

ออกแบบไอซีดิจิตอลด้วย **FPGA** และ **CPLD** ภาคปฏิบัติ โดยใช้วิธี

Schematic

ซอฟต์แวร์ทูล ISE WebPACK

นายเจริญ วงศ์ชุมเย็น

นาย ณรงค์ ทองนิม

ออกแบบไปชีดิจิตอลด้วย FPGA และ CPLD ภาคปฏิบัติ โดยใช้วิธี Schematic ซอฟต์แวร์ทุล ISE WebPACK
เขียนโดย เจริญ วงศ์ชุมยืน ณรงค์ ทองกิม

ราคา 275 บาท

ส่วนลิขสิทธิ์ © พ.ศ. 2556 โดย เจริญ วงศ์ชุมยืน ณรงค์ ทองกิม
ห้ามลอกเลียนแบบ ไม่ว่าส่วนหนึ่งส่วนใดของหนังสือเล่มนี้ นอกจากจะได้รับอนุญาต

พิมพ์ครั้งที่ 1 มิถุนายน 2556

ข้อมูลทางบรรณาธุรกรรมของหอสมุดแห่งชาติ

เจริญ วงศ์ชุมยืน.

ออกแบบไปชีดิจิตอลด้วย FPGA และ CPLD ภาคปฏิบัติ โดยใช้วิธี Schematic ซอฟต์แวร์ทุล ISE WebPACK.--

เชียงใหม่ : ทีดีอาร์ดี, 2556.

319 หน้า.

1. การออกแบบวงจรอิเล็กทรอนิกส์. I. ณรงค์ ทองกิม, ผู้แต่งร่วม. I. ชื่อเรื่อง.

621.38150285

ISBN 978-616-335-441-9

จัดทำโดย

เจริญ วงศ์ชุมยืน

301/183 หมู่บ้านรุ่งอรุณ 2 ซอยคลองกรุง 7 แขวงลำปลาทิวา เขตลาดกระบัง 10520

โทร 02-329-8341

จัดพิมพ์และจัดจำหน่ายโดย

ห้างหุ้นส่วนจำกัด ทีดีอาร์ดี

150/84 หมู่ 10 ต.ป่าแคน อ.เมืองเชียงใหม่ จ.เชียงใหม่ 51000

โทร 053-904-025

คำนำ

ดิจิตอลเข้ามามีบทบาทย่างมากในชีวิตประจำวันทุกวันนี้ อุปกรณ์และเครื่องใช้ต่างๆ ล้วนมีดิจิตอลเข้าไปเกี่ยวข้องไม่ว่าจะเป็นคอมพิวเตอร์ เครื่องใช้ภายในบ้าน ภายในสำนักงาน ทางทหาร ทางการแพทย์ ระบบเครือข่าย และ สื่อสาร เป็นต้น

การออกแบบhardwareดิจิตอลในปัจจุบันนั้นเปลี่ยนแปลงอย่างรวดเร็ว นับวันวงจรดิจิตอลจะมีขนาดใหญ่และซับซ้อนขึ้นเรื่อยๆ ทางเลือกในการออกแบบ ไอซี (Integrated circuit = IC) เพื่อผลิต hardwareดิจิตอลจำนวนมาก ว่ากันเป็นล้านชิ้นควรเป็นการออกแบบ ไอซีด้วย ASIC (Application-specific Integrated circuit) ซึ่งต้นทุนต่อหน่วยจะถูกมาก แต่ถ้าเป็นการสร้างต้นแบบหรือผลิต ไปถึงระดับหลักแสนชิ้น FPGA (Field Programmable Gate Array) หรือ CPLD (Complex Programmable Logic Device) จะเป็นทางเลือกที่ดีกว่า เนื่องจากต้นทุนเริ่มต้นต่ำกว่ามากและเมื่อเกิดข้อผิดพลาดก็สามารถทำการแก้ไขได้ง่าย

ผู้ประกอบการ หรือ นักวิจัยหลายคนที่มีความตั้งใจที่จะออกแบบวงจรดิจิตอลเองแต่ทำซื้อ ไอซีที่ตรงกับความต้องการไม่ได้ ทำให้บางคนต้องล้มเลิกความตั้งใจ หลายคนอาจสังสัยว่าทำไม่แพ่งงานของอุปกรณ์ที่ใช้เทคโนโลยีชั้นสูงจริงมีขนาดเล็กมากแต่กลับมีประสิทธิภาพสูง ความสำเร็จดังกล่าวนั้นส่วนหนึ่งมาจากเทคโนโลยีด้านการออกแบบ ไอซี (IC Design) FPGA และ CPLD เข้ามามีบทบาทย่างมากทางด้านการออกแบบ ไอซี (IC Design) ในปัจจุบัน โดยที่ประเทศไทยได้เริ่มใช้ FPGA ในงานวิจัยเมื่อประมาณ 10 กว่าปีมาแล้ว ทั้งๆ ที่มีการผลิต FPGA ประมาณกว่า 20 ปีแล้ว ปัญหาหนึ่งที่เกิดขึ้นในระยะเริ่มแรก คือ ซอฟต์แวร์ทุกและ FPGA มีราคาแพงมาก และคอมพิวเตอร์ในสมัยนั้นยังทำงานได้ช้า

หนังสือเล่มนี้ จะยกตัวอย่างการออกแบบวงจรดิจิตอลทั้งภาคทฤษฎีและภาคปฏิบัติ เพื่อสร้างทักษะและกระบวนการคิดเพื่อนำไปวิเคราะห์และแก้ปัญหาที่เกิดขึ้นในการออกแบบวงจรดิจิตอลที่มีความซับซ้อนໄ้ด้วยสามารถทดลองได้จริงทุกการทดลอง ได้แก่ ลอจิกเกต แลตช์ ฟลิปฟลופ วงจรคอมบินेशัน และ วงจรซีเควนเชียล รวมทั้งวงจรที่ใช้จริงในทางปฏิบัติ เช่น เครื่องนับจำนวน นาฬิกาดิจิตอล เป็นต้น โดยวงจรเหล่านี้ถูกออกแบบรวมไว้ใน ไอซี FPGA หรือ CPLD เพียงตัวเดียว เสมือนเป็น ไอซีเบอร์ใหม่ เราเรียกกระบวนการนี้ว่า “การออกแบบ ไอซี” วิธี Schematic เป็นวิธีวัดผังวงจรโดยใช้คอมพิวเตอร์แล้วนำไปดาวน์โหลดลง FPGA หรือ CPLD จึงทำได้รวดเร็วกว่าการนำ ไอซี เช่น ตระกูล TTL ต่างๆ ไปต่อวงจรบนแผงต่อวงจรหรือ โปรตอโพร์ต จึงลดความยุ่งยากในการต่อวงจรและการตรวจสอบลงมาได้มาก ทำให้ลดเวลาในการเรียนการสอนวิชาดิจิตอลในภาคปฏิบัติลง ไปได้มาก และในข้อสุดท้ายจะอธิบายวิธีนำไฟล์วงจรที่ซับซ้อนที่เขียนด้วยภาษา VHDL ซึ่งเป็นภาษาระดับสูงที่สามารถดาวน์โหลดได้ทางอินเตอร์เน็ตหรือมีในหนังสือทั่วไปนำไปสร้างไฟล์ Symbols หนังสือเล่มนี้ หมายถึงนักศึกษา อาจารย์ และ ผู้สนใจทางด้านการออกแบบวงจรดิจิตอลพื้นฐานจนถึงขั้นสูง โดยมีการออกแบบวงจรต่างๆ มากถึง 30 วงจร

ผู้เขียนหวังเป็นอย่างยิ่งว่าหนังสือเล่มนี้จะช่วยเติมเต็มความรู้ด้านดิจิตอลและ “การออกแบบ ไอซี” ซึ่งเป็นอุตสาหกรรม “ต้นน้ำ” ภายในประเทศไทย “มีมูลค่าทรัพย์สินทางปัญญาสูง” และอุตสาหกรรมต่อเนื่องทางด้านอิเล็กทรอนิกส์และคอมพิวเตอร์ hardware ให้มีความเจริญรุ่งหน้าต่อไป

นาย เจริญ วงศ์ชุมย์

นาย ณรงค์ ทองดิษฐ์

== หน้าว่าง ==

สารบัญ

ค่า牘า	3
สารบัญ.....	5
ส่วนประกอบของหนังสือเล่มนี้.....	11
บทที่ 1 การออกแบบวงจรดิจิตอลด้วย FPGA และ CPLD	13
กล่าวนำ.....	13
1.1 ไอซีมารฐาน.....	13
1.2 CPLD	14
1.3 FPGA	16
1.4 ข้อเปรียบเทียบระหว่าง FPGA และ CPLD กับไมโครคอนโทรลเลอร์	17
1.5 กระบวนการออกแบบ.....	17
1.6 ขั้นตอนการออกแบบวงจรดิจิตอลด้วย FPGA หรือ CPLD	19
1.7 การติดตั้งซอฟต์แวร์ทุก.....	21
1.8 บอร์ดทดลอง	24
1.8.1 CPLD Explorer XC9572XL.....	24
1.8.2 CPLD Explorer XC9572	29
1.8.3 FPGA Discovery-III XC3S200	30
1.8.4 บอร์ดทดลองรุ่นอื่นๆ	37
สรุป	37
คำถ้าท้ายบท.....	37
บทที่ 2 ออกแบบ logic ดิจิตอลด้วย FPGA และ CPLD โดยวิธี Schematic.....	39
กล่าวนำ.....	39
2.1 การออกแบบวงจรแบบชิงโกรนัสด้วย CPLD โดยวิธี Schematic	39
2.1.1 ขั้นตอนการออกแบบวงจร (Design Entry)	39
2.1.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification).....	54
2.1.3 การสังเคราะห์วงจร (Design synthesis)	54
2.1.4 Design Implementation.....	55
2.1.5 การโปรแกรมข้อมูลวงจรลงชิพ	59
2.1.6 การเปิดไฟล์ที่เก็บออกแบบไว้แล้วมาใช้งาน	62
2.2 การออกแบบวงจรแบบชิงโกรนัสโดยใช้ FPGA ด้วยวิธี Schematic	64
2.2.1 ขั้นตอนการออกแบบวงจร (Design Entry)	64

2.2.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification).....	69
2.2.3 การสังเคราะห์วงจร (Design synthesis).....	69
2.2.4 Design Implementation.....	70
2.2.5 การโปรแกรมข้อมูลวงจรลงชิป.....	74
2.3 การจำลองการทำงานเชิงพุทธิกรรมของวงจรที่ออกแบบด้วย FPGA และ CPLD.....	83
2.3.1 ขั้นตอนการออกแบบวงจร.....	84
2.3.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification).....	85
2.3.3 การสังเคราะห์วงจร (Design synthesis).....	88
2.3.4 Design Implementation.....	88
2.3.5 การโปรแกรมข้อมูลวงจรลงชิป.....	90
2.4 การสร้าง Source File ที่ได้จากการออกแบบเพื่อเก็บไฟล์ไว้ทำ Symbols สำหรับ CPLD.....	91
2.4.1 ขั้นตอนการออกแบบวงจร.....	91
2.5 การออกแบบวงจรนับ 4 โดยใช้ CPLD และมีโโนโนสเตเบิลอย่างจ่ายเป็นวงแร็กเก็บาร์.....	95
2.5.1 ขั้นตอนการออกแบบวงจร.....	95
2.5.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design verification).....	102
2.5.3 การสังเคราะห์วงจร (Design synthesis).....	102
2.5.4 Design Implementation.....	103
2.5.5 การโปรแกรมข้อมูลวงจรลงชิป.....	104
2.6 การสร้าง Source File ที่ได้จากการออกแบบเพื่อเก็บไฟล์ไว้ทำ Symbols สำหรับ FPGA.....	105
2.7 การออกแบบวงจรนับ 4 โดยใช้ FPGA และมีโโนโนสเตเบิลอย่างจ่ายเป็นวงแร็กเก็บาร์.....	105
2.8 การใช้บัส.....	105
สรุป	111
คำถามท้ายบท.....	111
บทที่ 3 ลอจิกเกตและวงจรคอมบินेशัน	113
กติกาคำนวณ.....	113
3.1 ลอจิกเกต	113
3.1.1 การออกแบบวงจรลอจิกเกต	115
3.1.2 การออกแบบวงจรลอจิกเกต (ต่อ).....	125
3.1.3 การออกแบบวงจรรับฟ้าเรอร์	127
3.2 วงจรบวก	127

3.2.1 การออกแบบบวกจาร Half adder	130
3.2.2 การออกแบบบวกจาร Full adder	131
3.3 วงจรคอดรหัส	132
3.3.1 การออกแบบบวกจาร 2 to 4 Decoder	135
3.3.2 การออกแบบบวกจารคอดรหัส BCD เป็นเลขฐานเซกเมนต์	137
3.4 วงจรเปรียบเทียบ	141
3.4.1 การออกแบบบวกจารเปรียบเทียบ	142
3.5 วงจรเข้ารหัส.....	143
3.5.1 การออกแบบบวกจารเข้ารหัส	145
3.6 วงจรแมตติเพล็กซ์อร์.....	147
3.6.1 การออกแบบบวกจารแมตติเพล็กซ์อร์แบบ 2 อินพุต 1 เอาต์พุต	150
3.6.2 การออกแบบบวกจารแมตติเพล็กซ์อร์แบบ 4 อินพุต 1 เอาต์พุต	151
3.7 วงจรคีมัคติเพล็กซ์อร์.....	153
3.7.1 การออกแบบบวกจารคีมัคติเพล็กซ์อร์	154
สรุป	154
คำถามท้ายบท.....	155
บทที่ 4 แล็ตช์ พลิปฟลอป และวงจรซีคูนเชียล.....	157
กติกาบasis.....	157
4.1 แล็ตช์	157
4.1.1 การออกแบบบวกจารแล็ตช์	159
4.2 พลิปฟลอป	161
4.2.1 การออกแบบบวกจาร D Flip-Flop	161
4.2.2 การออกแบบบวกจาร JK Flip-Flop.....	162
4.3.3 การออกแบบบวกจาร T Flip-Flop.....	163
4.2.1 การออกแบบบวกจาร D Flip-Flop.....	165
4.2.2 การออกแบบบวกจาร JK Flip-Flop และ T Flip-Flop	167
4.3 วงจรนับเลขฐานสอง (Binary counter)	170
4.3.1 การออกแบบบวกจารนับแบบอะซิงโกรันต์ส (Asynchronous counter)	170
4.3.2 การออกแบบบวกจารนับแบบซิงโกรันต์ส (Synchronous counter)	171
4.3.1 การออกแบบบวกจารนับขึ้นแบบบิบิกล	174

4.3.2 การออกแบบวงจรนับลงแบบบิบิล.....	178
4.3.3 การออกแบบวงจรนับขึ้นแบบชิงโครนัส	181
4.3.4 การออกแบบวงจรนับลงแบบชิงโครนัส	185
4.3.5 การออกแบบวงจรนับขึ้น-ลงแบบชิงโครนัส	188
4.3.6 การออกแบบวงจรนับขึ้น-ลง 2 หลัก (0-FF) ที่แสดงทางเข่วนเชกเม้นต์	193
4.3.7 การออกแบบวงจรนับขึ้น-ลง 4 หลัก (0-FFFF) ที่แสดงทางเข่วนเชกเม้นต์.....	200
4.4 การออกแบบวงจรนับ N แบบอะชิงโครนัสและแบบชิงโครนัส	206
4.4.1 การออกแบบวงจรนับ N แบบอะชิงโครนัส	206
4.4.2 การออกแบบวงจรนับ N หรือวงจรหารความถี่ N แบบชิงโครนัส.....	206
4.4.1 การออกแบบวงจรนับ 10 แบบนับขึ้นแบบอะชิงโครนัส	207
4.4.2 การออกแบบวงจรนับ 10 แบบนับขึ้นแบบชิงโครนัส.....	210
4.4.3 การออกแบบวงจรนาฬิกาดิจิตอลแบบตั้งเวลาหลักนาทีและชั่วโมงได้.....	213
4.5 ชิฟต์รีจิสเตอร์	217
4.5.1 การออกแบบวงจรชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรม–ออกแบบอนุกรมและ/หรือแบบขนาน	217
4.5.2 การออกแบบวงจรชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรมหรือขนาน-ออกแบบอนุกรมและ/หรือขนาน	217
4.5.3 การออกแบบวงจรดีเบาเซอร์	218
4.5.1 การออกแบบวงจรชิฟต์รีจิสเตอร์ 4 บิต.....	221
4.5.2 การออกแบบวงจรดีเบาเซอร์แบบต่างๆ.....	225
4.6 การออกแบบวงจรชีควนเชียล.....	228
4.6.1 Finite state machine.....	228
4.6.2 State diagram และ State table ของ Moor-type FSM.....	229
4.6.3 State diagram และ State table ของ Mealy-type FSM	230
4.6.4 ขั้นตอนการออกแบบ Finite state machine	231
4.6.1 การออกแบบวงจรตรวจจับลำดับแบบ Moor-type FSM	252
4.6.2 การออกแบบวงจรตรวจจับลำดับแบบ Mealy-type FSM.....	255
4.7 วิธี Schematic โดยใช้ Symbols ที่เป็นไฟล์ภาษา VHDL	256
4.7.1 VHDL พื้นฐาน.....	256
4.7.2 การตั้งชื่อ	259

4.7.3 ประเภทของพอร์ต	260
4.7.4 ชนิดข้อมูล.....	260
4.7.5 โอลีโอเปอร์เตอร์.....	261
4.7.6 ออบเจกต์.....	263
4.7.7 Attribute.....	264
4.7.8 คำสั่งคอนโทรลเรนต์	265
4.7.9 คำสั่งซีเคานเซียล	268
4.7.1 การออกแบบวงจรนับขึ้น-ลง 4 หลัก (0-9999) ที่แสดงทางเวนเชกเมนต์.....	276
สรุป	287
คำถามท้ายบท.....	288
บทที่ 5 การลดรูปสมการบูลีนด้วยวิธีการของ Quine McCluskey	289
กล่าวนำ.....	289
5.1 Quine McCluskey	289
5.2 ตัวอย่างการสร้างวงจรลดคราฟต์ตัวแสดงผลเจ็ดส่วนด้วยวิธีการของ Quine McCluskey.....	297
สรุป	309
คำถามท้ายบท.....	309
บรรณานุกรม	311
ด้วย	313

== หน้าว่าง ==

ส่วนประกอบของหนังสือเล่มนี้

บทที่ 1 การออกแบบวงจรดิจิตอลด้วย FPGA และ CPLD

บทนี้ได้กล่าวถึงการการออกแบบไปชีดจิตอลตั้งแต่การใช้ไอซีมาตรฐานที่มีผลลัพธ์มาที่ไปอย่างไร รวมถึงการคิดค้นไอซี CPLD และ FPGA ที่ถูกพัฒนาขึ้นเพื่อทดแทนการออกแบบจริงโดยใช้ไอซีมาตรฐานแบบเดิม โดยได้อธิบายถึงโครงสร้างและการออกแบบด้วยไอซีดังกล่าว รวมถึงเปรียบเทียบความแตกต่างระหว่าง FPGA และ CPLD กับไมโครคอนโทรลเลอร์ รวมถึงกระบวนการออกแบบจริงว่ามีขั้นตอนอย่างไรบ้าง และการติดตั้งซอฟต์แวร์ทุก ISE WebPack 8.1i ที่จะใช้เป็นตัวอธิบายกระบวนการออกแบบและตัวอย่างวงจรทั้งหมดในหนังสือ อีกทั้งยังอธิบายถึงบอร์ดทดลองรุ่นต่างๆ และการใช้งานบอร์ดทดลองนั้นๆ เพื่อใช้ประกอบการเรียนคู่กับหนังสือเล่มนี้อีกด้วย

บทที่ 2 การออกแบบไปชีดจิตอลด้วย FPGA และ CPLD โดยวิธี Schematic

บทนี้ได้กล่าวถึงการใช้งานซอฟต์แวร์ทุก ISE WebPack 8.1i เพื่อใช้ในการออกแบบไปชีดจิตอลด้วย FPGA และ CPLD โดยวิธี Schematic หรือการวาดผังวงจร โดยอธิบายตั้งแต่ขั้นตอนการออกแบบจริง(Design Entry) การสังเคราะห์วงจร(Design synthesis) Design implementation การโปรแกรมข้อมูลวงจรลงชิป รวมไปถึงการเปิดไฟล์ที่เกี่ยวกับออกแบบไว้แล้วมาใช้งาน ทั้งของ CPLD และ FPGA อีกทั้งยังอธิบายถึงการจำลองการทำงานเชิงพฤติกรรมของวงจรที่ออกแบบด้วย FPGA และ CPLD รวมถึงตัวอย่างวงจรอย่างง่ายเพื่อให้เข้าใจกระบวนการทั้งหมดด้วย

บทที่ 3 ЛОจิกเกต

บทนี้กล่าวถึงตัวอย่างการสร้างวงจรลอจิกเกตพื้นฐาน ได้แก่ แอนด์เกต ออร์เกต และ อินเวอร์เตอร์ โดยการวาดผังวงจรตั้งแต่เริ่มต้นจนถึงการดาวน์โหลดลงชิป FPGA และ CPLD หลังจากนั้นจะเป็นตัวอย่างการออกแบบวงจรบวก วงจรตอครหัส วงจรเปรียบเทียบ วงจรเข้ารหัส วงจรแมตติเพล็กเซอร์ และวงจรดิมแมตติเพล็กเซอร์ ตั้งแต่ต้นจนจบ ซึ่งวงจรที่กล่าวมาก็สามารถอ่านได้ทันทีว่าเป็นพื้นฐานของวงจรคอมบินेशันที่มีการใช้งานโดยทั่วไป

บทที่ 4 แลตซ์ พลิปฟลوب และวงจรซีเควนเชียล

บทนี้กล่าวถึงวงจรซีเควนเชียลซึ่งเป็นวงจรที่เข้าที่พุทธองค์วงจรไม่ได้ขึ้นอยู่กับสัญญาณอินพุตแต่เพียงอย่างเดียว แต่หากขึ้นอยู่กับสถานะของสัญญาณเข้าที่พุทธในช่วงเวลาหนึ่งๆ อีกด้วย โดยจะเป็นการอธิบายถึงวงจรแลตซ์ที่สร้างจากแนวคิดและนอร์เกต หลังจากนั้นจะอธิบายถึงวงจรพลิปฟลوبแบบต่างๆ อาทิเช่น คิฟลิปฟลوب เจเคฟลิปฟลوب ทีฟลิปฟลوب เป็นต้น และยังรวมไปถึงวงจรที่ใช้ฟลิปฟลوبประเภทต่างๆ มาประยุกต์ใช้งานให้เป็นวงจรซีเควนเชียล เช่น วงจรนับทั้งแบบอะซิงโกรนัสและแบบบิชิงโกรนัส วงจรนับแบบบิปเปิล รวมไปถึงการประยุกต์ใช้งานร่วมกับวงจรซีเควนเชียลแบบต่างๆ เพื่อสร้างนาฬิกาดิจิตอล และสุดท้ายยังรวมถึงการออกแบบวงจรซีเควนเชียลทั้งแบบมาร์คและเมียลล์ไมเคลด รวมทั้งการเขียนโปรแกรมภาษา VHDL เพื่อสร้างวงจรดิจิตอลอีกด้วย

บทที่ 5 การครุปสมการบูลีนด้วยวิธีการของ Quine McCluskey

บทนี้กล่าวถึงกระบวนการครุปสมการบูลีนด้วยวิธีการของ Quine McCluskey ตั้งแต่หลักการและขั้นตอนในการลดรูปแบบต่างๆ รวมไปถึงตัวอย่างการสร้างวงจรลดรหัสตัวแสวงผลเจ็ดส่วนด้วยวิธีการของ Quine McCluskey ด้วย

== หน้าว่าง ==

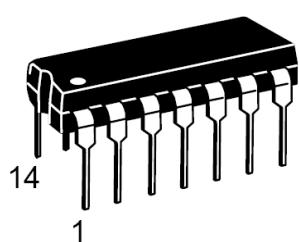
บทที่ 1 การออกแบบวงจรดิจิตอลด้วย FPGA และ CPLD

กล่าวนำ

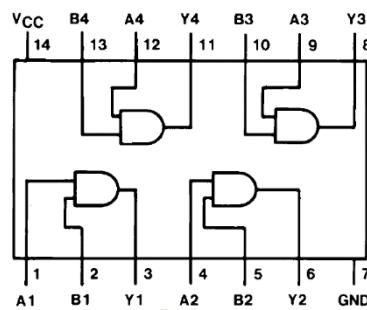
การออกแบบวงจรดิจิตอลด้วย FPGA และ CPLD ถือเป็นพัฒนาการของการออกแบบวงจรดิจิตอลในยุคสมัยใหม่ ที่ทั้งทันสมัย ง่ายต่อการออกแบบ ง่ายต่อการเรียนรู้ อีกทั้งยังย่นระยะเวลาในการออกแบบให้ลดน้อยลงกว่าเดิม ได้อย่างมากมาก ซึ่งแต่เดิมกระบวนการออกแบบวงจรดิจิตอลมักจะเริ่มจากการวิเคราะห์โจทย์ออกมาเป็นตารางความจริง(Truth table) หลังจากนั้นจึงนำไปplotรูปวงจรด้วยวิธีการต่างๆ อาทิเช่น การใช้พิชณิตบูลลีน(Boolean algebra) การใช้ตารางเคนเมฟ(K-map) หรือแม่ແຕ่การใช้วิธีการของควินแมคคลัสเก็ต(Quine McCluskey) แล้วจึงนำวงจรที่ทำการลดรูปแล้วไปวาดเป็นวงจรหรือที่เรียกว่าว่าลอจิกไดอะแกรม(Logic diagram) จากนั้นจึงทำการต่อวงจรตามที่ออกแบบไว้โดยใช้ไอซีเกตพื้นฐานสำเร็จรูปที่เรียกว่า ทีทีแอล(TTL) ซึ่งกระบวนการดังกล่าวทั้งหมดนี้ จำเป็นต้องใช้เวลา ประสบการณ์และความสามารถของผู้ออกแบบเป็นอย่างยิ่ง เพื่อจะทำให้วงจรที่ออกแบบทำงานได้ตามที่คาดหวัง และใช้ปริมาณเกตให้น้อยที่สุด อีกทั้งการต่อวงจรตามที่ออกแบบโดยใช้ไอซีที่ทีทีแอล ดังกล่าวยังมีความซับซ้อนและหากเกิดปัญหาวงจรทำงานไม่ได้ตามที่ต้องการ การแก้ไขหรือหาจุดผิดพลาดก็ทำได้ยากยิ่ง แต่ด้วยเทคโนโลยีที่ก้าวหน้าในปัจจุบัน ทำให้กระบวนการออกแบบวงจรดิจิตอลมีพัฒนาการไปอย่างมาก ทำให้การดำเนินงานในขั้นตอนต่างๆ ง่ายขึ้น เร็วขึ้น สะดวกขึ้น โดยเฉพาะขั้นตอนการออกแบบ และการต่อวงจร เนื่องจากแทนที่จะใช้การต่อวงจรโดยใช้ไอซีที่ทีทีแอลแบบเดิม สามารถใช้ไอซี FPGA หรือ CPLD มาทดแทนได้ ทำให้การทำงานสะดวกรวดเร็วมากยิ่งขึ้นดังจะได้กล่าวรายละเอียดต่างๆ ดังต่อไปนี้

1.1 ไอซีมาตรฐาน

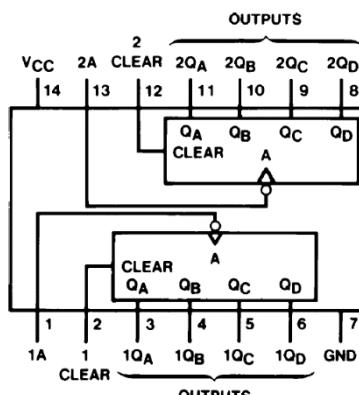
การออกแบบวงจรดิจิตอลขนาดเล็กโดยปกติจะนิยมใช้พิชณิตหรือไอซีมาตรฐาน เช่น ไอซี CMOS (อ่านว่าซีมอส) ตรรกะ 4000 และ 74HC00 หรือไอซี TTL (อ่านว่าทีทีแอล) ตรรกะ 74LS00 เป็นต้น ตัวอย่างแพคเกจ ไอซีแสดงดังรูปที่ 1.1(a) ไอซี TTL เช่น เบอร์ 74LS08 (แอนด์เกต 2 อินพุต 4 ตัว) และ CMOS เช่น เบอร์ 74HC393 (วงจรนับไบนาเรียหรือฐานสอง 4 บิต 2 ตัว) แสดงดังรูปที่ 1.1(b) และ CMOS เช่น เบอร์ 74HC393 (วงจรนับไบนาเรียหรือฐานสอง 4 บิต 2 ตัว) แสดงดังรูปที่ 1.1(c)[1-2] จะเห็นได้ว่าไอซีสำเร็จรูปเหล่านี้จะมีฟังก์ชันทำงานทางลักษณะเดียวกัน แต่ตัวอย่างภายในตัวไอซีและในสายสัญญาณ โดยความเร็วของสัญญาณต่างๆ ในลายเส้นทองแดงของ PCB (ชนิด FR4) หรือสายสัญญาณนั้นจะมีความเร็ว (ค่าคงตัว) ประมาณครึ่งหนึ่งของความเร็วแสงหรือ 15-18 เซนติเมตรต่อนาโนวินาที จากข้อจำกัดดังกล่าวทำให้การออกแบบแพลงวงจรขนาดใหญ่ที่ใช้ความถี่สูงหลายสิบเมกะเฮิร์ตซ์มีความยุ่งยากมากขึ้นและอาจทำไม่ได้



(a) แพคเกจ ไอซี



(b) รายละเอียดของ ไอซีที่เป็นแอนด์เกต



(c) รายละเอียดของ ไอซีที่เป็นวงจรนับ

รูปที่ 1.1 ตัวอย่าง ไอซีตรรกะ TTL และ CMOS

ต่อมาได้มีการคิดค้น ไอซีหรือชิปดิจิตอลออนไลน์กประสงค์ที่โปรแกรมได้เพื่อให้มีฟังก์ชันการทำงานตามที่ต้องการเป็นผลสำเร็จประมาณปี 1970 โดยเรียกว่า Programmable Logic Device = PLD (อ่านว่าพีแอลดี) ซึ่งภายในจะบรรจุวงจรลอกิจิกพื้นฐานที่มีฟังก์ชันทำงานแบบไม่ตายตัวไว้เป็นจำนวนมาก ปัจจุบันได้มีการออกแบบวงจรขนาดใหญ่โดยใช้ชิปดิจิตอลออนไลน์กประสงค์ แทนการออกแบบด้วยไอซีมาตรฐานตรรกะ TTL หรือ CMOS กันมากขึ้น บ่อยครั้งเราจะพบชิป FPGA (Field Programmable Gate Array = เอฟพีจีเอ) และ/หรือ CPLD (Complex Programmable Logic Device = ซีพีแอลดี) เป็นส่วนประกอบที่ติดตั้งอยู่บนแผงวงจร เช่น ทางด้านสื่อสาร การแพทย์ การทหาร ระบบเครือข่าย หรือ เครื่องมืออัตโนมัติ เป็นต้น

การที่เราออกแบบวงจรย่อยส่วนต่างๆ รวมไว้ใน FPGA และ/หรือ CPLD เพียงตัวเดียวหรือเพียงไม่กี่ตัว ได้นั้นจะทำให้ แผงวงจรมีขนาดลดลงอย่างมาก ทำให้ได้วงจรที่ทำงานเร็วขึ้น เพราะสายสัญญาณต่างๆ สั้นลงและสายสัญญาณส่วนใหญ่จะอยู่ภายในชิป ทำให้ได้ผลิตภัณฑ์ที่มีขนาดเล็กกะทัดรัด

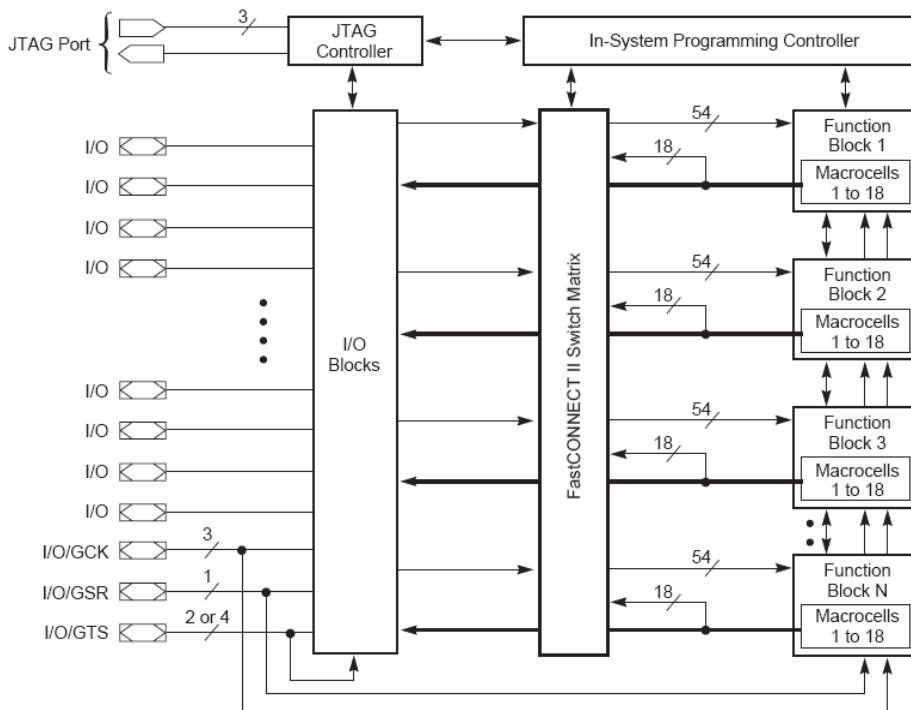
หนังสือเล่มนี้จะอธิบายการออกแบบสร้างวงจรดิจิตอลโดยใช้ FPGA และ CPLD ของบริษัท Xilinx เป็นหลัก[3-5] ซึ่งผู้ใช้งานเริ่มต้นอาจไม่มีความจำเป็นต้องศึกษารายละเอียดโครงสร้างปลีกย่อยที่อยู่ภายในชิพมากนัก การออกแบบวงจรในลักษณะนี้มีความจำเป็นต้องใช้คอมพิวเตอร์และซอฟต์แวร์ทูล (Software Tool) ช่วยในการออกแบบ

1.2 CPLD

CPLD เป็น ไอซีประเภท PLD ซึ่งเป็นชิปดิจิตอลออนไลน์กประสงค์ชนิดหนึ่งที่สามารถโปรแกรมให้มีฟังก์ชันทำงานตามที่ต้องการได้ ตัวอย่าง CPLD เบอร์ XC9536XL ที่มีความจุวงจร 800 เกตและมี 34 อินพุตเอาต์พุต (I/O) แสดงดังรูปที่ 1.2 โครงสร้างภายในชิป CPLD ตรรกะ XC9500XL แสดงดังรูปที่ 1.3

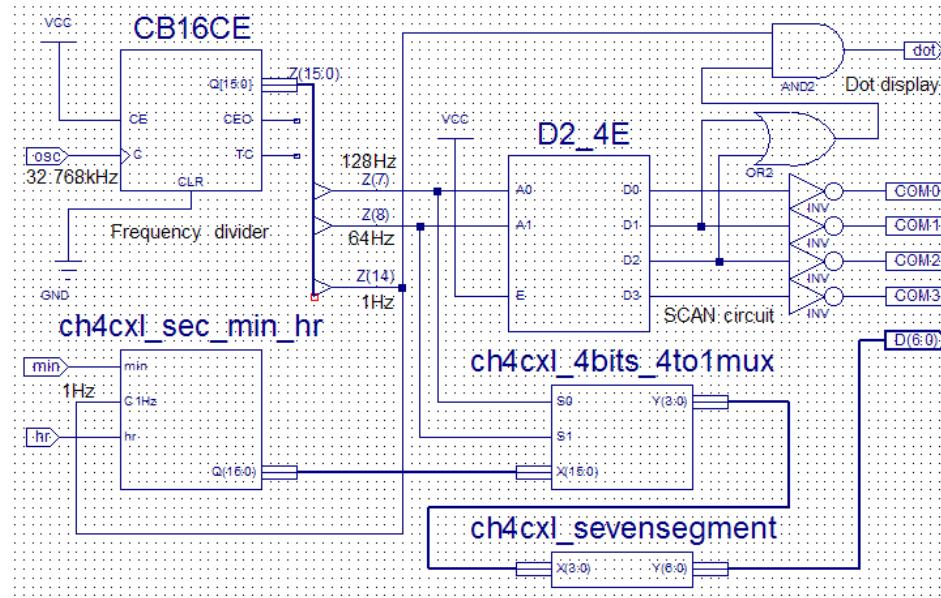


รูปที่ 1.2 ตัวอย่าง CPLD เบอร์ XC9536XL ที่มีความจุวงจร 800 เกต



รูปที่ 1.3 โครงสร้างภายในของ CPLD ตระกูล XC9500XL

โครงสร้าง CPLD จะประกอบด้วย Function Block (FB) และ I/O Block (IOB) ที่สามารถเชื่อมต่อถึงกันด้วย Switch matrix ภายใน Function Block ประกอบด้วยล็อกิกพื้นฐานต่างๆ ที่สามารถโปรแกรมได้ I/O Block จะทำหน้าที่เป็นบอร์ดที่อินพุตหรือเอาต์พุต วงจร In-system programming ใช้สำหรับโปรแกรม CPLD ผ่านทางพอร์ต JTAG (อ่านว่าเจแท็ก) โดยมี JTAG Controller เป็นตัวควบคุม ตัวอย่างวงจรนาฬิกาดิจิตอลดังรูปที่ 1.4 นี้อาจสร้างได้ด้วยไอซีมาตรฐานตระกูล 7400 ไม่น้อยกว่า 10 ตัวแต่เมื่อสร้างวงจรนี้ด้วย CPLD เบอร์ XC9572XL จะกินความจุวงจรสูงสุดเพียง 70% เท่านั้นดังรายละเอียดแสดงในรูปที่ 1.5 CPLD มีความจุวงจรต่ำมากเมื่อเทียบกับ FPGA ซึ่งจะอธิบายในหัวข้อต่อไป โดยทั่วๆ ไป CPLD จะมีความจุวงจรไม่เกิน 10,000 เกต ข้อจำกัดของ CPLD คือใช้เวลาในการโปรแกรมค่อนข้างนานเนื่องจากมีโครงสร้างตัวเก็บข้อมูลภายในแบบ EEPROM หรือ Flash และราคาแพง (ราคาก่อต้นตั้งสูงกว่า FPGA) แต่มีการโปรแกรมวงจรไว้ใน CPLD แล้วข้อมูลวงจรนั้นก็จะคงอยู่แม้ว่าจะไม่มีไฟเลี้ยงตัวชิปแล้วก็ตาม การโปรแกรมสามารถทำได้หลายครั้ง



รูปที่ 1.4 ตัวอย่างวงจรนาฬิกาดิจิตอล

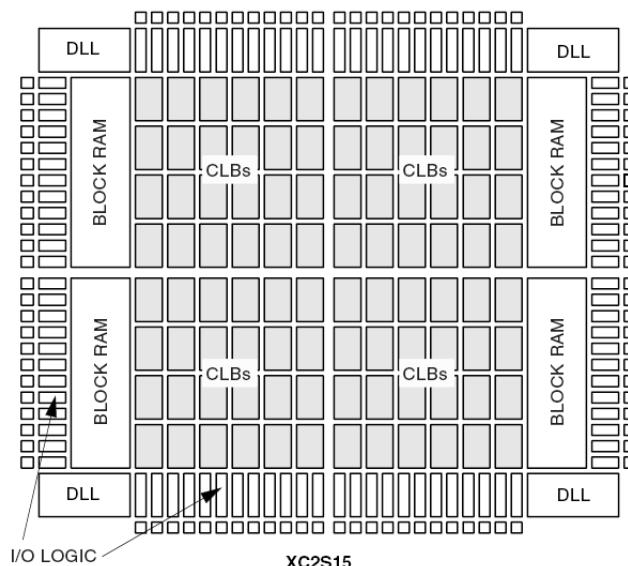
RESOURCES SUMMARY

Macrocells Used	Pterms Used	Registers Used	Pins Used	Function Block Inputs Used
50/72 (70%)	185/360 (52%)	37/72 (52%)	15/34 (45%)	98/216 (46%)

รูปที่ 1.5 แสดงรายการอุปกรณ์ที่ต้องใช้ใน CPLD

1.3 FPGA

FPGA เป็นชิพประเภท PLD เป็นชิพอนenkประสงค์ที่สามารถโปรแกรมให้เป็นวงจรดิจิตอลได้ตามที่ต้องการ แต่จะมีโครงสร้างภายในแตกต่างจาก CPLD อย่างสิ้นเชิงและซับซ้อนกว่ามาก ตัวอย่างโครงสร้างภายในของ FPGA แสดงดังรูปที่ 1.6



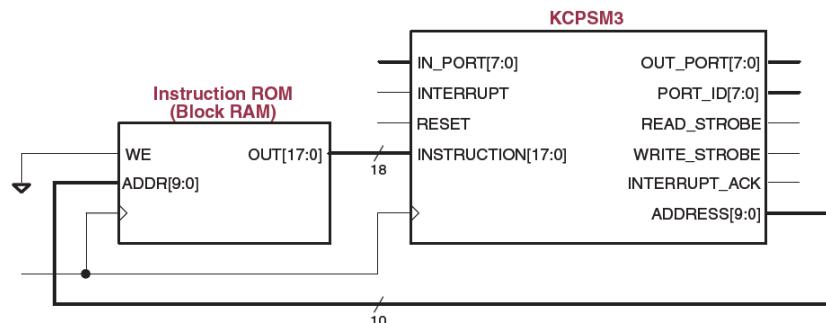
รูปที่ 1.6 โครงสร้างของ FPGA Spartan-II เบอร์ XC2S15

โครงสร้างภายใน FPGA ประกอบด้วย 2 ส่วนหลักๆ คือ Configurable Logic Block (CLB) ต่างๆ และ I/O LOGIC เพื่อใช้โปรแกรมให้เป็นวงจรคิจิตอลตามที่ต้องการได้ กายในมีอุปกรณ์ที่ทำหน้าที่เก็บพารามิเตอร์ เช่น Delay-Locked Loop (DLL) และหน่วยความจำ (Block RAM) เพื่ออำนวยความสะดวกในการออกแบบวงจร FPGA มีความจุวงจรสูงมากถึงแต่ระดับ 10,000 เกต ถึงเป็นหลักร้อยล้านเกต ซึ่งขึ้นอยู่กับเทคโนโลยีที่ใช้ในการผลิต การโปรแกรม FPGA สามารถทำได้โดยโหลดข้อมูลวงจร (Configuration data) ลงไปเก็บที่เซลล์หน่วยความจำแบบ RAM (คงเหลือ Block RAM) ที่อยู่ภายใน FPGA ดังนั้น FPGA จึงไม่มีข้อจำกัดในการโปรแกรมช้า และสามารถโปรแกรมข้อมูลวงจรลงไปได้เร็ว แต่มีข้อเสียที่สำคัญคือข้อมูลวงจรจะสูญหายหากไม่มีไฟเลี้ยง จึงต้องใช้หน่วยความจำภายนอกที่สามารถเก็บข้อมูลวงจรอยู่ได้แม้ว่าจะไม่มีไฟเลี้ยงตัวชิป เช่น Serial PROM เป็นต้น เพื่อใช้เก็บข้อมูล และจะมีการโหลดข้อมูลจาก Serial PROM ลงชิป FPGA โดยอัตโนมัติทุกครั้งที่มีการจ่ายไฟเลี้ยง

1.4 ข้อเปรียบเทียบระหว่าง FPGA และ CPLD กับไมโครคอนโทรลเลอร์

นักออกแบบวงจรคิจิตอลส่วนใหญ่จะคุ้นเคยกับการใช้งานไมโครคอนโทรลเลอร์กันดีอยู่แล้ว เช่น ตรรกะ MCS-51 และตรรกะ PIC เป็นต้น ซึ่งมีการออกแบบสถาปัตยกรรมและชุดคำสั่งไว้ภายในเรียบร้อยแล้ว จึงสามารถนำชุดคำสั่งต่างๆ มาโปรแกรมให้ไมโครคอนโทรลเลอร์ทำงานได้ตามที่ต้องการ ซึ่งต่างจาก FPGA และ CPLD ที่จะนำไปใช้ในงานออกแบบวงจรคิจิตอลขนาดใหญ่หรือความถี่สูงๆ การออกแบบวงจรคิจิตอลไว้ใน FPGA จะทำให้แพ่วงจรมีขนาดเล็กลงมาก วงจรจะทำงานได้เร็วขึ้นและมีความเชื่อถือได้สูง เพราะสายสัญญาณต่างๆ สั้นลงและมีจุดเชื่อมต่อกันจำนวนน้อยมาก

ปัจจุบันได้มีการออกแบบไมโครคอนโทรลเลอร์แบบฝังตัว (Embedded microcontroller) ดังรูปที่ 1.7 รวมไว้ภายใน FPGA ทำให้งานทำงานได้เร็วและมีความยืดหยุ่นสูง โดยไมโครคอนโทรลเลอร์จะกินพื้นที่วงจรใน FPGA ประมาณ 4-8% และสามารถ RUN ได้ที่ความถี่สูงมาก ซึ่งไมโครคอนโทรลเลอร์ที่สร้างจาก FPGA จะมีความเร็วสูงกว่าไมโครคอนโทรลเลอร์ทั่วไปที่มีขายในห้องทดลองประมาณ 2-20 เท่า และสร้างไว้ใน FPGA ได้พร้อมกันหลายๆ ตัว



รูปที่ 1.7 ตัวอย่างการออกแบบไมโครคอนโทรลเลอร์ตระกูลพิโคเบลซ์ฟังตัวใน FPGA ตระกูล Spartan-3

1.5 กระบวนการออกแบบวงจร

กระบวนการออกแบบวงจรโดยทั่วไปมักจะมีขั้นตอนหรือการทำงานเป็นขั้นตอนดังนี้

- System requirement
- System design
- System implementation
- Testing and debugging
- Documentation

กระบวนการทำ System requirement คือ การหาความต้องการว่าเราต้องการสร้างอะไร วงจรทำงานอย่างไร และมีอินพุตเอาต์พุตอะไรบ้าง กระบวนการทำ System design คือ การออกแบบและหาวิธีแก้ไขปัญหาจากขั้นตอนแรก จากนั้นจึงเป็นการทำ System implementation เพื่อสร้างงานตามที่ได้ออกแบบไว้ แล้วจึงทำการเพื่อทดสอบ (Testing) และหากมีปัญหา ก็ทำการแก้ไข (Debugging) จากนั้นจึงทำ Documentation เพื่อสร้างเอกสารอธิบายการทำงานของวงจรที่ออกแบบทั้งหมด

การออกแบบวงจรดิจิตอลด้วย FPGA หรือ CPLD นั้นจะเริ่มจากขั้นตอน System Implementation จนถึง Testing and debugging ซึ่งจำเป็นต้องใช้ซอฟต์แวร์ทุก (Software tool) ช่วยในการออกแบบ ซอฟต์แวร์ทุกของบริษัท Xilinx เวอร์ชันล่าสุด คือ ISE 10.1i ซึ่งหนังสือเล่มนี้จะใช้ ISE 8.1i เป็นหลักเนื่องจากการใช้งานต่างๆ เกือบทั้งหมดกับ ISE 10.1i แต่ ISE 10.1i จะรองรับ FPGA ตระกูลล่าสุด ได้แก่ Vertex-5 และ Spartan-3A เป็นต้น รวมทั้งคุณสมบัติใหม่ๆ บางประการที่เราเห็นจะไม่ได้ใช้ และ ISE 10.1i จะใช้หน่วยความจำบนฮาร์ดดิสก์มากกว่า ISE 8.1i ประมาณ 2 เท่า ISE 8.1i มี 2 ตัวเขียนกัน คือ ISE WebPack 8.1i (ไอเอสอีเว็บแพค 8.1i) หรือ WebPACK 8.1i (เป็น Limited version ที่อนุญาตให้ดาวน์โหลดฟรีทางอินเตอร์เน็ต) และ ISE Foundation 8.1i หรือ Foundation 8.1i (ต้องซื้อ) โดย ISE 8.1i สามารถ RUN ได้บนระบบปฏิบัติการ Microsoft Windows XP หรือ Microsoft Windows 2000 ISE 8.1i ใช้กับตระกูลชิปต่างๆ รวมทั้ง FPGA ตระกูล Vertex, Vertex-E และ Vertex-II สรุปได้ ดังตารางที่ 1.1 และขนาด RAM ของคอมพิวเตอร์ที่แนะนำให้ใช้กับ ISE 8.1i นั้นต้องไม่น้อยกว่า 256-512 MB ส่วน ISE10.1i มี 2 ตัวเขียนกัน คือ WebPACK 10.1i (เป็น Limited version ที่อนุญาตให้ดาวน์โหลดฟรีทางอินเตอร์เน็ต) และ Foundation 10.1i (ต้องซื้อ) WebPACK 10.1i สามารถ RUN บนระบบปฏิบัติการ Microsoft Windows XP Professional (32-bit) และ Microsoft Vista Business (32-bit) ได้ และขนาด RAM ของคอมพิวเตอร์ควรไม่น้อยกว่า 256-512 MB แต่อย่างไรก็ตามในทางปฏิบัติมักมีการเปิดใช้โปรแกรมอื่นรวมอยู่ด้วย ดังนั้นทั้ง ISE 8.1i และ ISE 10.1i ควรใช้ RAM ไม่น้อยกว่า 1-2 GB

ตารางที่ 1.1 ตระกูลของชิป FPGA และ CPLD ที่ซอฟต์แวร์ทุกร่องรับ

Family	ISE WebPack (MS Windows & Linux)	ISE Foundation (MS Windows, Linux, Linux64, & Solaris)
Virtex-4 devices	4VLX15, 4VLX25, 4VSX25, 4VFX12	All
QPro Virtex Hi-Rel devices	All	All
QPro VirtexE Hi-Rel devices	All	All
QPro Virtex2 Hi-Rel devices	All	All
QPro Virtex Rad-Hard devices	All	All
QPro Virtex2 Rad-Hard devices	All	All
Spartan-II devices	All	All
Spartan-III devices	All	All
Automotive Spartan-III devices	All	All
Spartan-3 devices	All	All
Automotive Spartan-3 devices	All	All
Spartan-3E devices	Up to 3S500E	All
XC9500/XL/XV devices	All	All
Automotive 9500XL devices	All	All
CoolRunner-II devices	All	All
Automotive CoolRunner-II devices	All	All
Device Programming Support	All	All

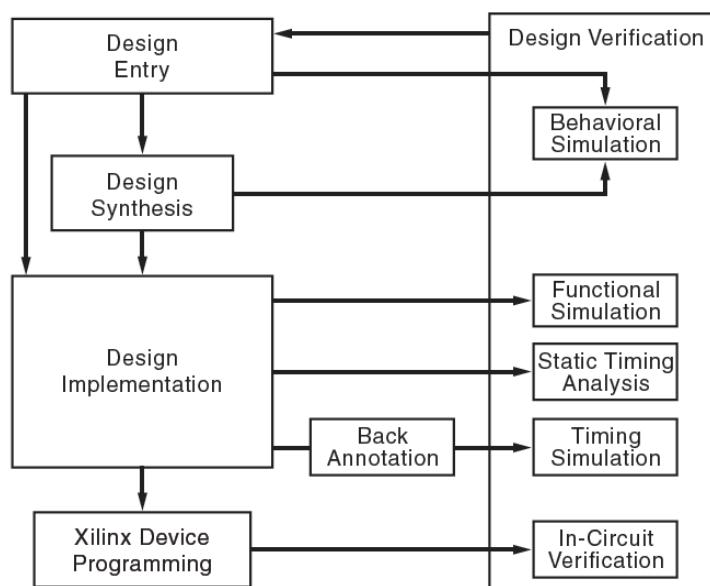
แม้ว่า WebPACK 8.1i จะเป็น Limited version ของ Foundation 8.1i ตามที่สรุปในตารางที่ 1.1 และตารางที่ 1.2 คือตาม แต่ถ้าเพียงพอสำหรับการออกแบบวงจรขนาดใหญ่ทั่วๆ ไป โดยมีความสามารถที่เพิ่มเข้ามาคือ ISE Simulator, CORE Generator และ FPGA editor

ตารางที่ 1.2 คุณสมบัติและความสามารถที่ซอฟต์แวร์ทูลรองรับ

Feature	ISE WebPACK (MS Windows & Linux)	ISE Foundation (MS WINDOWS, Linux, Linux64, & Solaris)
Schematic Editor	Yes	Yes
State Diagram Entry	MS Windows Only	MS Windows Only
XST Synthesis	Yes	Yes
Synplify/Synplify Pro Integration	Yes	Yes
Leonardo Spectrum Integration	Yes	Yes
Precision Integration	Yes	Yes
Waveform Editor	Yes	MS Windows & Linux Only
ISE Simulator	Limited version included	MS Windows & Linux Only Limited version included Unlimited version available as option
Modelsim Xilinx Edition	MS Windows only Starter included Full MXE available as option	MS Windows only Starter included Full MXE available as option
Modelsim Integration	Yes	Yes
CORE Generator	Yes	Yes
Floorplanner	Yes	Yes
FPGA Editor	Yes	Yes
Device Programming Support	Yes	Yes

1.6 ขั้นตอนการออกแบบวงจรดิจิทอลด้วย FPGA หรือ CPLD

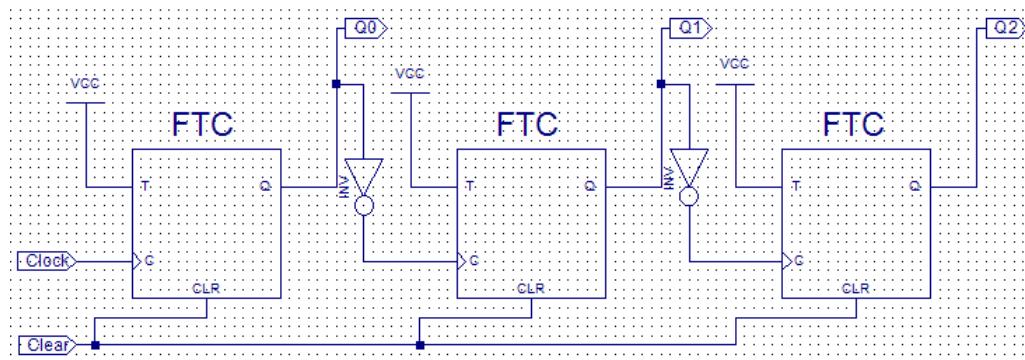
ขั้นตอนการออกแบบวงจรดิจิตอลด้วย FPGA หรือ CPLD มีการทำงานหลายขั้นตอน รายละเอียดแสดงดังรูปที่ 1.8



รูปที่ 1.8 ขั้นตอนการออกแบบวงจรดิจิตอลด้วย FPGA หรือ CPLD

1) การออกแบบวงจร

การออกแบบวงจร (Design entry) นั้นสามารถเลือกออกแบบด้วยวิธีการต่างๆ ได้แก่ วิธี Schematic หรือวิธีวาดผังวงจร ดังรูปที่ 1.9 ออกแบบด้วยภาษา硬件描述语言ที่เรียกว่า HDL (Hardware Description Language) เช่น VHDL (หรือ Verilog) แสดงดังรูปที่ 1.10 หรือ อาจออกแบบวงจรด้วยวิธีการใช้หน้าต่าง State diagram เพื่อให้ได้รับตามที่ต้องการ



