0):="0110110";
signal Q2: std_logic_vector(6 downto 0):="0110110";
signal Q3: std_logic_vector(6 downto 0):="0110110";
signal Q4: std_logic_vector(6 downto 0):="0110110";
signal Q5: std_logic_vector(6 downto 0):="0110110";
begin
process(clk,leddata,id,clear)
begin
if(rising_edge(clk)) then
 if(id='1' and (leddata/= "0110110" or Q3="1111110"))then
 --Q5<=Q4;
 --Q4<=Q3;
 Q3<=Q2;
 Q2<=Q1;
 Q1<=Q0;
 Q0<=leddata;
 end if;
end if;

end process;
data2led0<=Q0;
data2led1<=Q1;
data2led2<=Q2;
data2led3<=Q3;
data2led4<=Q4;
data2led5<=Q5;

end bhv;
```

lcd.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lcd is
port ( clk : in std_logic; --clock i/p
lcd_rw  : out std_logic; --read & write control
lcd_e : out std_logic; --enable control

lcd_rs  : out std_logic; --data or command control
data : out std_logic_vector(7 downto 0); --data line
ASCII : in std_logic_vector(7 downto 0));
end lcd;

architecture Behavioral of lcd is
constant N: integer :=21;
type arr is array (1 to N) of std_logic_vector(7 downto 0);

signal temp: std_logic_vector(7 downto 0);
begin
lcd_rw <= '0'; --lcd write
process(clk)
variable i : integer := 0;
variable j : integer := 1;
variable k : integer := 6;
variable datas : arr
:=(X"38",X"0c",X"06",X"01",X"80",x"20",x"20",x"20",x"20",x"20",x"20",x"20",x"20"
,x"20",x"20",x"20",x"20",x"20",x"20",x"20",x"20");
begin
if clk'event and clk = '1' then

      if (reset='0') then
            k:=6;
            data<=datas(4)(7 downto 0);
      end if;
      if ASCII /="00000000" then
            temp<=ASCII;
      if i <= 2000000 then
            i := i + 1;
            lcd_e <= '1';
            datas(k):=temp;
            data <= datas(k)(7 downto 0);
            elsif i > 2000000 and i < 4000000 then
            i := i + 1;
            lcd_e <= '0';
            elsif i = 4000000 then
            j := j + 1;
            k:=k+1;
```

```
            if k=21 then
            k:=6;
            data<=datas(4)(7 downto 0);
            end if;
            i := 0;
        end if;
        end if;



        if j <= 5 then
            lcd_rs <= '0'; --command signal
                if i <= 2000000 then
            i := i + 1;
            lcd_e <= '1';
            data <= datas(j)(7 downto 0);
            elsif i > 2000000 and i < 4000000 then
            i := i + 1;
            lcd_e <= '0';
            elsif i = 4000000 then
            j := j + 1;
            i := 0;
        end if;
        elsif j > 4 then
            lcd_rs <= '1'; --data signal
        end if;
        if j = 22 then --repeated display of data
            j := 4;
        end if;
end if;
end process;
end Behavioral;
```