รูปที่ 1.9 การออกแบบวงจรด้วยชีวิชผังวงจร (Schematic)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ADDER_2BIT is
7     port ( A,B : in STD_LOGIC_VECTOR (1 downto 0);
8             C : out STD_LOGIC_VECTOR (2 downto 0));
9 end ADDER_2BIT;
10
11 architecture BEHAVIORAL of ADDER_2BIT is
12 begin
13
14     C <= ('0' & A) + ('0' & B);
15
16 end BEHAVIORAL;

```

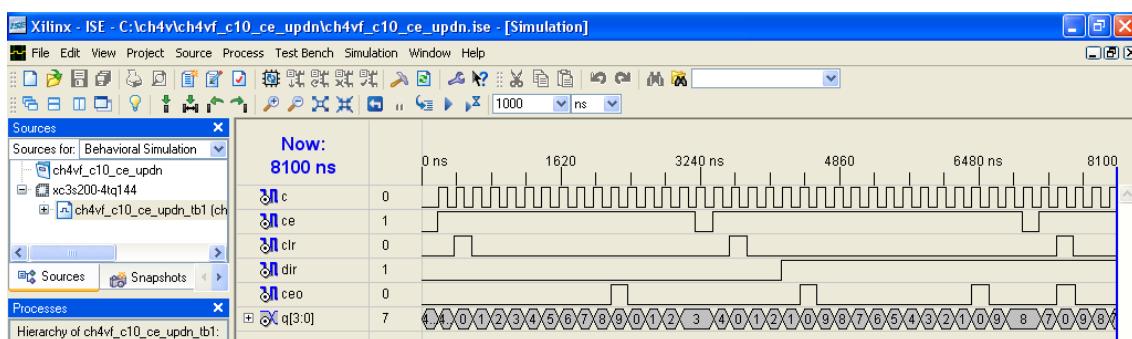
รูปที่ 1.10 การออกแบบวงจรด้วยภาษา VHDL

2) การสังเคราะห์วงจร

การสังเคราะห์วงจร (Design synthesis) คือ ขั้นตอนแปลงโค้ดภาษาระดับสูงให้เป็นระดับเกต (Gate level) ที่เขียนในรูป Text file หรือ Netlist file ส่วนการออกแบบโดยชีวิช Schematic ซึ่งเป็นระดับเกตอยู่แล้วก็จะเขียนในรูป Netlist file เข่นกันซึ่งถ้าใช้ Xilinx Synthesis Tool หรือ XST ก็จะได้ NGC file โดยจะประกอบด้วยไฟล์ Logical design data และ Constraints

3) การตรวจสอบความถูกต้องของวงจร

การตรวจสอบความถูกต้อง (Design verification) นั้นจะนำโค้ด HDL หรือวงจรที่ออกแบบไปตรวจสอบความถูกต้องด้วยการจำลองการทำงาน (Simulation) ได้แก่ Behavioral simulation และ Timing simulation ซึ่ง Timing simulation นี้จะคำนึงถึง Delay time ต่างๆ ที่เกิดขึ้นใน FPGA ตัวอย่างผลจำลองการทำงานแสดงดังรูปที่ 1.11 หนังสือเล่มนี้จะใช้ซอฟต์แวร์ทุล ISE Simulator ที่เดิมมาพร้อมกับ ISE 8.1i-10.1i ในการจำลองการทำงาน โดยในเบื้องต้นนี้จะใช้ Waveform Editor



รูปที่ 1.11 สัญญาณที่ได้จากการจำลองการทำงานของวงจรที่ออกแบบ

4) Design Implementation

Design implementation ของ FPGA นั้นจะแปลง Netlist file เป็น Physical file มีขั้นตอนดังนี้

- Translate เป็นขั้นตอนรวมไฟล์วงจรโลジคิกที่ออกแบบช่องอยู่ในรูปไฟล์ Netlists และ Constraints เข้าด้วยกัน
- Map เป็นขั้นตอนเลือกอุปกรณ์จากไฟล์ที่ได้จากขั้นตอน Translate หรือ NGD file ไปวางภายในหรือจับคู่กับอุปกรณ์พื้นฐานต่างๆ ที่มีอยู่ภายใน FPGA โดยจะเก็บอยู่ใน NCD file
- Place and Route เป็นขั้นตอนคำนวณหาตำแหน่งที่เหมาะสมเพื่อจะนำเอาวงจรโลจิกไปวาง (Place) โดยจะขึ้นอยู่กับ Timing constraints และเมื่อวางเรียบร้อยแล้วจึงเข้ามายังสัญญาณต่างๆ เข้าด้วยกัน (Route)
- Programming file generation เป็นขั้นตอนสร้างไฟล์ Bitstream เพื่อให้เหมาะสมสำหรับดาวน์โหลดลง FPGA

ส่วนกรณี CPLD นั้นมีความซับซ้อนน้อยกว่า FPGA มา ก็จะเรียกว่าขั้นตอนหลังจากการแปลง (Translate) ว่าขั้นตอน Fitting จากนั้นจะเป็นขั้นตอนสร้างไฟล์ Bitstream เพื่อสร้างไฟล์ให้เหมาะสมสำหรับการดาวน์โหลดลง CPLD

5) การโปรแกรมข้อมูลวงจรลงชิพ

การโปรแกรมข้อมูลวงจรลงชิพ (Device Programming) นั้นสามารถทำได้โดยการนำไฟล์ Bitstream ที่มีนามสกุล .bit (ไฟล์ข้อมูลวงจรของ FPGA) หรือไฟล์ที่มีนามสกุล .jed (ไฟล์ข้อมูลวงจรของ CPLD) ที่ได้ในขั้นตอน Design implementation (หรือขั้นตอน Generate Programming File) ไปดาวน์โหลดลงชิพโดยใช้เครื่องโปรแกรมหรือใช้ดาวน์โหลดเคเบิลก์ได้

1.7 การติดตั้งซอฟต์แวร์ทุก

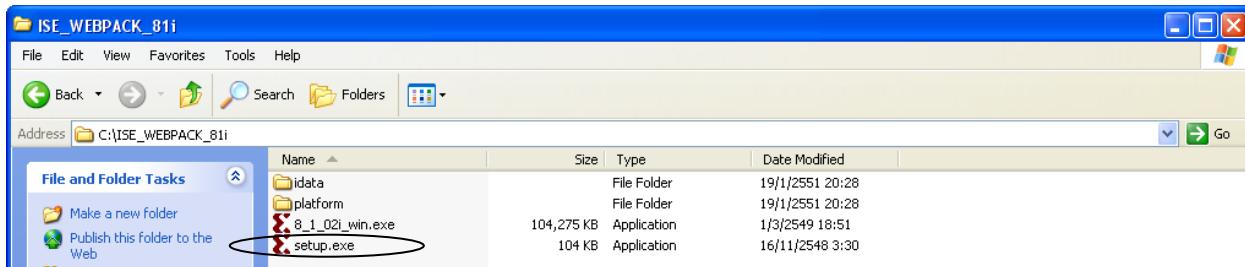
1) ขั้นตอนการขอและดาวน์โหลดซอฟต์แวร์

ในหนังสือเล่มนี้จะมีแผ่น DVD ให้ไปด้วย 1 แผ่น ซึ่งจะมีซอฟต์แวร์ทุก ISE WebPACK 8.1i และ Service Pack 2 (ชื่อ 8_1_02i_win.exe) อยู่ใน Folder ชื่อ C:\ISE_WebPACK_81i ส่วน Folder อื่นจะเป็นซอฟต์แวร์ทุก ISE WebPACK 10.1i และ Service Pack 3 อยู่ใน Folder ชื่อ C:\ISE_WebPACK_101i และ Folder ที่เป็นไฟล์การทดลองต่างๆ

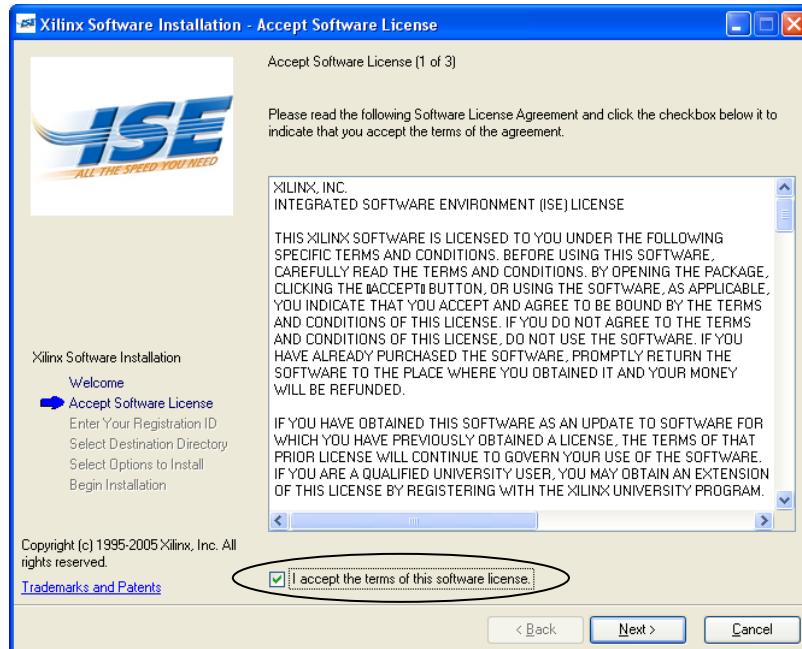
สำหรับซอฟต์แวร์ทุก ISE WebPACK 10.1i หรือเวอร์ชันล่าสุดสามารถดาวน์โหลดได้ฟรีที่ <http://www.xilinx.com> โดยจะต้องลงทะเบียนก่อน จากนั้นทาง Xilinx จะบันทึก User name และ Pass word โดยส่วนมากอีเมล์ แล้วนำไป Sign in เพื่อดาวน์โหลดซอฟต์แวร์ทุก ถ้ามีไฟล์ ISE WebPACK 10.1i แล้วก็ເອົາເພີ່ມ Registration ID ไปใช້ເພີ່ມຕົດຕັ້ງซอฟต์แวร์ทุก

2) วิธีการติดตั้ง ISE WebPACK 8.1i

- 1) ทำการ Uninstall ซอฟต์แวร์เวอร์ชันก่อน (ถ้าเคยติดตั้งมาแล้ว) ที่จะ จากนั้น Copy ไฟล์ใน Folder ชื่อ ISE_WebPACK_81i จากแผ่น DVD ไปไว้ในไดร์ฟ C เมื่อค้นเบิลคลิก Folder ชื่อ ISE_WebPACK_81i (ในไดร์ฟ C) แล้วจะได้ดังรูปที่ 1.12
- 2) ดับเบิลคลิกที่ไฟล์ setup.exe แล้วจะได้หน้าต่าง Accept Software License ให้คลิก “” ที่หน้า “I accept the term of this software license” ดังรูปที่ 1.13 แล้วคลิกปุ่ม Next แล้วจะได้หน้าต่างถัดไป (ทำซ้ำจนครบ 3 ครั้ง) เสร็จแล้วคลิกปุ่ม Next ไปอีก 2 ครั้งและคลิกปุ่ม Install แล้วโปรแกรมจะเริ่มติดตั้งไฟล์ต่างๆ ลงในคอมพิวเตอร์ ดังรูปที่ 1.14



รูปที่ 1.12 ไฟล์ ISE WebPACK 8.1i (ที่เด็กไฟล์แล้ว) และ Service Pack 2 (ชื่อ 8_1_02i_win.exe)

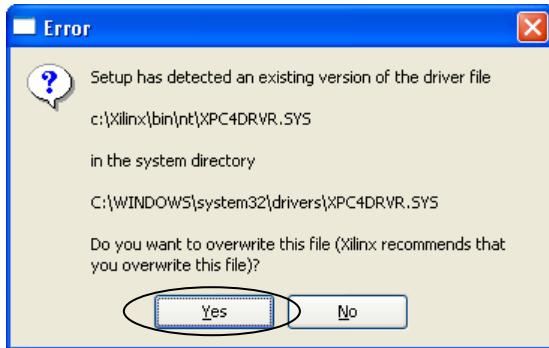


รูปที่ 1.13 หน้าต่าง Accept Software License (มี 3 หน้า)



รูปที่ 1.14 เริ่มติดตั้งซอฟต์แวร์

3) ในกรณีที่คอมพิวเตอร์เคยติดตั้งซอฟต์แวร์เวอร์ชันก่อนหน้านี้ในระหว่างติดตั้งอาจมีหน้าต่าง Error ปรากฏขึ้นดังรูปที่ 1.15 ให้กด Yes และจะได้หน้าต่างถัดไปเพื่อทำการ Setup เมื่อ Setup แล้วเสร็จให้กด OK ก็จะถือว่าติดตั้งซอฟต์แวร์แล้วเสร็จ



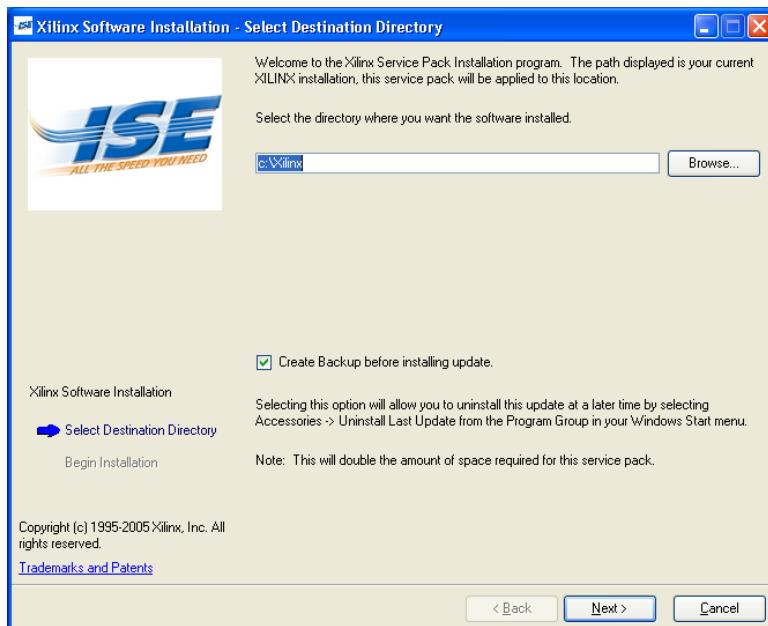
รูปที่ 1.15 ข้อเตือนไฟล์ Driver ของซอฟต์แวร์เวอร์ชั่นเก่า

3) วิธีการติดตั้ง Service Pack 2

3.1) ในรูปที่ 1.12 จะมีไฟล์ Service Pack 2 (ชื่อ 8_1_02i_win.exe) อยู่ จากนั้นดับเบิลคลิกไฟล์ 8_1_02i_win.exe แล้วจะได้ดังรูปที่ 1.16 แล้วโปรแกรมจะทำการแตกไฟล์ .exe ออกโดยอัตโนมัติ เมื่อแตกไฟล์เสร็จแล้วจะได้ดังรูปที่ 1.17



รูปที่ 1.16 โปรแกรมจะแตกไฟล์ 8_1_02i_win.exe



รูปที่ 1.17 เลือก Directory และ Create Backup before installing update

3.2) คลิก Next ในรูปที่ 1.17 เพื่อเริ่มแบ็กอัพไฟล์ก่อนทำการติดตั้ง เมื่อบนเเบ็คอัพไฟล์เสร็จแล้วให้คลิก OK จากนั้นคลิก Install เพื่อเริ่มติดตั้งซอฟต์แวร์ดังรูปที่ 1.18 ไปจนแล้วเสร็จ คลิก OK ก็จะถือว่าการติดตั้งซอฟต์แวร์แล้วเสร็จสมบูรณ์ **หมายเหตุ** การใช้ซอฟต์แวร์ ISE WebPACK 8.1i นั้นมีความจำเป็นต้องติดตั้งซอฟต์แวร์ชื่อ Adobe Reader 7 ขึ้นไปควบคู่กันไปด้วยเพื่อใช้งานหรืออ่านไฟล์ .pdf ดังนั้นหากยังไม่ได้ติดตั้งก็ให้ผู้ใช้จะติดตั้งซอฟต์แวร์ชื่อ Adobe Reader นี้ด้วย

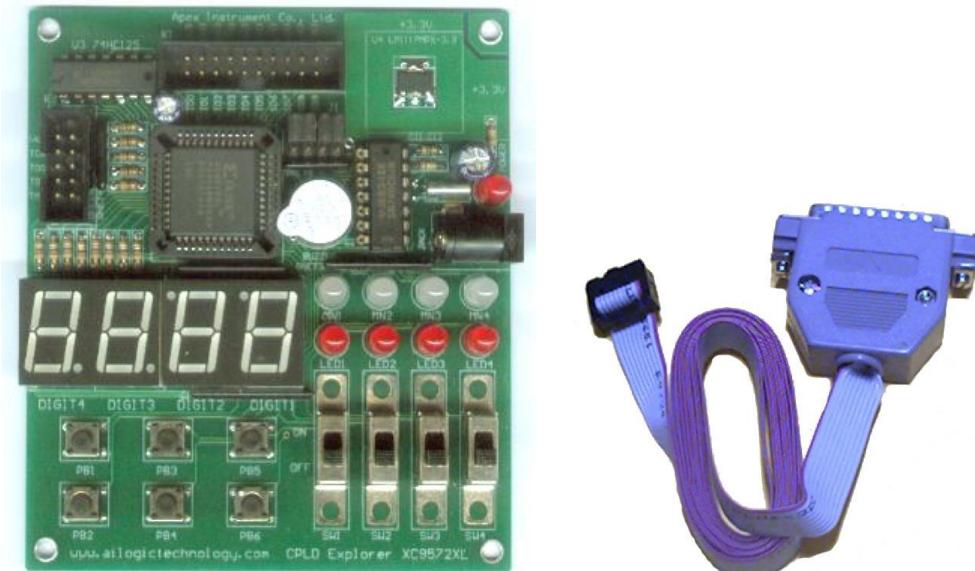


รูปที่ 1.18 หน้าต่าง Begin Installation ขณะกำลังติดตั้ง

1.8 บอร์ดทดลอง

1.8.1 CPLD Explorer XC9572XL

บอร์ดทดลองonenกประสงค์รุ่น CPLD Explorer XC9572XL เป็นบอร์ดทดลองที่เหมาะสมสำหรับผู้เริ่มต้นศึกษาการออกแบบวงจรดิจิตอตโนมัติ FPGA และ CPLD มีรายละเอียดแสดงดังรูปที่ 1.19 คุณสมบัติทั่วๆไปของบอร์ดทดลองเป็นดังนี้



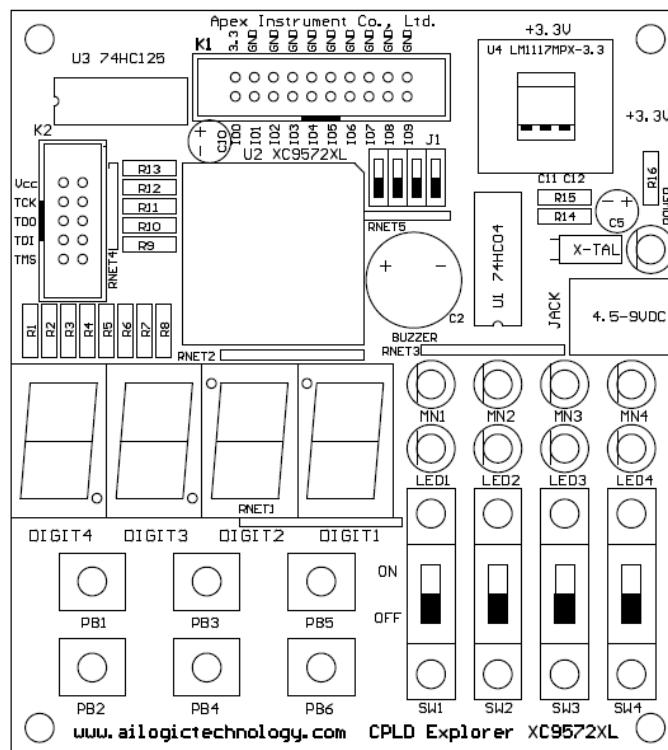
รูปที่ 1.19 CPLD Explorer XC9572XL พร้อมสายดาวน์โหลด

1) คุณสมบัติทั่วไป

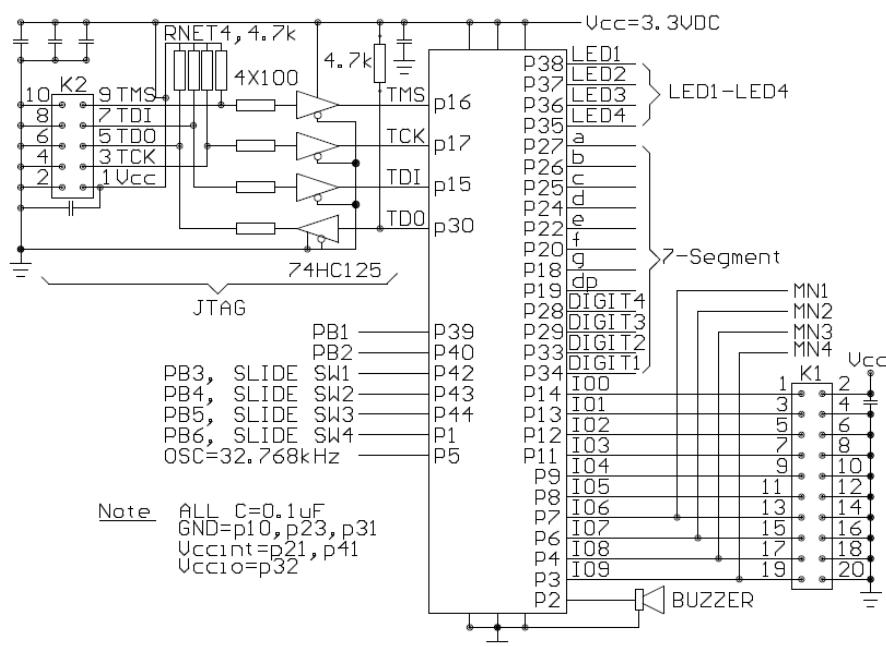
- CPLD เบอร์ XC9572XL (1,600 เกต) แบบ PLCC 44 ขา (PC44)1-Speed Grade 0
- เซรัวนเซกเมนต์ จำนวน 4 หลัก
- LED แสดงผล 2 สถานะ 4 ดวง และ Logic monitor ที่เป็น LED แสดงผล 3 สถานะ 4 ดวง (ใช้ร่วมกับ K1)
- ออค (Buzzer) จำนวน 1 ตัว
- Push button switch 6 ตัว และ Slide switch 4 ตัว (ใช้ร่วมกับ Push button switch)
- พอร์ต K1 เป็น I/O 10 บิต ซึ่งสามารถใช้กับ I/O 5 V.(ตรรกะ TTL , 74HCT00 ,...) และ 3.3 V.ได้
- Onboard Oscillator 32.768 kHz

2) หลักการทำงานของบอร์ด

รายละเอียดการจัดวางอุปกรณ์บนบอร์ดเป็นดังรูปที่ 1.20 โดยที่การจัดวาง I/O ต่างๆ แสดงดังรูปที่ 1.21 และตารางที่ 1.3



รูปที่ 1.20 การจัดวางอุปกรณ์บน CPLD Explorer XC9572XL



รูปที่ 1.21 การจัดวาง I/O ต่างๆของ CPLD Explorer XC9572XL

ตารางที่ 1.3 แสดงตำแหน่งขาของชิป CPLD ที่ต่ออยู่กับสาร์ดแวร์ภายนอกที่อยู่บนบอร์ด

LED	
I/O Name	CPLD Pin No.
LED1 (L1)	p38
LED2 (L2)	p37
LED3 (L3)	p36
LED4 (L4)	p35

Push Button Switch	
I/O Name	CPLD Pin No.
PB1	p39
PB2	p40
PB3 (SW1)	p42
PB4 (SW2)	p43
PB5 (SW3)	p44
PB6 (SW4)	p1

Slide Switch	
I/O Name	CPLD Pin No.
SW1 (PB3)	p42
SW2 (PB4)	p43
SW3 (PB5)	p44
SW4 (PB6)	p1

Misc.	
I/O Name	CPLD Pin No.
Buzzer	p2
OSC =32.768kHz	p5

7 Segment		Remark
I/O Name	CPLD Pin No.	
a	p27	
b	p26	
c	p25	
d	p24	
e	p22	
f	p20	
g	p18	
dp	p19	
DIGIT4 (SEG1)	p28	Common Cathode
DIGIT3 (SEG2)	p29	Common Cathode
DIGIT2 (SEG3)	p33	Common Cathode
DIGIT1 (SEG4)	p34	Common Cathode

I/O ของ K1	Remark	
I/O Name	CPLD Pin No.	
IO0	p14	Pin no.1 of K1
IO1	p13	Pin no.3 of K1
IO2	p12	Pin no.5 of K1
IO3	p11	Pin no.7 of K1
IO4	p9	Pin no.9 of K1
IO5	p8	Pin no.11 of K1
IO6, MN1	p7	Pin no.13 of K1, JUMPER, J1-1
IO7, MN2	p6	Pin no.15 of K1, JUMPER, J1-2
IO8, MN3	p4	Pin no.17 of K1, JUMPER, J1-3
IO9, MN4	p3	Pin no.19 of K1, JUMPER, J4-4

ทางด้านเอาต์พุตของบอร์ดทดลองเป็นดังนี้

1) LED แบบ 2 สถานะ 4 ดวง คือ LED1-LED4 โดยต่อขา cathode (Cathode) ลงกราวด์และต่อขาแอนโอด (Anode) เข้ากับ I/O ของ CPLD โดยมีตัวด้านทาน RNET2 470 โอห์มต่ออนุกรมอยู่เพื่อจำกัดกระแสและแสดงดังรูปที่ 1.22(a) ดังนี้ถ้า CPLD ส่งลอจิก ‘1’ จะทำให้ LED ดวงนั้นติดสว่าง

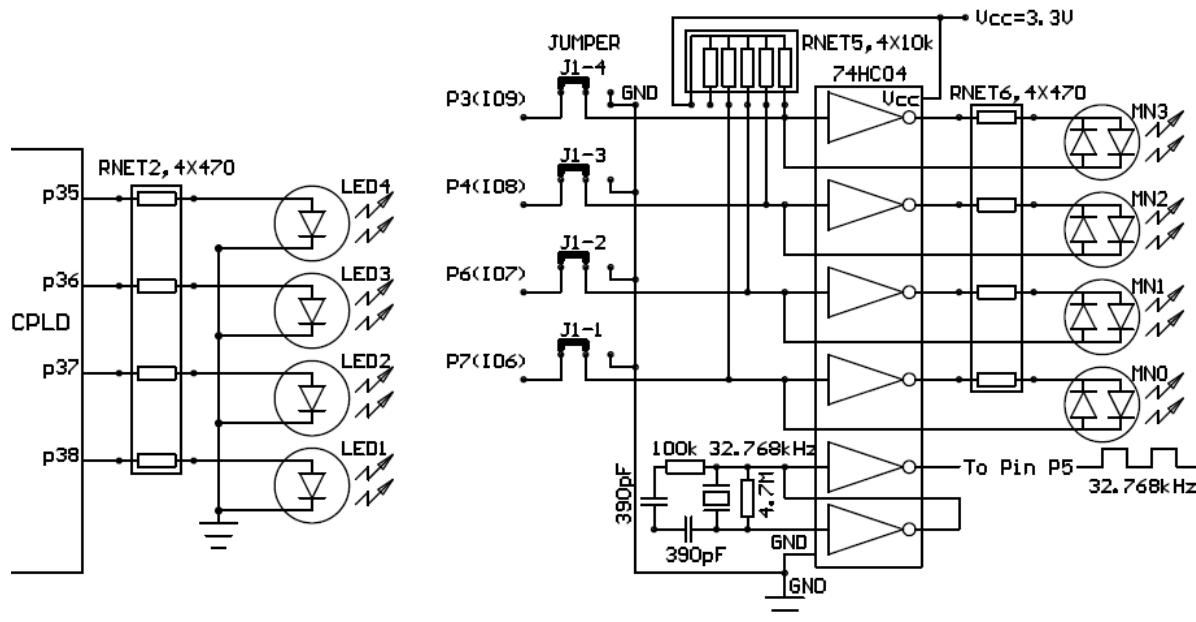
2) Logic monitor เป็น LED และแสดงผล 3 สถานะ 4 ดวง คือ MN-1MN 4(J1-1 ถึง J1-4) ต่อพ่วงกับ I/O ของ CPLD ที่คอนเนกเตอร์ K1 (I/O6-I/O9) โดยใช้ Jumper J1 ดังรูปที่ 1.22(b) ถ้า I/O ของ CPLD เป็นลอจิก ‘1’ LED ติดเป็นสีเหลือง แต่ถ้าลอจิก ‘0’ จะติดเป็นสีแดง แต่ถ้าเป็น High impedance ก็จะดับ (ดับเกือบสนิทเนื่องจากมี RNET5 10 kΩ ให้อห์มต่อพูลอัพเข้ากับ Vcc)

3) Onboard Oscillator (OSC) = 32.768 kHz ใช้เป็นสัญญาณนาฬิกาประจำบอร์ด รายละเอียดของแสดงดังรูปที่ 1.22(b)

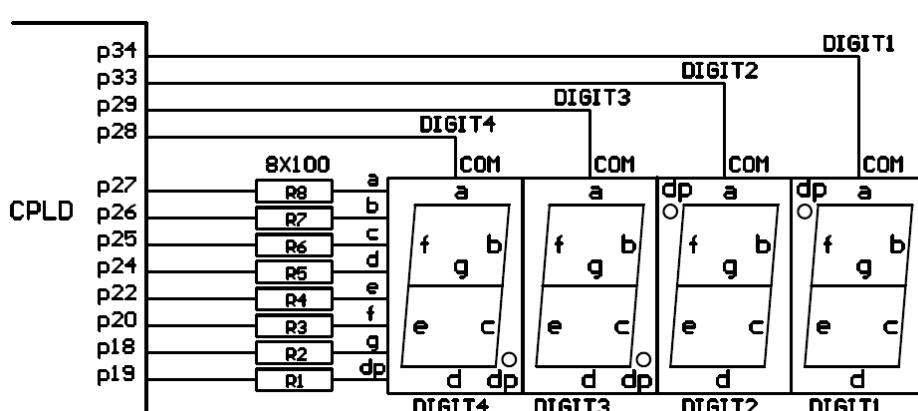
4) ออก (Buzzer) จำนวน 1 ตัว ซึ่งต่อขาบลนgross แดตต่อขาอีกข้างซึ่งเป็นขั้วบวกกับขาอินพุตเอาต์พุต (I/O) p2 ของ CPLD ดังนี้ถ้า CPLD ส่งลอจิก ‘1’ จะทำให้ออดดับกีส์ส่งเป็นลอจิก ‘0’

5) ตัวแสดงผลเซเว่นเซกเมนต์ (7-Segment) มี 4 หลัก คือ DIGIT4-DIGIT1 โดยทุกหลักจะใช้สายสัญญาณร่วมกัน แต่สาย cathode (Common) แต่ละหลักจะแยกออกจากกันดังรูปที่ 1.23 การแสดงตัวเลขพร้อมกันทั้ง 4 หลักจึงใช้เทคนิคการสแกน โดยมีหลักการ คือ เมื่อส่งตัวเลขไปที่หลักแรกแล้วส่งลอจิก ‘0’ ที่ขา cathode ร่วมของหลักแรกแล้วจึงทำที่หลักต่อๆไปจนครบทั้ง 4 หลักจากนั้นจึงวนกลับเพื่อทำซ้ำ ปกติตามเราระมองแยกได้ที่ประมาณ 30 ครั้งต่อวินาที จึงต้องสแกนด้วยความเร็ว

ໄມ່ນ້ອຍກວ່າ 30 ຄຽ້ງ \times 4 ລັກ = ຄວາມດີ 120 Hz ຈຶ່ງຈະໄມ່ເກີນກາຮຽນພຣິບ ຕັ້ງຕ້ານທານ R1-R8 ໃຊ້ໃນກາຈຳດກຣະແສ ສ່ວນທີ່
ຕັ້ງແສດງພລ DIGIT2 ແລະ DIGIT1 ຈະມີກາຮອກແນບກັບຫ້າໃຫ້ຈຸດ (dp) ຂຶ້ນໄປອູ່ດໍານັນເພື່ອກາແສດງເຄື່ອງໝາຍ (noloC) ”：“
ໃນກາທຳນາພິກາຮູ້ອ່າແສດງພລອງຄາຂອງອຸນຫະນີ ແຕ່ກາແສດງຕົວເລເດືອນ ຂັງໃຊ້ສາຍສັງພາຜີເບັກັນ DIGIT3 ແລະ DIGIT4



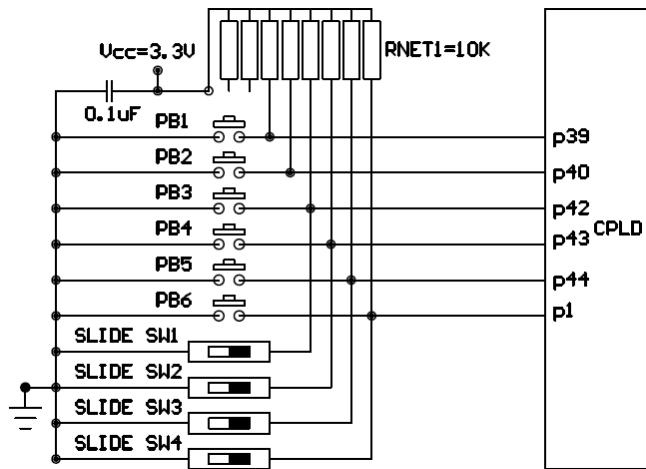
ຮູບທີ່ 1.22 ກາຮອກຕໍ່ LED ເຂົ້າກັບຫາຂອງ CPLD, ວັຈຮັບ Logic trainer ແລະ 32.768 kHz Oscillator



ຮູບທີ່ 1.23 ຕັ້ງແສດງພລເຂວານເຊກເມນຕໍ່ (Segment-7) ຈຳນວນ 4 ລັກ

ດ້ານອິນພຸດຂອງນອർດທດລອງເປັນດັ່ງນີ້

- 1) ສວິຕົກຕົກປົກລ່ອຍດັບ (Push button Switch) ອູ້ 6 ຕັ້ງກື່ອ PB1-PB6 ຕ່ອອູ້ກັບຫາຂອງ CPLD ແລະ ມີສ່ໄລດ໌ສວິຕົກ (Slide Switch) ອືກ 4 ຕັ້ງ ກື່ອ Slide SW1-Slide SW4 ຕ່ອພ່ວງອູ້ກັບ PB3-PB6 ຕາມລຳດັບແສດງດັງຮູບທີ່ 1.24 ຜ່ອກາໄມ່ກົດສວິຕົກຈະໄກ້
ລອຈິກ ‘1’ ແຕ່ຄ້າກຈະໄຫ້ລອຈິກ ‘0’ ເນື່ອຈາກມີຕັ້ງຕ້ານທານພຸລອັພ RNET1 ຕ່ອອູ້ ແລະ ອາກເລື່ອນສ່ໄລດ໌ສວິຕົກໄປທີ່ OFF ຈະໄກ້
ລອຈິກ ‘1’ ແຕ່ຄ້າ ON ຈະໄກ້ລອຈິກ ‘0’ ກາຮົດ PB3 ອັງໄວ້ກຈະເໝືອນກັບກາຮົດເລື່ອນສ່ໄລດ໌ສວິຕົກ SW1 ຂຶ້ນໄປທີ່ ON

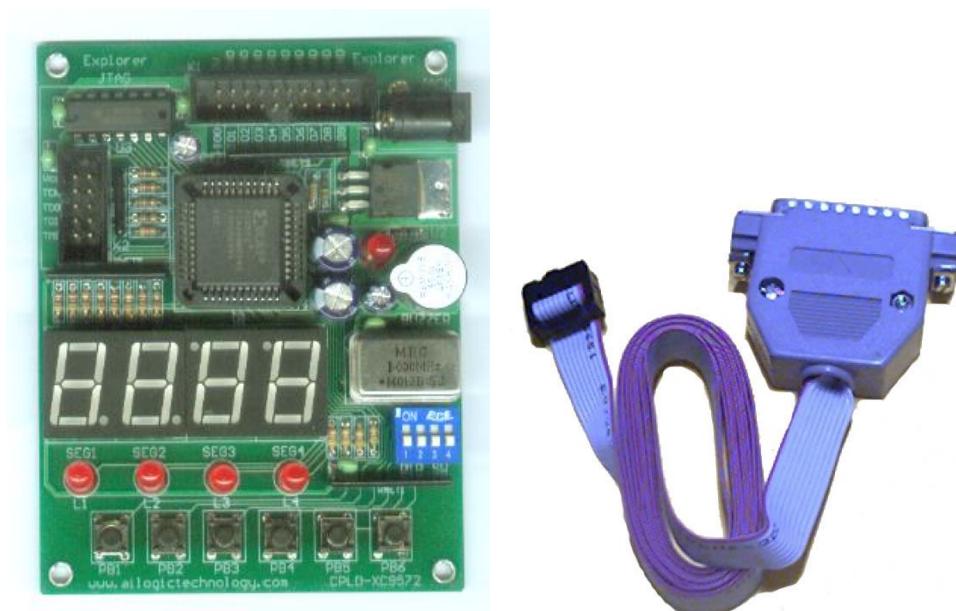


รูปที่ 1.24 ถ่วงชักกดติดปล่อยดับและสไลด์สวิตช์

2) JTAG คอนเนกเตอร์ K2 ใช้เพื่อโปรแกรมข้อมูลวงจร (Configuration data) ลง CPLD ผ่านทางพอร์ตขนาน (Parallel Port) ของคอมพิวเตอร์ โดยขาสัญญาณทุกเส้นจะต่อผ่านบaffleอร์เบอร์ 74HC125 เพื่อบังกันการรบกวนในสาย JTAG ส่วน R10-R13 ต่อໄว์เพื่อลดสัญญาณสะท้อนในสาย JTAG และก่อนโปรแกรมวงจรลงชิปจะต้องมีการกำหนดขา CPLD เป็นตามตารางที่ 1.3 จากนั้นต่อสาย JTAG และไฟเลี้ยงเข้าบอร์ด ในการที่ใช้บอร์ดทดลองนี้เป็นเครื่องโปรแกรมนั้นจะใช้ได้กับเบอร์ XC9536XL และ XC9572XL ที่มีขาแบบ PLCC 44 ขา

1.8.2 CPLD Explorer XC9572

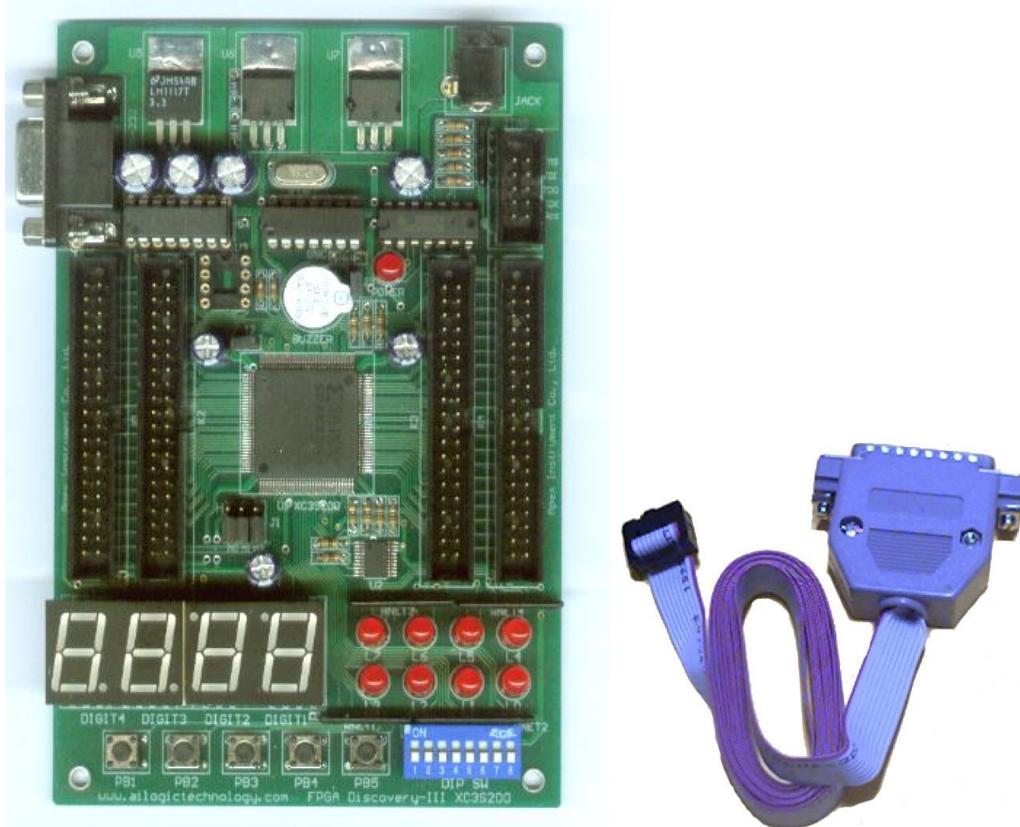
บอร์ดทดลองonenกประสงค์ CPLD Explorer XC9572 แสดงดังรูปที่ 1.25 เป็นบอร์ดทดลองที่มีคุณสมบัติทั่วๆไป เช่นกับบอร์ด CPLD Explorer XC9572XL และมีรายการ I/O ตรงกันทุกประการตามตารางที่ 1.3 แต่จะแตกต่างตรงที่จะไม่มี Logic monitor 4 ดวงและมี I/O เป็นระบบ 5V ใช้ DIP Switch แทน Slide Switch และใช้ Oscillator ความถี่ 1 MHz แทน 32.768 kHz และสามารถใช้เป็นเครื่องโปรแกรม CPLD ได้เช่นเดียวกัน คือ เบอร์ XC9536 และ XC9572 แบบ PLCC 44 ขา



รูปที่ 1.25 บอร์ดทดลองonenกประสงค์ CPLD Explorer XC9572

1.8.3 FPGA Discovery-III XC3S200

บอร์ดทดลองonenกประสงค์รุ่น FPGA Discovery-III XC3S200 มีรายละเอียดแสดงดังรูปที่ 1.26 มีความจุวงจรสูง 200,000-400,000 เกตและใช้ Platform Flash PROM สำหรับเก็บข้อมูลวงจร สามารถโปรแกรมวงจรลง Platform Flash PROM ผ่านทางสายดาวน์โหลดแบบ JTAG ได้โดยตรง บอร์ดมีอุปกรณ์อำนวยความสะดวกที่เพียงพร้อมด้วยอุปกรณ์อินพุตเอาต์พุตอย่างครบครัน จึงเหมาะสมสำหรับ LAB ออกแบบวงจรดิจิตอลและออกแบบไอซีขั้นสูง



รูปที่ 1.26 บอร์ดทดลองonenกประสงค์ FPGA Discovery-III XC3S200

1) คุณสมบัติทั่วไป

บอร์ดทดลองonenกประสงค์ FPGA Discovery-III XC3S200 มี 2 รุ่นด้วยกัน คือ รุ่น FPGA Discovery-III XC3S200F และ FPGA Discovery-III XC3S200F4 ซึ่งมีลักษณะทั่วไปเหมือนกันทุกประการ แต่จะแตกต่างกันคือ FPGA Discovery-III XC3S200F จะใช้ FPGA ตระกูล Spartan-3 ของ Xilinx เบอร์ XC3S200-4TQ144C ซึ่งเป็น FPGA ขนาดความจุวงจร 200,000 เกต, Package แบบ TQ144, Speed Grade:4 และ Platform Flash PROM เบอร์ XCF01SVO20C (ที่สามารถโปรแกรมข้อมูลวงจรช้าได้ถึง 20,000 ครั้ง) ในขณะที่ FPGA Discovery-III XC3S200F4 จะใช้ชิปเบอร์ XC3S400-4TQ144C ซึ่งเป็น FPGA ขนาดความจุวงจร 400,000 เกต Package แบบ TQ144, Speed Grade:44 และ Platform Flash PROM เบอร์ XCF02SVO20C บอร์ดทดลอง FPGA Discovery-III XC3S200 มีคุณสมบัติต่างๆ เป็นดังนี้

- 7-Segment จำนวน 4 หลัก (ใช้ร่วมกับ Expansion ports และสามารถกดออกได้)
- LED จำนวน 8 ดวง (ใช้ร่วมกับ Expansion ports สามารถแยกออกจาก I/O ได้โดยหักเอา RNET3 และ RNET4 ออก)
- Buzzer จำนวน 1 ตัว (ใช้ร่วมกับ Expansion ports)
- DIP Switch 8 ปีต

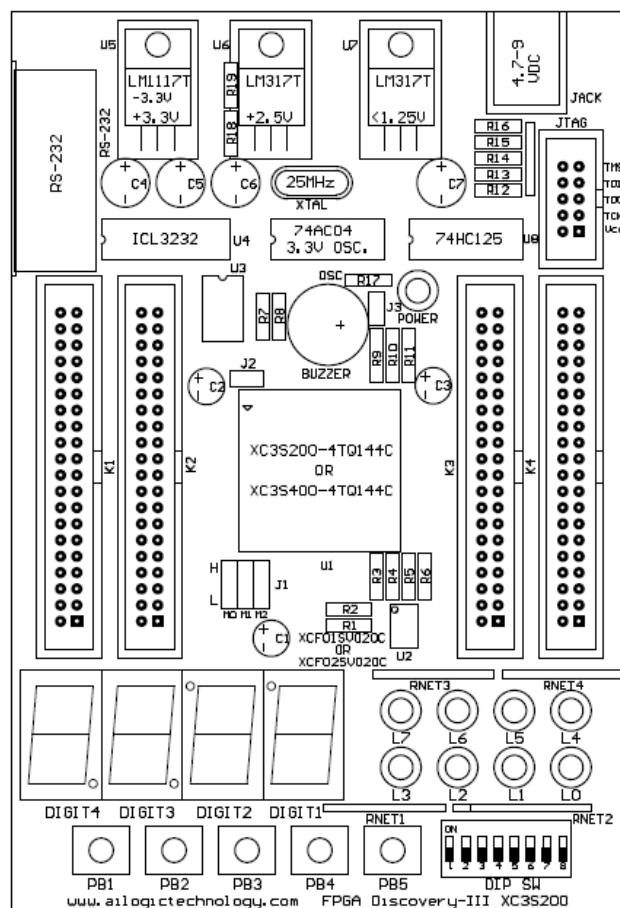
- Push Button Switch จำนวน 5 ตัว
- Expansion ports (80 Bits 3.3V. I/O)
- RS-232C Port 1 Port (ใช้ร่วมกับ Expansion ports)
- I²C Socket สำหรับ EEPROM (ใช้ร่วมกับ Expansion ports)
- 25 MHz Oscillator (สามารถโปรแกรมเป็นความถี่อื่นๆ ได้โดยใช้ Digital Frequency Synthesizer มีที่อยู่ใน FPGA)

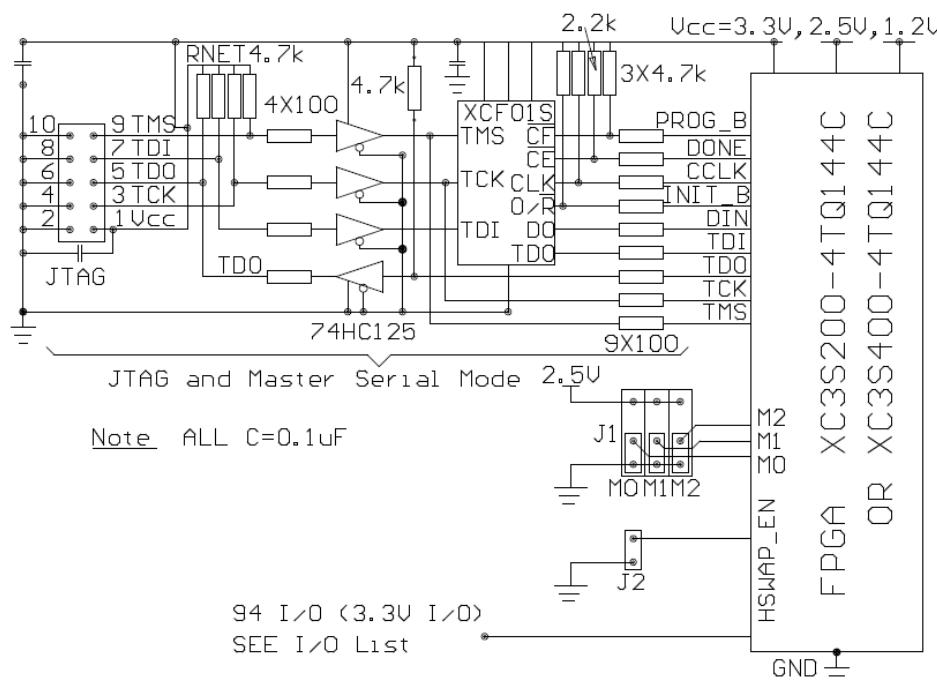
2) คุณสมบัติที่สำคัญของชิป FPGA ตระกูล Spartan-3 เบอร์ XC3S200

- ความจุวงจร 200,000 เกต
- 18 Kb block RAM จำนวน 12 ชุด (รวม 216 Kb)
- 18x18 hardware multiplier จำนวน 12 ชุด
- Digital Clock Manager (DCM) จำนวน 4 ชุด

3) หลักการทำงานของบอร์ดอเนกประสงค์

บอร์ดทดลองอเนกประสงค์ FPGA Discovery-III XC3S 200 มีการจัดวางอุปกรณ์ดังรูปที่ 1.27 รายละเอียดการจัดวาง I/O (โดยย่อ) แสดงดังรูปที่ 1.28 และอุปกรณ์ที่ต่ออยู่กับขา FPGA (I/O List) ตามตารางที่ 1.4 หลักการทำงานของ 7-Segment, LED, Buzzer, DIP Switch และ Push Button Switch จะทำงานเช่นเดียวกับบอร์ดทดลอง CPLD Explorer XC9572XL





รูปที่ 1.28 การจัดวาง I/O ต่างๆของ FPGA Discovery-III XC3S200 (โดยย่อ) และวงจรสำหรับดาวน์โหลด

ตารางที่ 1.4 รายละเอียดของอุปกรณ์ที่ต้องอยู่กับขา FPGA (I/O List)

Segment-7	Pinout FPGA	Descriptions
a	p40	a
b	p35	b
c	p32	c
d	p30	d
e	p27	e
f	p25	f
g	p23	g
dp	p20	Decimal Point
DIGIT1	p31	CATHODE DIGIT1 , COMMON
DIGIT2	p33	DIGIT2 , COMMON CATHODE
DIGIT3	p36	DIGIT3 , COMMON CATHODE
DIGIT4	p41	DIGIT4 , COMMON CATHODE

Pushbutton	FPGA Pinout	Descriptions
PB1	p44	PushbuttonNo. 1
PB2	p46	PushbuttonNo. 2
PB3	p47	PushbuttonNo. 3
PB4	p50	PushbuttonNo. 4
PB5	p51	PushbuttonNo. 5

EEPROM	FPGA Pinout	Descriptions
SCL-I2C	p128	LCXX24
SDA-I2C	p129	LCXX24

232RS	FPGA Pinout	Descriptions
TX	p131	ICLCP3232
RX	p132	ICLCP3232

LED	FPGA Pinout	Descriptions
L0	p70	L0
L1	p77	L1
L2	p69	L2
L3	p76	L3
L4	p74	L4
L5	p79	L5
L6	p73	L6
L7	p78	L7

Dip SW	FPGA Pinout	Description
1	p52	Dip Switch No.1
2	p53	Dip Switch No.2
3	p55	Dip Switch No.3
4	p56	Dip Switch No.4
5	p59	Dip Switch No.5
6	p60	Dip Switch No.6
7	p63	Dip Switch No.7
8	p68	Dip Switch No.8

Oscillator	FPGA Pinout	Descriptions
OSC	p127	MHz , GCLK625

BUZZER	FPGA Pinout	Descriptions
BUZZER	p125	BUZZER

ตารางที่ 1.4 รายละเอียดของอุปกรณ์ที่ต่ออยู่กับขา FPGA (I/O List) (ต่อ)

K1 CONNECTOR					
Descriptions	FPGA Pinout	K1 Pinout	K1 Pinout	FPGA Pinout	Descriptions
GND	-	40	39	p128	I/O , I2C-SCL
GND	-	38	37	p130	I/O
GND	-	36	35	p132	I/O , RS232-RX
GND	-	34	33	p137	I/O
GND	-	32	31	p141	I/O
GND	-	30	29	p2	I/O
GND	-	28	27	p5	I/O
GND	-	26	25	p7	I/O
GND	-	24	23	p10	I/O
GND	-	22	21	p12	I/O
GND	-	20	19	p14	I/O
GND	-	18	17	p17	I/O
GND	-	16	15	p20	I/O,dp-7 Segment
GND	-	14	13	p23	I/O , g-7 Segment
GND	-	12	11	p25	I/O , f-7 Segment
GND	-	10	9	p27	I/O , e-7 Segment
GND	-	8	7	p30	I/O , d-7 Segment
GND	-	6	5	p32	I/O , c-7 Segment
GND	-	4	3	p35	I/O , b-7 Segment
+3.3V,Vcc	-	2	1	p40	I/O , a-7 Segment

ตารางที่ 1.4 รายละเอียดของอุปกรณ์ที่ต่ออยู่กับขา FPGA (I/O List) (ต่อ)

K2 CONNECTOR					
Descriptions	FPGA Pinout	K2 Pinout	K2 Pinout	FPGA Pinout	Descriptions
GND	-	40	39	p129	I/O , I2C-SDA
GND	-	38	37	p131	I/O , RS232-TX
GND	-	36	35	p135	I/O
GND	-	34	33	p140	I/O
GND	-	32	31	p1	I/O
GND	-	30	29	p4	I/O
GND	-	28	27	p6	I/O
GND	-	26	25	p8	I/O
GND	-	24	23	p11	I/O
GND	-	22	21	p13	I/O
GND	-	20	19	p15	I/O
GND	-	18	17	p18	I/O
GND	-	16	15	p21	I/O
GND	-	14	13	p24	I/O
GND	-	12	11	p26	I/O
GND	-	10	9	p28	I/O
GND	-	8	7	p31	I/O , DIGIT1
GND	-	6	5	p33	I/O , DIGIT2
GND	-	4	3	p36	I/O , DIGIT3
+3.3V, Vcc	-	2	1	p41	I/O , DIGIT4

ตารางที่ 1.4 รายละเอียดของอุปกรณ์ที่ต่ออยู่กับขา FPGA (I/O List) (ต่อ)

K3 CONNECTOR					
Descriptions	FPGA Pinout	K3 Pinout	K3 Pinout	FPGA Pinout	Descriptions
I/O , BUZZER	p124	40	39	-	GND
I/O	p122	38	37	-	GND
I/O	p118	36	35	-	GND
I/O	p113	34	33	-	GND
I/O	p108	32	31	-	GND
I/O	p105	30	29	-	GND
I/O	p103	28	27	-	GND
I/O	p100	26	25	-	GND
I/O	p98	24	23	-	GND
I/O	P96	22	21	-	GND
I/O	p93	20	19	-	GND
I/O	p90	18	17	-	GND
I/O	p87	16	15	-	GND
I/O	p85	14	13	-	GND
I/O	p83	12	11	-	GND
I/O	p80	10	9	-	GND
I/O , L7	p78	8	7	-	GND
I/O , L3	p76	6	5	-	GND
I/O , L6	p73	4	3	-	GND
I/O , L2	p69	2	1	-	+3.3 V. Vcc

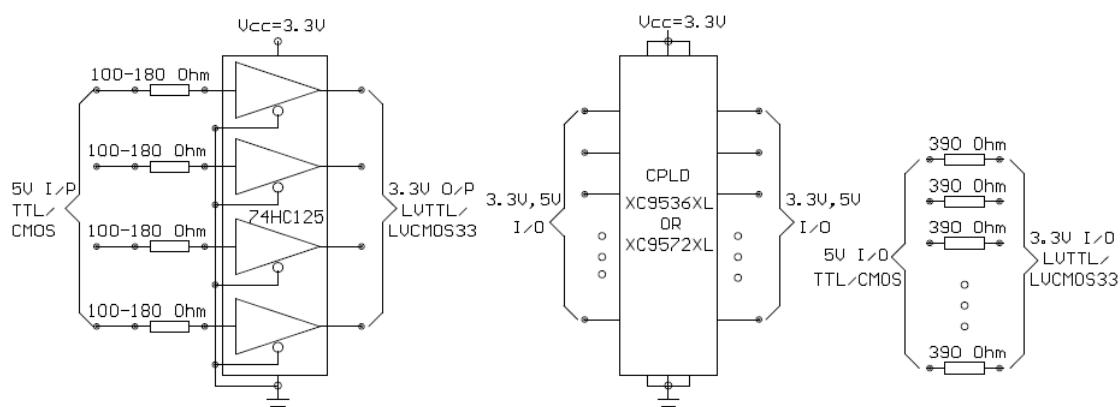
ตารางที่ 1.4 รายละเอียดของอุปกรณ์ที่ต่ออยู่กับขา FPGA (I/O List) (ต่อ)

K4 CONNECTOR					
Descriptions	FPGA Pinout	K4 Pinout	K4 Pinout	FPGA Pinout	Descriptions
I/O	p125	40	39	-	GND
I/O	p123	38	37	-	GND
I/O	p119	36	35	-	GND
I/O	p116	34	33	-	GND
I/O	p112	32	31	-	GND
I/O	p107	30	29	-	GND
I/O	p104	28	27	-	GND
I/O	p102	26	25	-	GND
I/O	p99	24	23	-	GND
I/O	p97	22	21	-	GND
I/O	p95	20	19	-	GND
I/O	p92	18	17	-	GND
I/O	p89	16	15	-	GND
I/O	p86	14	13	-	GND
I/O	p84	12	11	-	GND
I/O	p82	10	9	-	GND
I/O , L5	p79	8	7	-	GND
I/O , L1	p77	6	5	-	GND
I/O , L4	p74	4	3	-	GND
I/O , L0	p70	2	1	-	+3.3V. Vcc

เนื่องจากไฟเลี้ยง V_{cc} ของ I/O FPGA เป็น 3.3V จึงสามารถเลือก I/O ได้เพียง 2 ชนิด คือ LVCMOS33 หรือ LVTTL ดังตารางที่ 1.5 ซึ่งเป็น I/O ระบบ 3.3V ดังนั้นถ้า I/O ของ FPGA เป็นเอาต์พุตที่จะสามารถขับอินพุตของระบบ 3.3V และระบบ 5V ได้ (เช่น ตรรคุล 74LS00, 74HCT) แต่ถ้า I/O ของ FPGA เป็นอินพุตที่จะใช้ได้เฉพาะระบบ 3.3V เท่านั้น ตัวอย่างการเชื่อมต่อระบบ 5V เข้ากับอินพุตของ FPGA แสดงดังรูปที่ 1.29(a) ซึ่งรูปที่ 1.29(c) จะใช้ความต้านทานประมาณ 390 โอห์มเพื่อจำกัดกระแสไฟ流เข้าอินพุตของ FPGA ซึ่งปกติไม่เกิน 5-10 mA ตัวต้านทานที่ใช้มีค่าอินดักแนนซ์ แฟรงก์ไตน์ R $\pm 5\%$ 1/8W ถ้าความต้านทานต่ำเกินไปจะทำให้อินพุตเสียหายได้ (ดูรายละเอียดจาก Data sheet ของ FPGA)

ตารางที่ 1.5 Single-ended I/O standards (Values in volts)

Signal Standard	V _{cc}	
	For Outputs	For Inputs
LVCMOS12	1.2	1.2
LVCMOS15	1.5	1.5
LVCMOS18	1.8	1.8
LVCMOS25	2.5	2.5
LVCMOS33	3.3	3.3
LVTTL	3.3	3.3



(a) ต่อผ่านบัฟเฟอร์ 3.3V

(b) ต่อผ่าน CPLD ที่มี I/O 3.3V

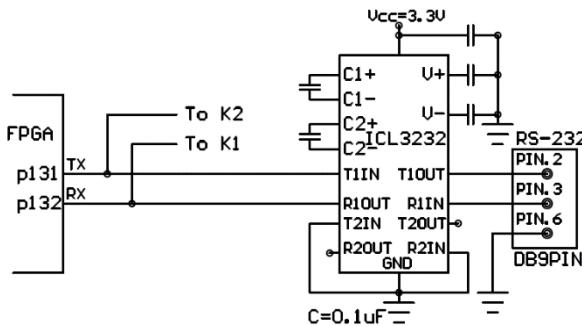
(c) ต่อผ่านตัวต้านทานอนุกรม

รูปที่ 1.29 ตัวอย่างการต่อเอาต์พุต 5 V กับอินพุตของ FPGA

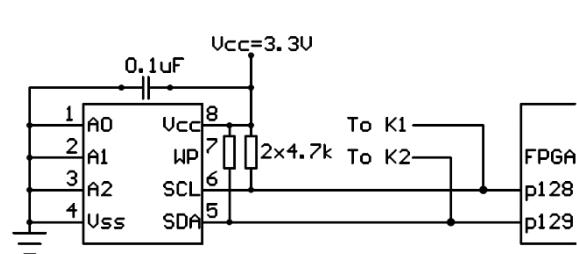
เนื่องจากคอนเนกเตอร์ K1-K4 ถูกออกแบบให้มีสายสัญญาณและกราวด์อย่างละครึ่งเพื่อแก้ปัญหาการรบกวนข้ามช่องสัญญาณ (Cross talk) ดังนั้นการต่อสายสัญญาณ I/O ออกจากคอนเนกเตอร์ K1-K4 นั้นถ้าต้องการใช้งานที่ความถี่สูงๆ ควรใช้สายแพร์ (Flat Cable) โดยที่ความยาวสูงสุด (ของเส้นลากทองแดงของ PCB และสายแพร์รวมกัน) ที่ไม่เกิดการสะท้อน (หรือ Transmission Line effect) $< (2''/\text{nS}) \times \text{ช่วงเวลาขึ้น} (\text{Rise time})$ เมื่อ O/P ของ FPGA เป็น Fast slew rate มี Rise time $< 1 \text{ nS}$ และ Slow slew rate จะมี Rise time $< 3 \text{ nS}$ ดังนั้น $(2''/\text{nS}) \times 3 \text{ nS} = 6''$ (ประมาณ 15 เซนติเมตร) ถ้าที่สายแพร์ยาวกว่านี้จะต้อง Terminate ด้วยวิธีที่เหมาะสม การต่อ埠อร์ดทดลองเข้ากับ埠อร์ดภายนอกนั้นถ้าสายแพร่มี V_{cc} รวมอยู่ด้วยจะต้องต่อตัวเก็บประจุ 0.1 uF และ 10 nF แบบมัลติเดเยอร์หรือ Chip capacitor เข้ากับขั้ว V_{cc} และกราวด์ (กราวด์เพลน) ที่บอร์ดนั้นด้วยเพื่อให้สาย V_{cc} มีคุณสมบัติทางไฟฟ้าแบบ AC เสมือนว่าเป็นกราวด์ ถ้าเกิดอสซิลเลตในสายของ I/O ก็ใช้ตัวต้านทานค่า 56-220 โอห์มต่ออนุกรมเข้ากับ I/O เพื่อหน่วงอสซิลเลต ถ้าค่าความต้านทานสูงเกินไปจะทำให้วงจรทำงานช้าลง

พอร์ตอนุกรม (RS-232C) ใช้ไอซีเบอร์ ICL3232CP (U4) ซึ่งเป็นระบบ 3.3V เพื่อให้สามารถต่อ กับ I/O ของ FPGA ได้โดยตรง โดยมีวงจรแสดงดังรูปที่ 1.30 และเนื่องจาก I/O ของพอร์ตอนุกรมต่อพ่วงอยู่กับขาที่ 35 ของ K1 และขาที่ 39 ของ K2 ดังนั้นถ้าต้องการใช้ขาตักกล่าวของ K1 และ K2 เพื่อวัดคุณสมบัติที่สามารถทำได้โดยการอ่านไอซี ICL3232CP ออกได้

ชุดคีย์ต 8 ขา (U3) สำหรับใส่ไอซี EEPROM แบบ I²C เช่น เบอร์ 24LC256 เพื่อเก็บข้อมูลนั้นจะต่อพ่วงกับขาที่ 39 ของ K1 และ K2 ตามลำดับ แสดงดังรูปที่ 1.31 มีตัวต้านทาน R7 และ R8 ขนาด 4.7 กิโลโอด์มต่อ Pull up



รูปที่ 1.30 พอร์ตอนุกรม (RS-232C)

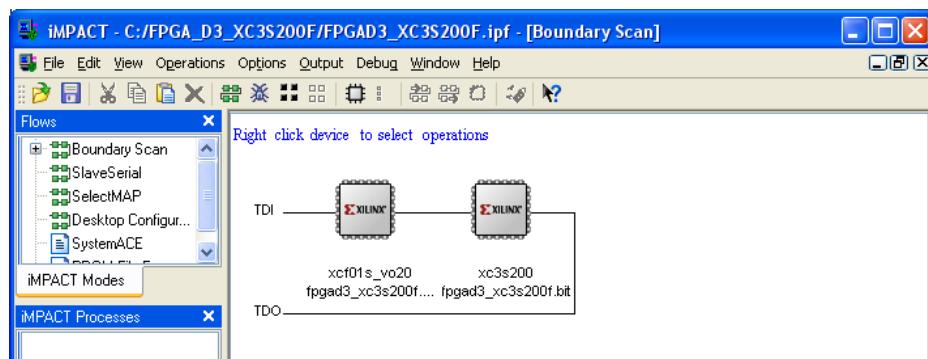


รูปที่ 1.31 พอร์ตอนุกรมแบบ I²C

JTAG ค่อนนคเตอร์ใช้สำหรับต่อสายดาวน์โหลด (JTAG Cable (เข้ากับพอร์ตบนาน)Printer Port(ของคอมพิวเตอร์ เพื่อโปรแกรมข้อมูลวงจร) Configuration data (ลง FPGA มีรายละเอียดวงจรดังรูปที่ 1.28 ซึ่งได้ถูกออกแบบให้โปรแกรม Serial Flash PROM และ FPGA ได้ทั้งใน Master Serial Mode และ JTAG Mode (Boundary Scan Mode) ซึ่งในขั้นตอนดาวน์โหลดนี้ที่จะคอมพิวเตอร์จะประมวลผล Platform Flash PROM และ FPGA พร้อมกันแสดงดังรูปที่ 1.32 เพราะมีการออกแบบให้โหมด JTAG ต่อถึงกันและวนแบบลูปโซ่ เราจึงสามารถดาวน์โหลดข้อมูลลงชิปทั้งสองตัวหรือตัวใดตัวหนึ่งก็ได้ และเนื่องจากในตารางที่ 1.6 นั้น Jumper J1 ถูกเซตใน Master Serial Mode คือ M0, M1, M2 = "000" ดังนั้นเพื่อป้องกันการโปรแกรม FPGA ข้อผิดพลาดเนื่องจากการโปรแกรมผิดโหมด (JTAG) ทาง Xilinx แนะนำใช้ไฟล์ใน Serial Flash PROM ทั้งก่อนทุกครั้ง หากไฟล์ใน Serial Flash PROM เป็นคนละไฟล์ (ไฟล์ไม่เหมือนกัน) กับไฟล์ที่จะโปรแกรมลง FPGA

ตารางที่ 1.6 รายการเซตโหมดในการโปรแกรม FPGA

Configuration Mode	M0	M1	M2
Master Serial	0	0	0
Slave Serial	1	1	1
JTAG	1	0	1



รูปที่ 1.32 ขั้นตอนดาวน์โหลดที่จะคอมพิวเตอร์จะประมวลผล Flash PROM และ FPGA พร้อมกัน

1.8.4 บอร์ดทดลองรุ่นอื่นๆ

สำหรับบอร์ดทดลองรุ่นอื่นๆ นั้นผู้ใช้ควรจะต้องศึกษาข้อมูลจากคู่มือของบอร์ดทดลองรุ่นนั้นๆ การออกแบบวงจรและการโปรแกรมจะต้องออกแบบลงบนบอร์ดทดลองจะมีลักษณะเหมือนทุกประการกัน

สรุป

ในอดีตการออกแบบวงจรดิจิตอลจะใช้ไอซีสำเร็จรูปที่เรียกว่าทีพีเออลในการต่อวงจร แต่ในปัจจุบันมีเทคโนโลยีที่เรียกว่า FPGA และ CPLD มาช่วยให้การออกแบบง่ายขึ้นกว่าเดิม โดยโครงสร้างภายในของ CPLD จะประกอบไปด้วย Function Block และ I/O Block ที่สามารถเชื่อมต่อถึงกันได้ด้วย Switch matrix ซึ่งภายใน Function Block มีลอกิพื้นฐานที่สามารถโปรแกรมได้ทำให้ CPLD สามารถถูกโปรแกรมให้เป็นวงจรดิจิตอลต่างๆ ตามที่ผู้ใช้ต้องการได้ ในขณะที่โครงสร้างของ FPGA จะประกอบไปด้วย Configurable Logic Block และ I/O Logic ที่สามารถโปรแกรมให้เป็นลอกิพื้นฐานต่างๆ ได้ตามต้องการ เช่นเดียวกัน แต่ FPGA มีความซุ้งมากกว่า CPLD มาก ทำให้สามารถออกแบบวงจรที่มีขนาดใหญ่มากๆ ได้

ปกติแล้วการแก้ปัญหางานทางด้านดิจิตอลส่วนใหญ่มีการใช้งานไมโครคอนโทรลเลอร์กันอย่างกว้างขวาง ปัจจุบันจึงมีการออกแบบไมโครคอนโทรลเลอร์เอาไว้ใน FPGA เพื่อให้ระบบทั้งหมดสามารถทำงานอยู่บนไอซีเดียวได้ โดยกระบวนการออกแบบวงจรดิจิตอลโดยใช้ FPGA และ CPLD จะประกอบไปด้วย

1. การออกแบบวงจร
2. การสังเคราะห์วงจร
3. การตรวจสอบความถูกต้องของวงจร
4. Design implementation
5. การโปรแกรมข้อมูลลงชิป

คำถามท้ายบท

1. โครงสร้าง FPGA และ CPLD แตกต่างกันอย่างไร
2. ขั้นตอนการออกแบบวงจร(Design Entry) มีวิธีใดบ้าง
3. ขั้นตอนการทำ Design implementation ของ FPGA และ CPLD แตกต่างกันอย่างไร
4. บอร์ดทดลองรุ่นใดที่สามารถนำมาสร้างเป็นนาฬิกาปลุกได้ เนื่องจากมีตัวแสดงผลเจ็ดส่วน สวิทซ์อินพุท และ Buzzer เพียงพอต่อการใช้งาน
5. โดยทั่วไป FPGA หรือ CPLD ที่มีความซุ้งมากกว่ากัน เพราะเหตุใด
6. การใช้งาน FPGA หรือ CPLD แทนการออกแบบแบบเดิมที่ใช้ไอซีทีพีเออลมีข้อดีข้อเสียอย่างไร
7. การโหลดวงจรหรือ Design ลง FPGA และ CPLD โดยตรง วิธีที่หากตัดไฟเลี้ยง FPGA หรือ CPLD แล้ว ทำให้วงจรที่โหลดลงไปสูญหาย
8. หากต้องการให้วงจรที่โหลดลง FPGA และ CPLD คงอยู่ถึงแม้จะตัดไฟเลี้ยงแล้วก็ตาม จะต้องทำอย่างไร

== หน้าร่อง ==

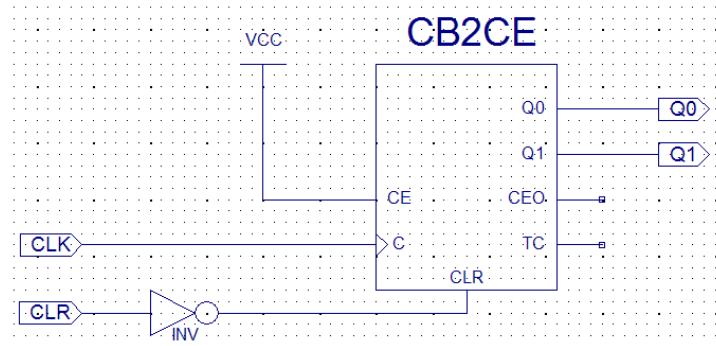
บทที่ 2 ออกแบบบอร์ดดิจิตอลด้วย FPGA และ CPLD โดยวิธี Schematic

กล่าวนำ

ในบทนี้จะอธิบายการออกแบบบอร์ดดิจิตอลด้วย FPGA และ CPLD โดยวิธี Schematic หรือวิธีการดูวงจร ซึ่งเป็นการออกแบบโดยใช้อุปกรณ์แวร์ทุก ISE WebPACK 8.1i หรือ ISE WebPACK 10.1i (สำหรับ Microsoft Vista Business 32-bit) โดยจะใช้โปรแกรม Schematic editor ในการวาดผังวงจร โดยนำเอาวงจรดิจิตอลย่อๆ ในรูป Symbols ที่ถูกจัดเตรียมไว้แล้วในไลบรารี (Library) เช่น เกตต่างๆ ฟลิปฟล็อป และ วงจรนับ เป็นต้น ไปรวมกันเพื่อให้ได้วงจรที่มีขนาดใหญ่ ซึ่งคล้ายกับการออกแบบวงจรโดยใช้ไอซีมาตรฐานตระกูล TTL หรือ CMOS หลายๆ ตัวมาต่อรวมกันบนแผงวงจร ดังนั้นผู้ที่เคยออกแบบด้วยไอซีมาตรฐานอยู่แล้วก็จะออกแบบวงจรโดยวิธี Schematic ได้ทันทีโดยไม่จำเป็นต้องเรียนรู้ภาษาระดับสูง เช่น ภาษา VHDL และภาษา Verilog เป็นต้น แต่ยังไรมีความรู้จากไฟล์วงจรดิจิตอลที่เขียนด้วยภาษาระดับสูงนั้นสามารถดาวน์โหลดได้ทางอินเตอร์เน็ตหรือมีในหนังสือทั่วไป ดังนั้นหากมีความรู้ VHDL หรือ Verilog ในเบื้องต้นเราก็สามารถนำไฟล์เหล่านั้นไปใช้โดยสร้าง Symbols ได้เท่านั้น ซึ่งจะลดความยุ่งยากในการวาดผังวงจรลงได้มาก ในการวาดผังวงจรนั้นแนะนำให้ตั้งความละเอียดของภาพ (Screen resolution) ที่ 1024 x 768 pixels ตัวอย่างในข้อ 2.1 และข้อ 2.2 เป็นการออกแบบวงจรนับ 4 แบบชิ้งโครงสร้างโดยใช้ CPLD และ FPGA ตามลำดับ โดยมีขั้นตอนการออกแบบเกือบจะเหมือนกันทุกประการ ยกเว้นขั้นตอนโปรแกรมลง FPGA และ Flash PROM เท่านั้น การใช้งาน ISE WebPACK 8.1i และ ISE WebPACK 10.1i จะมีลักษณะคล้ายๆ กัน

2.1 การออกแบบวงจรนับ 4 แบบชิ้งโครงสร้างด้วย CPLD โดยวิธี Schematic

ตัวอย่างที่ 2.1 ออกแบบวงจรนับ 4 แบบชิ้งโครงสร้างโดยใช้ CPLD และดังรูปที่ 2.1 (ไฟล์ตัวอย่างที่ 2 บทที่ 3 และบทที่ 4 อยู่ในแผ่น DVD หากต้องการใช้ ให้ Copy Folder "ไฟล์ต่างๆ" ไปไว้ในไดร์ฟ C โดยตรง) ขั้นตอนการออกแบบเป็นดังนี้

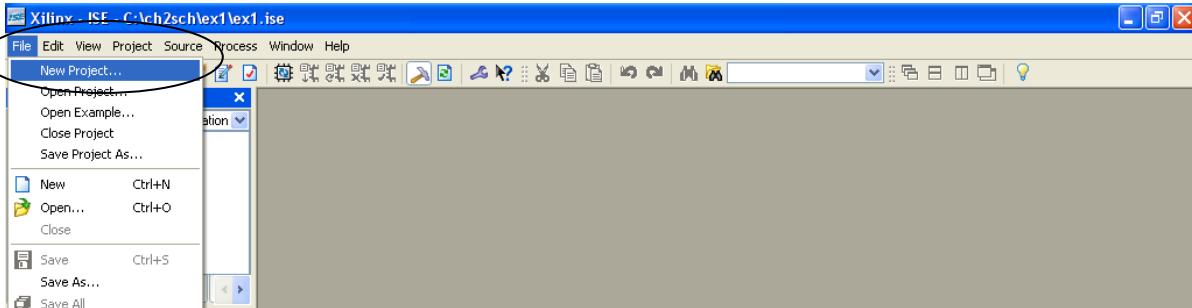


รูปที่ 2.1 วงจรนับ 4 แบบชิ้งโครงสร้าง

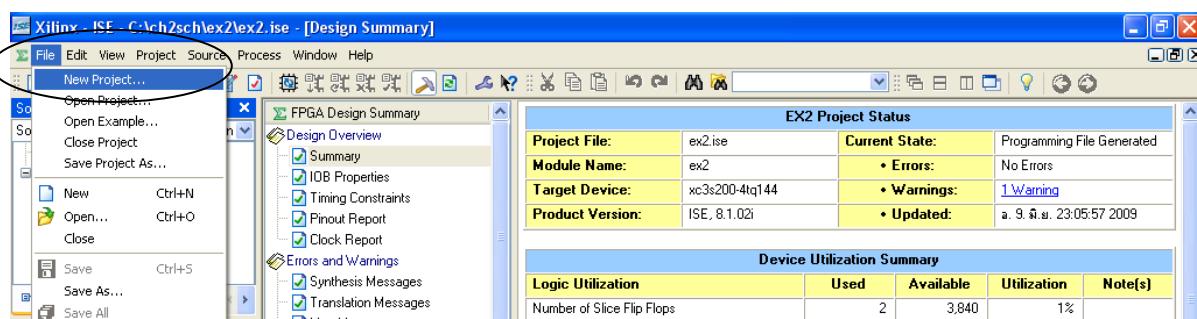
2.1.1 ขั้นตอนการออกแบบ (Design Entry)

- ก่อนจะใช้โปรแกรม ISE WebPACK 8.1i ให้สร้าง Folder ใหม่ชื่อ ch2sch (หรือชื่ออื่นๆ ก็ได้)ไว้ในไดร์ฟ C เสร็จแล้วจึงเริ่มการใช้งาน ISE WebPACK 8.1i โดยที่หน้าจอคอมพิวเตอร์ คลิกปุ่ม Start -> Programs -> Xilinx ISE 8.1i -> Project Navigator หรือค้นเบิลคลิกที่ แล้วจะได้หน้าต่าง Xilinx-ISE (ถ้ามีหน้าต่าง Tip of the Day ข้อนี้มาให้คลิกที่ปุ่ม OK) จากนั้นคลิก File -> New Project ดังรูปที่ 2.2 แล้วจะได้หน้าต่าง (หรือ Dialog box) New Project Wizard

2) ที่หน้าต่าง New Project Wizard>Create New Project พิมพ์ชื่อ C:\ch2sch ในช่อง Project Location (ชื่อ Folder) และลากจิ้งพิมพ์ชื่อ ex1 ที่ช่อง Project Name คลิกที่ Top-Level Source Type เป็น Schematic ดังรูปที่ 2.3 คลิก Next !!แล้วจะได้หน้าต่างดังไป

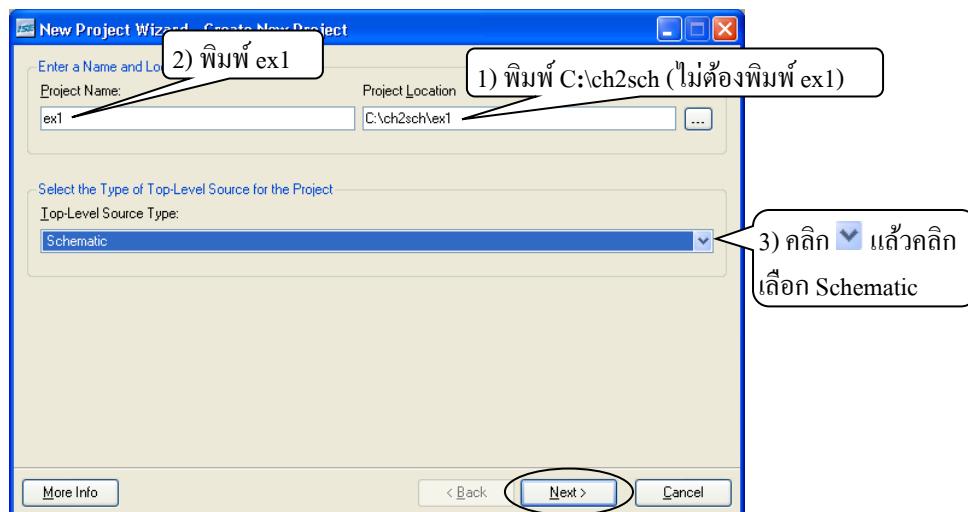


(a) หน้าต่าง Xilinx-ISE



(b) หน้าต่าง Xilinx-ISE

รูปที่ 2.2 หน้าต่าง Xilinx-ISE (ความละเอียดของภาพ 1024 x 768 pixels) อาจจะเป็นดังรูป 2.2(a) หรือรูปที่ 2.2(b)

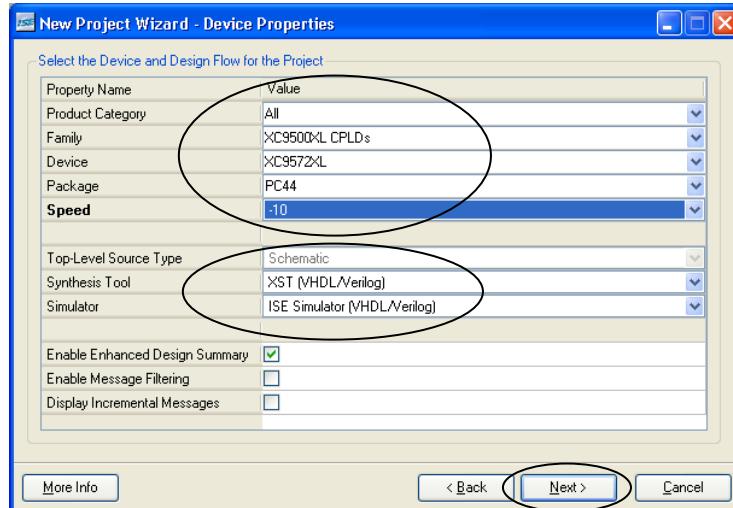


รูปที่ 2.3 หน้าต่าง New Project Wizard>Create New Project

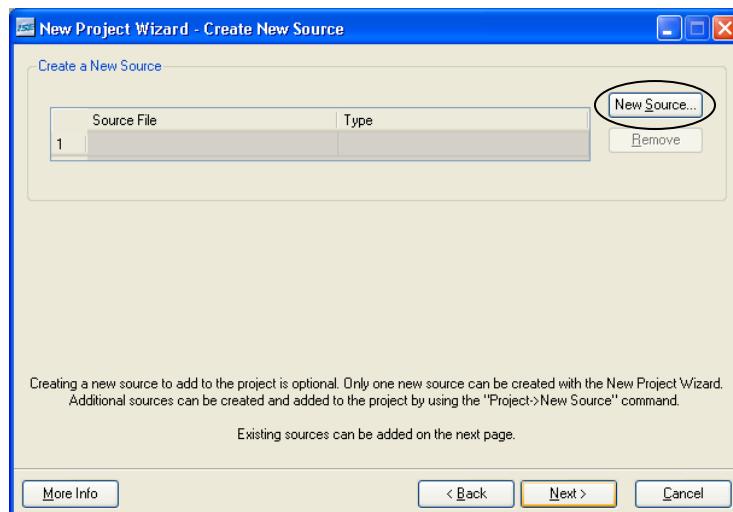
หมายเหตุ การตั้งชื่อไฟล์ต่างๆ หรือ Project Location หรือชื่อ Folder นั้นจะต้องขึ้นต้นด้วย A-Z, a-z และตัวถัดไปอาจจะเป็น A-Z, a-z, 0-9 หรือ Underscores (_) แต่ห้ามจบด้วย Underscores และห้ามเว้นช่องว่างระหว่างตัวอักษร

3) ที่หน้าต่าง New Project Wizard-Device Properties คลิกเลือกดังรูปที่ 2.4 ซึ่งจะใช้ CPLD เบอร์ XC9572XL-10PC44C ที่ใช้บนบอร์ด CPLD Explorer XC9572XL ดังนั้นต้องเลือกตระกูล (Family) XC9500XL เบอร์ (Device) XC9572XL ที่มี Package แบบ PLCC 44 ขา (Package : PC44) และ Speed Grade : -10 คลิกเลือกที่ Synthesis Tool เป็น XST (VHDL/Verilog) และ

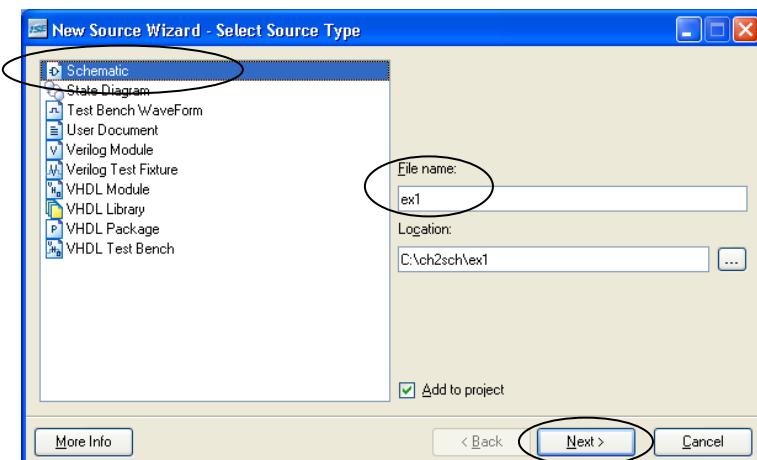
Simulator เป็น ISE Simulator (VHDL/Verilog) คลิก Next แล้วจะได้หน้าต่างดังรูปที่ 2.5 จากนั้นคลิกปุ่ม New Source แล้วจะได้หน้าต่างถัดไป พิมพ์ชื่อ ex1 ลงในช่อง File Name แล้วคลิกที่ Schematic ในรูปที่ 2.6



รูปที่ 2.4 หน้าต่าง New Project Wizard–Device Properties

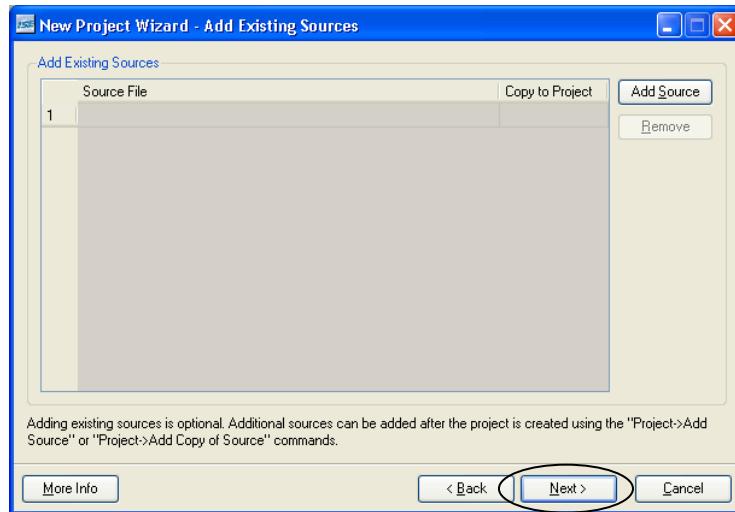


รูปที่ 2.5 New Project Wizard–Create New Source

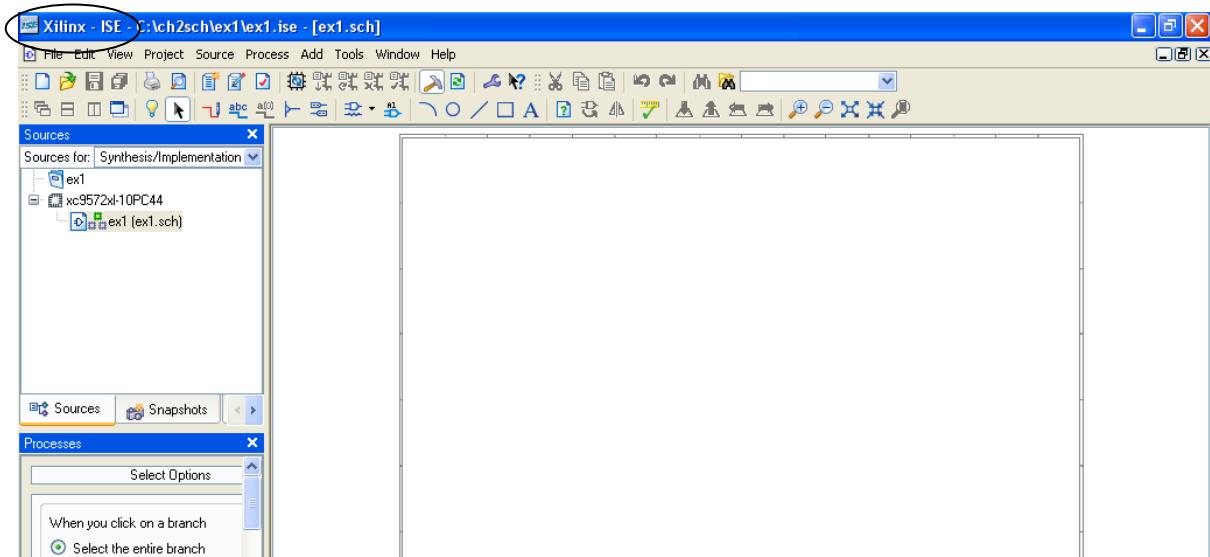


รูปที่ 2.6 หน้าต่าง New Source Wizard–Select Source Type

- 4) จากรูปที่ 2.6 คลิก Next คลิก Finish คลิก Next ก็จะได้นำต่อไปที่ 2.7 (ซึ่งจะอธิบายในภายหลัง) เสร็จแล้วคลิก Next และคลิก Finish จะได้นำต่อไป Xilinx-ISE สำหรับ Schematic editor (โปรแกรม Schematic editor หรือ ECS) ดังรูปที่ 2.8

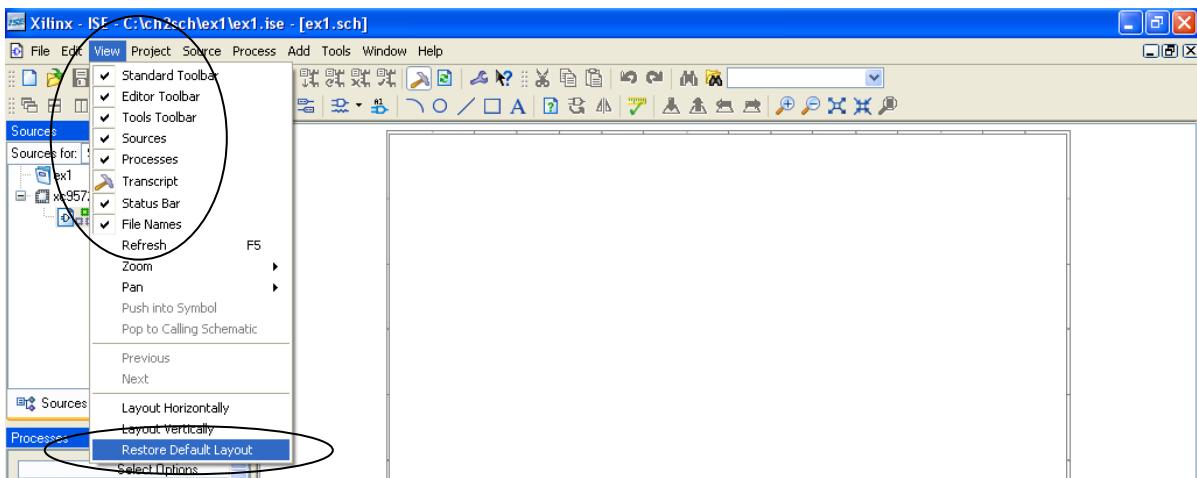


รูปที่ 2.7 หน้าต่าง New Project Wizard-Add Existing Source



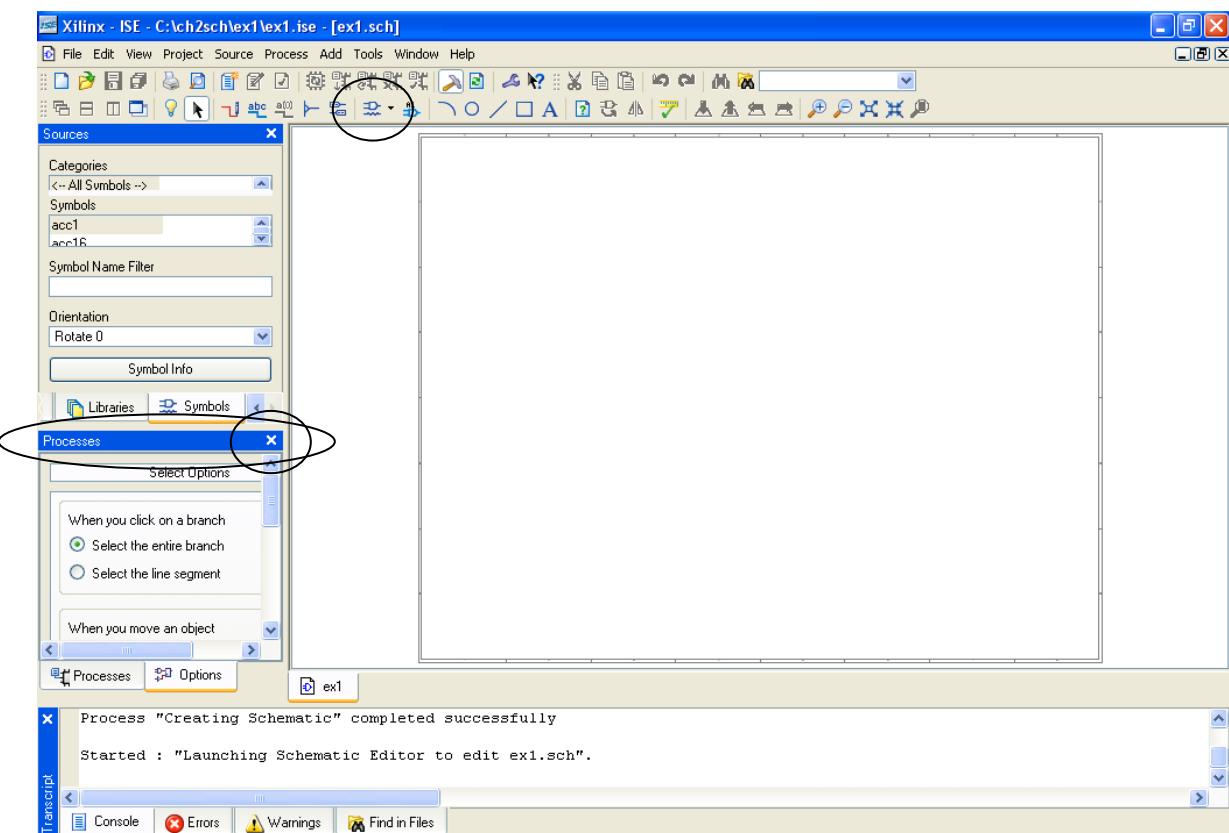
รูปที่ 2.8 หน้าต่าง Xilinx-ISE สำหรับ Schematic editor

- 5) ให้ทำความคุ้นเคยกับหน้าต่างรูปที่ 2.8 โดยคลิก View แล้วคลิกที่แบบข้อความที่ลักษณะ แล้วคุ่าว่าหน้าต่างเปลี่ยนแปลงอย่างไรบ้าง ถ้าต้องการให้หน้าต่างกลับมาไม้ลักษณะเหมือนเดิม ให้คลิกແเล็บล่างสุดคือ Restore Default Layout ดังในรูปที่ 2.9

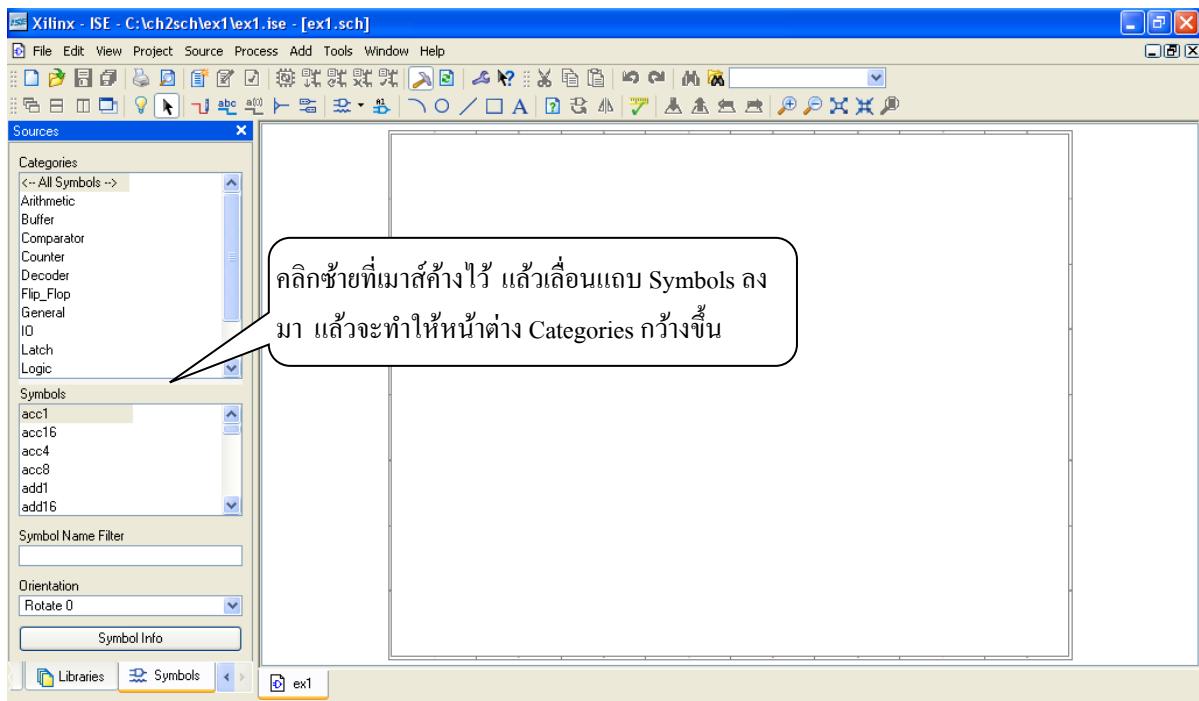


รูปที่ 2.9 การใช้ View

6) การวางแผนจร คลิก และคลิก ของหน้าต่าง Processes ดังในรูปที่ 2.10 แล้วจะได้รายการของหมวดอุปกรณ์ใน Categories และชนิดอุปกรณ์ใน Symbols ดังรูปที่ 2.11 (ถ้าต้องการกลับไปที่รูปที่ 2.10 อีกครั้งให้คลิกที่ View -> Processes)

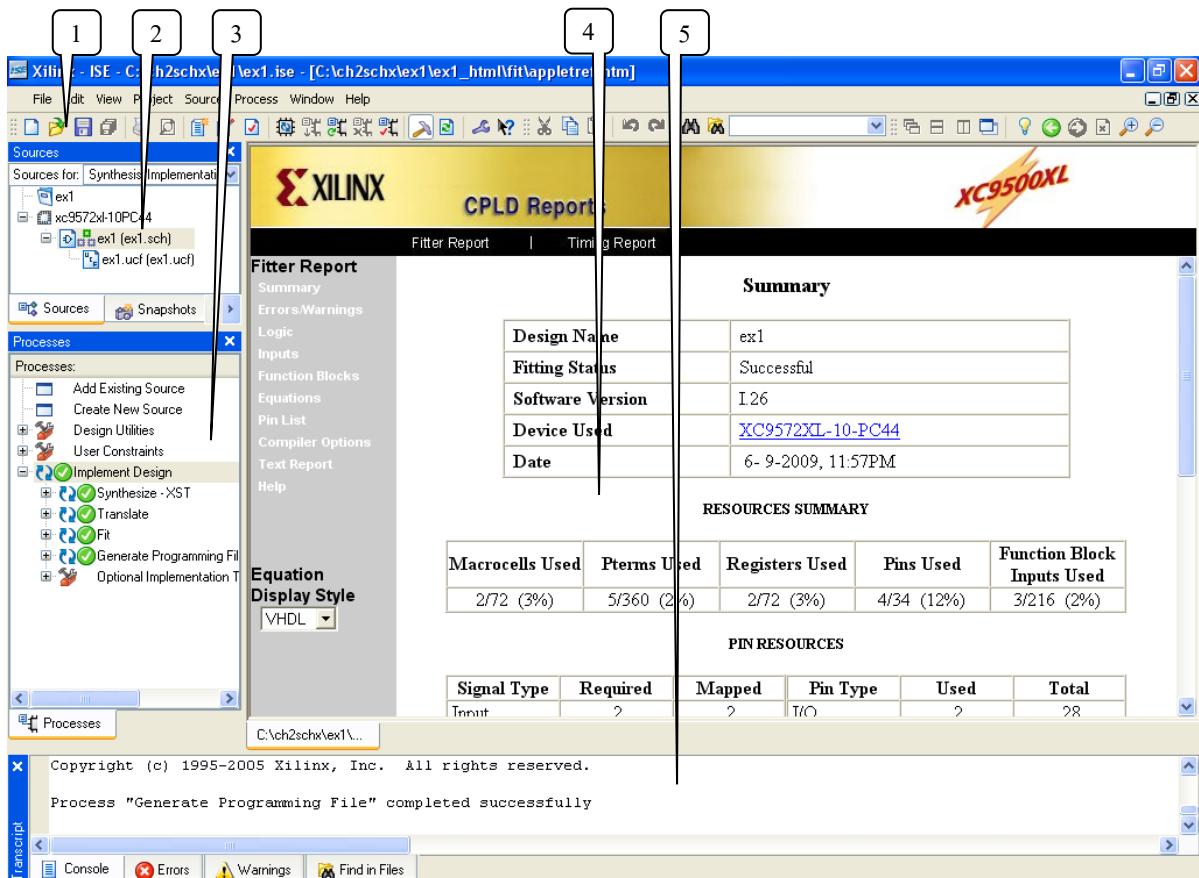


รูปที่ 2.10 รายการอุปกรณ์



รูปที่ 2.11 รายการอุปกรณ์ที่เห็นรายละเอียด

7) ก่อนทำขั้นตอนต่อไปควรทำความรู้จักกับหน้าต่างหลัก Project Navigator (Project Navigator main window) ดังรูปที่ 2.12



รูปที่ 2.12 หน้าต่างหลัก Project Navigator หรือ Xilinx-ISE (ความละเอียด 1024 by 768 pixels)

หน้าต่างหลัก Project Navigator ดังรูปที่ 2.12 จะประกอบด้วยหมายเลข

1. ทูลบาร์ (Toolbar)
2. หน้าต่าง Sources (Sources window)
3. หน้าต่าง Processes (Processes window)
4. หน้าต่างหลัก (Workspace)
5. หน้าต่าง Transcript (Transcript window)

Standard toolbar จะเป็นคำสั่งที่ถูกใช้บ่อยใน Project Navigator มีรายละเอียดดังนี้

	New
	Open
	Save
	Save All
	Print
	Print Preview
	New Source
	Open Source
	Edit Project Properties

	Implement Top Module
	Run Process
	Rerun All Process
	Stop Process
	Edit Process Properties
	Toggle Transcript
	Refresh
	Support and Services
	Help

Editor toolbar เป็น Edit Menu commands ที่ใช้บ่อยและสัญลักษณ์ที่ใช้ส่วนใหญ่เราคุ้นเคยกันดีอยู่แล้วมีดังดังนี้

	Cut
	Copy
	Paste
	Undo
	Redo
	Find
	Find in Files

ในการตรวจสอบจะใช้คำสั่งที่ Toolbar บนเมนูบาร์ มีคำสั่งที่ควรทราบมีดังนี้

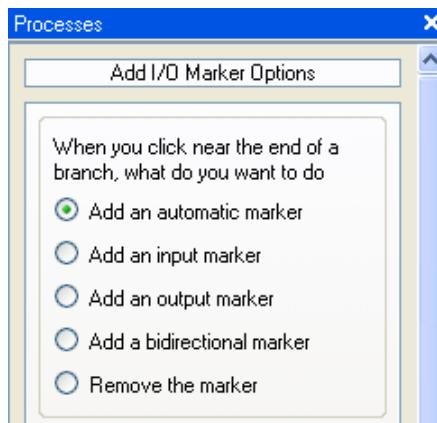
- a) คือ Zoom In ใช้ในการขยายผังวงจร ทำได้โดยการคลิกปุ่ม
- b) คือ Zoom Out ใช้ในการย่อผังวงจร ทำได้โดยการคลิกปุ่ม
- c) คือ Zoom Full View ใช้ในการย่อผังวงจรออกให้เห็นเต็มพื้นที่ ซึ่งทำได้โดยการคลิกปุ่ม
- d) คือ Zoom To Box ใช้ขยายผังวงจรให้เห็นบริเวณพื้นที่เฉพาะส่วนที่ต้องการ ทำได้โดยการคลิกปุ่ม แล้วเลื่อนเมาส์ไปยังจุดเริ่มต้นที่ต้องการขยายแล้วคลิก (ซ้าย) เมาส์ค้างไว้ จากนั้นลากเมาส์ตามแนวภาพ直到วางเมาส์ไว้ที่จุดที่ต้องการ ทำให้ผังวงจรแสดงเฉพาะส่วนที่เลือกไว้
- e) คือ Select ใช้สำหรับเลือกตัวอุปกรณ์เพื่อทำงานฟังก์ชันอื่นๆ ต่อไป หรือใช้เมื่อต้องการหยุด

f) คือ Add Symbols ใช้เพื่อมอุปกรณ์ในผังวงจร โดยจะเลือกอุปกรณ์จากรายการที่อยู่ใน Symbols ของ Categories ต่างๆ ที่ต้องการนำมาใช้ โดยคลิก ที่อยู่บน Toolbar ก็จะปรากฏรายการหมวดของอุปกรณ์ในช่อง Categories และชนิดของอุปกรณ์ในช่อง Symbols ที่ได้อธิบายในข้อ 6) ใน Categories มีหมวดหมู่อุปกรณ์ เช่น Counter, Flip-flop และ Logic เป็นต้น ส่วนใน Symbols จะมีรายละเอียดอุปกรณ์ เช่น cb16ce (วงจรนับเลขไบนาเรีย 16 แบบซิงโตรนัส), fd (D Flip-flop) และ and2 (แอนด์เกต 2 อินพุต) เป็นต้น เมื่อเลือกชนิดของอุปกรณ์ตามต้องการแล้วให้เลื่อนมาส์ไฟที่พื้นที่ที่ต้องการวางอุปกรณ์ คลิก (ซ้าย) เม้าส์เพื่อวางอุปกรณ์ ณ จุดนั้น และถ้าใช้อุปกรณ์แบบเดียวกันหลายๆ ตัว ก็สามารถคลิกเม้าส์เพื่อวางอุปกรณ์ ณ จุดที่ต้องการได้เรื่อยๆ ไปจนกว่าจะครบ เมื่อต้องการหยุดก็ให้กดลับไปคลิก

g) คือ Rotate ใช้หมุนตัวอุปกรณ์ที่เลือกไว้ โดยคลิกเลือกที่ตัวอุปกรณ์ที่ต้องการจนกลายเป็นสีแดง จากนั้นคลิก จะทำให้ตัวอุปกรณ์ที่เลือกไว้หมุนในทิศทางทวนหรือตามเข็มนาฬิกาครึ่งละ 90 องศา และในกรณีที่ต้องการให้หมุนกลับทางให้คลิก ก่อนจากนั้นจึงคลิก

h) คือ Add Wire ใช้ลากเส้นสายสัญญาณเชื่อมระหว่างจุดสองจุด ทำได้โดยคลิก จากนั้นเลื่อนมาส์ไฟยังจุดเริ่มต้นแล้วคลิกเม้าส์ทิ้งไว้แล้วเลื่อนมาส์ไฟยังจุดปลายหรือข้างของอินพุตเอาต์พุตและคลิกเม้าส์ไว้ โปรแกรมจะลากเส้นโดยอัตโนมัติ หรือจะเราลากเส้นที่ละส่วนไปเรื่อยๆ ก็ได้ และเมื่อต้องการหยุดให้กดลับไปคลิก

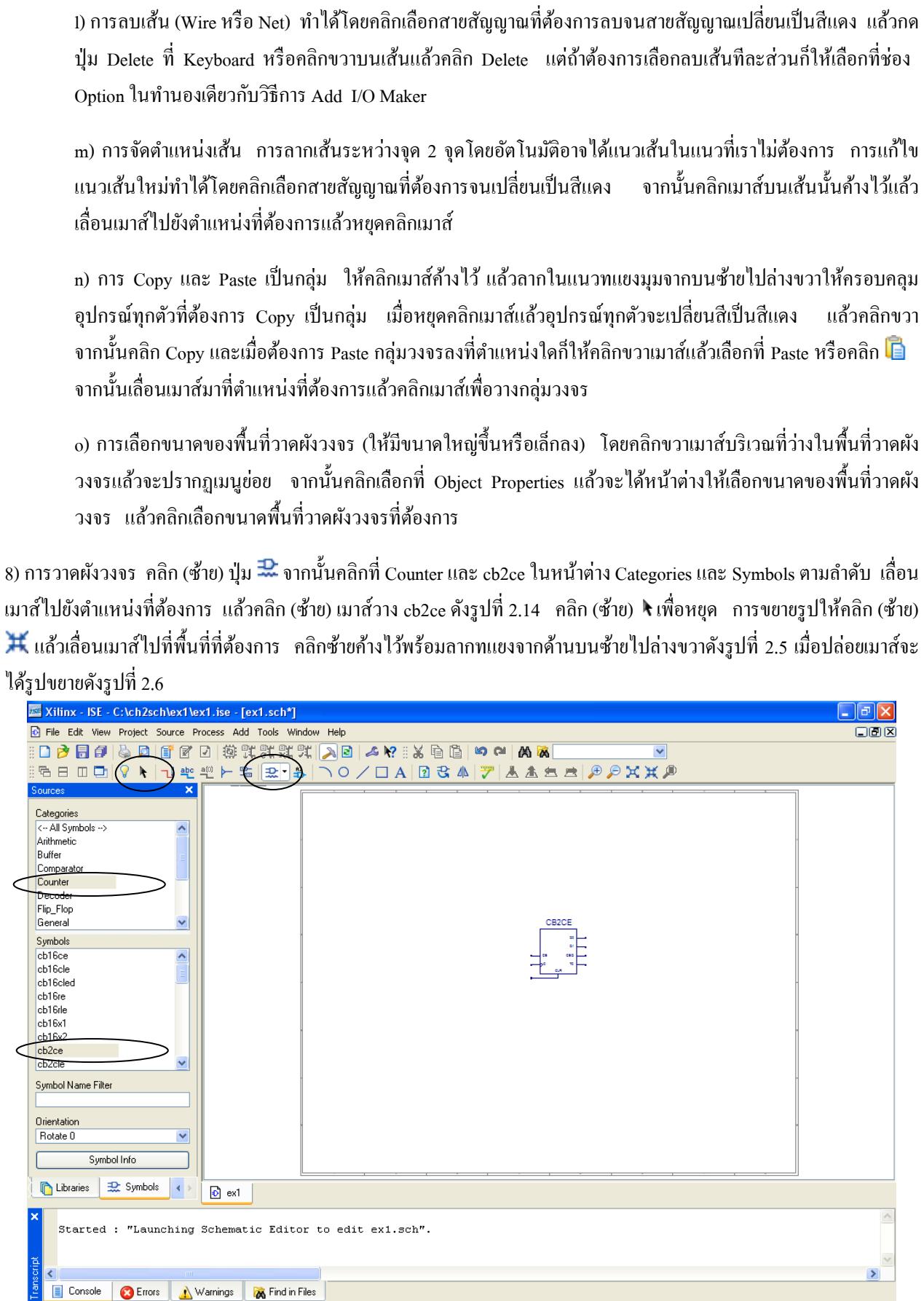
i) คือ Add I/O Maker ใช้สำหรับกำหนดจุดปลายของสายสัญญาณ (Net) เข้ากับข้ออินพุตเอาต์พุต (I/O) แต่ละ I/O ของวงจรที่ออกแบบ โดยใช้ I/O Maker ให้สายสัญญาณอินพุต หรือ เอาต์พุต หรือสองทิศทาง (Bi-directional) ซึ่งสามารถทำได้โดยคลิก แล้วเลื่อนมาส์ไฟยังจุดปลายหรือข้างของอินพุตเอาต์พุตและคลิกเม้าส์ไว้ ตามที่ต้องการ จากนั้นวาง I/O Maker ต่อไปจนครบ ขอให้สังเกตว่าเมื่อคลิก แล้วจะได้หน้าต่าง Add I/O Marker Options ดังรูปที่ 2.13 เราสามารถเลือกว่า Add an automatic marker, Add an input maker, Add an output maker, Add a bidirectional maker หรือ Remove the marker และเมื่อต้องการหยุดก็ให้กดลับไปคลิก



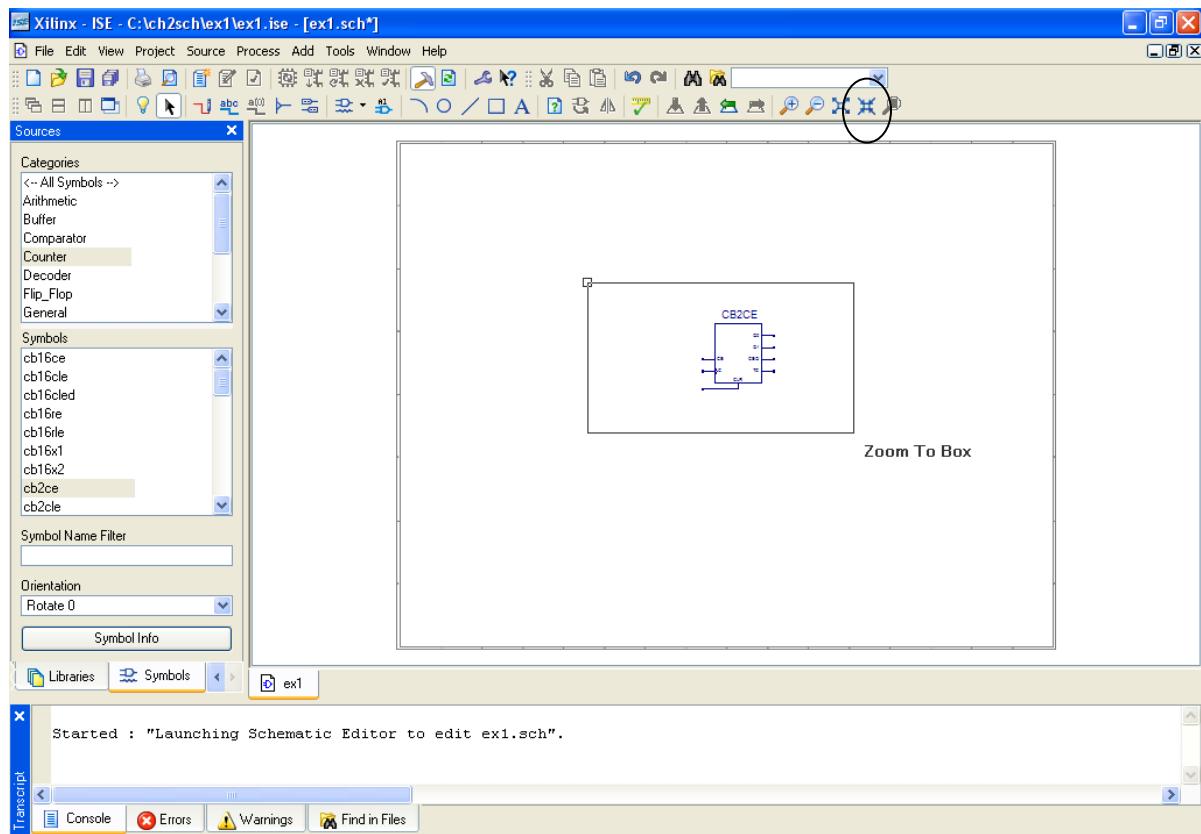
รูปที่ 2.13 หน้าต่าง Add I/O Marker Options

j) คือ Add Bus tap ใช้สำหรับแยกสัญญาณย่อยออกจากบัสหลัก โดยคลิก จากนั้นเลื่อนมาส์ไฟยังตำแหน่งบนบัสที่ต้องการแล้วคลิกเม้าส์เพื่อวาง Bus tap ลงบนบัสวางจร เมื่อต้องการหยุดก็ให้กดลับไปคลิก

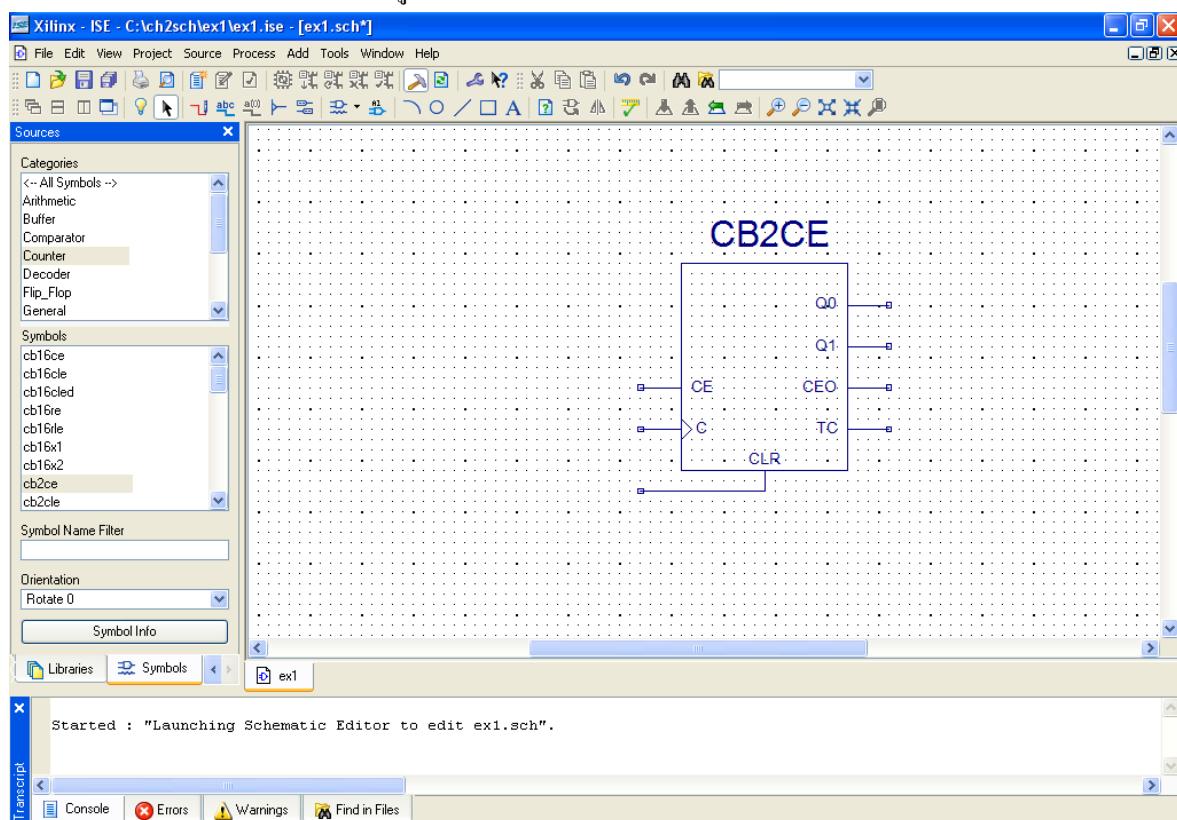
k) การลบ Symbol ของอุปกรณ์ ทำได้โดยคลิกเลือกที่ Symbols ตัวอุปกรณ์ที่ต้องการลบจะเปลี่ยนเป็นสีแดง แล้วกดปุ่ม Delete ที่ Keyboard หรือคลิกขวาที่ Symbols แล้วคลิก Delete



รูปที่ 2.14 การคลิกเลือกหมวดและชนิดอุปกรณ์ที่อยู่ใน Categories และ Symbols ตามลำดับ

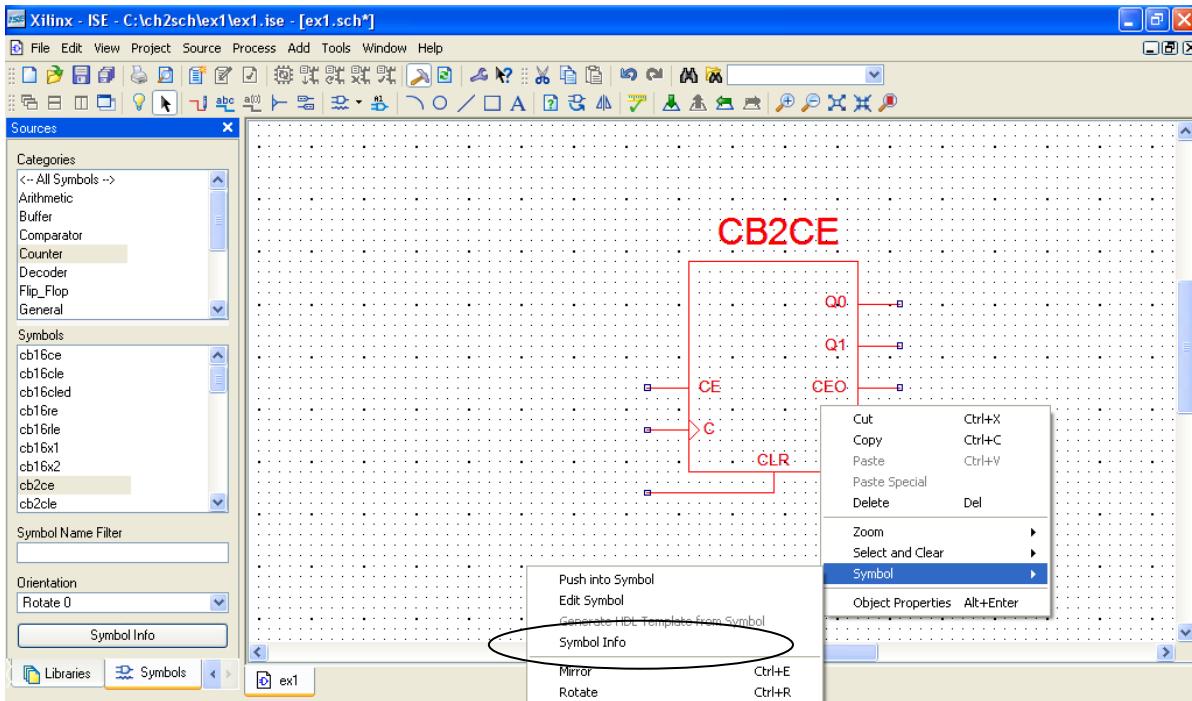


รูปที่ 2.15 แสดงขั้นตอนการ Zoom ขยาย



รูปที่ 2.16 รูปที่ Zoom ขยายแล้ว

9) การใช้คำสั่งต่างๆ คลิกมาส์ที่อุปกรณ์ (Symbols) CB2CE จนเป็นสีแดง (Select) แล้วคลิกขวาจะปรากฏเมนูคำสั่ง เช่น Cut Copy Paste และ Delete (เป็นคำสั่งเดียวกับที่ແນ Toolbar) คนที่ยังไม่คุ้นเคย Symbols ต่างๆ ให้คลิก CB2CE จนเป็นสีแดงแล้ว คลิกແນ Symbol -> Symbol Info ดังรูปที่ 2.17 จะได้ไฟล์แสดงรายละเอียด CB2CE ดังรูปที่ 2.18 แล้วคลิก ปิดไฟล์



รูปที่ 2.17 ขั้นตอนการใช้คำสั่งเพื่อคุ้นเคย Symbol ต่างๆ

CB2CE, CB4CE, CB8CE, CB16CE

2-, 4-, 8-,16-Bit Cascadable Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

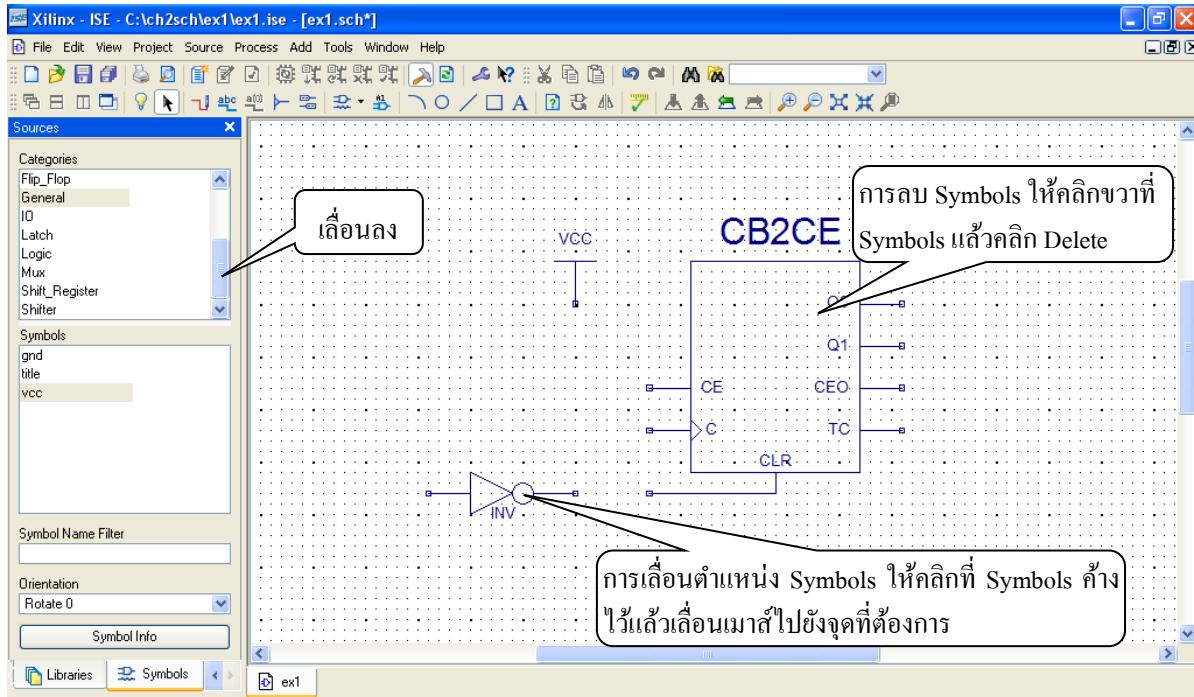
CB2CE, CB4CE, CB8CE, CB16CE	
Spartan-II, Spartan-IIIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500V, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

CB2CE, CB4CE, CB8CE, and CB16CE are, respectively, 2-, 4-, 8-, and 16-bit (stage), asynchronous, clearable, cascadable binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the CEO output of the first stage to the CE

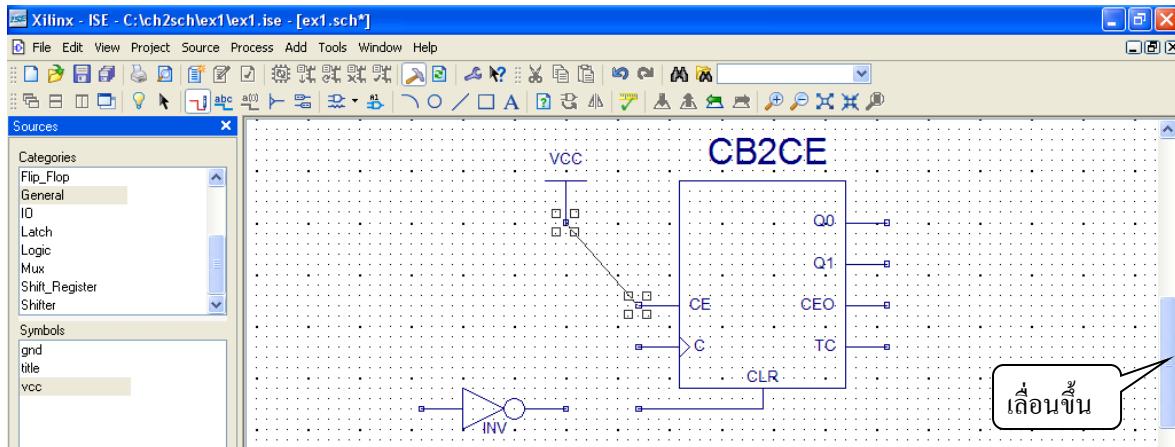
รูปที่ 2.18 รายละเอียดของ Symbol cb2ce

- 10) คลิกปุ่ม แล้วคลิกที่ Logic และ inv ใน Categories และ Symbols ตามลำดับ เลื่อนมาส์มา ณ ตำแหน่งที่ต้องการ คลิกมาส์ว่าง inv แล้วคลิก จากนั้นวาง Vcc โดยคลิกเลือกที่ General และ Vcc แล้วนำมาร่างดังรูปที่ 2.19 แล้วคลิก

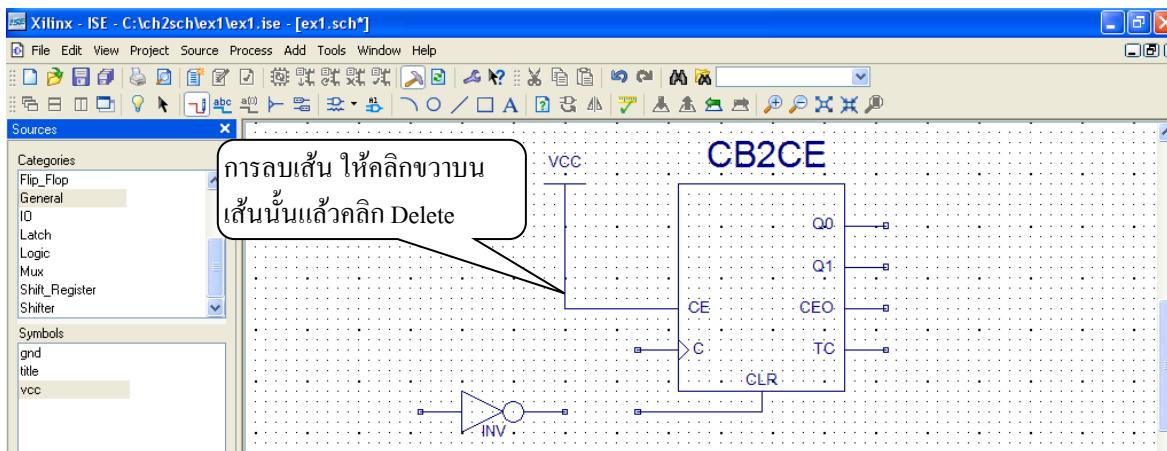


รูปที่ 2.19 วางแผนบนเวอร์เตอร์และ Vcc

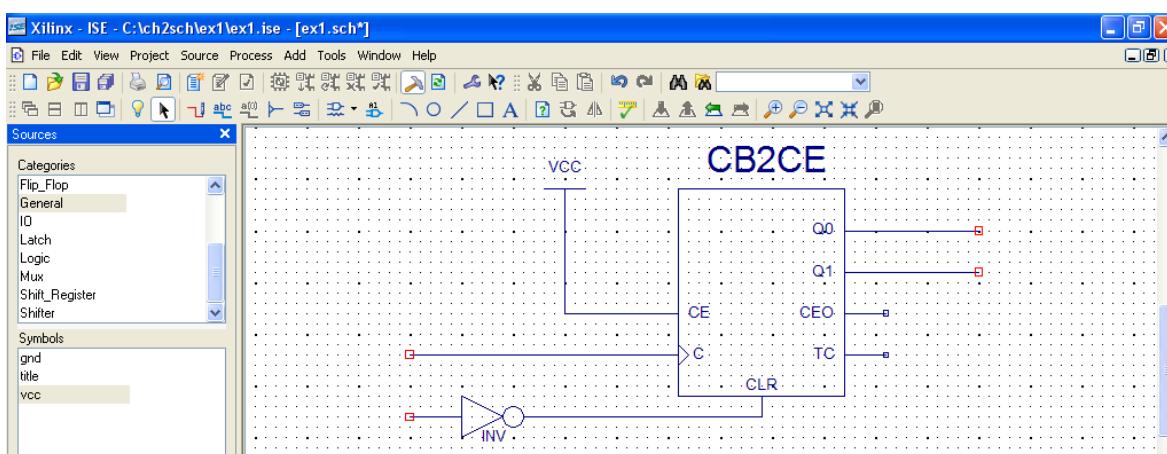
- 11) การลากเส้น คลิก แล้วเลื่อนมาส์ไปที่จุดปลาย Symbols คลิกมาส์ค้างไว้แล้วลากไปที่จุดปลายอีกด้านดังรูปที่ 2.20 เมื่อหยุดคลิกแล้วเส้นจะถูกวาดเองโดยอัตโนมัติดังรูปที่ 2.21 จากนั้นลากสายสัญญาณเส้นต่อๆ ไปจนครบทุกเส้นดังรูปที่ 2.22



รูปที่ 2.20 แสดงการลากเส้นระหว่างจุด 2 จุด

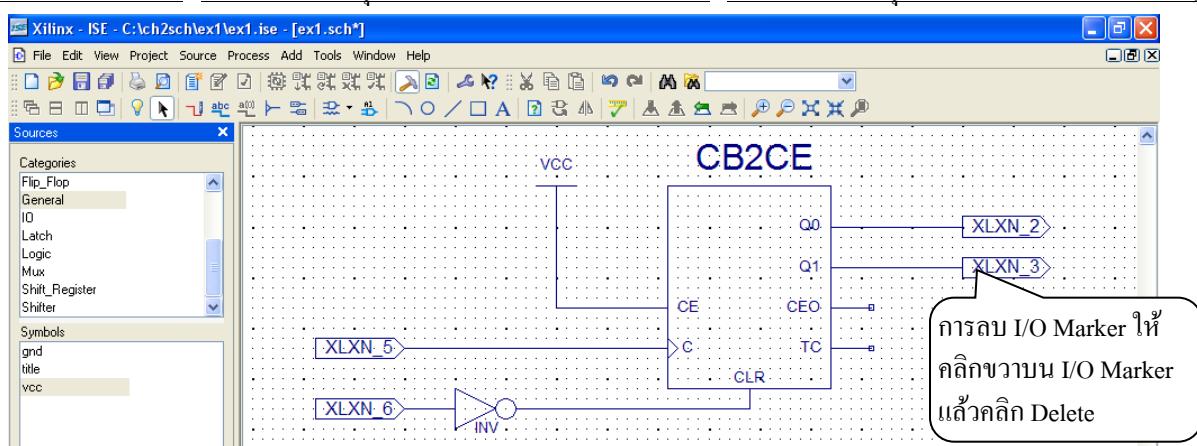


รูปที่ 2.21 สายสัญญาณที่ถูกความโดยอัตโนมัติ



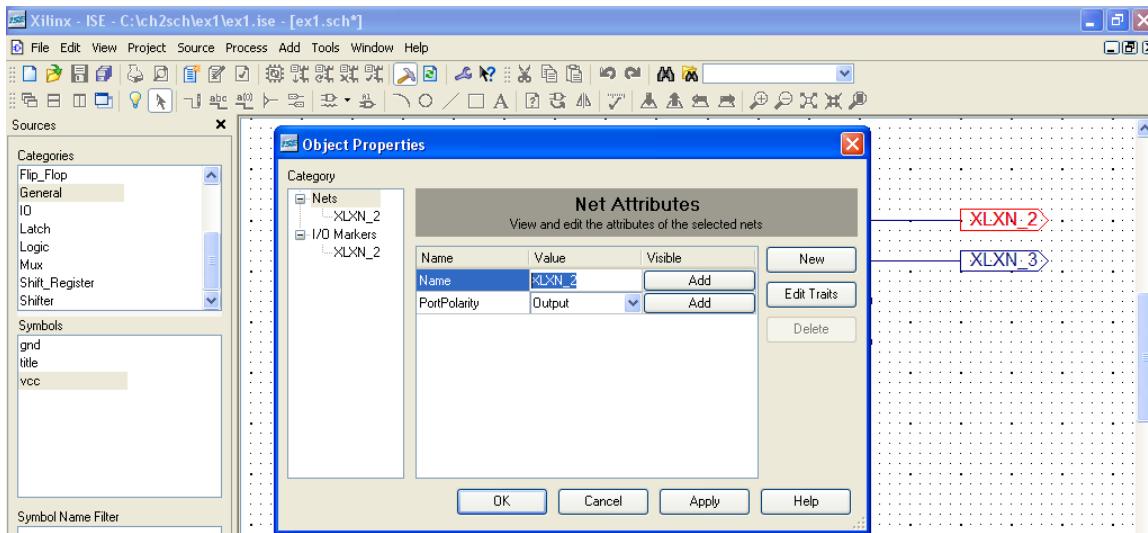
รูปที่ 2.22 แสดงการลากสายสัญญาณต่างๆ จนครบตามที่ออกแบบไว้

12) การใส่ I/O Marker เพื่อกำหนดจุดปลายสายสัญญาณ คลิก จากนั้นเลื่อนมาส์ปีเพื่อขึ้นของอินพุตหรือเอาต์พุต แล้วคลิกเมามือส่อง I/O Marker ตามที่ต้องการ เมื่อกรอบแล้วให้หยุดโดยคลิก จะได้ดังรูปที่ 2.23 (กรณีนี้ CEO และ TC ไม่ได้ใช้ จึงไม่ต้องการใส่ I/O Marker แต่ถ้าเป็นขาอินพุตห้ามปล่อยขาอยู่ไว้อย่างเด็ดขาด ซึ่งอาจต้องนำอินพุตเข้ากับ VCC หรือ GND ก็ได้)

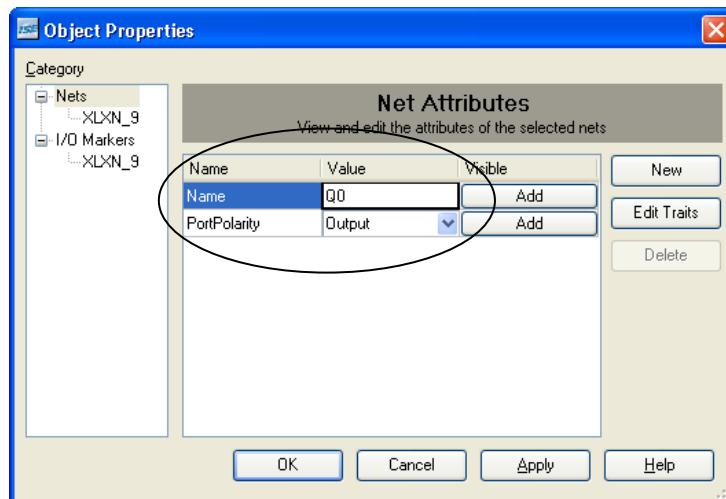


รูปที่ 2.23 การใส่ I/O Marker เพื่อกำหนดจุดปลายของสัญญาณ

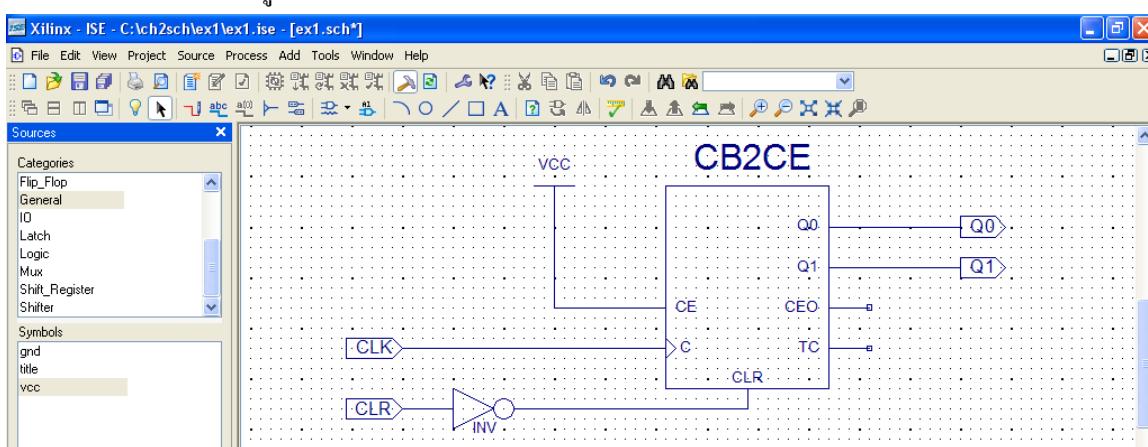
13) จากรูปที่ 2.23 การแก้ไขชื่อของสายสัญญาณที่ I/O Marker (ชื่อของ Wire หรือ Net) ทำได้โดยดับเบิลคลิกที่ I/O Marker ที่ต้องการแล้วจะได้ดังรูปที่ 2.24 จากนั้นทำการแก้ไขที่คอลัมน์ Value และ Name เป็นชื่อ Q0 ส่วนที่acco PortPolarity นั้นเป็น Output ถูกต้องแล้ว (ไม่ต้องแก้ไข) ดังรูปที่ 2.25 คลิก OK จากนั้นแก้ไข I/O Marker จนแล้วเสร็จ จะได้ดังรูปที่ 2.26



รูปที่ 2.24 ขั้นตอนการแก้ไขชื่อของ Wire หรือ Net ที่ I/O Marker

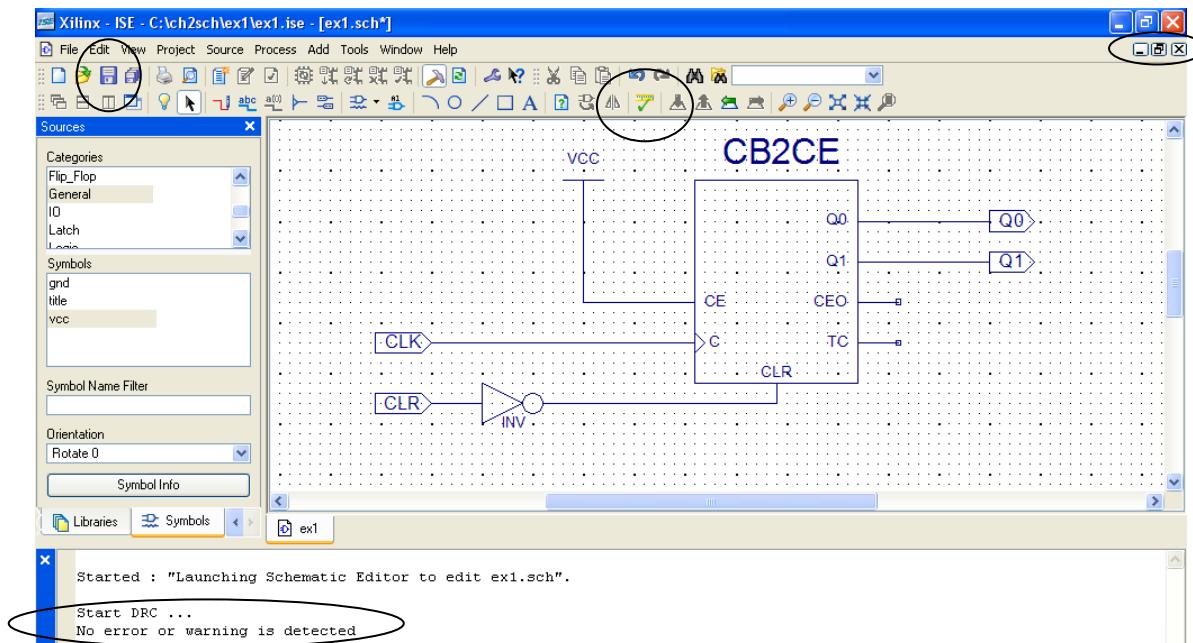


รูปที่ 2.25 ขั้นตอนการแก้ไขชื่อของ Wire หรือ Net ที่ I/O Marker

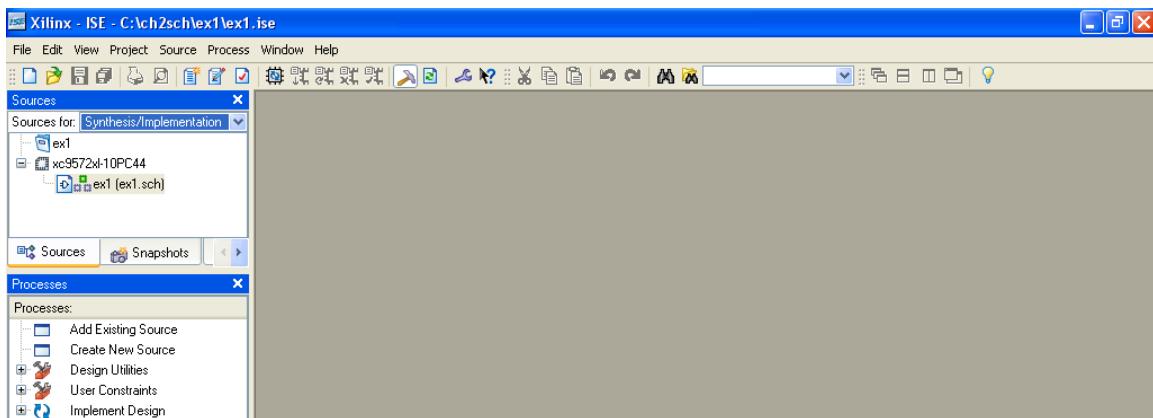


รูปที่ 2.26 การแก้ไขชื่อของ Wire หรือ Net ที่ I/O Marker จนได้จริงที่แล้วเสร็จสมบูรณ์

- 14) การตรวจความถูกต้อง (Syntax) ทำได้โดยคลิก ถ้าไม่มีข้อผิดพลาดจะปรากฏข้อความ No error or warning is detected ดังรูปที่ 2.27 ถือว่าการวางแผนวงจรเสร็จสมบูรณ์ คลิก เพื่อบันทึกไฟล์ คลิก (สีดำ) ปิดโปรแกรมเพื่อกลับไปที่หน้าจอ Xilinx-ISE อีกครั้ง คลิก View -> Restore Default Layout ดังในรูปที่ 2.9 แล้วจะได้ดังรูปที่ 2.28 เพื่อการทำขั้นตอนต่อไป

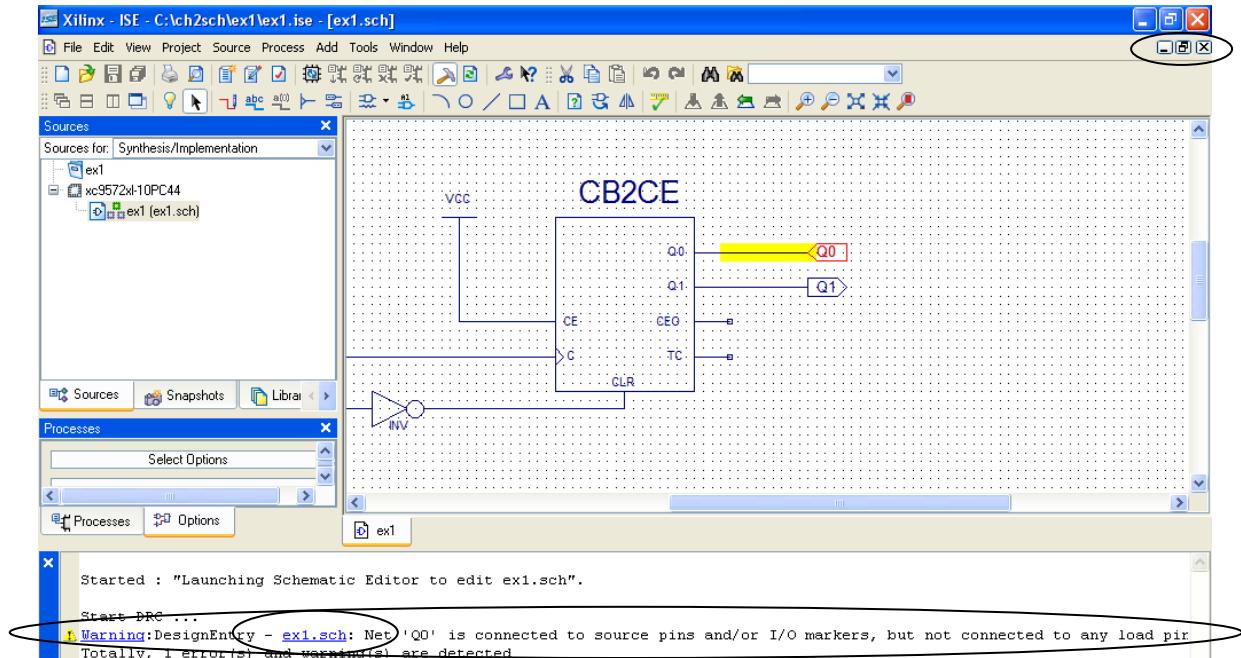


รูปที่ 2.27 การตรวจสอบความถูกต้อง



รูปที่ 2.28 หน้าต่าง Xilinx-ISE ของโปรเจกต์ไฟล์ชื่อ ex1

- 15) ในกรณีที่มีข้อผิดพลาด เมื่อคลิก แล้วจะแสดงข้อผิดพลาดที่หน้าต่าง Transcript (ล่างสุด) จากนั้นคลิกที่ ex1.sch เพื่อแสดงจุดที่ผิดพลาด จะได้ดังรูปที่ 2.29 การแก้ไขให้ดับเบิลคลิก I/O Marker ‘Q0’ แล้วแก้ไขเป็น ‘Output’ และคลิก OK แล้วคลิก บันทึกไฟล์ คลิก (สีดำ) ปิดโปรแกรม Schematic editor เพื่อกลับไปที่หน้าต่าง Xilinx-ISE อีกครั้งดังรูปที่ 2.28



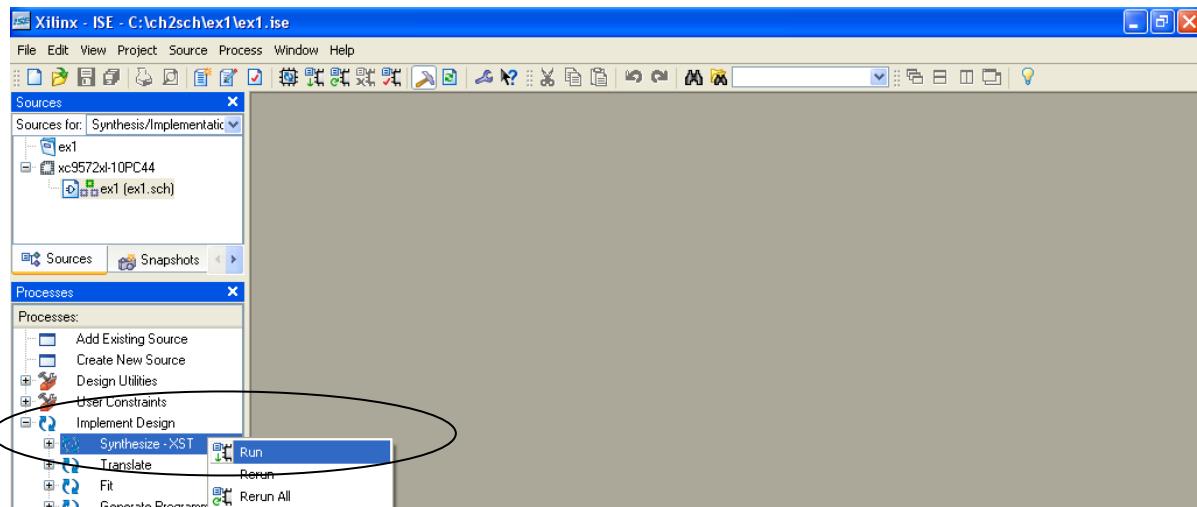
รูปที่ 2.29 การตรวจสอบความถูกต้องที่มีข้อผิดพลาด

2.1.2 การตรวจสอบความถูกต้องของจริงที่ออกแบบ (Design Verification)

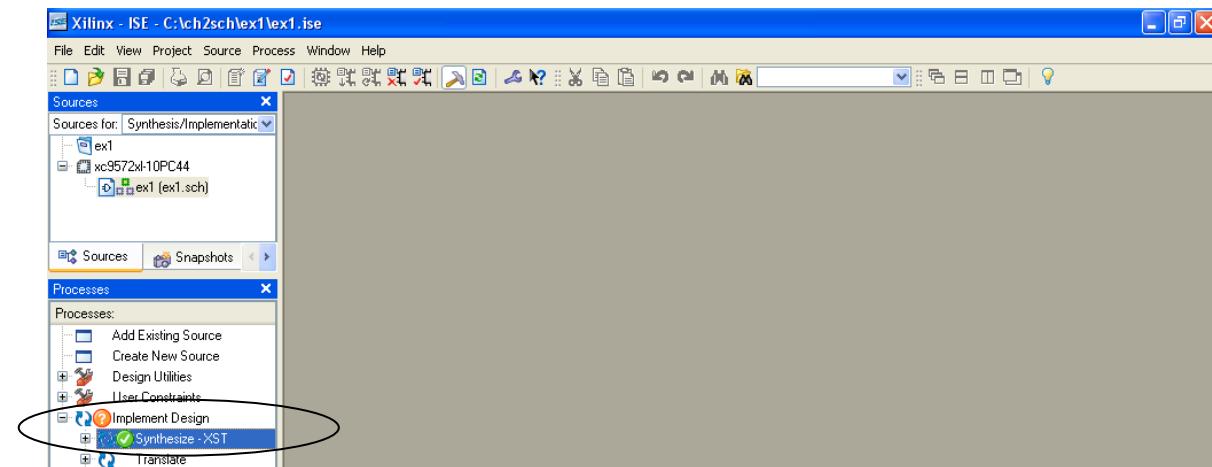
การออกแบบวงจรอาจไม่มีความจำเป็นต้องทำขั้นตอนนี้ก็ได้ ซึ่งรายละเอียดจะอธิบายในภายหลัง

2.1.3 การสังเคราะห์วงจร (Design synthesis)

ในรูปที่ 2.28 คลิก “+” หน้า Implement Design ในหน้าต่าง Processes จนเป็น “-” จากนั้นคลิกขวาแล้วคลิก Run ที่ Synthesize-XST ดังรูปที่ 2.30 หรือดับเบิลคลิก ถ้าได้ ✓ หรือ ⚡ หน้า Synthesize-XST ดังรูปที่ 2.31 ถือว่าสังเคราะห์ผ่าน



รูปที่ 2.30 หน้าต่าง Processes ที่แสดงขั้นตอนสังเคราะห์วงจร

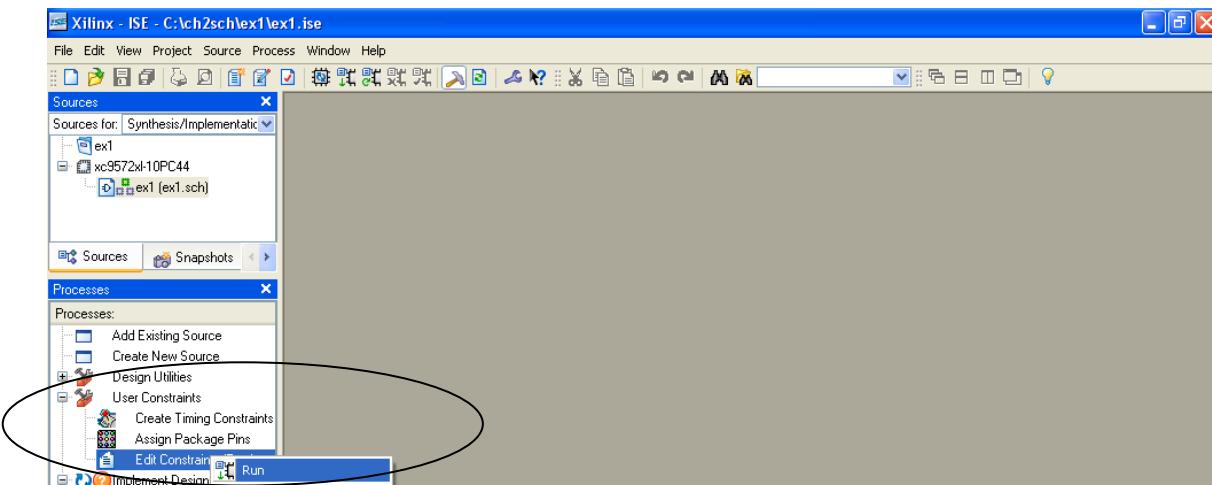


รูปที่ 2.31 หน้าต่าง Processes เมื่อสังเคราะห์วงจรเสร็จเรียบร้อยแล้ว

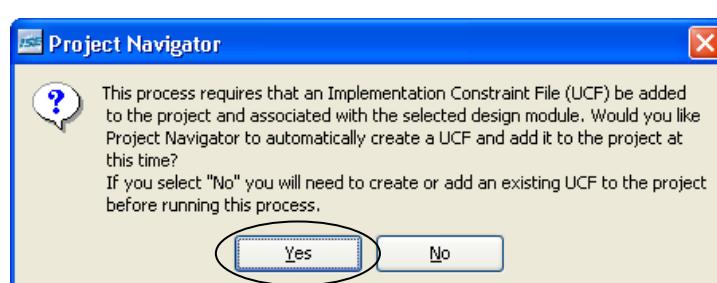
หมายเหตุ ขอให้สังเกตว่า Process status icon มีดังนี้ = Running, = Up-to-date (ถือว่าผ่าน), = Warnings Reported (ถือว่าผ่าน), = Errors Reported (ถือว่าไม่ผ่าน), = Out-of-Date (คือไม่อัพเดทแล้ว ต้องคลิก Run ใหม่อีกครั้ง)

2.1.4 Design Implementation

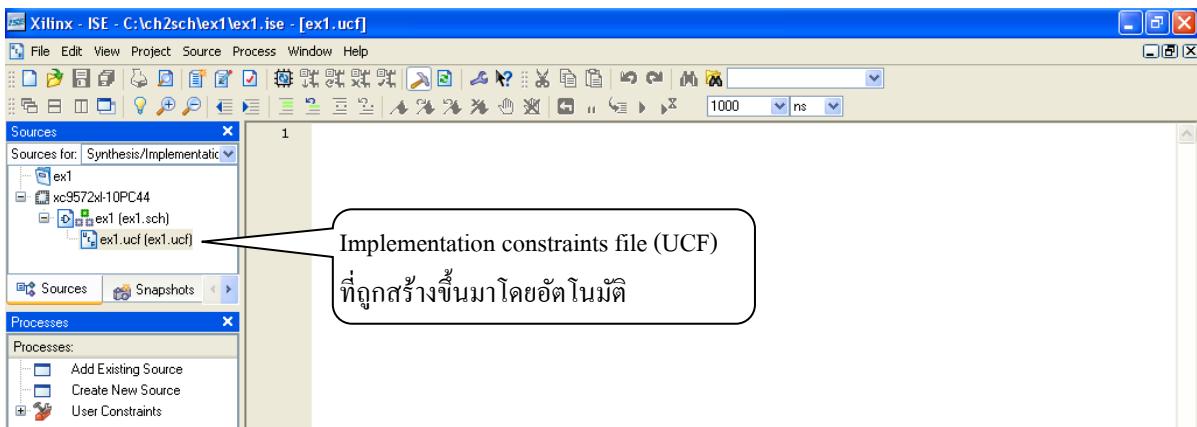
1) ขั้นตอนสร้าง Implementation constraints file หรือ User constraints file (UCF) ให้คลิก “+” หน้า User Constraints ในหน้าต่าง Processes จะเป็น “-” คลิกขวาแล้วคลิก Run ที่ Edit Constraints (Text) ดังรูปที่ 2.32 หรือดับเบิลคลิกแล้วคลิก Yes ในรูปที่ 2.33 เพื่อสร้าง UCF โดยรูปที่ 2.34 ในหน้าต่าง Sources นั้นจะมีไฟล์ ex1.ucf เพิ่มเข้ามา



รูปที่ 2.32 ขั้นตอนการสร้าง Implementation constraints file โดยอัตโนมัติ

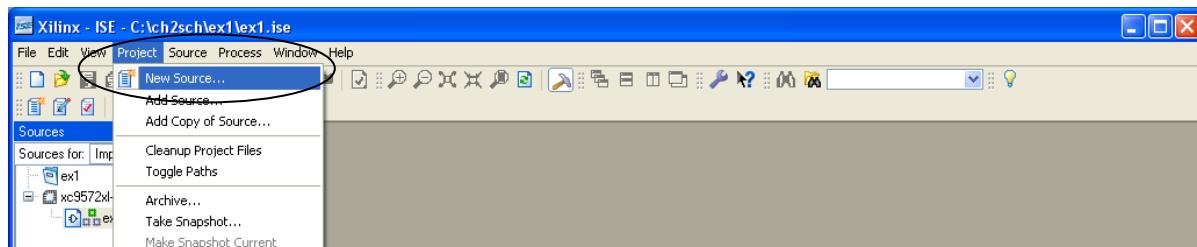


รูปที่ 2.33 การยืนยันการสร้าง Implementation constraints file โดยอัตโนมัติ

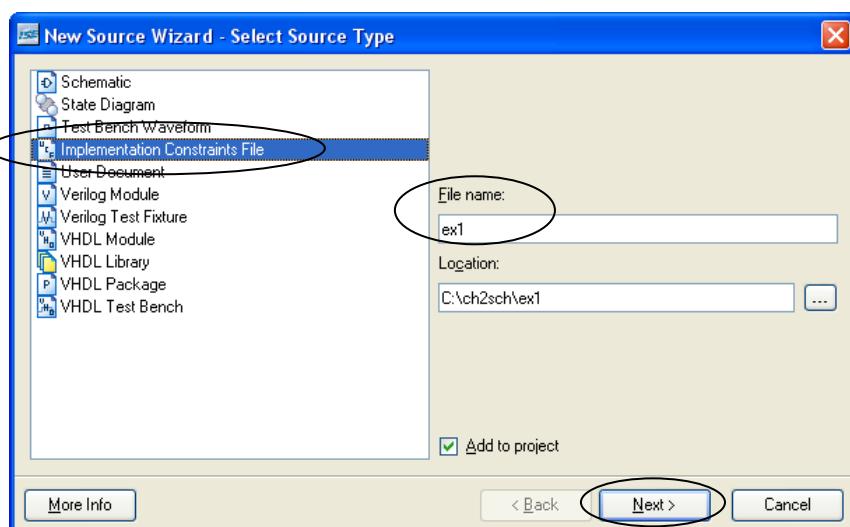


รูปที่ 2.34 หน้าต่างที่ใช้กำหนดขา CPLD ใน Edit Constraints (Text)

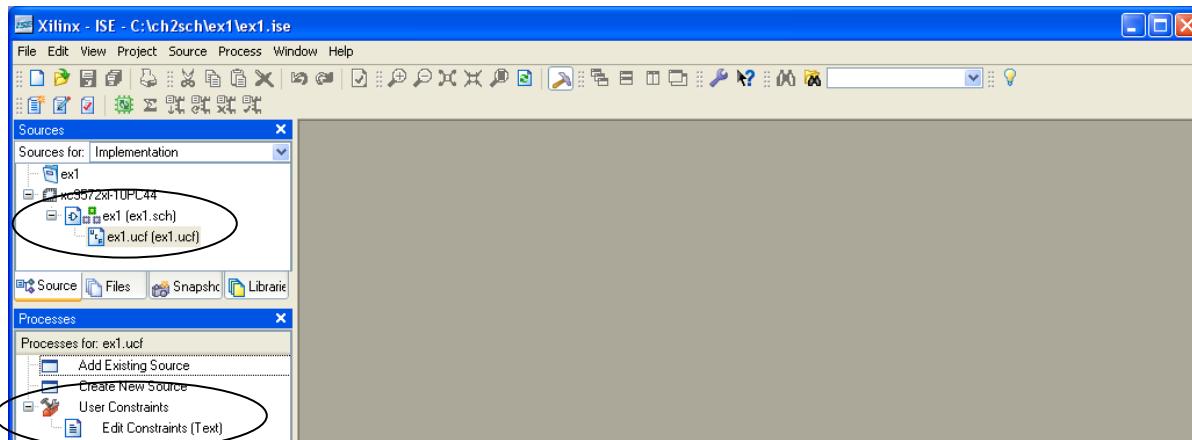
สำหรับคนที่ใช้ออฟต์แวร์ทุก ISE WebPACK 10.1i นั้นควรใช้วิธีการสร้าง Implementation constraints file หรือ User constraints file (UCF) โดยเริ่มที่หน้าต่าง Xilinx-ISE คลิก Project -> New Source ดังรูปที่ 2.35 แล้วจะได้หน้าต่าง New Source Wizard-Select Source Type จากนั้นพิมพ์ชื่อ ex1 ลงในช่อง File name และคลิกที่ Implementation constraints file ดังรูปที่ 2.36 คลิก Next คลิก Finish และคลิก “+” หน้า ex1(ex1.sch) ในหน้าต่าง Source จะเป็น “-” แล้วจะได้ ex1.ucf(ex1.ucf) ซึ่งเป็นไฟล์ UCF เพิ่มเข้ามา จากนั้นคลิก “+” หน้า User Constraints ในหน้าต่าง Processes จะเป็น “-” แล้วจะได้ดังรูปที่ 2.37 จากนั้นคลิกขวาแล้วคลิก Run ที่ Edit Constraints (Text) ดังรูปที่ 2.38 หรือดับเบิลคลิก แล้วจะได้ผลเช่นเดียวกับรูปที่ 2.34



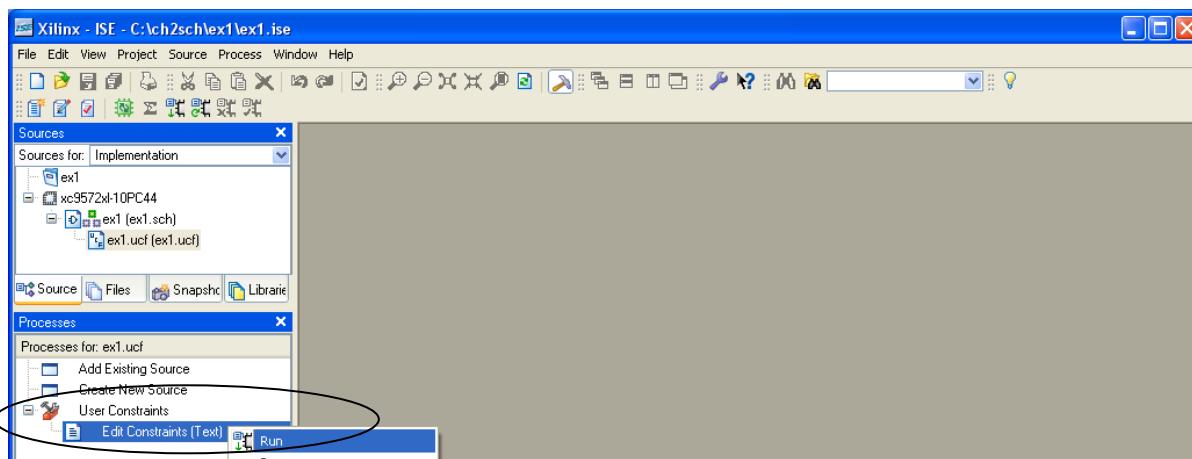
รูปที่ 2.35 หน้าต่าง Xilinx-ISE



รูปที่ 2.36 หน้าต่าง New Source Wizard-Select Source Type

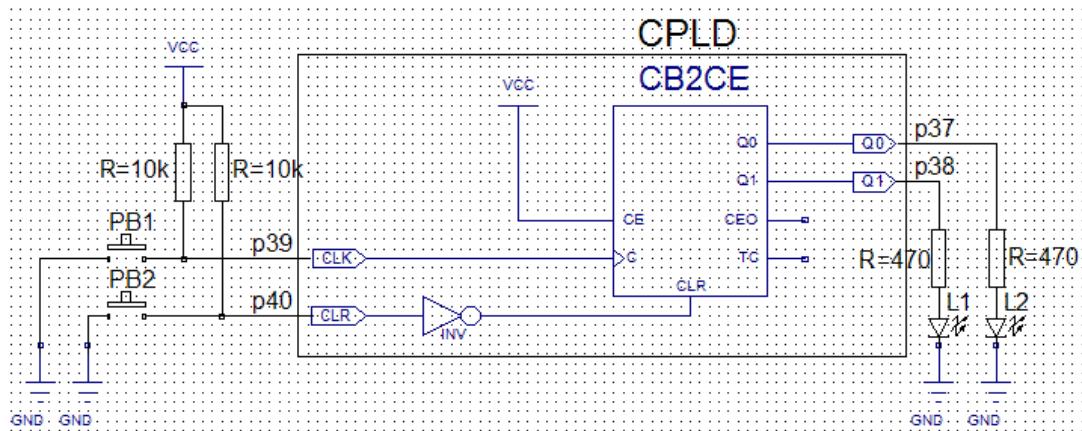


รูปที่ 2.37 หน้าต่าง Xilinx–ISE หลังจากสร้างไฟล์ UCF เพิ่มเข้าไปแล้ว



รูปที่ 2.38 หน้าต่าง Xilinx–ISE ขณะคลิกขวาแล้วคลิก Run ที่ Edit Constraints (Text) เพื่อสร้างไฟล์ UCF

2) ขั้นตอนกำหนดสายสัญญาณต่างๆ ของวงจรที่ออกแบบในรูปที่ 2.1 เข้ากับขาของ CPLD นั้นเราจำเป็นต้องกำหนดให้อินพุตและเอาต์พุตของ CPLD สอดคล้องกับอุปกรณ์ที่เตรียมไว้บนบอร์ดทดลอง CPLD Explorer XC9572XL ดังมีรายละเอียดตามตารางที่ 1.3 ในบทที่ 1 โดยที่เรากำหนดให้ปุ่มกด (Push button switch) PB1 และ PB2 จะต่ออยู่กับขา p39 และ p40 ตามลำดับ ส่วน LED1 (L1) และ LED2 (L2) ต่ออยู่กับขา p38 และ p37 แสดงดังรูปที่ 2.39



รูปที่ 2.39 การต่อ I/O ของ CPLD (เฉพาะขาที่ใช้งาน) กับอุปกรณ์ที่อยู่บนบอร์ดทดลอง (ไม่แสดง Vcc และ Gnd)

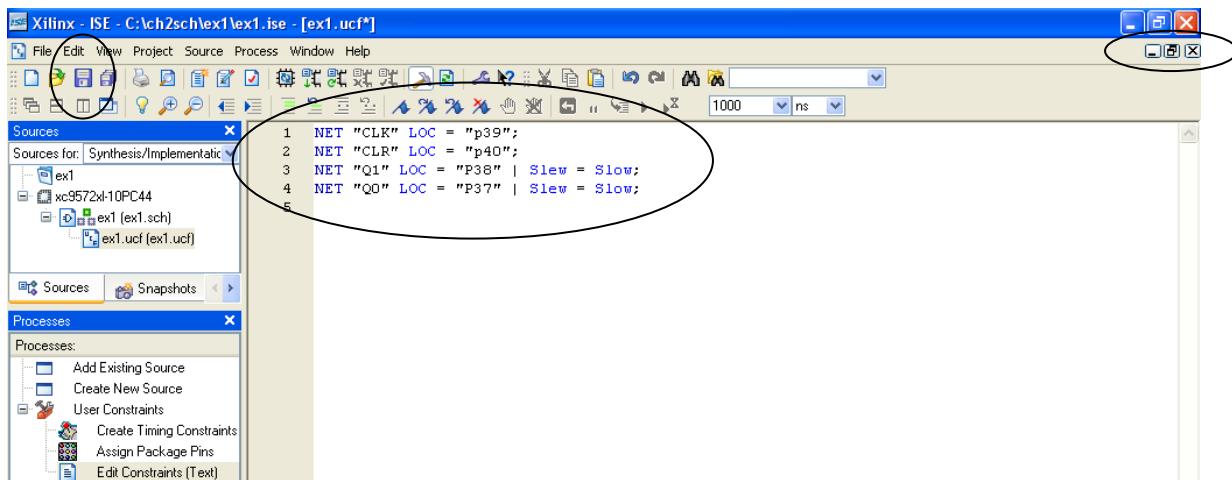
ดังนั้นการกำหนดสายสัญญาณต่างๆ ของวงจรที่ออกแบบในรูปที่ 2.1 เช้ากับขาของ CPLD จะได้ดังนี้

$$\begin{array}{ll} \text{CLK} = \text{PB1} = \text{INPUT} = \text{p39} & \text{Q1} = \text{LED1 (L1)} = \text{OUTPUT} = \text{p38} \\ \text{CLR} = \text{PB2} = \text{INPUT} = \text{p40} & \text{Q0} = \text{LED2 (L2)} = \text{OUTPUT} = \text{p37} \end{array}$$

ขั้นตอนการกำหนดสายสัญญาณเข้ากับขา CPLD ให้พิมพ์รายละเอียดต่างๆ ใน Edit Constraints (Text) ดังนี้

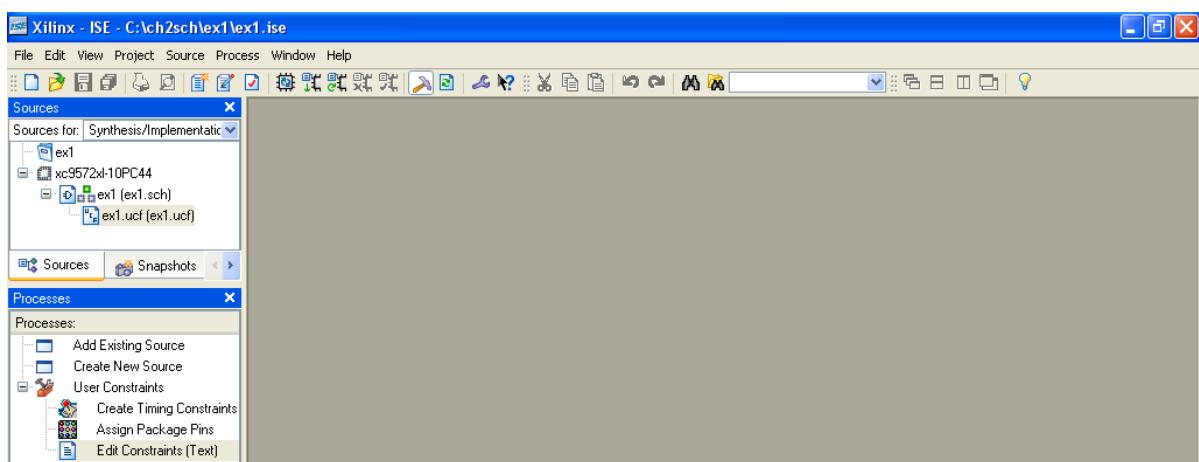
```
NET "CLK" LOC = "p39";
NET "CLR" LOC = "p40";
NET "Q1" LOC = "p38" | Slew = Slow ;
NET "Q0" LOC = "p37" | Slew = Slow ;
```

เมื่อพิมพ์เสร็จแล้วจะได้ดังรูปที่ 2.40 จากนั้นบันทึกไฟล์โดยคลิก คลิก (สีดำ) ที่มุมบนขวาในรูปที่ 2.40 เพื่อปิดโปรแกรม Edit Constraints (Text) และกลับไปที่หน้าต่าง Xilinx-ISE ดังรูปที่ 2.41



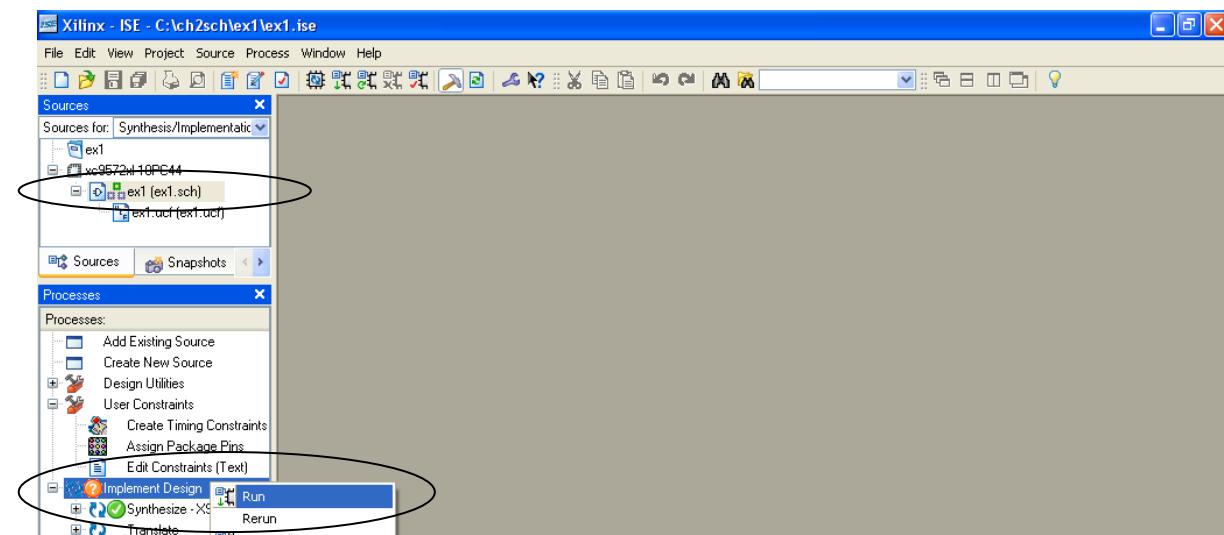
รูปที่ 2.40 การกำหนดขาสัญญาณเข้ากับขาชิป CPLD ใน Edit Constraints (Text)

- หมายเหตุ**
- 1) เรากำหนด Slew = Slow เฉพาะ I/O ที่เป็นขาเอาต์พุตเท่านั้น เพื่อลดสัญญาณรบกวนข้ามช่องและกราวเดบาร์
 - 2) สัญลักษณ์ | จะตรงกับคีย์บอร์ดตัวอักษร “ศ” และการใช้วิธี Copy และ Paste ช่วยเพื่อทำให้พิมพ์ໄลเร็วขึ้น

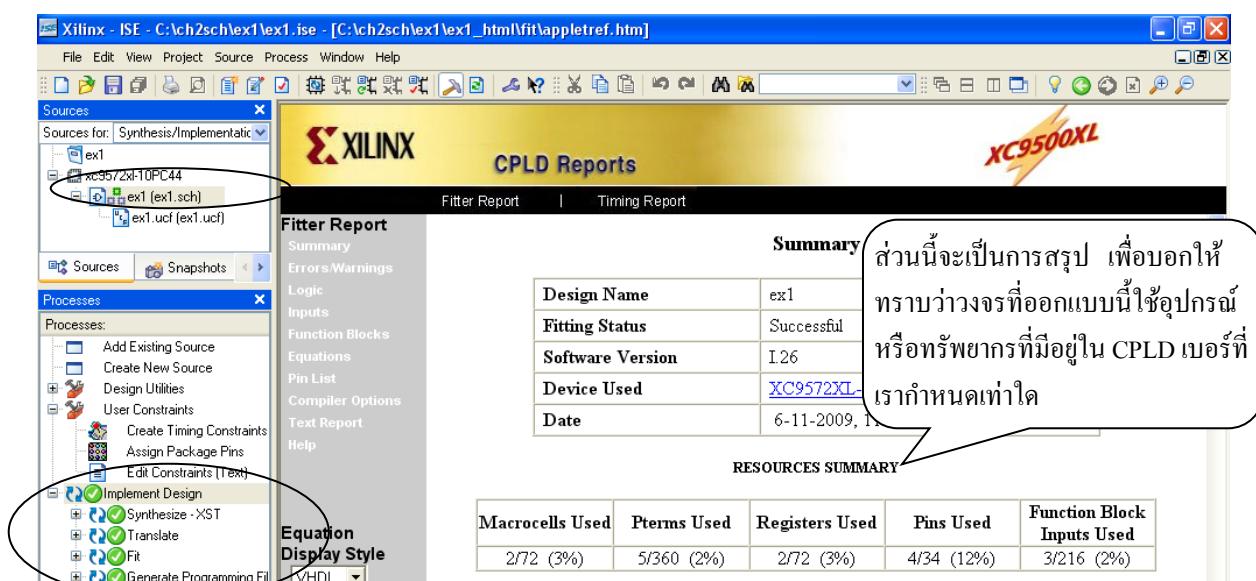


รูปที่ 2.41 หน้าต่าง Xilinx-ISE

3) ขั้นตอน Implement Design ให้คลิกไฟล์ ex1(ex1.sch) ในหน้าต่าง Source แล้วคลิก “+” ที่หน้า Implement Design จนเป็น “-” และให้คลิกขวาที่ Implement Design ในหน้าต่าง Processes แล้วคลิก Run ดังรูปที่ 2.42 (หรือดับเบิลคลิก) ถ้าใช่ ✓ ดังรูปที่ 2.43 หรือ ⚠ ถ้า Implement ผ่าน แต่ถ้าใช่ ✗ หน้า Implement Design ถือว่า Implement ไม่ผ่าน และถ้าใช่ ✗ หน้า Translate ถือว่า Translate ไม่ผ่าน ก็ให้ปิด Edit Constraints (Text) ตรวจสอบใหม่อีกรอบว่ามีการกำหนดขา CPLD ครบถ้วนหรือเกินหรือขาดหรือพิมพ์ผิดพลาดหรือไม่ แล้วบันทึกไฟล์ แล้วทำ Implement ซ้ำอีกรอบ ในขั้นตอน Implement Design นี้ซอฟต์แวร์ทุกจะทำขั้นตอนย่อยคือ Synthesize (ทำซ้ำ) Translate, Fit และ Generate Programming File ในขั้นตอนเดียว



รูปที่ 2.42 การทำ Implement Design



รูปที่ 2.43 ขั้นตอนเมื่อทำ Implementation เสร็จเรียบร้อยแล้ว

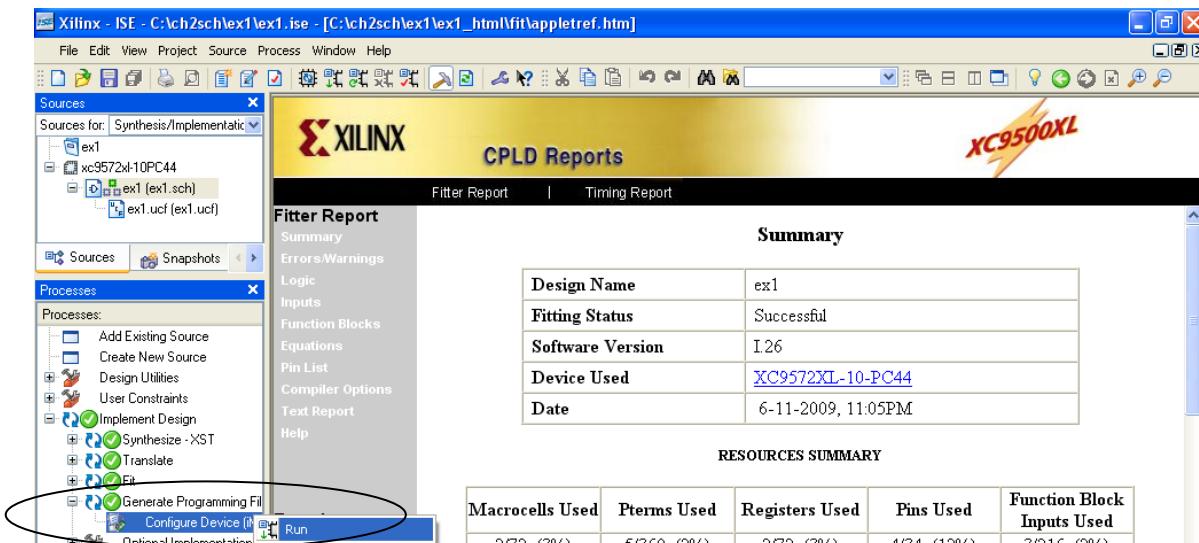
2.1.5 การโปรแกรมข้อมูลวงจรลงชิพ

การทำ Generate Programming File ในของกรณี CPLD นั้นซอฟต์แวร์ทุกได้ทำไปแล้วในขั้นตอน Implement Design จากนั้นให้นำข้อมูลวงจรมาดาวน์โหลดลง CPLD โดยใช้สาย JTAG และดังรูปที่ 2.44 ซึ่งมีขั้นตอนดังนี้

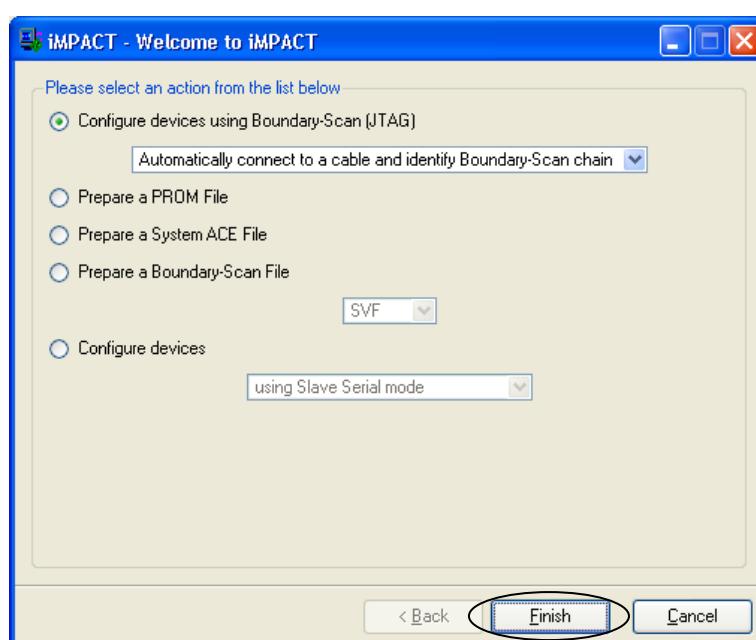
- 1) ต่อสาย JTAG เข้ากับพอร์ตบนนанคอมพิวเตอร์และข้าม JTAG พร้อมทั้งจ่ายไฟเลี้ยงเข้าบอร์ด CPLD Explorer XC9572XL
- 2) คลิก “+” หน้า Generate Programming File จะเป็น “-” คลิกขวาและคลิก Run ที่ Configure Device (iMPACT) ดังรูปที่ 2.45 รอซักครู่แล้วจะได้หน้าต่าง iMPACT-Welcome to iMPACT ดังรูปที่ 2.46 คลิก Finish และจะได้หน้าต่างดังรูปที่ 2.47



รูปที่ 2.44 ตัวอย่างความโน้มถ่วงในโหมด JTAG (สาย JTAG)

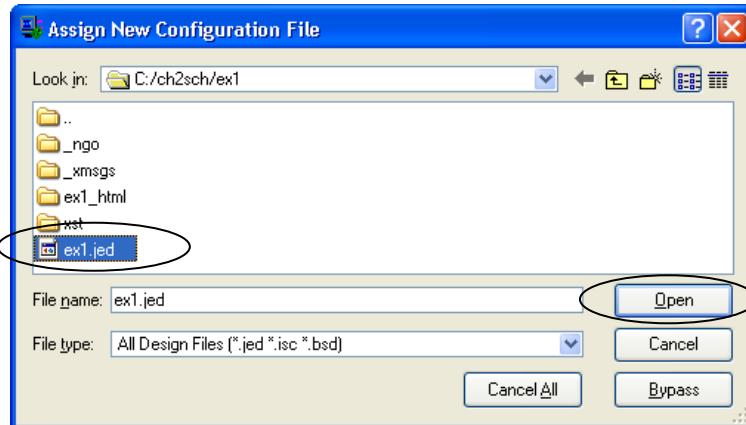


รูปที่ 2.45 ขั้นตอน Configure Device (iMPACT)

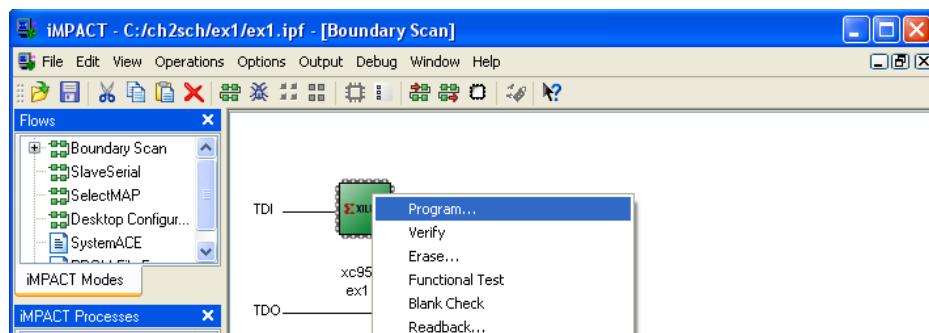


รูปที่ 2.46 หน้าต่าง iMPACT-Welcome to iMPACT

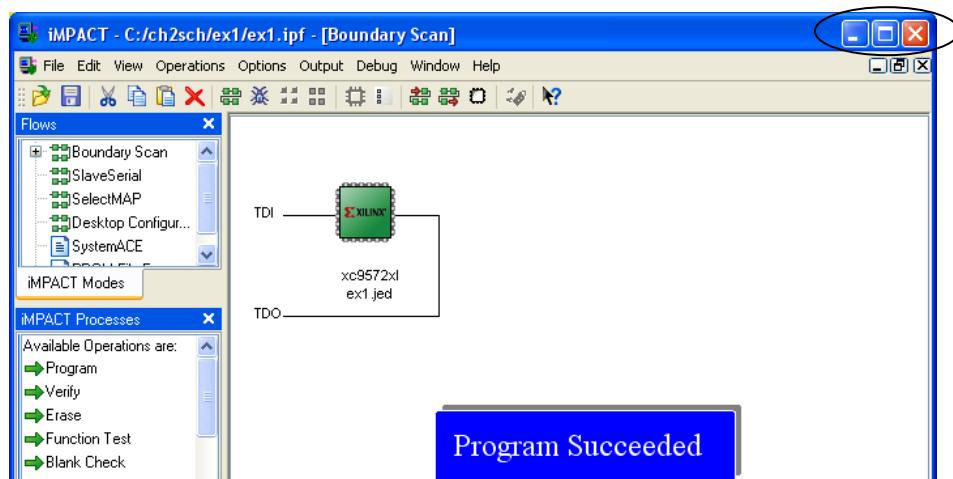
3) ที่รูปที่ 2.47 คลิกที่ ex1.jed แล้วคลิก Open จะได้หน้าต่าง iMPACT คลิกขวาที่ตัวไอซี (สีเขียว) แล้วคลิกที่選項 Program.. ดังรูปที่ 2.48 แล้วจะได้หน้าต่าง Programming Properties จากนั้นให้คลิก OK แล้วก็จะเป็นการเริ่มดาวน์โหลดข้อมูลลง CPLD เสร็จแล้วจะได้ดังรูปที่ 2.49 ในกรณีที่ดาวน์โหลดแล้วไม่ผ่านนั้นเพื่อความแน่ใจให้ดาวน์โหลดซ้ำอีกรอบ



รูปที่ 2.47 หน้าต่าง Assign New Configuration File

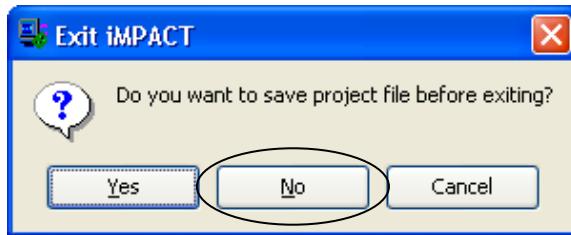


รูปที่ 2.48 หน้าต่าง iMPACT



รูปที่ 2.49 หน้าต่าง IMPACT

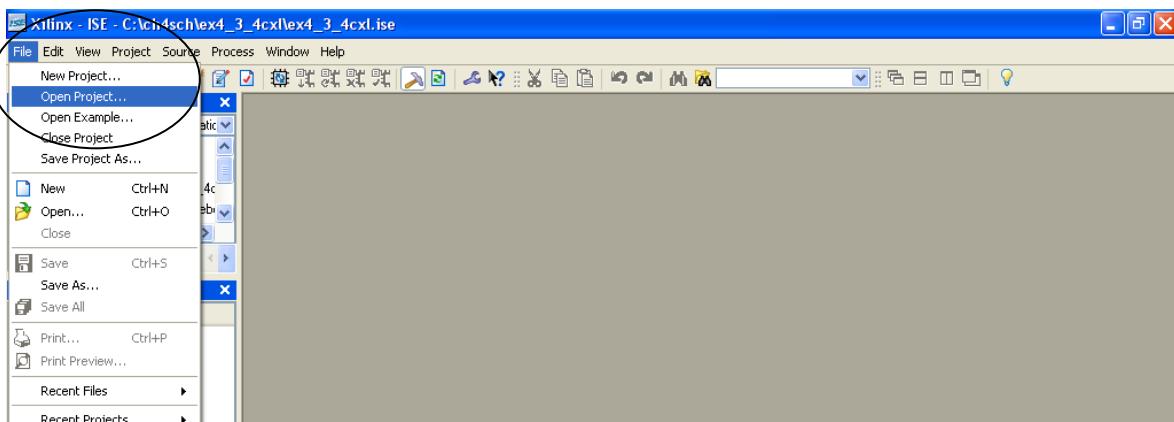
4) ให้ทดลองกดปุ่ม PB1 และ PB2 แล้วสังเกตที่ LED1 (L1) และ LED2 (L2) ว่าให้ผลตามที่ออกแบบหรือไม่ เสร็จแล้วปิดหน้าต่าง iMPACT โดยคลิก (สีแดง) แล้วจะได้ดังรูปที่ 2.50 คลิก No เพื่ออกจาก iMPACT จากนั้นคลิก (สีแดง) เพื่อปิดโปรแกรม ISE WebPACK 8.1i



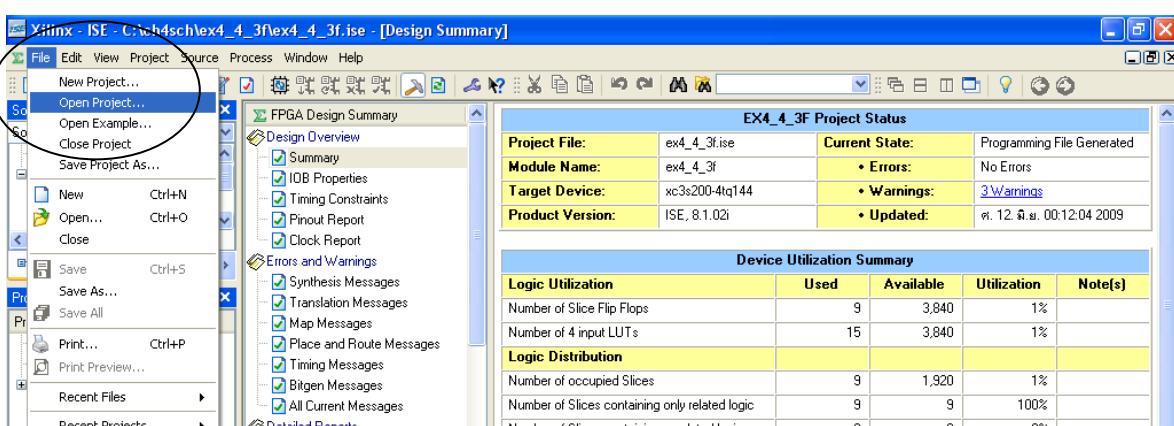
รูปที่ 2.50 หน้าต่าง Exit iMPACT

2.1.6 การเปิดไฟล์ที่เคยออกแบบไว้แล้วมาใช้งาน

- 1) เช่นถ้าต้องการเปิดไฟล์ชื่อ ex1 ที่มีอยู่ใน Folder ชื่อ ch2sch ให้เริ่มที่หน้าจอคอมพิวเตอร์แล้วให้ดับเบิลคลิกที่ แล้วจะได้หน้าต่าง Xilinx-ISE คลิก File -> Open Project ดังรูปที่ 2.51(a) หรือรูปที่ 2.51(b) แล้วจะได้หน้าต่าง Open Project จากนั้นคลิกไฟล์ ex1 แสดงดังรูปที่ 2.52 แล้วให้ดับเบิลคลิกที่ไฟล์ ex1 แล้วจะได้ดังรูปที่ 2.53
- 2) จากรูปที่ 2.53 เมื่อดับเบิลคลิกที่ไฟล์ ex1.ise ก็จะได้หน้าต่าง Xilinx-ISE ของไฟล์ ex1(ex1.sch) ดังรูปที่ 2.54

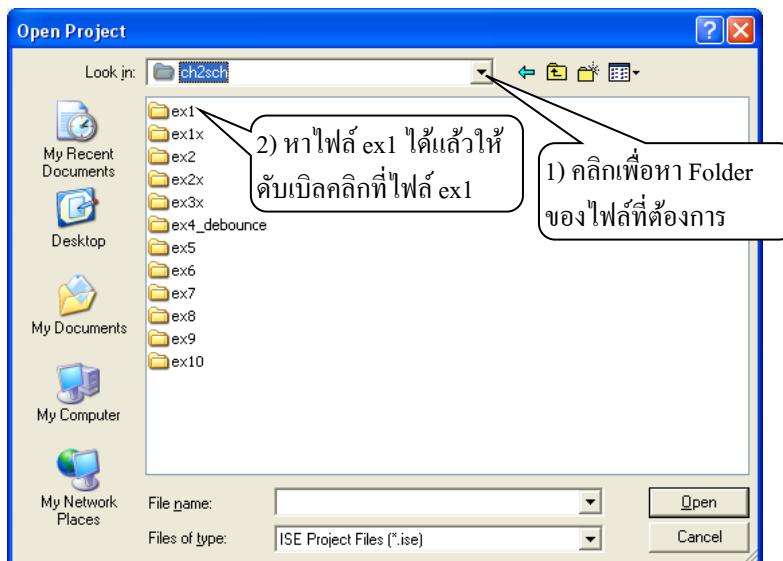


(a) หน้าต่าง Xilinx-ISE

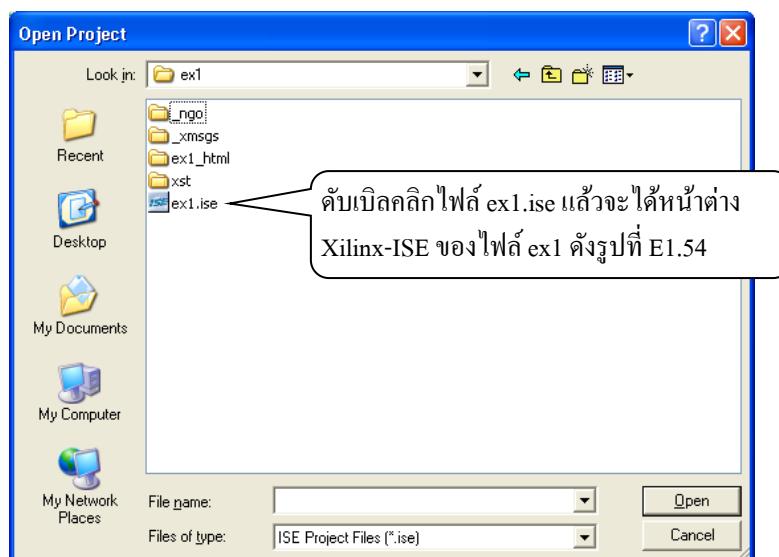


(b) หน้าต่าง Xilinx-ISE

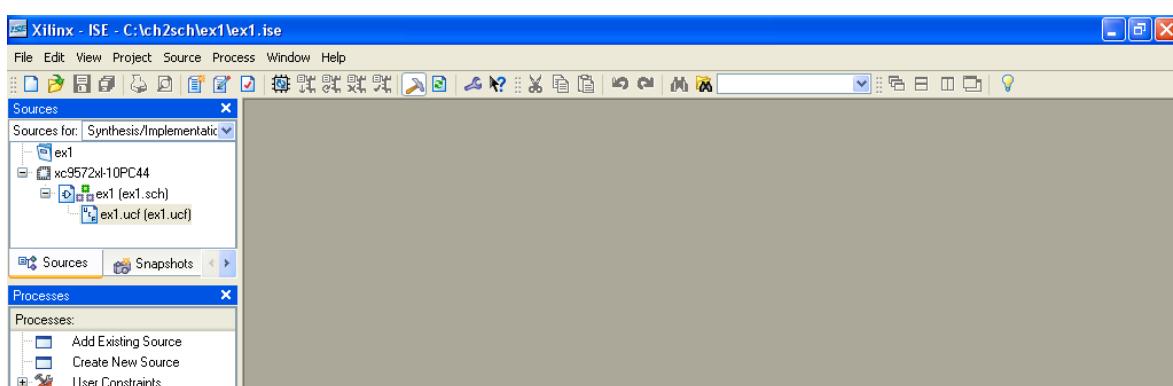
รูปที่ 2.51 หน้าต่าง Xilinx-ISE อาจจะเป็นรูปใดรูปหนึ่งก็ได้



รูปที่ 2.52 หน้าต่าง Open Project (ตำแหน่งไฟล์ ex1 จะไม่แน่นอน ขึ้นอยู่กับจำนวนไฟล์ใน Folder นั้น)

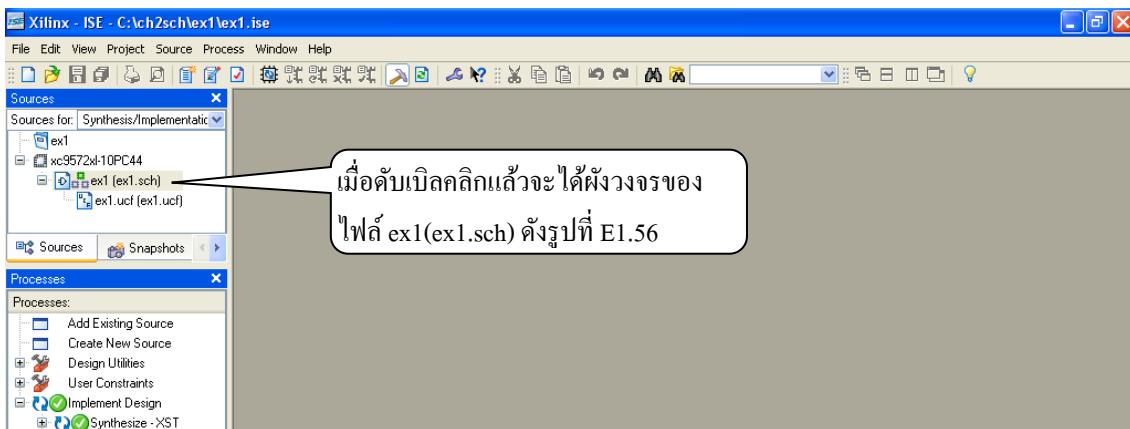


รูปที่ 2.53 หน้าต่าง Open Project

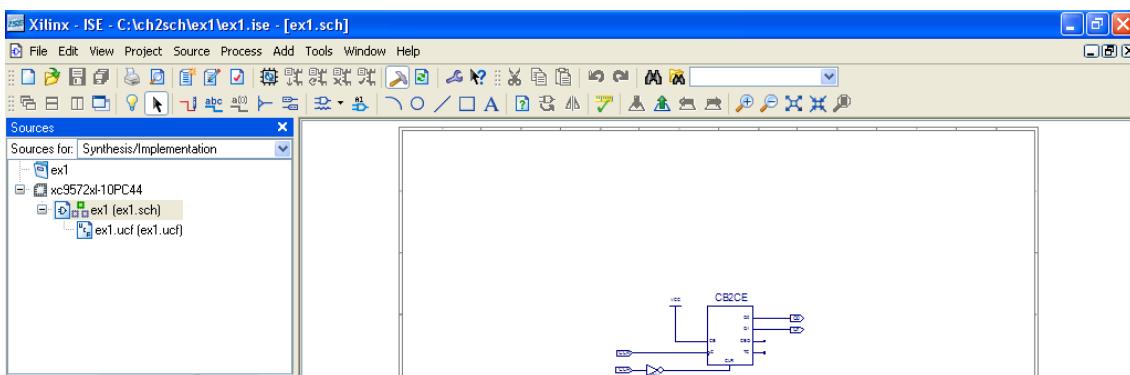


รูปที่ 2.54 หน้าต่าง Xilinx-ISE ของไฟล์ ex1

3) จากรูปที่ 2.54 คลิกที่ ex1(ex1.sch) ในหน้าต่าง Source แล้วคลิก “+” ที่อยู่หน้า Implement Design ในหน้าต่าง Processes เป็น “-” แล้วจะได้ดังรูปที่ 2.55 ซึ่งเป็นหน้าต่าง Xilinx-ISE ของไฟล์ ex1 เหมือนกับตอนที่เราออกแบบทุกประการ ถ้าเราต้องการดูรูปผังวงจรของไฟล์ ex1 ก็ให้ดับเบิลคลิกที่ ex1(ex1.sch) ในหน้าต่าง Source แล้วจะได้ดังรูปที่ 2.56



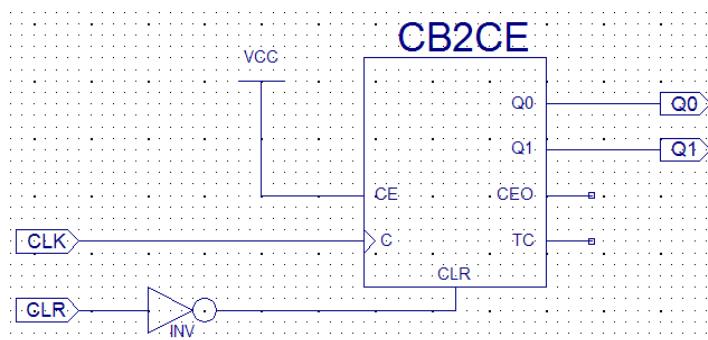
รูปที่ 2.55 หน้าต่าง Xilinx-ISE ของไฟล์ ex1



รูปที่ 2.56 หน้าต่าง Xilinx-ISE สำหรับ Schematic editor ของไฟล์ ex1

2.2 การออกแบบวงจรนับ 4 แบบชิงโครนัสโดยใช้ FPGA ด้วยวิธี Schematic

ตัวอย่างที่ 2.2 ออกแบบวงจรนับ 4 แบบชิงโครนัสโดยใช้ FPGA แสดงดังรูปที่ 2.57 นั้นจะมีขั้นตอนต่างๆ เกือบจะเหมือนกับการออกแบบด้วย CPLD ในตัวอย่างที่ 2.1 ทุกประการ ผู้อ่านควรทำความเข้าใจในตัวอย่างที่ 2.1 ก่อน (ไฟล์ตัวอย่างบทที่ 2 บทที่ 3 และบทที่ 4 อยู่ในแผ่น DVD หากต้องการใช้ ให้ Copy Folder ไฟล์ไปไว้ในไดรฟ์ C) ขั้นตอนการออกแบบเป็นดังนี้

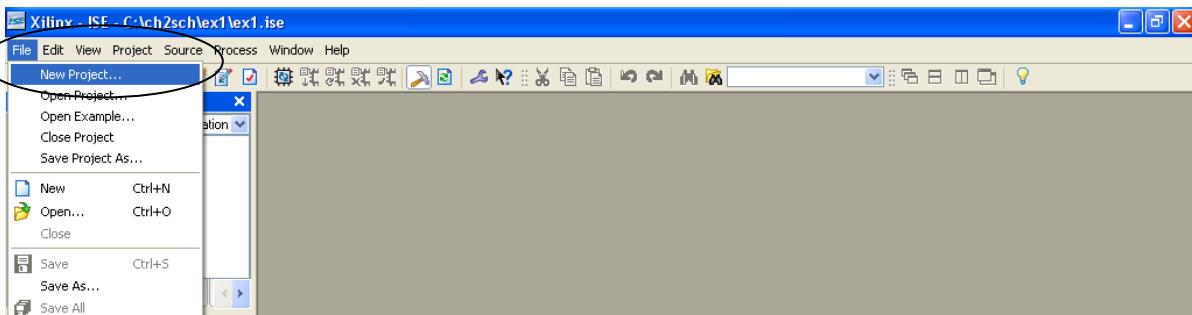


รูปที่ 2.57 วงจรนับ 4 แบบชิงโครนัสโดยใช้ FPGA

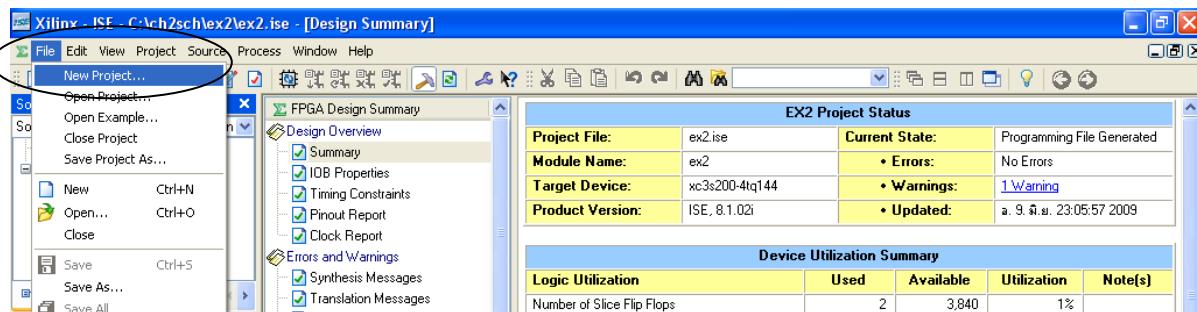
2.2.1 ขั้นตอนการออกแบบวงจร (Design Entry)

- สร้างไฟล์ไว้ใน Folder ชื่อ ch2sch ในไดรฟ์ C (หากยังไม่สร้าง Folder นี้ไว้ในข้อ 2.1) การใช้ ISE WebPACK ให้เริ่มที่หน้าจอคอมพิวเตอร์ ดับเบิลคลิกที่ แล้วจะได้หน้าต่าง Xilinx-ISE (ถ้ามีหน้าต่าง Tip of the Day ขึ้นมาให้คลิกที่

ปุ่ม OK) จากนั้นคลิก File -> New Project อาจจะเป็นดังรูป 2.58(a) หรือรูปที่ 2.58(b) แล้วจะได้หน้าต่าง New Project Wizard



(a) หน้าต่าง Xilinx-ISE



(b) หน้าต่าง Xilinx-ISE

รูปที่ 2.58 หน้าต่าง Xilinx-ISE (ความละเอียดภาพ 1024 x 768 pixels) อาจจะได้ดังรูปที่ 2.58(a) หรือรูปที่ 2.58(b)

2) ที่หน้าต่าง New Project Wizard-Create New Project พิมพ์ชื่อ C:\ch2sch ในช่อง Project Location (ชื่อ Folder) แล้วพิมพ์ชื่อ ex2 ที่ช่อง Project Name คลิกที่ Top-Level Source Type เป็น Schematic ดังรูปที่ 2.59 คลิก Next แล้วจะได้หน้าต่างถัดไป

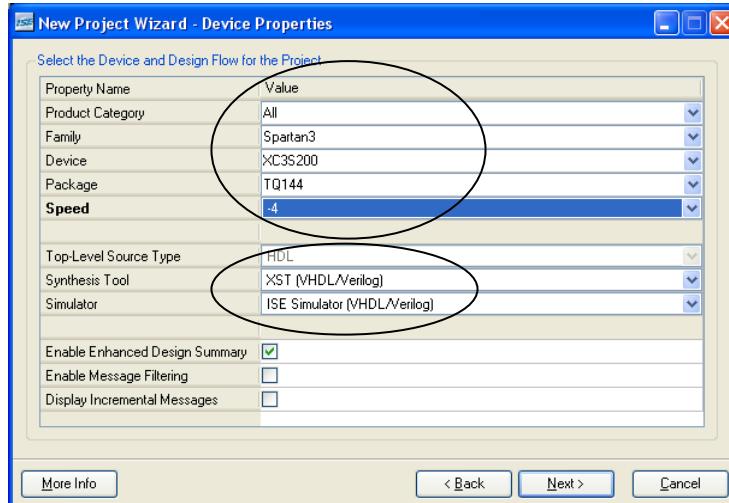
หมายเหตุ การตั้งชื่อต้องเป็นตัวอักษร A-Z หรือ a-z และห้ามมี (_) ติดกัน ห้ามจบด้วย (_) และห้ามเว้นช่องว่างระหว่างตัวอักษร



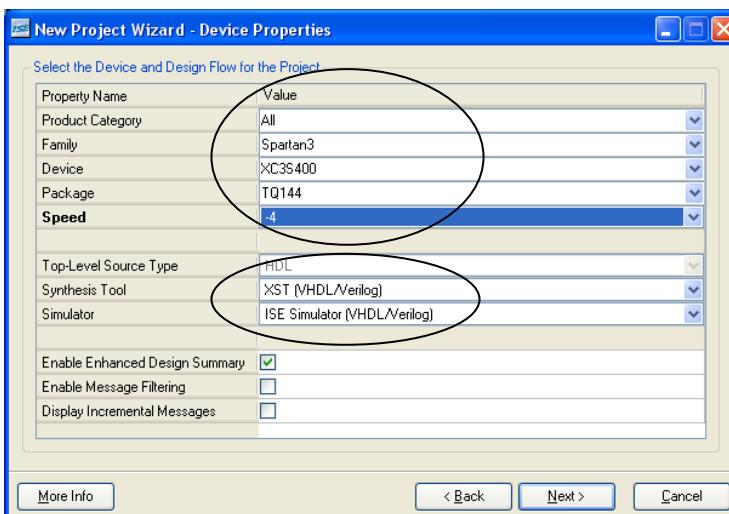
รูปที่ 2.59 หน้าต่าง New Project Wizard- Create New Project

3) ที่หน้าต่าง New Project Wizard-Device Properties ถ้าใช้ FPGA Discovery-III XC3S200F คลิกดังรูปที่ 2.60(a) คือ FPGA ตระกูล (Family) Spartan3, เบอร์ (Device) XC3S200, Package แบบ 144 ขา (Package:TQ144) และ Speed Grade : -4 แต่ถ้าใช้

FPGA Discovery-III XC3S200F4 คลิกดังรูปที่ 2.60b (เบอร์ XC3S400) เลือก Synthesis tool เป็น XST (VHDL/Verilog) และ Simulator เป็น ISE Simulator (VHDL/Verilog) จากนั้นคลิก Next ในรูปที่ 2.60a) (หรือรูปที่ 2.60b) และจะได้ดังรูปที่ 2.61

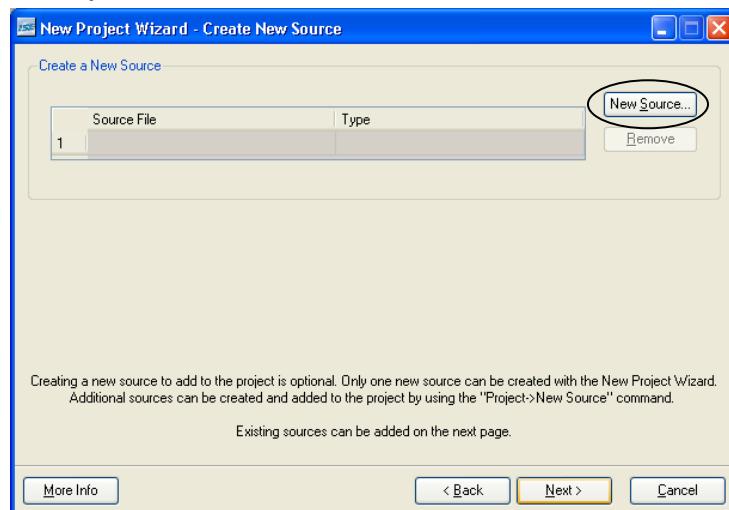


(a) ในการเลือก FPGA เบอร์ XC3S200-4TQ144C (ใช้บอร์ด FPGA Discovery-III XC3S200F)



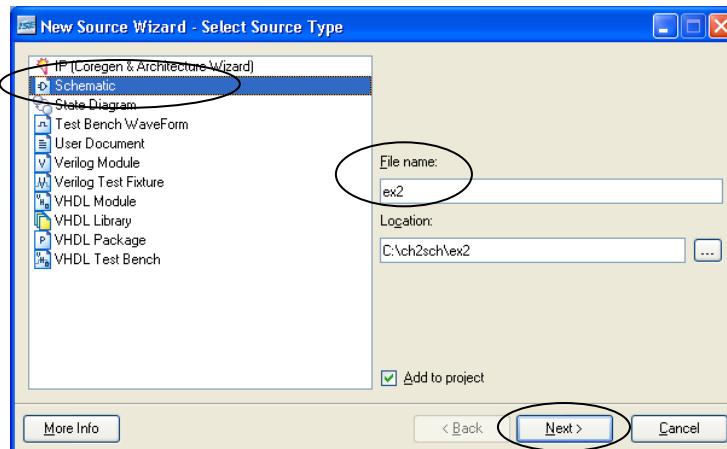
(b) ในการเลือก FPGA เบอร์ XC3S400-4TQ144C (ใช้บอร์ด FPGA Discovery-III XC3S200F4)

รูปที่ 2.60 หน้าต่าง New Project Wizard–Device Properties



รูปที่ 2.61 New Project Wizard-Create New Source

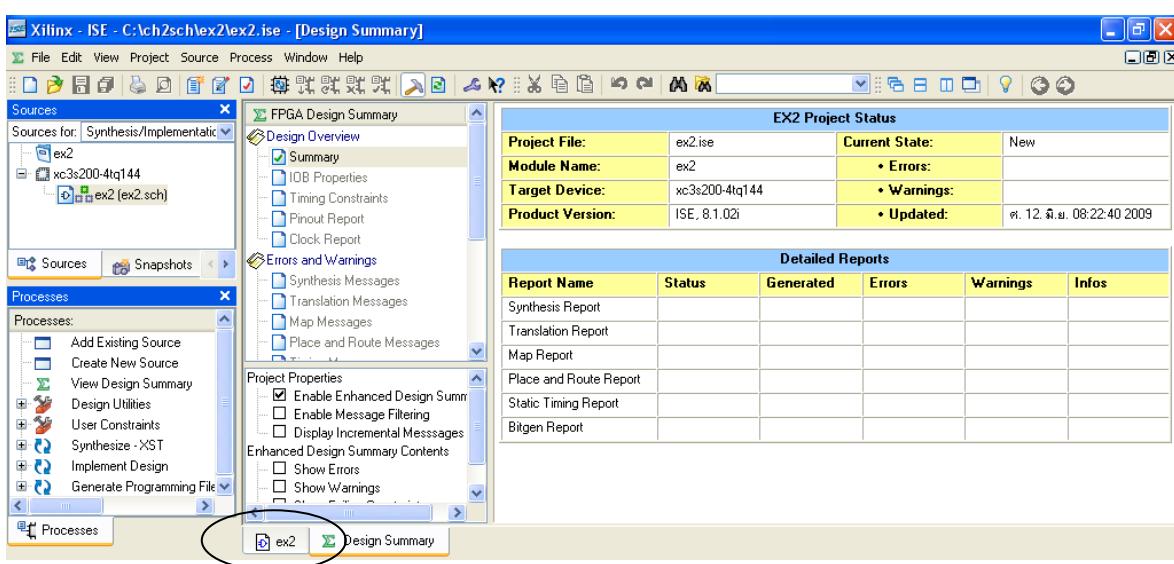
- 4) จากรูปที่ 2.61 คลิกปุ่ม New Source แล้วจะได้หน้าต่างดังที่ไป พิมพ์ชื่อ ex2 ในช่อง File Name และคลิกที่ Schematic ดังรูปที่ 2.62 คลิก Next คลิก Finish และคลิก Next แล้วจะได้หน้าต่างดังรูปที่ 2.63 (อธิบายในภายหลัง) เสร็จแล้วคลิก Next คลิก Finish และคลิกแท็บ ex2 จะได้หน้าต่าง Xilinx-ISE สำหรับวางแผน (โปรแกรม Schematic editor หรือ ECS) ดังรูปที่ 2.64



รูปที่ 2.62 หน้าต่าง New Source Wizard-Select Source Type

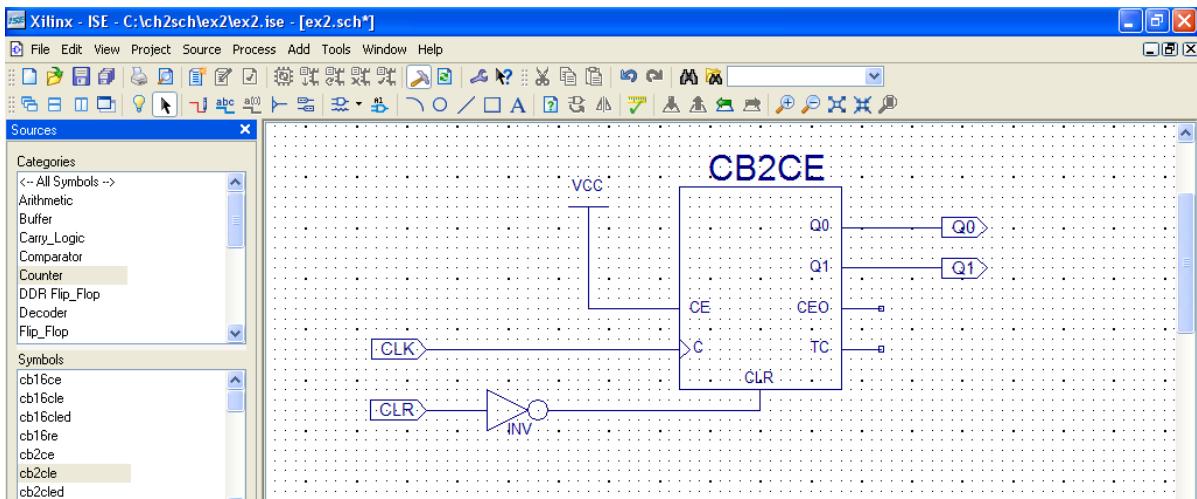


รูปที่ 2.63 หน้าต่าง New Project Wizard-Add Existing Source



รูปที่ 2.64 หน้าต่าง Xilinx-ISE

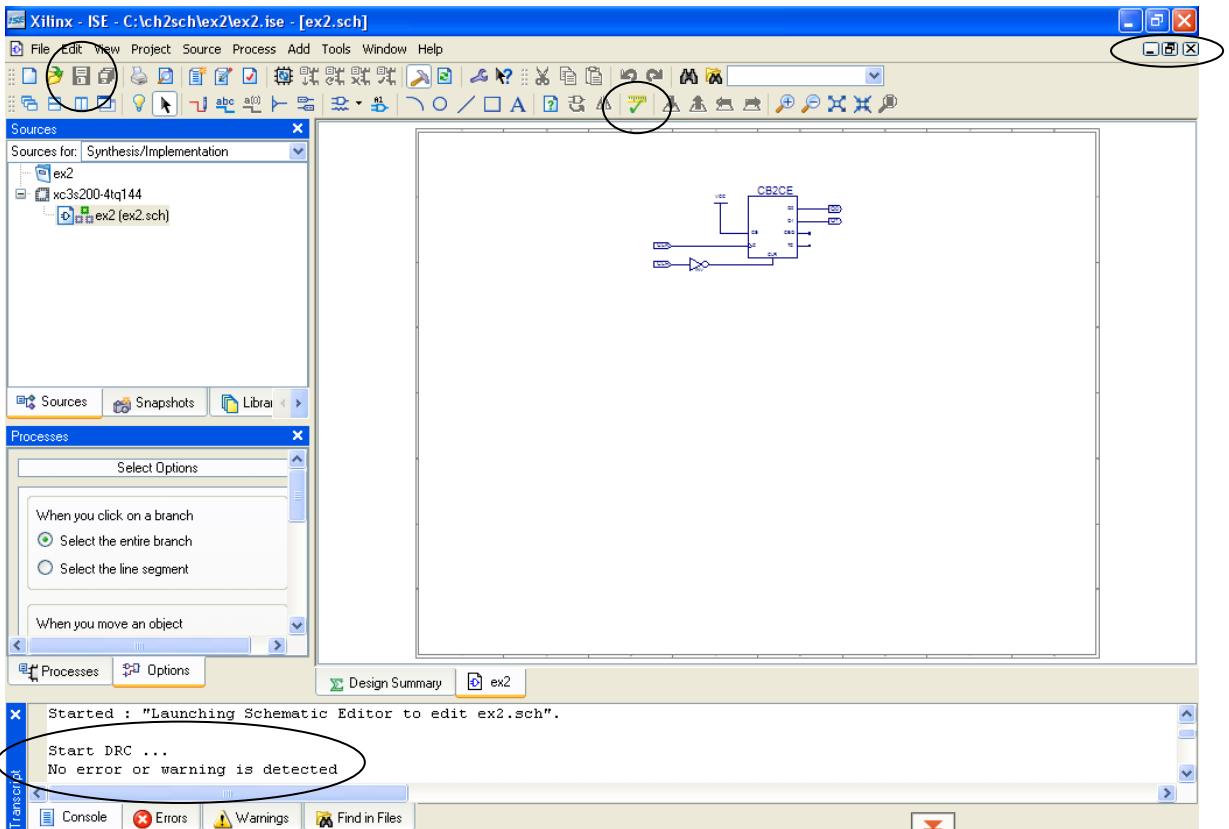
5) ทำการตรวจสอบวงจรรูปที่ 2.57 โดยมีขั้นตอนต่างๆ เมื่อมองข้อ 2.1.1 ในตัวอย่างที่ 2.1 ทุกประการ เสรีจແລ້ວຈະได้ดังรูปที่ 2.65



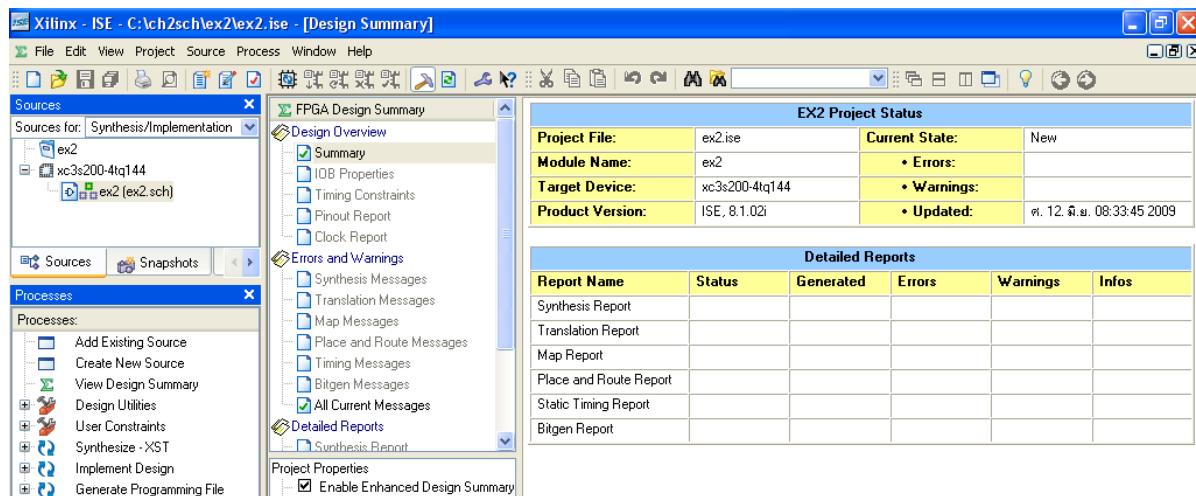
รูปที่ 2.65 ผังวงจรที่ແລ້ວເສື່ອງສາມນູຽນ

6) การตรวจความถูกต้อง (Syntax) ทำໄດ້ໂດຍຄລິກ ທ່ານີ້ມີຂໍ້ຜິດພາດຈະປາກູ້ຂໍ້ຄວາມ No error or warning is detected ດັ່ງຮູບທີ 2.66 ແລ້ວບັນທຶກໄຟໂດຍຄລິກ ໃນຂັ້ນຕອນນີ້ເຊື່ອວ່າກາຮອກແບບວາງເສື່ອງສາມນູຽນແລ້ວ ຄລິກ (ສິດຳ) ປຶກໂປຣແກຣມ Schematic editor ເພື່ອກລັນໄປທີ່ໜ້າຕ່າງ Xilinx-ISE ອີກຮັງ ຄລິກ View -> Restore Default Layout ທຳອອງເດີວັນກັນໃນຮູບທີ 2.9 (ໃນຕ້າວຍໆທີ 2.1) ແລ້ວຈະໄດ້ດັ່ງໃນຮູບທີ 3.67 ເພື່ອກາຮັດຂັ້ນຕອນດ່ອໄປ

7) ໃນກຣັບທີ່ມີຂໍ້ຜິດພາດ ເມື່ອຄລິກ ແລ້ວຈະແສດງຂໍ້ຜິດພາດທີ່ໜ້າຕ່າງ Transcript (ລ່າງສຸດ) ແລ້ວໃຫ້ກາຮັດແກ້ໄຂໃນທຳອອງເດີວັນກັນກັນກາຮອກແບບດ້ວຍ CPLD ໃນຕ້າວຍໆທີ 2.1



ຮູບທີ 2.66 ກາຮັດສອບຄວາມຄຸກຕົ້ງ



รูปที่ 2.67 หน้าต่าง Xilinx-ISE ของโครงการไฟล์ชื่อ ex2

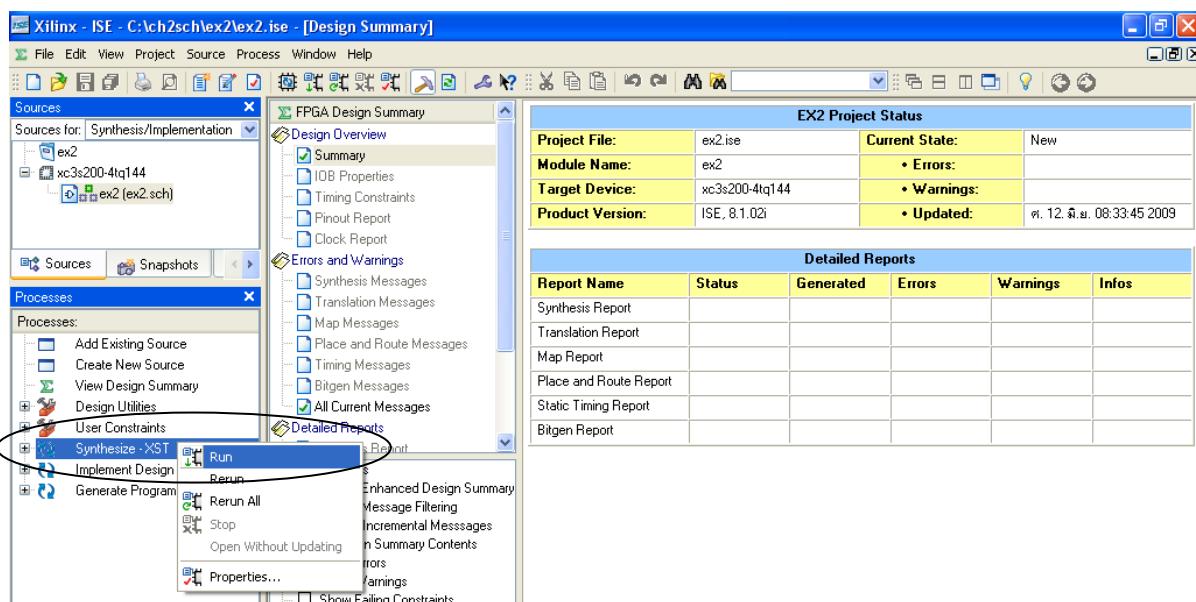
2.2.2 การตรวจสอบความถูกต้องของจริงที่ออกแบบ (Design Verification)

การออกแบบวงจรอาจไม่มีความจำเป็นต้องทำขั้นตอนนี้ก็ได้ ซึ่งรายละเอียดจะอธิบายในภายหลัง

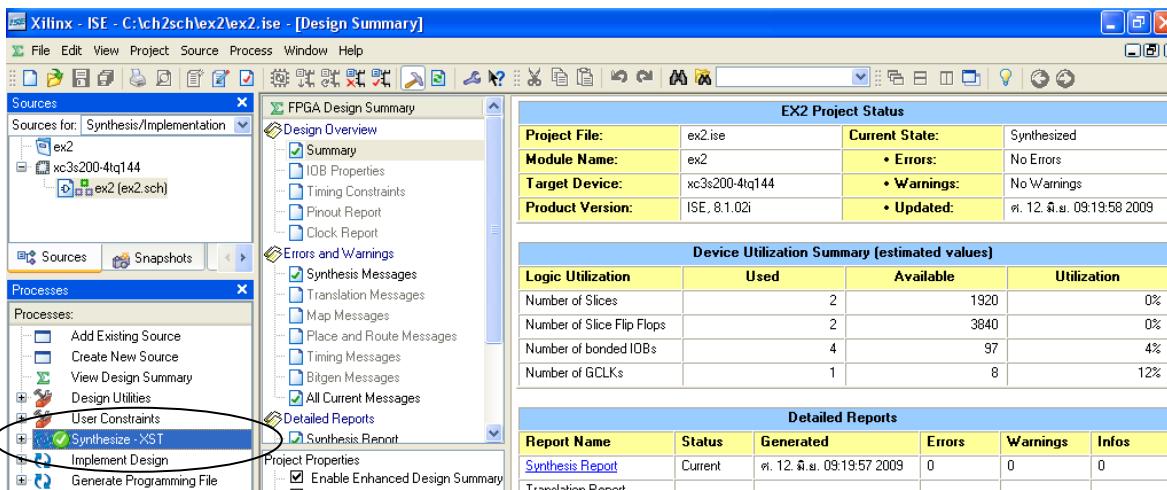
2.2.3 การสังเคราะห์วงจร (Design synthesis)

การสังเคราะห์วงจร คลิกขวาแล้วคลิก Run ที่ Synthesize-XST (หรือดับเบิลคลิก) ที่หน้าต่าง Processes ดังรูปที่ 2.68 แล้วถ้าใช่ ✓ หรือ ! ที่หน้า Synthesize-XST ดังรูปที่ 2.69 ก็ให้ถือว่าสังเคราะห์วงจรผ่าน ขอให้สังเกตว่าในกรณี FPGA นั้น จะต่างจากการมี CPLD ตรงที่ Synthesize-XST อยู่นอก Implement Design เพื่อเปิดโอกาสให้เราใช้ Synthesis tool ตัวอื่นได้

หมายเหตุ Process status icon มีดังนี้คือ = Running, = Up-to-date (ถือว่าผ่าน), = Warnings Reported (ถือว่าผ่าน), = Errors Reported(ถือว่าไม่ผ่าน), = Out-of-Date (คือไม่มีอัปเดตแล้ว ต้องคลิก RUN ใหม่อีกรอบ)



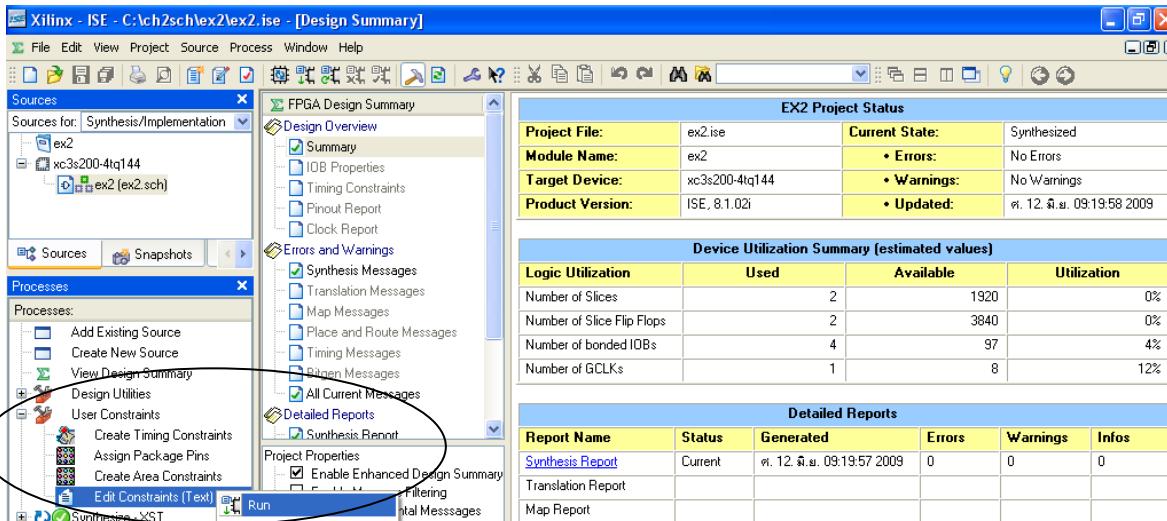
รูปที่ 2.68 แสดงขั้นตอนสังเคราะห์วงจรในหน้าต่าง Processes



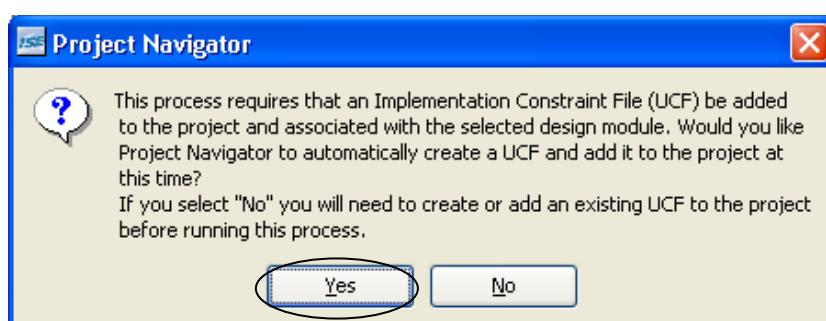
ຮູບທີ 2.69 ແສດໜ້າຕ່າງ Processes ເມື່ອສັງເກຣະຫົວຂະໜາດ

2.2.4 Design Implementation

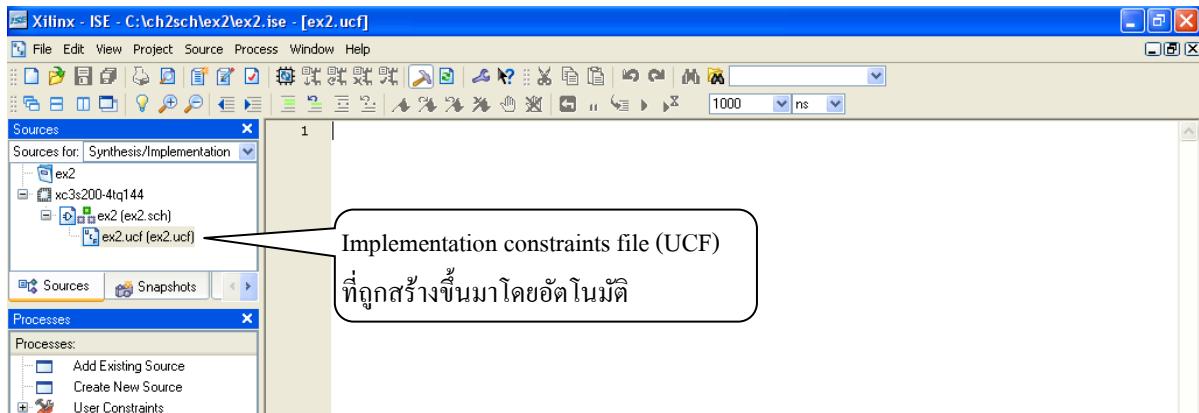
1) ຫຼັນຄອນສ່ວັງ Implementation constraints file ອີ່ວ່າ User constraints file(UCF) ໃນຫຼັນຄອນນີ້ເຮົາຈະສ່ວັງ User constraints file (UCF) ໂດຍອັດໂນມັດໄດ້ໂດຍຄົກ “+” ທີ່ອໝ່າໜ້າ User Constraints ໃນຫຼັນຕ່າງ Processes ຈະເປັນ “-” ຈາກນັ້ນຄົກຂວາແລ້ວຄົກ Run ທີ່ Edit Constraints (Text) ດັ່ງຮູບທີ 2.70 (ຫຼືອັນເປີຄົກຄືກໍໄດ້) ຈາກນັ້ນໄທ້ຄົກ Yes ໃນຮູບທີ 2.71 ເພື່ອສ່ວັງ User constraints file (UCF) ໂດຍອັດໂນມັດຕັ້ງໃນຮູບທີ 2.72 ຜົ່ງໃນຫຼັນຕ່າງ Sources ຈະມີໄຟລ໌ ex2.ucf ເພີ່ມເຂົ້າມາ



ຮູບທີ 2.70 ການສ່ວັງ Implementation constraints file ໂດຍອັດໂນມັດ

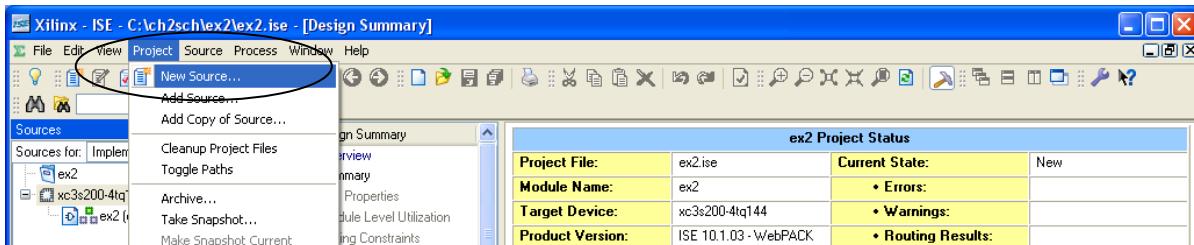


ຮູບທີ 2.71 ການເຢັນເຫັນເພື່ອສ່ວັງ Implementation constraints file ໂດຍອັດໂນມັດ

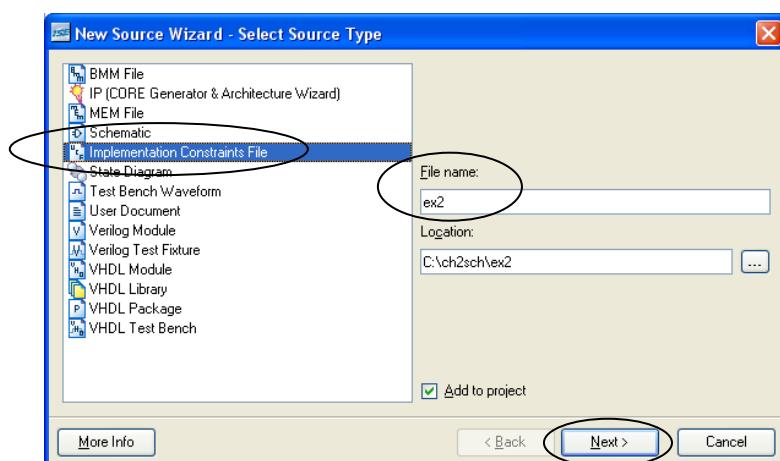


รูปที่ 2.72 หน้าต่างที่ใช้กำหนดขา FPGA ใน Edit Constraints (Text)

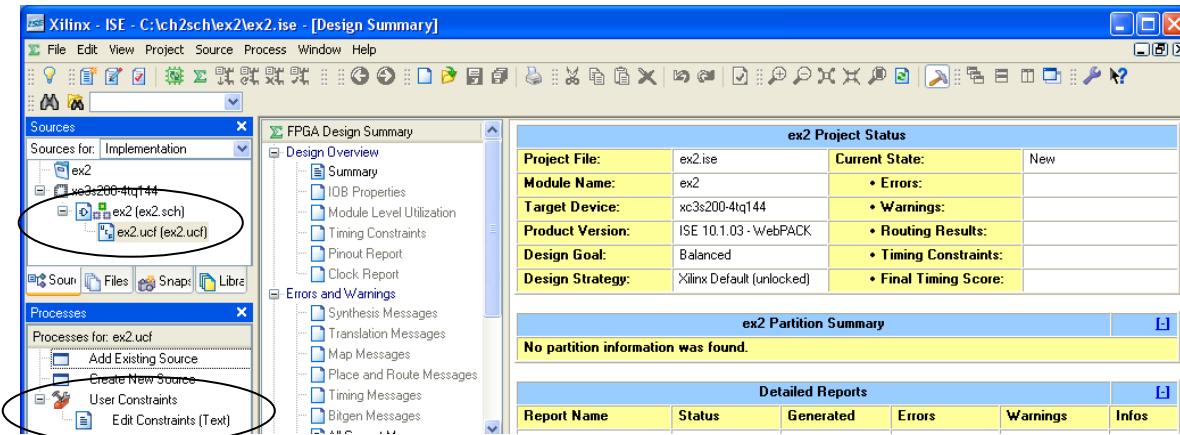
สำหรับคนที่ใช้ออฟต์แวร์ทุก ISE WebPACK 10.1i นั้นควรใช้วิธีการสร้าง Implementation constraints file หรือ User constraints file (UCF) โดยเริ่มที่หน้าต่าง Xilinx-ISE คลิก Project -> New Source ดังรูปที่ 2.73 และจะได้หน้าต่าง New Source Wizard-Select Source Type จากนั้นพิมพ์ชื่อ ex2 ลงในช่อง File name และคลิกที่ Implementation constraints file ดังรูปที่ 2.74 คลิก Next คลิก Finish และคลิก “+” หน้า ex2(ex2.sch) ในหน้าต่าง Source จะเป็น “-” และจะได้ ex2.ucf(ex2.ucf) ซึ่งเป็นไฟล์ UCF เพิ่มเข้ามา จากนั้นคลิก “+” หน้า User Constraints ในหน้าต่าง Processes จะเป็น “-” และจะได้ดังรูปที่ 2.75 จากนั้นคลิกขวาแล้วคลิก Run ที่ Edit Constraints (Text) ดังรูปที่ 2.76 หรือดับเบิลคลิก แล้วจะได้ผลเช่นเดียวกับรูปที่ 2.72



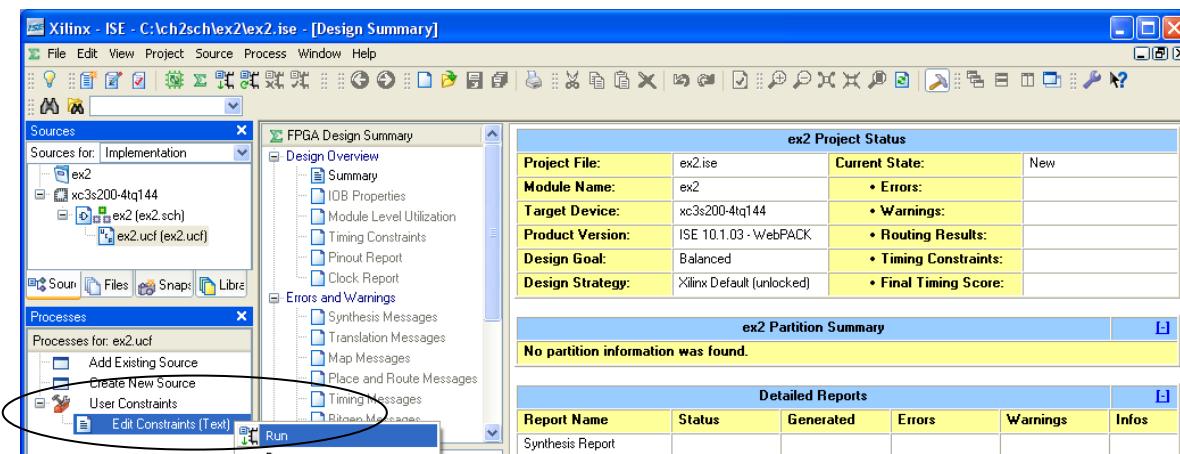
รูปที่ 2.73 หน้าต่าง Xilinx-ISE



รูปที่ 2.74 หน้าต่าง New Source Wizard-Select Source Type

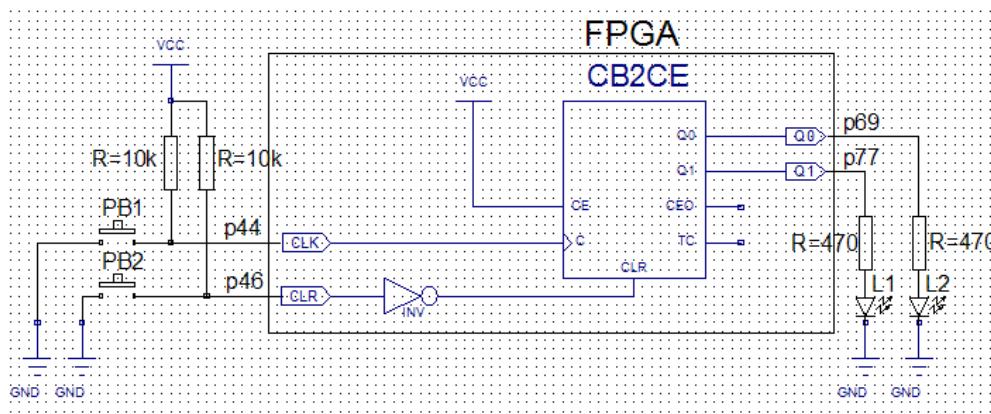


รูปที่ 2.75 หน้าต่าง Xilinx-ISE หลังจากสร้างไฟล์ UCF เพิ่มเข้าไปแล้ว



รูปที่ 2.76 หน้าต่าง Xilinx-ISE ขณะคลิกขวาแล้วคลิก Run ที่ Edit Constraints (Text) เพื่อสร้างไฟล์ UCF

2) ขั้นตอนกำหนดสายสัญญาณต่างๆ ของวงจรรูปที่ 2.57 เข้ากับขาของ FPGA นั้นจำเป็นที่จะต้องกำหนดอินพุตและเอาต์พุตให้สอดคล้องกับอุปกรณ์ที่เตรียมไว้ที่ FPGA Discovery-III XC3S200F ตามตารางที่ 1.4 ของบทที่ 1 โดยที่เรากำหนดให้ปุ่มกด PB1 และ PB2 จะต่ออยู่กับขา p44 และ p46 ตามลำดับ ส่วน LED L1 และ L2 ต่ออยู่กับขา p77 และ p69 แสดงดังรูปที่ 2.77



รูปที่ 2.77 การต่อ I/O ของ FPGA (เฉพาะขาที่ใช้งาน) กับอุปกรณ์ที่อยู่บนบอร์ดทดลอง (โดยไม่แสดง Vcc และ Gnd)

ดังนั้นการกำหนดสายสัญญาณต่างๆ ของวงจรที่ออกแบบในรูปที่ 2.57 เข้ากับขาของ FPGA จะได้ดังนี้

$$\text{CLK} = \text{PB1} = \text{INPUT} = \text{p44}$$

$$\text{Q1} = \text{L1} = \text{OUTPUT} = \text{p77}$$

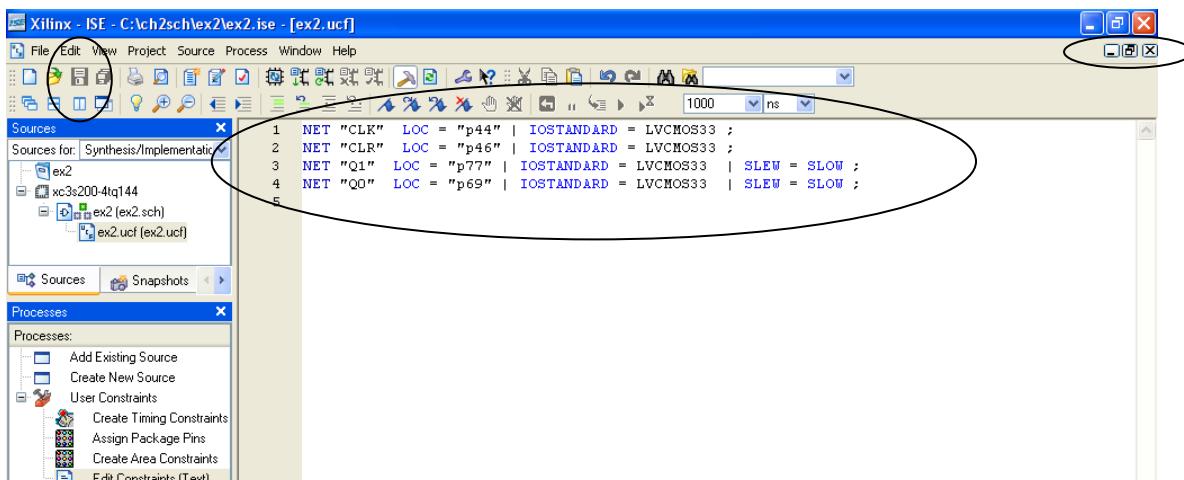
$$\text{CLR} = \text{PB2} = \text{INPUT} = \text{p46}$$

$$\text{Q0} = \text{L2} = \text{OUTPUT} = \text{p69}$$

ในกรณี FPGA นั้นเราอาจกำหนดให้ I/O เป็น Low Voltage CMOS 3.3V (LVCMOS33) หรือ Low Voltage TTL 3.3V (LVTTL) ได้ โดยที่ระดับแรงดันอินพุต/เอาต์พุต LVCMOS33 จะสูงกว่า LVTTL เล็กน้อย แต่สามารถเข้ากันได้หรือต่อ กันได้โดยตรง ขั้นตอนการกำหนดสายสัญญาณเข้ากับขา FPGA โดยพิมพ์รายละเอียดต่างๆ ใน Edit Constraints (Text) ดังนี้คือ

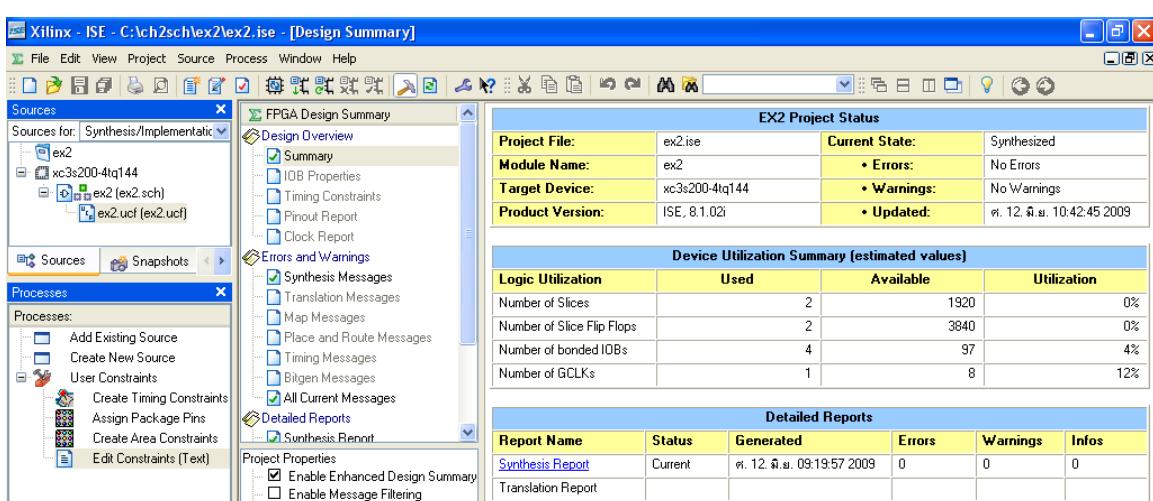
```
NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "Q1" LOC = "p77" | IOSTANDARD = LVCMOS33 | Slew = Slow ;
NET "Q0" LOC = "p69" | IOSTANDARD = LVCMOS33 | Slew = Slow ;
```

เมื่อพิมพ์เสร็จแล้วจะได้ดังรูปที่ 2.78 จากนั้นบันทึกไฟล์โดยคลิก คลิก (สีดำ) ที่มุมบนขวาในรูปที่ 2.78 เพื่อปิดโปรแกรม Edit Constraints (Text) และกลับไปที่หน้าต่าง Xilinx-ISE ดังรูปที่ 2.79



รูปที่ 2.78 การกำหนดขาสัญญาณเข้ากับขาชิป FPGA ใน Edit Constraints (Text)

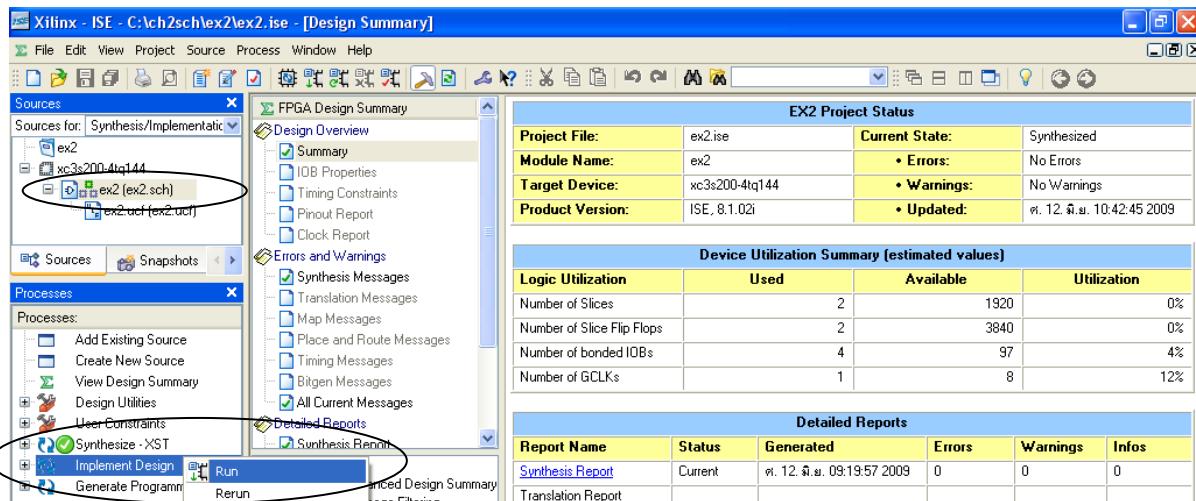
- หมายเหตุ**
- 1) เรากำหนด Slew = Slow เฉพาะ I/O ที่เป็นขาเอาต์พุตเท่านั้น เพื่อลดสัญญาณรบกวนข้ามช่องและรวดเร็วขึ้น
 - 2) สัญลักษณ์ | จะตรงกับคีย์บอร์ดตัวอักษร “כ” และการใช้วิธี Copy และ Paste ช่วยเพื่อทำให้พิมพ์ได้เร็วขึ้น



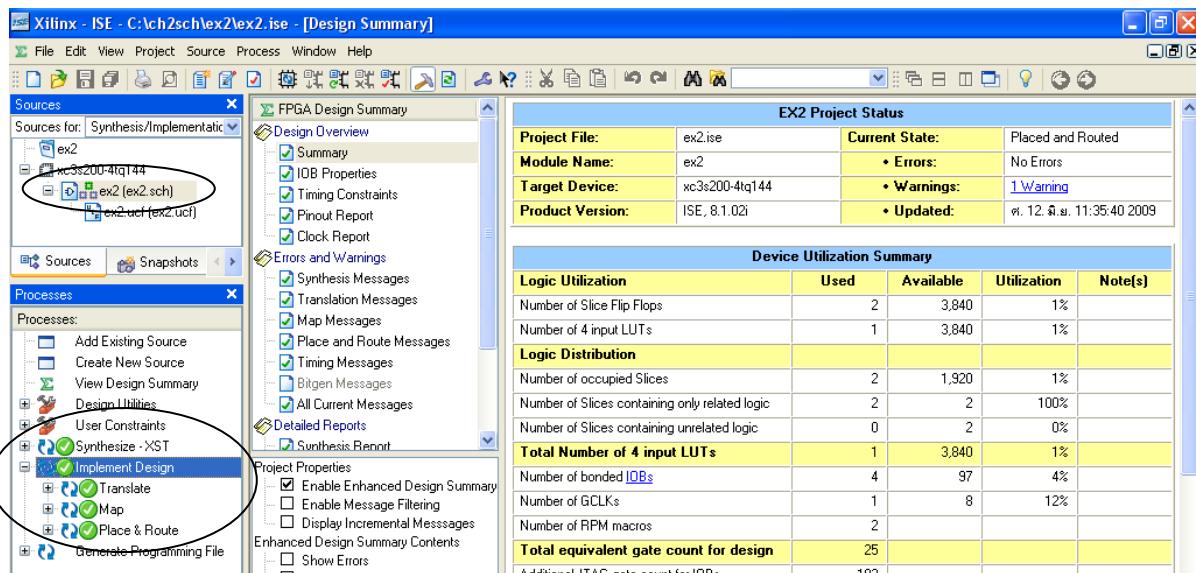
รูปที่ 2.79 หน้าต่าง Xilinx-ISE

- 3) ขั้นตอน Implement Design คลิกที่ไฟล์ ex2(ex2.sch) ในหน้าต่าง Source และคลิกขวาที่ Implement Design ในหน้าต่าง Processes และคลิก Run ดังรูปที่ 2.80 (หรือดับเบิลคลิกก็ได้) เสร็จแล้วคลิก “+” ที่หน้า Implement Design จะเป็น “-” แล้วรอ

ສັກຄູ່ ຄ້າໂຫວ່າງ ດັງຮຽບທີ່ 2.81 ທີ່ວ່າ Implement ພ່ານ ແຕ່ຄ້າໂຫວ່າງ ມີຄໍາວ່າ Implement Design ປີ່ວ່າ Implement ໄມ່ພ່ານ ແລະ ຄ້າໂຫວ່າງ ມີຄໍາວ່າ Translate ສະເ້ອງວ່າ Translate ໄມ່ພ່ານ ກໍ່ໃຫ້ປຶກ Edit Constraints (Text) ຕຽບສອນໃໝ່ເອົາຄົງວ່າມີການກຳນົດຂາໆ FPGA ຄຽບຄຸນຫົວໜ້າເກີນຫົວໜ້າຂາດຫົວໜ້າພິມພົດພາດຫົວໜ້າໄມ່ ບັນທຶກໄຟລ໌ແລ້ວທຳ Implement ຫ້າເອົາຄົງ ສໍາຮັບຄົນທີ່ໃໝ່ ISE WebPACK 10.1i ຄ້າໂຫວ່າງ ມີຄໍາວ່າ Place & Route ສະເ້ອງວ່າ Place & Route ໄມ່ພ່ານ ໃຫ້ເລື່ອນມາສັ່ນໄປຄຸກຂວາ່ທີ່ User Constraints ແລ້ວຄືກ (ຫ້າຍ) ອີ່ Properties... ເມື່ອໄດ້ໜ້າຕ່າງ Process Properties-User Constraints Properties ແລ້ວຄືກທີ່ນ້າ Use LOC Constraints ເພື່ອເອົາ “√” ອອກແລ້ວຄືກ OK ຈາກນັ້ນຄືກ ປຶກໜ້າຕ່າງນີ້ແລ້ວທຳ Implement ຫ້າເອົາຄົງ ຂອໃຫ້ສັງເກດວ່າ ຂັ້ນຕອນ Implement Design ນີ້ຂອົບແວຮ່ຽນຈະກຳນົດຕອນຢ່ອຍຍາ ຄື່ອ Translate, Map ແລະ Place & Route ໃນຂັ້ນຕອນເດືອນ



ຮູບທີ່ 2.80 ການທຳ Implement Design

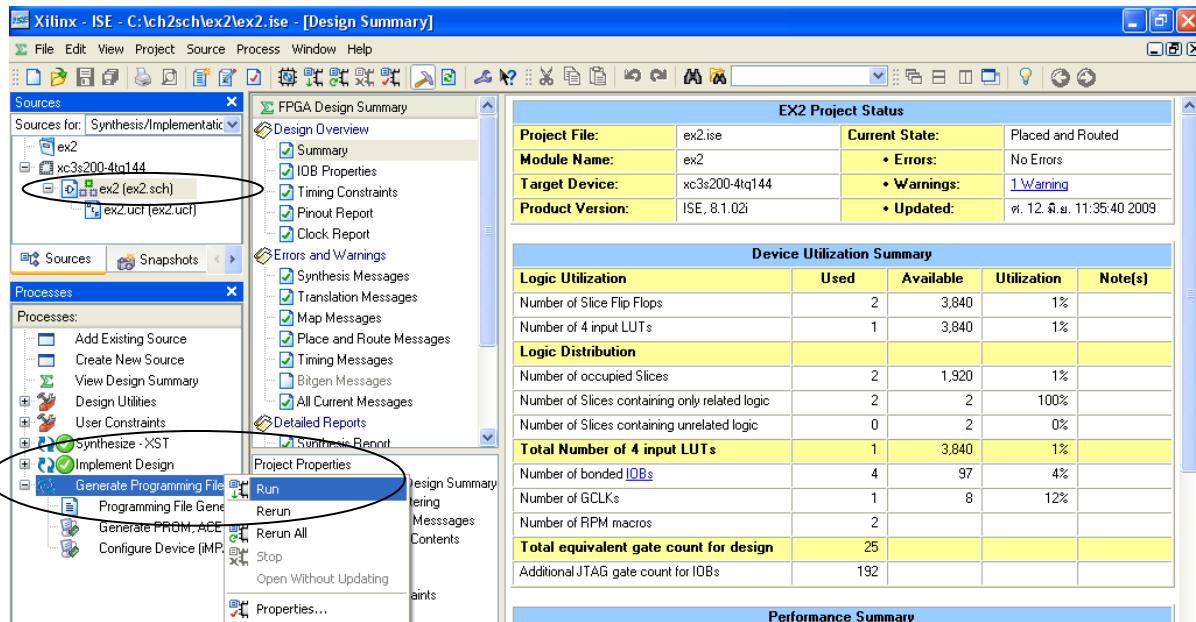


ຮູບທີ່ 2.81 ເມື່ອການທຳ Implementation ເຮັດວຽກ

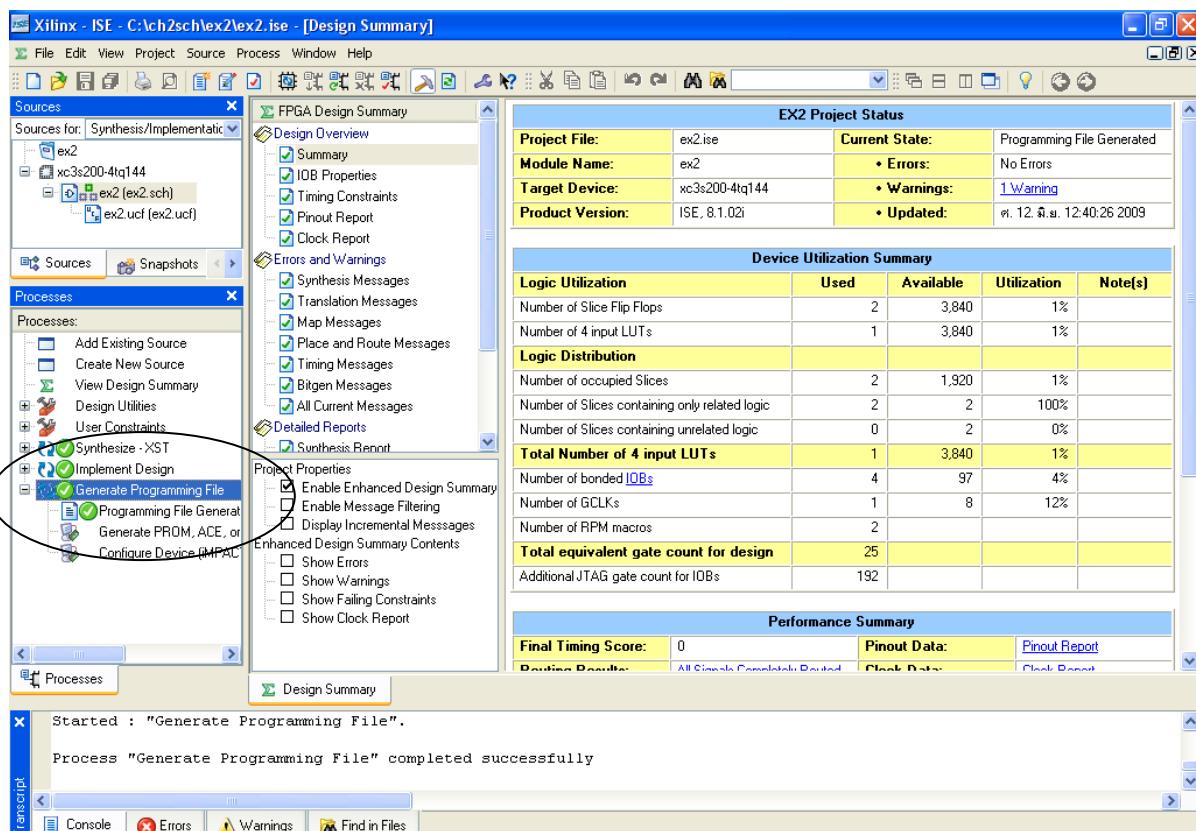
2.2.5 ການໂປຣແກຣມຂໍ້ມູນລວງຈາລອງຊີ່ພ

1) ຂັ້ນຕອນ Generate Programming File

ຄືກ ex2(ex2.sch) ທີ່ນ້າຕ່າງ Source ຄືກ “+” ທີ່ນ້າ Generate Programming File ທີ່ນ້າຕ່າງ Processes ເປັນ “-” ແລະ ຄືກຂວາ່ທີ່ Generate Programming File ແລ້ວຄືກ Run ດັງຮຽບທີ່ 2.82 (ຫຼືດັ່ງເປີດຄືກກີ່ໄດ້) ເມື່ອແລ້ວເສົ່າງຈະໄດ້ດັງຮຽບທີ່ 2.83



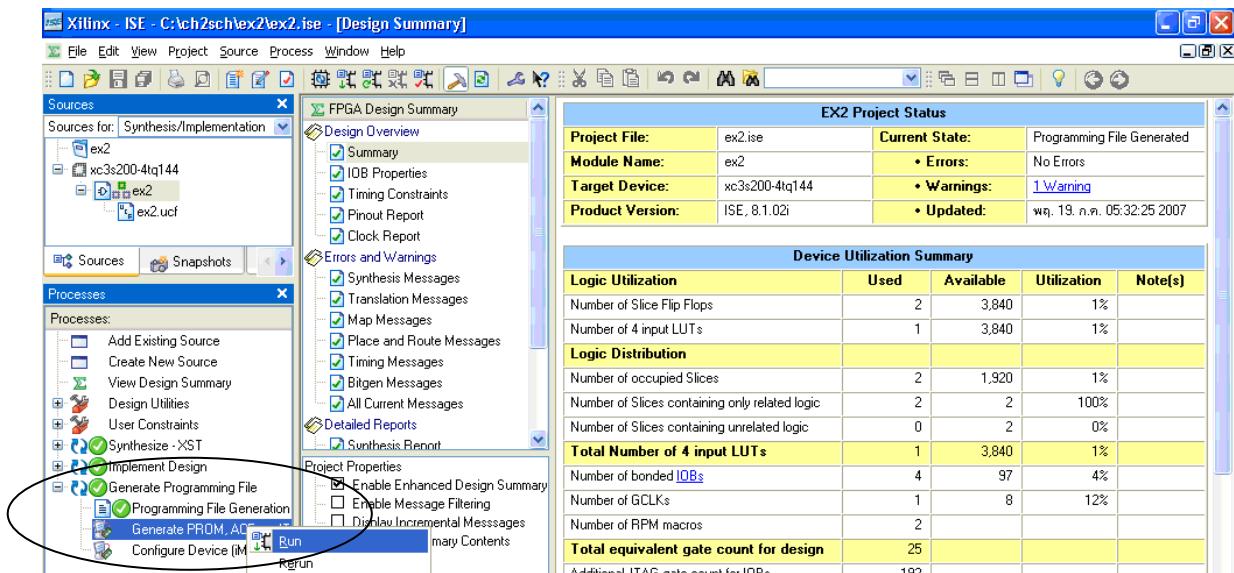
รูปที่ 2.82 ขั้นตอน Generate Programming File



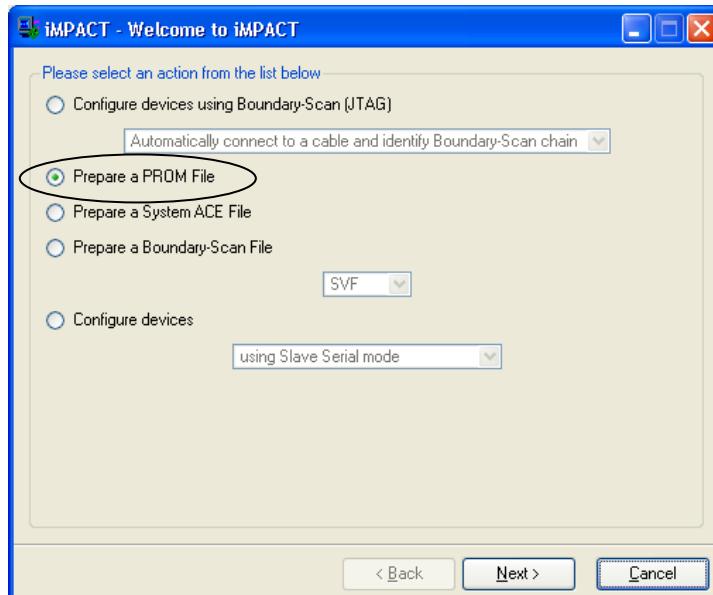
รูปที่ 2.83 เมื่อการทำ Generate Programming File เสร็จเรียบร้อยแล้ว

2) ขั้นตอนสร้างไฟล์ PROM

2.1) ให้คลิกขวาที่ 'Generate PROM, ACE, or JTAG File' ในหน้าต่าง Processes !!เลือก Run ดังรูปที่ 2.84 รอสักครู่แล้วจะได้หน้าต่างขึ้นมา แล้วเลือกช่อง Prepare a PROM File ดังรูปที่ 2.85

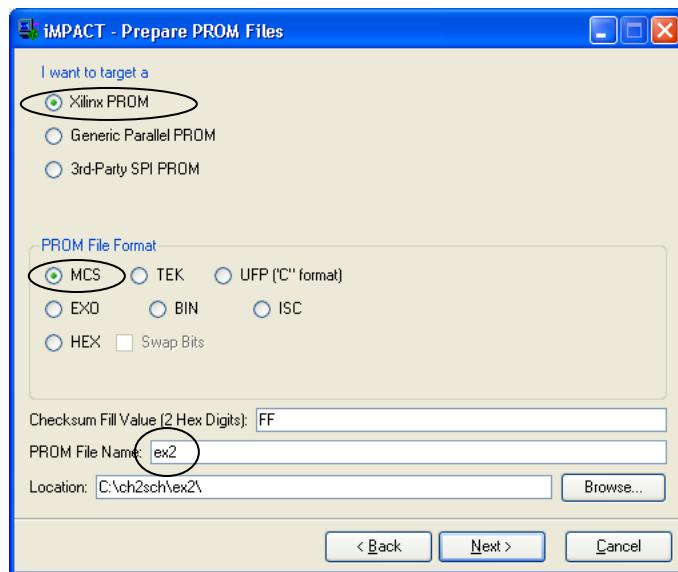


รูปที่ 2.84 ขั้นตอน Generate PROM, ACE, or JTAG File

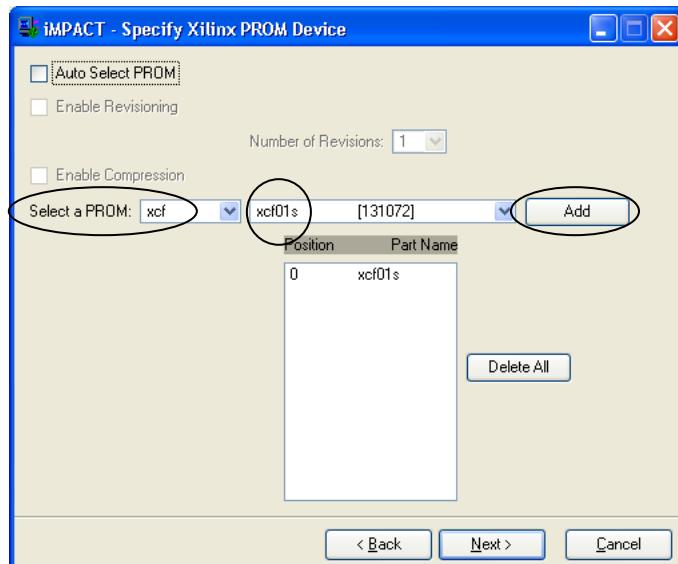


รูปที่ 2.85 หน้าต่าง iMPACT-Welcome to iMPACT

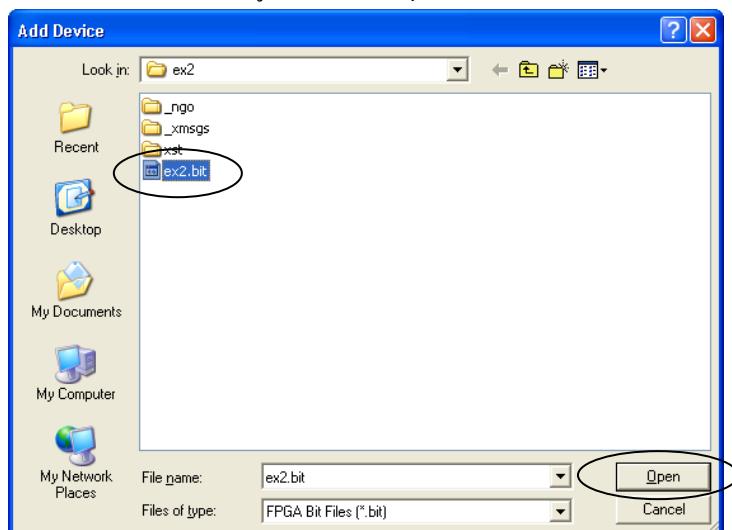
2.2) จากรูปที่ 2.85 คลิก Next แล้วจะได้หน้าต่างถัดไปแล้วพิมพ์ชื่อ ex2 ดังรูปที่ 2.86 คลิก Next แล้วจะได้หน้าต่างถัดไป และเนื่องจาก Flash PROM ที่ใช้เป็นตระกูล xcf เมอร์ xcf01s อ่ายแล้ว (XCF01SVO20C) จึงไม่ต้องแก้ไข (แต่ถ้าใช้บอร์ด FPGA Discovery-III XC3S200F4 จำเป็นต้องคลิกมาส์เพื่อแก้ไข Flash PROM ที่ใช้เป็นเมอร์ xcf02s หรือ XCF02SVO20C) จากนั้น คลิกปุ่ม Add ดังรูปที่ 2.87 คลิก Next แล้วจะได้หน้าต่างถัดไป แล้วคลิก Finish แล้วจะได้หน้าต่างถัดไปดังรูปที่ 2.89 คลิก OK แล้วจะได้หน้าต่าง Add Device ดังรูปที่ 2.88 คลิกที่ไฟล์ชื่อ ex2.bit และคลิก Open แล้วจะได้หน้าต่างถัดไปดังรูปที่ 2.89 คลิก No และคลิก OK แล้วจะได้หน้าต่างถัดไป จากนั้นดับเบิลคลิกที่เมนู Generate File แล้วจะได้ดังรูปที่ 2.90 ซึ่งถือว่าได้สร้างไฟล์ Flash PROM แล้วเสร็จ คลิก **X** เพื่อปิดโปรแกรมที่สร้างไฟล์ Flash PROM แล้วคลิก No ดังรูปที่ 2.91



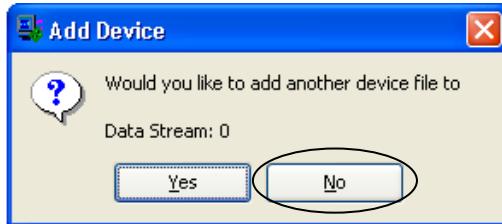
รูปที่ 2.86 พิมพ์ชื่อ ex2 ที่ช่อง PROM File Name



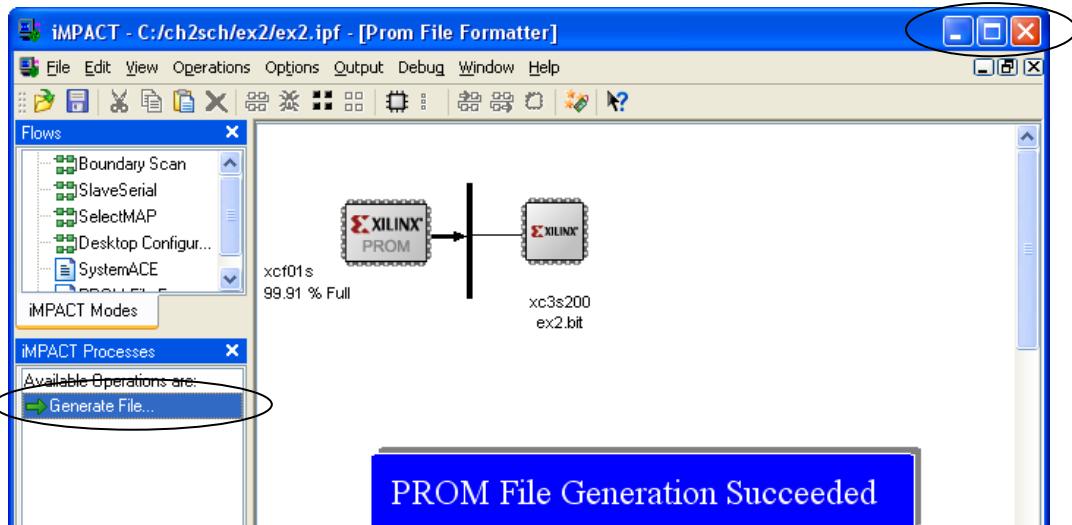
รูปที่ 2.87 คลิกปุ่ม Add



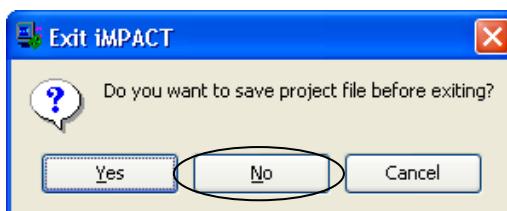
รูปที่ 2.88 หน้าต่าง Add Device



รูปที่ 2.89 หน้าต่าง Add Device



รูปที่ 2.90 หน้าต่าง iMPACT



รูปที่ 2.91 หน้าต่าง Exit iMPACT

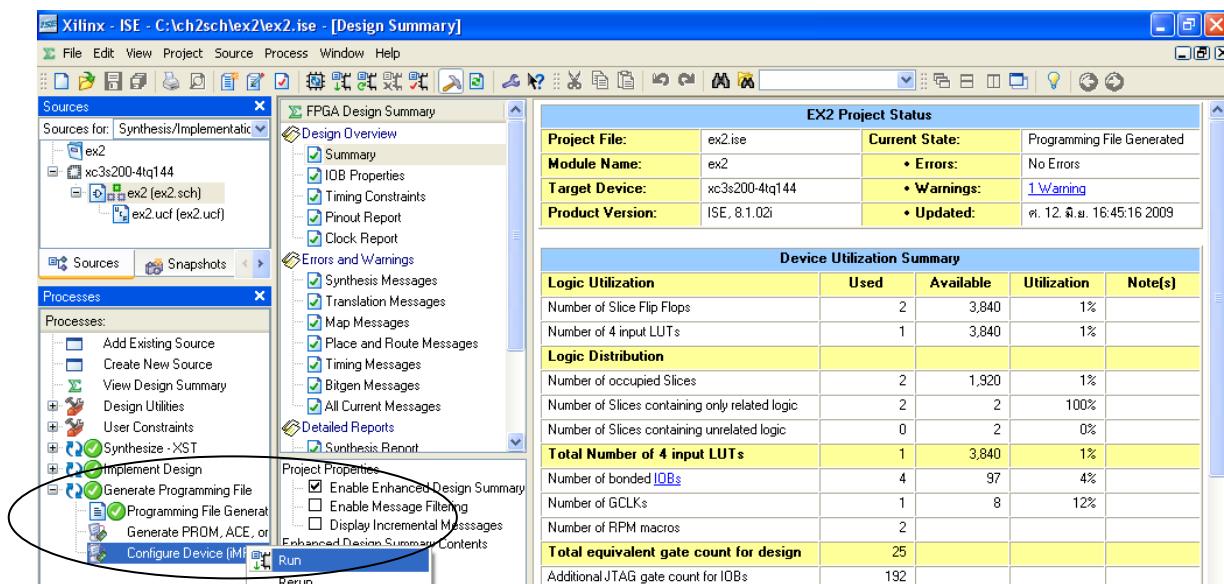
3) ขั้นตอนการโปรแกรมลง Flash PROM และ/หรือ FPGA

3.1) ขั้นตอนการเซตจัมเปอร์ J1 เพื่อเซตค่าของ M0,M1,M2 โดยปกติจัมเปอร์ J1 บนบอร์ด FPGA Discovery-III XC3S200F จะถูกเซต $M0,M1,M2 = "000"$ ไว้แล้ว ซึ่งเป็นโหมด Master serial (ดูตามตารางที่ 2.1) ข้อมูลวงจรใน Flash PROM จึงถูกดาวน์โหลดลง FPGA โดยอัตโนมัติทันทีที่เริ่มจ่ายไฟเลี้ยง การดาวน์โหลดไฟล์ข้อมูลวงจรใหม่ลง FPGA ผ่านทางสาย JTAG โดยไม่เซตโหมดเป็น JTAG จึงอาจเกิดข้อผิดพลาดขึ้นได้ก็ต่อเมื่อ วงจรอาจจะทำงานไม่ตรงตามที่ออกแบบไว้เนื่องจากเป็นการดาวน์โหลดผิดโหมด การแก้ไขปัญหาวิธีที่ 1 ดาวน์โหลดไฟล์ข้อมูลวงจรใหม่ (นามสกุล.mcs) ลง Flash PROM ก่อน จากนั้นจึงใช้ไฟล์วงจรเดียวกัน (นามสกุล.bit) ดาวน์โหลดลง FPGA วิธีที่ 2 ลงไฟล์ใน Flash PROM ทึ่งก่อนทำการดาวน์โหลดลง FPGA หรืออาจใช้วิธีที่ 3 โดยเซตจัมเปอร์ J1 ในโหมด JTAG ซึ่งมีค่า $M0,M1,M2 = "101"$ ก่อนดาวน์โหลดไฟล์ข้อมูลวงจรใหม่ลง FPGA แต่ข้อเสียของวิธีนี้ คือ เมื่อเริ่มจ่ายไฟเลี้ยงจะแล้วไม่มีการดาวน์โหลดข้อมูลใน Flash PROM ลง FPGA โดยอัตโนมัติ

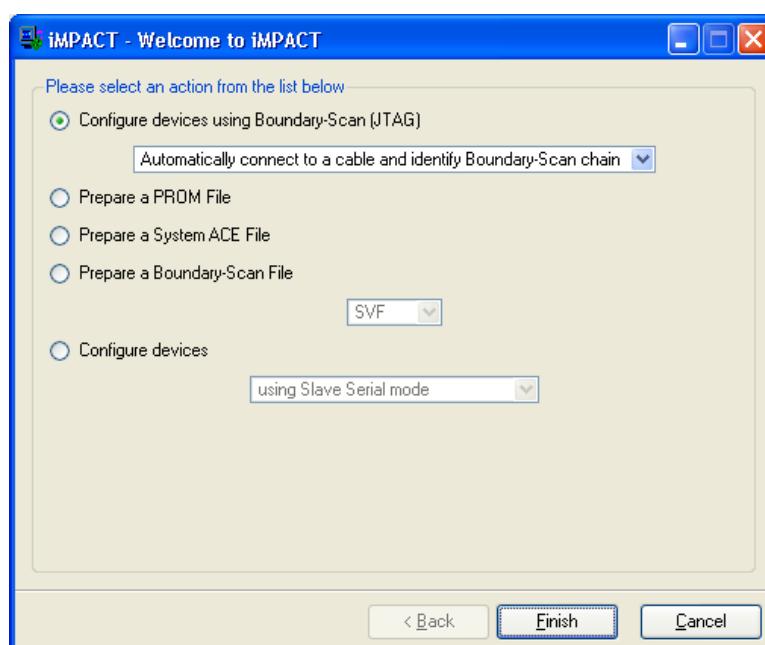
ตารางที่ 2.1 รายการเซตโหมดในการโปรแกรม FPGA

Configuration Mode	M0	M1	M2
Master Serial	0	0	0
Slave Serial	1	1	1
Master Parallel	1	1	0
Slave Parallel	0	1	1
JTAG	1	0	1

3.2) ขั้นตอนดาวน์โหลดไฟล์ลง Flash PROM และ FPGA ให้ต่อสาย JTAG เข้ากับพอร์ตบนนาฬองคอมพิวเตอร์และขึ้น JTAG ที่บอร์ดทดลอง FPGA Discovery-III XC3S200F แล้วจ่ายไฟเลี้ยงเข้าบอร์ดโดยต่อเข้ากับอะแดปเตอร์ 9 VDC จากนั้นคลิกขวาที่ Configure Device (iMPACT) และคลิก Run แสดงในรูปที่ 2.92 รอสักครู่แล้วจะได้หน้าต่างดังรูปที่ 2.93

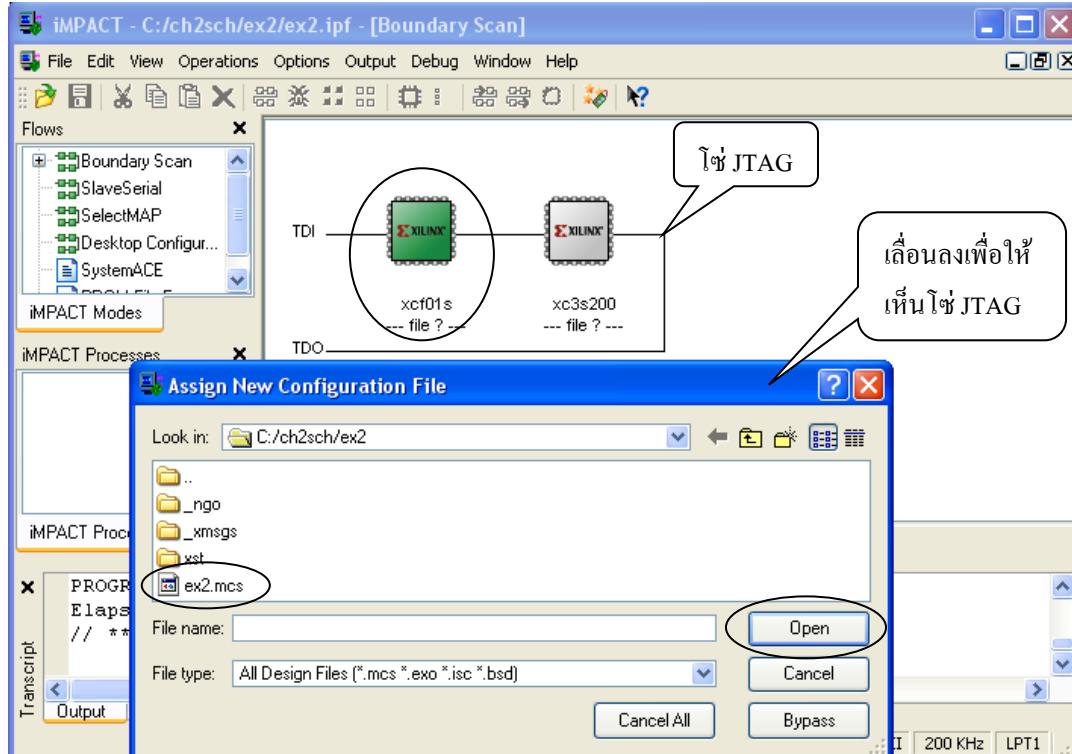


รูปที่ 2.92 แสดงขั้นตอน Configure Device (IMPACT)

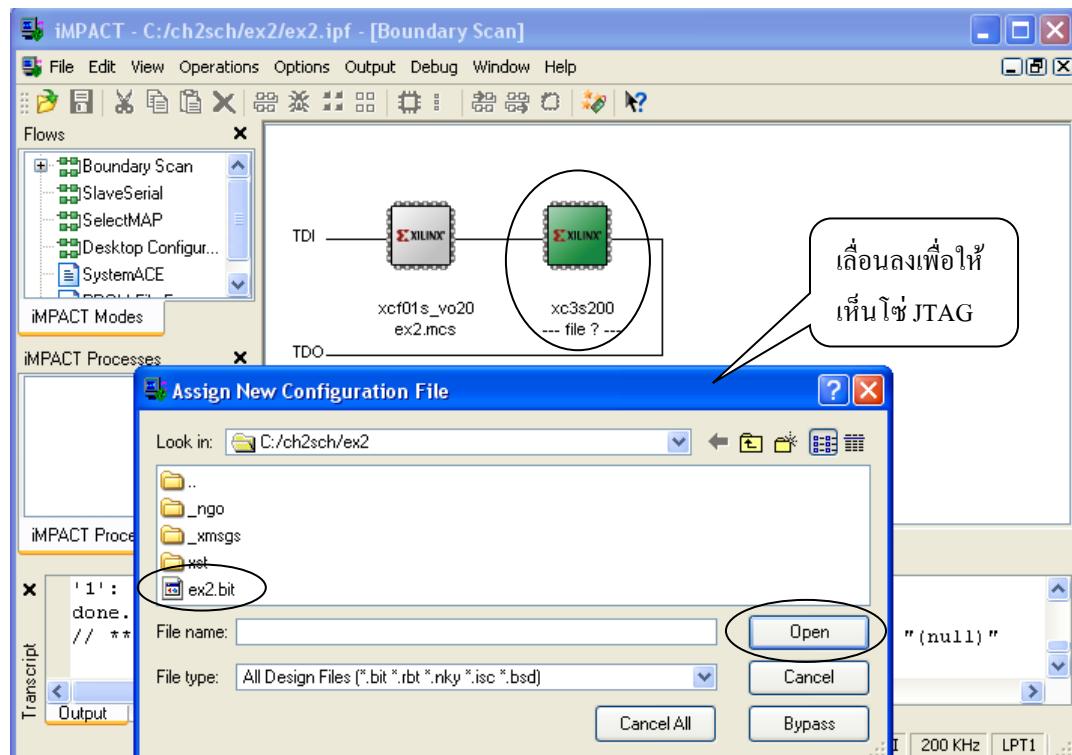


รูปที่ 2.93 หน้าต่าง iMPACT

3.3) จากรูปที่ 2.93 คลิก Finish และจะได้หน้าต่าง Assign New Configuration File ของ Flash PROM ดังรูปที่ 2.94 คลิกที่ไฟล์ ex2.mcs (PROM configuration file) และคลิก Open (หรือให้คลิก Cancel หากไม่ได้สร้างไฟล์ PROM ในข้อ 2) และจะได้หน้าต่าง Assign New Configuration File ของ FPGA ดังรูปที่ 2.95 คลิกที่ไฟล์ ex2.bit (FPGA configuration file) คลิก Open และจะได้หน้าต่าง Warning ดังรูปที่ 2.96 จากนั้นคลิก OK และจะได้หน้าต่าง iMPACT



รูปที่ 2.94 หน้าต่าง Assign New Configuration เพื่อเตรียมข้อมูลที่จะดาวน์โหลดลง PROM

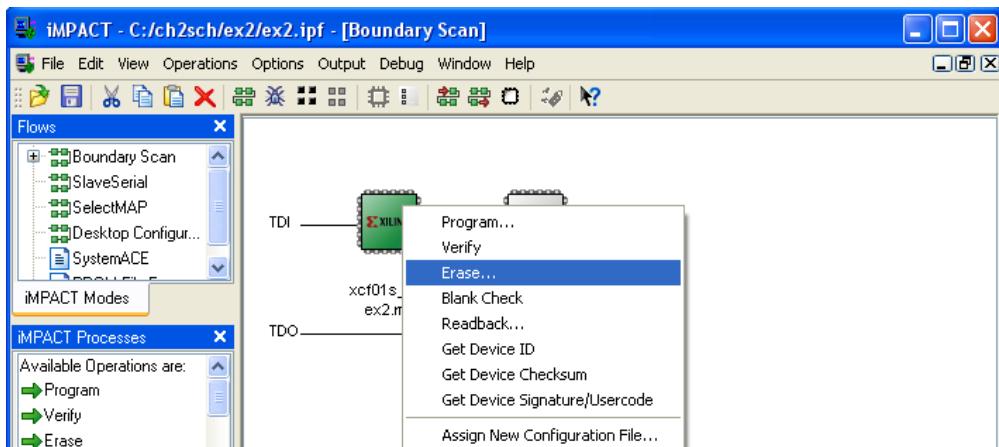


รูปที่ 2.95 หน้าต่าง Assign New Configuration เพื่อเตรียมข้อมูลที่จะดาวน์โหลด FPGA

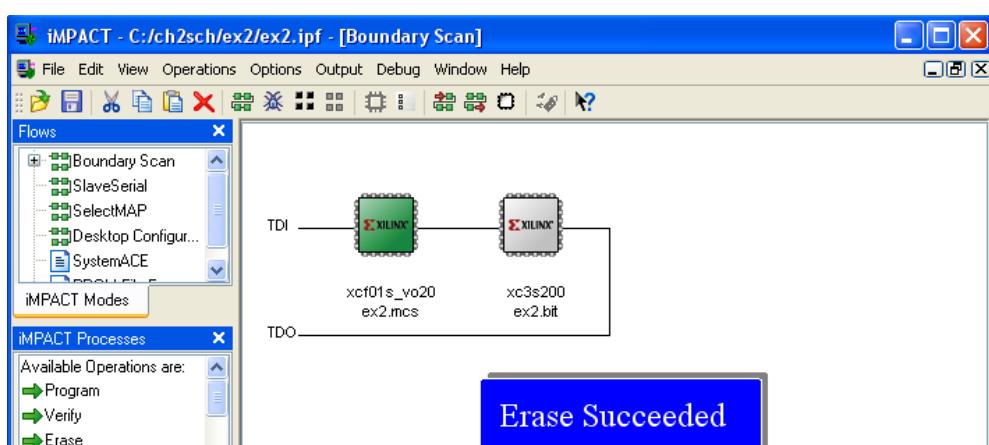


รูปที่ 2.96 หน้าต่าง Warning

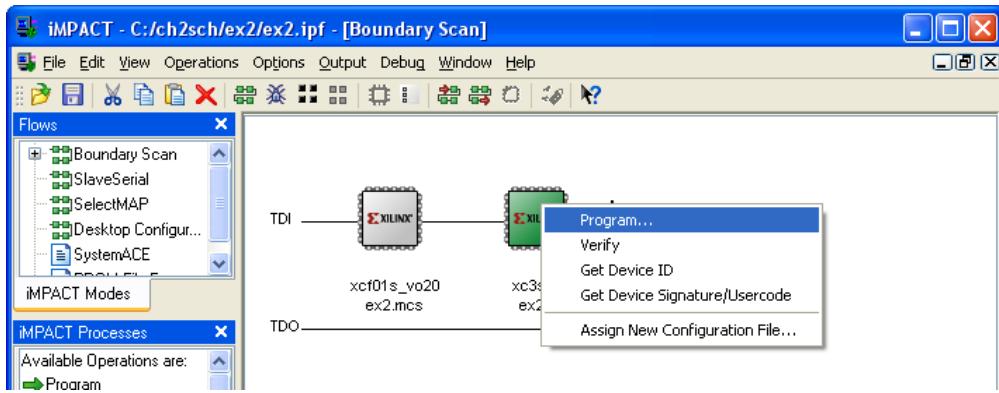
3.4) การดาวน์โหลดข้อมูลความจริง FPGA เนื่องจากจัมเปอร์ J1 ของบอร์ดทดลองถูกเซตอยู่ในโหมด Master serial ดังนั้นก่อนโปรแกรม FPGA จะต้องลบ (Erase) ข้อมูลจริงที่เคยโปรแกรมลง PROM ทึ้งโดยการคลิกขวาที่รูป PROM และคลิก Erase ดังรูปที่ 2.97 เมื่อแล้วเสร็จจะได้ดังรูปที่ 2.98 จากนั้นคลิกขวาที่รูป FPGA และคลิก Program และดังรูปที่ 2.99 แล้วจะได้หน้าต่างคลิกไป คลิก OK เสร็จแล้วจะได้ดังรูปที่ 2.100 ถ้าดาวน์โหลดไม่ผ่านให้ดาวน์โหลดซ้ำอีกครั้ง และในการกรณีที่ดาวน์โหลดไม่ผ่านพยายามครั้ง ให้อ่านไฟล์เดิมๆ ก่อนและนำไฟล์เดิมเข้าบอร์ดอีกครั้ง แล้วทำการดาวน์โหลดซ้ำ และหากดาวน์โหลดข้อมูลความจริง Flash PROM ก่อนดาวน์โหลดข้อมูลความจริง FPGA ก็ไม่จำเป็นต้องลบข้อมูล Flash PROM ทึ้ง



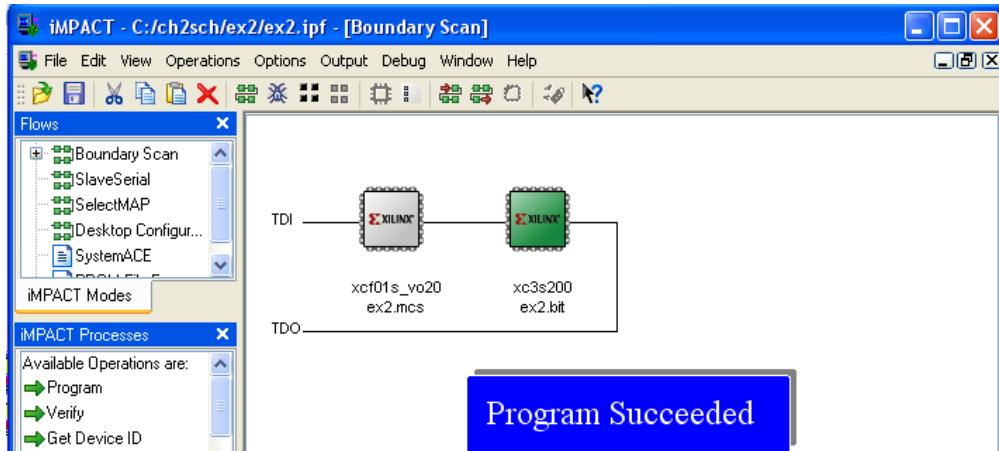
รูปที่ 2.97 ขั้นตอนลบ (Erase) ข้อมูลจริงที่เคยโปรแกรมลง PROM ทึ้ง (ถ้าเคลียร์แล้วไม่ต้องลบซ้ำอีก)



รูปที่ 2.98 ลบ (Erase) PROM เรียบร้อยแล้ว



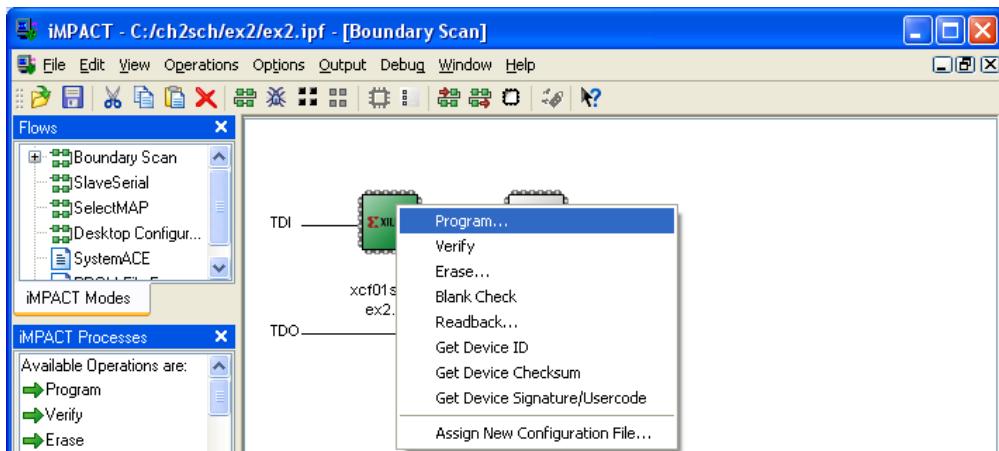
รูปที่ 2.99 หน้าต่าง iMPACT



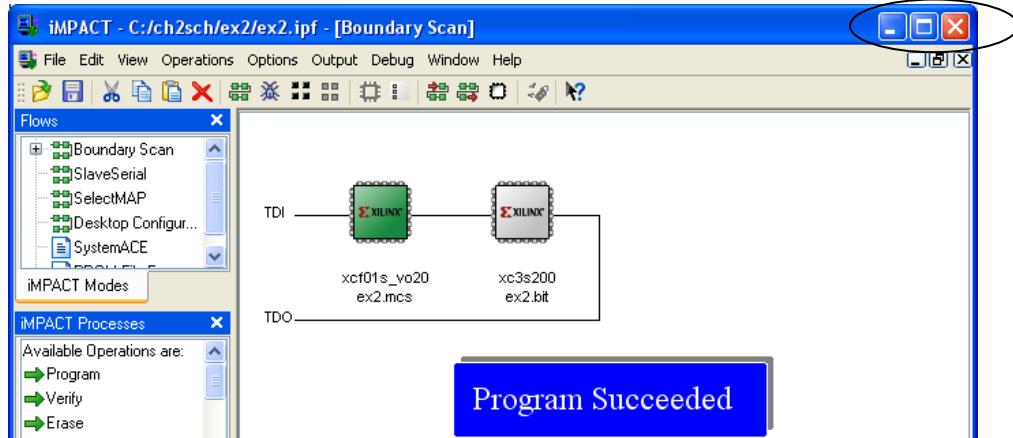
รูปที่ 2.100 หน้าต่าง iMPACT

3.5) การดาวน์โหลดข้อมูลวงจรลง Flash PROM ในกรณีที่ไม่สร้างไฟล์ PROM ในข้อ 2) หรือไม่ต้องการโปรแกรม Flash PROM ก็ไม่จำเป็นต้องทำในข้อนี้ การดาวน์โหลดข้อมูลวงจรลง Flash PROM ให้คลิกขวาที่รูป PROM และคลิก Program ดังรูปที่ 2.101 แล้วจะได้หน้าต่างกดไป คลิก OK เสร็จแล้วจะได้ดังรูปที่ 2.102 ถ้าดาวน์โหลดไม่ผ่านก็ให้ดาวน์โหลดซ้ำ

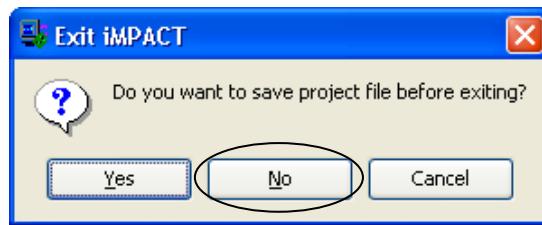
3.6) ให้ทดลองกดปุ่ม PB1 และ PB2 แล้วสังเกตที่ L1 และ L2 ว่าให้ผลตามที่ออกแบบหรือไม่ ปิดหน้าต่าง iMPACT โดยคลิก แล้วจะได้หน้าต่าง Exit iMPACT ซึ่งอนันนมาดังรูปที่ 2.103 แล้วคลิก No จากนั้นคลิก เพื่อปิดโปรแกรม ISE WebPACK



รูปที่ 2.101 หน้าต่าง iMPACT



รูปที่ 2.102 หน้าต่างแสดงเมื่อการดาวน์โหลดผ่าน

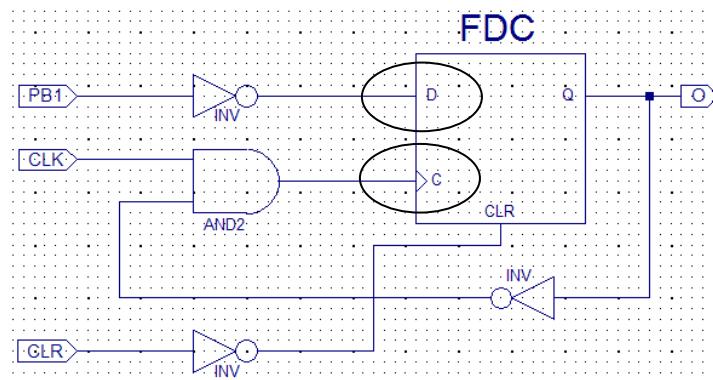


รูปที่ 2.103 หน้าต่าง Exit

2.3 การจำลองการทำงานเชิงพฤติกรรมของวงจรที่ออกแบบด้วย FPGA และ CPLD

การตรวจสอบความถูกต้อง (Design verification) ของวงจรนี้สามารถทำได้ด้วยวิธีการจำลองการทำงาน (Simulation) โดยในเบื้องต้นนี้เป็นการจำลองการทำงานเชิงพฤติกรรม (Behavioral simulation) ซึ่งจะไม่นำ Delay time ต่างๆ ที่เกิดขึ้นภายในมาคำดิค ขั้นตอนการจำลองการทำงานเชิงพฤติกรรมของวงจรที่ออกแบบด้วย FPGA หรือ CPLD จะเหมือนกันทุกประการ

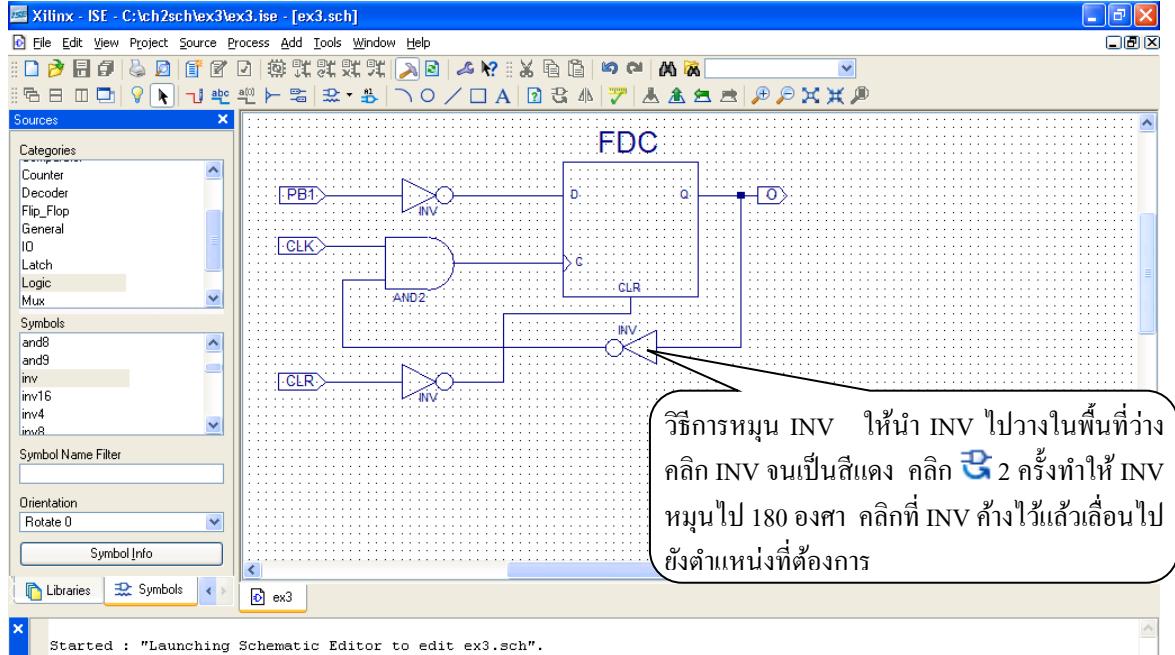
ตัวอย่างที่ 2.3 ออกแบบและจำลองการทำงานวงจรโนโนสเตเดมิล็อกย่างง่าย โดยใช้ CPLD แสดงดังรูปที่ 2.104 ที่มีหลักการทำงานดังนี้ สัญญาณนาฬิกา (CLK) ความถี่สูง (32.768KHz) จะถูกป้อนผ่านแอนด์เกตไปทริกท์ขา C ของ D Flip-Flop ได้ตลอดเวลา ทราบได้ที่เอาต์พุต O ยังคงเป็นโลจิก ‘0’ แต่เมื่อกดปุ่ม PB1 (โลจิก ‘1’->‘0’) ทำให้ขา D โลจิกเปลี่ยน ‘0’->‘1’ ทำให้อเอต์พุต Q หรือ O เปลี่ยนจาก ‘0’->‘1’ เป็นผลให้ CLK ไม่สามารถผ่านแอนด์เกตไปทริกท์ขา C ของ D Flip-Flop ได้อีก ดังนั้นเอาต์พุต O จึงเป็น ‘1’ ไปจนกระทั่งมีการกดปุ่มเคลียร์ที่ขา CLR แล้วเอาต์พุต O จึงกลับมาเป็น ‘0’ อีกครั้ง นั่นก็หมายความว่าเมื่อเรากดปุ่ม PB1 ก็จะเป็นการส่งสัญญาณทริกค้ายของขาขึ้นหรือบนบวกที่เอาต์พุต O ขั้นตอนการออกแบบเป็นดังนี้



รูปที่ 2.104 วงจรโนโนสเตเดมิล็อกย่างง่าย

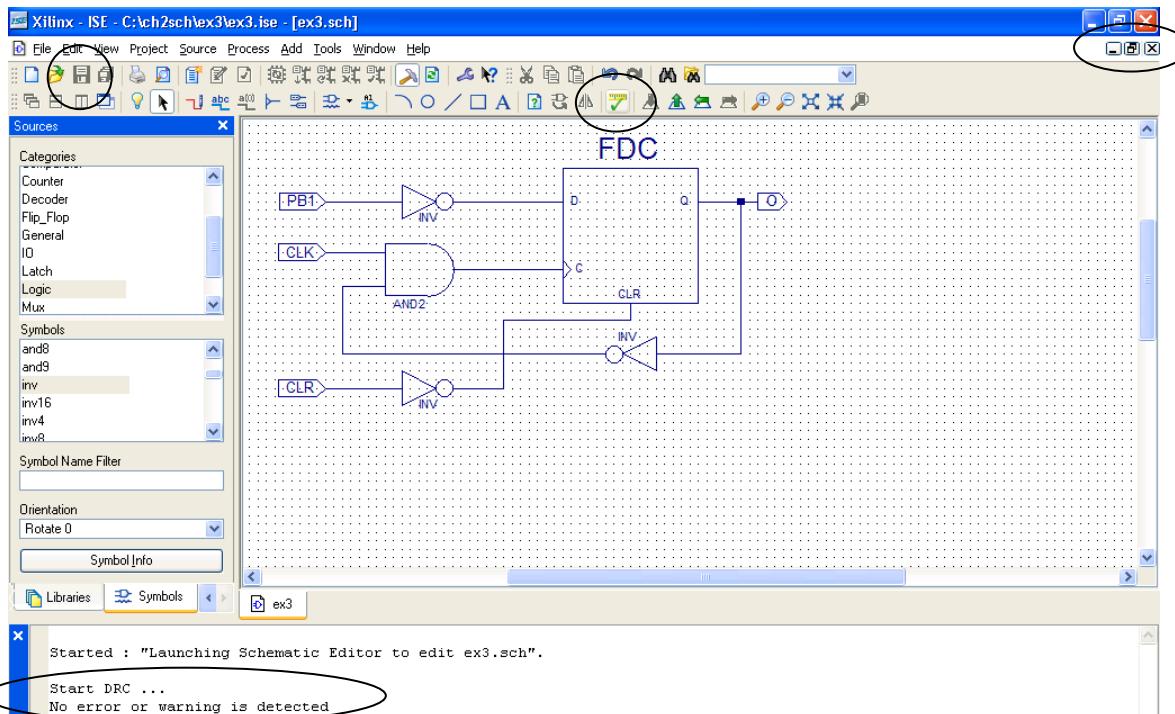
2.3.1 ขั้นตอนการออกแบบวงจร

1) ขั้นตอนการออกแบบวงจรด้วย CPLD จะเหมือนกับข้อ 2.1.1 ตัวอย่างที่ 2.1 ทุกประการ เมื่อทำการวางแผนผังวงจรโโนนสเตเบิล ในรูป 2.104 แล้วเสร็จจะได้ดังรูปที่ 2.105



รูปที่ 2.105 วงจรที่แล้วเสร็จสมบูรณ์

2) ตรวจสอบความถูกต้อง (Syntax) โดยคลิกที่ ถ้าไม่มีข้อผิดพลาดจะได้ปรากฏข้อความ No error or warning is detected ดังรูปที่ 2.106 บันทึกไฟล์โดยคลิก ในขั้นตอนนี้ถือว่าการออกแบบวงจรแล้วเสร็จ จากนั้นคลิก (สีดำ) เพื่อปิดโปรแกรม Schematic editor เพื่อกลับไปที่หน้าต่าง Xilinx-ISE อีกรั้ง จากนั้นคลิก View -> Restore Default Layout เพื่อทำขั้นตอนต่อไป

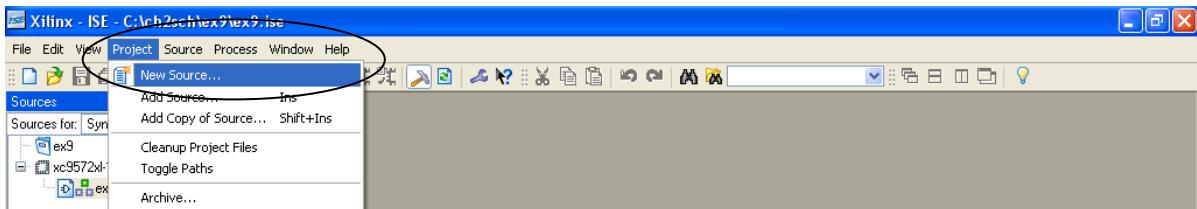


รูปที่ 2.106 การตรวจสอบข้อผิดพลาด

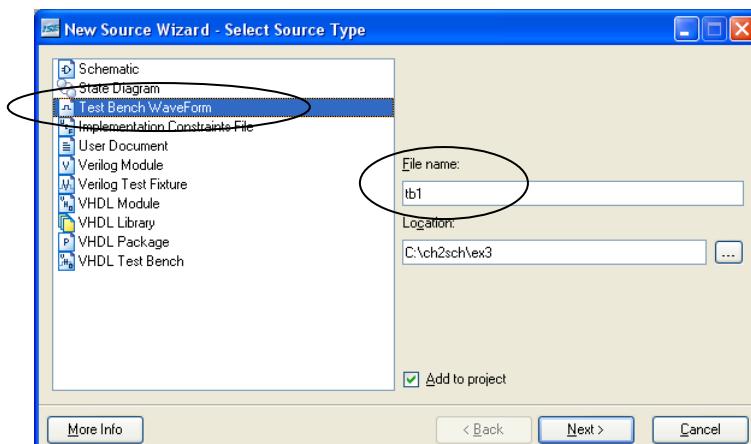
2.3.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification)

การทำ Behavioral simulation จะทำหลังจากที่วัดผังวงจรเสร็จเรียบร้อยแล้ว มีขั้นตอนดังนี้

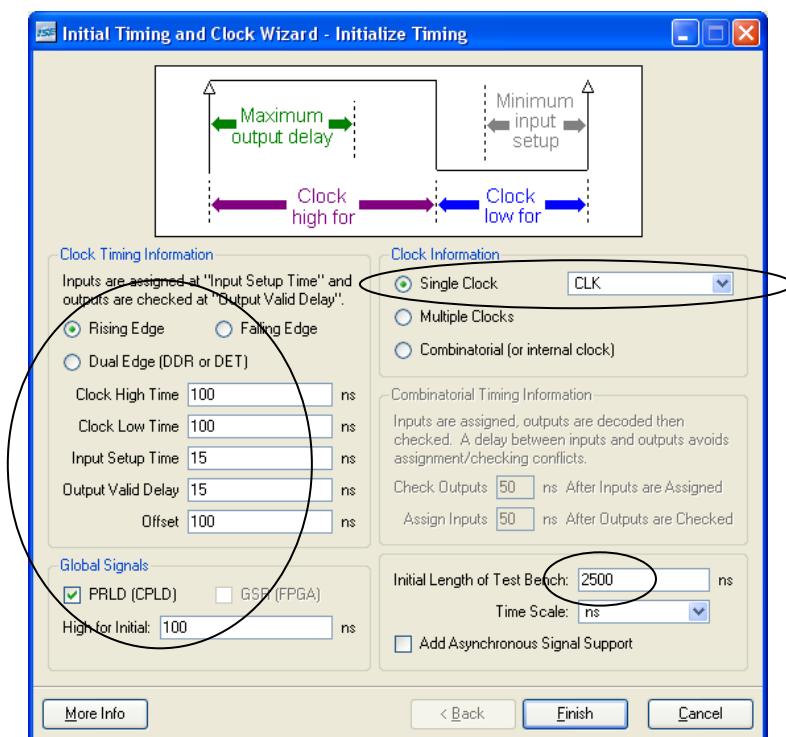
- คลิก Project -> New Source ดังรูปที่ 2.107 และจะได้หน้าต่าง New Source Wizard พิมพ์ชื่อ tb1 (ห้ามตั้งชื่อซ้ำกับ ex3) และคลิก Test Bench Waveform ดังรูปที่ 2.108 คลิก Next คลิก Next คลิก Finish และจะได้หน้าต่างกดไป ใส่ค่าต่างๆ ดังรูปที่ 2.109 โดยที่ค่าของ Clock = Clock High Time + Clock Low Time = $100 + 100 = 200$ ns



รูปที่ 2.107 หน้าต่าง Xilinx-ISE

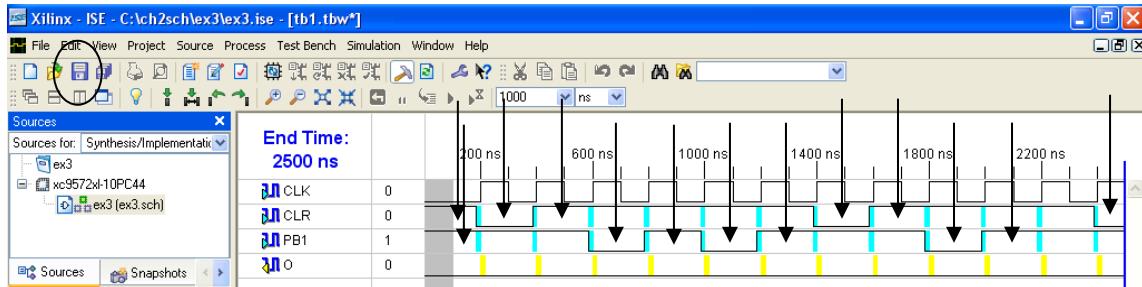


รูปที่ 2.108 หน้าต่าง New Source Wizard

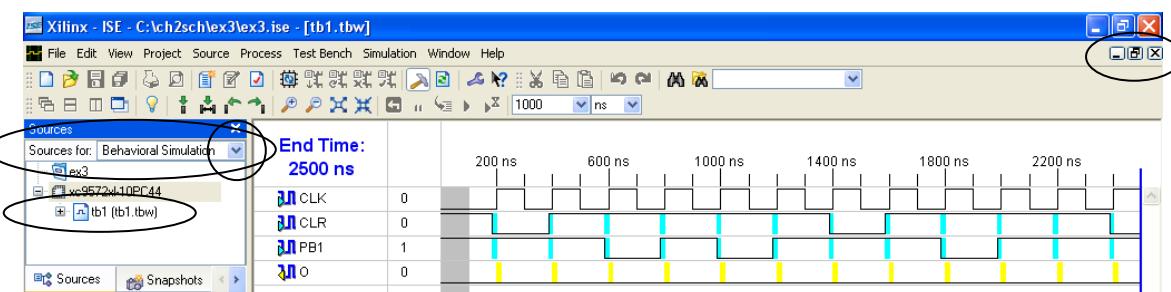


รูปที่ 2.109 หน้าต่าง Initial Timing and Clock Wizard

- 2) ປຶ້ອນອິນພຸດ CLR ແລະ PB1 ໂດຍຄລິກຂ່ອງທີ່ປລາຍລູກຄຣີດັ່ງຮຽບທີ່ 2.110 ຄລິກ ນັ້ນທຶກໄຟລ໌ ຄລິກ ແລະຄລິກທີ່ Behavioral Simulation ເພື່ອເພີ່ມໄຟລ໌ tb1 ໂດຍອັດໂນມຕິດັ່ງຮຽບທີ່ 2.111 ຄລິກ (ສີດຳ) ປຶ້ມໂປຣແກຣມແລກລັບໄປທີ່ໜ້າຕ່າງ Xilinx–ISE

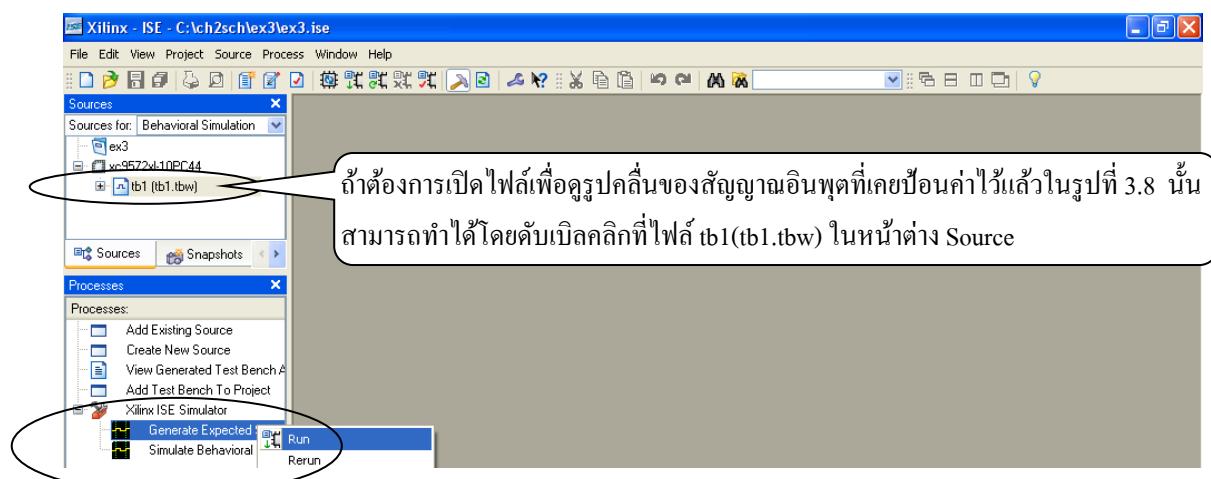


ຮູບທີ່ 2.110 ໜ້າຕ່າງສໍາຮັບກໍານົດສ້າງສ້າງທີ່ປຶ້ອນໄກ້ກັບອິນພຸດຂອງຈະຈຸກຕ່າງທີ່ເວົ້າຕ່າງກັນ

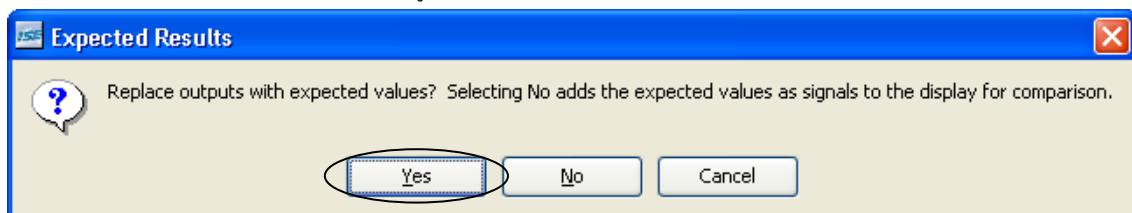


ຮູບທີ່ 2.111 ໜ້າຕ່າງສໍາຮັບກໍານົດສ້າງສ້າງທີ່ ລັງຈາກຄລິກທີ່ Behavioral Simulation ເພື່ອເພີ່ມໄຟລ໌ tb1

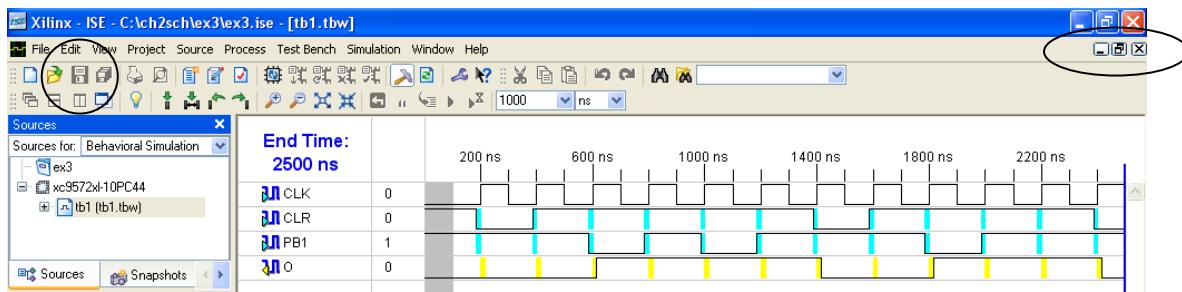
- 3) ຂັ້ນຕອນ Create a self-checking test bench ເພື່ອສ່ວັງເອົາຕຸພູດທີ່ຄວາມເປັນ (ຫຼືເອົາຕຸພູດຕາມທຸກຢືນ) ໂດຍຄລິກເລືອກໄຟລ໌ຂ່ອງ tb1(tb1.tbw) ທີ່ໜ້າຕ່າງ Source ຄລິກ “+” ໜ້າ Xilinx ISE Simulator ທີ່ໜ້າຕ່າງ Processes ຈະເປັນ “-” ຄລິກຫາວ່າທີ່ Generate Expected Simulation Results ແລ້ວຄລິກ Run ດັ່ງຮຽບທີ່ 2.112 ແລ້ວຈະໄດ້ດັ່ງຮຽບທີ່ 2.113 ເມື່ອຄລິກ Yes ແລ້ວຈະໄດ້ດັ່ງຮຽບທີ່ 2.114 ຄລິກ ນັ້ນທຶກໄຟລ໌ ຄລິກ (ສີດຳ) ເພື່ອປຶ້ມໂປຣແກຣມແລກລັບໄປທີ່ໜ້າຕ່າງ Xilinx–ISE



ຮູບທີ່ 2.112 ໜ້າຕ່າງ Xilinx–ISE

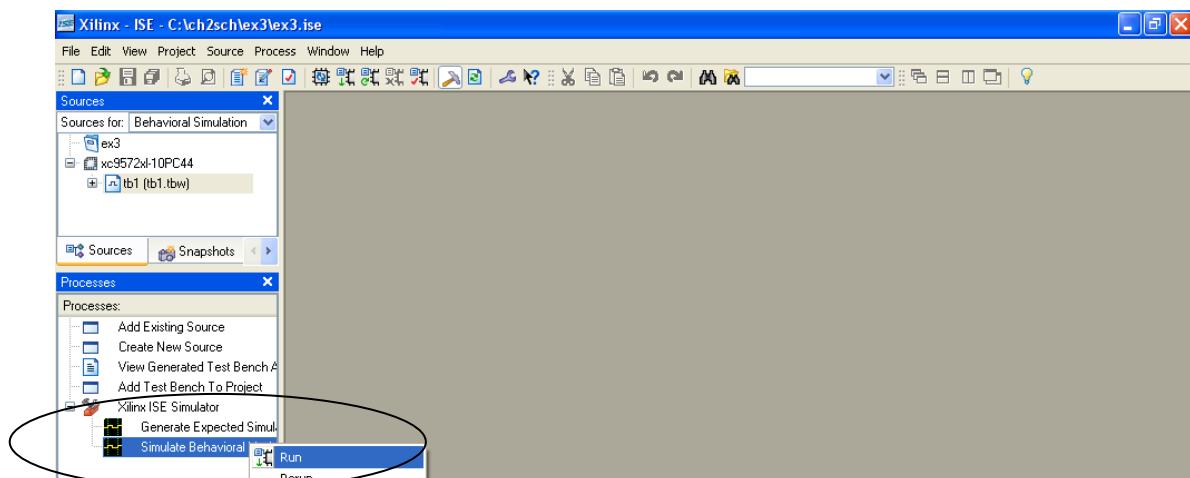


ຮູບທີ່ 2.113 ໜ້າຕ່າງ Expected Results

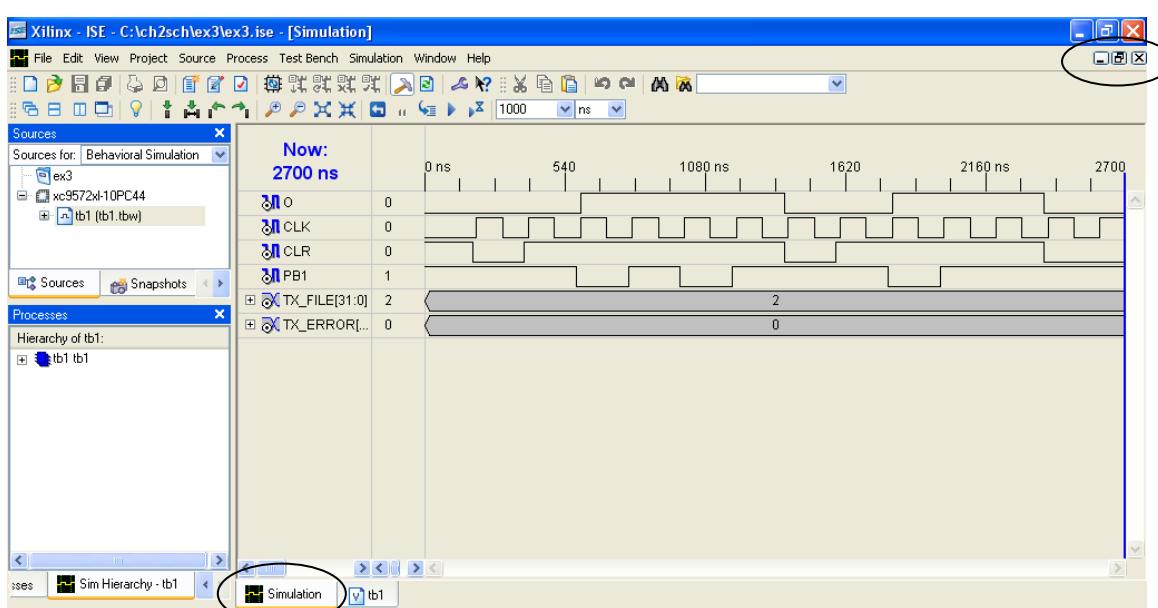


รูปที่ 2.114 ขั้นตอน Create a self-checking test bench

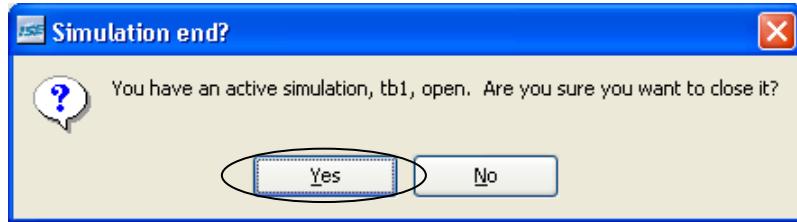
4) จากรูปที่ 2.114 ถ้าได้อาจพูดเป็นไปตามที่ออกแบบไว้ก็จะไปทำ Behavioral simulation โดยคลิกขวาที่ Simulate Behavioral Model แล้วคลิก Run ดังรูปที่ 2.115 เสร็จแล้วคลิกแท็บ Simulation ด้านล่างของรูปที่ 2.116 เพื่อแสดงผล Behavioral simulation ซึ่งขั้นตอน Simulation นี้จะไม่นำผล Delay time ต่างๆ มาคิด จึงเป็นการตรวจสอบความถูกต้องในเบื้องต้นเท่านั้น คลิก **X** (ลีฟ์) ปิดโปรแกรม แล้วคลิก Yes ในรูปที่ 2.117 เพื่อกลับไปที่หน้าต่าง Xilinx-ISE จากนั้นที่หน้าต่าง Source ให้คลิก **▼** และคลิกที่ Behavioral Simulation กลับไปเป็น Synthesis/Implementation ตามเดิม เพื่อทำในขั้นตอนต่อไป



รูปที่ 2.115 ขั้นตอน Behavioral simulation



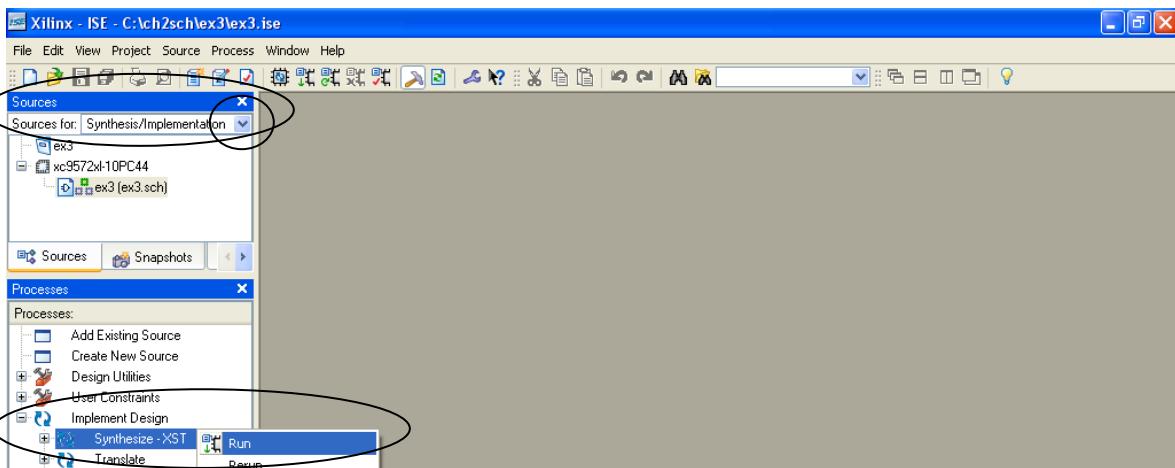
รูปที่ 2.116 แสดงผล Behavioral simulation ของวงจร Half adder (ในขั้นตอนนี้จะไม่นำผลเวลาล่าช้ามาคิด)



รูปที่ 2.117 หน้าต่าง Simulation end

2.3.3 การสังเคราะห์วงจร (Design synthesis)

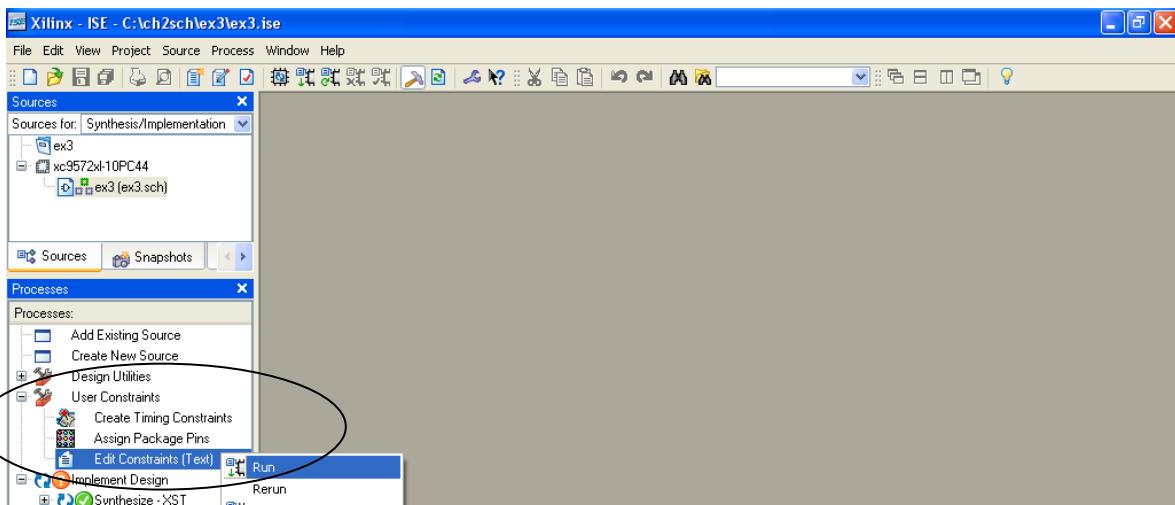
ที่หน้าต่าง Xilinx-ISE คลิก “+” ที่หน้า Implement Design ในหน้าต่าง Processes จนเป็น “-” คลิกขวาแล้วคลิก Run ที่ Synthesize-XST ดังรูปที่ 2.118 เสร็จแล้วถูก ✓ หรือ ⚠ หน้า Synthesize-XST ก็ถือว่าสังเคราะห์วงจรผ่าน



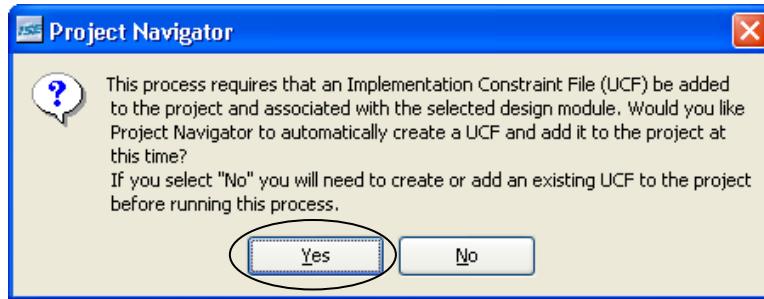
รูปที่ 2.118 แสดงหน้าต่าง Processes แสดงขั้นตอนสังเคราะห์วงจร

2.3.4 Design Implementation

1) ขั้นตอนสร้าง Implementation constraints file (UCF) โดยอัตโนมัติ ทำได้โดยคลิก “+” ที่อยู่หน้า User Constraints ในหน้าต่าง Processes จนเป็น “-” คลิกขวาแล้วคลิก Run ที่ Edit Constraints(Text) ดังรูปที่ 2.119 จากนั้นให้คลิก Yes ในรูปที่ 2.120 เพื่อสร้าง Implementation constraints file โดยอัตโนมัติ

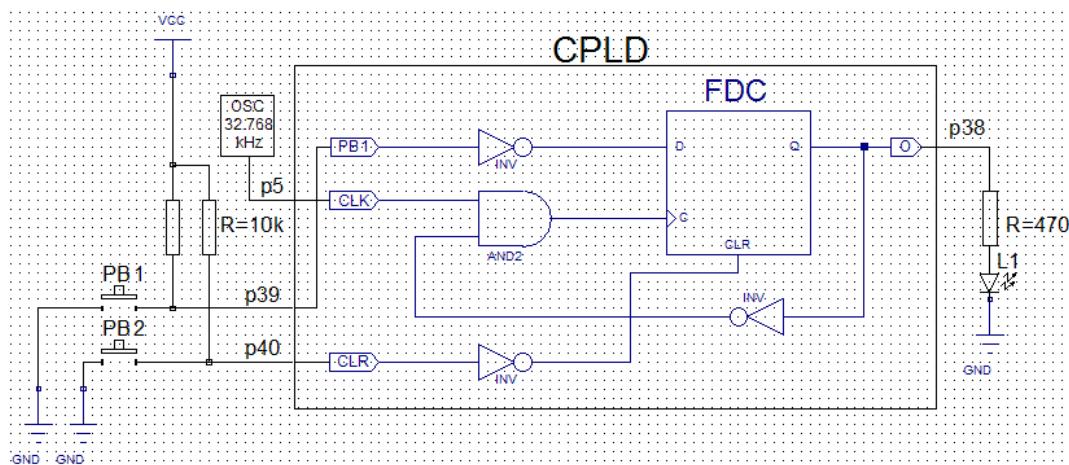


รูปที่ 2.119 ขั้นตอนการสร้าง Implementation constraints file โดยอัตโนมัติ



รูปที่ 2.120 การยืนยันสร้าง Implementation constraints file โดยอัตโนมัติ

2) ขั้นตอนกำหนดสายสัญญาณต่างๆของวงจรในรูปที่ 2.104 เข้ากับขาของ CPLD นั้นจำเป็นต้องกำหนดให้สอดคล้องกับอุปกรณ์ที่เตรียมไว้ที่บอร์ด CPLD Explorer XC9572XL ตามตารางที่ 1.3 ในบทที่ 1 ซึ่งมีรายละเอียดังรูปที่ 2.121



รูปที่ 2.121 การต่อ I/O ของ CPLD (เฉพาะขาที่ใช้งาน) กับอุปกรณ์ที่อยู่บนบอร์ดทดลอง

ดังนี้การกำหนดสายสัญญาณต่างๆ ของวงจรที่ออกแบบในรูปที่ 2.121 เข้ากับขาของ CPLD จะได้ดังนี้

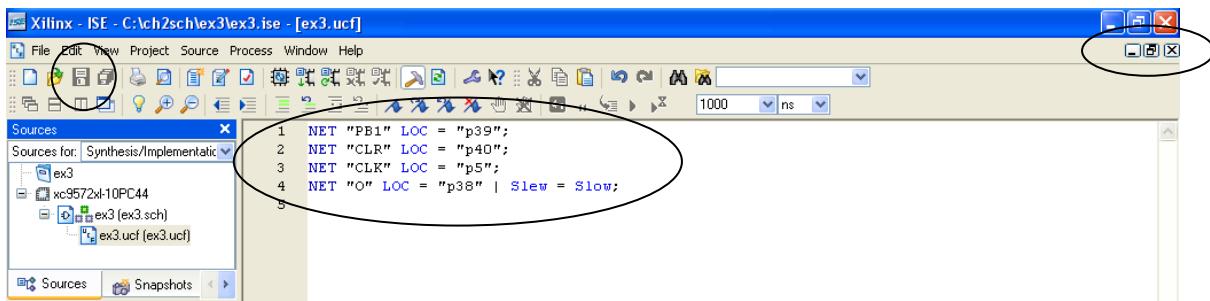
PB1 = PB1 = INPUT = p39 CLK = OSC = INPUT = p5

CLR = PB2 = INPUT = p40 O = L1 = OUTPUT= p38

ขั้นตอนการกำหนดสายสัญญาณเข้ากับขา CPLD ให้พิมพ์รายละเอียดต่างๆใน Edit Constraints (Text) ดังนี้

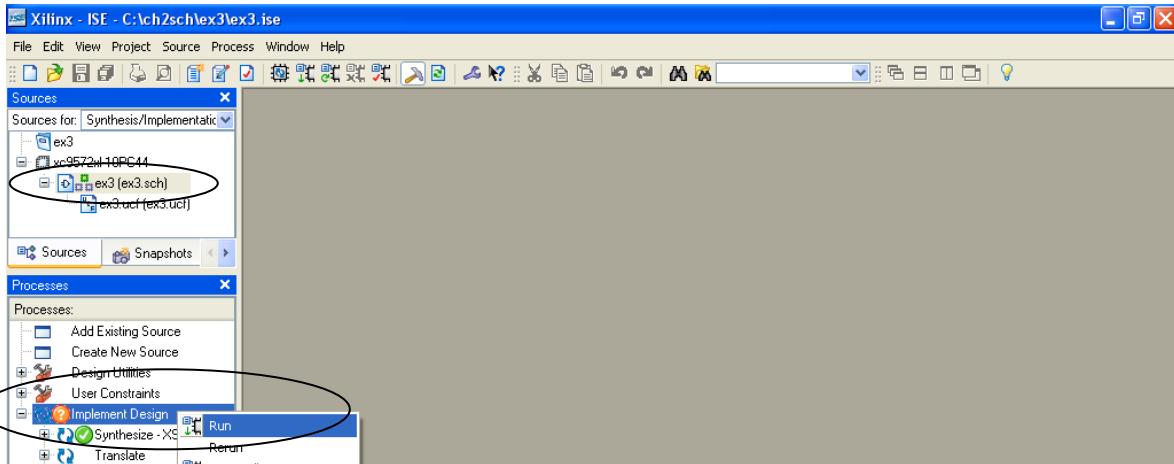
```
NET "PB1" LOC = "p39";
NET "CLR" LOC = "p40";
NET "CLK" LOC = "p5";
NET "O" LOC = "p38" | Slew = Slow;
```

เมื่อพิมพ์เสร็จแล้วจะได้รูปที่ 2.122 จากนั้นบันทึกไฟล์โดยคลิก คลิก (ลีคำ) ที่มุมบนขวาในรูปที่ 2.122 เพื่อปิดโปรแกรม Edit Constraints (Text) และกลับไปที่หน้าต่าง Xilinx-ISE อีกรอบ



รูปที่ 2.122 การกำหนดขาสัญญาณเข้ากับขาชิพ CPLD ใน Edit Constraints (Text)

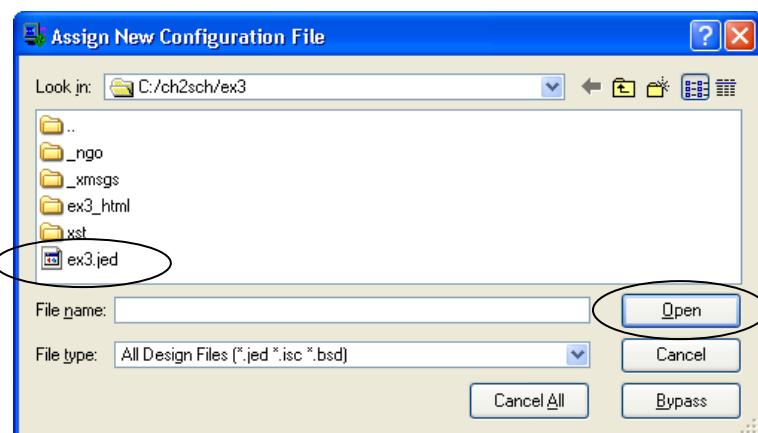
3) ขั้นตอน Implement Design คลิก ex3(ex3.sch) ที่หน้าต่าง Source คลิกขวาที่ Implement Design แล้วคลิก Run ที่หน้าต่าง Processes ดังรูปที่ 2.123 แล้วถ้าใช่ ✓ หรือ ⚡ ที่หน้า Implement Design อีกว่าขั้นตอน Implement ผ่าน



รูปที่ 2.123 การทำ Implement Design

2.3.5 การโปรแกรมข้อมูลวงจรลงชิพ

Generate Programming File นั้นซอฟต์แวร์ทุกได้ทำไปแล้วในขั้นตอน Implement Design นำข้อมูลนี้ไปดาวน์โหลดลงบอร์ด CPLD Explorer XC9572XL โดยใช้สาย JTAG ซึ่งขั้นตอนต่างๆ จะเหมือนกับตัวอย่างที่ 2.1 ทุกประการ แต่จะใช้ไฟล์ชื่อ ex3.jed ที่อยู่ในหน้าต่าง Assign New Configuration File ดังรูปที่ 2.124 เมื่อดาวน์โหลดข้อมูลลง CPLD แล้วให้กดปุ่ม PB1 และ PB2 แล้วสังเกตที่ LED1 ว่าให้ผลตามที่ออกแบบหรือไม่ แล้วปิดหน้าต่าง iMPACT และปิดโปรแกรม ISE WebPACK



รูปที่ 2.124 หน้าต่าง Assign New Configuration File

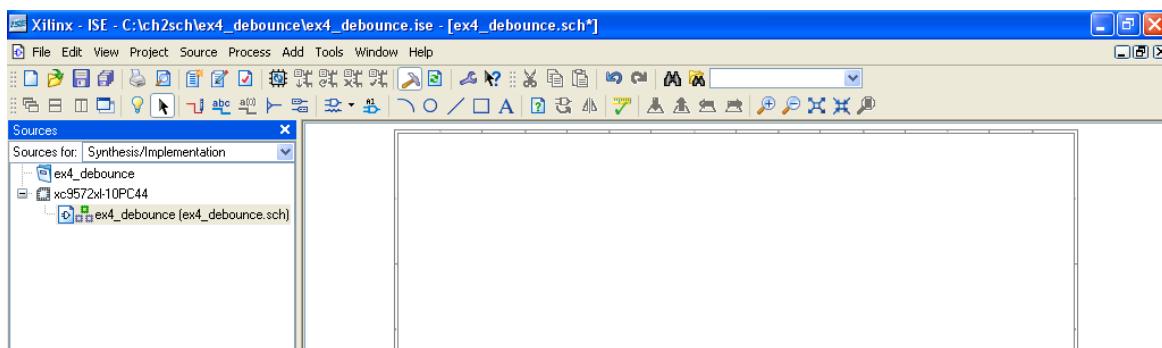
2.4 การสร้าง Source File ที่ได้จากวิธี Schematic เพื่อเก็บไฟล์ไว้ทำ Symbols สำหรับ CPLD

วงจรที่ 2.104 ประกอบด้วยช่วงจราจรย่อย เช่น วงจรนับสิบ, 2 to 4 decode, BCD to seven segments และ โนมโนนสเตเบิล (Mono stable) เป็นต้น เราจึงนิยมเก็บไฟล์วงจรเหล่านี้ไว้ทำ Symbols เพื่อนำไปใช้งาน ไม่ต้องเสียเวลาออกแบบซ้ำอีก

ตัวอย่างที่ 2.4 เราจะนำวงจรโนมโนนสเตเบิลในรูปที่ 2.104 ที่เคยออกแบบไว้แล้วไปสร้างไฟล์ Symbols เพื่อเก็บไว้ใช้งาน ซึ่งในการสร้างไฟล์ Symbols ก็คือขั้นตอนการออกแบบวงจร (Design entry)

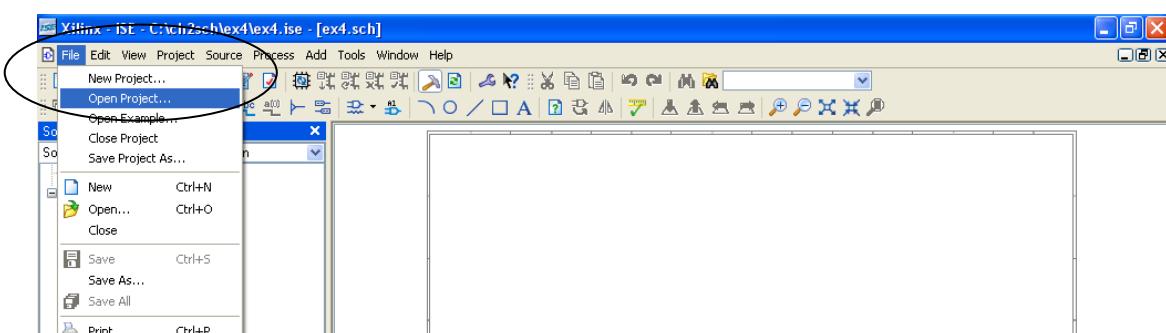
2.4.1 ขั้นตอนการออกแบบวงจร

1) การออกแบบ CPLD ตัวอย่างนี้ให้ใช้ Project Location ชื่อ ch2sch โดยมี Project Name และ Source File ชื่อ ex4_debounce เลือก CPLD ตระกูล (Family) XC9500XL เบอร์ (Device) XC9572XL ที่มี Package แบบ PLCC 44 ขา (Package : PC44) และ Speed Grade : -10 จากนั้นทำการออกแบบวงจร (Design entry)

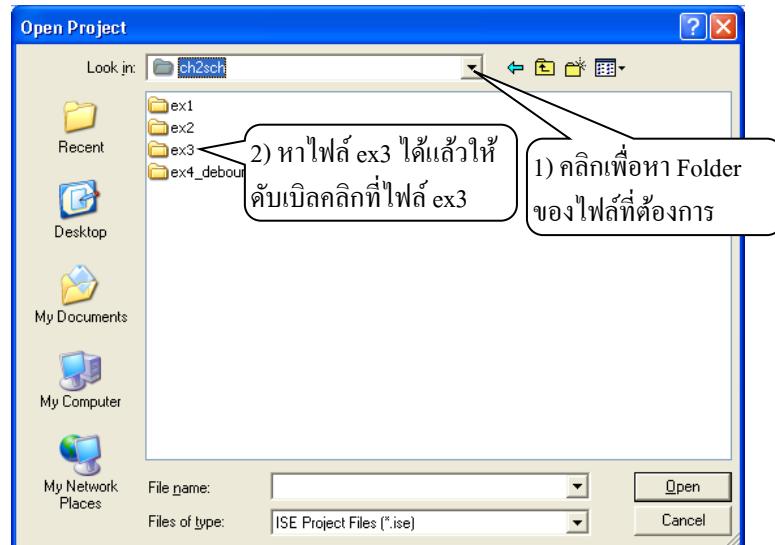


รูปที่ 2.125 หน้าต่าง Xilinx-ISE สำหรับ Schematic editor ของไฟล์ ex4_debounce

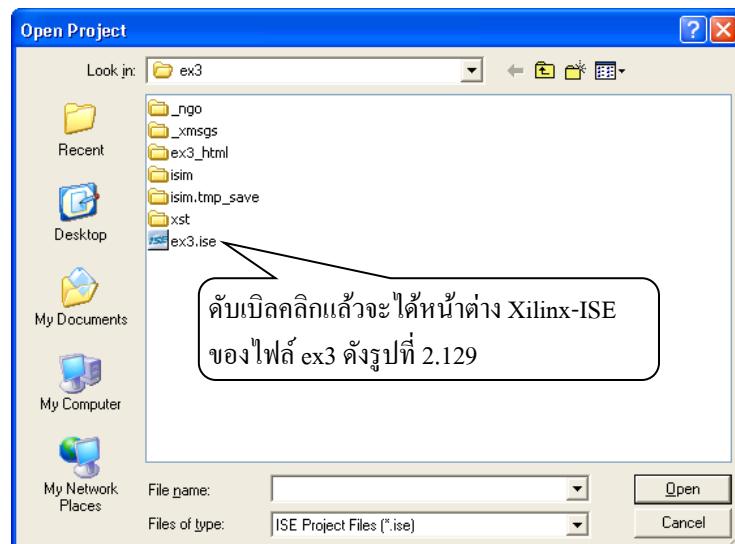
2) การตรวจสอบ ทำการตรวจสอบในรูปที่ 2.104 หรืออาจใช้วิธี Copy ผังวงจรในรูปที่ 2.106 ไปใช้ โดยคลิก File -> Open Project ดังรูปที่ 2.126 แล้วจะหน้าต่างดังไป จากนั้นคลิกหาไฟล์ชื่อ ex3 ดังรูปที่ 2.127 (ซึ่งตำแหน่งไฟล์ในรูปที่ 2.127 อาจจะไม่เหมือนกัน ขึ้นอยู่กับจำนวนไฟล์ที่อยู่ใน Folder นั้น) จากนั้นดับเบิลคลิกที่ ex3 และจะได้ ex3.ise ดังรูปที่ 2.128 เมื่อดับเบิลคลิกที่ไฟล์ ex3.ise ก็จะได้หน้าต่าง Xilinx-ISE ของไฟล์ ex3 ดังรูปที่ 2.129



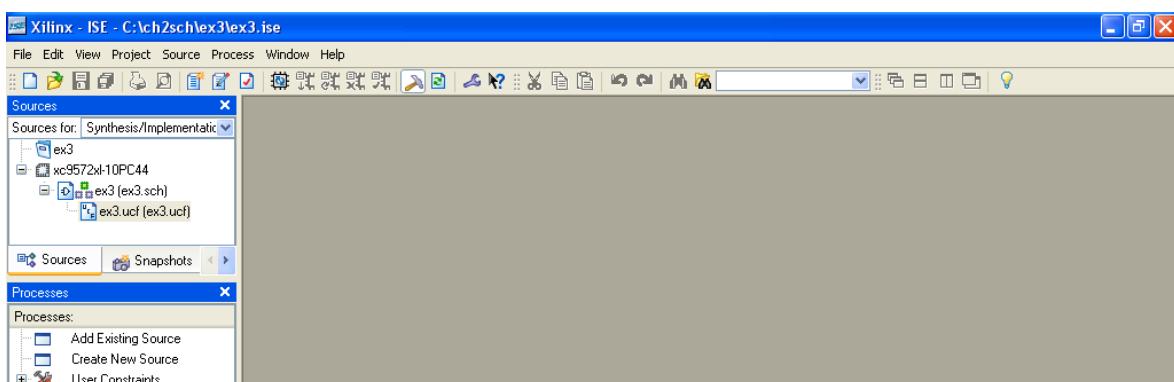
รูปที่ 2.126 หน้าต่าง Xilinx-ISE ขณะคลิกเพื่อ Open Project ที่เคยออกแบบไว้แล้ว



รูปที่ 2.127 หน้าต่าง Open Project (ตำแหน่งไฟล์ ex3 จะไม่แน่นอน ขึ้นอยู่กับจำนวนไฟล์ใน Folder นั้น)

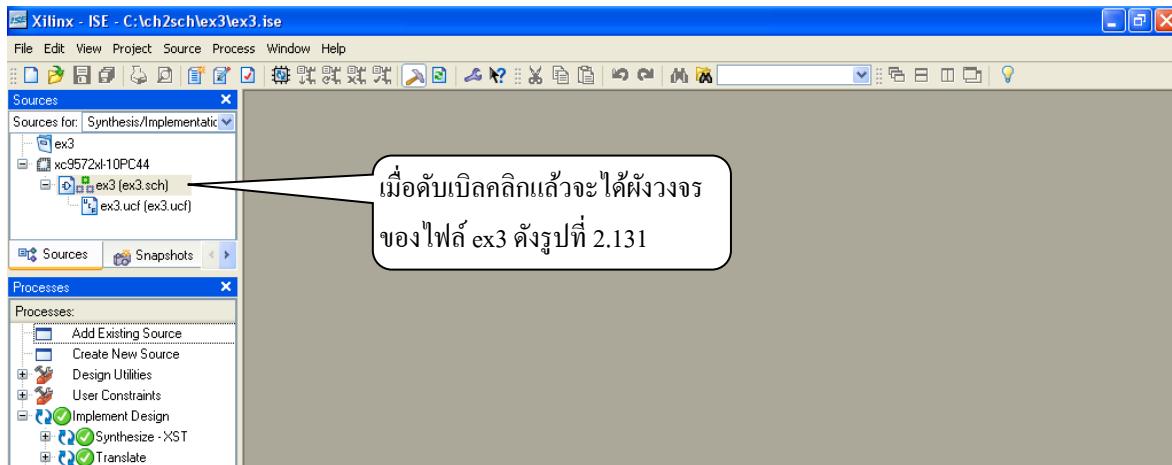


รูปที่ 2.128 หน้าต่าง Open Project

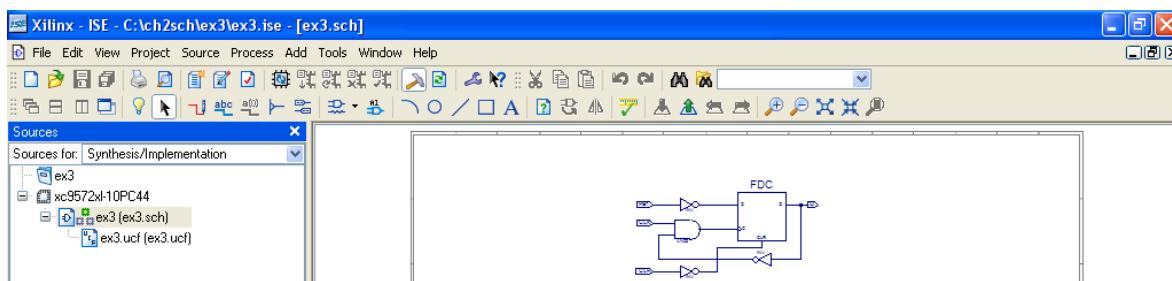


รูปที่ 2.129 หน้าต่าง Xilinx-ISE ของไฟล์ ex3

3) ที่หน้าต่าง Xilinx-ISE คลิกที่ ex3 ในหน้าต่าง Source ดังรูปที่ 2.130 แล้วคลิก “+” ที่อยู่หน้าแถบ Implement Design ในหน้าต่าง Processes จะเป็น “-” ก็จะได้หน้าต่าง หน้าต่าง Xilinx-ISE สำหรับ Schematic editor ของไฟล์ ex3 ดังรูปที่ 2.131

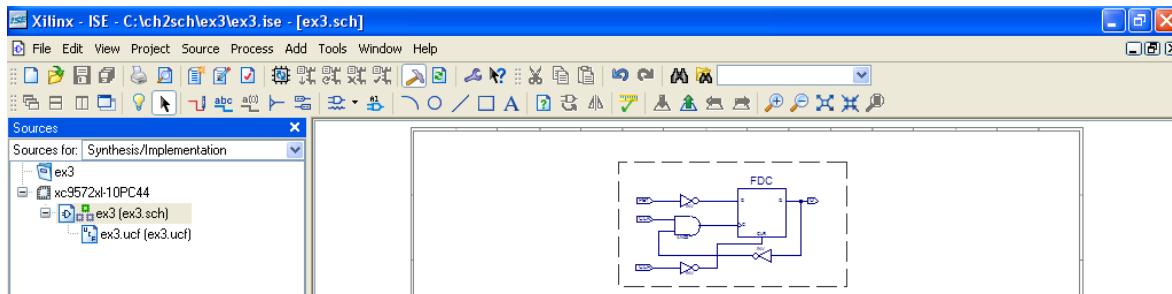


รูปที่ 2.130 หน้าต่าง Xilinx-ISE ของไฟล์ ex3

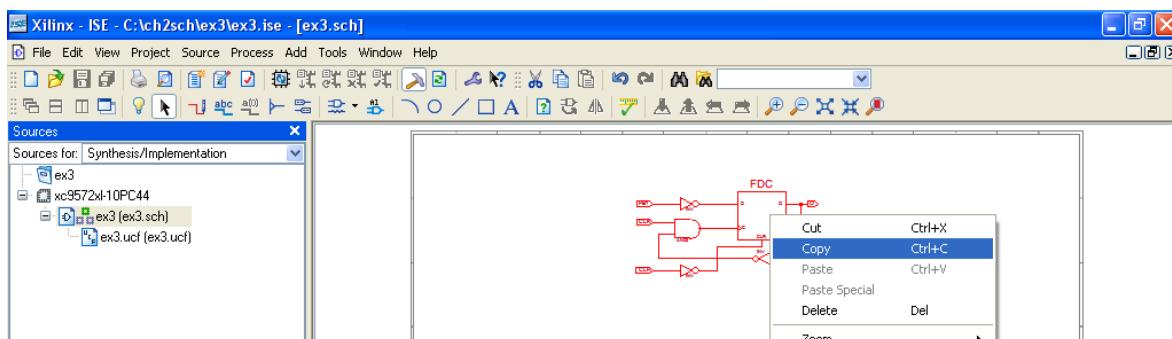


รูปที่ 2.131 หน้าต่าง Xilinx-ISE สำหรับ Schematic editor ของไฟล์ ex3

4) ที่หน้าต่าง หน้าต่าง Xilinx-ISE สำหรับ Schematic editor คลิกมาส์ค้างไว้แล้วลากท移交จากซ้ายไปขวาดังรูปที่ 2.132 แล้ว ผังวงจรจะถูกปิดเป็นสีแดง (ถูก Select) คลิกขวาแล้วคลิก Copy ดังรูปที่ 2.133

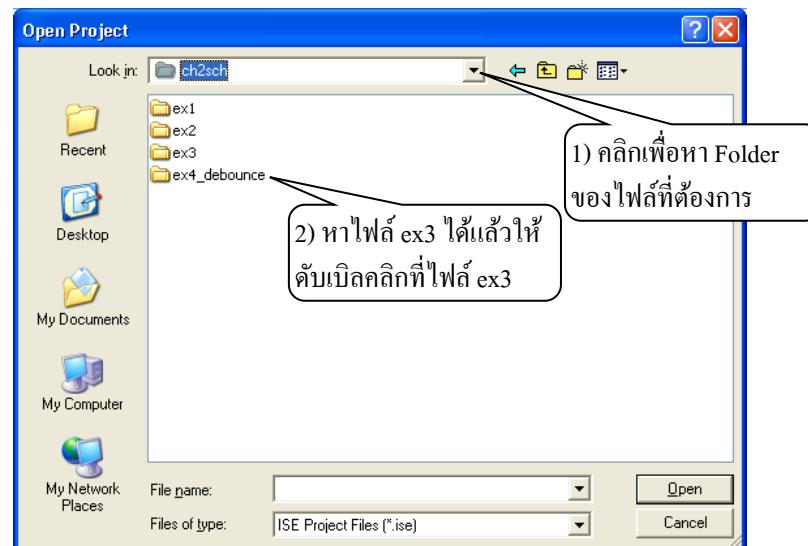


รูปที่ 2.132 หน้าต่าง Xilinx-ISE

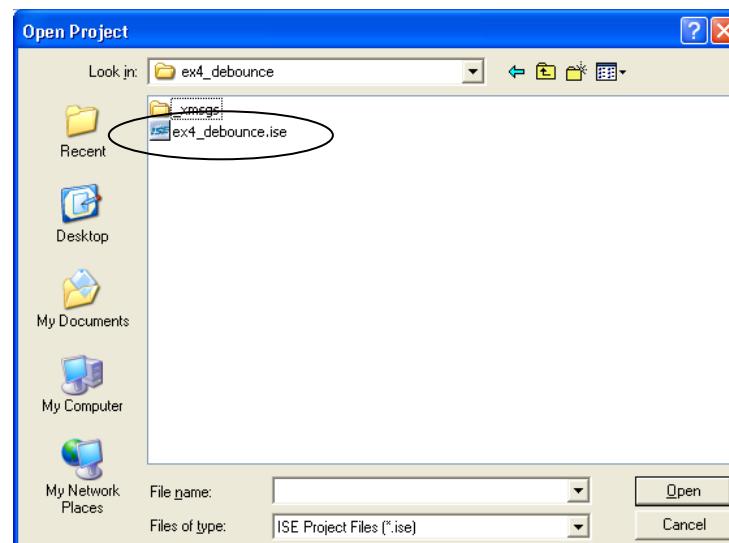


รูปที่ 2.133 หน้าต่าง Xilinx-ISE ขณะ Copy

5) คลิก File > Open Project และจะได้หน้าต่าง Open Project จากนั้นคลิกท่า Folder ชื่อ ch2sch และหาไฟล์ชื่อ ex4_debounce ดังรูปที่ 2.134 และให้ดับเบิลคลิกที่ ex4_debounce และจะได้ดังรูปที่ 2.135 ดับเบิลคลิก ex4_debounce.ise และจะได้หน้าต่าง Xilinx-ISE ดังรูปที่ 2.136 ดับเบิลคลิกที่ ex4_debounce ในหน้าต่าง Source และจะได้หน้าต่างสำหรับตรวจสอบผังวงจร จากนั้นคลิกปุ่ม (Paste) และจึงเลื่อนมาสู่ไปยัง ณ ตำแหน่งที่ต้องการ และคลิกตรวจสอบผังวงจรดังรูปที่ 2.137 คลิก บันทึกไฟล์ ขึ้นตอนนี้ถือว่าการตรวจสอบสำหรับเก็บไว้ทำ Symbol และเสร็จสมบูรณ์ คลิก เพื่้ออกจากโปรแกรม ISE WebPACK



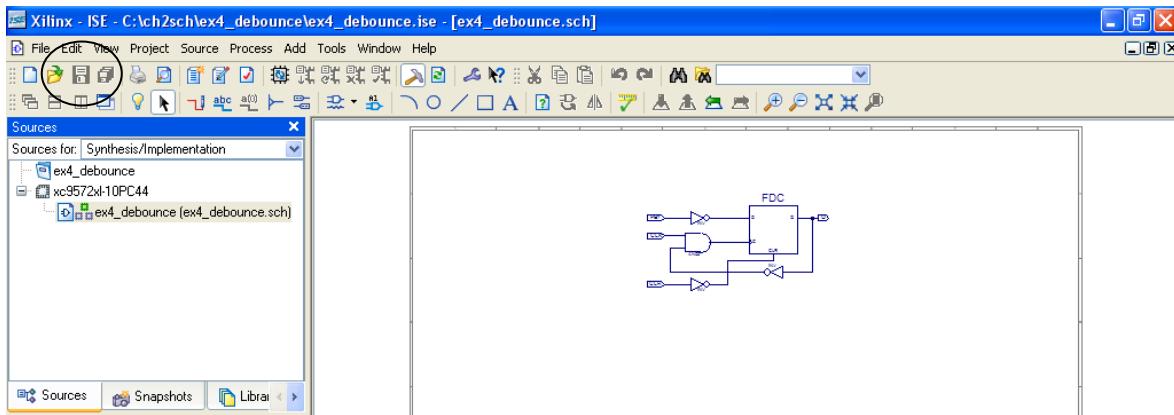
รูปที่ 2.134 หน้าต่าง Open Project (ตำแหน่งไฟล์ ex4_debounce จะไม่แน่นอน ขึ้นอยู่กับจำนวนไฟล์ใน Folder นั้น)



รูปที่ 2.135 หน้าต่าง Open Project



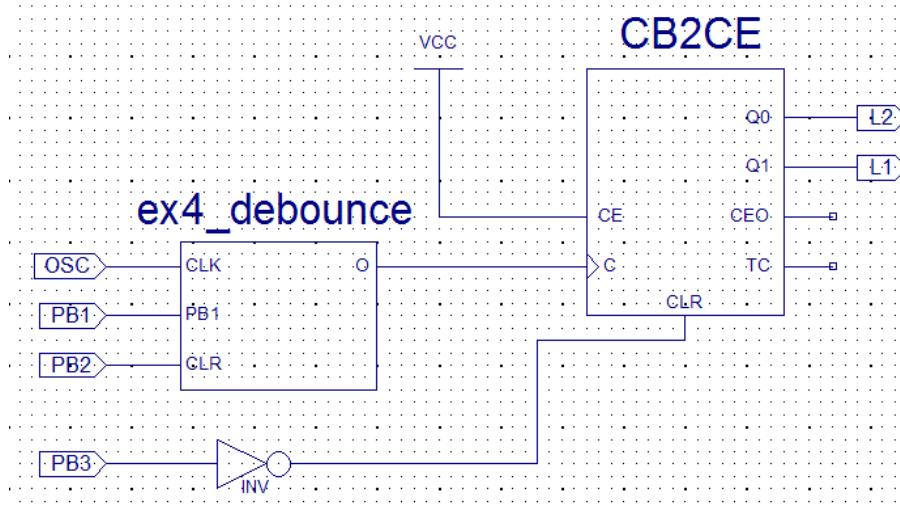
รูปที่ 2.136 หน้าต่าง Xilinx-ISE ของไฟล์ชื่อ ex4_debounce



รูปที่ 2.137 หน้าต่าง Xilinx-ISE สำหรับ Schematic editor ของไฟล์ ex4_debounce หลังจาก Paste แล้ว

2.5 การออกแบบวงจรนับ 4 โดยใช้ CPLD และมีโมโนสเตเบิลอย่างง่ายเป็นวงจรแก้เบาช์

ตัวอย่างที่ 2.5 ออกแบบวงจรนับ 4 แบบซิงไครนัสที่มีวงจรโมโนสเตเบิลในการแก้ปัญหาการเกิดเบาช์ (Bouncing) ของปุ่มกด โดยวงจรทั้งหมดแสดงดังรูปที่ 2.138 ขั้นตอนการออกแบบเป็นดังนี้

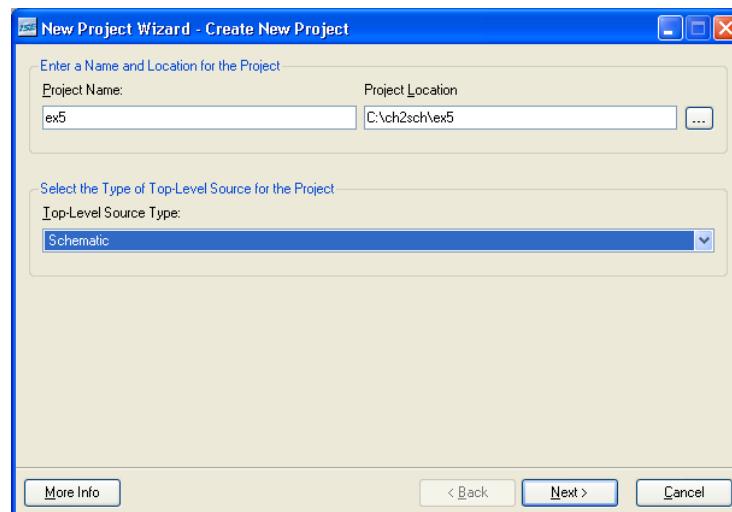


รูปที่ 2.138 วงจรนับ 4 แบบซิงไครนัสที่มีวงจรโมโนสเตเบิลในการแก้ปัญหาการเกิดเบาช์

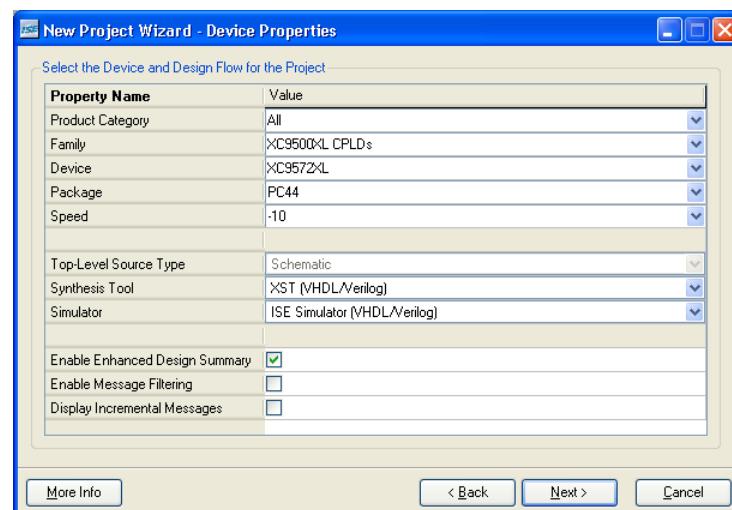
2.5.1 ขั้นตอนการออกแบบวงจร



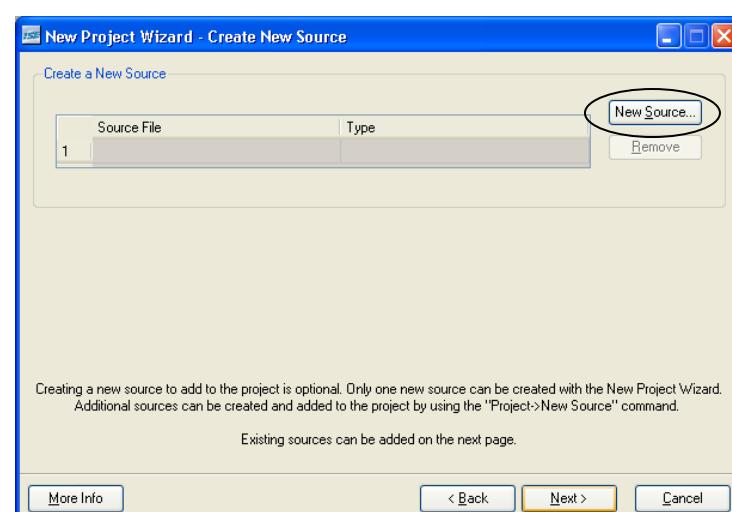
- 1) ดับเบิลคลิกที่ Xilinx ISE 8.1i แล้วจะได้หน้าต่าง Xilinx ISE ซึ่มีหน้าต่าง Tip of the Day ข้อนี้มาให้คลิก OK จากนั้นคลิก File -> New Project จะได้หน้าต่าง New Project Wizard พิมพ์ชื่อ ch2sch ที่ Project Location และชื่อ ex5 ที่ Project Name คลิกที่ Top-Level Module Type เป็น Schematic ดังรูปที่ 2.139 คลิก Next แล้วจะได้หน้าต่าง New Project Wizard–Device
2) ที่หน้าต่าง New Project Wizard–Device Properties คลิกเลือกดังรูปที่ 2.140 คลิก Next แล้วจะได้หน้าต่างถัดไปดังรูปที่ 2.141



รูปที่ 2.139 New Project Wizard-Create New Project

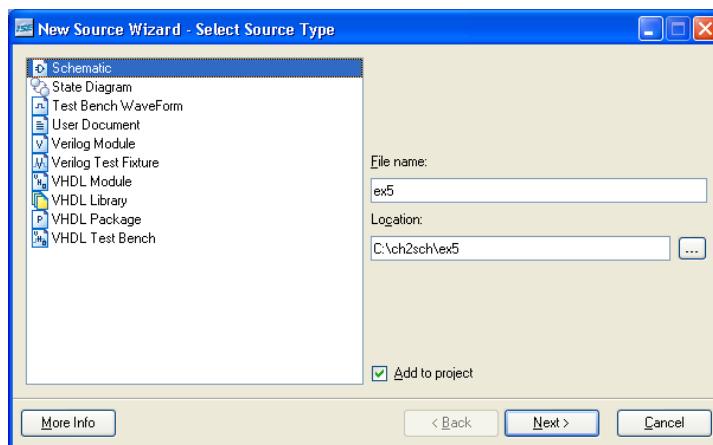


รูปที่ 2.140 New Project Wizard-Device Properties



รูปที่ 2.141 New Project Wizard-Create New Source

- 4) คลิกปุ่ม New Source จะได้หน้าต่างดังไป พิมพ์ชื่อ ex5 และคลิกที่ Schematic ดังรูปที่ 2.142 คลิก Next แล้วคลิก Finish และคลิก Next อีกครั้งแล้วจะได้หน้าต่าง New Project Wizard-Add Existing Source เพื่อใช้เพิ่มไฟล์ที่จะนำไปสร้างเป็น Symbols

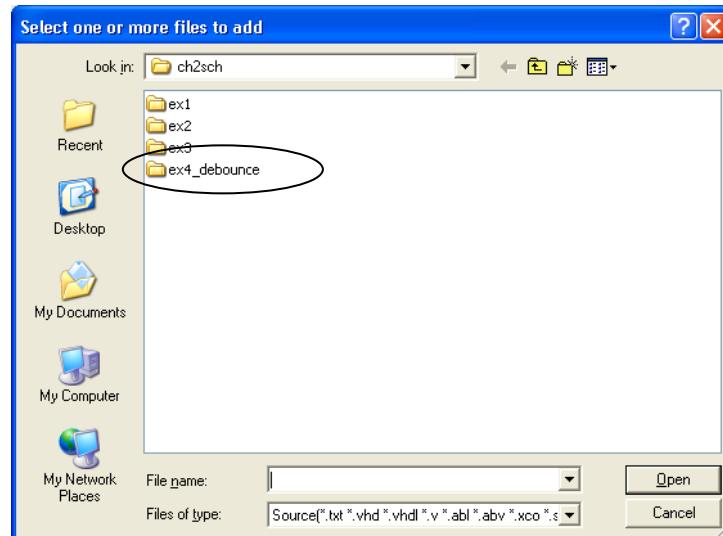


รูปที่ 2.142 New Source Wizard-Select Source Type

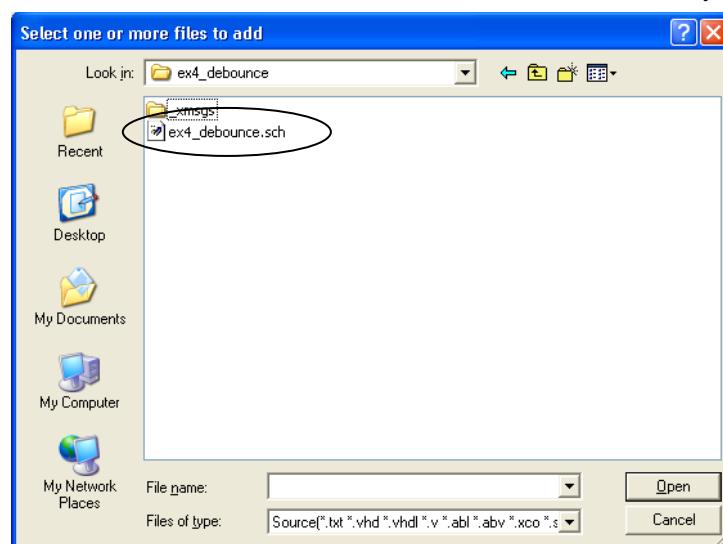
- 5) คลิกปุ่ม Add Source ดังรูปที่ 2.143 แล้วจะได้หน้าต่างดังไป หาไฟล์วงจรโน�ต์เดบอนซ์ ex4_debounce และดับเบิลคลิกที่ ex4_debounce ในรูปที่ 2.144 จากนั้นดับเบิลคลิกที่ ex4_debounce.sch ในรูปที่ 2.145 แล้วจะได้หน้าต่างดังรูปที่ 2.146



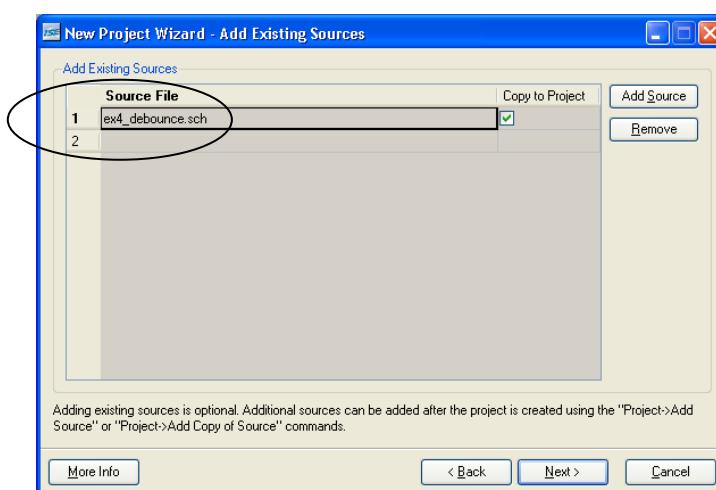
รูปที่ 2.143 New Project Wizard-Add Existing Source เพื่อใช้เพิ่มไฟล์ที่จะนำไปสร้างเป็น Symbols



รูปที่ 2.144 Select one or more files to add (เลือกไฟล์ที่จะนำไปสร้างเป็น Symbols)

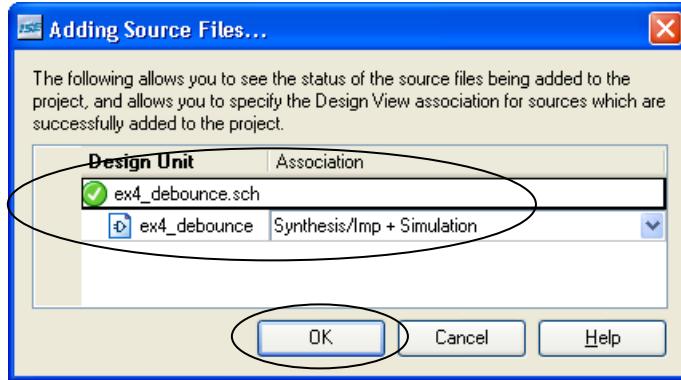


รูปที่ 2.145 Select one or more files to add (เลือกไฟล์ที่จะนำไปสร้างเป็น Symbols)

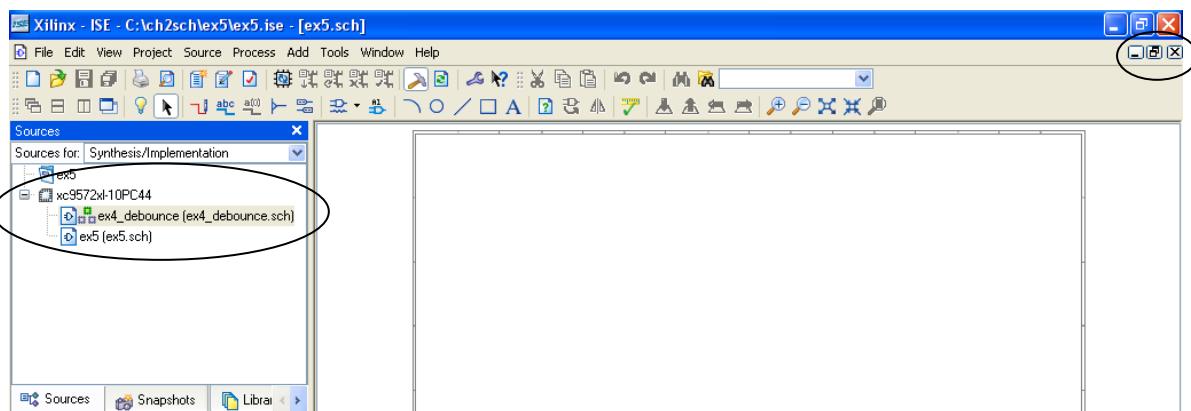


รูปที่ 2.146 หน้าต่าง New Project Wizard-Add Existing Source หลังจากเพิ่มไฟล์ที่จะนำไปสร้างเป็น Symbolsแล้ว

6) ในรูปที่ 2.146 ไฟล์ ex4_debounce.sch จะถูก Add เข้าไป (และถ้าต้องการ Add หลายไฟล์ให้ทำขั้นตอนซ้ำในรูปที่ 2.143 ถึงรูปที่ 2.146 จากนั้นคลิก Next แล้วคลิก Finish และคลิก OK ในรูปที่ 2.147 แล้วจะได้หน้าต่าง Xilinx-ISE ดังรูปที่ 2.148

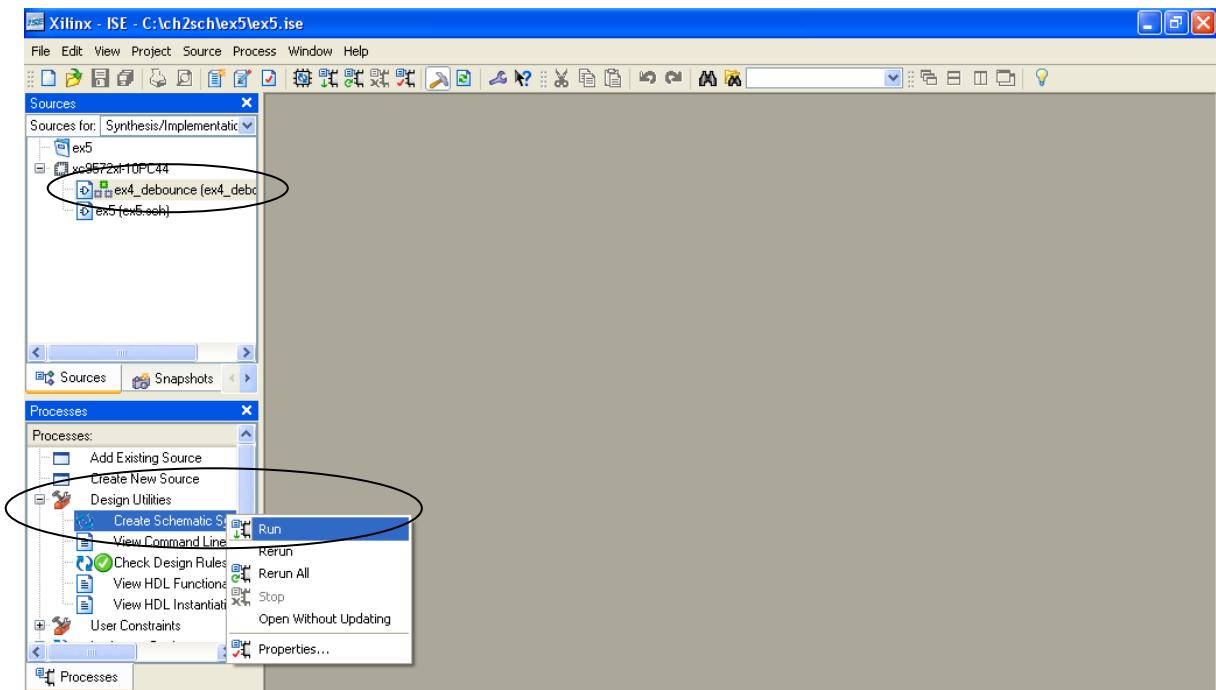


รูปที่ 2.147 Adding Source Files



รูปที่ 2.148 หน้าต่าง Xilinx-ISE (จะปรากฏไฟล์ ex4_debounce ที่จะนำไปสร้างเป็น Symbols)

7) ขั้นตอนการสร้าง Symbols (Create schematic symbol) ของวงจรโนนิสเตเบิลชื่อ ex4_debounce คลิก **X** (สีดำ) ที่หน้าต่าง Xilinx-ISE ในรูปที่ 2.148 เพื่อเปิดโปรแกรม Schematic editor หรือ ECS และไปทำขั้นตอนสร้าง Symbols จากนั้นจึงคลิกไฟล์ ex4_debounce(ex4_debounce.sch) ที่หน้าต่าง Source และคลิก“+”หน้า Design Utilities ที่หน้าต่าง Processes จนเป็น“-” คลิกขวาที่ Create Schematic Symbol แล้วคลิก Run ดังรูปที่ 2.149 ซึ่งขั้นตอนนี้ถือว่าสร้าง Symbols วงจรโนนิสเตเบิลแล้วเสร็จ

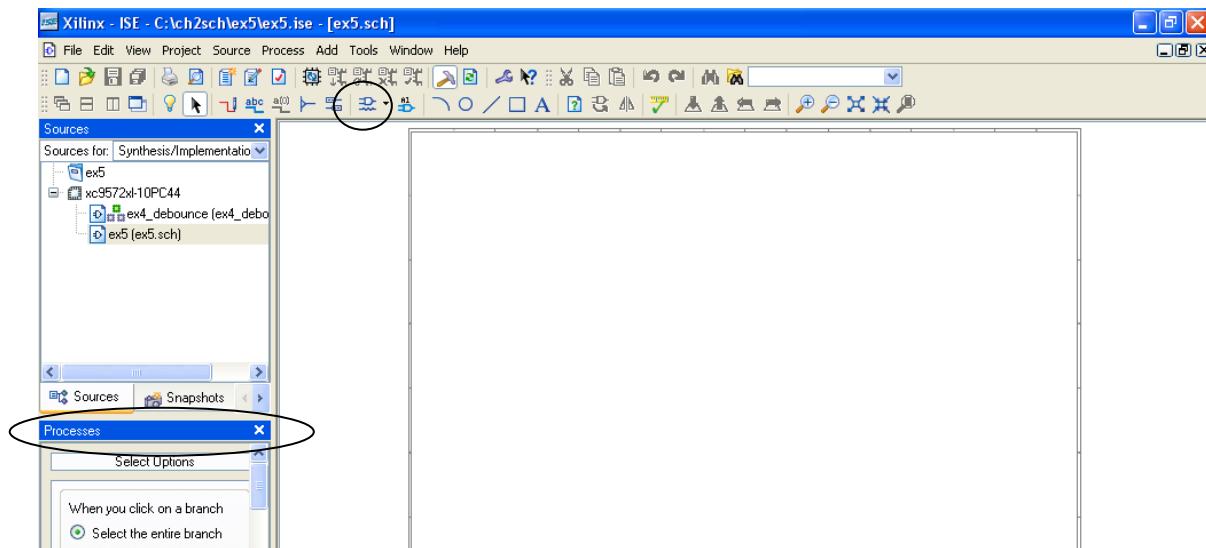


รูปที่ 2.149 หน้าต่าง Xilinx-ISE ขณะ Create Schematic Symbol

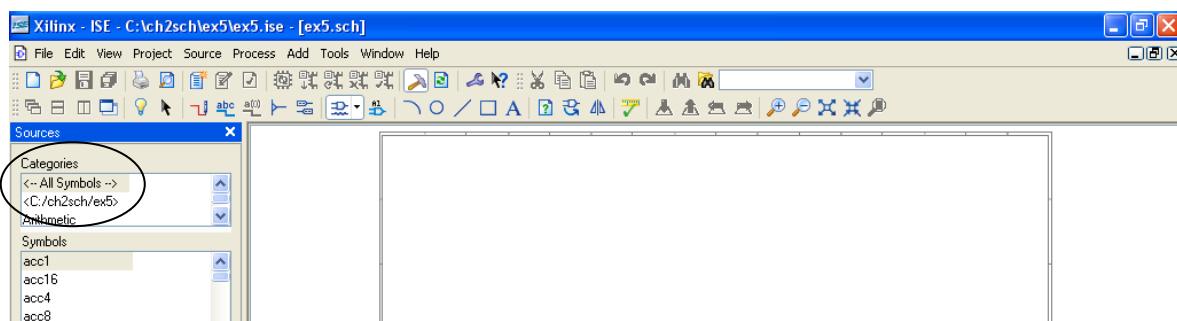
8) ขั้นตอนวิธีการพัฒนา ดับเบิลคลิก ex5(ex5.sch) ที่หน้าต่าง Source ดังรูปที่ 2.150 เพื่อเปิดโปรแกรม Schematic editor อีกครั้ง แล้วจะได้ดังรูปที่ 2.151 เมื่อกลับไปอีกครั้ง ที่อยู่บน Tool Bar และคลิก ของแถบสีน้ำเงินของหน้าต่าง Processes แล้วจะได้รายการของหมวดอุปกรณ์ใน Categories และชนิดอุปกรณ์ใน Symbols ดังรูปที่ 2.152 ซึ่งมีรายการ <c:/ch2sch/ex5> อยู่ในหน้าต่าง Categories เพิ่มเข้ามา เมื่อเราคลิกที่รายการ <c:/ch2sch/ex5> ในหน้าต่าง Categories ก็จะปรากฏอุปกรณ์ตัวใหม่ในหน้าต่าง Symbols คือ ex4_debounce ดังรูปที่ 2.153 ซึ่งเป็นวงจรโน�टิฟิลท์เรสอร์วิสใน Symbols นั่นเอง จากนั้นทำการวิเคราะห์วงจรและแก้ไขตามที่ต้องการ



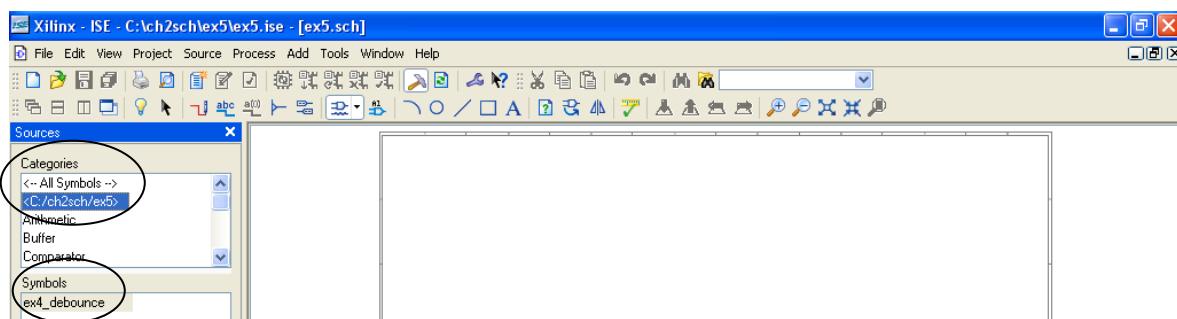
รูปที่ 2.150 ขั้นตอนดับเบิลคลิก ex5(ex5.sch) ที่หน้าต่าง Source เพื่อเปิดโปรแกรม Schematic editor หรือ ECS



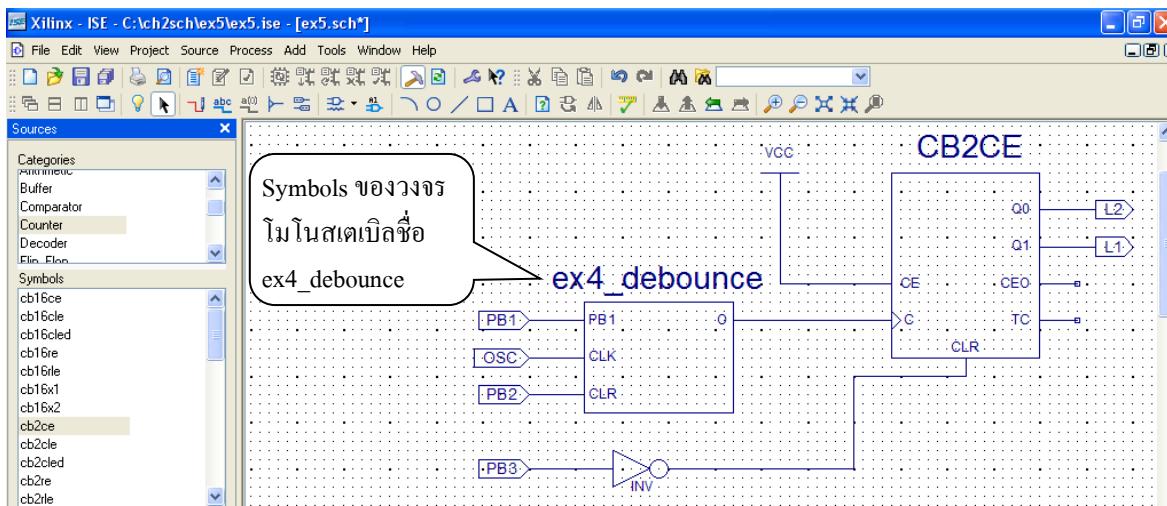
รูปที่ 2.151 หน้าต่าง Xilinx-ISE สำหรับใช้วัดผังวงจรหลักที่เราต้องการออกแบบ



รูปที่ 2.152 หน้าต่าง Xilinx-ISE ที่มีรายการ <c:/ch2sch/ex5> อยู่ในหน้าต่าง Categories

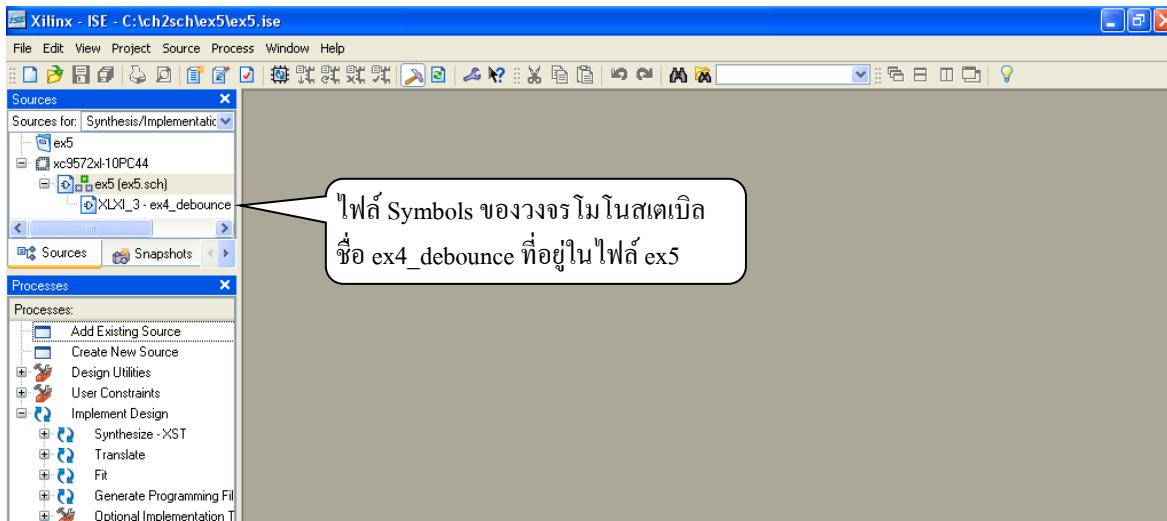


รูปที่ 2.153 หน้าต่าง Xilinx-ISE ที่มีอุปกรณ์ตัวใหม่ในหน้าต่าง Symbols คือ ex4_debounce



รูปที่ 2.154 หน้าต่าง Xilinx-ISE ที่วาดผังวงจรแล้วเสร็จสมบูรณ์

8) ตรวจความถูกต้อง (Syntax) คลิกที่ ถ้าไม่มีข้อผิดพลาดก็ให้บันทึกไฟล์โดยคลิก ขั้นตอนนี้ถือว่าการออกแบบวงจรแล้วเสร็จ คลิก (สีดำ) เพื่อปิดโปรแกรม Schematic editor เพื่อกลับไปที่หน้าต่าง Xilinx-ISE อีกครั้ง คลิก View -> Restore Default Layout แล้วคลิก “+” ที่หน้า ex5(ex5.sch) ที่หน้าต่าง Sources จะเป็น “-” และคลิก “+” ที่หน้า Implement Design ที่หน้าต่าง Processes จะเป็น “-” จะได้ดังรูปที่ 2.155



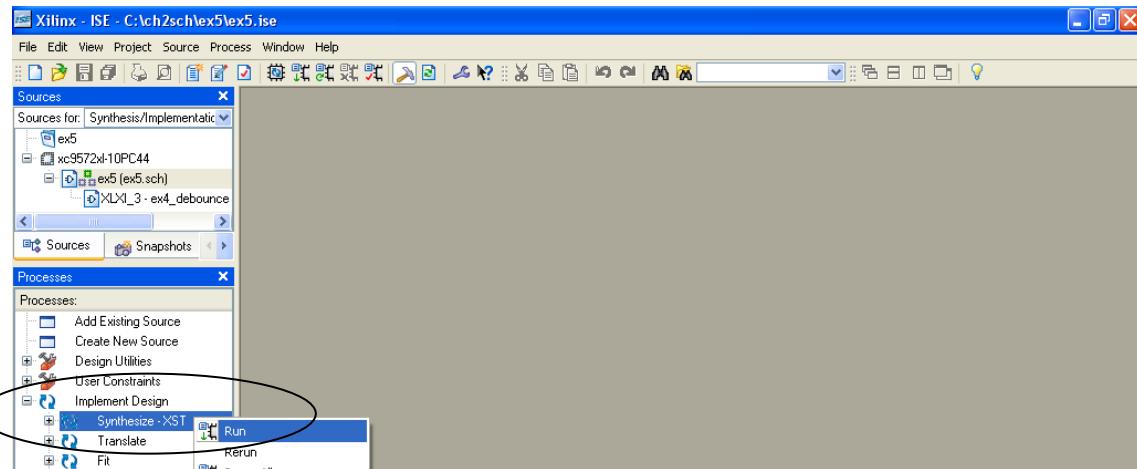
รูปที่ 2.155 หน้าต่าง Xilinx-ISE

2.5.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design verification)

การออกแบบวงจรอย่างง่ายอาจไม่มีความจำเป็นต้องทำขั้นตอนนี้

2.5.3 การสังเคราะห์วงจร (Design synthesis)

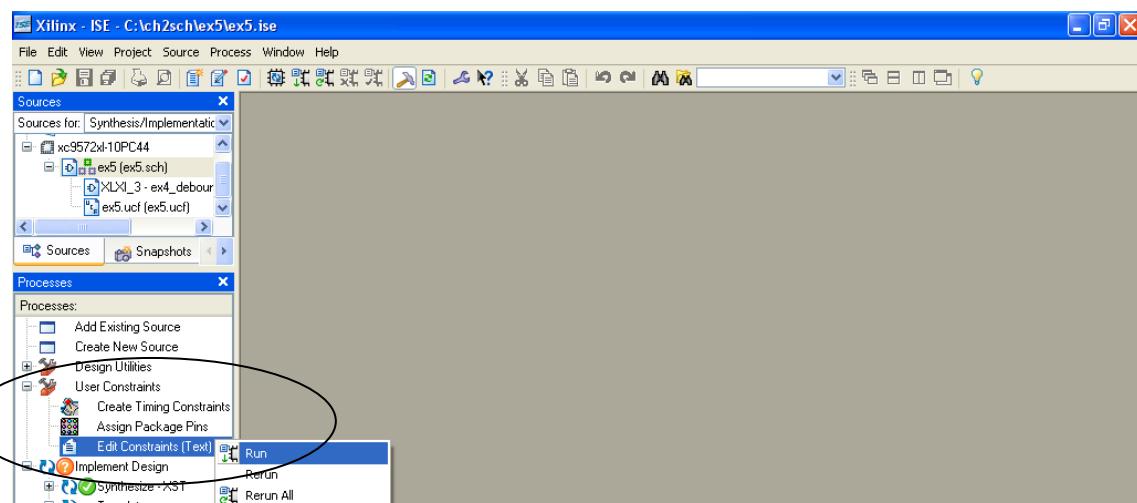
ที่หน้าต่าง Xilinx-ISE คลิก “+” ที่หน้า Implement Design ในหน้าต่าง Processes จะเป็น “-” คลิกขวาแล้วคลิก Run ที่ Synthesize-XST ดังรูปที่ 2.156 เตรียมแล้วล้า荷ร์ หรือ ที่หน้า Synthesize-XST ถ้าไม่ต้องการคำแนะนำ



รูปที่ 2.156 แสดงหน้าต่าง Processes แสดงขั้นตอนสังเคราะห์วงจร

2.5.4 Design Implementation

- 1) ขั้นตอนสร้าง Implementation constraints file (UCF) โดยอัตโนมัติทำได้โดยคลิก “+” ที่อยู่หน้า User Constraints ในหน้าต่าง Processes จนเป็น “-” คลิกขวาแล้วคลิก Run ที่ Edit Constraints(Text) ดังรูปที่ 2.157 จากนั้นจะมีหน้าต่างซ่อนขึ้นมา ให้คลิก Yes ที่หน้าต่าง Project Navigator แล้ว Implementation constraints file ก็จะถูกสร้างโดยอัตโนมัติ



รูปที่ 2.158 ขั้นตอนการสร้าง Implementation constraints file โดยอัตโนมัติ

- 2) ขั้นตอนกำหนดสายสัญญาณต่างๆของวงจรในรูปที่ 2.138 เข้ากับขา CPLD นั้นจำเป็นต้องกำหนดให้สอดคล้องกับอุปกรณ์ที่เตรียมไว้ที่บอร์ด CPLD Explorer XC9572XL ตามตารางที่ 1.3 ในบทที่ 1 ซึ่งมีรายละเอียดดังนี้

$$\text{PB1} = \text{PB1} = \text{INPUT} = \text{p39} \quad (\text{ไม่มีชื่อ}) = \text{PB3} = \text{INPUT} = \text{p42} \quad \text{L1} = \text{LED1} = \text{OUTPUT} = \text{p38}$$

$$\text{CLR} = \text{PB2} = \text{INPUT} = \text{p40} \quad \text{CLK} = \text{OSC} = \text{INPUT} = \text{p5} \quad \text{L2} = \text{LED2} = \text{OUTPUT} = \text{p37}$$

ขั้นตอนการกำหนดสายสัญญาณเข้ากับขา CPLD ให้พิมพ์รายละเอียดต่างๆใน Edit Constraints (Text) เป็นดังนี้คือ

NET "PB1" LOC = "p39";

NET "PB2" LOC = "p40";

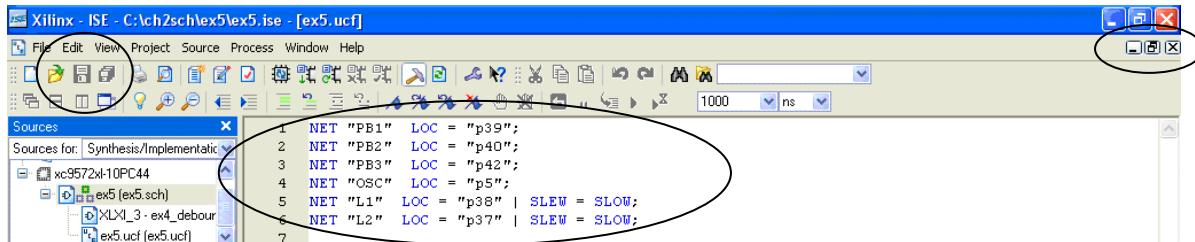
NET "PB3" LOC = "p42";

NET "OSC" LOC = "p5";

NET "L1" LOC = "p38" | Slew = Slow ;

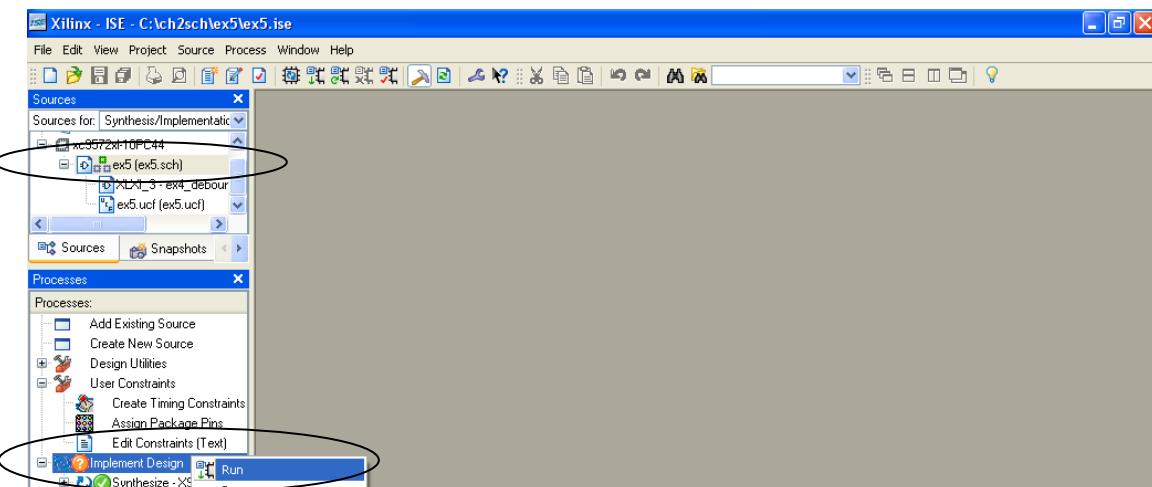
NET "L2" LOC = "p37" | Slew = Slow ;

พิมพ์ดังรูปที่ 2.159 คลิก บันทึกไฟล์ คลิก ปิด Edit Constraints (Text) และกลับไปที่หน้าต่าง Xilinx-ISE



รูปที่ 2.159 การกำหนดขาสัญญาณเข้ากับขาชิพ CPLD ใน Edit Constraints (Text)

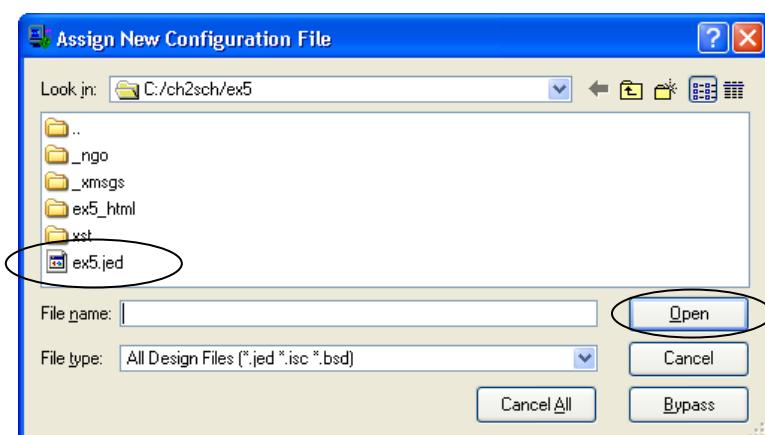
3) ขั้นตอน Implement Design คลิก ex5(ex5.sch) ที่หน้าต่าง Source และคลิก “+” หน้า Implement Design ที่หน้าต่าง Processes จนเป็น “-” คลิกขวาที่ Implement Design แล้วคลิก Run ดังรูปที่ 2.160 ถ้าใช่ หรือ ที่หน้า Implement Design ถือว่าผ่าน



รูปที่ 2.160 การทำ Implement Design

2.5.5 การโปรแกรมข้อมูลวงจรลงชิพ

ขั้นตอนดาวน์โหลดลง CPLD นี้ขั้นตอนต่างๆ จะเหมือนกับตัวอย่างที่ 2.1 ทุกประการ แต่จะใช้ไฟล์ชื่อ ex5.jed ที่อยู่ในหน้าต่าง Assign New Configuration File ดังรูปที่ 2.161



รูปที่ 2.161 หน้าต่าง Assign New Configuration File

เมื่อความนิ่วลดข้อมูลลง CPLD แล้วให้กดปุ่ม PB1 และ PB2 ให้สังเกตที่ LED1 ว่าให้ผลตามที่ออกแบบหรือไม่ แล้วปิดหน้าต่าง iMPACT และปิดโปรแกรม ISE WebPACK

2.6 การสร้าง Source File ที่ได้จากวิธี Schematic เพื่อเก็บไฟล์ไว้ทำ Symbols สำหรับ FPGA

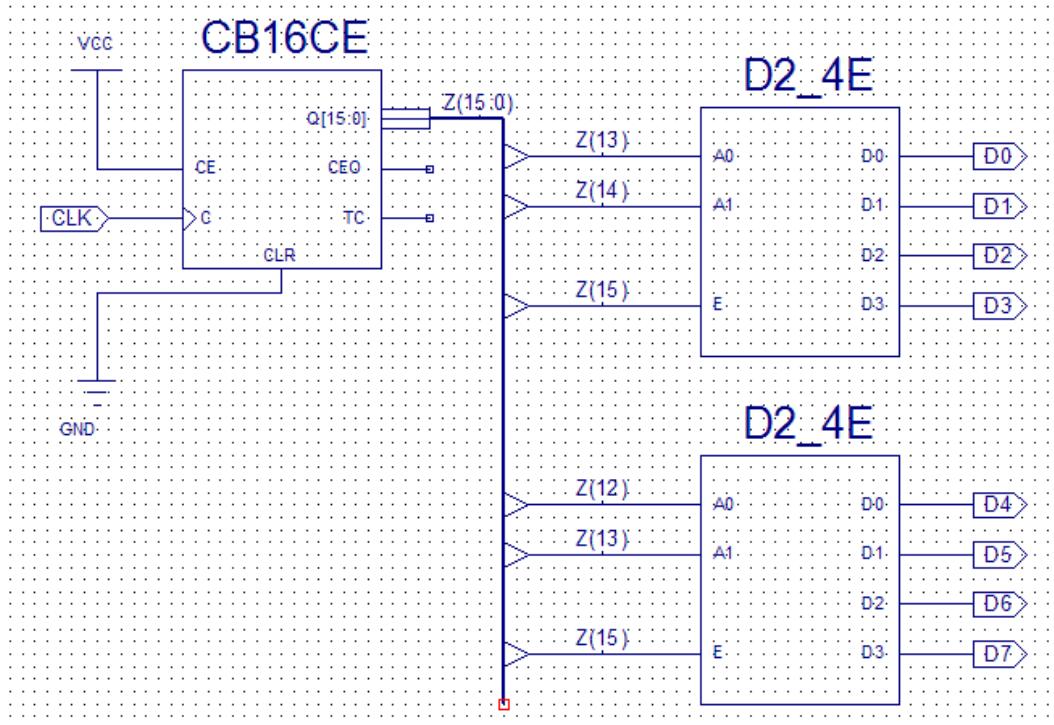
ตัวอย่างที่ 2.6 การสร้าง Source File ที่ได้จากวิธี Schematic เพื่อเก็บไฟล์ไว้ทำ Symbols สำหรับ FPGA มีขั้นตอนเหมือนกับ CPLD ในข้อ 2.4 (ตัวอย่างที่ 2.4) ทุกประการ

2.7 การออกแบบวงจรนับ 4 โดยใช้ FPGA และมีโมโนสเตเบิลอย่างง่ายเป็นวงจรแก๊บเบาช์

ตัวอย่างที่ 2.7 การออกแบบวงจรนับ 4 แบบซิงโครนัสที่มีวงจรโมโนสเตเบิลในการแก้ปัญหาการเก็บแก๊บเบาช์ (Bouncing) ของปุ่มกดโดยใช้ FPGA จะใช้วงจรคังรูปที่ 2.138 เขียนเดียวกับกรณี CPLD แต่เปลี่ยน L1 เป็น L3 ขั้นตอนการออกแบบจะเหมือนกับการออกแบบโดยใช้ CPLD ทุกประการ โดยเริ่มจากการสร้าง Source File ของ FPGA เพื่อเก็บไฟล์ไว้ทำ Symbols ขึ้นมาก่อน ซึ่งมีขั้นตอนการสร้าง Symbols นี้เหมือนกับ CPLD ในข้อ 2.4 ทุกประการ จากนั้นจึงออกแบบวงจรโดยใช้ FPGA โดยมีขั้นตอนเหมือนกับ CPLD ในข้อ 2.5 ทุกประการ ยกเว้นขั้นตอนความนิ่วลดลง FPGA ซึ่งจะมีขั้นตอนเหมือนกับข้อ 2.2.5 ทุกประการ

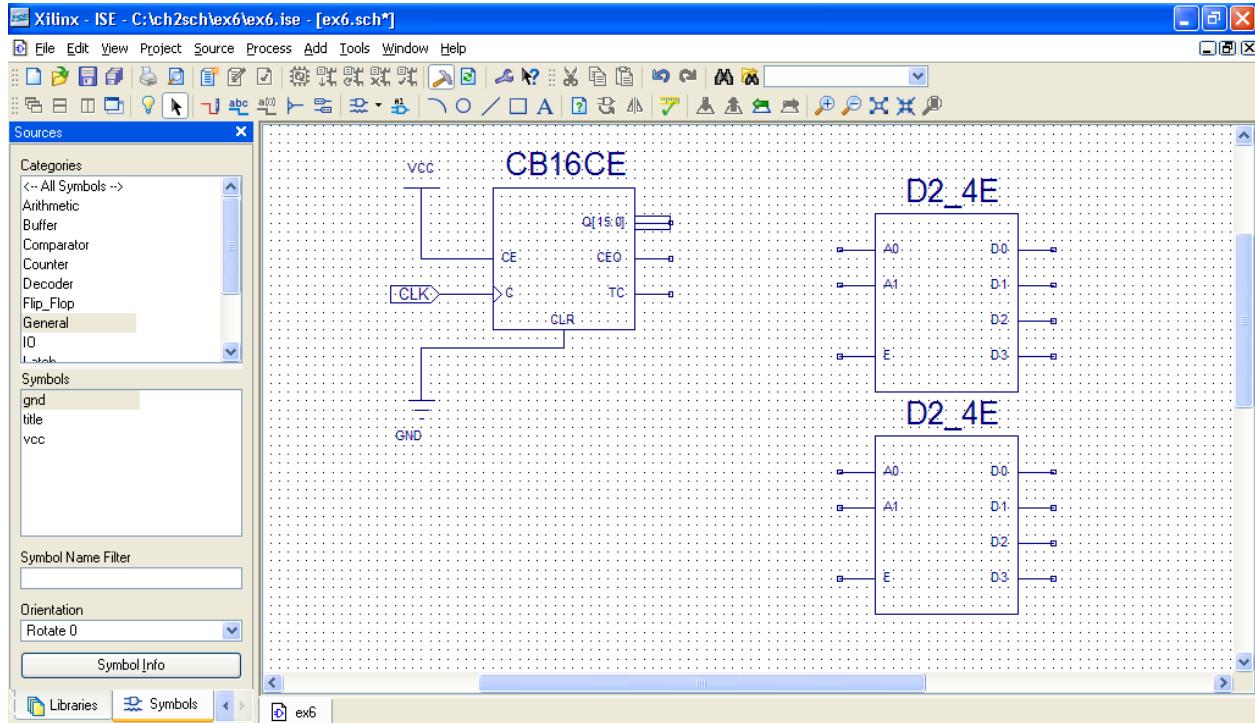
2.8 การใช้บัส

ตัวอย่างที่ 2.8 วงจรทดสอบตัวถอดรหัสคำนับแบบแบนงบัสแสดงดังรูปที่ 2.162 ซึ่งมีขั้นตอนการวางแผนผังวงจรเฉพาะส่วนที่เป็นบัสดังนี้

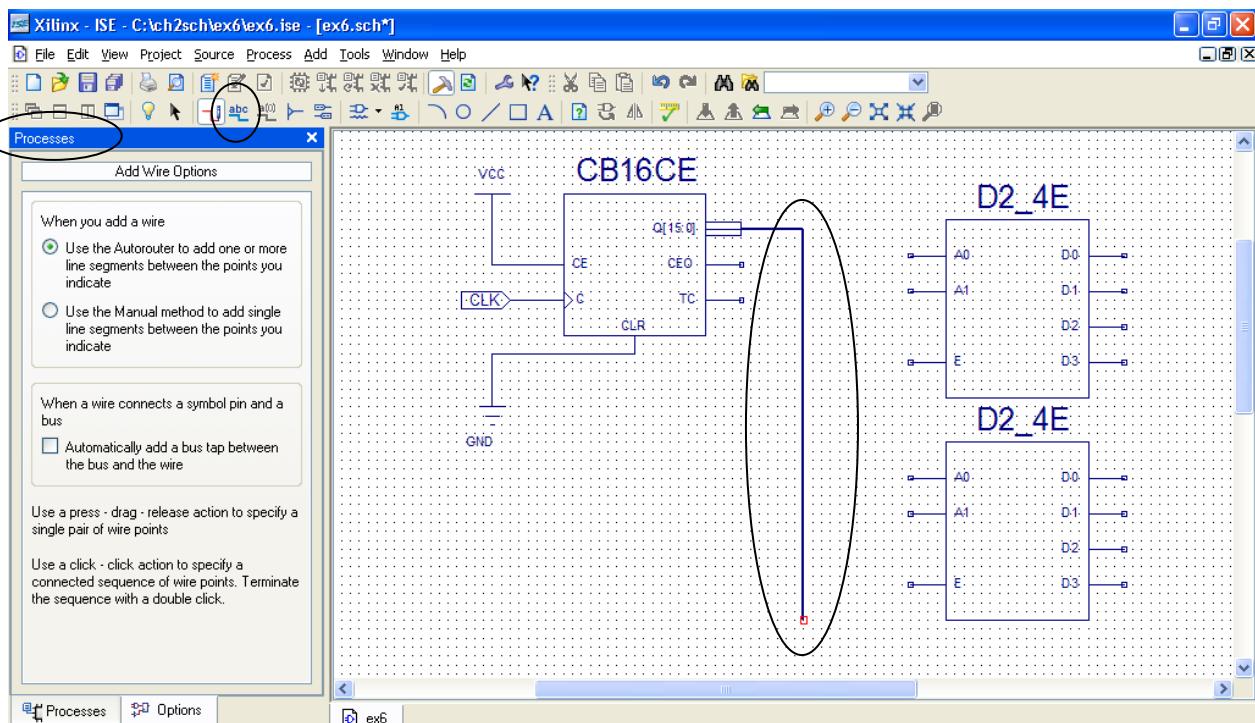


รูปที่ 2.162 วงจรทดสอบตัวถอดรหัสคำนับแบบแบนงบัส

1) สมมุติอุปกรณ์บางส่วนที่วาดไว้ข้างแล้วเป็นดังรูปที่ 2.163 แล้วปิดหน้าต่าง Sources และเปิดหน้าต่าง Processes จากนั้นจึงคลิก แล้วเลื่อนมาสู่ไปลากเส้นดังรูปที่ 2.164

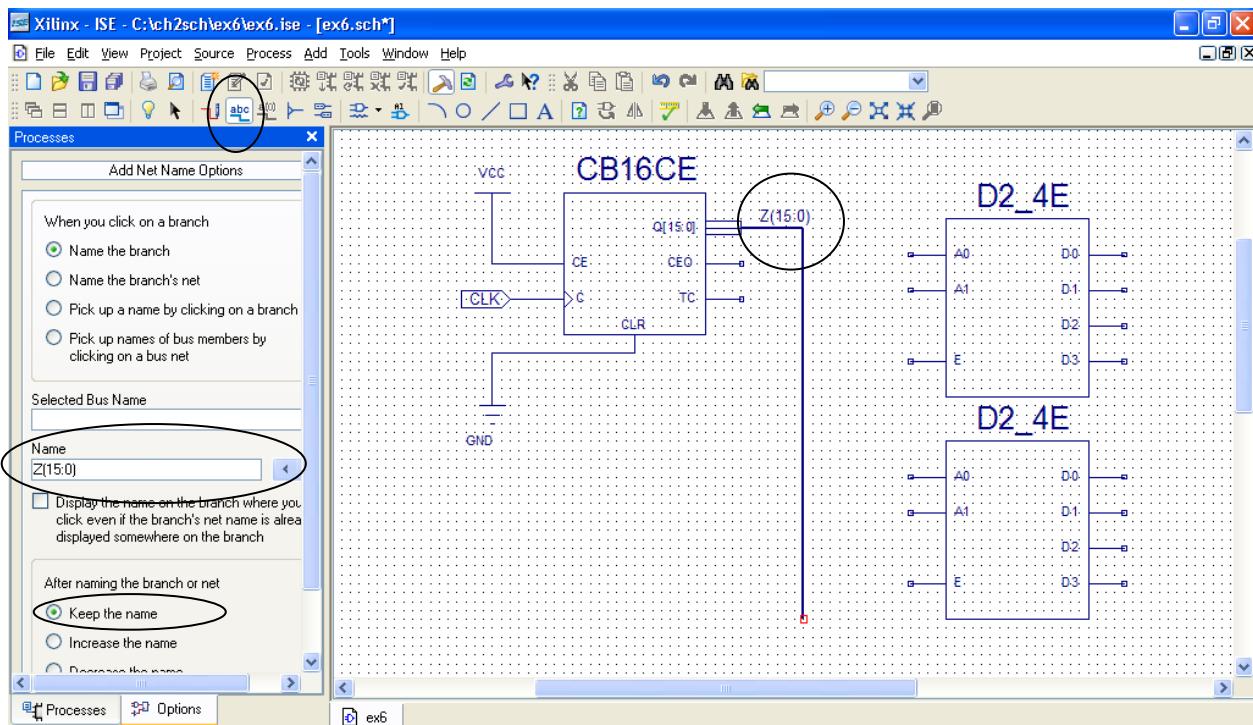


รูปที่ 2.163 ขั้นตอนการวางแผนอุปกรณ์



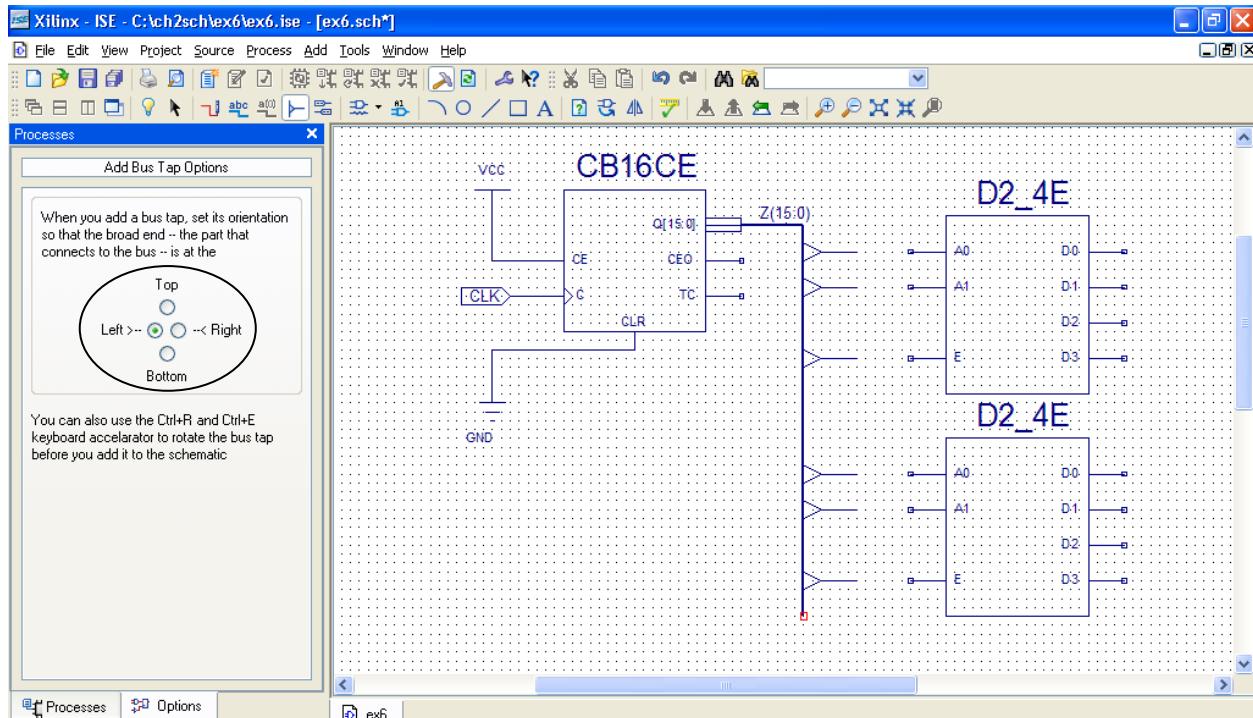
รูปที่ 2.164 ขั้นตอนการวางแผนเส้นบัส

2) คลิก (Add Net Name) เพื่อตั้งชื่อบัส (Bus) พร้อมพิมพ์ชื่อบัส Z(15:0) (คือ Z(15), Z(14), Z(13), ..., Z(0)) ในช่อง Name จากนั้นคลิกในช่องหน้า Keep the Name แล้วเลื่อนมาสู่ไปคลิกวงบนบัสที่ด้านบน (จุดเดียวกัน) แสดงดังรูปที่ 2.165

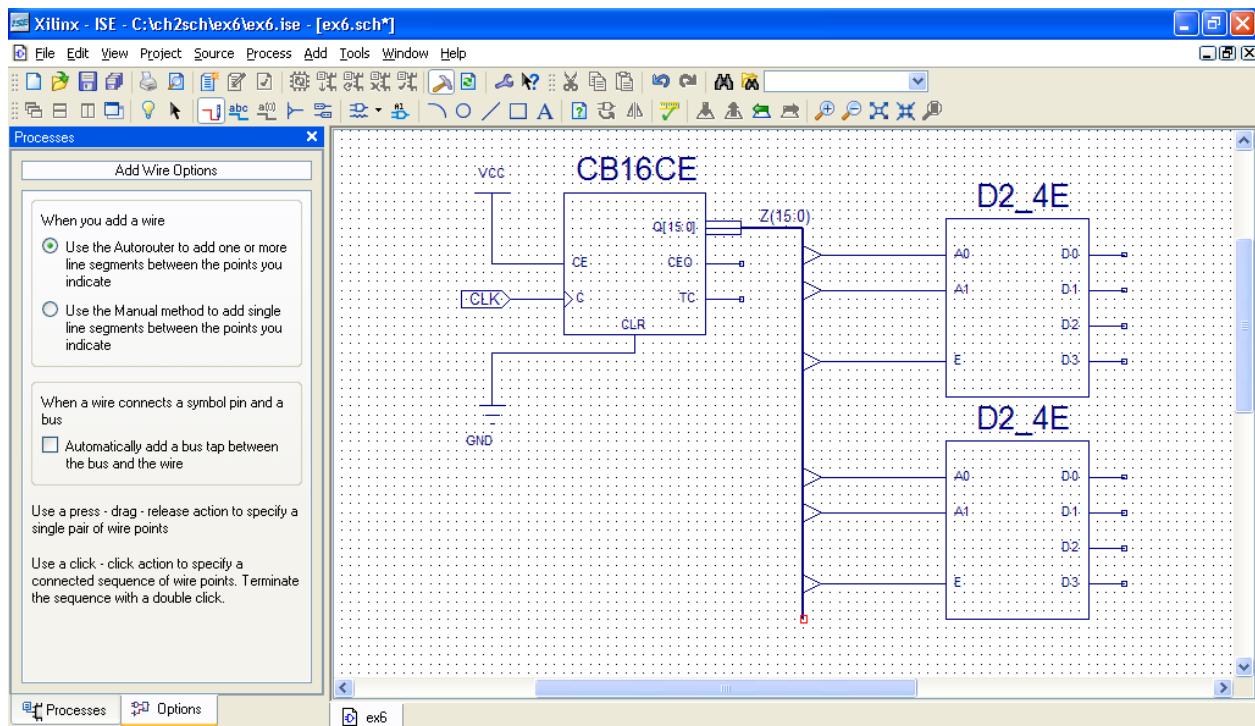


รูปที่ 2.165 การตั้งชื่อบัส

3) คลิก (Add Bus Tap) แล้วเลื่อนมาส์กไปคลิกที่ Left จากนั้นเลื่อนมาส์เพื่อวางบัสแทป (Bus Tap) ทางด้านขวาบนครบ แสดงดังรูปที่ 2.166 จากนั้นลากเส้น (ต้องลากเส้นหลังจากตั้งชื่อบัสเสร็จแล้ว) จนครบแสดงดังรูปที่ 2.167

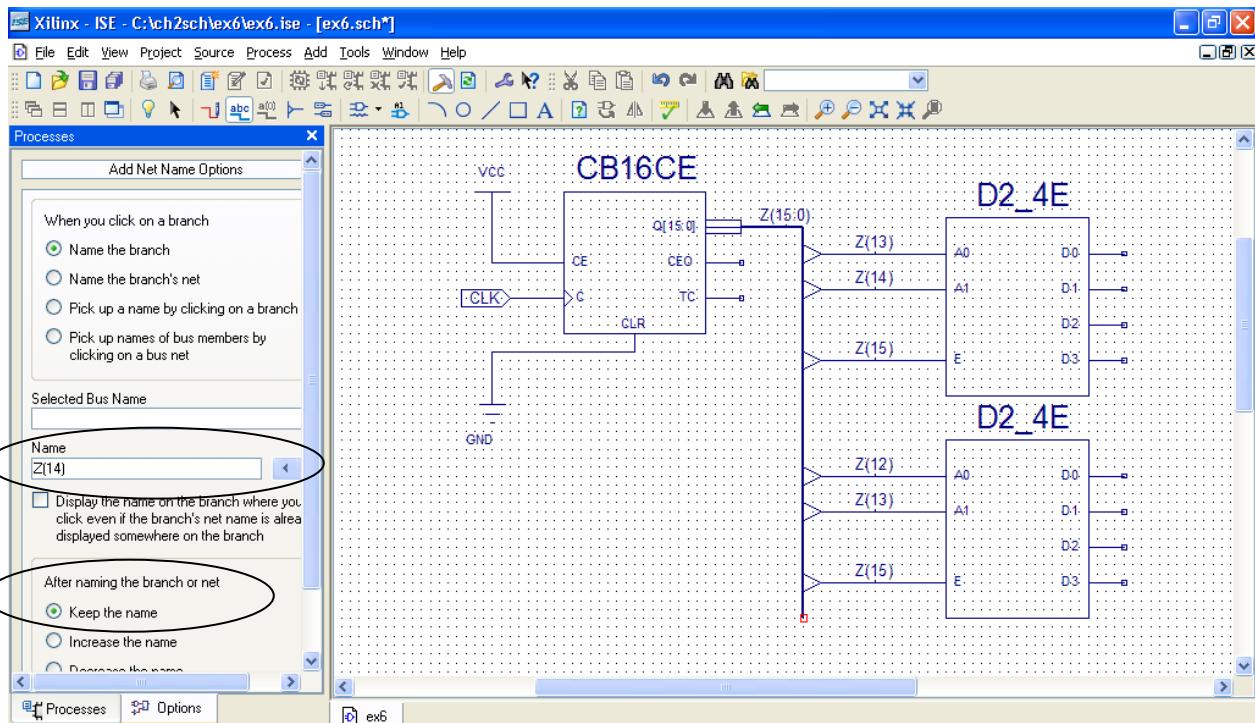


รูปที่ 2.166 การวางบัสแทปทางด้านขวา

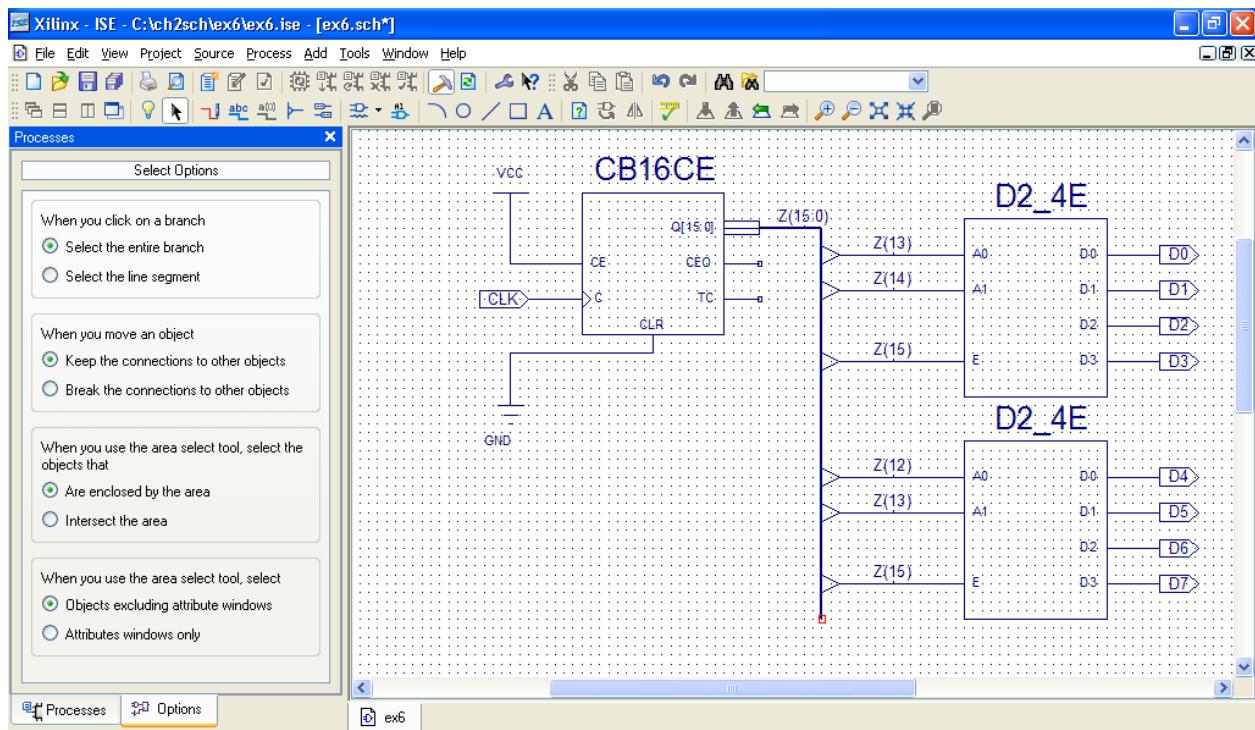


รูปที่ 2.167 การลากเส้น

4) ตั้งชื่อสายสัญญาณ Z(13), Z(14) และ Z(15) ให้กับ D2_4E ตัวที่อยู่ด้านบน เริ่มที่ช่อง Name พิมพ์ Z(13) เลื่อนเมาส์ไปคลิกในช่องหน้า Increase the Name แล้วเลื่อนเมาส์ไปคลิกบนเส้นที่ต่อเข้ากับ A0 , A1 และ E ตามลำดับจนถึงเส้น Z(15) ที่จะต้องไม่ต้องเปลี่ยนชื่อช่อง Name แล้วพิมพ์สเป็นชื่อ Z(12), Z(13) และ Z(15) ที่จะต้องไม่ต้องเปลี่ยนชื่อช่อง Name แล้วเลื่อนเมาส์ไปคลิกที่ช่อง Name ที่ต่อเข้ากับ D2_4E ตัวที่อยู่ด้านล่างตามลำดับจนครบทุกเส้นดังรูปที่ 2.167 จากนั้นใส่ I/O Marker ดังรูปที่ 2.169

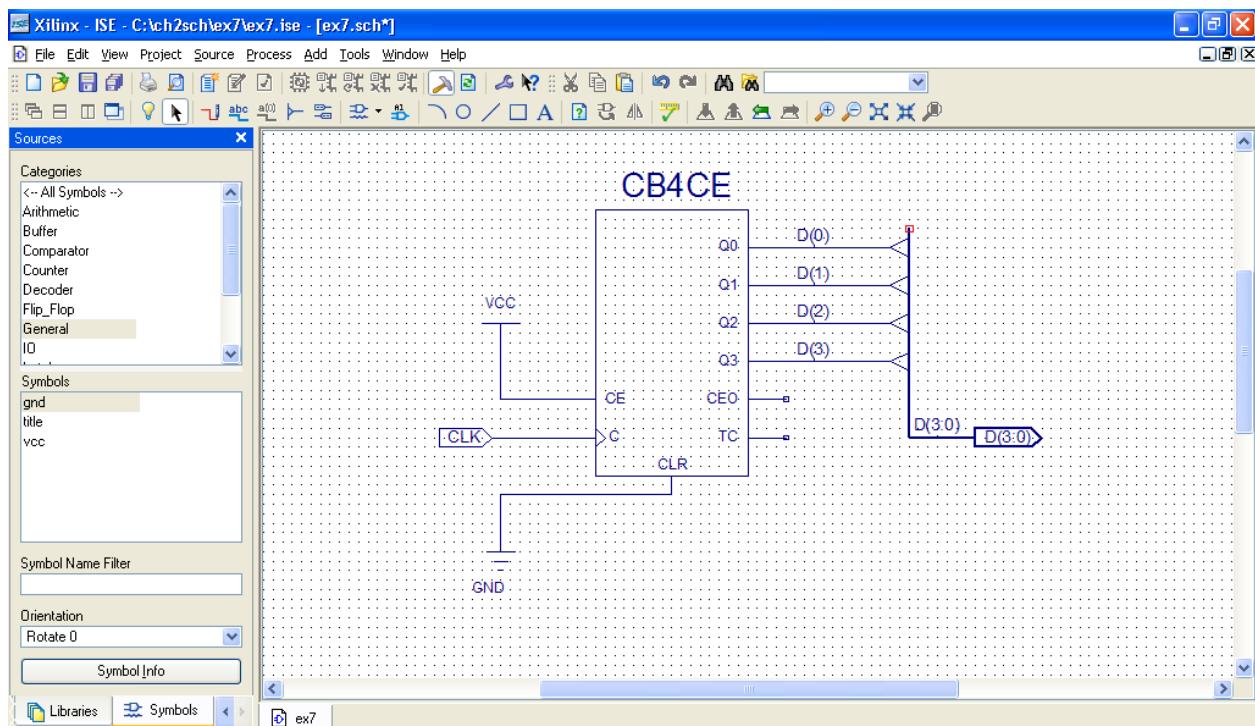


รูปที่ 2.168 การตั้งชื่อสายสัญญาณ

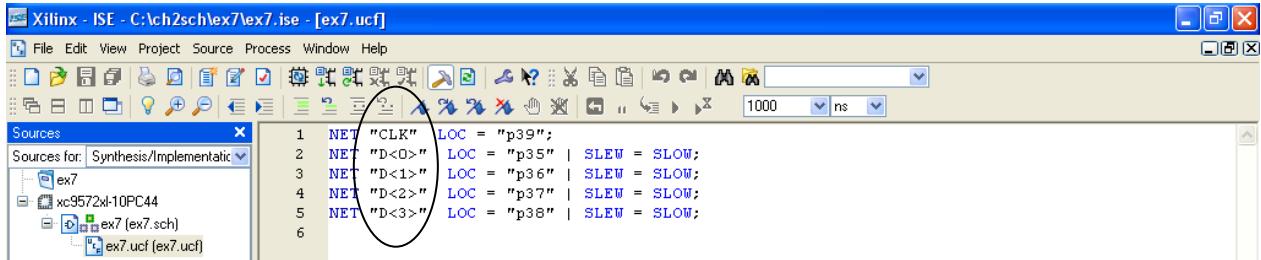


รูปที่ 2.169 วงจรที่ออกแบบเรียบร้อยแล้ว

ตัวอย่างที่ 2.9 การสร้างบัสดังรูปที่ 2.170 มีขั้นตอนในการสร้างบัส คือ หลังจากภาคผังอุปกรณ์เรียบร้อยแล้ว ให้เริ่มจากขั้นตอน ลากเก็บบัส จากนั้นตั้งชื่อบัสชื่อ D(3:0) แล้วจึงใส่บัสแทพ จากนั้นจึงลากเก็บบัสแทพแต่ละอันเข้ากับขา Q0, Q1, Q2 และ Q3 ตามลำดับ เสรีชແล้าตั้งชื่อเก็บ (Wire หรือ Net) D(0), D(1), D(2) และ D(3) แล้วใส่ I/O Marker สำหรับกำหนดไฟล์ใน Edit Constraints (Text) จะเป็นดังรูปที่ 2.171 ซึ่งจะแตกต่างจากที่ผ่านๆ มา ก็ต่อเมื่อจะพิมพ์เป็น D<0>, D<1>, D<2> และ D<3>

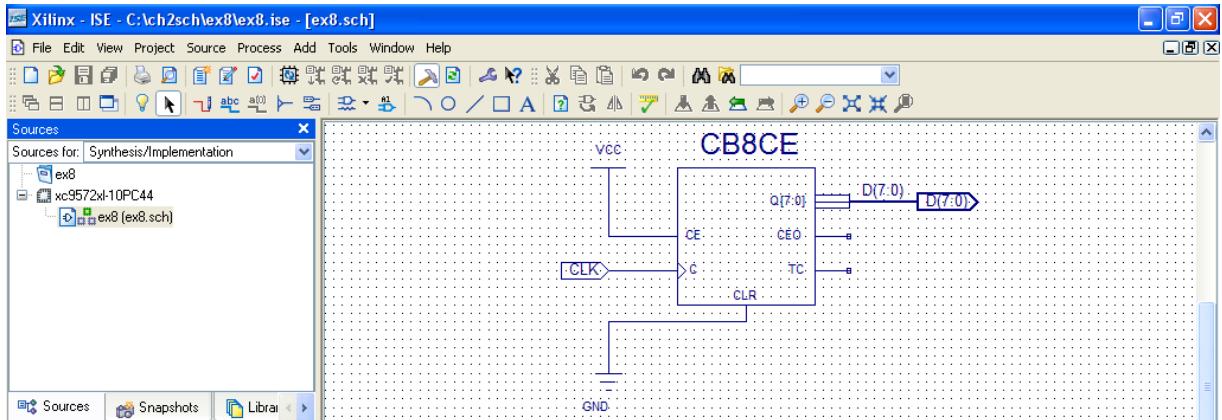


รูปที่ 2.170 ตัวอย่างบัส



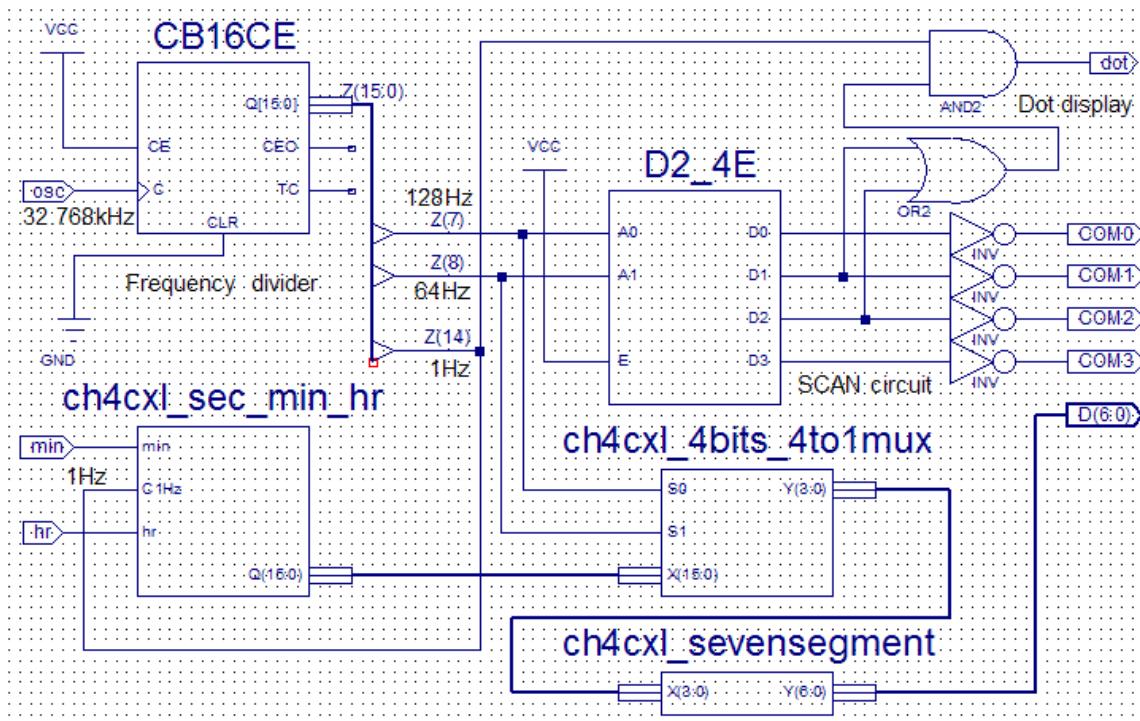
รูปที่ 2.171 การกำหนดไฟล์ใน Edit Constraints (Text)

ตัวอย่างที่ 2.10 ในกรณี I/O Marker ต่อเข้าบัสโดยตรงดังรูปที่ 2.172 คือ D(7:0) นั้นไม่ต้องใช้บัสແທพ การกำหนดไฟล์ใน Edit Constraints (Text) จะเป็นลักษณะเดียวกันกับตัวอย่างที่ 2.171



รูปที่ 2.172 ตัวอย่างบัส

ตัวอย่างที่ 2.11 ผังวงจรนาฬิกาดิจิตอล การใช้บัสและบัสแท็ปрутแบบต่างๆ ดังรูปที่ 2.173 ส่วนเครื่องมือ  / □ A ที่อยู่บน Tool bar จะใช้ในการวิเคราะห์หรือเขียนตัวอักษรต่างๆ ที่ใช้ช่วยอธิบาย ซึ่งไม่เกี่ยวข้องกับการออกแบบวงจรแต่อย่างใด



รูปที่ 2.173 ผังวงจรนาฬิกาดิจิตอล

สรุป

ขั้นตอนการออกแบบ ไอซีดิจิตอลด้วย FPGA และ CPLD โดยวิธีการ Schematic โดยทั่วไปมีขั้นตอนหลักๆ ดังนี้

1. การออกแบบวงจร(Design Entry) เป็นการแปลงวงจรดิจิตอลที่ต้องการออกแบบให้อยู่ในรูปแบบที่ซอฟต์แวร์ทูล ISE WebPack 8.1i เข้าใจ ซึ่งในที่นี้จะใช้วิธีการวาดผังวงจร(Schematic)
2. การสังเคราะห์วงจร(Design synthesis) เป็นการสั่งให้ซอฟต์แวร์ทูลสร้างวงจรตามที่เราออกแบบไว้
3. Design Implementation เป็นการสร้างวงจรที่ต้องการโดยกำหนดรายละเอียดขาต่างๆ ที่ต่อ กับ ขาของ FPGA และ CPLD ตามที่ผู้ใช้กำหนดไว้ใน User Constrain File
4. การโปรแกรมชิป เป็นการสร้างไฟล์ที่เก็บข้อมูลวงจรในรูปแบบที่พร้อมจะส่งไปลง FPGA และ CPLD โดยการดาวน์โหลดผ่านสาย JTAG

โดยขั้นตอนต่างๆ ของทั้ง FPGA และ CPLD มีขั้นตอนคล้ายคลึงกัน ยกเว้นขั้นตอนการโปรแกรมชิปของ FPGA ผู้ใช้สามารถสร้างไฟล์ PROM เพื่อใช้ดาวน์โหลด PROM แทนที่จะดาวน์โหลด FPGA โดยตรง ซึ่งมีข้อดีกว่า คือ ไม่ต้องดาวน์โหลดใหม่ทุกๆ ครั้งที่จำเป็นไฟฟ้าให้ FPGA เมื่อจาก FPGA ไม่สามารถเก็บข้อมูลวงจรที่ดาวน์โหลดลงไปแล้ว ได้เหมือน CPLD หากมีการตัดไฟ จึงจำเป็นต้องอาศัย PROM เก็บข้อมูลให้แทน

คำถามท้ายบท

1. การตรวจสอบความถูกต้องของ Syntax ของวงจรที่เราได้ออกแบบโดยวิธีการ Schematic สามารถทำได้อย่างไร
2. ไฟล์ XST ใน Xilinx ISE WebPack 8.1i คืออะไร
3. ไฟล์ UCF มีความสำคัญอย่างไรในการออกแบบ ไอซีดิจิตอลด้วย FPGA และ CPLD
4. การต่อตัวต้านทาน 10KOhm ระหว่างขา VCC กับ PB1 และ PB2 มีไว้เพื่อเหตุผลใด และสามารถใช้ค่าอื่นๆ ได้หรือไม่
5. สาย JTAG คืออะไร มีไว้ทำไม
6. หากต้องการ ดาวน์โหลดข้อมูลจาก PROM ลง FPGA โดยอัตโนมัติจะต้องตั้งโหมดการทำงานเป็นโหมดใด และต้องตั้งจ้มเปอร์อย่างไรบ้าง
7. การจำลองการทำงานเชิงพฤติกรรมเป็นอย่างไร
8. การตรวจสอบความถูกต้องของวงจร(Design Verification) มีขั้นตอนอย่างไรบ้าง

== หน้าว่าง ==

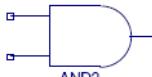
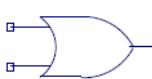
บทที่ 3 ล้อจิกเกตและวงจรคอมบิเนชัน

ຄລ່າວນໍາ

การออกแบบวงจรดิจิตอลโดยทั่วไป มักจะเป็นการออกแบบโดยใช้งานกรเกตพื้นฐานมาเชื่อมต่อกันให้ถูกต้องเป็นวงจรคอมบินेशัน ซึ่งโดยทั่วไป จะเป็นองค์ประกอบหลักๆ ในวงจรดิจิตอลทั่วไปเกือบทุกวงจร ดังนั้นหากผู้ใช้งานสามารถสร้างวงจรลอจิกกรเกตพื้นฐานได้เองจะทำให้การสร้างวงจรดิจิตอลทำได้ง่ายขึ้น โดยไม่ต้องต่อโดยใช้อิชีพีที่มีแล็ปอีกต่อไป และเมื่อสามารถสร้างวงจรดิจิติกกรเกตพื้นฐานได้แล้ว ก็สามารถนำมาระบบประยุกต์เป็นวงจรคอมบินेशัน เช่นวงจรบวก วงจรตอครหัส วงจรเข้ารหัส และวงจรอื่นๆ ได้ ดังจะได้แสดงตัวอย่างวิธีการสร้างตกลดค่าจันไปถึงวิธีการดาวน์โหลดลงชิป FPGA และ CPLD เพื่อทดสอบการทำงานจริงๆ บนบอร์ดทดลองต่อไป

3.1 ລອຈິກເກຕ

ลอจิกเกตต่างๆ และสมการบูลีน (Logic gates and boolean expression) สรุปดังรูปที่ 3.1 สำหรับคนที่มีความรู้พื้นฐานด้านคณิตศาสตร์แล้วก็จะเข้าใจตารางลอจิกเกตต่างๆ ได้เป็นอย่างดี

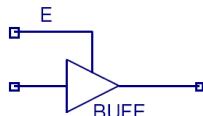
LOGIC	SYMBOL	BOOLEAN EXPRESSION	INPUT		OUTPUT
			A	B	Y
AND		$A \cdot B = Y$	0	0	0
			0	1	0
			1	0	0
			1	1	1
OR		$A + B = Y$	0	0	0
			0	1	1
			1	0	1
			1	1	1
NAND		$\overline{A \cdot B} = Y$	0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$\overline{A + B} = Y$	0	0	1
			0	1	0
			1	0	0
			1	1	0
XOR		$A \oplus B = Y$	0	0	0
			0	1	1
			1	0	1
			1	1	0
XNOR		$\overline{A \oplus B} = Y$	0	0	1
			0	1	0
			1	0	0
			1	1	1
INVERTER		$\overline{A} = Y$	0		1
			1		0

รูปที่ 3.1 ลักษณะต่างๆ

นอกจากคลอจิกเกตต่างๆแล้วยังมีบีฟเฟอร์ (Buffer) และไทร-สเตตบีฟเฟอร์ (Tri-state buffer) อีกด้วยแสดงดังรูปที่ 3.2 และรูปที่ 3.3 ตามลำดับ อินพุตและเอาต์พุตของบีฟเฟอร์จะมีคลอจิกเหมือนกัน ยกเว้นไทร-สเตตบีฟเฟอร์ดังตัวอย่างในรูปที่ 3.3 ถ้า $E = '0'$ เอาต์พุตจะเป็น $'Z'$ หรือ High impedance แต่ถ้า $E = '1'$ อินพุตและเอาต์พุตจะมีคลอจิกเหมือนกัน

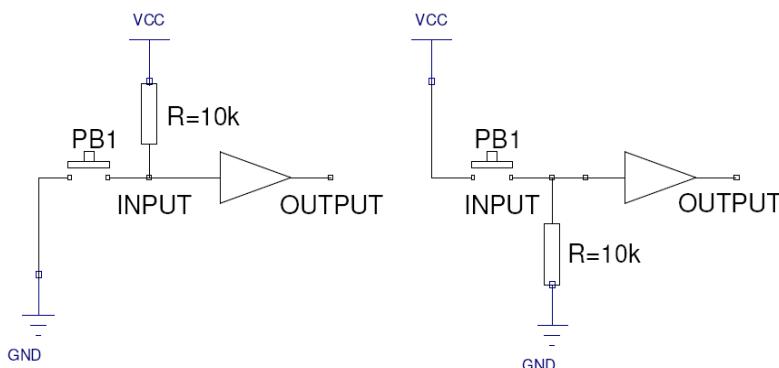


รูปที่ 3.2 บัฟเฟอร์ (Buffer)



รูปที่ 3.3 ไตร-สเตตบัฟเฟอร์ (Tri-state buffer) แบบ Active high (E = '1')

อินพุตของไอซีล็อกิกเกตไม่ว่าเป็นตรรกะ TTL หรือ CMOS หรือแม้กระทั่ง FPGA และ CPLD ที่รับจากสวิตช์คีย์บอร์ด หรือหน้าสัมผัสของรีเลย์ ในทางปฏิบัตินั้นจะใช้ตัวด้านท่านpull up (Pull up) ต่อเข้ากับแหล่งจ่าย (Vcc) หรือตัวด้านท่านpull down (Pull down) ต่อเข้ากับกราวด์ (GND) เพื่อบังคับให้อินพุตมี值อิกเป็น '1' หรือ '0' ในขณะที่ยังไม่กดสวิตช์แสดงดังรูปที่ 3.4 การปล่อยอินพุตอยู่ไว้ (Floating) อาจให้ค่าล็อกิกไม่แน่นอน เนื่องจากลัญญาณรบกวนต่างๆ อาจทำให้อินพุตเปลี่ยนสถานะได้ (ยกเว้นไอซีตรรกะ TTL อาจถือว่าการปล่อยอินพุตอยู่ไว้เป็นล็อกิก '1' แต่อาจถูกรบกวนได้) อินพุตที่ไม่ได้ใช้งานควรต่อเข้ากับ Vcc หรือ GND โดยตรงเพื่อบังคับให้อินพุตนั้นมี值อิกเป็น '1' หรือ '0' ยกเว้นขาอินพุตเอาต์พุตของ FPGA หรือ CPLD ที่ไม่ได้กำหนดให้ใช้งานนั้นสามารถปล่อยขาล้อได้ ในทางปฏิบัติตัวด้านท่านpull upหรือpull downที่ใช้มีค่าประมาณ 2,200 ถึง 10,000 โอห์ม ขึ้นอยู่กับกระแสไฟล์ที่อินพุต ผู้อ่านควรศึกษารายละเอียดจาก Data sheet ของผู้ผลิต



(a) ตัวด้านท่านpull up

(b) ตัวด้านท่านpull down

รูปที่ 3.4 วงจรล็อกิกเกตต่างๆ ที่ต้องขาอินพุตเข้ากับตัวด้านท่านpull upหรือpull down

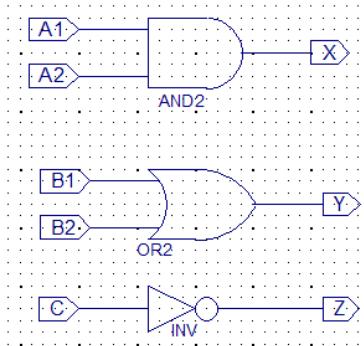
จากรูปที่ 3.4(a) ถ้าไม่กดปุ่มกดอินพุตจะเป็นล็อกิก '1' แต่ถ้ากดปุ่มกดจะเป็นล็อกิก '0' (เรียกว่า Active Low) ส่วนในรูปที่ 3.4(b) ถ้าไม่กดปุ่มกดอินพุตจะเป็นล็อกิก '0' แต่ถ้ากดปุ่มกดจะเป็นล็อกิก '1' (เรียกว่า Active high) บอร์ดทดลองทุกรุ่นที่ใช้งานอยู่ในหนังสือเล่มนี้จะใช้สวิตช์ที่เป็นปุ่มกดหรือดิพสวิตช์แบบ Active low ทั้งหมด

3.1.1 การออกแบบวงจรลอจิกเกต

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างແອນດັກຕ ອອຮ່ເກຕ ແລະ ອິນເວອຣ່ເທອຣ່ດ້ວຍ CPLD

สร้างวงจรແອນດັກຕ ອອຮ່ເກຕ ແລະ ອິນເວອຣ່ເທອຣ່ດ້ວຍ CPLD ດັງຮູບທີ 3.5 ຕາມບັນດອນໃນຫຼຸດ 2.1 (ຕົວຢ່າງທີ 2.1) ແລະ ຫຼຸດ 2.3 (ຕົວຢ່າງທີ 2.3) ໃນບົທີ 2

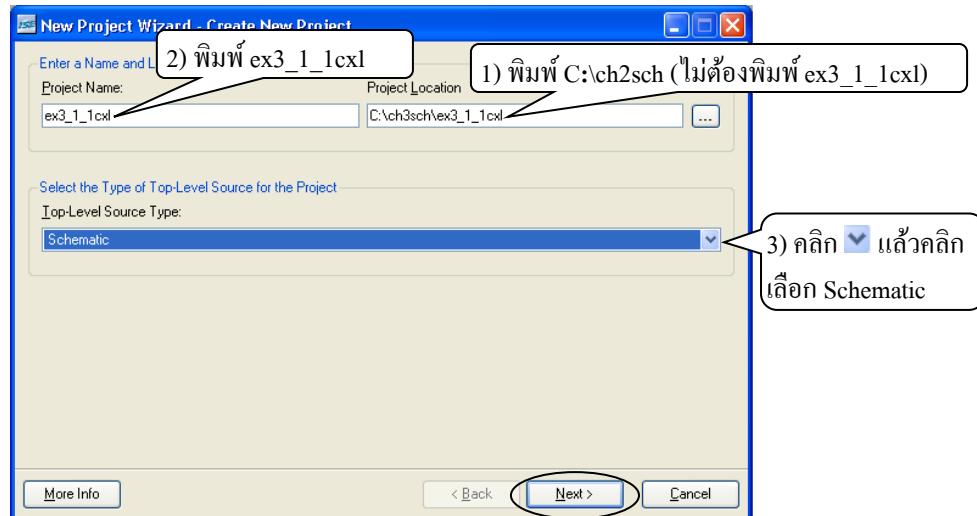


ຮູບທີ 3.5 ວິຈາຮລອຈິກເກຕຕ່າງໆ

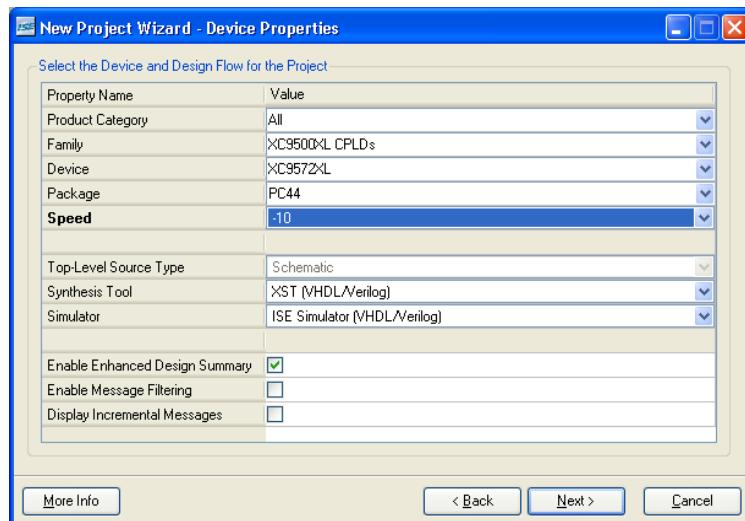
1.1 ບັນດອນກາຮອກແບບວິຈາຮ (Design Entry)



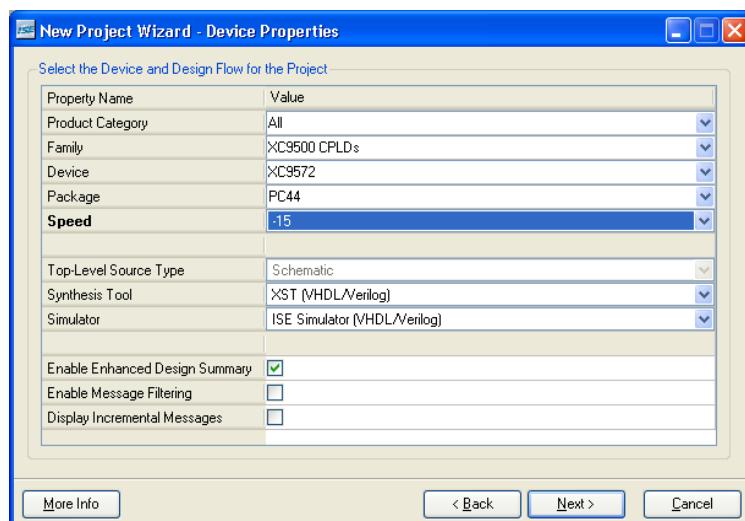
- 1) ໃຫ້ສ້າງ Folder ຊື່ ch3sch ໄວໃນໄໂຣຟີ C ດັບເປີລົຄລິກ xilinx ISE 8.1i ແລ້ວຈະໄດ້ໜ້າຕ່າງ Xilinx–ISE (ຄໍາເນີ້ນໜ້າຕ່າງ Tip of the Day ຊ່ອນບິ່ນມາໃຫ້ຄົກລິກ OK) ຈາກນັ້ນຄົກລິກ File -> New Project ແລ້ວຈະໄດ້ໜ້າຕ່າງ (ຫຼື Dialog box) New Project Wizard
- 2) ສ້າງໂປຣເກຕຕີໄຟລ໌ທີ Project Location (ຫຼື Folder) ຊື່ ch3sch ແລ້ວກໍາທັນດ Project Name ຊື່ ex3_1_1cxl ຕາມລຳດັບ ຄົກທີ Top-Level Source Type ເປັນ Schematic ແສດງດັງຮູບທີ 3.6 ຄົກລິກ Next ແລ້ວຈະໄດ້ໜ້າຕ່າງຄັດໄປ
- 3) ທີ່ໜ້າຕ່າງ New Project Wizard ຄົກເລືອກຫີພທີ່ຕ້ອງກາຮັດຮູບທີ 3.7 ໂດຍໃໝ່ CPLD ຕະກູລ XC9500XL ເບອ້ XC9572XL ຊື່ນີ້ມີ Package ແບບ PLCC 44 ຂາ (Package : PC44) ແລະ Speed Grade :-10 ທີ່ໃຊ້ບັນບອົດ CPLD Explorer XC9572XL (ກຣົມທີ່ໃໝ່ບອົດ CPLD Explorer XC9572 ໃຫ້ຄົກເລືອກ CPLD ຕະກູລ XC9500 ເບອ້ XC9572 Package : PC44 ແລະ Speed Grade :-15)



รูปที่ 3.6 หน้าต่าง New Project Wizard



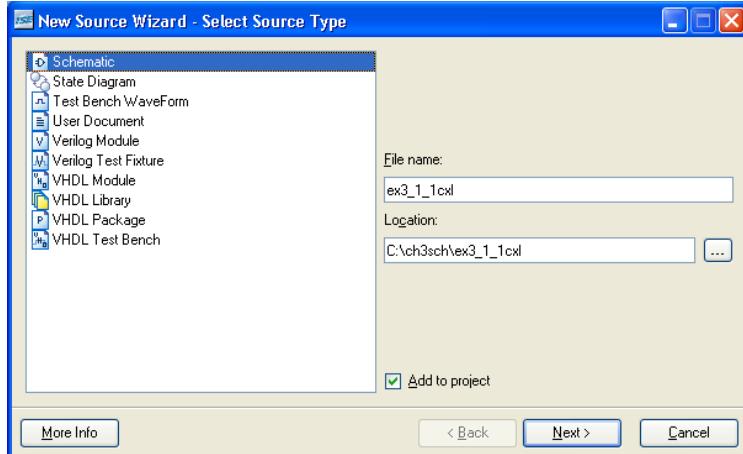
(a) CPLD ที่ใช้กับบอร์ด CPLD Explorer XC9572XL



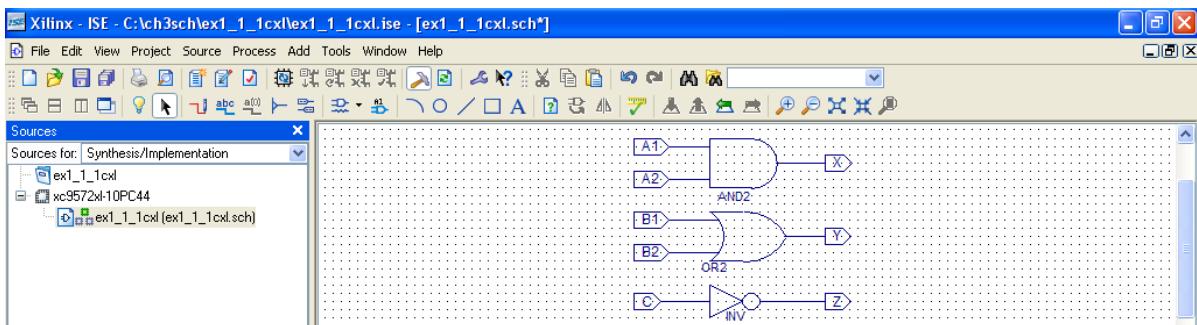
(b) CPLD ที่ใช้กับบอร์ด CPLD Explorer XC9572

รูปที่ 3.7 หน้าต่าง New Project Wizard–Device Properties

- 4) คลิก Next ในรูปที่ 3.7 แล้วจะได้หน้าต่างอัดไป คลิกปุ่ม New Source แล้วจะได้หน้าต่างอัดไป จากนั้นพิมพ์ชื่อ Source File ชื่อ ex3_1.cxl ลงในช่องของ File Name และคลิกที่ Schematic ดังรูปที่ 3.8 คลิก Next คลิก Finish คลิก Next อีก 2 ครั้งและคลิก Finish แล้วจะได้หน้าต่าง Xilinx-ISE คลิกไอคอน แล้วรับทราบตามรูปที่ 3.5 จนแล้วเสร็จดังรูปที่ 3.9
 5) การตรวจสอบความถูกต้อง ถ้าไม่มีข้อผิดพลาดก็ให้บันทึกไฟล์ คลิก **X** (สีดำ) ปิดโปรแกรมเพื่อกลับไปที่หน้าต่าง Xilinx-ISE



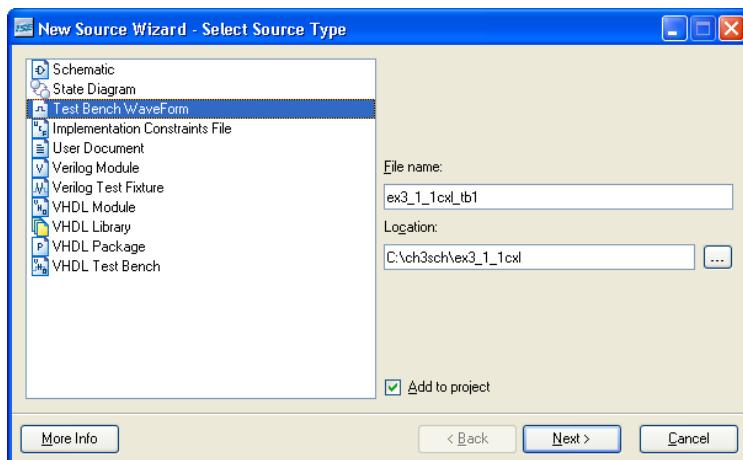
รูปที่ 3.8 หน้าต่าง New Project Wizard-Select Source Type



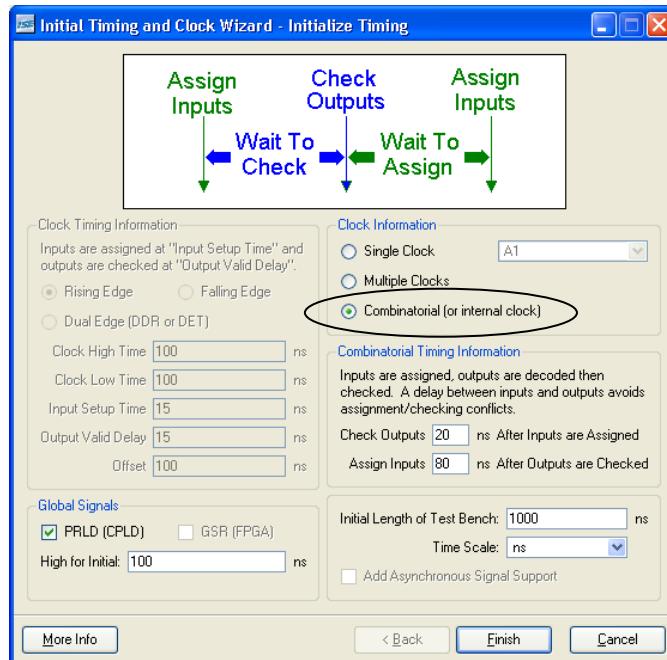
รูปที่ 3.9 หน้าต่าง Xilinx-ISE สำหรับการผังวงจร

1.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification)

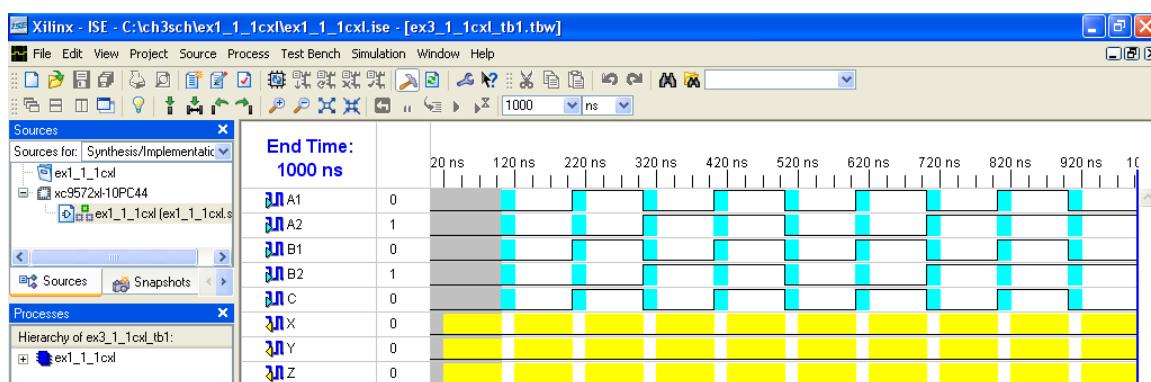
ทำการขึ้นตอนในข้อ 2.3 (ตัวอย่างที่ 2.3) ในบทที่ 2 พิมพ์ชื่อ ex3_1_cxl_tb1 ดังรูปที่ 3.10 จากนั้นกำหนดค่าเวลา และ Waveform ดังรูปที่ 3.11 และรูปที่ 3.12 เตรียมแล้วให้พิจารณาว่าผล Behavioral simulation เป็นไปตามทฤษฎีหรือไม่



รูปที่ 3.10 หน้าต่าง New Source Wizard



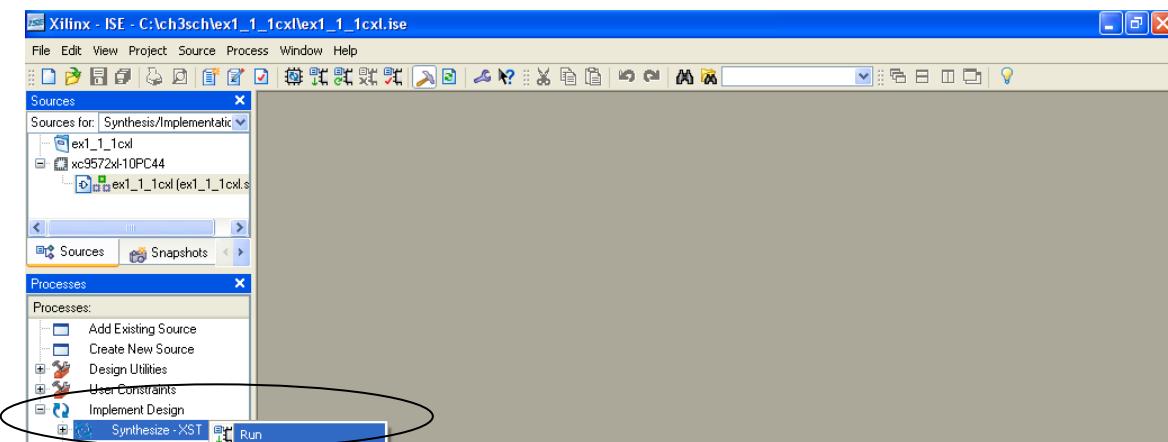
รูปที่ 3.11 หน้าต่าง Initial Timing and Clock Wizard



รูปที่ 3.12 หน้าต่างสำหรับกำหนดสัญญาณต่างๆที่ป้อนให้กับอินพุตของวงจรที่เราต้องการทดสอบ

1.3 การสังเคราะห์วงจร (Design synthesis)

ที่หน้าต่าง Processes คลิก “+” ที่อยู่หน้า Implement Design จนกลายเป็น “-” จากนั้นคลิกขวา Synthesize-XST แล้วคลิก Run ดังรูปที่ 3.13 เสร็จแล้วถ้าใช้ ✓ หรือ ⚡ ที่หน้า Synthesize-XST คือว่าขั้นตอนสังเคราะห์วงจรผ่าน



รูปที่ 3.13 ขั้นตอนการสังเคราะห์วงจร

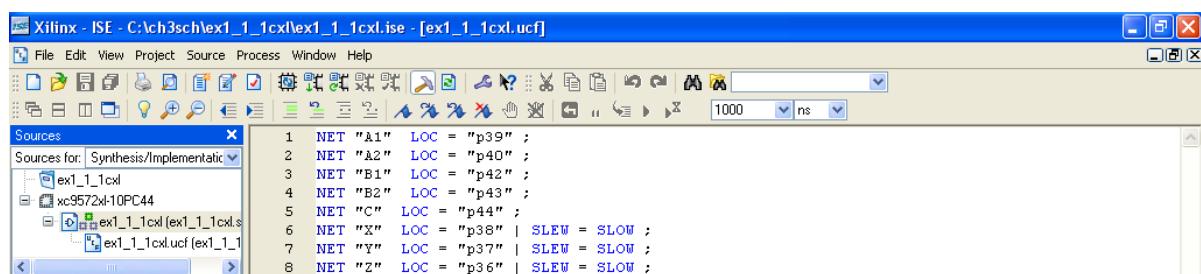
1.4 Design Implementation

- 1) สร้าง Implementation constraints file (UCF) โดยอัตโนมัติ
- 2) ขั้นตอนกำหนดสายสัญญาณต่างๆ ของวงจรในรูปที่ 3.5 เช้ากับขาของ CPLD จะเป็นต้องกำหนดให้อินพุตและเอาต์พุตของ CPLD صدقกับอุปกรณ์ที่เตรียมไว้บนบอร์ดทดลองดังมีรายละเอียดตามตารางที่ 1.3 ในบทที่ 1 ซึ่งจะใช้ปุ่มกด PB1 ถึง PB5 เป็นอินพุต และ LED1 ถึง LED3 เป็นเอาต์พุต กล่าวคือ

$A_1 = PB1 = INPUT = p39$	$X = LED1 = OUTPUT = p38$
$A_2 = PB2 = INPUT = p40$	$Y = LED2 = OUTPUT = p37$
$B_1 = PB3 = INPUT = p42$	$Z = LED3 = OUTPUT = P36$
$B_2 = PB4 = INPUT = p43$	$C = PB5 = INPUT = p44$

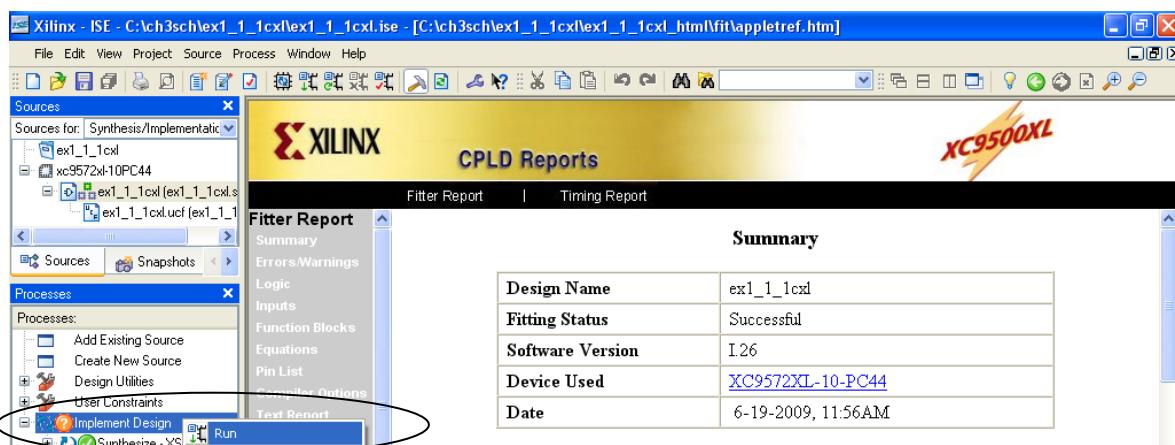
ให้พิมพ์รายละเอียดต่างๆ ใน Edit Constraints (Text) เมื่อพิมพ์เสร็จแล้วจะได้ดังรูปที่ 3.14 จากนั้นบันทึกไฟล์โดยคลิก

แล้วคลิก **X** (สีดำ) เพื่อปิดโปรแกรม Edit Constraints (Text) เพื่อกลับไปที่หน้าต่าง Xilinx-ISE อีกรัง



รูปที่ 3.14 การกำหนดขาในหน้าต่าง Edit Constraints (Text)

- 3) ขั้นตอน Implementation คลิกที่ ex3_1_cxl ในหน้าต่าง Source และคลิกขวาที่ Implement Design ในหน้าต่าง Processes แล้วคลิก Run ดังรูปที่ 3.15 เสร็จแล้วถ้าโชว์ **✓** หรือ **!** ที่หน้า Implement Design ที่ว่าขั้นตอน Implement Design ผ่าน



รูปที่ 3.15 ขั้นตอน Implementation

1.5 การโปรแกรมข้อมูลวงจรลงชิป

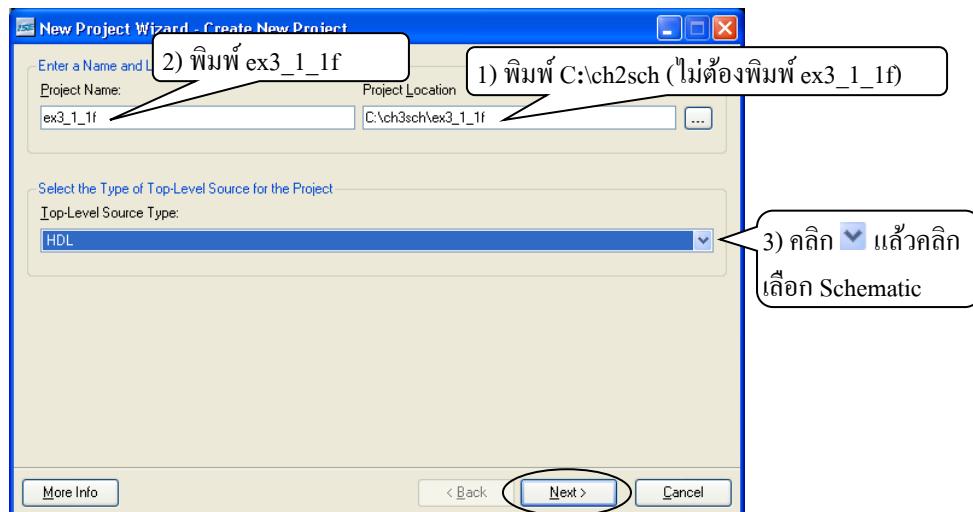
ขั้นตอนดาวน์โหลดจะเหมือนกับขั้นตอนที่ 2.1 ในบทที่ 2 โดยดาวน์โหลดไฟล์ชื่อ ex3_1_cxl.jed ลง CPLD แล้วให้กดปุ่ม PB1-PB5 และคุณปั๊บ LED1-LED3 ว่าให้อเอาต์พุตเป็นไปตามที่ต้องการหรือไม่ แล้วบันทึกผลการทดลอง และต้องพึงระลึกอยู่เสมอว่า Push button switch ทำงานเป็นแบบ Active low กล่าวคือ ถ้ากดปุ่มจะเป็นโลจิก '0' แต่ถ้าไม่กดจะเป็นโลจิก '1'

2. สร้างแอนด์เกต ออร์เกต และอินเวอร์เตอร์ด้วย FPGA

สร้างวงจรแอนด์เกต ออร์เกต และอินเวอร์เตอร์ด้วย FPGA ดังรูปที่ 3.5 ตามขั้นตอนในข้อ 2.1 (ตัวอย่างที่ 2.1) และข้อ 2.3 (ตัวอย่างที่ 2.3) ในบทที่ 2

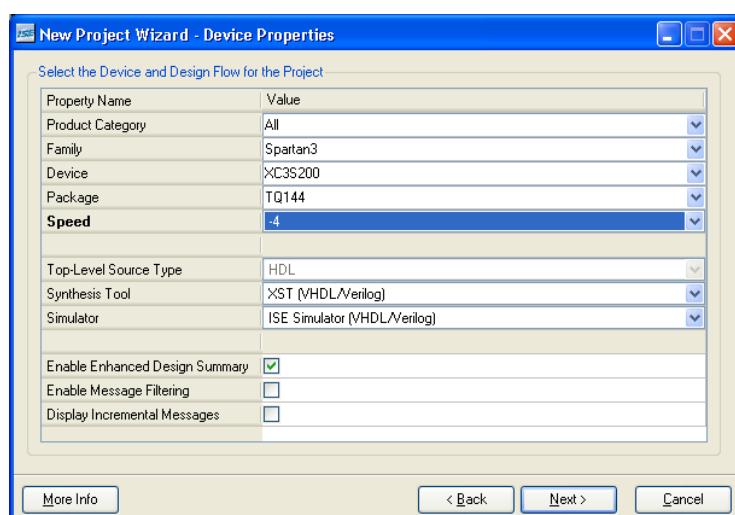
2.1 ขั้นตอนการออกแบบวงจร (Design Entry)

- 1) สร้าง Folder ชื่อ ch3sch ไว้ในไดรฟ์ C (ถ้าไม่มี) ดับเบิลคลิก Xilinx ISE 8.1i แล้วจะได้หน้าต่าง Xilinx-ISE (ถ้ามีหน้าต่าง Tip of the Day ข้อนี้มาให้คลิก OK) จากนั้นคลิก File -> New Project และจะได้หน้าต่าง (หรือ Dialog box) New Project Wizard
- 2) สร้างโปรเจกต์ไฟล์ที่ Project Location (หรือ Folder) ชื่อ ch3sch และกำหนด Project Name ชื่อ ex3_1_1f ตามลำดับ และคลิกที่ Top-Level Source Type เป็น Schematic และดังรูปที่ 3.16 คลิก Next และจะได้หน้าต่างตัดไป

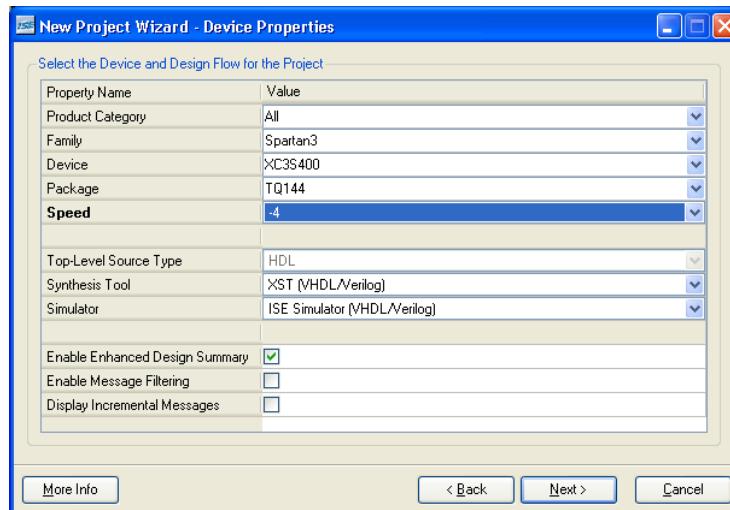


รูปที่ 3.16 หน้าต่าง New Project Wizard

- 3) ที่หน้าต่าง New Project Wizard–Device Properties เลือกชิปดังรูปที่ 3.17



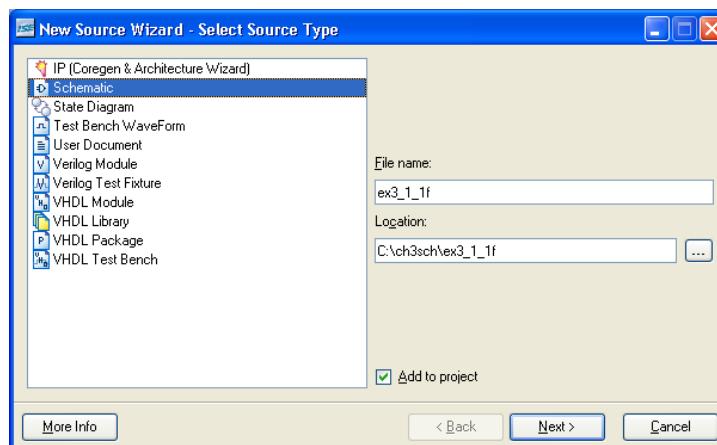
(a) FPGA ที่ใช้กับบอร์ด FPGA Discovery-III XC3S200F (รุ่น 200,000 เกต)



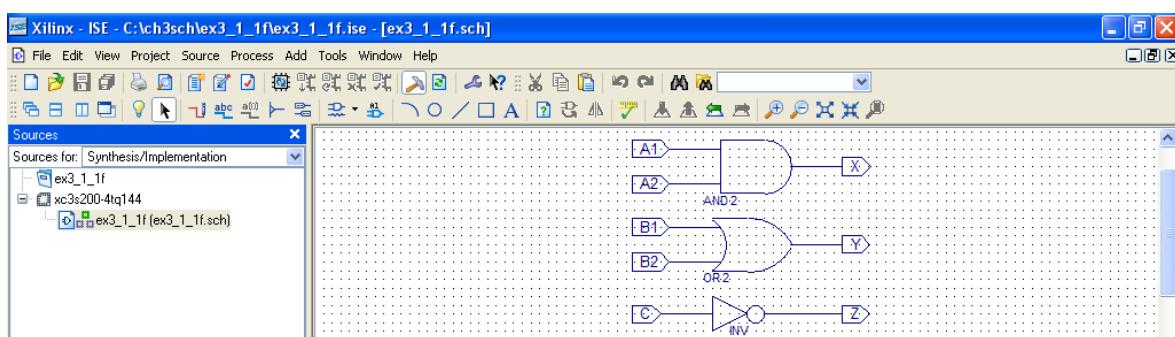
(b) FPGA ที่ใช้กับบอร์ด FPGA Discovery-III XC3S200F4 (รุ่น 400,000 เกต)

รูปที่ 3.17 หน้าต่าง New Project Wizard-Device Properties

- 4) คลิกปุ่ม Next ในรูปที่ 3.17 แล้วจะได้หน้าต่างกดไป คลิกปุ่ม New Source แล้วจะได้หน้าต่างกดไป จากนั้นพิมพ์ชื่อ Source File ชื่อ ex3_1_1f ลงในช่อง File Name และคลิกที่ Schematic ดังรูปที่ 3.18 คลิก Next แล้วคลิก Finish คลิก Next อีก 2 ครั้ง และคลิก Finish แล้วจะได้หน้าต่าง Xilinx-ISE คลิกไอคอน แล้วตรวจสอบตามรูปที่ 3.16 จนแล้วเสร็จได้ดังรูปที่ 3.19
 5) การตรวจสอบความถูกต้อง ถ้าไม่มีข้อผิดพลาดก็ให้บันทึกไฟล์ คลิก (สีดำ) ปิดโปรแกรมเพื่อกลับไปที่หน้าต่าง Xilinx-ISE



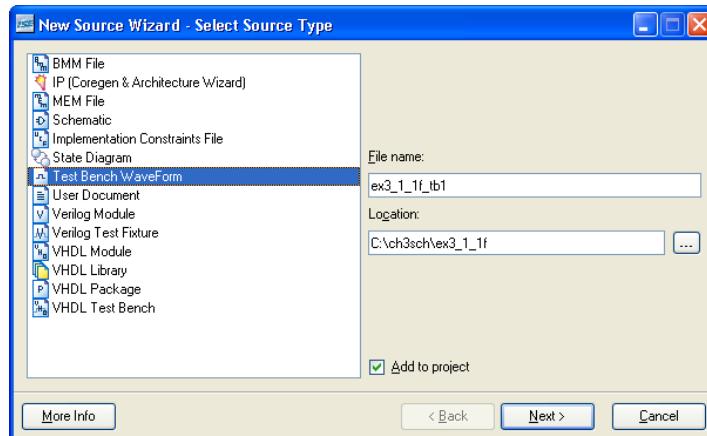
รูปที่ 3.18 หน้าต่าง New Project Wizard-Select Source Type



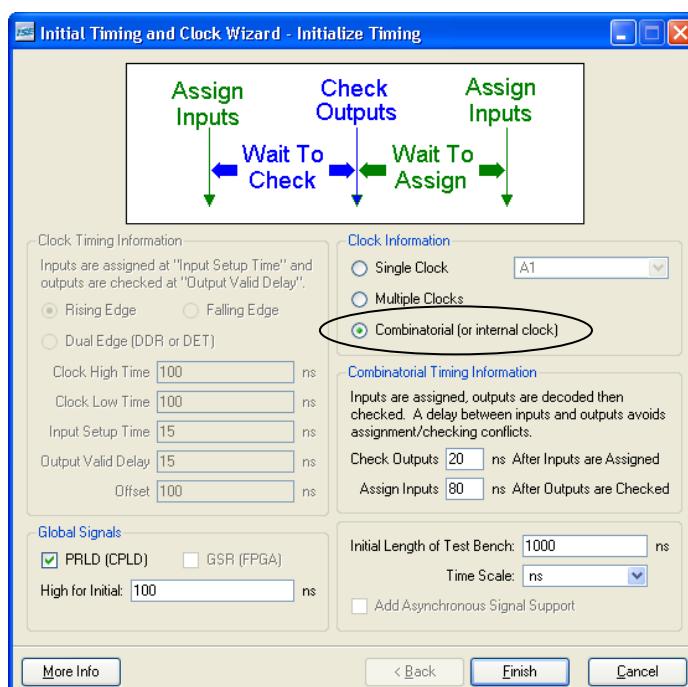
รูปที่ 3.19 หน้าต่าง Xilinx-ISE สำหรับEDAผังวงจร

2.2 การตรวจสอบความถูกต้องของจริงที่ออกแบบ (Design Verification)

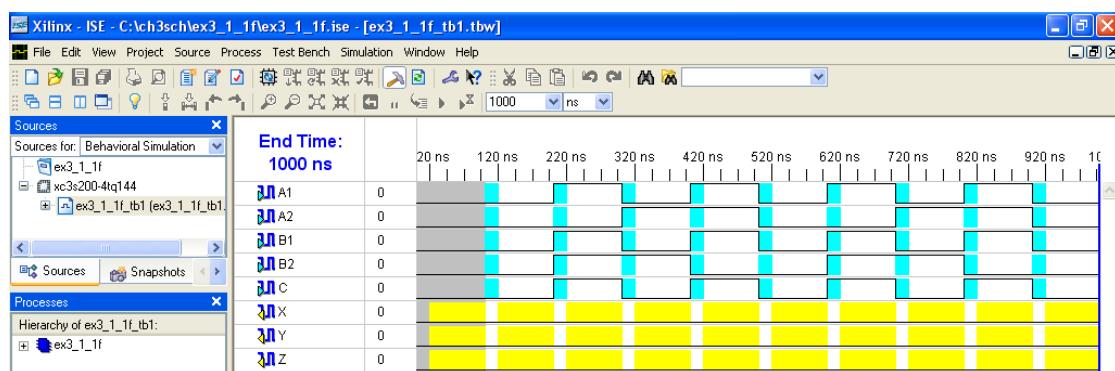
ให้ทำการขั้นตอนในข้อ 2.3 (ตัวอย่างที่ 2.3) ในบทที่ 2 พิมพ์ชื่อ ex3_1f_tb1 ดังรูปที่ 3.20 จากนั้นกำหนดค่าเวลา และ Waveform ดังรูปที่ 3.21 เพื่อจัดเตรียมให้พิจารณาผล Behavioral simulation เป็นไปตามทฤษฎีหรือไม่



รูปที่ 3.20 หน้าต่าง New Source Wizard



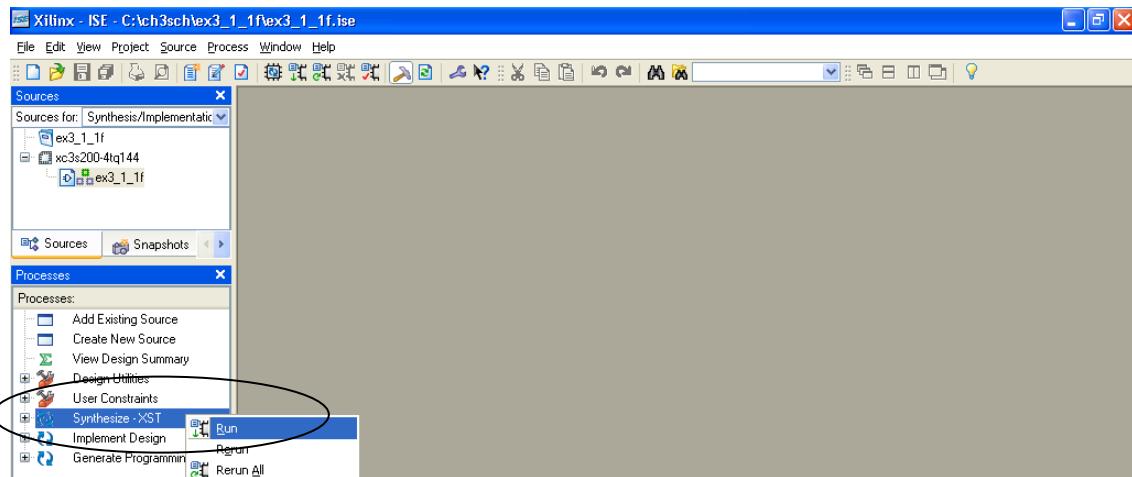
รูปที่ 3.21 หน้าต่าง Initial Timing and Clock Wizard



รูปที่ 3.22 หน้าต่างสำหรับกำหนดสัญญาณต่างๆ ที่ป้อนให้กับอินพุตของจริงที่เราต้องการทดสอบ

2.3 การสังเคราะห์วงจร (Design synthesis)

คลิกที่ ex3_1_1f ในหน้าต่าง Source และคลิกขวา Synthesize-XST แล้วคลิก Run ดังรูปที่ 3.23 ถ้าใช่ ✓ หรือ ⚠️ หน้า Synthesize-XST ถือว่าขั้นตอนนี้ผ่าน



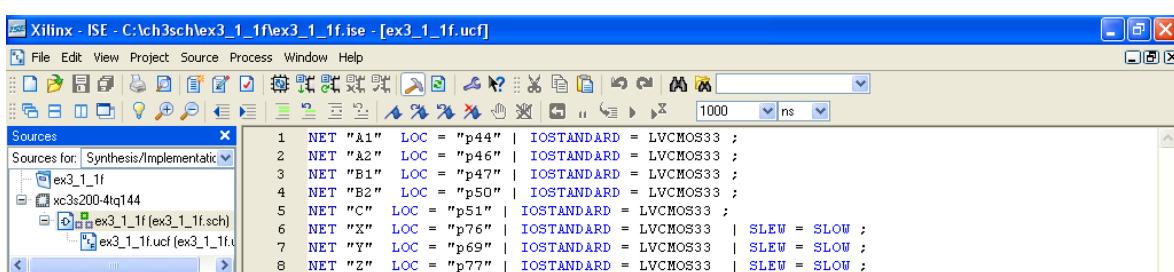
รูปที่ 3.23 ขั้นตอนการสังเคราะห์วงจร

2.4 Design implementation

- 1) สร้าง Implementation constraints file (UCF) โดยอัตโนมัติ
- 2) ขั้นตอนกำหนดสายสัญญาณต่างๆ ของวงจรในรูปที่ L1.1 เข้ากับขาของ FPGA จำเป็นต้องกำหนดให้อินพุตและเอาต์พุตของ FPGA สอดคล้องกับอุปกรณ์ที่เตรียมไว้บนบอร์ดทดลองดังนี้รายละเอียดตามตารางที่ 1.4 ในบทที่ 1 ซึ่งจะใช้ปุ่มกด PB1 ถึง PB5 เป็นอินพุต และ LED L1 ถึง L3 เป็นเอาต์พุต กล่าวคือ

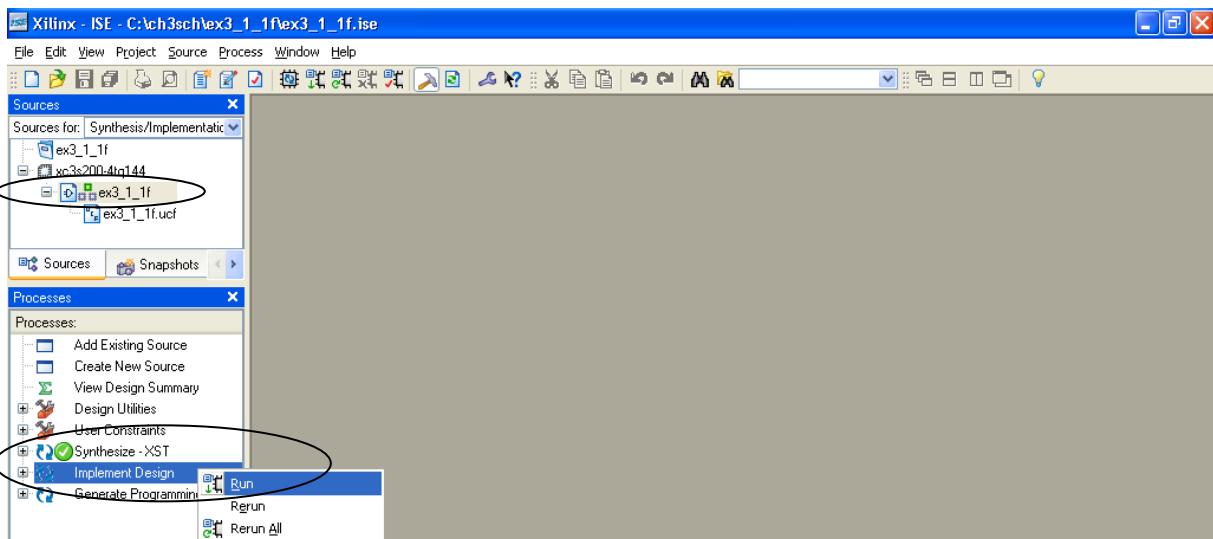
$A1 = PB1 = INPUT = p44$	$X = L3 = OUTPUT = p76$
$A2 = PB2 = INPUT = p46$	$Y = L2 = OUTPUT = p69$
$B1 = PB3 = INPUT = p47$	$Z = L1 = OUTPUT = p77$
$B2 = PB4 = INPUT = p50$	$C = PB5 = INPUT = p51$

ให้พิมพ์รายละเอียดต่างๆ ใน Edit Constraints (Text) เมื่อพิมพ์ครึ่งจะได้ดังรูปที่ 3.24 จากนั้นบันทึกไฟล์โดยคลิก คลิก ปิดโปรแกรม Edit Constraints (Text) เพื่อกลับไปที่หน้าต่าง Xilinx-ISE อีกครั้ง



รูปที่ 3.24 การกำหนดขาในหน้าต่าง Edit Constraints (Text)

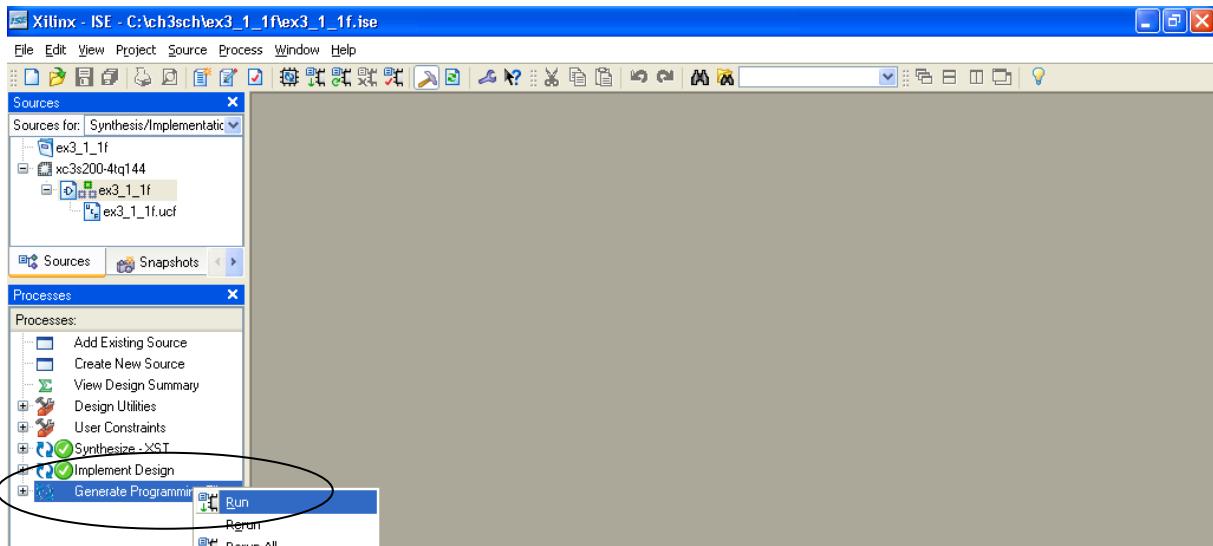
- 3) ขั้นตอน Implementation คลิกที่ ex3_1_1f ในหน้าต่าง Source และคลิกขวาที่ Implement Design ในหน้าต่าง Processes แล้วคลิก Run ดังรูปที่ 3.25 เสร็จแล้วถ้าใช่ ✓ หรือ ⚠️ ที่หน้า Synthesize-XST ถือว่าขั้นตอน Implement Design ผ่าน



รูปที่ 3.25 ขั้นตอน Implementation

2.5 การโปรแกรมข้อมูลวงจรลงชิป

ขั้นตอน Generate Programming File คลิกที่ ex3_1_1f ในหน้าต่าง Source และคลิกที่ Generate Programming File ในหน้าต่าง Processes แล้วคลิก run ดังรูปที่ 3.26 เมื่อเครื่องเด้งจะได้ ✓ ที่หน้าແນก Generate Programming File



รูปที่ 3.26 ขั้นตอน Generate Programming File

ขั้นตอนดาวน์โหลดจะเหมือนกับตัวอย่างที่ 2.2 ในบทที่ 2 โดยดาวน์โหลดไฟล์ชื่อ ex3_1_1f.bit ลง FPGA และให้กดปุ่ม PB1-PB5 และดูผลที่ LED L1- L3 ว่าให้อาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นบันทึกผลการทดลอง และต้องพึงระลึกอยู่เสมอว่า Push button switch ทำงานเป็นแบบ Active low กล่าวคือ ป้ำกดปุ่มก็จะเป็นโลจิก ‘0’ แต่ถ้าไม่กดจะเป็นโลจิก ‘1’

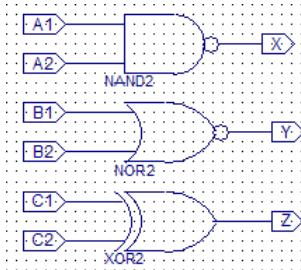
หมายเหตุ ถ้าต้องการโปรแกรมข้อมูลวงจรลง PROM ด้วยนั้นเราจะต้องสร้างไฟล์ PROM ก่อน ให้ครายละเอียดทั้งหมดในข้อที่ 2.2.5 (ในตัวอย่างที่ 2.2) ของบทที่ 2 โดยดาวน์โหลดไฟล์ชื่อ ex3_1_1f.mcs ลง Flash PROM

3.1.2 การออกแบบวงจรลอจิกเกต (ต่อ)

อุปกรณ์ที่สามารถใช้ออกแบบวงจรด้วย CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างແນນດີເກຕ ນອຣເກຕ ແລະ ເອັກຄູ້ໜີ່ພອອ່ຽວໜີ່

ให้ออกแบบวงจรແນນດີເກຕ ນອຣເກຕ ແລະ ເອັກຄູ້ໜີ່ພອອ່ຽວໜີ່ ດ້ວຍ CPLD ດັງຮູບທີ່ 3.27 ຕາມຂັ້ນຕອນໃນບົທີ່ 2 ໂດຍໃຫ້ Project Location (ຫຼື້ວ່າ Folder) ປື້ອ ch3sch ແລ້ວກຳນົດ Project Name ແລະ Source File ປື້ອ ex3_1_2cxl



ຮູບທີ່ 3.27 ວິທີສ່ວນຕົວຈິງ

ການກຳນົດຂາສັນນາມຕ່າງໆ ຈະໃຫ້ປຸ່ມກົດ PB1 ປຶ້ງ PB6 ເປັນອິນພຸດ ແລະ LED1 ປຶ້ງ LED3 ເປັນເອົາດີພຸດ ກ່າວກີ່ອ

$$\begin{array}{ll}
 A1 = PB1 = \text{INPUT} = p39 & X = \text{LED1} = \text{OUTPUT} = p38 \\
 A2 = PB2 = \text{INPUT} = p40 & Y = \text{LED2} = \text{OUTPUT} = p37 \\
 B1 = PB3 = \text{INPUT} = p42 & Z = \text{LED3} = \text{OUTPUT} = p36 \\
 B2 = PB4 = \text{INPUT} = p43 & C1 = PB5 = \text{INPUT} = p44 \\
 & C2 = PB6 = \text{INPUT} = p1
 \end{array}$$

ໂດຍພິມຟື້ນໃນ Edit Constraints (Text) ສໍາງປັດຈຸນີ້ເກີດ

```

NET "A1" LOC = "p39" ;
NET "A2" LOC = "p40" ;
NET "B1" LOC = "p42" ;
NET "B2" LOC = "p43" ;
NET "C1" LOC = "p44" ;
NET "C2" LOC = "p1" ;
NET "X" LOC = "p38" | SLEW = SLOW ;
NET "Y" LOC = "p37" | SLEW = SLOW ;
NET "Z" LOC = "p36" | SLEW = SLOW ;
  
```

ໜັງຈາກໂປຣແກຣມວິທີອຳນວຍກົດປຸ່ມ PB1-PB6 ແລະ ໄກສັ່ງເກຕຄູ້ໜີ່ LED1-LED3 ວ່າ
ໄກ້ລອງເອົາດີພຸດຕິດສ່ວ່າງເປັນໄປຕາມທຸນຢູ່ຫຼື່ອໄມ່ ຈາກນັ້ນຈຶ່ງນັບທີກຳລັງການທີ່

2. ສ້າງ ແນນດີເກຕ ນອຣເກຕ ແລະ ເອັກຄູ້ໜີ່ພອອ່ຽວໜີ່

ให้อຳນວຍກົດປຸ່ມ PB1-PB6 ແລະ ເອັກຄູ້ໜີ່ພອອ່ຽວໜີ່ ດ້ວຍ CPLD ດັງຮູບທີ່ 3.27 ຕາມຂັ້ນຕອນໃນບົທີ່ 2 ໂດຍໃຫ້ Project Location (ຫຼື້ວ່າ Folder) ປື້ອ ch3sch ແລ້ວກຳນົດ Project Name ແລະ Source File ປື້ອ ex3_1_2f

ການກຳນົດຂາສັນນາມຕ່າງໆ ຈະໃຫ້ Dip SW1-Dip SW6 ເປັນອິນພຸດ ແລະ LED L1-L3 ເປັນເອົາດີພຸດ ກ່າວກີ່ອ

A1 = Dip SW1 = INPUT = p52	X = L3 = OUTPUT = p76
A2 = Dip SW2 = INPUT = p53	Y = L2 = OUTPUT = p69
B1 = Dip SW3 = INPUT = p55	Z = L1 = OUTPUT = p77
B2 = Dip SW4 = INPUT = p56	C1 = Dip SW5 = INPUT = p59
	C2 = Dip SW6 = INPUT = p60

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้คือ

```

NET "A1" LOC = "p52" | IOSTANDARD = LVCMOS33 ;
NET "A2" LOC = "p53" | IOSTANDARD = LVCMOS33 ;
NET "B1" LOC = "p55" | IOSTANDARD = LVCMOS33 ;
NET "B2" LOC = "p56" | IOSTANDARD = LVCMOS33 ;
NET "C1" LOC = "p59" | IOSTANDARD = LVCMOS33 ;
NET "C2" LOC = "p60" | IOSTANDARD = LVCMOS33 ;
NET "X" LOC = "p76" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Y" LOC = "p69" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Z" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;

```

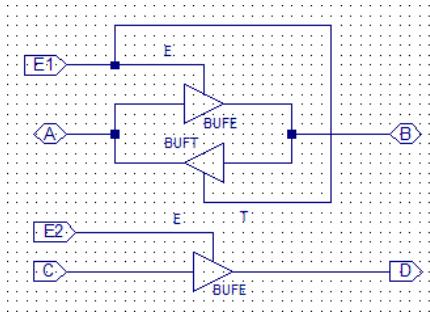
หลังจากโปรแกรมจะที่ออกแบบลงชิพ FPGA แล้วให้ทดลอง ON-OFF Dip SW1-Dip SW6 และให้สังเกตดูผลที่ LED L1-L3 ว่าให้ล็อกเอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

3.1.3 การออกแบบวงจรบับเฟอร์

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL

1. สร้างไตร-สเตตบับเฟอร์และบับเฟอร์สองทิศทางด้วย CPLD

ให้ออกแบบวงจรไตร-สเตตบับเฟอร์และบับเฟอร์สองทิศทางด้วย CPLD ดังรูปที่ 3.28 ตามขั้นตอนในบทที่ 2 โดยใช้ Project Location (หรือ Folder) ชื่อ ch3sch แล้วกำหนด Project Name และ Source File ชื่อ ex3_1_3cxl



รูปที่ 3.28 วงจร logic เกตต่างๆ

การกำหนดขาสัญญาณต่างๆจะใช้ IO6-IO8 เป็นอินพุตเอาต์พุตโดยมี LED Logic monitor MN1-MN3 ที่ต่อพ่วงอยู่ด้วย เป็นเอาต์พุตและ Slide SW1-Slide SW3 เป็นอินพุต ก่อร่างกาย

$$A = IO6 = \text{INPUT/ OUTPUT}, MN1 = p7$$

$$E1 = \text{Slide SW1} = \text{INPUT} = p42$$

$$B = IO7 = \text{INPUT/ OUTPUT}, MN2 = p6$$

$$E2 = \text{Slide SW2} = \text{INPUT} = p43$$

$$C = \text{Slide SW3} = \text{INPUT} = p44$$

$$D = IO8 = \text{OUTPUT}, MN3 = p4$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้คือ

```
NET "A" LOC = "p7" | SLEW = SLOW ;
NET "B" LOC = "p6" | SLEW = SLOW ;
NET "C" LOC = "p44" ;
NET "D" LOC = "p4" | SLEW = SLOW ;
NET "E1" LOC = "p42" ;
NET "E2" LOC = "p43" ;
```

หลังจากโปรแกรมวงจรลง CPLD แล้ว ให้ OFF Slide SW2 ('1') และ ON/OFF Slide SW3 สลับกันไปมาแล้วสังเกตคุณ LED MN3 ('1'= เหลือง, '0'= แดง, ดับ = High impedance) จากนั้นให้ ON Slide SW2 ('0') แล้วให้ ON/OFF Slide SW3 สลับกันไปมาแล้วสังเกตคุณ LED MN3 อีกครั้ง ให้สูญเสียจิกที่ได้เป็นไปตามทฤษฎีหรือไม่ จากนั้นให้ OFF Slide SW1 ('1') แล้วให้ต่อสายไฟที่ขั้ว IO6 (Pin No.13 ของ K1) กับขั้ว Vcc= 3.3V (Pin No.2 ของ K1) สลับกันไปมากับขั้ว GND (Pin No.4 ของ K1) และสังเกตคุณ LED MN2 (IO7) จากนั้นให้ ON Slide SW1 ('0') แล้วให้ต่อสายไฟที่ขั้ว IO7 (Pin No.15 ของ K1) กับขั้ว Vcc= 3.3V (Pin No.2 ของ K1) สลับกันไปมากับขั้ว GND (Pin No.4 ของ K1) และสังเกตคุณ LED MN1 (IO6) จากนั้นจึงบันทึกผลการทดลองและสรุปว่าจิกที่ได้เป็นไปตามทฤษฎีหรือไม่

3.2 วงจรบวก

วงจรบวกเป็นวงจรที่ประกอบด้วยล็อกจิกเกตต่างๆหรือวงจรคอมบินेशัน (Combinational logic circuits ซึ่งจะอธิบายในที่นี้เพียง 2 แบบ กือ Half adder และ Full adder วงจร Half adder เป็นวงจรบวกที่มี 2 อินพุตและ 2 เอาต์พุตคือ A, B, ผลรวม (Sum) หรือ S และตัวทอล (Carry) หรือ Co ตั้งตารางความจริง (Truth Table) ในรูปที่ 3.29 วงจร Full adder เป็นวงจรบวกขนาด 1

บิตที่ประกอบด้วย 3 อินพุตและ 2 เอ้าต์พุต คือ A, B, ตัวทดเข้า (Carry-in) หรือ Cin, ผลรวม (Sum) หรือ S และ ตัวทดออก (Carry-out) หรือ Co ดังตารางความจริงในรูปที่ 3.30

Input		Output	
A	B	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

รูปที่ 3.29 ตารางความจริงของวงจร Half adder

Input			Output	
A	B	Cin	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

รูปที่ 3.30 ตารางความจริงของวงจร Full adder

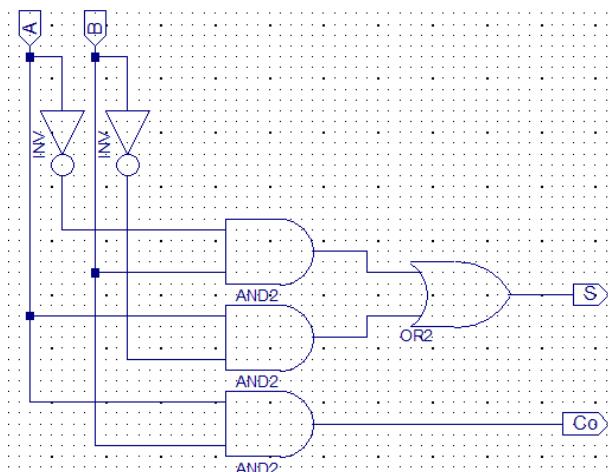
การสังเคราะห์วงจร

จากตารางความจริงของวงจร Half adder ขนาด 1 บิตในรูปที่ 3.29 เขียนสมการบูลีน (พิจารณาเฉพาะช่องเอ้าต์พุตที่เป็นลอจิก ‘1’ และแทน $A = '0'$ ด้วย \overline{A} และ $B = '0'$ ด้วย \overline{B} ส่วน $A = '1'$ แทนด้วย A และ $B = '1'$ แทนด้วย B) จะได้ดังนี้

$$S = \overline{A} \cdot B + A \cdot \overline{B} = A \oplus B$$

$$Co = A \cdot B$$

เมื่อนำสมการบูลีนไปเขียนเป็นวงจรจะได้ดังรูปที่ 3.31 ซึ่งในที่นี่เพื่อความประยัดควรจะใช้อินเวอร์เตอร์ร่วมกัน



รูปที่ 3.31 วงจร Half adder

ในทำนองเดียวกันจากตารางความจริงของวงจร Full adder ขนาด 1 บิต ในรูปที่ 3.30 เขียนเป็นสมการบูลีน ได้ดังนี้

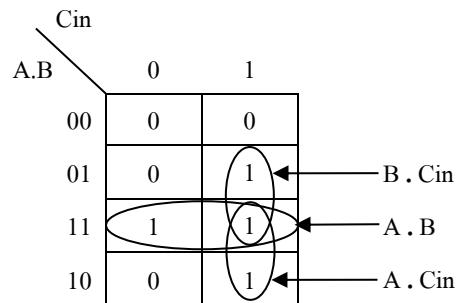
$$S = \overline{A} \cdot \overline{B} \cdot Cin + \overline{A} \cdot B \cdot \overline{Cin} + A \cdot \overline{B} \cdot \overline{Cin} + A \cdot B \cdot Cin \quad (\text{สมการ SOP})$$

$$= A \oplus B \oplus Cin$$

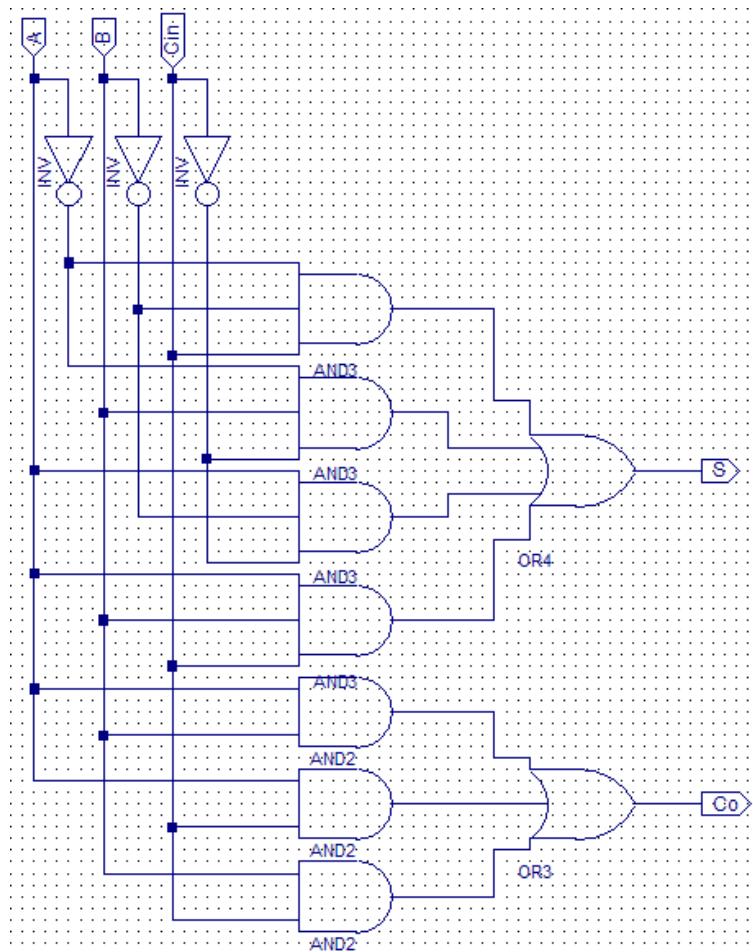
$$Co = \overline{A} \cdot B \cdot Cin + A \cdot \overline{B} \cdot Cin + A \cdot B \cdot \overline{Cin} + A \cdot B \cdot Cin \quad (\text{สมการ SOP})$$

= $A \cdot B + A \cdot Cin + B \cdot Cin$ (ลดรูปโดยใช้ K-map ดังรูปที่ 3.32)

และเมื่อนำเอา S และ Co ที่ได้จากสมการ SOP มาเขียนเป็นวงจรจะได้ดังรูปที่ 3.33



รูปที่ 3.32 แสดงวิธีลดรูป Co ของวงจร Full adder โดยใช้ K-map



รูปที่ 3.33 วงจร Full adder

3.2.1 การออกแบบวงจร Half adder

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจร Half adder ด้วย CPLD

ให้ออกแบบวงจร Half adder ด้วย CPLD ดังรูปที่ 3.31 ตามขั้นตอนที่ได้เรียนมาแล้วในบทที่ 2 โดยการทดลองนี้จะใช้ Project Location (หรือ Folder) ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_2_1cxl

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB2 เป็นอินพุต และ LED1-LED2 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{ll} A = PB1 = \text{INPUT} = p39 & S = LED2 = \text{OUTPUT} = p37 \\ B = PB2 = \text{INPUT} = p40 & Co = LED1 = \text{OUTPUT} = p38 \end{array}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "A" LOC = "p39" ;
NET "B" LOC = "p40" ;
NET "Co" LOC = "p38" | SLEW = SLOW ;
NET "S" LOC = "p37" | SLEW = SLOW ;
```

หลังจากโปรแกรมจะที่ออกแบบลงชิป CPLD แล้วให้ทดลองกดปุ่ม PB1-PB2 และให้สังเกตดูผลที่ LED1-LED2 ว่า ให้ล็อกเอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

2. สร้างวงจร Half adder ด้วย FPGA

ให้ออกแบบวงจร Half adder ด้วย FPGA ดังรูปที่ 3.31 ตามขั้นตอนที่ได้เรียนมาแล้วในบทที่ 2 โดยการทดลองนี้จะใช้ Project Location (หรือ Folder) ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_2_1f

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB2 เป็นอินพุต และ LED L2-L3 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{ll} A = PB1 = \text{INPUT} = p44 & S = L2 = \text{OUTPUT} = p69 \\ B = PB2 = \text{INPUT} = p46 & Co = L3 = \text{OUTPUT} = p76 \end{array}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "A" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "B" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "Co" LOC = "p76" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "S" LOC = "p69" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
```

หลังจากโปรแกรมจะที่ออกแบบลง FPGA แล้วให้ทดลองกดปุ่ม PB1-PB2 และให้สังเกตดูผลที่ LED L2-L3 ว่าให้ล็อกเอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

3.2.2 การออกแบบวงจร Full adder

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจร Full adder ขนาด 1 บิตด้วย CPLD

ให้ออกแบบวงจร Full adder ขนาด 1 บิตด้วย CPLD ดังรูปที่ 3.33 ตามขั้นตอนที่ได้อธิบายมาแล้วในบทที่ 2 โดยการทดลองนี้จะใช้ Project Location (หรือ Folder) ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_2_2cxl

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB3 เป็นอินพุต และ LED3-LED4 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{lll} A = PB1 = \text{INPUT} = p39 & Cin = PB3 = \text{INPUT} = p42 & S = \text{LED4} = \text{OUTPUT} = p35 \\ B = PB2 = \text{INPUT} = p40 & & Co = \text{LED3} = \text{OUTPUT} = p36 \end{array}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้คือ

```
NET "A" LOC = "p39" ;
NET "B" LOC = "p40" ;
NET "Cin" LOC = "p42" ;
NET "Co" LOC = "p36" | SLEW = SLOW ;
NET "S" LOC = "p35" | SLEW = SLOW ;
```

เมื่อโปรแกรม CPLD แล้ว กดปุ่ม PB1-PB3 และดูผล LED3-LED4 ว่าเป็นตามทฤษฎีหรือไม่ บันทึกผลการทดลอง

2. สร้างวงจร Full adder ขนาด 1 บิตด้วย FPGA

ให้ออกแบบวงจร Full adder ขนาด 1 บิตด้วย FPGA ดังรูปที่ 3.33 ตามขั้นตอนที่ได้อธิบายมาแล้วในบทที่ 2 โดยการทดลองนี้จะใช้ Project Location (หรือ Folder) ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_2_2f

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB3 เป็นอินพุต และ LED L0-L1 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{lll} A = PB1 = \text{INPUT} = p44 & Cin = PB3 = \text{INPUT} = p47 & S = L0 = \text{OUTPUT} = p70 \\ B = PB2 = \text{INPUT} = p46 & & Co = L1 = \text{OUTPUT} = p77 \end{array}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "A" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "B" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "Cin" LOC = "p47" | IOSTANDARD = LVCMOS33 ;
NET "Co" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "S" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
```

เมื่อโปรแกรม FPGA แล้ว กดปุ่ม PB1-PB3 และดูผลที่ LED L0-L1 ว่าเป็นตามทฤษฎีหรือไม่ บันทึกผลการทดลอง

3.3 วงจรดอร์หัส

วงจรดอร์หัสจัดเป็นประเภทวงจรคอมบินेशัน วงจรดอร์หัส เช่น 2 to 4 Decoder และ วงจรดอร์หัส BCD เป็นวงจรเซกเมนต์ (BCD to 7 Segment Decoder) เป็นต้น รูปที่ 3.34 แสดงตารางความจริงของวงจร 2 to 4 Decoder ดังนี้ถ้าขา E = '0' จะทำให้อเอต์พุต D0-D3 = '0' แต่ถ้าขา E = '1' อเอต์พุต D0-D3 จะเป็นตามตารางความจริง

Input			Output			
E	A1	A0	D3	D2	D1	D0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

รูปที่ 3.34 ตารางความจริงของวงจร 2 to 4 Decoder

จากตารางความจริงเขียนสมการ SOP ได้ดังนี้ (เลือกเฉพาะช่องที่เอต์พุตเป็นค่าจิก '1')

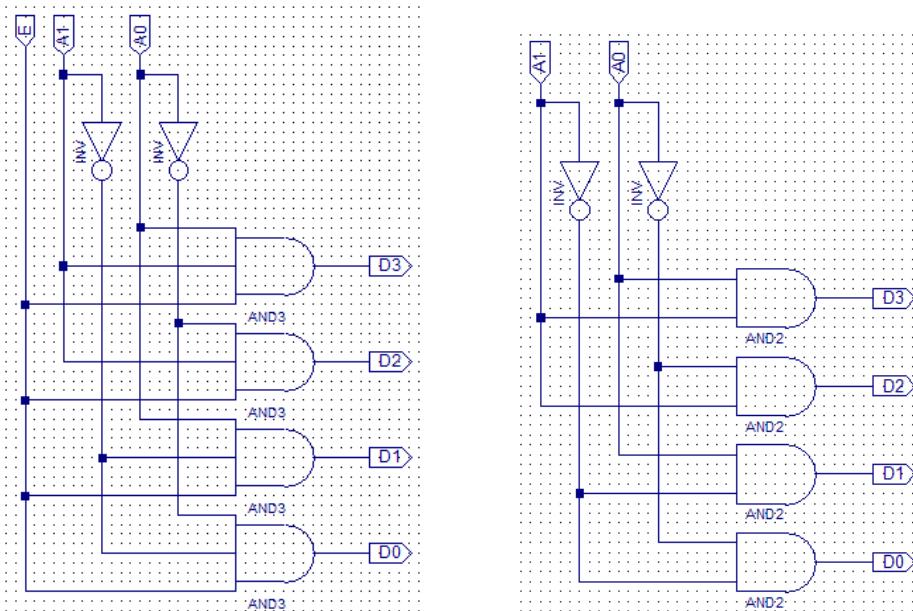
$$D0 = E \cdot \overline{A0} \cdot \overline{A1}$$

$$D1 = E \cdot \overline{A0} \cdot A1$$

$$D2 = E \cdot A0 \cdot \overline{A1}$$

$$D3 = E \cdot A0 \cdot A1$$

จากสมการบูรณาจัดได้วงจร 2 to 4 Decoder แบบมีขา Enable (E) ดังรูปที่ 3.35(a) วงจร 2 to 4 Decoder แบบบูรณาจัด (หรือ E = '1') นั้นวงจรจะครุ่นลงได้ดังในรูปที่ 3.35(b)



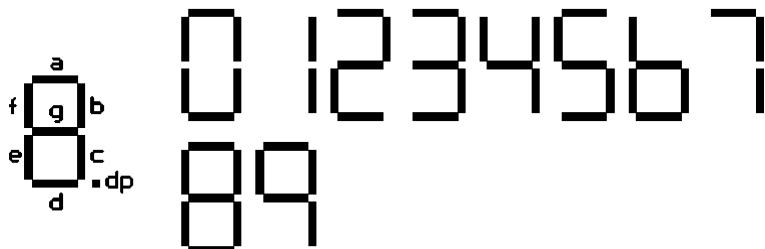
(a) วงจร 2 to 4 Decoder แบบมีขา Enable

(b) วงจร 2 to 4 Decoder

รูปที่ 3.35 วงจร 2 to 4 Decoder

วงจรดอร์หัสอีกประเภทหนึ่ง คือ วงจรดอร์หัสตัวแสดงผลเซกเมนต์ (7-Segment LED display) ได้แก่ วงจร ดอร์หัสตัวแสดงผลเซกเมนต์ (7-Segment decoder) ที่อยู่ในรูปแบบเลขฐานสิบ (Decimal digit display format) และ

รูปแบบเลขฐานสิบหก (Hexadecimal digit display format) แสดงดังรูปที่ 3.36(a) และ รูปที่ 3.36(b) ตามลำดับ ซึ่งตัวแสดงผลใช้เวนเซกเม้นต์ประกอบด้วย LED แสดงผล 7 ส่วน คือ a, b, c, d, e, f, g และจุด (dp = Decimal point) โดยที่ความแตกต่างระหว่างรูปแบบเลขฐานสิบและฐานสิบหกจะอยู่ที่เลข 6 และเลข 9 ที่มีส่วนหรือของเซกเม้นต์ (Segment) a และ d เพิ่มเข้ามา ตามลำดับเพื่อทำให้เลข 6 ไม่ไปข้ามกับตัว 6 และรูปแบบเลขฐานสิบหกจะมี A, b, C, d, E, และ F เพิ่มเข้ามา



(a) รูปแบบเลขฐานสิบ



(b) รูปแบบเลขฐานสิบหก

รูปที่ 3.36 รูปแบบการแสดงผลบนตัวแสดงผลใช้เวนเซกเม้นต์

ตัวแสดงผลใช้เวนเซกเม้นต์มี 2 ชนิด คือ แบบแอนโครร่วม (Common Anode) และแบบแคโอดร่วม (Common cathode) โดยที่ตัวแสดงผลแบบแคโอดร่วมนั้นแต่ละส่วน (Segment) จะติดสว่างก็ต่อเมื่อมีการส่งลอจิก ‘1’ ให้ LED แต่ละส่วน (ผ่านทางตัวต้านทานเพื่อจำกัดกระแส) และส่งลอจิก ‘0’ ให้หัวร่วม (Common) หรือต่องกราร้าด์ แต่ถ้าตัวแสดงผลเป็นแบบแอนโครร่วมก็จะทำงานตรงข้ามกัน ตารางความจริงของวงจรดอครหัสตัวแสดงผลใช้เวนเซกเม้นต์แบบเลขฐานสิบ และ เลขฐานสิบหกแบบแคโอดร่วมนั้นแสดงดังรูปที่ 3.37 และรูปที่ 3.38 ตามลำดับ

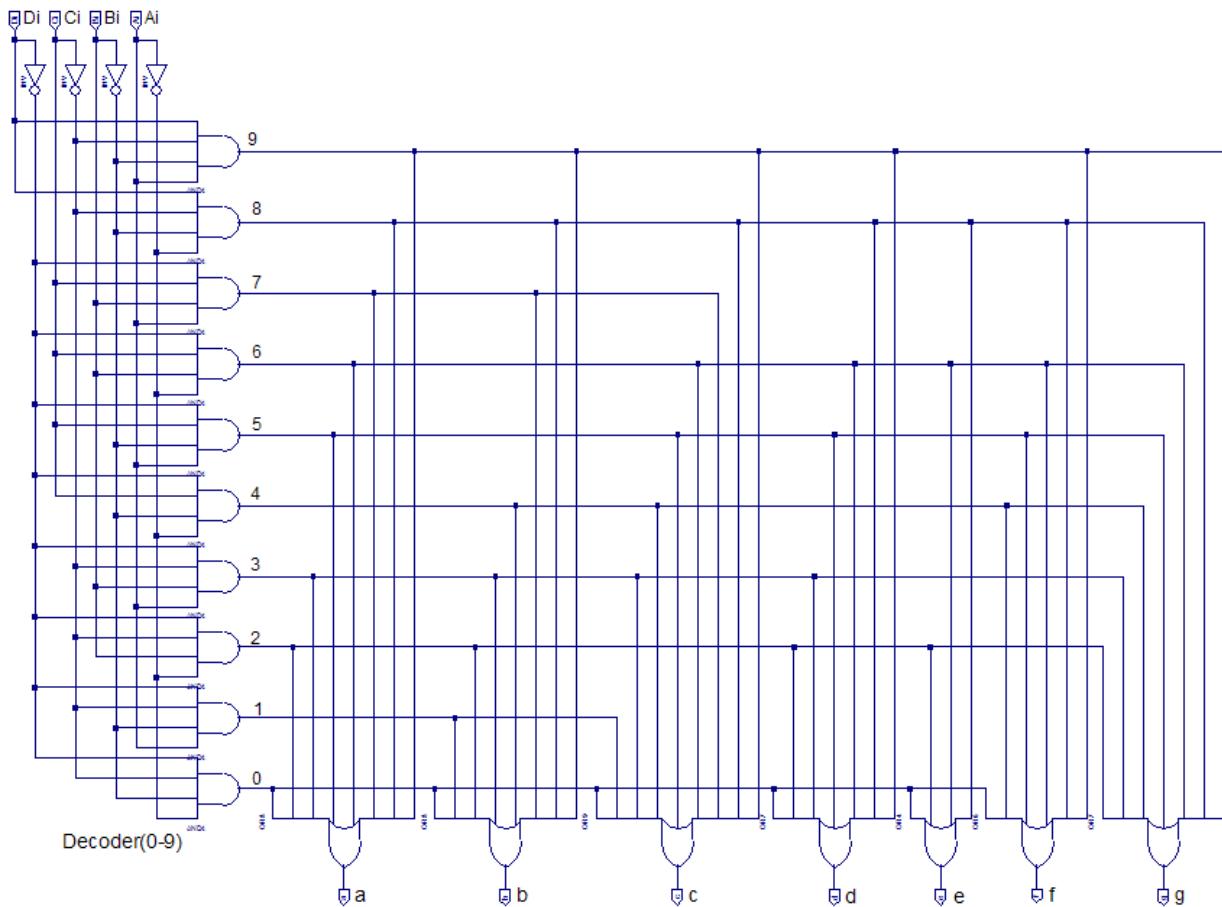
No.	Input				Output						
	Di	Ci	Bi	Ai	g	f	e	d	c	b	a
0	0	0	0	0	0	1	1	1	1	1	1
1	0	0	0	1	0	0	0	0	1	1	0
2	0	0	1	0	1	0	1	1	0	1	1
3	0	0	1	1	1	0	0	1	1	1	1
4	0	1	0	0	1	1	0	0	1	1	0
5	0	1	0	1	1	1	0	1	1	0	1
6	0	1	1	0	1	1	1	1	1	0	0
7	0	1	1	1	0	0	0	0	1	1	1
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	0	0	1	1	1

รูปที่ 3.37 ตารางความจริงของตัวดอครหัส Decimal digit display format แบบแคโอดร่วม

No.	Input				Output						
	Di	Ci	Bi	Ai	g	f	e	d	c	b	a
0	0	0	0	0	0	1	1	1	1	1	1
1	0	0	0	1	0	0	0	0	1	1	0
2	0	0	1	0	1	0	1	1	0	1	1
3	0	0	1	1	1	0	0	1	1	1	1
4	0	1	0	0	1	1	0	0	1	1	0
5	0	1	0	1	1	1	0	1	1	0	1
6	0	1	1	0	1	1	1	1	1	0	1
7	0	1	1	1	0	0	0	0	1	1	1
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	0	1	1	1	1
10 = A	1	0	1	0	1	1	1	0	1	1	1
11 = b	1	0	1	1	1	1	1	1	1	0	0
12 = C	1	1	0	0	0	1	1	1	0	0	1
13 = d	1	1	0	1	1	0	1	1	1	1	0
14 = E	1	1	1	0	1	1	1	1	0	0	1
15 = F	1	1	1	1	1	1	1	0	0	0	1

รูปที่ 3.38 ตารางความจริงของตัวถอดรหัส Hexadecimal digit display format แบบแคล็อกร่วม

ในการออกแบบวงจรถอดรหัส BCD เป็นเซเว่นเซกเมนต์ (BCD to 7-Segment Decoder) ในหนังสือเล่มนี้จะใช้รูปแบบเลขฐานสิบหกดังในรูปที่ 3.36(b) และจากตารางความจริงรูปที่ 3.38 นั้นเราจะสร้างวงจรถอดรหัส BCD เป็นเลขฐานสิบ (BCD to Decimal Decoder) ขึ้นมาก่อน แล้วนำเอาต์พุตที่ได้ในขั้นตอนนี้ไป OR กันเพื่อสร้างเป็นเอาต์พุตของวงจรแต่ละเซกเมนต์ ซึ่งวงจรถอดรหัสแบบแคล็อกร่วมที่ออกแบบสมบูรณ์แล้วแสดงดังรูปที่ 3.39



รูปที่ 3.39 วงจรถอดรหัส BCD เป็นเซเว่นเซกเมนต์แบบแคล็อกร่วม

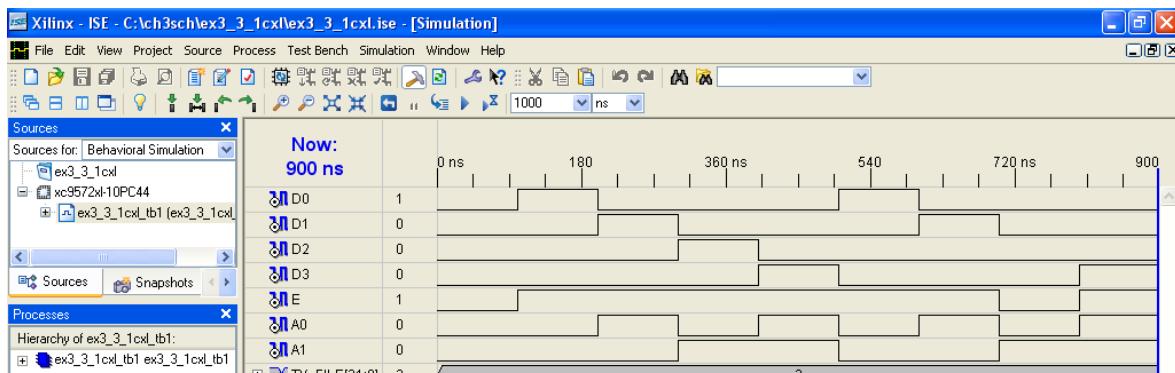
3.3.1 การออกแบบวงจร 2 to 4 Decoder

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจร 2 to 4 Decoder ขนาด 1 บิตด้วย CPLD

ให้ออกแบบวงจร วงจรต่อครึ่ง เช่น 2 to 4 Decoder ด้วย CPLD ดังรูปที่ 3.11(a) ตามขั้นตอนที่อธิบายในบทที่ 2 โดยการทดลองนี้จะใช้ Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_3_1cxl

ทำการตรวจสอบวงจรที่ออกแบบด้วย Behavioral Simulation โดยให้ทำตามขั้นตอนในบทที่ 2 ข้อ 2.3 โดยใช้ไฟล์ชื่อ ex3_3_1cxl_tb1 เสร็จแล้วให้พิจารณาผลในรูปที่ 3.40 ว่า Behavioral simulation เป็นไปตามทฤษฎีหรือไม่



รูปที่ 3.40 ผล Behavioral simulation

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB3 เป็นอินพุต และ LED1-LED4 เป็นเอาต์พุต กล่าวคือ

$A1 = PB1 = INPUT = p39$	$D0 = LED1 = OUTPUT = p38$
$A0 = PB2 = INPUT = p40$	$D1 = LED2 = OUTPUT = p37$
$E = PB3 = INPUT = p42$	$D2 = LED3 = OUTPUT = p36$
	$D3 = LED4 = OUTPUT = p35$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "A0" LOC ="P40";
NET "A1" LOC ="P39";
NET "E" LOC ="P42";
NET "D0" LOC ="P38" | SLEW=SLOW;
NET "D1" LOC ="P37" | SLEW=SLOW;
NET "D2" LOC ="P36" | SLEW=SLOW;
NET "D3" LOC ="P35" | SLEW=SLOW;

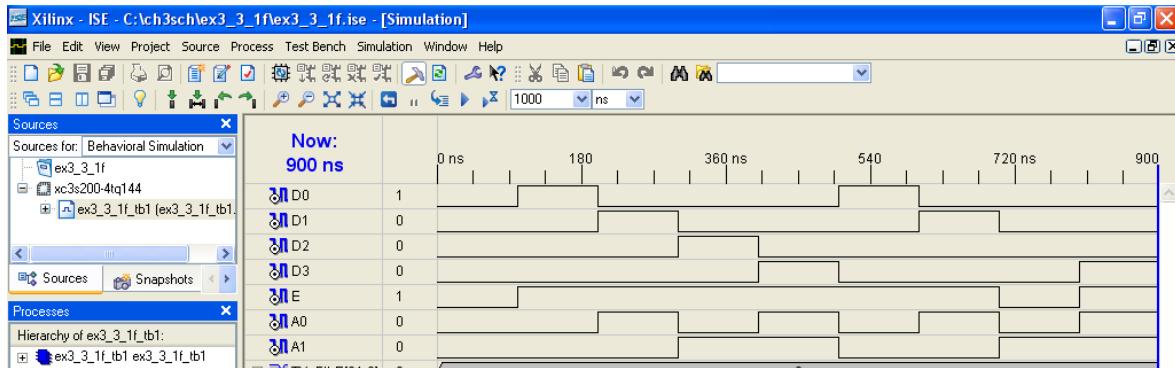
```

หลังจากโปรแกรมวงจรที่ออกแบบลงชิป CPLD แล้วให้ทดลองกดปุ่ม PB1-PB3 และให้สังเกตคุณลักษณะของ LED1-LED4 ว่า ให้ล้อจิกเอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

2. สร้างวงจร 2 to 4 Decoder ขนาด 1 บิตด้วย FPGA

ให้ออกแบบวงจร วงจรลดครึ่ง เข่น 2 to 4 Decoder ด้วย FPGA ดังรูปที่ 3.11(a) ตามขั้นตอนที่อธิบายในบทที่ 2 โดยการทดลองนี้จะใช้ Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_3_1f

ตรวจสอบวงจรที่ออกแบบด้วย Behavioral Simulation ตามขั้นตอนในบทที่ 2 ข้อ 2.9 โดยใช้ไฟล์ชื่อ ex3_3_1f_tb1 เสร็จแล้วให้พิจารณาผลในรูปที่ 3.41 ว่า Behavioral simulation เป็นไปตามทฤษฎีหรือไม่



รูปที่ 3.41 ผล Behavioral simulation

จากนั้นกำหนดขาสัญญาณต่างๆจะใช้ปุ่มกด PB1-PB3 เป็นอินพุตและ LED L0-L3 เป็นเอาต์พุต ก่อร่างกาย

$$\begin{array}{ll}
 A1 = PB1 = \text{INPUT} = p44 & D0 = L3 = \text{OUTPUT} = p76 \\
 A0 = PB2 = \text{INPUT} = p46 & D1 = L2 = \text{OUTPUT} = p69 \\
 E = PB3 = \text{INPUT} = p47 & D2 = L1 = \text{OUTPUT} = p77 \\
 & D3 = L0 = \text{OUTPUT} = p70
 \end{array}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "AO" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "A1" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "DO" LOC = "p76" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1" LOC = "p69" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D2" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D3" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "E" LOC = "p47" | IOSTANDARD = LVCMOS33 ;

```

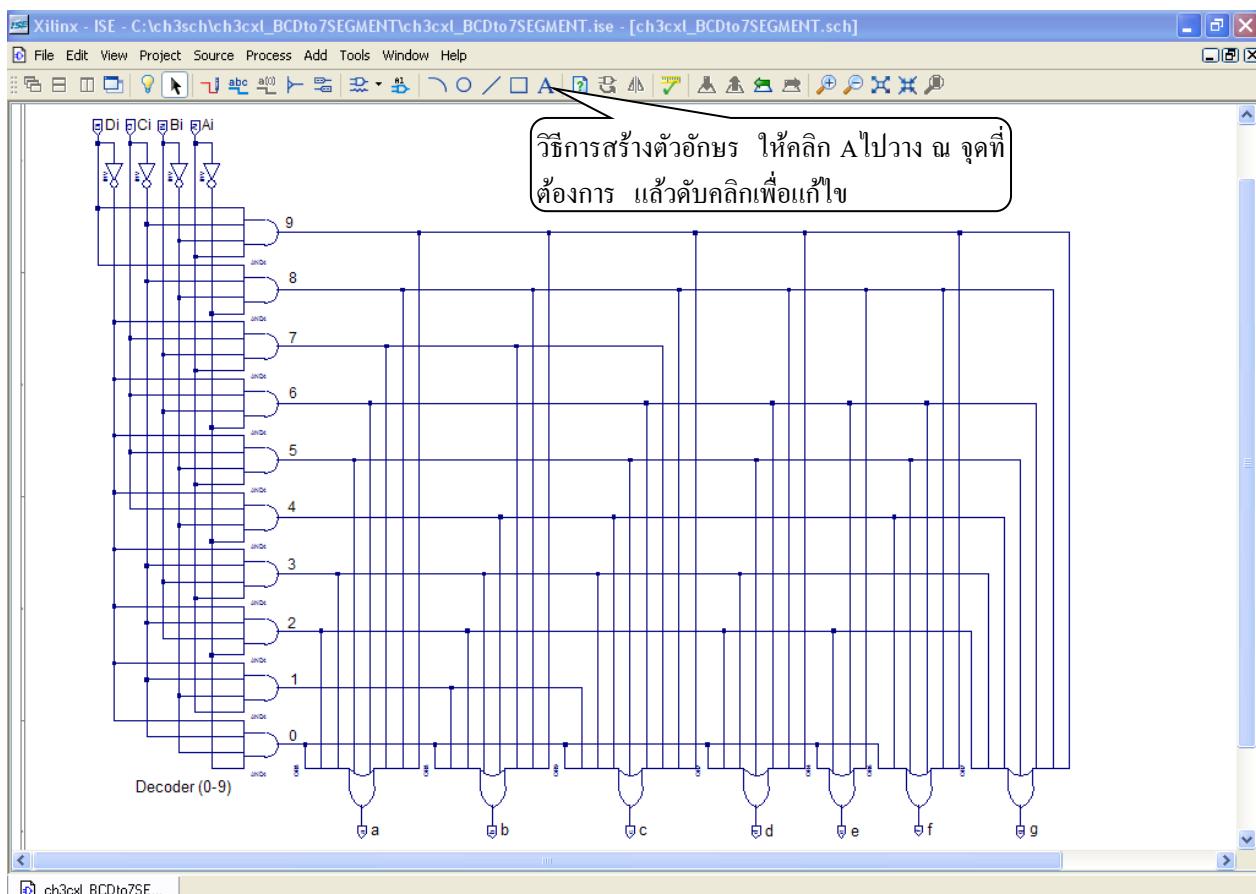
หลังจากโปรแกรมวงจรที่ออกแบบลง FPGA แล้วให้ทดลองกด ปุ่ม PB1-PB3 และให้สังเกตคุณลักษณะ LED L0-L3 ว่าให้ลอกเอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

3.3.2 การออกแบบวงจรอдрหัส BCD เป็นเลขเวนเชกเมนต์

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

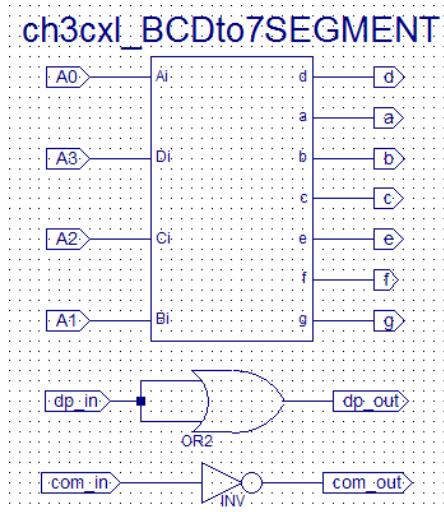
1. สร้างวงจรอдрหัส BCD เป็นเลขเวนเชกเมนต์ด้วย CPLD

นำวงจรอдрหัส BCD เป็นเลขเวนเชกเมนต์แบบแบนโคดร่วมในรูปที่ 3.39 ไปคาดผังวงจรโดยทำตามขั้นตอนในบทที่ 2 เพื่อทำเป็น Symbols ก่อน โดยสร้าง Symbols ใน Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ch3cxl_BCDto7SEGMENT เสร็จแล้วจะได้ดังรูปที่ 3.42 ถ้าพื้นที่วงจรเล็กไป ให้คลิกขวามาสับริเวณที่ว่างในพื้นที่ว่าง ผังวงจรแล้วคลิกเลือกที่ Object Properties แล้วคลิกเลือกขนาดพื้นที่วงจรที่ต้องการ



รูปที่ 3.42 ผังวงจรอдрหัส BCD เป็นเลขเวนเชกเมนต์แบบแบนโคดร่วมที่ใช้ทำ Symbols

จากนั้นคาดผังวงจรทดสอบของวงจรอдрหัส BCD เป็นเลขเวนเชกเมนต์แบบแบนโคดร่วมแสดงดังรูปที่ 3.43 โดยจะใช้ Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_3_2cxl ซึ่งวงจรทดสอบนี้จะมีขาของจุด (dp) และขาร่วม (Common) เพิ่มเข้ามา



รูปที่ 3.43 วงจรทดสอบวงจรอдрหัส BCD เป็นเซเว่นเซกเมนต์แบบแคล็อกคร่วม

หมายเหตุ ในการสร้าง Symbols นั้นแม้จะสร้างจากไฟล์เดียวกัน แต่รูป Symbols ที่ได้อาจจะมีตำแหน่ง I/O ไม่ตรงกัน

การกำหนดขาสัญญาณต่างๆ ใช้ PB1-PB6 เป็นอินพุต มีตัวแสดงผลหลักที่ 1 คือ DIGIT1 เป็นเอาต์พุต กล่าวคือ

A3 = PB1 = INPUT = p39	a = a = OUTPUT = p27	f = f = OUTPUT = p20
A2 = PB2 = INPUT = p40	b = b = OUTPUT = p26	g = g = OUTPUT = p18
A1 = PB3(Slide SW1) = INPUT = p42	c = c = OUTPUT = p25	dp_out = dp = OUTPUT = p19
A0 = PB4(Slide SW2) = INPUT = p43	d = d = OUTPUT = p24	com_out = DIGIT1 = OUTPUT = p34
dp_in = PB5(Slide SW3) = INPUT = p44	e = e = OUTPUT = p22	
com_in = PB6(Slide SW4) = INPUT = p1		

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "a" LOC = "P27" | SLEW = SLOW ;
NET "AO" LOC = "p43" ;
NET "A1" LOC = "p42" ;
NET "A2" LOC = "p40" ;
NET "A3" LOC = "p39" ;
NET "b" LOC = "P26" | SLEW = SLOW ;
NET "c" LOC = "P25" | SLEW = SLOW ;
NET "com_in" LOC = "p1" ;
NET "com_out" LOC = "P34" | SLEW = SLOW ;
NET "d" LOC = "P24" | SLEW = SLOW ;
NET "dp_in" LOC = "P44" ;
NET "dp_out" LOC = "P19" | SLEW = SLOW ;
NET "e" LOC = "P22" | SLEW = SLOW ;
NET "f" LOC = "P20" | SLEW = SLOW ;
NET "g" LOC = "P18" | SLEW = SLOW ;

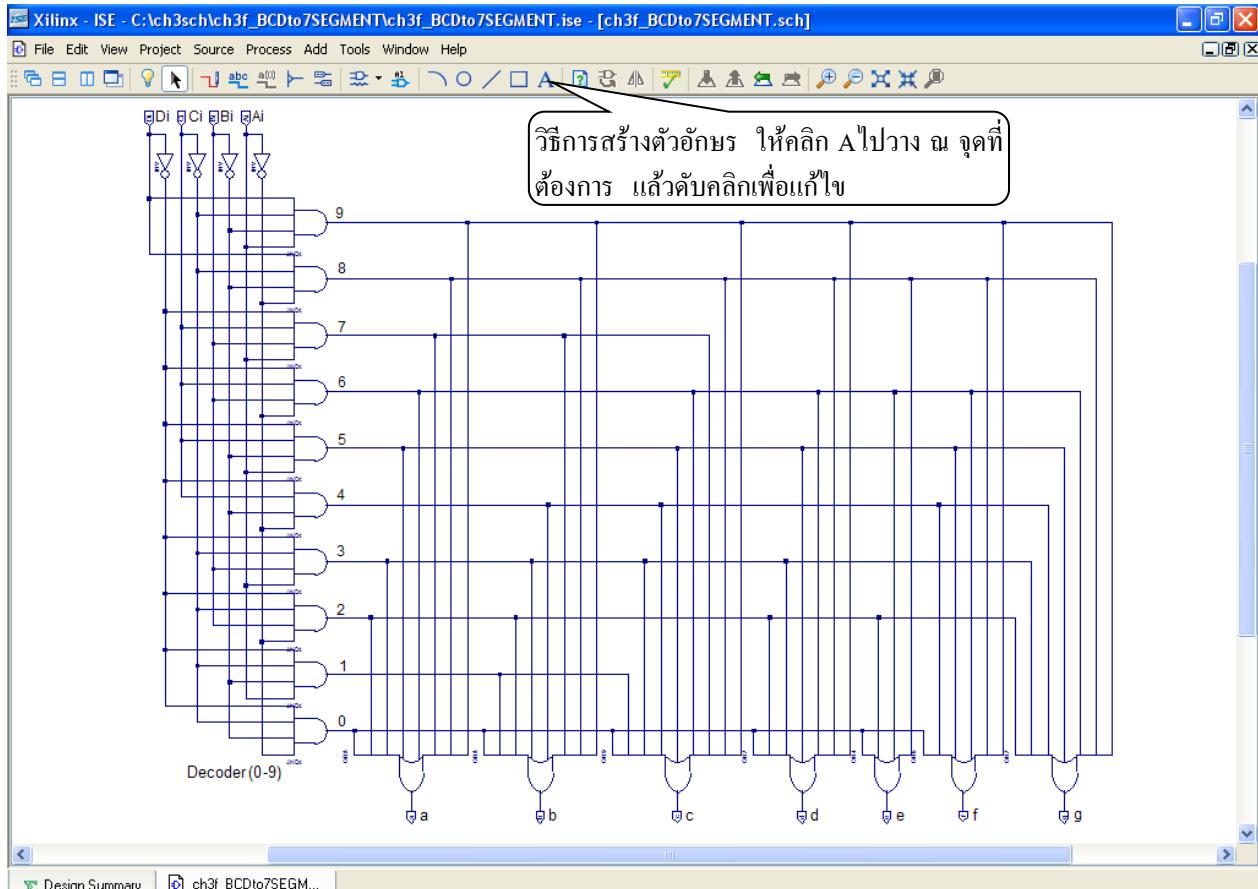
```

หลังจากโปรแกรม CPLD แล้วให้ทดสอบกับปุ่ม PB1-PB6 แล้วคุณที่ตัวแสดงผลเซเว่นเซกเมนต์ว่าให้อเอต์พุตเป็นไปตามที่คุณต้องการหรือไม่ จากนั้นบันทึกผลการทดสอบ (ตัวเซเว่นเซกเมนต์จะดับหมดเมื่อแสดงผลตัวเลข 10-15 หรือ 1010-1111)

2. สร้างวงจรอдрหัส BCD เป็นเซเว่นเซกเมนต์ด้วย FPGA

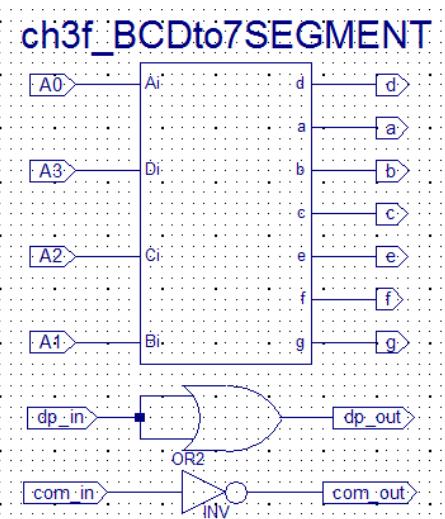
นำวงจรอдрหัส BCD เป็นเซเว่นเซกเมนต์แบบแคล็อกคร่วมในรูปที่ 3.39 ไปคาดผังวงจร โดยทำตามขั้นตอนที่อธิบายในบทที่ 2 เพื่อทำเป็น Symbols โดยสร้างไฟล์ใน Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File

ข้อ ch3f_BCDto7SEGMENT ตามขั้นตอนในบทที่ 2 เสร็จแล้วจะได้ดังรูปที่ 3.44 ถ้าพื้นที่ว่างผังวงจรเล็กไป ให้คลิกขวามาส์บริเวณที่ว่างในพื้นที่ว่างผังวงจรแล้วคลิกเลือกที่ Object Properties แล้วคลิกเลือกขนาดพื้นที่ว่างผังวงจรที่ต้องการ



รูปที่ 3.44 ผังวงจรดอรหัส BCD เป็นเซเวนเซกเม้นต์แบบแคนโอดร่วมที่ใช้ทำ Symbols

จากนั้นวัดผังวงจรดอรหัส BCD เป็นเซเวนเซกเม้นต์แบบแคนโอดร่วมดังรูปที่ 3.45 โดยใช้ Project Location ข้อ ch3sch และกำหนด Project Name และ Source File ข้อ ex3_3_2f



รูปที่ 3.45 วงจรดอรหัส BCD เป็นเซเวนเซกเม้นต์แบบแคนโอดร่วม

หมายเหตุ ในการสร้าง Symbols นั้นแม้จะสร้างจากไฟล์เดียวกัน แต่รูป Symbols ที่ได้อาจจะมีตำแหน่ง I/O ไม่ตรงกัน

การกำหนดขาสัญญาณต่างๆ ใช้ PB1-PB6 และ Dip SW1 เป็นอินพุต ตัวแสดงผลหลักที่ 1 คือ DIGIT1 เป็นเอาต์พุต กล่าวคือ

A3 = PB1 = INPUT = p44	a = a = OUTPUT = p40	f = f = OUTPUT = p25
A2 = PB2 = INPUT = p46	b = b = OUTPUT = p35	g = g = OUTPUT = p23
A1 = PB3 = INPUT = p47	c = c = OUTPUT = p32	dp_out = dp = OUTPUT = p20
A0 = PB4 = INPUT = p50	d = d = OUTPUT = p30	com_in = Dip SW1 = INPUT = p52
dp_in = PB5 = INPUT = p51	e = e = OUTPUT = p27	
com_out = DIGIT1 = OUTPUT = p31		

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "a" LOC = "p40" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "AO" LOC = "p50" | IOSTANDARD = LVCMOS33 ;
NET "A1" LOC = "p47" | IOSTANDARD = LVCMOS33 ;
NET "A2" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "A3" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "b" LOC = "p35" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "c" LOC = "p32" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "com_in" LOC = "p52" | IOSTANDARD = LVCMOS33 ;
NET "com_out" LOC = "p31" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "d" LOC = "p30" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "dp_in" LOC = "p51" | IOSTANDARD = LVCMOS33 ;
NET "dp_out" LOC = "p20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "e" LOC = "p27" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "f" LOC = "p25" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "g" LOC = "p23" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;

```

หลังจากโปรแกรมลง FPGA แล้วให้ทดสอบกับปุ่ม PB1 ถึง PB5 และ Dip SW1 แล้วสังเกตดูผลที่ตัวแสดงผลว่ามีส่วนติดต่อว่างโดยใช้ห้องอิกิเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ กรณีนั้นบันทึกผลการทดลอง (ตัวแสดงผลจะดับหมดเมื่อแสดงผลตัวเลข 10-15 หรือ 1010-1111)

3.4 วงจรเปรียบเทียบ

วงจรเปรียบเทียบจัดเป็นวงจรคอมบินेशันที่ใช้ในการเปรียบเทียบเลขไบนารี 2 จำนวน ตารางความจริงสำหรับวงจรเปรียบเทียบขนาด 1 บิตแสดงดังรูปที่ 3.46

A	B	$G = A > B$	$L = A < B$
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

รูปที่ 3.46 ตารางความจริงสำหรับวงจรเปรียบเทียบขนาด 1 บิต

การสังเคราะห์วงจร

จากตารางความจริงรูปที่ 3.46 สามารถเขียนสมการ SOP ได้ดังนี้

$$G = A \cdot \overline{B}$$

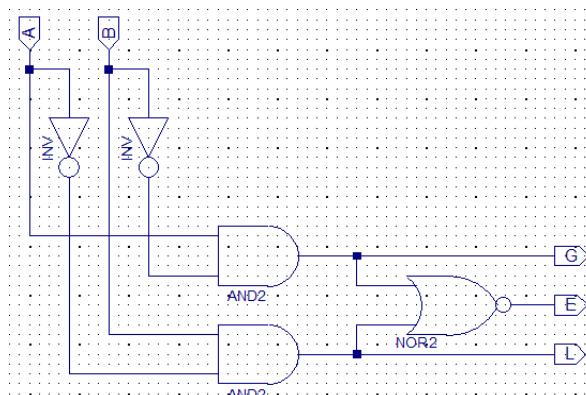
$$L = \overline{A} \cdot B$$

ในกรณีที่ $A = B$ จะได้อาต์พุต $G = '0'$ และ $L = '0'$ ถ้าให้อาต์พุตเป็น E แทนกรณีที่ $A = B$ จะได้

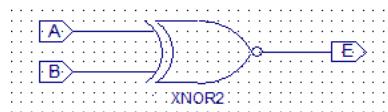
$$E = \overline{L + G}$$

$$E = \overline{\overline{A} \cdot B + A \cdot \overline{B}} = \overline{A \oplus B}$$

และจากสมการนี้สามารถนำมาเขียนเป็นวงจรเปรียบเทียบได้ดังรูปที่ 3.47



(a) วงจรเปรียบเทียบมากกว่า น้อยกว่า และเท่ากัน



(b) วงจรเปรียบเทียบท่ากัน

รูปที่ 3.47 วงจรเปรียบเทียบขนาด 1 บิต

3.4.1 การออกแบบวงจรเบรี่ยมเที่ยบ

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรเบรี่ยมเที่ยบขนาด 1 บิตด้วย CPLD

การทดลองนี้ใช้ Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_4_1cxl จากนั้นทำการคาดผังวงจรดังรูปที่ 3.47

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1 และ PB2 เป็นอินพุต มี LED1-LED3 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{lll} A = PB1 = \text{INPUT} = p39 & G = \text{LED1} = \text{OUTPUT} = p38 & L = \text{LED3} = \text{OUTPUT} = p36 \\ B = PB2 = \text{INPUT} = p40 & E = \text{LED2} = \text{OUTPUT} = p37 & \end{array}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "A" LOC = "p39" ;
NET "B" LOC = "p40" ;
NET "E" LOC = "p37" | SLEW = SLOW ;
NET "G" LOC = "p38" | SLEW = SLOW ;
NET "L" LOC = "p36" | SLEW = SLOW ;
```

หลังจากโปรแกรมวงจรที่ออกแบบลง CPLD แล้วให้ทดลองกดปุ่ม PB1-PB2 และให้สังเกตคุณลักษณะของ LED1-LED3 ว่าให้ล้อจิกเอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

2. สร้างวงจรเบรี่ยมเที่ยบขนาด 1 บิตด้วย FPGA

การทดลองนี้ใช้ Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_4_1f จากนั้นทำการคาดผังวงจรดังรูปที่ 3.47

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1 และ PB2 เป็นอินพุต มี LED L1-L3 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{lll} A = PB1 = \text{INPUT} = p44 & G = L3 = \text{OUTPUT} = p76 & L = L1 = \text{OUTPUT} = p77 \\ B = PB2 = \text{INPUT} = p46 & E = L2 = \text{OUTPUT} = p69 & \end{array}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "A" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "B" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "E" LOC = "p69" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "G" LOC = "p76" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "L" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
```

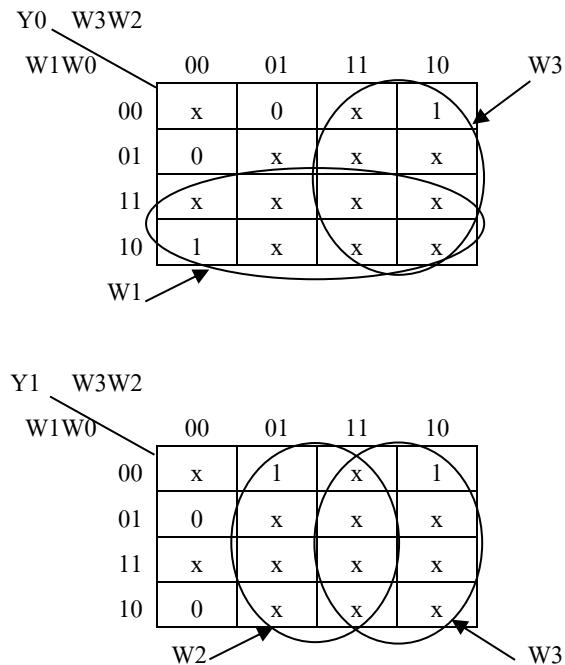
หลังจากโปรแกรมวงจรที่ออกแบบลง FPGA แล้วให้ทดลองกดปุ่ม PB1-PB2 และให้สังเกตคุณลักษณะของ LED L1-L3 ว่าให้ล้อจิกเอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

3.5 วงจรเข้ารหัส

วงจรเข้ารหัสเป็นวงจรคอมบินेशันที่ใช้ทำหน้าที่ตรงข้ามกับวงจรอคลอคหรือหัส เช่น ตัวเข้ารหัสสวิตช์ (Switch encoder) ที่ใช้ในการเปลี่ยนสัญญาณalog ที่ได้จากการกดปุ่มของสวิตช์ ณ ตำแหน่งต่างๆเพื่อแปลงเป็นรหัสเลขไบนาเรีย รูปที่ 3.48 แสดงตารางความจริงของวงจร 4 to 2 Binary encode โดยมีอินพุตเป็น W0 ,W1 ,W2 , W3 และเอาต์พุตคือ Y0 และ Y1

การสังเคราะห์วงจร

Input				Output	
W3	W2	W1	W0	Y1	Y0
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	x	x
0	1	0	0	1	0
0	1	0	1	x	x
0	1	1	0	x	x
0	1	1	1	x	x
1	0	0	0	1	1
1	0	0	1	x	x
1	0	1	0	x	x
1	0	1	1	x	x
1	1	0	0	x	x
1	1	0	1	x	x
1	1	1	0	x	x
1	1	1	1	x	x



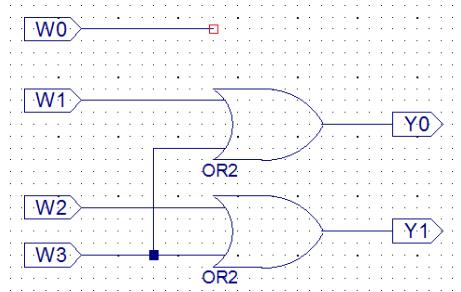
รูปที่ 3.47 ตารางความจริงของวงจร 4 to 2 Binary encode (x = Don't care คือเป็น 0 หรือ 1 ก็ได้)

เมื่อใช้ K-map มาทำการลดรูปสมการที่ได้จากตารางความจริงในรูปที่ 3.48 จะได้สมการบูลีนอย่างง่ายคือ

$$Y_0 = W_1 + W_3$$

$$Y_1 = W_2 + W_3$$

จากการบูลีนสามารถนำมาเขียนวงจรได้ดังรูปที่ 3.49 ซึ่งไม่ขึ้นกับ W0 แต่อย่างใด แต่อย่างไรก็ตามเมื่อคลิกสวิตช์พร้อมกันมากกว่า 1 อินพุตจะทำให้การเข้ารหัสผิดพลาด ดังนั้นจึงมีการคิดคืน Priority Encoder เพื่อแก้ไขข้อบกพร่องดังกล่าว



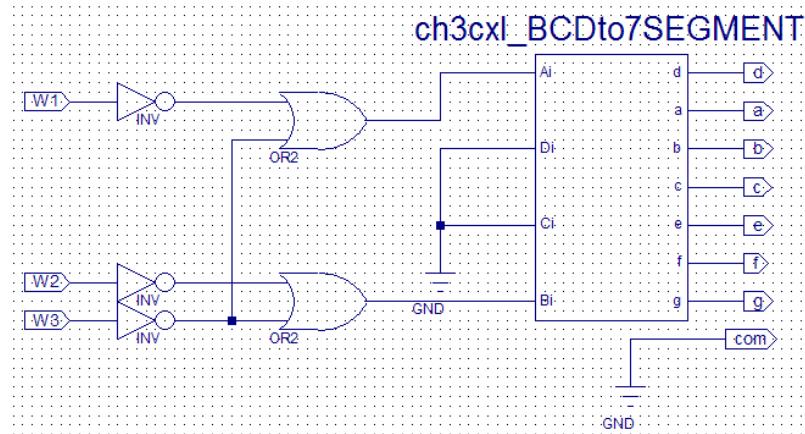
รูปที่ 3.49 แสดงวงจร 4 to 2 Binary Encoder

3.5.1 การออกแบบวงจรเข้ารหัส

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจร 4 to 2 Binary Encoder ด้วย CPLD

การทดลองนี้ใช้ Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_5_1cxl จากนั้นทำการดาวน์โหลดรังรูปที่ 3.50 ซึ่งที่อินพุตทุกตัวจะต่อผ่านอินเวอร์เตอร์เพื่อให้สเมือนว่าสวิตช์เป็นแบบ Active high



รูปที่ 3.50 วงจร 4 to 2 Binary Encoder

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB4 เป็นอินพุต มีเซเวนเซกเมนต์ (DIGIT1) เป็นเอาต์พุต กล่าวคือ

$W_0 = PB_1 = \text{INPUT} = p39$ (ไม่ต้องกำหนดขา)	$a = a = \text{OUTPUT} = p27$	$e = e = \text{OUTPUT} = p22$
$W_1 = PB_2 = \text{INPUT} = p40$	$b = b = \text{OUTPUT} = p26$	$f = f = \text{OUTPUT} = p20$
$W_2 = PB_3(\text{Slide SW1}) = \text{INPUT} = p42$	$c = c = \text{OUTPUT} = p25$	$g = g = \text{OUTPUT} = p18$
$W_3 = PB_4(\text{Slide SW2}) = \text{INPUT} = p43$	$d = d = \text{OUTPUT} = p24$	$\text{com} = \text{DIGIT1} = \text{OUTPUT} = p34$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังรูปที่ 3.51 หลังจากโปรแกรมจะเรียกออกแบบลงชิป CPLD และให้ทดลองกด ปุ่ม PB1-PB4 และให้สังเกตคุณลักษณะที่ตัวแสดงผล เซเวนเซกเมนต์ว่ามีส่วนติดสว่าง โดยให้ออกเจกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นให้ทดลองกด W1 และ W2 พร้อมกันแล้วที่ยินผลการทดลองกับการกด W3 เพียงปุ่มเดียวว่าให้ผลเป็นอย่างไร และจึงบันทึกผลการทดลอง

```

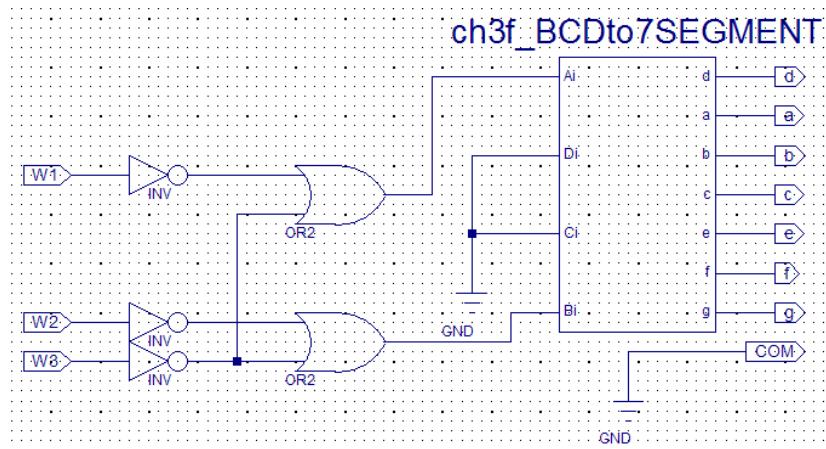
NET "a" LOC = "p27" | SLEW = FAST ;
NET "b" LOC = "p26" | SLEW = SLOW ;
NET "c" LOC = "p25" | SLEW = SLOW ;
NET "com" LOC = "p34" | SLEW = SLOW ;
NET "d" LOC = "p24" | SLEW = SLOW ;
NET "e" LOC = "p22" | SLEW = SLOW ;
NET "f" LOC = "p20" | SLEW = SLOW ;
NET "g" LOC = "p18" | SLEW = SLOW ;
NET "W1" LOC = "p40" ;
NET "W2" LOC = "p42" ;
NET "W3" LOC = "p43" ;

```

รูปที่ 3.51 พิมพ์ใน Edit Constraints (Text)

2. สร้างวงจร 4 to 2 Binary Encoder ด้วย FPGA

การทดลองนี้ใช้ Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_5_1f จากนั้นทำการวัดผังวงจรดังรูปที่ 3.52



รูปที่ 3.52 4 to 2 Binary Encoder

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB4 เป็นอินพุต มีช่วงเวลาเชคเม้นต์ (DIGIT1) เป็นเวลาพุต กล่าวคือ

$W_0 = PB1 = INPUT = p44$ (ไม่ต้องกำหนดขา)	$a = a = OUTPUT = p40$	$e = e = OUTPUT = p27$
$W_1 = PB2 = INPUT = p46$	$b = b = OUTPUT = p35$	$f = f = OUTPUT = p25$
$W_2 = PB3 = INPUT = p47$	$c = c = OUTPUT = p32$	$g = g = OUTPUT = p23$
$W_3 = PB4 = INPUT = p50$	$d = d = OUTPUT = p30$	$com = DIGIT1 = OUTPUT = p31$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

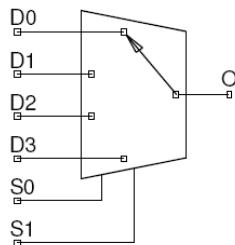
NET "a" LOC = "p40" | IOSTANDARD = LVCMS33 | SLEW = SLOW ;
NET "b" LOC = "p35" | IOSTANDARD = LVCMS33 | SLEW = SLOW ;
NET "c" LOC = "p32" | IOSTANDARD = LVCMS33 | SLEW = SLOW ;
NET "COM" LOC = "p31" | IOSTANDARD = LVCMS33 | SLEW = SLOW ;
NET "d" LOC = "p30" | IOSTANDARD = LVCMS33 | SLEW = SLOW ;
NET "e" LOC = "p27" | IOSTANDARD = LVCMS33 | SLEW = SLOW ;
NET "f" LOC = "p25" | IOSTANDARD = LVCMS33 | SLEW = SLOW ;
NET "g" LOC = "p23" | IOSTANDARD = LVCMS33 | SLEW = SLOW ;
NET "W1" LOC = "p46" | IOSTANDARD = LVCMS33 ;
NET "W2" LOC = "p47" | IOSTANDARD = LVCMS33 ;
NET "W3" LOC = "p50" | IOSTANDARD = LVCMS33 ;

```

หลังจากโปรแกรมลง FPGA และให้กดปุ่ม PB1-PB4 และคุณลักษณะตัวแสดงผลว่าให้อาต์พุตเป็นไปตามทฤษฎีหรือไม่ างานนี้ให้กด W1 และ W2 พร้อมกันแล้วเท่านั้นกับการกด W3 เพียงปุ่มเดียวว่าเป็นอย่างไร างานนี้บันทึกผลการทดลอง

3.6 วงจรแมติกเล็กเชอร์

วงจรแมติกเล็กเชอร์หรือ MUX เป็นวงจรคอมบินेशันที่ใช้ในการเลือกสัญญาณจากอินพุตเพื่อส่งออกไปที่เอาต์พุตครึ่ง ละ 1 อินพุต รูปที่ 3.53 แสดงหลักการทำงานอย่างง่ายของแมติกเล็กเชอร์ขนาด 1 บิตแบบ 4 อินพุต 1 เอาต์พุต ซึ่งสามารถเลือก อินพุต D0–D3 ไปออกที่เอาต์พุต O ครั้งละ 1 อินพุต โดยใช้สัญญาณ Control S0 และ S1 เป็นตัวกำหนดว่าจะเลือกอินพุตใด



รูปที่ 3.53 แมติกเล็กเชอร์ขนาด 1 บิตแบบ 4 อินพุต 1 เอาต์พุต

การสังเคราะห์วงจร

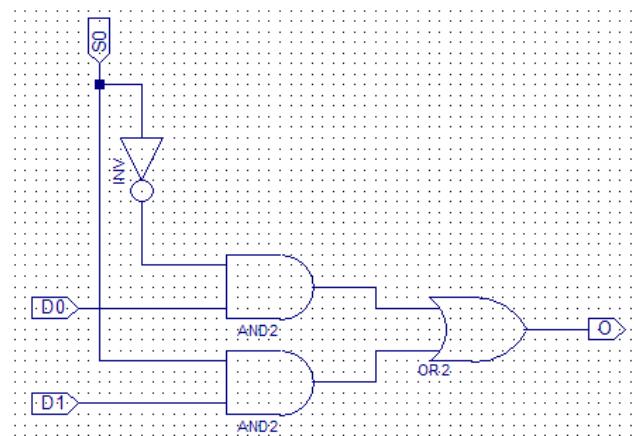
วงจรแมติกเล็กเชอร์แบบ 2 อินพุต 1 เอาต์พุต (2 to 1 Multiplexer) และแบบ 4 อินพุต 1 เอาต์พุต (4 to 1 Multiplexer) แสดงดังรูปที่ 3.54 และรูปที่ 3.55 จากตารางความจริงในรูปที่ 3.54(a) และรูปที่ 3.55(a) สามารถเขียนเป็นสมการบูลีนได้ดังนี้

$$\text{กรณี 2 to 1 Multiplexer} \quad O = \overline{S_0} \cdot D_0 + S_0 \cdot D_1$$

$$\text{และ 4 to 1 Multiplexer} \quad O = \overline{S_1} \cdot \overline{S_0} \cdot D_0 + \overline{S_1} \cdot S_0 \cdot D_1 + S_1 \cdot \overline{S_0} \cdot D_2 + S_1 \cdot S_0 \cdot D_3$$

Control S0	Output O	Remark
0	D0	$O = D0$
1	D1	$O = D1$

(a) ตารางความจริงวงจรแมติเพล็กเซอร์แบบ 2 อินพุต 1 เอาต์พุต

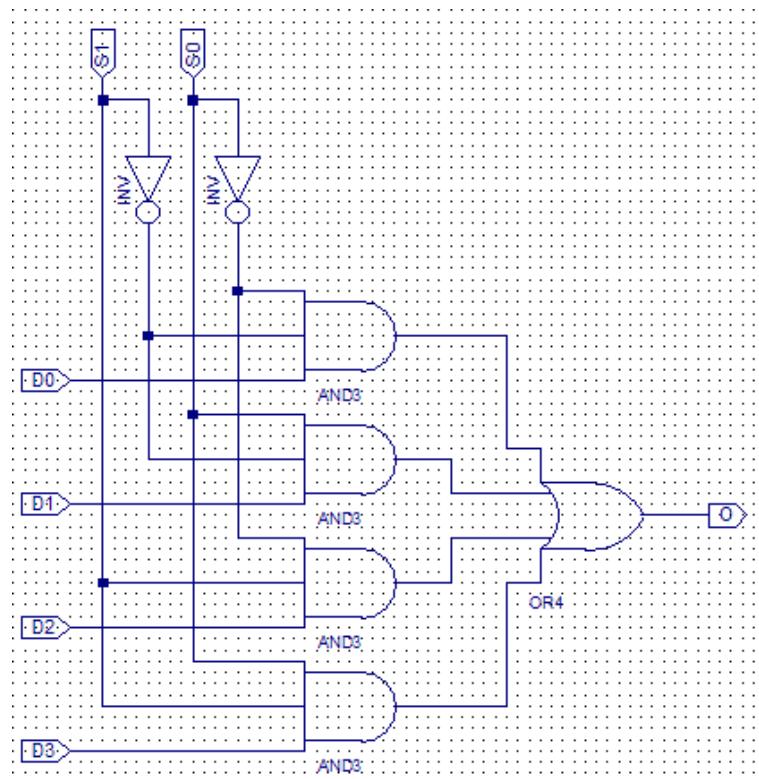


(b) ตารางผังวงจรแมติเพล็กเซอร์แบบ 2 อินพุต 1 เอาต์พุต

รูปที่ 3.54 ตารางความจริงและผังวงจรแมติเพล็กเซอร์แบบ 2 อินพุต 1 เอาต์พุต

Control		Output	Remark
S1	S0	O	
0	0	D0	$O = D0$
0	1	D1	$O = D1$
1	0	D2	$O = D2$
1	1	D3	$O = D3$

(a) ตารางความจริงของมัลติเพล็กเซอร์แบบ 4 อินพุต 1 เอาต์พุต



(b) ผังวงจร มัลติเพล็กเซอร์แบบ 4 อินพุต 1 เอาต์พุต

รูปที่ 3.55 ตารางความจริงและผังวงจร มัลติเพล็กเซอร์แบบ 4 อินพุต 1 เอาต์พุต

3.6.1 การออกแบบวงจรแมตติเพลสิกเซอร์แบบ 2 อินพุต 1 เอาต์พุต

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรแมตติเพลสิกเซอร์แบบ 2 อินพุต 1 เอาต์พุต (2 to 1 Multiplexer) ด้วย CPLD

สร้างไฟล์ใน Project Location ชื่อ ch3sch แล้วกำหนด Project Name และ Source File ชื่อ ex3_6_1cxl จากนั้นทำการวดผังวงจรดังรูปที่ 3.54b)

การกำหนดขาสัญญาณต่างๆจะใช้ปุ่มกด PB1-PB2 และ PB3 (Slide SW1) เป็นอินพุต มี LED1 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{ll} D0 = PB1 = p39 & O = LED1 = p38 \\ D1 = PB2 = p40 & S0 = PB3 (\text{Slide SW1}) = p42 \end{array}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "D0" LOC = "p39" ;
NET "D1" LOC = "p40" ;
NET "O" LOC = "p38" | SLEW = SLOW ;
NET "S0" LOC = "p42" ;
```

หลังจากโปรแกรมจะที่ออกแบบลงชิป CPLD แล้วให้ทดลองกดปุ่ม PB1-PB2 และ PB3 (Slide SW1) และให้สังเกต คุณลักษณะของ LED1 ว่าติดสว่างโดยไห้ล้อจิกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

2. สร้างวงจรแมตติเพลสิกเซอร์แบบ 2 อินพุต 1 เอาต์พุต (2 to 1 Multiplexer) ด้วย FPGA

สร้างไฟล์ใน Project Location ชื่อ ch3sch แล้วกำหนด Project Name และ Source File ชื่อ ex3_6_1f จากนั้นทำการวดผังวงจรดังรูปที่ 3.54b)

การกำหนดขาสัญญาณต่างๆจะใช้ปุ่มกด PB1, PB2 และ Dip SW1 เป็นอินพุต มี LED L3 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{ll} D0 = PB1 = \text{INPUT} = p44 & O = L3 = \text{INPUT} = p76 \\ D1 = PB2 = \text{INPUT} = p46 & S0 = \text{Dip SW1} = \text{OUTPUT} = p52 \end{array}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "D0" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "D1" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "O" LOC = "p76" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "S0" LOC = "p52" | IOSTANDARD = LVCMOS33 ;
```

หลังจากโปรแกรมจะที่ออกแบบลง FPGA แล้วให้ทดลองกดปุ่ม PB1-PB2 และ Dip SW1 ให้สังเกตคุณลักษณะของ LED L3 ว่าติดสว่างโดยไห้ล้อจิกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

3.6.2 การออกแบบวงจรแมตติเพล็กเซอร์แบบ 4 อินพุต 1 เอาต์พุต

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรแมตติเพล็กเซอร์แบบ 4 อินพุต 1 เอาต์พุต (4 to 1 Multiplexer) ด้วย CPLD

สร้างไฟล์ใน Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_6_2cxl จากนั้นทำการวางแผนจรดังรูปที่ 3.55(b) การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1 ถึง PB6 เป็นอินพุต มี LED1 เป็นเอาต์พุต กล่าวคือ

D0 = PB1 = INPUT = p39	S1 = PB5 (Slide SW3) = INPUT = p44
D1 = PB2 = INPUT = p40	S0 = PB6 (Slide SW4) = INPUT = p1
D2 = PB3 (Slide SW1) = INPUT = p42	O = LED1 = OUTPUT = p38
D3 = PB4 (Slide SW2) = INPUT = p43	

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "D0" LOC = "P39" ;
NET "D1" LOC = "P40" ;
NET "D2" LOC = "P42" ;
NET "D3" LOC = "P43" ;
NET "O" LOC = "P38" | SLEW = SLOW ;
NET "S0" LOC = "P1" ;
NET "S1" LOC = "P44" ;

```

หลังจากโปรแกรมจะที่ออกแบบลง CPLD แล้วให้ทดลองกดปุ่ม PB1-PB4, PB5 (Slide SW3) และ PB6 (Slide SW4) ให้สังเกตผลลัพธ์ LED1 ว่าติดสว่างโดยไฟลอกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ หากนั้นจึงบันทึกผลการทดลอง

2. สร้างวงจรแมตติเพล็กเซอร์แบบ 4 อินพุต 1 เอาต์พุต (4 to 1 Multiplexer) ด้วย FPGA

สร้างไฟล์ใน Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_6_2f จากนั้นทำการวางแผนจรดังรูปที่ 3.55(b) การกำหนดขาใช้ PB1-PB4 และ Dip SW1-Dip SW2 เป็นอินพุต มี LED L3 เป็นเอาต์พุต กล่าวคือ

D0 = PB1 = INPUT = p44	S1 = Dip SW1 = INPUT = p52
D1 = PB2 = INPUT = p46	S0 = Dip SW2 = INPUT = p53
D2 = PB3 = INPUT = p47	O = L3 = OUTPUT = p76
D3 = PB4 = INPUT = p50	

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

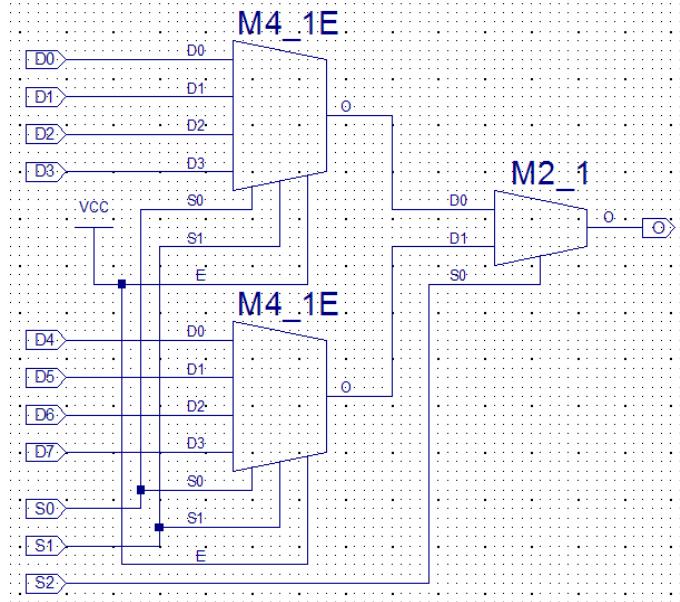
```

NET "D0" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "D1" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "D2" LOC = "p47" | IOSTANDARD = LVCMOS33 ;
NET "D3" LOC = "p50" | IOSTANDARD = LVCMOS33 ;
NET "O" LOC = "p76" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "S0" LOC = "p53" | IOSTANDARD = LVCMOS33 ;
NET "S1" LOC = "p52" | IOSTANDARD = LVCMOS33 ;

```

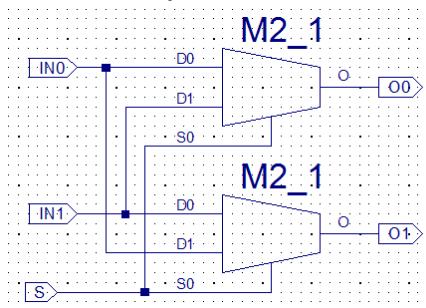
หลังจากโปรแกรมจะที่ออกแบบลงชิป FPGA แล้วให้ทดลองกด ปุ่ม PB1-PB4 , Dip SW1 และ Dip SW2 ให้สังเกต คุณลักษณะของ LED L3 ว่าติดสว่างโดยไห้ล้อจิกເອຕີພຸດເປັນໄປຕາມທຸກໆຢູ່ໃໝ່ ຈະຈົບຕິດຕັ້ງກຳທີ່ກຳລົງ

ถ้าทดลองออกแบบวงจรມັດດີເພື່ອເຊື່ອຮັບນາດ 1 ບີຕ 8 ອິນພຸດ 1 ເອຕີພຸດ (1 Bit 8 to 1 Multiplexer) ໂດຍໃຫ້ Symbol ຊື່ “m2_1” ແລະ “m4_1e” ທີ່ອີ່ມໃນ Categories ຊື່ “MUX” ຈະໄດ້ວັງຈາກຮັບຮູບປັບທີ່ 3.56



ຮູບປັບທີ່ 3.56 ວັງຈາກມັດດີເພື່ອເຊື່ອຮັບນາດ 1 ບີຕ 8 ອິນພຸດ 1 ເອຕີພຸດ

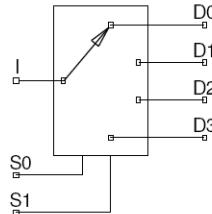
ถ้าทดลองออกแบบวงจร 2x2 Crossbar switch ຈະໄດ້ວັງຈາກຮັບຮູບປັບທີ່ 3.57



ຮູບປັບທີ່ 3.57 ວັງຈາກ 2x2 Crossbar switch

3.7 วงจรดิจิตอลด้วย FPGA และ CPLD ภาคปฏิบัติ โดยใช้วิธี Schematic

วงจรดิจิตอลดีมัลติเพล็กเซอร์ (Demultiplexer) หรือ DEMUX เป็นวงจรคอมบินेशันที่ใช้สวิตช์เพื่อกระจายสัญญาณที่เข้ามาทางอินพุตออกไปทางเอต์พุตที่ต้องการดังรูปที่ 3.58 ซึ่งวงจรดีมัลติเพล็กเซอร์แบบ 1 อินพุต 4 เอต์พุต มี I เป็นอินพุต และมี D0-D3 เป็นเอต์พุต โดยมี S0 และ S1 เป็นตัวควบคุมว่าจะให้สัญญาณออกที่เอต์พุตใด โดยมีตารางความจริงแสดงดังรูปที่ 3.61



รูปที่ 3.58 แสดงวงจรดิจิตอลดีมัลติเพล็กเซอร์แบบเข้า 1 อินพุตออก 4 เอต์พุต

Control		Output				Remark
S1	S0	D0	D1	D2	D3	
0	0	I	0	0	0	D0 = I
0	1	0	I	0	0	D1 = I
1	0	0	0	I	0	D2 = I
1	1	0	0	0	I	D3 = I

รูปที่ 3.59 ตารางความจริงของวงจรดิจิตอลดีมัลติเพล็กเซอร์แบบเข้า 1 อินพุตออก 4 เอต์พุต

จากตารางความจริงเพียงสมการ SOP ได้ดังนี้ (พิจารณาเฉพาะช่องที่เอต์พุตเป็น I)

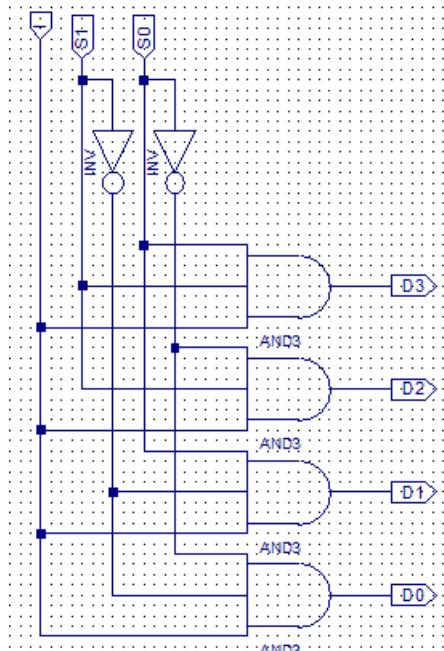
$$D0 = I \cdot \overline{S1} \cdot \overline{S0}$$

$$D1 = I \cdot \overline{S1} \cdot S0$$

$$D2 = I \cdot S1 \cdot \overline{S0}$$

$$D3 = I \cdot S1 \cdot S0$$

จากสมการนี้ เขียนวงจรดิจิตอลดีมัลติเพล็กเซอร์ได้ดังรูปที่ 3.60 ซึ่งเป็นวงจรเดียวกับ 2 to 4 Decoder ในรูปที่ 3.11(a)



รูปที่ 3.60 ผังวงจรดิจิตอลดีมัลติเพล็กเซอร์แบบ 1 อินพุต 4 เอต์พุต

3.7.1 การออกแบบวงจรดิจิตอลด้วย CPLD

อุปกรณ์ที่สามารถเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจร 1 to 4 ดิมัลติเพล็กเซอร์ขนาด 1 บิตด้วย CPLD

ออกแบบวงจร 1 to 4 Demultiplexer ด้วย CPLD ดังรูปที่ 3.60 โดยใช้ Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_7_1cxl และกำหนดขาสัญญาณต่างๆ ดังนี้

S1 = PB1 = INPUT = p39	D0 = LED1 = OUTPUT = p38	D2 = LED3 = OUTPUT = p36
S0 = PB2 = INPUT = p40	D1 = LED2 = OUTPUT = p37	D3 = LED4 = OUTPUT = p35
I = PB3 = INPUT = p42		

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "I" LOC = "P42" ;
NET "S0" LOC = "P40" ;
NET "S1" LOC = "P39" ;
NET "D0" LOC = "P38" | SLEW = SLOW ;
NET "D1" LOC = "P37" | SLEW = SLOW ;
NET "D2" LOC = "P36" | SLEW = SLOW ;
NET "D3" LOC = "P35" | SLEW = SLOW ;
```

เมื่อโปรแกรม CPLD และ ให้กดปุ่ม PB1-PB3 และคุณลักษณะ LED1-LED4 ว่าตามทฤษฎีหรือไม่ บันทึกผลการทดลอง

2. สร้างวงจร 1 to 4 ดิมัลติเพล็กเซอร์ขนาด 1 บิตด้วย FPGA

ออกแบบวงจร 1 to 4 Demultiplexer ด้วย FPGA ดังรูปที่ 3.60 โดยใช้ Project Location ชื่อ ch3sch และกำหนด Project Name และ Source File ชื่อ ex3_7_1f และกำหนดขาสัญญาณต่างๆ ดังนี้

S1 = PB1 = INPUT = p44	D0 = L3 = OUTPUT = p76	D2 = L1 = OUTPUT = p77
S0 = PB2 = INPUT = p46	D1 = L2 = OUTPUT = p69	D3 = L0 = OUTPUT = p70
I = PB3 = INPUT = p47		

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "D0" LOC = "p76" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1" LOC = "p69" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D2" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D3" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "I" LOC = "p47" | IOSTANDARD = LVCMOS33 ;
NET "S0" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "S1" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
```

เมื่อโปรแกรม FPGA และ ให้กดปุ่ม PB1-PB3 และคุณลักษณะ LED L0-L3 ว่าเป็นตามทฤษฎีหรือไม่ บันทึกผลการทดลอง

สรุป

การออกแบบวงจรดิจิตอลทั้งที่เป็นล็อกอินก็เกตพื้นฐานและวงจรคอมบินेशันจะเริ่มด้วยการสร้างตารางความจริงเพื่อกำหนดสัญญาณอินพุตและรูปแบบเอาท์พุตที่ต้องการ และวิจัยเป็นสมการบูลีน หลังจากนั้นจึงนั่นสมการบูลีนที่ได้ไปวัดผังวงจรในซอฟต์แวร์ทุก ISE WebPack 8.11i หลังจากนั้นจึงเป็นการกำหนดขาต่างๆ ของวงจรเข้ากับอุปกรณ์อินพุตเอาพุทธองบอร์ดทดลองใน User Constrain File และวิจัยทำการสั่งให้ซอฟต์แวร์ทุลสังเคราะห์สร้างวงจรตามที่ได้ออกแบบไว้ และดาวน์โหลดลง FPGA หรือ CPLD เพื่อทดสอบต่อไป

คำถามท้ายบท

1. ให้ออกแบบวงจร 4 to 16 Decoder โดยใช้ Symbol D4_16E (คำแนะนำ สร้างวงจรควบคุมที่ขา E ให้ทำงานทีละ ชุด)
2. จงสร้างวงจรบวกเลขจำนวนเต็มบวกขนาด 2 บิตที่แสดงผลทางตัวแสดงผลเซกเมนต์
3. หากมีวงจรลอจิกเกตพื้นฐานเพียงแค่แอนด์เกต ออร์เกต และ อินเวอร์เตอร์ จะสร้างเกตพื้นฐานอื่นๆ เช่น แอนด์เกต นอร์เกต เอ็กคูลูซีฟออร์เกตและ เอ็กคูลูซีฟนอร์เกตได้อย่างไร
4. จงสร้างวงจรลบเลขแบบฟลูซัพแทรคเตอร์
5. วงจรคอมทรหัสและวงจรเข้ารหัสแตกต่างกันอย่างไร
6. จะสามารถประยุกต์ใช้งานรเข้ารหัสและคอมทรหัสเพื่อสร้างวงจรแมลติเพล็กเซอร์และวงจรคอมแมลติเพล็กเซอร์ตามลำดับได้อย่างไร
7. จะสามารถตัดแปลงวงจรบวกแบบชาฟเฟอร์ได้ก็ต้ายเป็นวงจรฟูลแอคเดอร์ได้อย่างไร
8. แต่ละสถานะของวงจรไตรستีทบบไฟฟอร์สามารถนำไปประยุกต์ใช้ประโยชน์ได้อย่างไรบ้าง จงยกตัวอย่าง
9. การต่อตัวถังท่านพูลอัพหรือพูลดาวน์มีประโยชน์อย่างไร
10. การจำลองการทำงานแบบ Behavioral คืออย่างไร

== หน้าว่าง ==

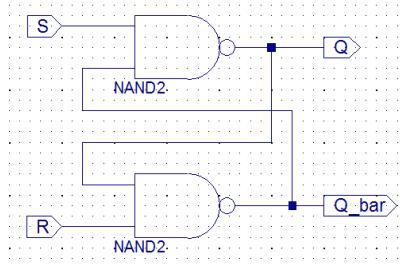
บทที่ 4 แลตช์ พลิปฟลอป และวงจรซีเควนเชียล

กล่าวนำ

วงจรคอมบินेशันเป็นวงจรที่สัญญาณเข้าที่พุทเปลี่ยนตามลักษณะของสัญญาณอินพุตแต่เพียงอย่างเดียวและมีการใช้งานที่หลากหลายในวงจรดิจิตอลโดยทั่วไป แต่อย่างไรก็ดีอย่างมีวงจรดิจิตอลอิกประเทกหนึ่งที่สัญญาณเข้าที่พุทไม่ได้ขึ้นอยู่กับลักษณะของสัญญาณอินพุตแต่เพียงอย่างเดียว แต่ยังขึ้นอยู่กับสถานะปัจจุบันของสัญญาณเข้าที่พุทในช่วงเวลาหนึ่งๆ อีกด้วย นั่นหมายความว่าถึงแม้สัญญาณอินพุตจะมีลักษณะเดียวกัน แต่หากสถานะของสัญญาณเข้าที่พุทไม่ตรงกันในแต่ละช่วงเวลา สัญญาณเข้าที่พุทที่จะเกิดขึ้นก็ไม่จำเป็นต้องเหมือนกันทุกๆ ครั้ง ซึ่งวงจรที่มีคุณลักษณะแบบนี้เรียกว่างจรซีเควนเชียล อันประกอบไปด้วยวงจรแลตช์ พลิปฟลอปและวงจรเค้าท์เตอร์ เป็นต้น

4.1 แลตช์

แลตช์ (Latch) และพลิปฟลอป (Flip-Flop) เป็นหน่วยความจำ (Memory element) พื้นฐาน แต่พลิปฟลอปจะแตกต่างจากแลตช์ตรงที่เอาต์พุตจะเปลี่ยนสถานะเมื่อทริกเกอร์ด้วยขอบ (Edge triggered) ของสัญญาณนาฬิกา (Clock) แลตช์พื้นฐานได้แก่ แลตช์แบบ NAND Gate และแบบ NOR Gate และรูปที่ 4.1 และรูปที่ 4.2 ตามลำดับ โดยที่สถานะ Invalid จะเป็นสถานะที่ Q และ \bar{Q} หรือ \bar{Q} มีสถานะลอกิจเหมือนกัน ซึ่งไม่เป็นจริง เรายังไม่นำสถานะนี้มาใช้ ส่วนสถานะ Hold หรือ No change หรือ Latch จะเป็นสถานะที่วงจรแลตช์คงค่าลอกิจเดิมของเอาต์พุตไว้ ซึ่งอาจจะเป็นลอกิจ '0' หรือ '1' ก็ได้

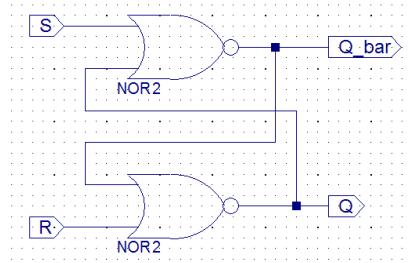


(a) ผังวงจร NAND Gate Latch

Input		Output		Remark
S	R	Q	\bar{Q}	
1	1	$Q(0/1)$	$\bar{Q}(1/0)$	Hold (No change)
0	1	1	0	Set
1	0	0	1	Clear
0	0	1	1	Invalid

(b) ตารางความจริงของ NAND Gate Latch

รูปที่ 4.1 NAND Gate Latch



(a) ผังวงจรของ NOR Gate Latch

Input		Output		Remark
S	R	Q (0/1)	Q_bar (1/0)	
0	0	Q (0/1)	Q_bar (1/0)	Hold (No change)
1	0	1	0	Set
0	1	0	1	Clear
1	1	0	0	Invalid

(b) ตารางความจริงของ NOR Gate Latch

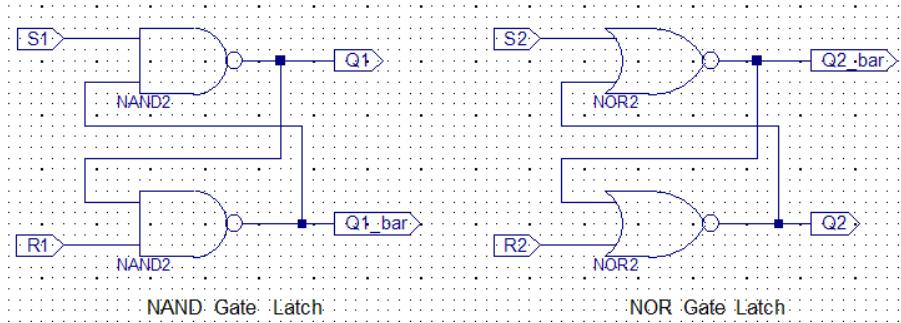
รูปที่ 4.2 NOR Gate Latch

4.1.1 การออกแบบวงจรแล็ช

อุปกรณ์ที่สามารถใช้ทดสอบคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจร NAND Gate latch และ NOR Gate latch ด้วย CPLD

สร้าง Folder ชื่อ ch4sch ในโฟลเดอร์ C จากนั้นออกแบบวงจร NAND Gate latch และ NOR Gate latch ด้วย CPLD โดยมีผังวงจรแสดงดังรูปที่ 4.3 ตามขั้นตอนในบทที่ 2 สร้างไฟล์โดยใช้ Project Location (หรือ Folder) ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_1_1cxl



รูปที่ 4.3 วงจร NAND Gate latch และ NOR Gate latch

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB4 เป็นอินพุต และ LED1-LED4 เป็นเอาต์พุต กล่าวคือ

$$S1 = PB1 = \text{INPUT} = p39$$

$$Q1 = LED1 = \text{OUTPUT} = p38$$

$$R1 = PB2 = \text{INPUT} = p40$$

$$Q1_bar = LED2 = \text{OUTPUT} = p37$$

$$S2 = PB3 = \text{INPUT} = p42$$

$$Q2 = LED3 = \text{OUTPUT} = p36$$

$$R2 = PB4 = \text{INPUT} = p43$$

$$Q2_bar = LED4 = \text{OUTPUT} = p35$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "Q1"      LOC = "p38" | SLEW = SLOW ;
NET "Q1_bar"   LOC = "p37" | SLEW = SLOW ;
NET "Q2"      LOC = "p36" | SLEW = SLOW ;
NET "Q2_bar"   LOC = "p35" | SLEW = SLOW ;
NET "R1"       LOC = "p40" ;
NET "R2"       LOC = "p43" ;
NET "S1"       LOC = "p39" ;
NET "S2"       LOC = "p42" ;

```

หลังจากโปรแกรมลง CPLD แล้ว ให้ทดสอบกับปุ่ม PB1-PB4 และให้สังเกตคุณลักษณะของวงจรที่ LED1-LED4 ว่าให้ล็อกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ สถานะเอาต์พุตของแล็ชที่เป็น “Invalid” มีสถานะเอาต์พุต Q1, Q1_bar, Q2 และ Q2_bar ตรงข้ามกันทุกรูปนี้หรือไม่ จากนั้นบันทึกผลการทดสอบ

2. สร้างวงจร NAND Gate latch และ NOR Gate latch ด้วย FPGA

ออกแบบ NAND Gate latch และ NOR Gate latch ด้วย FPGA ดังรูปที่ 4.3 ตามขั้นตอนในบทที่ 2 โดยใช้ Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_1_1f การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB4 เป็นอินพุต และ LED L0-L3 เป็นเอาต์พุต กล่าวคือ

S1 = PB1 = INPUT = p44	Q1 = L3 = OUTPUT = p76
R1 = PB2 = INPUT = p46	Q1_bar = L2 = OUTPUT = p69
S2 = PB3 = INPUT = p47	Q2 = L1 = OUTPUT = p77
R2 = PB4 = INPUT = p50	Q2_bar = L0 = OUTPUT = p70

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "Q1"      LOC = "p76" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q1_bar"   LOC = "p69" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q2"      LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q2_bar"   LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "R1"       LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "R2"       LOC = "p50" | IOSTANDARD = LVCMOS33 ;
NET "S1"       LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "S2"       LOC = "p47" | IOSTANDARD = LVCMOS33 ;

```

หลังจากโปรแกรมลง FPGA แล้วให้ทดลองกดปุ่ม PB1-PB4 และให้สังเกตคุณลักษณะ LED L0-L3 ว่าให้คล้อยอ่าต์พุตเป็นไปตามทฤษฎีหรือไม่ สถานะเอาต์พุตของแล็คช์ที่เป็น “Invalid” มีสถานะเอาต์พุต Q1, Q1_bar, Q2 และ Q2_bar ตรงข้ามกันทุกรقمหรือไม่ จอกนั้นบันทึกผลการทดลอง

4.2 พลิปฟล็อป

4.2.1 การออกแบบวงจร D Flip-Flop

D Flip-Flop แต่ละชนิดแสดงดังรูปที่ 4.4 ถึงรูปที่ 4.6 โดยมี C เป็นขาสัญญาณนาฬิกา (Clock) ที่ทริกคัวบของขา (ขอบขาขึ้นหรือ Positive edge-triggered) หลักการทำงานของ D Flip-Flop (หรือ FD) ในรูปที่ 4.4 เป็นดังนี้

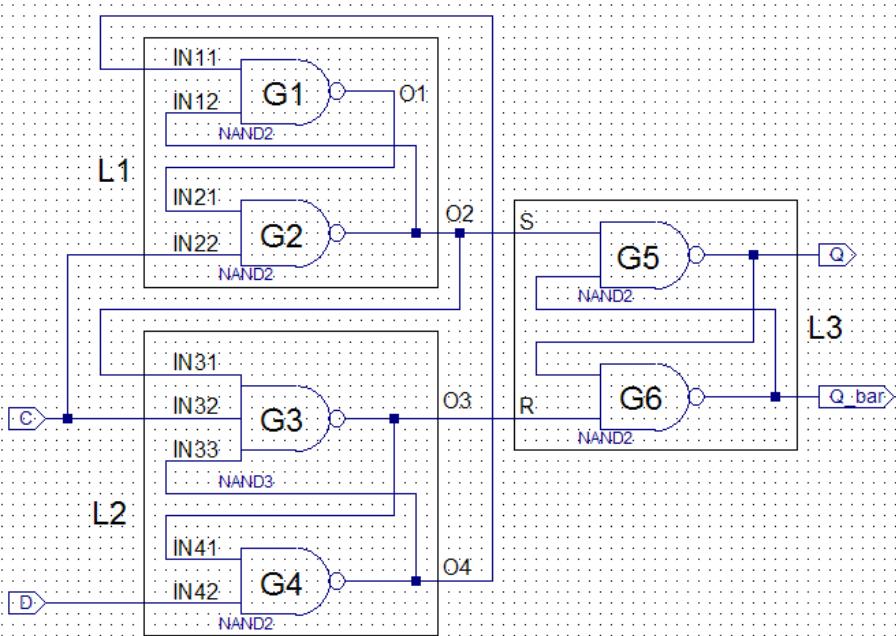
a) เมื่อขาสัญญาณนาฬิกา C = '0' ทำให้อาต์พุต $O_2 = O_3 = '1'$ (เพราะว่าถ้าอินพุตใดอินพุตนั่งของแคนเนกต์ G2, G3 เป็น '0' และขาต่อพุตจะต้องเป็น '1') ดังนั้นแล็คช์ L3 จึงอยู่ในสถานะ Hold ทำให้ Q และ Q_{bar} ไม่เปลี่ยนสถานะ และในขณะเดียวกันก็ทำให้ $O_4 = \overline{D}$ (เพราะว่า $IN_{41} = O_3 = '1'$) และ $O_1 = D$ (เพราะว่า $IN_{12} = O_2 = '1'$)

b) เมื่อขาสัญญาณนาฬิกา C = '1' ทำให้อาต์พุต $O_2 = \overline{O_1}$ (เพราะว่า $IN_{22} = C = '1' = \overline{D}$ และในขณะเดียวกัน $Q_3 = D$ (เพราะว่า $IN_{32} = C = '1'$, $IN_{33} = O_4 = \overline{D}$ และ $IN_{31} = O_2 = \overline{D}$) ดังนั้นที่อินพุตแล็คช์ L3 จะได้ $S = O_2 = \overline{D}$ และ $R = O_3 = D$ ซึ่งแสดงว่า S และ R มีค่าตรงข้ามกัน ทำให้สถานะของแล็คช์ L3 ที่เป็นไปได้คือสถานะ Set หรือ Reset จะได้ $Q = \overline{S} = D$

c) เมื่อขาสัญญาณนาฬิกา C = '0' และช์ L3 จึงกลับไปอยู่ในสถานะ Hold อีกรั้ง ซึ่งก็หมายความว่าทุกครั้งที่ทริกคัวบของขาขึ้นของสัญญาณนาฬิกาแล้วอาต์พุตจะมีการเปลี่ยนสถานะตามอินพุตคือ $Q = D$

Input		Output		Remark
D	C	Q	Q_{bar}	
0		0	1	Reset
1		1	0	Set

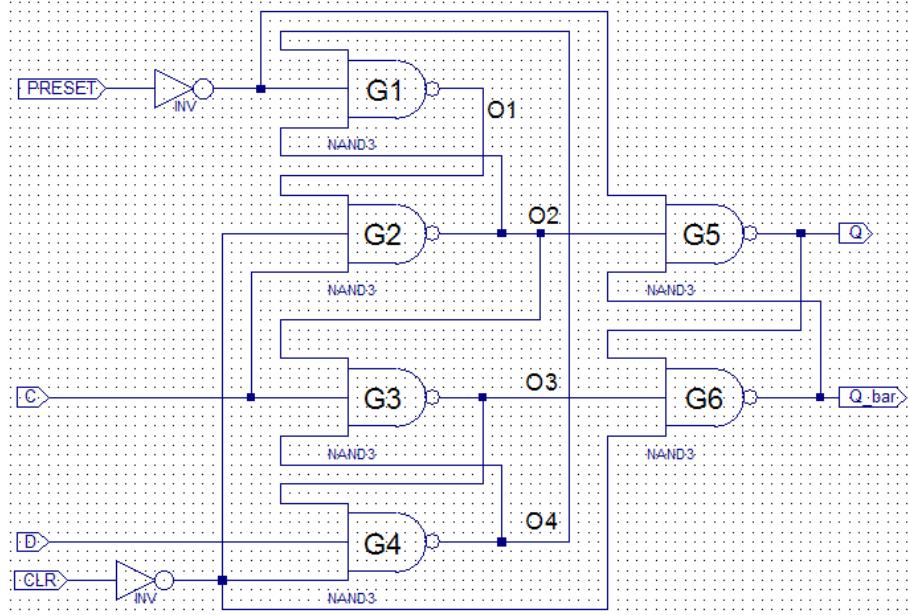
(a) ตารางความจริงของ D Flip-Flop !!แบบทริกคัวบของขาขึ้น



(b) ผังวงจรของ D Flip-Flop แบบทริกคัวบของขาขึ้น

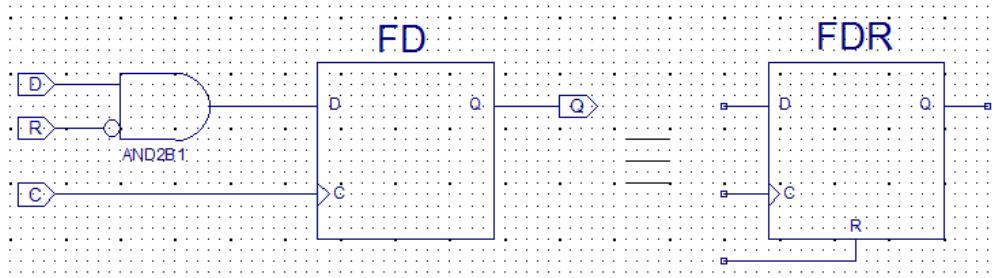
รูปที่ 4.4 ตารางความจริงและผังวงจรของ D Flip-Flop แบบทริกคัวบของขาขึ้น (ซึ่งเทียบเท่า Symbols FD)

หลักการทำงาน D Flip-Flop แบบอะซิงโกรนัสพรีเซ็ตและเคลียร์ (Asynchronous preset and clear D Flip-Flop) ในรูปที่ 4.5 จะเหมือนกับ D Flip-Flop ที่อธิบายไปแล้วในรูปที่ 4.4 แต่จะมีขา PRESET และ CLR (Clear) เพิ่มเข้ามา โดยจะมีการ PRESET ($Q = '1'$) หรือ CLR ($Q = '0'$) กีต่อเมื่อขาอินพุต PRESET หรือ CLR มีค่าลําจิกเป็น ' 1 '



รูปที่ 4.5 ผังวงจร D Flip-Flop แบบอะซิงโกรนัสพรีเซ็ตและเคลียร์

หลักการทำงานของ D Flip-Flop แบบชิงโกรนัสสีเซต (Synchronous reset D Flip-Flop) ในรูปที่ 4.6 จะมีลักษณะเหมือนกับ D Flip-Flop แต่เมื่อ R (Reset) = ' 1 ' แล้วเอาต์พุต Q จะยังไม่รีเซ็ตจนกว่าจะทริกคัวข้อมูลขึ้นของสัญญาณนาฬิกา C



รูปที่ 4.6 ผังวงจร D Flip-Flop แบบชิงโกรนัสสีเซตและทริกคัวข้อมูลขึ้น (ซึ่งเทียบเท่า Symbols FDR)

หมายเหตุ

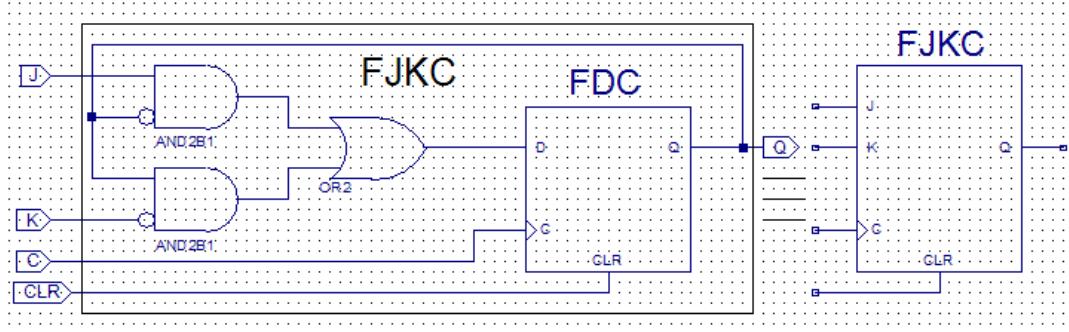
- 1 Preset (Asynchronous preset) ต่างกับ Set (Synchronous set) ตรงที่ต้องค่า $Q = '1'$ ได้ทันทีโดยไม่ต้องส่ง入สัญญาณนาฬิกา
- 2 Clear (Asynchronous clear) ต่างกับ Reset (Synchronous reset) ตรงที่ต้องค่า $Q = '0'$ ได้ทันทีโดยไม่ต้องส่ง入สัญญาณนาฬิกา

4.2.2 การออกแบบวงจร JK Flip-Flop

JK Flip-Flop คือ ฟลิปฟล็อปที่คัดแปลงมาจาก D Flip-Flop แบบอะซิงโกรนัสเคลียร์ (Asynchronous clear D Flip-Flop หรือ FDC) โดยมีตารางความจริงดังรูปที่ 4.7 และผังวงจรดังรูปที่ 4.8 โดยที่ FDC คือ ฟลิปฟล็อปในรูปที่ 4.5 ที่ $\text{PRESET} = '0'$

Input				Ouput	Remark
CLR	J	K	C	Q	
1	x	x	x	0	Clear
0	0	0	↑	Q (0/1)	Hold (No change)
0	0	1	↑	0	Reset
0	1	0	↑	1	Set
0	1	1	↑	\bar{Q} (1/0)	Toggle

รูปที่ 4.7 ตารางความจริง JK Flip-Flop



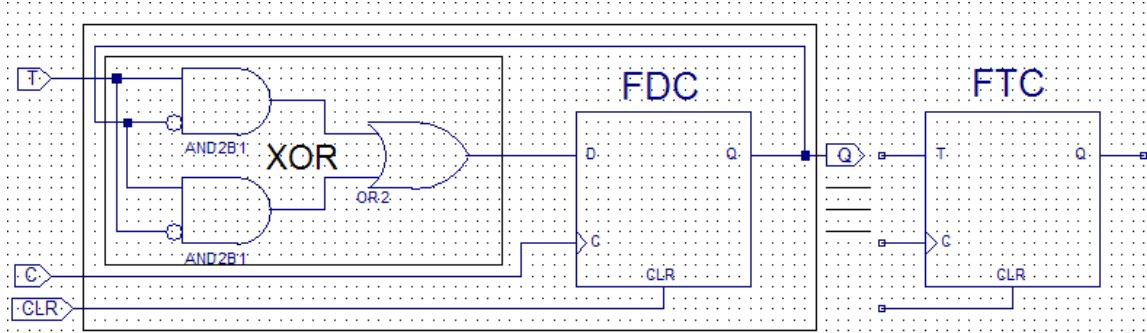
รูปที่ 4.8 ผังวงจรของ JK Flip-Flop แบบทริกค์ด้วยขอบขาขึ้น (ซึ่งเทียบเท่า Symbols FJKC)

4.3.3 การออกแบบวงจร T Flip-Flop

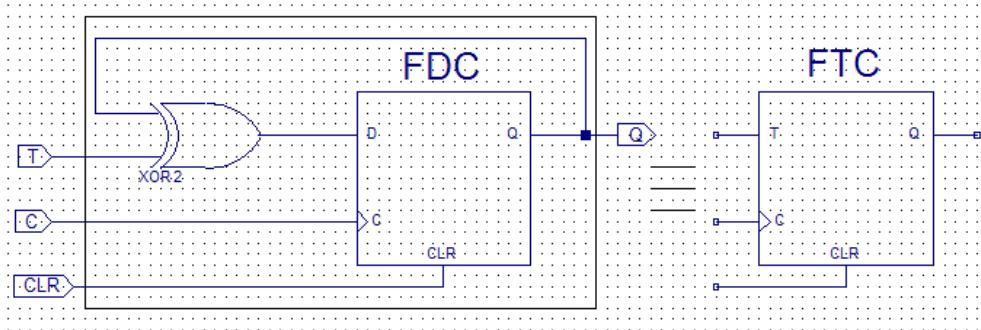
T Flip-Flop จะดัดแปลงมาจาก JK Flip-Flop โดยรวม J และ K เป็นเดียวกันเป็นอินพุต T โดยมีตารางความจริงแสดงดังรูปที่ 4.9 และผังวงจร T Flip-Flop และดังรูปที่ 4.10 และรูปที่ 4.11 โดยที่ในรูปที่ 4.11 จะเขียน T Flip-Flop ในฟอร์ม XOR

Input			Ouput	Remark
CLR	T	C	Q	
1	x	x	0	Clear
0	0	↑	Q (0/1)	Hold (No change)
0	1	↑	\bar{Q} (1/0)	Toggle

รูปที่ 4.9 ตารางความจริง T Flip-Flop



รูปที่ 4.10 ผังวงจร T Flip-Flop แบบทริกค์ด้วยขอบขาขึ้น (ซึ่งเทียบเท่า Symbols FTC)



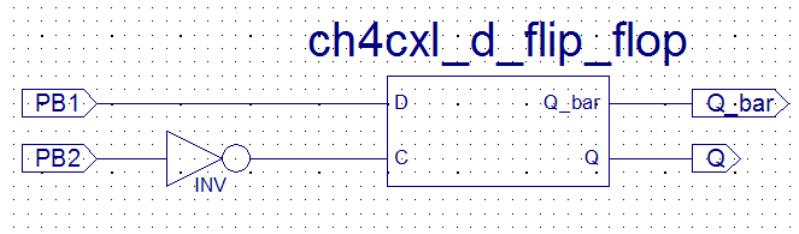
รูปที่ 4.11 ผังวงจร T Flip-Flop แบบทริกด้วยขอบขั้นที่เขียนในฟอร์แมต XOR (ชิ้นเทียบเท่า Symbols FTC)

4.2.1 การออกแบบวงจร D Flip-Flop

อุปกรณ์ที่ควรมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจร D Flip-Flop ด้วย CPLD

นำวงจร D Flip-Flop ในรูปที่ 4.4b มาทำเป็น Symbols โดยสร้างไว้ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4cxl_d_flip_flop ตามขั้นตอนในบทที่ 2 คาดผังวงจรดังรูปที่ 4.12 โดยใช้ Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_2_1cxl และเนื่องจาก PB2 ทำงานแบบ Active low การทดลองนี้จึงใส่ตัวอินเวอร์เตอร์ก่อนป้อนเข้าขาอินพุต C ของ D Flip-Flop เพื่อให้ PB2 เสมือนว่าทำงานแบบ Active high



รูปที่ 4.12 D Flip-Flop

หมายเหตุ ในการสร้าง Symbols นั้นแม้จะสร้างจากไฟล์เดียวกัน แต่รูป Symbols ที่ได้อาจจะมีตำแหน่ง I/O ไม่ตรงกัน

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB2 เป็นอินพุต และ LED1-LED2 เป็นเอาต์พุต กล่าวคือ

PB1 = PB1 = INPUT = p39 Q = LED1 = OUTPUT = p38

PB2 = PB2 = INPUT = p40 Q_bar = LED2 = OUTPUT = p37

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

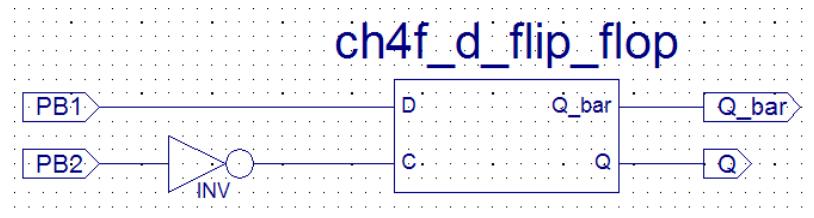
```

NET "PB1"      LOC = "p39" ;
NET "PB2"      LOC = "p40" ;
NET "Q"         LOC = "p38" | SLEW = SLOW ;
NET "Q_bar"    LOC = "p37" | SLEW = SLOW ;
    
```

หลังจากโปรแกรมลง CPLD แล้ว ให้ทดลองกดปุ่ม PB1-PB2 และให้สังเกตคุณลักษณะ LED1-LED2 ว่าให้ลอดิจิกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ หากนั้นจึงบันทึกผลการทดลอง

2. สร้างวงจร D Flip-Flop ด้วย FPGA

นำวงจร D Flip-Flop ในรูปที่ 4.3b มาทำเป็น Symbols โดยสร้างไว้ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4f_d_flip_flop ตามขั้นตอนในบทที่ 2 คาดผังวงจรดังรูปที่ 4.13 โดยใช้ Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_2_1f และเนื่องจาก PB2 ทำงานแบบ Active low การทดลองนี้จึงใส่ตัวอินเวอร์เตอร์ก่อนป้อนเข้าขาอินพุต C ของ D Flip-Flop เพื่อให้ PB2 เสมือนว่าทำงานแบบ Active high



รูปที่ 4.13 D Flip-Flop

หมายเหตุ ในการสร้าง Symbols นี้นแม่จะสร้างจากไฟล์เดียวกัน แต่รูป Symbols ที่ได้อาจจะมีตำแหน่ง I/O ไม่ตรงกัน

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB2 เป็นอินพุต และ LED L0-L1 เป็นเอาต์พุต กล่าวคือ

PB1 = PB1 = INPUT = p44 Q = L1 = OUTPUT = p77

PB2 = PB2 = INPUT = p46 Q_bar = L0 = OUTPUT = p70

โดยพิมพ์ใน Edit Constraints (Text) ดังนี้

```

NET "PB1"      LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "PB2"      LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "Q"         LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q_bar"    LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
    
```

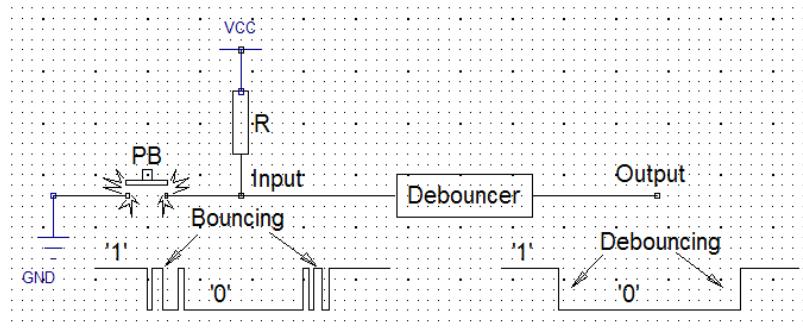
หลังจากโปรแกรมลง FPGA แล้ว ให้ทดลองกดปุ่ม PB1-PB2 และให้สังเกตดูผลที่ LED L0-L1 ว่าให้ล้อจิกเอาต์พุต เป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

4.2.2 การออกแบบวงจร JK Flip-Flop และ T Flip-Flop

อุปกรณ์ที่ควรมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

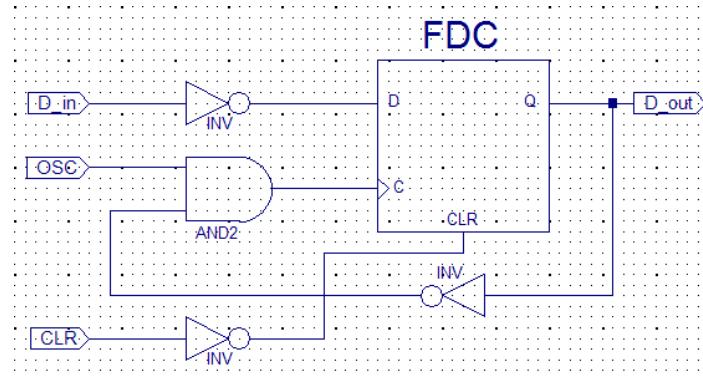
1. สร้างวงจร JK Flip-Flop ด้วย CPLD

จากตารางความจริงของ JK Flip-Flop ในรูปที่ 4.6 ถ้า J และ K เป็นล็อกจิก ‘1’ แล้วมีสัญญาณนาฬิกาหรือ Clock เข้ามาที่ก็จะทำให้อาต์พุตเปลี่ยนสถานะเป็นค่าตรงข้าม ในทางปฏิบัติ การกดคีย์บอร์ดหรือปุ่มกดเพื่อสร้าง Clock 1 ครั้งอาจให้ Clock ออกมาหลายลูก เนื่องจากมีช่วงเวลาที่หน้าสัมผัสขังแตะหรือแยกจากกันไม่สนิท (Bouncing) ทำให้ค่าอาต์พุตไม่แน่นอน ดังตัวอย่างในรูปที่ 4.14 ปกติเวลาเกิดเบนซ์จะไม่เกิน 20 มิลลิวินาที ดังนั้นการกดคีย์บอร์ดเพื่อทริก JK Flip-Flop นั้นหากพัลส์ (Pulse) ที่ได้เป็นเลขคี่ เอาต์พุตจะมีสถานะเป็นค่าตรงข้าม แต่ถ้าเป็นเลขคู่ เอาต์พุตจะมีสถานะเดิม ปัญหานี้สามารถแก้ไขโดยใช้วงจรดีเบนเชอร์ (Debouncer) หรือวงจรโมโนสเตเบิล (Monostable) เพื่อทำให้ได้สัญญาณนาฬิกาไปทริกที่อินพุตครั้งละ 1 ลูก



รูปที่ 4.14 ตัวอย่างสัญญาณอินพุตและอาต์พุตของวงจรดีเบนเชอร์บนบอร์ดหรือปุ่มกด

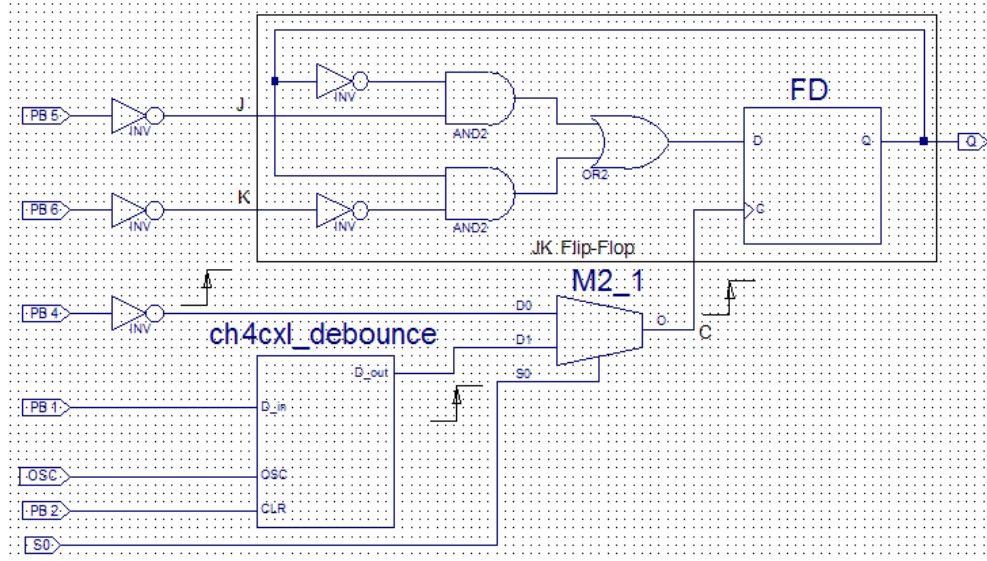
วงจรดีเบนเชอร์หรือโมโนสเตเบิลอย่างง่ายแสดงดังรูปที่ 4.15 โดยมีขา D_in รับสัญญาณจากปุ่มกด ขา OSC จะต่อเข้ากับอสซิลเลเตอร์จากภายนอก ใช้สำหรับทริก D Flip-Flop และขา CLR ใช้เคลียร์อาต์พุต D_out ของ D Flip-Flop (FDC)



รูปที่ 4.15 วงจร โมโนสเตเบิลอย่างง่ายสำหรับปุ่มกดแบบ Active low

หลักการทำงานของวงจร โมโนสเตเบิล เมื่อ D_in เป็น ‘0’ ($D = 1'$) แล้ว D Flip-Flop จะถูกทริกโดย Clock จาก OSC ผ่านทางแอนด์เกตทำให้อาต์พุต $D_{out} = 1'$ ดังนั้นขาอินพุตของแอนด์เกตที่มาจากตัวอินเวอร์เตอร์จึงเป็น ‘0’ ทำให้ Clock จาก OSC ไม่สามารถผ่านแอนด์เกตไปทริก D Flip-Flop ได้อีก พลัตต์ลูกต่อๆ ไปจากปุ่มกดที่ D_in จึงไม่สามารถส่งผ่านไปที่อาต์พุตของ D Flip-Flop ได้จนกว่าจะกดปุ่ม CLR ดังนั้นในการสร้าง Clock 1 ลูกจะต้องกดปุ่ม D_in 1 ครั้งและกดปุ่ม CLR 1 ครั้ง การเคลียร์ค่าอาต์พุตของวงจร โมโนสเตเบิลโดยอัตโนมัติสามารถทำได้โดยการป้อนความถี่ประมาณ 2-4 Hz ที่ขา CLR

การทดลองนี้ ให้วัดผังวงจร ไม่โนสเตเบิลอย่างง่ายในรูปที่ 4.15 เพื่อทำ Symbols ไว้ใน Project Location ชื่อ ch4sch และกำหนดให้ Project Name และ Source File ชื่อ ch4cxl_debounce จากนั้นวัดผังวงจรดังรูปที่ 4.16 ไว้ใน Project Location ชื่อ ch4sch โดยให้กำหนด Project Name และ Source File ชื่อ ex4_2_2cxl การทดสอบ JK Flip-Flop (ในกรอบสี่เหลี่ยมสี่เหลี่ยม) นี้ สามารถใช้มัลติเพล็กเซอร์เลือกได้ว่าจะให้สัญญาณทริกมาจาก PB1 (ผ่านวงจรโนโนสเตเบิล) หรือ PB4 โดยใช้ SO



รูปที่ 4.16 วงจรทดสอบ JK Flip-Flop

การกำหนดขาสัญญาณ ใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และ PB1-PB6 เป็นอินพุต LED1 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{ll} \text{PB1} = \text{PB1} = \text{INPUT} = \text{p39} & \text{PB4} = \text{PB4} (\text{Slide SW2}) = \text{INPUT} = \text{p43} \quad \text{Q} = \text{LED1} = \text{OUTPUT} = \text{p38} \\ \text{PB2} = \text{PB2} = \text{INPUT} = \text{p40} & \text{PB5} = \text{PB5} (\text{Slide SW3}) = \text{INPUT} = \text{p44} \quad \text{OSC} = \text{OSC} = \text{INPUT} = \text{p5} \\ \text{S0} = \text{PB3} (\text{Slide SW1}) = \text{INPUT} = \text{p42} & \text{PB6} = \text{PB6} (\text{Slide SW4}) = \text{INPUT} = \text{p1} \end{array}$$

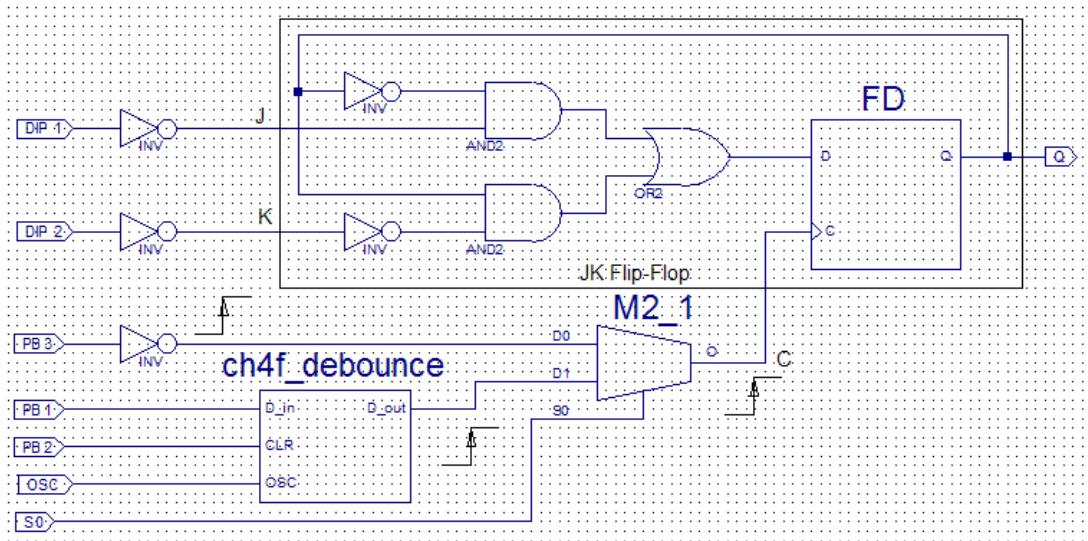
โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "PB5" LOC = "p44" ;
NET "PB6" LOC = "p1" ;
NET "OSC" LOC = "p5" ;
NET "PB1" LOC = "p39" ;
NET "PB2" LOC = "p40" ;
NET "PB4" LOC = "p43" ;
NET "Q" LOC = "p38" | SLEW = SLOW ;
NET "S0" LOC = "p42" ;
```

เมื่อโปรแกรม CPLD แล้วให้เซตค่า J และ K เป็นค่าต่างๆ ที่จะกำหนดโดยใช้ Slide SW3 และ Slide SW4 (Slide SW3 ON แล้ว J = '1', OFF แล้ว J = '0' และ Slide SW4 ON แล้ว K = '1', OFF แล้ว K = '0') จากนั้น ON Slide SW1 (PB3) แล้วกดปุ่ม PB4 (ไม่ผ่านวงจรดีเบาเซอร์) ดูที่ LED1 ว่าเป็นตามทฤษฎีหรือไม่ (ถ้าใช้ JK Flip Flop เป็น T Flip Flop ให้เซต J = K = '1') แล้วทำการทดลองซ้ำ OFF Slide SW1 (PB3) เพื่อเลือกปุ่มกด PB1 และ PB2 (ผ่านวงจรดีเบาเซอร์) แล้วจึงบันทึกผลการทดลอง

2. สร้างวงจร JK Flip-Flop ด้วย FPGA

การทดลองนี้ ให้วัดผังวงจร ไม่โนสเตเบิลตามรูปที่ 4.15 เพื่อทำ Symbols ไว้ใน Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ch4f_debounce จากนั้นวัดผังวงจรดังรูปที่ 4.17 โดยใช้ Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ex4_2_2f การทดสอบ JK Flip-Flop (ในกรอบสี่เหลี่ยมสี่เหลี่ยม) นี้ สามารถใช้มัลติเพล็กเซอร์เลือกได้ว่าจะให้สัญญาณทริกมาจาก PB1 (ผ่านวงจรดีเบาเซอร์) หรือ PB3 โดยใช้ S0



รูปที่ 4.17 ผังวงจรทดสอบ JK Flip-Flop

การกำหนดขาสัญญาณ ใช้ออสซิลเลเตอร์ OSC = 25 MHz และ PB1-PB3 และ Dip SW1-Dip SW3 เป็นอินพุต โดยเมื่อ LED L3 เป็นขาต์พุต ก่อร่องคือ

PB1 = PB1 = INPUT = p44	DIP1 = Dip SW1 = INPUT = p52	Q = L3 = OUTPUT = p76
PB2 = PB2 = INPUT = p46	DIP2 = Dip SW2 = INPUT = p53	OSC = OSC = INPUT = p127
PB3 = PB3 = INPUT = p47	S0 = Dip SW3 = INPUT = p55	

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "DIP1" LOC = "p52" | IOSTANDARD = LVCMOS33 ;
NET "DIP2" LOC = "p53" | IOSTANDARD = LVCMOS33 ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;
NET "PB1" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "PB2" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "PB3" LOC = "p47" | IOSTANDARD = LVCMOS33 ;
NET "Q" LOC = "p76" | SLEW = SLOW | IOSTANDARD = LVCMOS33 ;
NET "S0" LOC = "p55" | IOSTANDARD = LVCMOS33 ;

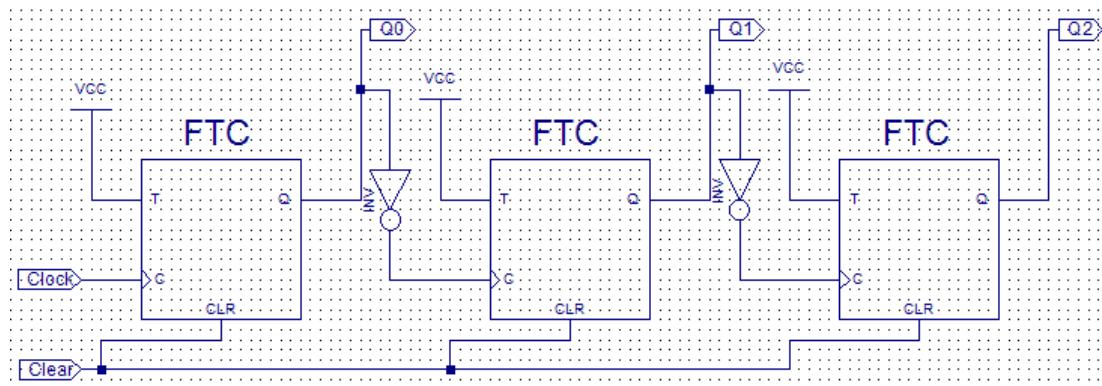
```

หลังจากโปรแกรมลง FPGA แล้วให้เซ็ตค่า J และ K เป็นค่าต่างๆ ที่จะค่าโดยใช้ Dip SW1 (เฉพาะในกรณี Dip SW1 ON แล้ว J = '1', OFF แล้ว J = '0') และ Dip SW2 (เฉพาะในกรณี Dip SW2 ON แล้ว K = '1', OFF แล้ว K = '0') แล้วเลื่อน Dip SW3 ไปที่ตำแหน่ง ON แล้วกดปุ่ม PB3 (ไม่ผ่านวงจรดีเบาเซอร์) จากนั้นดูที่ LED L3 ว่าติดสว่างเป็นตามทฤษฎีหรือไม่ (กรณีที่ใช้ JK Flip Flop เป็น T Flip Flop ให้เซ็ต J = K = '1') เสร็จแล้วทำการทดสอบซ้ำแต่เปลี่ยนเป็นเลื่อน Dip SW3 ไปที่ตำแหน่ง OFF เพื่อเลือกปุ่มกด PB1 และ PB2 (ผ่านวงจรดีเบาเซอร์) จากนั้นบันทึกผลการทดสอบ

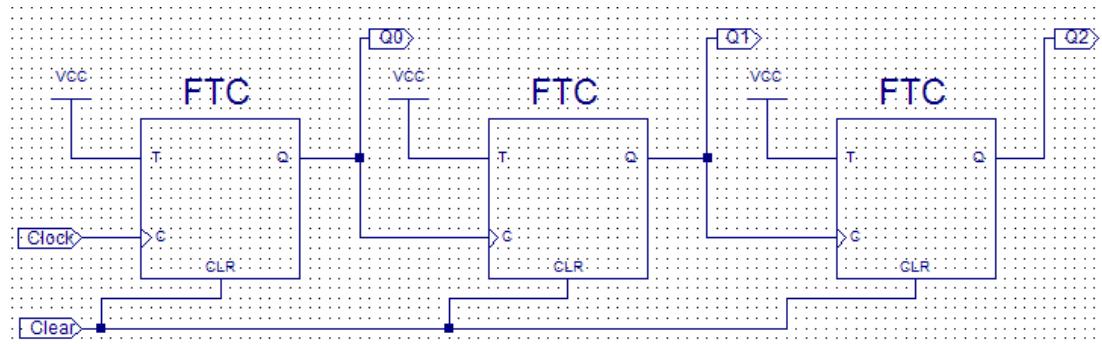
4.3 วงจรนับเลขฐานสอง (Binary counter)

4.3.1 การออกแบบวงจรนับแบบซิงโครนัส (Asynchronous counter)

วงจรนับแบบซิงโครนัสหรือแบบริปเปล (Ripple counter) จะใช้ T Flip-Flop โดยเซตค่า $T = '1'$ ทุกตัวด้วยการต่อเข้ากับ V_{CC} หรือถ้าใช้ JK Flip-Flop จะเซตค่า $J = K = '1'$ เพื่อทำงานในโหมด Toggle ทุกๆ ครั้งที่มีการทริกเกอร์ด้วยขอบขาขึ้นของสัญญาณนาฬิกา วงจรนับขึ้นแบบเลขฐานสอง (Binary up-counter) แสดงดังรูปที่ 4.18 และนับลงแสดงดังรูปที่ 4.19 ซึ่งเป็นวงจรนับ 8 กล่าวคือนับได้สูงสุด $= 2^N - 1$ โดยที่ N คือ จำนวนบิตหรือจำนวน Flip-Flop เมื่อ $\text{Clear} = '1'$ จะทำให้อเรต์พุตของ Flip-Flop ทุกตัวเป็น ' 0 ' และวงจรพร้อมที่จะนับเมื่อค่า $\text{Clear} = '0'$ วงจรนับขึ้นจะใส่อินเวอร์เตอร์ INV เพื่อทำให้อเรต์พุตของ INV จาก Q_0 และ Q_1 กลายเป็นขอบขาขึ้นเพื่อใช้ทริกเกอร์ Flip-Flop ตัวถัดไป และขอให้สังเกตว่าการ Toggle ของ Flip-Flop แต่ละตัวจะทำให้ความถี่สัญญาณนาฬิกา (Clock) ที่เออเรต์พุตลดลงครึ่งหนึ่งจากความถี่เดิมหรือทำหน้าที่หารสองของความถี่เดิมนั่นเอง



รูปที่ 4.18 วงจรนับขึ้นแบบซิงโครนัส 3 บิต (3-bit asynchronous up-counter)



รูปที่ 4.19 วงจรนับลงแบบซิงโครนัส 3 บิต (3-bit asynchronous down-counter)

4.3.2 การออกแบบวงจรนับแบบซิงโครนัส (Synchronous counter)

วงจรนับแบบซิงโครนัสเป็นวงจรที่ถูกออกแบบให้สัญญาณนาฬิกา (Clock) หรือ Flip-Flop ทุกตัวพร้อมกัน ซึ่งทำให้ วงจนับมีเวลาหน่วงภายในน้อย จึงสามารถทำงานที่ความเร็วสูงได้ วงจนับขึ้น 4 บิตแสดงดังรูปที่ 4.20 นั้นจะใช้ T Flip-Flop หรือใช้ JK Flip-Flop (โดยเซตให้ $J = K = '1'$) และสามารถสรุปเป็นเงื่อนไขทั่วๆ ไปในการออกแบบได้ดังนี้

$$T_0 = 1$$

$$T_1 = Q_0$$

$$T_2 = Q_1 \cdot T_1 = Q_1 \cdot Q_0$$

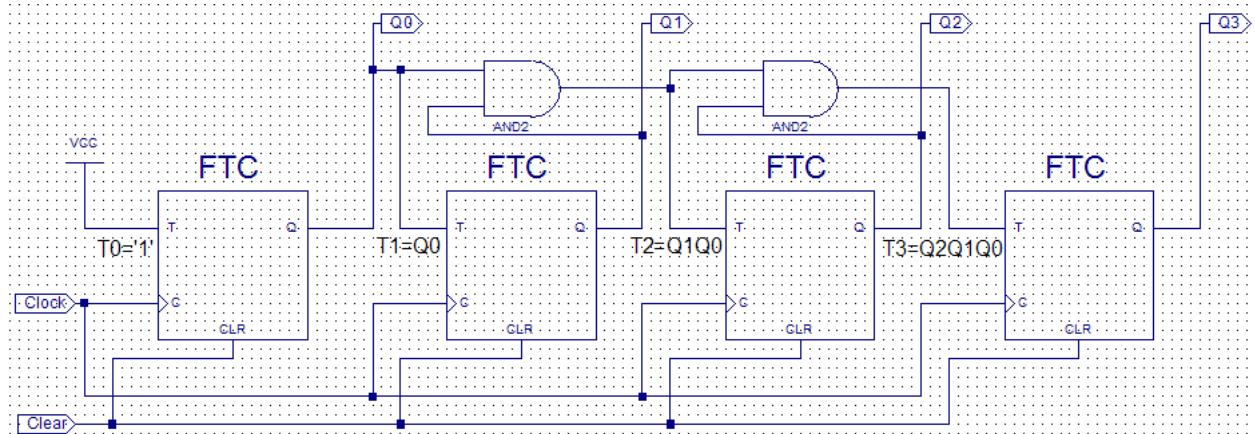
$$T_3 = Q_2 \cdot T_2 = Q_2 \cdot Q_1 \cdot Q_0$$

สมการทั่วไป

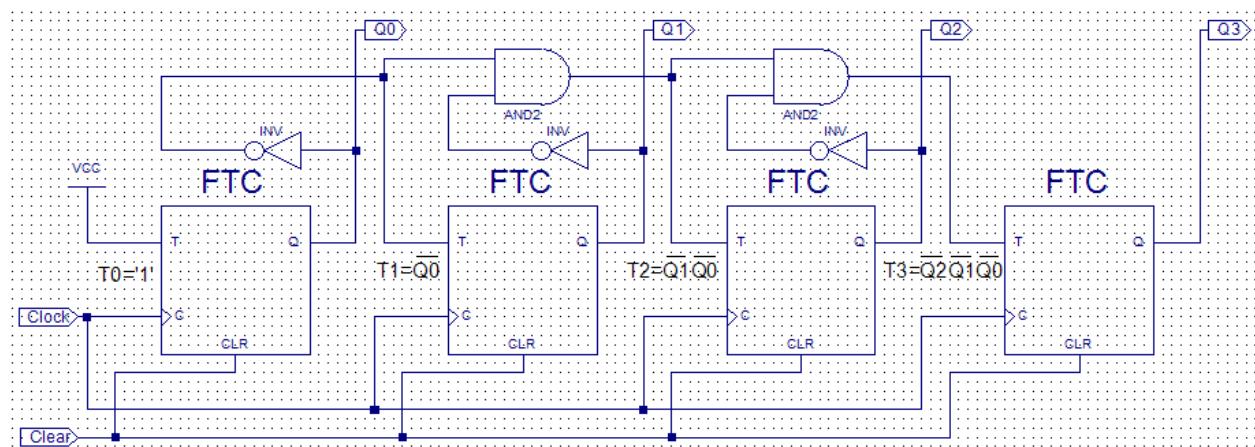
$$T(N) = Q(N-1) \cdot T(N-1) = Q(N-1) \cdot Q(N-2) \dots Q_1 \cdot Q_0$$

โดยที่ $T(N)$ คือ เอ้าต์พุตของสัญญาณที่ใช้เป็นอินพุตของ T Flip-Flop บิตที่ N

วงจนับขึ้นแบบซิงโครนัส 4 บิต (4-bit synchronous up-counter) และแสดงดังรูปที่ 4.20 และนับลงแสดงรูปที่ 4.21 ซึ่ง ค่าสูงสุดที่นับได้ $= 2^N - 1$ โดยที่ N คือ จำนวนบิต หลักการทำงานของวงจนับนี้จะอาศัยการ Toggle ของ Flip-Flop เช่นเดียวกัน โดยที่เอ้าต์พุต Flip-Flop ตัวที่เราระบุจะ Toggle เนพาะเวลาที่เอ้าต์พุตของ Flip-Flop ตัวก่อนหน้านั้นเป็นลอจิก ‘1’ เท่านั้น

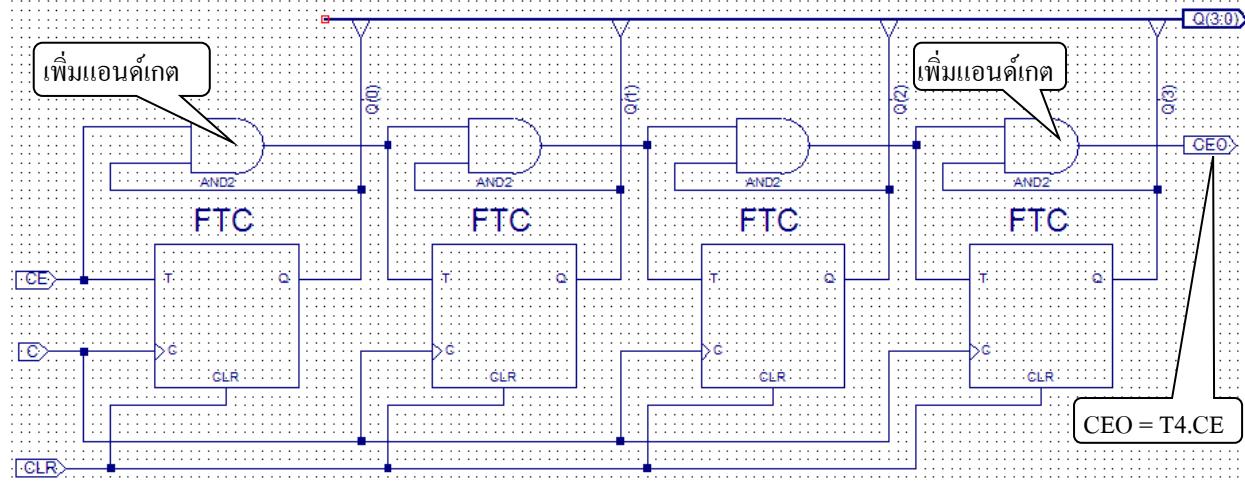


รูปที่ 4.20 วงจนับขึ้นแบบซิงโครนัส 4 บิต (4-bit synchronous up-counter)

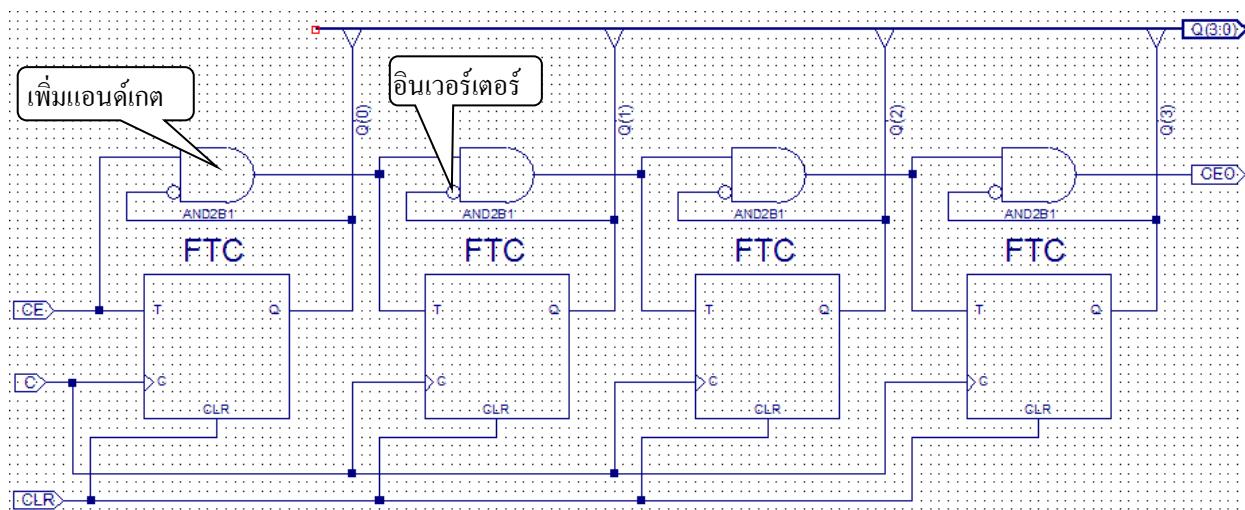


รูปที่ 4.21 วงจนับลงแบบซิงโครนัส 4 บิต (4-bit synchronous down-counter)

จากหลักการที่อธิบายมาแล้ว เราสามารถออกแบบวงจรนับขึ้นแบบค่าสเกล (Cascadable counter) โดยมีขา CE (Clock enable input) และขาตัวตัด CEO (Clock enable output) แสดงดังรูปที่ 4.15a) โดยที่วงจรจะนับเมื่อ $CE = '1'$ และ $CEO = '1'$ เมื่อนับถึงค่าสูงสุด (15) ก็ค่าว่าคือ นับได้สูงสุด $= 2^N - 1$ โดยที่ $N = \text{จำนวนบิต}$ และ $CEO = T(N).CE$ ในทำนองเดียวกันวงจรนับลงแบบค่าสเกลจะได้ดังรูปที่ 4.22b) และตัวอย่างวงจรนับขึ้นแบบค่าสเกล 12 บิตที่สร้างจากการนับ 4 บิตแสดงดังรูปที่ 4.23

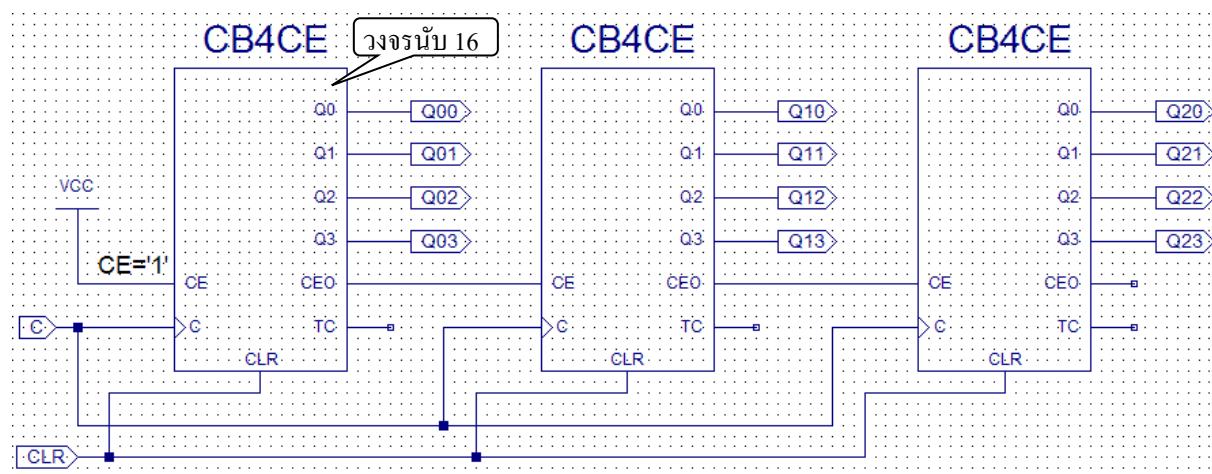


(a) วงจรนับขึ้นแบบค่าสเกล 4 บิต (4-bit cascadable up-counter)



(b) วงจรนับลงแบบค่าสเกล 4 บิต (4-bit cascadable down-counter)

รูปที่ 4.22 วงจรนับแบบค่าสเกล 4 บิต (4-bit cascadable counter with clock enable and asynchronous clear)



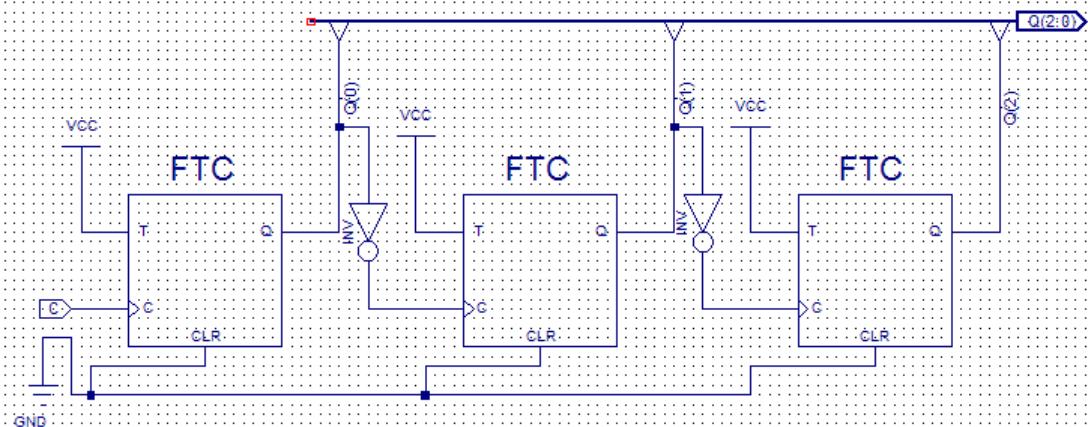
รูปที่ 4.23 วงจรนับขีน 12 บิต (12-bit binary counter with asynchronous clear)

4.3.1 การออกแบบวงจรนับขึ้นแบบริบเบิล

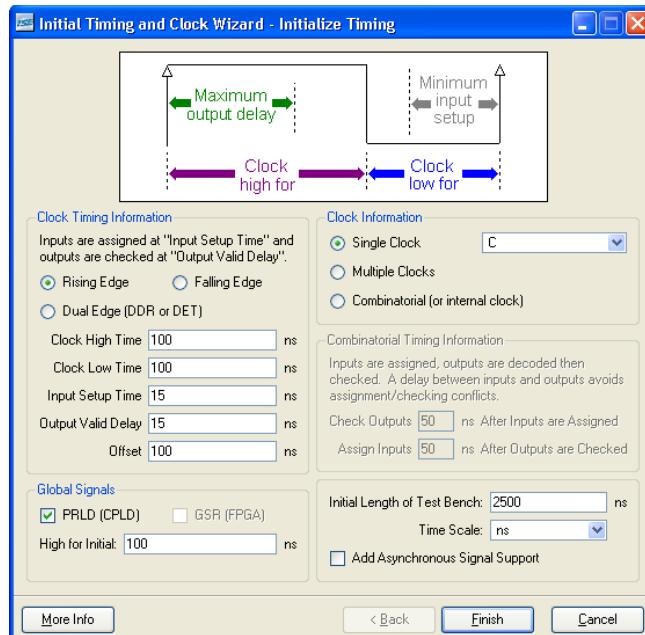
อุปกรณ์ที่ควรมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรนับขึ้นแบบริบเบิลด้วย CPLD

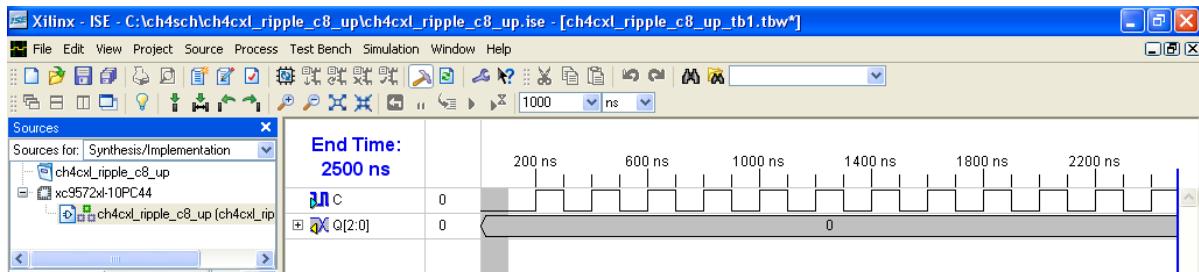
a) สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ch4cxl_ripple_c8_up และวัดผังวงจรดังรูปที่ 4.24 ทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 โดยใช้ไฟล์ชื่อ ch4cxl_ripple_c8_up_tb1 และกำหนดค่าและ Waveform ดังรูปที่ 4.25 และรูปที่ 4.26 แล้วพิจารณาผล Behavioral simulation ในรูปที่ 4.27 ว่าเป็นไปตามทฤษฎีหรือไม่



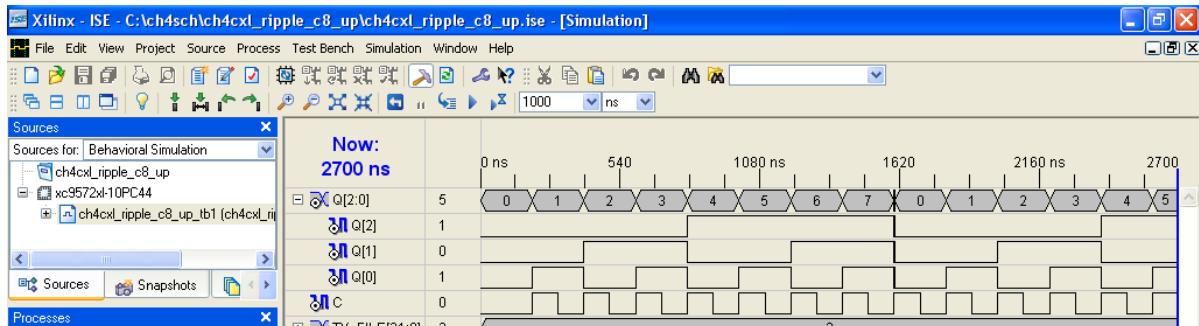
รูปที่ 4.24 ผังวงจรนับขึ้นแบบริบเบิล 3 บิต (วงจรนับ 8 แบบนับขึ้น)



รูปที่ 4.25 หน้าต่าง Initial Timing and Clock Wizard

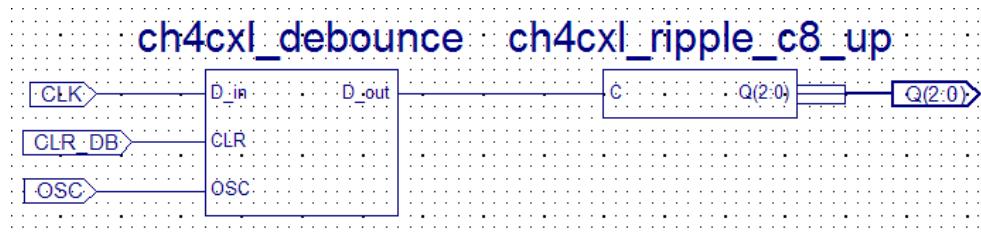


รูปที่ 4.26 หน้าต่างสำหรับกำหนดสัญญาณต่างๆ ที่ป้อนให้กับอินพุตของวงจรที่เราต้องการทดสอบ



รูปที่ 4.27 ผล Behavioral simulation ของวงจรนับ 8 แบบบันทึก

b) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_3_1cxl จากนั้นนำไฟล์วงจรโมโนสเตเบิลชื่อ ch4cxl_debounce และไฟล์วงจรนับชื่อ ch4cxl_ripple_c8_up มาทำ Symbols และวิเคราะห์วงจรดังรูปที่ 4.28



รูปที่ 4.28 วงจรทดสอบวงจรนับ 8

การกำหนดขาสัญญาณต่างๆ ใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB2 เป็นอินพุต มี LED2-LED4 เป็นเอาต์พุต กล่าวคือ

CLK = PB1 = INPUT = p39

Q(2) = LED2 = OUTPUT = p37

CLR_DB = PB2 = INPUT = p40

Q(1) = LED3 = OUTPUT = p36

OSC = OSC = INPUT = p5

Q(0) = LED4 = OUTPUT = p35

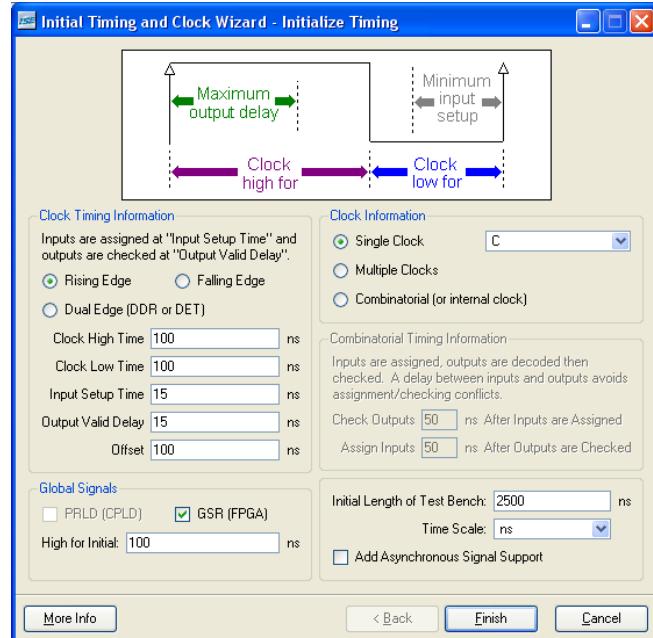
โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "CLK" LOC = "p39" ;
NET "CLR_DB" LOC = "p40" ;
NET "OSC" LOC = "p5" ;
NET "Q<0>" LOC = "p35" | SLEW = SLOW ;
NET "Q<1>" LOC = "p36" | SLEW = SLOW ;
NET "Q<2>" LOC = "p37" | SLEW = SLOW ;
```

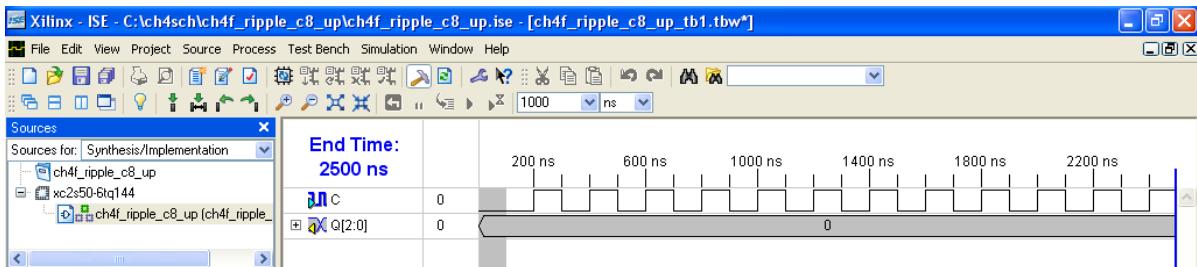
หลังจากโปรแกรมลง CPLD แล้ว ให้กดปุ่ม PB1 และ PB2 ลับกันไปเรื่อยๆ แล้วให้สังเกตคุณผลที่ LED2-LED4 ว่าให้ลอกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดสอบ

2. สร้างวงจรนับขั้นแบบรีปเปิลด้วย FPGA

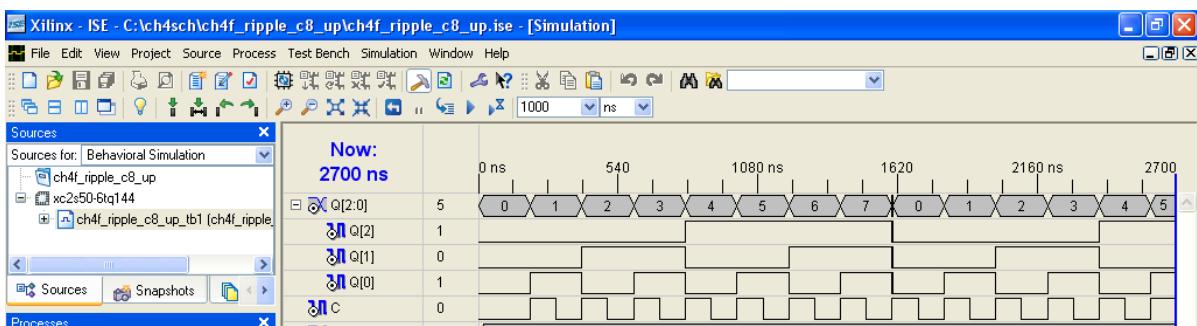
a) สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ch4f_ripple_c8_up และวิเคราะห์การดังรูปที่ 4.24 จากนั้นทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 โดยใช้ไฟล์ชื่อ ch4f_ripple_c8_up_tb1 และกำหนดค่าและ Waveform ดังรูปที่ 4.29 และรูปที่ 4.30 แล้วให้พิจารณาผล Behavioral simulation รูปที่ 4.31 ว่าเป็นไปตามทฤษฎีหรือไม่



รูปที่ 4.29 หน้าต่าง Initial Timing and Clock Wizard

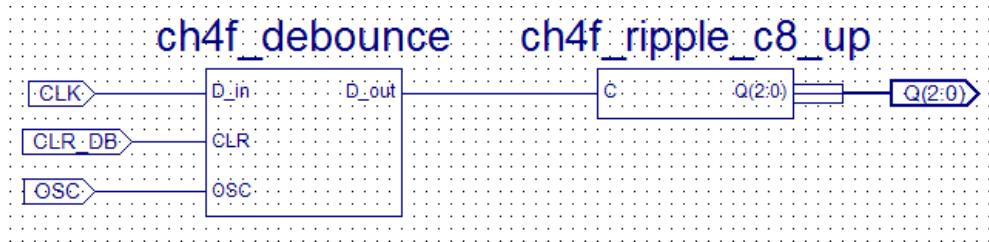


รูปที่ 4.30 หน้าต่างสำหรับกำหนดสัญญาณต่างๆ ที่ป้อนให้กับอินพุตของวงจรที่เราต้องการทดสอบ



รูปที่ 4.31 ผล Behavioral simulation ของวงจรนับ 8 แบบนับขึ้น

b) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนดให้ Project Name และ Source File ชื่อ ex4_3_1f จากนั้นนำไฟล์วงจรโนนิสเดบอนซ์ชื่อ ch4f_debounce และไฟล์วงจรนับชื่อ ch4f_ripple_c8_up มาทำ Symbols และวิเคราะห์การดังรูปที่ 4.32



รูปที่ 4.32 วงจรทดสอบวงจรนับ 8 แบบนับขึ้น

การกำหนดขาสัญญาณต่างๆ ใช้ออสซิลเลเตอร์ OSC = 25 MHz และปุ่มกด PB1-PB2 เป็นอินพุต มี LED L0-L2 เป็นเอ้าต์พุต กล่าวคือ

$$\text{CLK} = \text{PB1} = \text{INPUT} = \text{p44}$$

$$\text{Q}(2) = \text{L2} = \text{OUTPUT} = \text{p69}$$

$$\text{CLR_DB} = \text{PB2} = \text{INPUT} = \text{p46}$$

$$\text{Q}(1) = \text{L1} = \text{OUTPUT} = \text{p77}$$

$$\text{OSC} = \text{OSC} = \text{INPUT} = \text{p127}$$

$$\text{Q}(0) = \text{L0} = \text{OUTPUT} = \text{p70}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR_DB" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;
NET "Q<0>" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q<1>" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q<2>" LOC = "p69" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
```

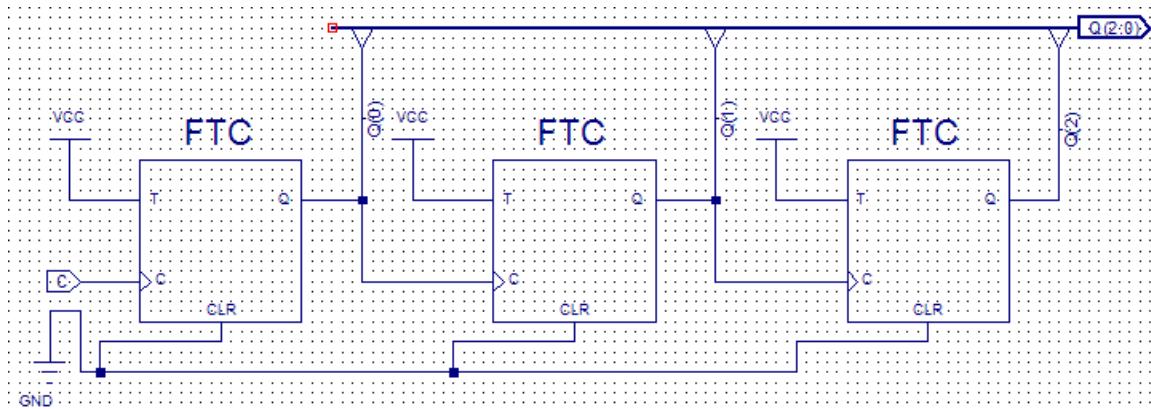
หลังจากโปรแกรมลง FPGA แล้ว ให้กดปุ่ม PB1 และ PB2 สลับกันไปเรื่อยๆ แล้วให้สังเกตคุณผลที่ LED L0-L2 ว่าให้ล้อจิกเอ้าต์พุตเป็นไปตามทฤษฎีหรือไม่ จำนวนนับขึ้นบันทึกผลการทดลอง

4.3.2 การออกแบบวงจรนับลงแบบริปปิล

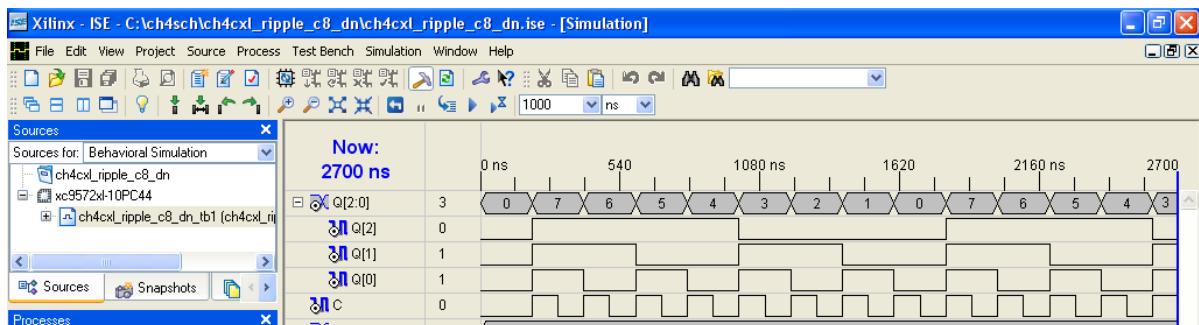
อุปกรณ์ที่ควรมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรนับลงแบบริปปิลด้วย CPLD

- a) สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ch4cxl_ripple_c8_dn และวาดผังวงจรดังรูปที่ 4.33 ทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 โดยใช้ไฟล์ชื่อ ch4cxl_ripple_c8_dn_tb1 และพิจารณาผล Behavioral simulation ในรูปที่ 4.34 ว่าเป็นไปตามทฤษฎีหรือไม่

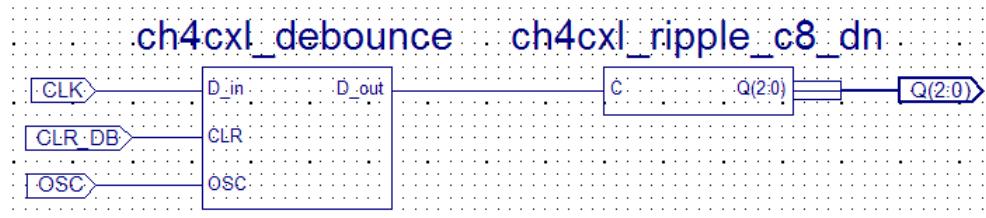


รูปที่ 4.33 ผังวงจรนับลงแบบริปปิล



รูปที่ 4.34 ผล Behavioral simulation ของวงจรนับ 8 (3 บิต) แบบนับลง

- b) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_3_2cxl จากนั้นนำไฟล์วงจรไมโนนสเตเบิลชื่อ ch4cxl_debounce และไฟล์วงจรนับลงแบบริปปิลชื่อ ch4cxl_ripple_c8_dn ที่มีอยู่แล้วมาทำ Symbols จากนั้นวาดผังวงจรดังรูปที่ 4.35



รูปที่ 4.35 วงจรทดสอบของชิ้น 8 แบบนับลง

การกำหนดขาสัญญาณต่างๆ ใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB2 เป็นอินพุต มี LED2-LED4 เป็นเอาต์พุต กล่าวคือ

$$\text{CLK} = \text{PB1} = \text{INPUT} = \text{p39}$$

$$\text{Q}(2) = \text{LED2} = \text{OUTPUT} = \text{p37}$$

$$\text{CLR_DB} = \text{PB2} = \text{INPUT} = \text{p40}$$

$$\text{Q}(1) = \text{LED3} = \text{OUTPUT} = \text{p36}$$

$$\text{OSC} = \text{OSC} = \text{INTPUT} = \text{p5}$$

$$\text{Q}(0) = \text{LED4} = \text{OUTPUT} = \text{p35}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

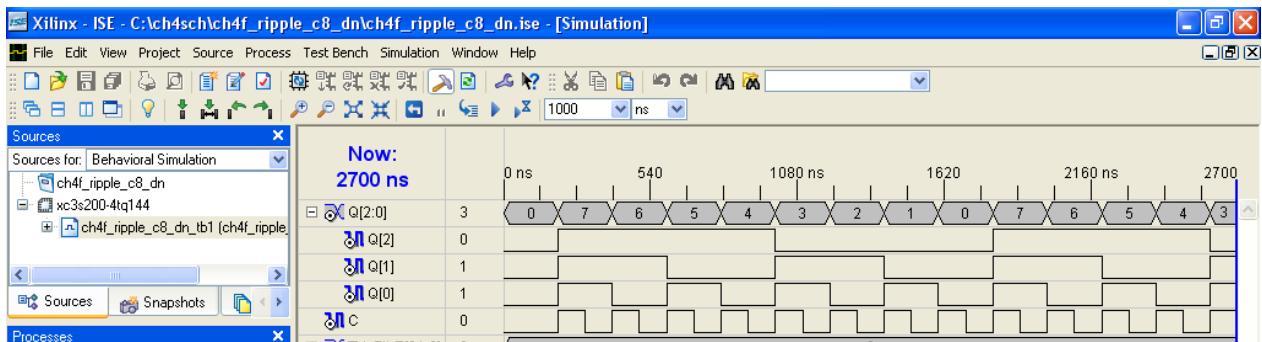
NET "CLK" LOC = "p39" ;
NET "CLR_DB" LOC = "p40" ;
NET "OSC" LOC = "p5" ;
NET "Q<0>" LOC = "p35" | SLEW = SLOW ;
NET "Q<1>" LOC = "p36" | SLEW = SLOW ;
NET "Q<2>" LOC = "p37" | SLEW = SLOW ;

```

หลังจากโปรแกรมลง CPLD แล้ว ให้กดปุ่ม PB1 และ PB2 สลับกันไปเรื่อยๆ แล้วให้สังเกตคุณผลที่ LED2-LED4 ว่าติด ส่องไฟให้ถูกต้องเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

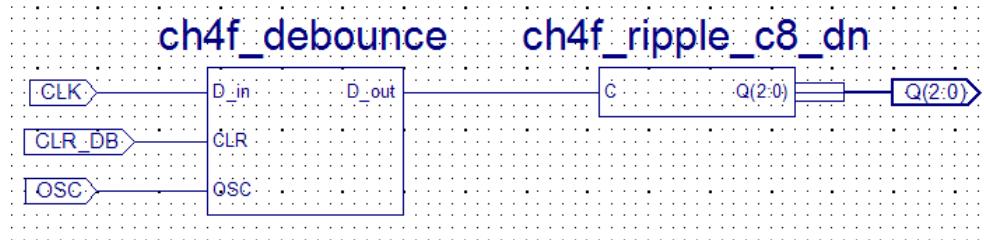
2. สร้างวงจรนับลงแบบรีปีล์ด้วย FPGA

a) สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ch4f_ripple_c8_dn และวัดผังวงจรดังรูปที่ 4.33 จากนั้นทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 โดยใช้ไฟล์ชื่อ ch4f_ripple_c8_dn_tb1 และให้พิจารณาผล Behavioral simulation ดังรูปที่ 4.36 ว่าเป็นไปตามทฤษฎีหรือไม่



รูปที่ 4.36 ผล Behavioral simulation ของวงจรนับ 8 (3 บิต) แบบนับลง

b) สร้างไฟล์ใน Project Location ชื่อ ch4sch และให้กำหนด Project Name และ Source File ชื่อ ex4_3_2f จากนั้นนำไฟล์งงานไม่อนาล็อกชื่อ ch4f_debounce และไฟล์งงานนับชื่อ ch4f_ripple_c8_dn มาทำ Symbols และวัดผังวงจรดังรูปที่ 4.37



รูปที่ 4.37 วงจรทดสอบวงจรนับ 8 แบบนับลง

การกำหนดขาสัญญาณต่างๆ ใช้ออสซิลเลเตอร์ OSC = 25 MHz และปุ่มกด PB1-PB2 เป็นอินพุต มี LED L0-L2 เป็นเออต์พุต กดล่างคือ

CLK = PB1 = INPUT = p44 Q(2) = L2 = OUTPUT = p69

CLR_DB = PB2 = INPUT = p46 Q(1) = L1 = OUTPUT = p77

OSC = OSC = INPUT = p127 Q(0) = L0 = OUTPUT = p70

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR_DB" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;
NET "Q<0>" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q<1>" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q<2>" LOC = "p69" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
```

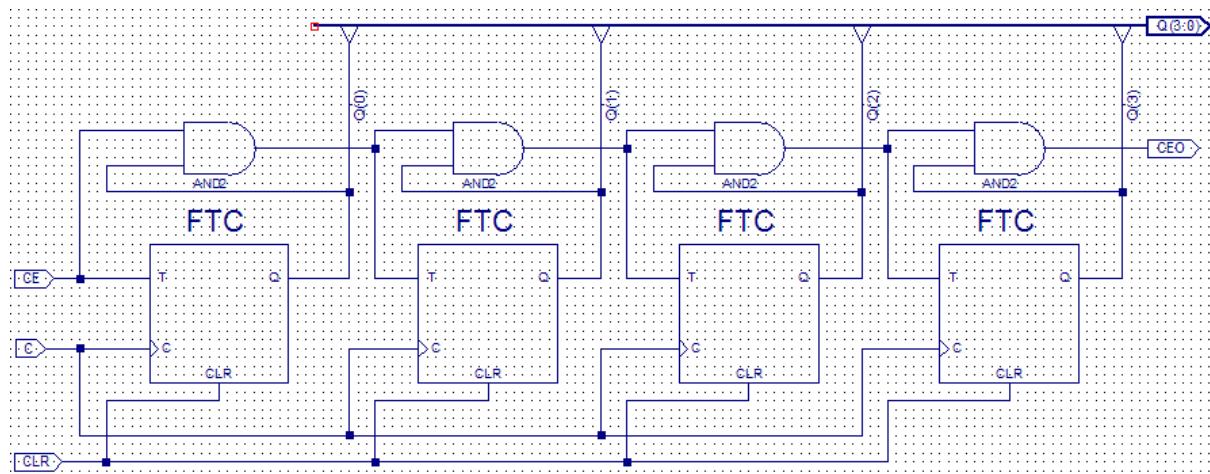
หลังจากโปรแกรมลง FPGA แล้วให้กดปุ่ม PB1 และ PB2 สลับกันไปเรื่อยๆ แล้วให้สังเกตคุณลักษณะที่ LED L0-L2 ว่าติดส่วนไหนบ้างโดยใช้ค่า 0 代表 ไม่ติด และ 1 代表 ติด

4.3.3 การออกแบบวงจรนับขั้นแบบชิงโกรนัส

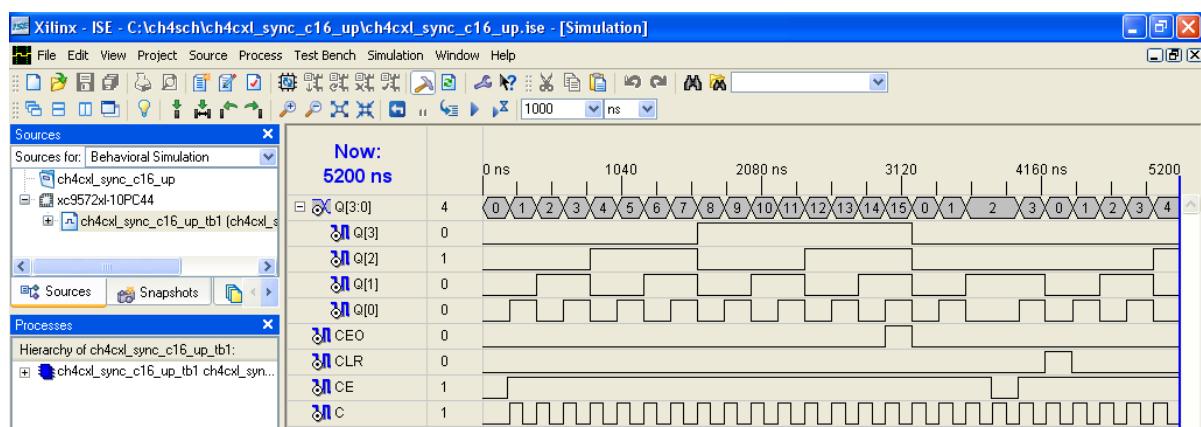
อุปกรณ์ที่ควรมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรนับขั้นแบบชิงโกรนัสด้วย CPLD

a) สร้างไฟล์ใน Project Location ชื่อ ch4sch ให้กำหนด Project Name และ Source File ชื่อ ch4cxl_sync_c16_up และวัดผังวงจรดังรูปที่ 4.38 ทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 โดยใช้ไฟล์ชื่อ ch4cxl_sync_c16_up_tb1 และพิจารณาผล Behavioral simulation ในรูปที่ 4.39 ว่าเป็นไปตามทฤษฎีหรือไม่



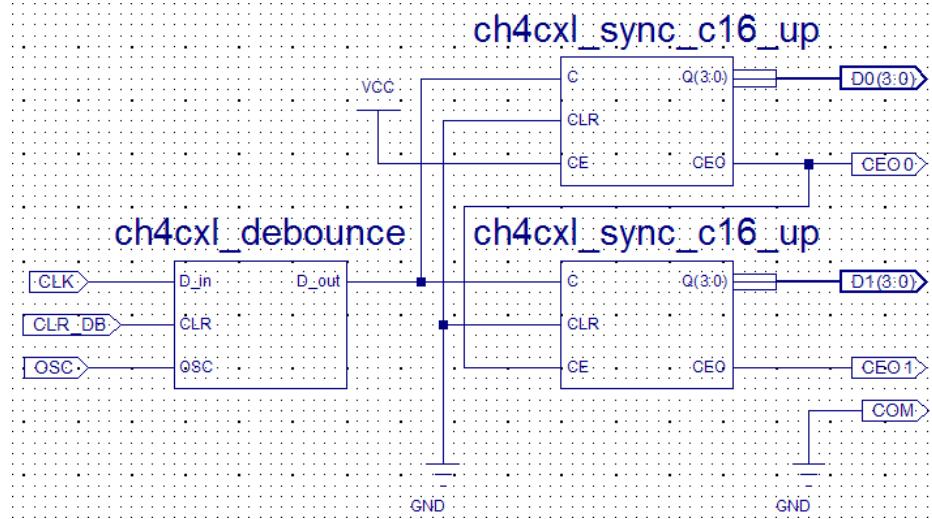
รูปที่ 4.38 พังวงจรนับขั้นแบบชิงโกรนัส 4 บิต (วงจรนับ 16 แบบนับขึ้น)



รูปที่ 4.39 ผล Behavioral simulation ของวงจรนับ 16 แบบนับขึ้น

b) สร้างไฟล์วงจร FF (0-255) ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_3_3cxl จากนั้นนำไฟล์วงจรโนโนสเตเบิลชื่อ ch4cxl_debounce และไฟล์วงจรนับชื่อ ch4cxl_sync_c16_up มาทำ Symbols และวัดผังวงจรดังรูปที่ 4.40 การแสดงผลวงจรนับ 16 ทั้ง 2 หลัก จะใช้ LED3 และ LED4 แสดงผลตัวทด CEO1 และ CEO0 และใช้

LED ທັງໝາດ 8 ດຽວທີ່ອູ່ໃນເຊວນເຊັກມັນຕີ DIGIT1 ໃນການແສດງຜລ D0(3:0) (ໜັກທີ່ 0) ແລະ D1(3:0) (ໜັກທີ່ 1) ດັ່ງນັ້ນໃນຜັງວົງຈົງທີ່ອັນດີຕ່ອງຕ່ອງຂາຄາໂຄດຮ່ວມ (COM) ຂອງ DIGIT1 ລົງກວາດ໌



ຮູບທີ່ 4.40 ວົງຈົງສອນວາງຈານນັ້ນ 16 ແບນນັ້ນເພື່ອ 2 ຜລັກ (0-255)

ການກຳໜາໜາສັນຍານຕ່າງໆ ໃຊ້ອອສີລາເລເຕອຣ໌ OSC = 32.768 kHz ແລະ ປູມກົດ PB1-PB2 ເປັນອິນພູດ ມີ LED3-LED4 ແລະ ເຊວນເຊັກມັນຕີ DIGIT1 ເປັນເອົາເຕີພູດ ໂດຍໃຫ້ເຊັກມັນຕີ a, b, c ແລະ d ໃນການແສດງຜລ D0(3:0) ແລະ ໃຫ້ເຊັກມັນຕີ e, f, g ແລະ dp ໃນການແສດງຜລ D1(3:0) ດັ່ງນັ້ນໃນຜັງວົງຈົງທີ່ອັນດີຕ່ອງຕ່ອງຂາຄາໂຄດຮ່ວມ (COM) ຂອງ DIGIT1 ລົງກວາດ໌ດ້ວຍ ກລ່າວກື່ອ

CLK = PB1 = INPUT = p39	D0(3) = d = OUTPUT = p24	D1(3) = dp = OUTPUT = p19
CLR_DB = PB2 = INPUT = p40	D0(2) = c = OUTPUT = p25	D1(2) = g = OUTPUT = p18
OSC = OSC = INPUT = p5	D0(1) = b = OUTPUT = p26	D1(1) = f = OUTPUT = p20
COM = DIGIT1 = OUTPUT = p34	D0(0) = a = OUTPUT = p27	D1(0) = e = OUTPUT = p22
CEO0 = LED4 = OUTPUT = p35		
CEO1 = LED3 = OUTPUT = p36		

ໂດຍພິມພຶ່ນໃນ Edit Constraints (Text) ສຽງຕັ້ງນີ້

```

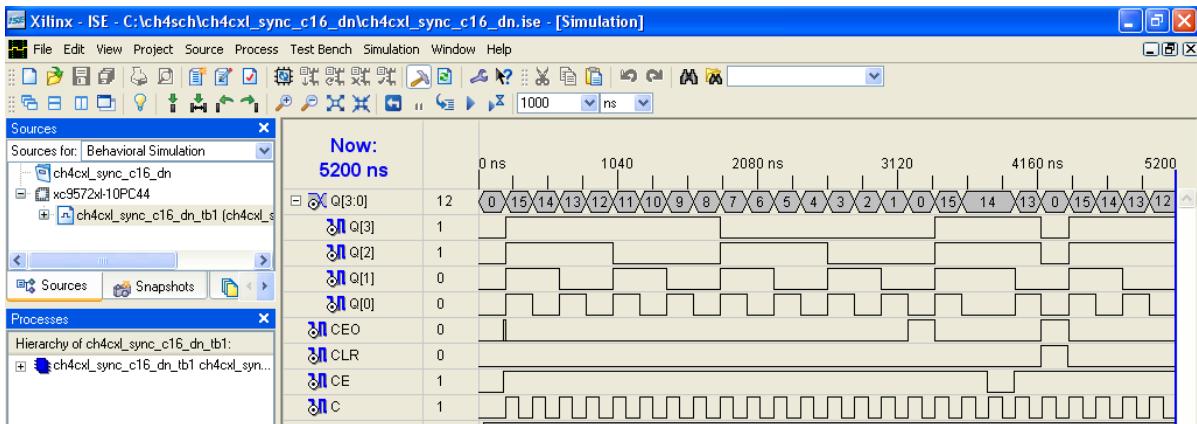
NET "CEO0" LOC = "p35" | SLEW = SLOW ;
NET "CEO1" LOC = "p36" | SLEW = SLOW ;
NET "CLK" LOC = "p39" ;
NET "CLR_DB" LOC = "p40" ;
NET "COM" LOC = "p34" | SLEW = SLOW ;
NET "DO<0>" LOC = "p27" | SLEW = SLOW ;
NET "DO<1>" LOC = "p26" | SLEW = SLOW ;
NET "DO<2>" LOC = "p25" | SLEW = SLOW ;
NET "DO<3>" LOC = "p24" | SLEW = SLOW ;
NET "D1<0>" LOC = "p22" | SLEW = SLOW ;
NET "D1<1>" LOC = "p20" | SLEW = SLOW ;
NET "D1<2>" LOC = "p18" | SLEW = SLOW ;
NET "D1<3>" LOC = "p19" | SLEW = SLOW ;
NET "OSC" LOC = "p5" ;

```

ໜັງຈາກໂປຣແກຣມລັດ CPLD ແລ້ວ ໃຫ້ກົມ PB1 ແລະ PB2 ສລັບກັນໄປເຮືອຍໆ ແລ້ວໃຫ້ສັງເກດຄູພລື່ ແລະ ເຊວນເຊັກມັນຕີ DIGIT1 ວ່າໃຫ້ລອຈິກເອົາເຕີພູດເປັນໄປຕາມທຄມງິນ້ ອີ່ໄວ່ ຈາກນັ້ນຈຶ່ງບັນທຶກພາກທົດລອງ

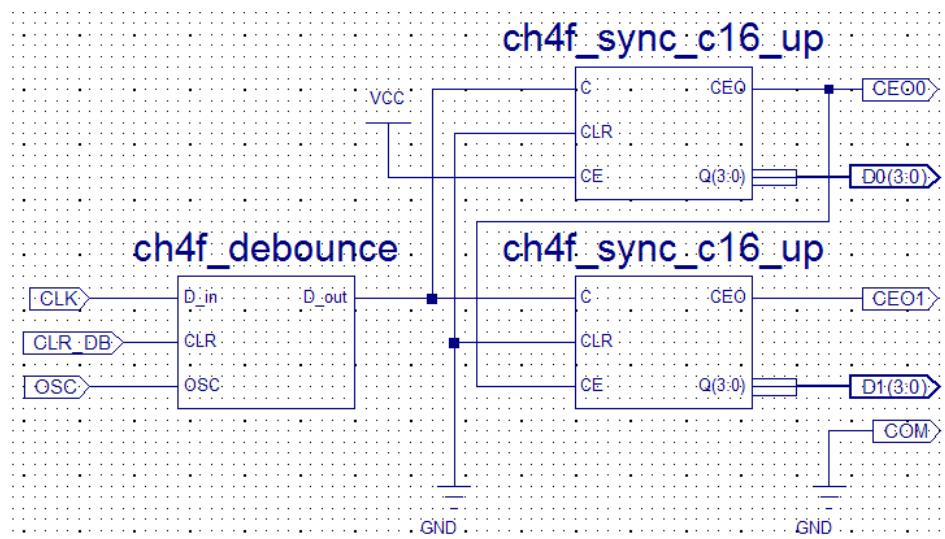
2. สร้างวงจรนับขั้นแบบชิงโคนนัสด้วย FPGA

a) สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4f_sync_c16_up และวัดผังวงจรดังรูปที่ 4.38 จากนั้นทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 โดยใช้ไฟล์ชื่อ ch4f_sync_c16_up_tb1 และวิเคราะห์ผล Behavioral simulation ดังรูปที่ 4.41 ว่าเป็นไปตามทฤษฎีหรือไม่



รูปที่ 4.41 ผล Behavioral simulation ของวงจรนับ 16 แบบนับขั้น

b) สร้างไฟล์วงจรนับ FF (0-255) ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_3_3f จากนั้นนำไฟล์วงจรโนโนสเตเบิลชื่อ ch4f_debounce และไฟล์วงจรนับชื่อ ch4f_sync_c16_up มาทำ Symbols และวัดผังวงจรดังรูปที่ 4.42 การแสดงผลวงจรนับ 16 ทั้ง 2 หลัก จะใช้ LED L1 และ L0 และผลตัวทด CEO1 และ CEO0 และใช้ LED ทั้งหมด 8 ดวงที่อยู่ในเซเว่นเซกเมนต์ DIGIT1 ในการแสดงผล D0(3:0) (หลักที่ 0) และ D1(3:0) (หลักที่ 1) ดังนั้นในผังวงจรจึงต้องต่อขาโถคร่วม (COM) ของ DIGIT1 ลงกราวด์



รูปที่ 4.42 วงจรทดสอบวงจรนับ 16 แบบนับขั้น 2 หลัก (0-255)

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25 MHz และ ปุ่มกด PB1-PB2 เป็นอินพุต มี LED L0-L1 และเซเว่นเซกเมนต์ DIGIT1 เป็นเอาต์พุต โดยใช้เซกเมนต์ a, b, c และ d ในการแสดงผล D0(3:0) และใช้เซกเมนต์ e, f, g และ dp ในการแสดงผล D1(3:0) ดังนั้นในผังวงจรจึงต้องต่อขาโถคร่วม (COM) ของ DIGIT1 ลงกราวด์ด้วย ก่อร่องคือ

CLK = PB1 = INPUT = p44	D0(3) = d = OUTPUT = p30	D1(3) = dp = OUTPUT = p20
CLR_DB = PB2 = INPUT = p46	D0(2) = c = OUTPUT = p32	D1(2) = g = OUTPUT = p23
OSC = OSC = INPUT = p127	D0(1) = b = OUTPUT = p35	D1(1) = f = OUTPUT = p25
COM = DIGIT1 = OUTPUT = p31	D0(0) = a = OUTPUT = p40	D1(1) = f = OUTPUT = p25
CEO0 = L0 = OUTPUT = p70		
CEO1 = L1 = OUTPUT = p77		

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "CEO0" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "CEO1" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR_DB" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "COM" LOC = "p31" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<0>" LOC = "p40" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<1>" LOC = "p35" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<2>" LOC = "p32" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<3>" LOC = "p30" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<0>" LOC = "p27" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<1>" LOC = "p25" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<2>" LOC = "p23" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<3>" LOC = "p20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;

```

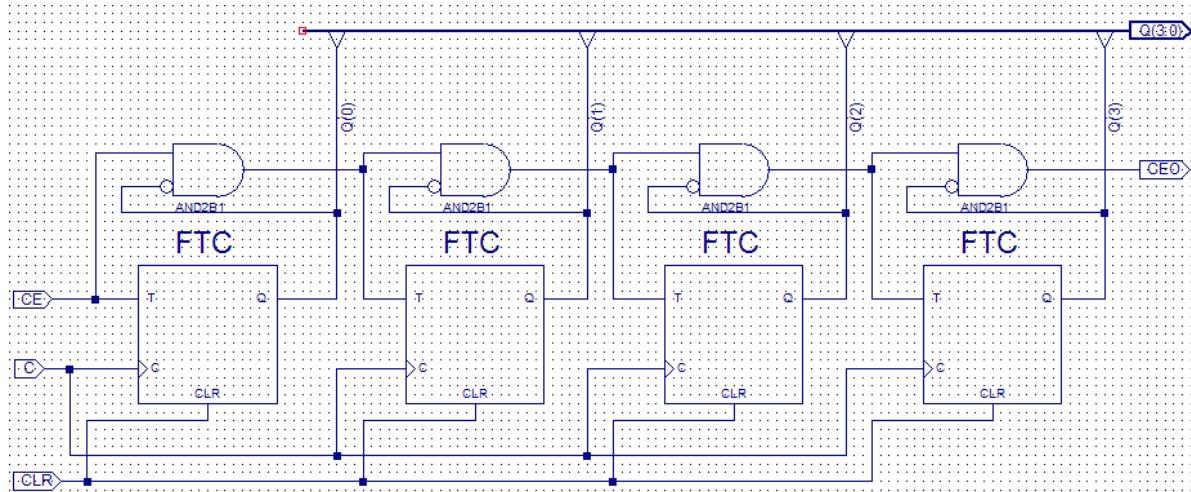
หลังจากโปรแกรมลง FPGA แล้ว ให้กดปุ่ม PB1 และ PB2 สลับกันไปเรื่อยๆ แล้วให้สังเกตคุณลักษณะ LED L0-L1 และ เช่วนเซกเมนต์ DIGIT1 ว่าแต่ละเซกเมนต์ติดสว่าง โดยให้อาต์พุตเป็นไปตามทุกถูกต้องหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

4.3.4 การออกแบบวงจรนับลงแบบชิงโครนัส

อุปกรณ์ที่ควรมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

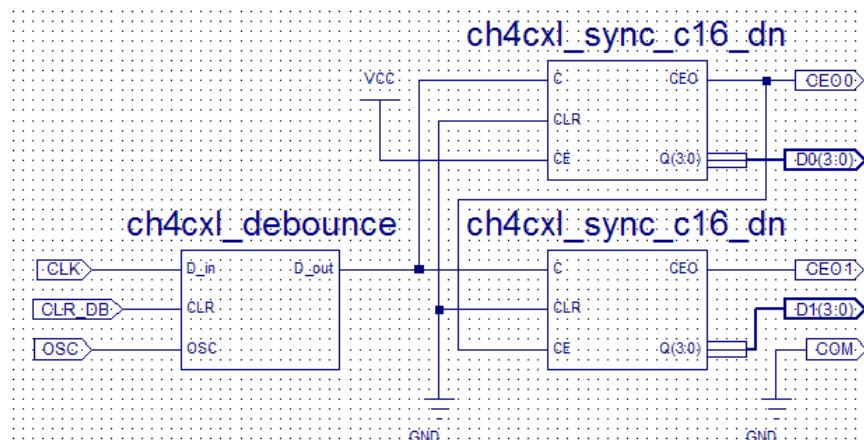
1. สร้างวงจรนับลงแบบชิงโครนัสด้วย CPLD

- a) สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4cxl_sync_c16_dn และวาดผังวงจรดังรูปที่ 4.43 ทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 โดยใช้ไฟล์ชื่อ ch4cxl_sync_c16_dn_tb1 และกำหนดค่าและ Waveform เมื่อ้อนการทดลองที่ 4.3.3 แล้วพิจารณาผล Behavioral simulation ที่ได้ว่าเป็นไปตามทฤษฎีหรือไม่



รูปที่ 4.43 ผังวงจรนับลงแบบชิงโครนัส

- b) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_3_4cxl จากนั้นนำไฟล์ร่างรูปที่ 4.43 ไม่โอนสเตเบิลชื่อ ch4cxl_debounce และไฟล์ร่างรูปที่ 4.44 ชื่อ ch4cxl_sync_c16_dn มาทำ Symbols และวาดผังวงจรดังรูปที่ 4.44



รูปที่ 4.44 วงจรทดลองรูปที่ 4.43 แบบนับลง 2 หลัก

การกำหนดขาสัญญาณต่างๆ ใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB2 เป็นอินพุต มี LED3-LED4 และเซเวนเซกเมนต์ DIGIT1 เป็นเอาต์พุต โดยใช้เซกเมนต์ a, b, c และ d ในการแสดงผล D0(3:0) และใช้เซกเมนต์ e, f, g และ dp ในการแสดงผล D1(3:0) ดังนี้ในผังวงจรจึงต้องต่อขาไปคร่าวม (COM) ของ DIGIT1 ลงกราวด์ด้วย กล่าวคือ

CLK = PB1 = INPUT = p39	D0(3) = d = OUTPUT = p24	D1(3) = dp = OUTPUT = p19
CLR_DB = PB2 = INPUT = p40	D0(2) = c = OUTPUT = p25	D1(2) = g = OUTPUT = p18
OSC = OSC = INPUT = p5	D0(1) = b = OUTPUT = p26	D1(1) = f = OUTPUT = p20
COM = DIGIT1 = OUTPUT = p34	D0(0) = a = OUTPUT = p27	D1(0) = e = OUTPUT = p22
CEO0 = LED4 = OUTPUT = p35	CEO1 = LED3 = OUTPUT = p36	

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

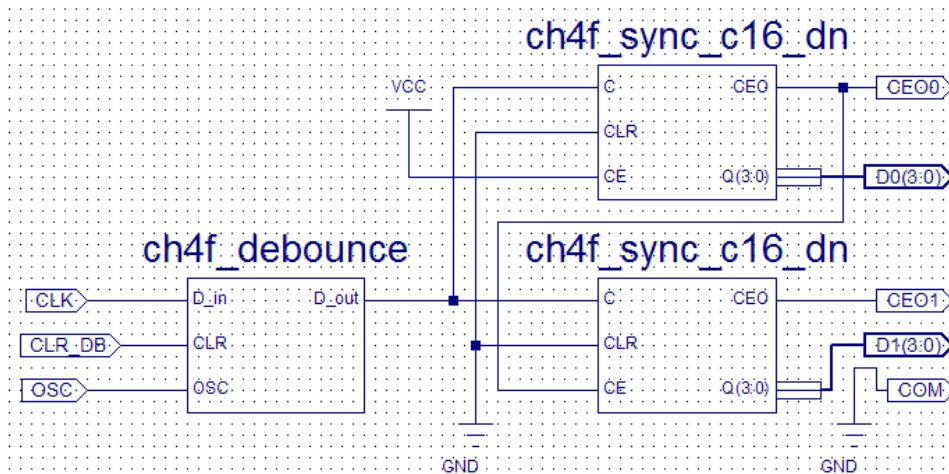
NET "CEO0" LOC = "p35" | SLEW = SLOW ;
NET "CEO1" LOC = "p36" | SLEW = SLOW ;
NET "CLK" LOC = "p39" ;
NET "CLR_DB" LOC = "p40" ;
NET "COM" LOC = "p34" | SLEW = SLOW ;
NET "DO<0>" LOC = "p27" | SLEW = SLOW ;
NET "DO<1>" LOC = "p26" | SLEW = SLOW ;
NET "DO<2>" LOC = "p25" | SLEW = SLOW ;
NET "DO<3>" LOC = "p24" | SLEW = SLOW ;
NET "D1<0>" LOC = "p22" | SLEW = SLOW ;
NET "D1<1>" LOC = "p20" | SLEW = SLOW ;
NET "D1<2>" LOC = "p18" | SLEW = SLOW ;
NET "D1<3>" LOC = "p19" | SLEW = SLOW ;
NET "OSC" LOC = "p5" ;

```

หลังจากโปรแกรมลง CPLD แล้ว ให้กดปุ่ม PB1 และ PB2 ลับกันไปเรื่อยๆ แล้วให้สังเกตคุณภาพที่ LED3-LED4 และ เช่วนเซกเมนต์ DIGIT1 ว่าให้หลอด桔าต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

2. สร้างวงจรนับลงแบบชิงโครนัสด้วย FPGA

- a) สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4f_sync_c16_dn และวัดผังวงจร ดังรูปที่ 4.43 จากนั้นทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 โดยใช้ไฟล์ชื่อ ch4f_sync_c16_dn_tb1 แล้วกำหนดค่า และ Waveform เมื่อ้อนการทดลองที่ 4.3.3 แล้วพิจารณาผล Behavioral simulation ที่ได้ว่าเป็นไปตามทฤษฎีหรือไม่
- b) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_3_4f จากนั้นให้นำไฟล์ วงจร โน้มโน่นสเกบเบิลชื่อ ch4f_debounce และไฟล์วงจรนับชื่อ ch4f_sync_c16_dn มาทำ Symbols และวัดผังวงจรดังรูปที่ 4.45



รูปที่ 4.45 วงจรทดสอบวงจรนับ 16 แบบนับลง 2 หลัก

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25 MHz และ ปุ่มกด PB1-PB2 เป็นอินพุต มี LED L0-L1 และเช่วนเซกเมนต์ DIGIT1 เป็นเอาต์พุต โดยใช้เซกเมนต์ a, b, c และ d ในการแสดงผล D0(3:0) และใช้เซกเมนต์ e, f, g และ dp ในการแสดงผล D1(3:0) ดังนั้นในผังวงจรจึงต้องต่อขามาโดยร่วม (COM) ของ DIGIT1 ลงกราวด์ด้วย กล่าวคือ

CLK = PB1 = INPUT = p44	D0(3) = d = OUTPUT = p30	D1(3) = dp = OUTPUT = p20
CLR_DB = PB2 = INPUT = p46	D0(2) = c = OUTPUT = p32	D1(2) = g = OUTPUT = p23
OSC = OSC = INPUT = p127	D0(1) = b = OUTPUT = p35	D1(1) = f = OUTPUT = p25
COM = DIGIT1 = OUTPUT = p31	D0(0) = a = OUTPUT = p40	D1(0) = e = OUTPUT = p27
CEO0 = L0 = OUTPUT = p70		
CEO1 = L1 = OUTPUT = p77		

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "CEO0" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "CEO1" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR_DB" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "COM" LOC = "p31" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D0<0>" LOC = "p40" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D0<1>" LOC = "p35" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D0<2>" LOC = "p32" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D0<3>" LOC = "p30" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<0>" LOC = "p27" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<1>" LOC = "p25" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<2>" LOC = "p23" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<3>" LOC = "p20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;

```

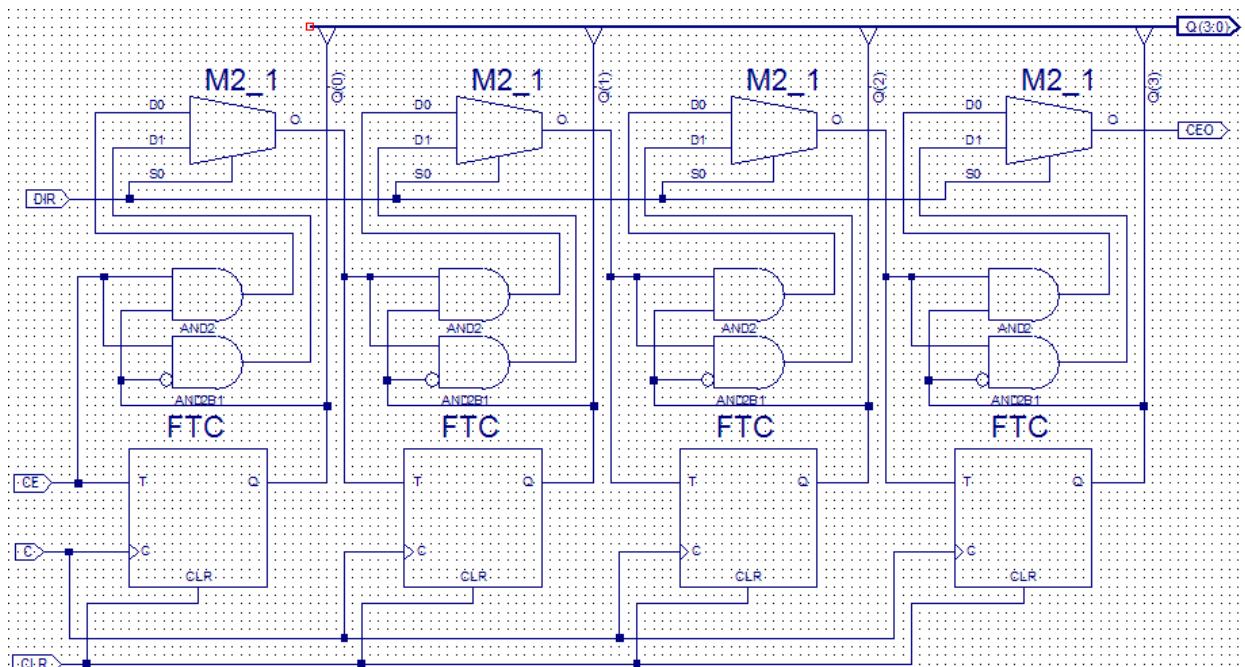
หลังจากโปรแกรมลง FPGA แล้ว ให้กดปุ่ม PB1 และ PB2 สลับกันไปเรื่อยๆ แล้วให้สังเกตคุณลักษณะ LED L0-L1 และ เช่วนเซกเมนต์ DIGIT1 ว่าแต่ละเซกเมนต์ให้ออกจิกอาที่พุดเป็นไปตามทฤษฎีหรือไม่ จากนั้นบันทึกผลการทดลอง

4.3.5 การออกแบบวงจรนับขีน-ลงแบบชิงโครนัส

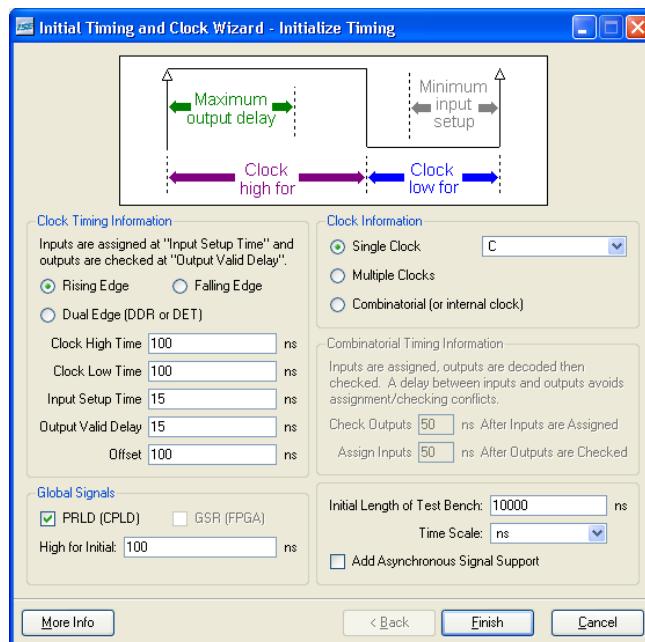
อุปกรณ์ที่ควรมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรนับขีน-ลงแบบชิงโกรนัส 4 บิตด้วย CPLD

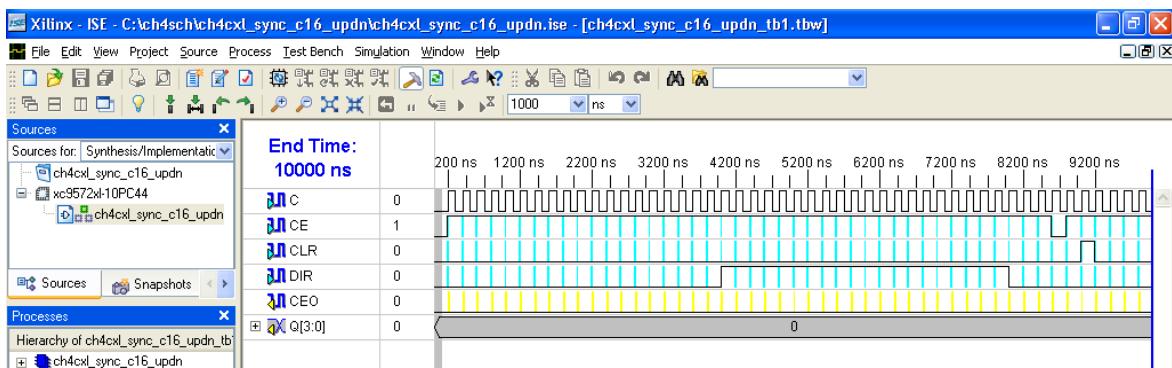
a) สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4cxl_sync_c16_updwn และวิธีผังวงจรนับขีน-ลงแบบชิงโกรนัส 4 บิต (4-bit bidirectional binary counter) ดังรูปที่ 4.46 โดยมีขา DIR ควบคุมการนับขึ้นนับลง ซึ่งจะนับขึ้นเมื่อ DIR = '0' และจะนับลงเมื่อ DIR = '1' จากนั้นให้ทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 โดยใช้ไฟล์ชื่อ ch4cxl_sync_c16_updwn_tb1 แล้วกำหนดค่าและ Waveform ดังรูปที่ 4.47 และรูปที่ 4.48 แล้วพิจารณาผล Behavioral simulation ในรูปที่ 4.49 ว่าเป็นไปตามทฤษฎีหรือไม่



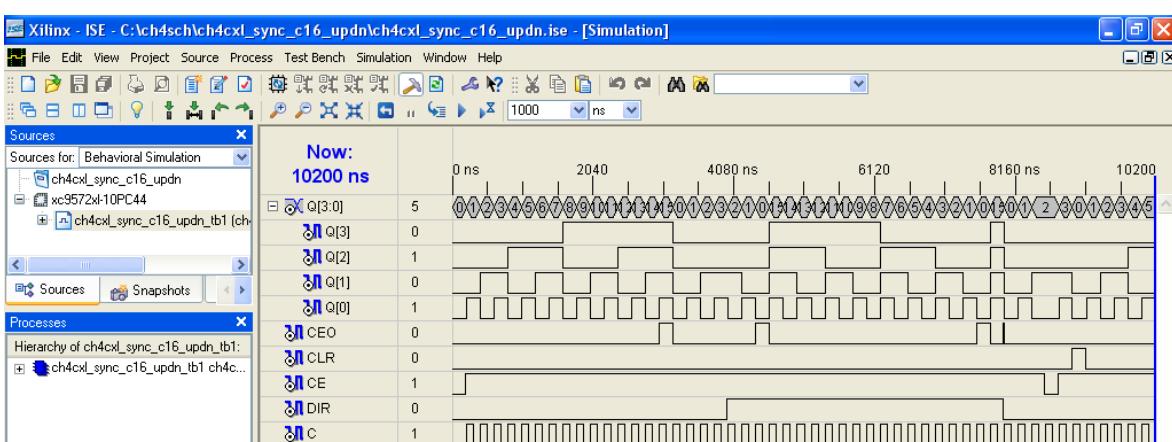
รูปที่ 4.46 ผังวงจรนับขีน-ลงแบบชิงโกรนัส 4 บิต



รูปที่ 4.47 หน้าต่าง Initial Timing and Clock Wizard



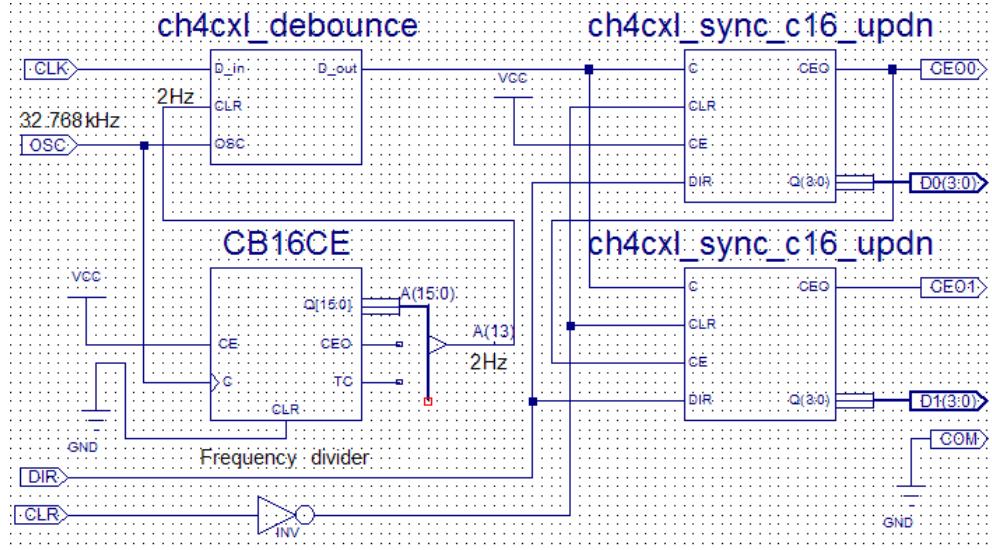
รูปที่ 4.48 หน้าต่างสำหรับกำหนดสัมบานด์ต่างๆ ที่ป้อนให้กับอินพุตของวงจรที่เราต้องการทดสอบ



รูปที่ 4.49 ผล Behavioral simulation ของวงจรนับขี๊บ-ลงแบบซิงโครนัส 4 บิต

b) สร้างไฟล์สำหรับทดสอบวงจรนับ FF (0-255) โดยใช้งานรับบินชื่อ-ลงแบบซิง โกรนัส 4 บิตจำนวน 2 หลักใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_3_5cxl และนำไฟล์ลงจริงในสตูเบิลชื่อ ch4cxl_debounce และไฟล์วงจรนับบินชื่อ-ลงแบบซิงโกรนัส 4 บิตชื่อ ch4cxl sync c16 updn ที่มีอยู่แล้วมาทำ Symbols จากนั้นนำวงจรดังรูปที่

4.50 ชิ้งวงจรนี้จะเคลียร์ค่าเอตพุตของจนับเป็น 00 (Hex) เมื่อ CLR = '0' และจะนับขึ้นเมื่อ DIR = '0' ถ้า DIR = '1' จะเป็นการนับลง ขอให้สังเกตว่าที่วงจรดีเบาเซอร์จะมีสัญญาณความถี่ 2 Hz (จากวงจร Frequency divider) ป้อนที่ขา CLR เพื่อเคลียร์เอตพุตของวงจรดีเบาเซอร์โดยอัตโนมัติทุกๆ $1 / (2 \text{ Hz}) = 0.5$ วินาที



รูปที่ 4.50 วงจรนับ FF (0-255) โดยใช้วงจรนับขึ้น-ลงแบบซิงโครนัส 4 บิตจำนวน 2 หลัก

การกำหนดขาสัญญาณต่างๆ ใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และบีบูมกค PB1-PB3 เป็นอินพุต มี LED3-LED4 และเซเวนเซกเมนต์ DIGIT1 เป็นเอาต์พุต โดยใช้เซกเมนต์ a, b, c และ d ในการแสดงผล D0(3:0) และใช้เซกเมนต์ e, f, g และ dp ในการแสดงผล D1(3:0) ดังนั้นในผังวงจรจึงต้องต่อขาโอดครัวว์ (COM) ของ DIGIT1 ลงกราวด์ด้วย กล่าวคือ

CLK = PB1 = INPUT = p39	D0(3) = d = OUTPUT = p24	D1(3) = dp = OUTPUT = p19
CLR = PB2 = INPUT = p40	D0(2) = c = OUTPUT = p25	D1(2) = g = OUTPUT = p18
OSC = OSC = INPUT = p5	D0(1) = b = OUTPUT = p26	D1(1) = f = OUTPUT = p20
DIR = PB3 = INPUT = p42	D0(0) = a = OUTPUT = p27	D1(0) = e = OUTPUT = p22
CEO0 = LED4 = OUTPUT = p35	CEO1 = LED3 = OUTPUT = p36	COM = DIGIT1 = OUTPUT = p34

โดยพิมพ์ใน Edit Constraints (Text) สรุปนี้

```

NET "CLK" LOC = "p39" ;
NET "CLR" LOC = "p40" ;
NET "COM" LOC = "p34" | SLEW = SLOW ;
NET "DO<0>" LOC = "p27" | SLEW = SLOW ;
NET "DO<1>" LOC = "p26" | SLEW = SLOW ;
NET "DO<2>" LOC = "p25" | SLEW = SLOW ;
NET "DO<3>" LOC = "p24" | SLEW = SLOW ;
NET "D1<0>" LOC = "p22" | SLEW = SLOW ;
NET "D1<1>" LOC = "p20" | SLEW = SLOW ;
NET "D1<2>" LOC = "p18" | SLEW = SLOW ;
NET "D1<3>" LOC = "p19" | SLEW = SLOW ;
NET "CEO0" LOC = "p35" | SLEW = SLOW ;
NET "CEO1" LOC = "p36" | SLEW = SLOW ;
NET "DIR" LOC = "p42" ;
NET "OSC" LOC = "p5" ;

```

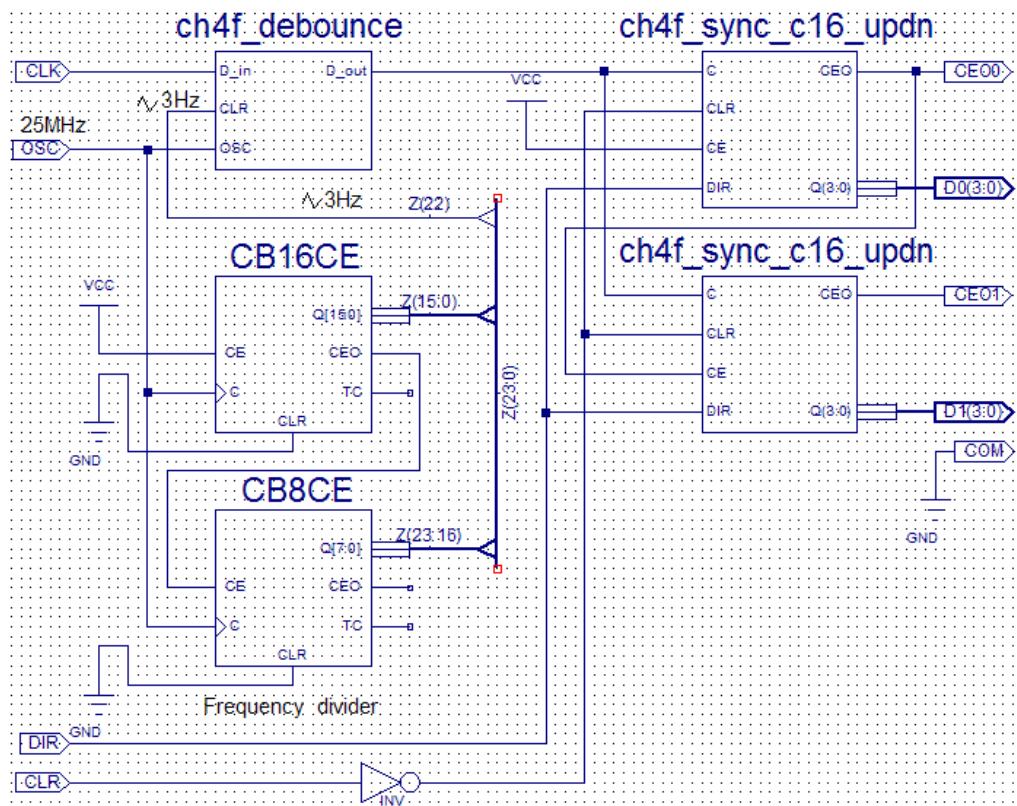
หลังจากโปรแกรมลง CPLD แล้ว ให้กดลง ON Slide SW1 (PB3) และกดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกดปุ่ม PB1 ค้างไว้แล้วให้สังเกตดูผลที่ LED3-LED4 และเซนเซอร์เมนต์ DIGIT1 ว่าแต่ละเซนเซอร์ที่ต้องการจะเป็นไปตามทฤษฎี

หรือไม่ จากนั้นให้กด PB2 แล้วสังเกตเซนเซอร์ DIGIT1 อีกครั้ง เสรีจแล้วให้ OFF Slide SW1 (PB3) แล้วทำการทดลองซ้ำอีกครั้ง ทำการบันทึกผลการทดลอง

2. สร้างวงจรรับสัญญาณแบบชิ้น-ลงแบบชิ้งครอนส์ 4 บิตด้วย FPGA

a) สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4f_sync_c16_updn และวัดผังวงจรดังรูปที่ 4.46 จากนั้นทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 โดยใช้ไฟล์ชื่อ ch4f_sync_c16_updn_tb1 แล้วกำหนดค่าและ Waveform เช่นเดียวกับที่ใช้รูปที่ 4.47 และรูปที่ 4.48 แล้วให้พิจารณาผล Behavioral simulation ของวงจรร่วมเป็นไปตามทฤษฎีหรือไม่

b) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_3_5f จากนั้นนำไฟล์วงจรโมโนสเตเบิล ชื่อ ch4f_debounce และไฟล์วงจรรับสัญญาณแบบชิ้น-ลงแบบชิ้งครอนส์ 4 บิต ชื่อ ch4f_sync_c16_updn ที่มีอยู่แล้วมาทำ Symbols จากนั้นทำการวัดผังวงจรดังรูปที่ 4.51 โดยจะป้อนความถี่ประมาณ 3 Hz (25MHz/2²³) จากการ Frequency divider เข้าที่ขาเคลียร์ (CLR) เพื่อเคลียร์เอาต์พุตของวงจรตีเบาเซอร์ได้ข้อต่อในมิติทุกๆ $1 / (3 \text{ Hz}) = 0.33$ วินาที



รูปที่ 4.51 วงจรรับ FF (0-255) โดยใช้งานรับสัญญาณแบบชิ้น-ลงแบบชิ้งครอนส์ 4 บิตจำนวน 2 หลัก

การกำหนดขาสัญญาณต่างๆ ใช้ออสซิลเลเตอร์ OSC = 25 MHz ปุ่มกด PB1-PB2 และ Dip SW1 เป็นอินพุต มี LED L0-L1 และเซนเซอร์ DIGIT1 เป็นเอาต์พุต โดยใช้เซกเมนต์ a, b, c และ d ในการแสดงผล D0(3:0) และใช้เซกเมนต์ e, f, g และ dp ในการแสดงผล D1(3:0) ดังนั้นในผังวงจรจึงต้องต่อขาโอดคร่าวม (COM) ของ DIGIT1 ลงกราวด์ด้วย กล่าวคือ

CLK = PB1 = INPUT = p44	D0(3) = d = OUTPUT = p30	D1(3) = dp = OUTPUT = p20
CLR = PB2 = INPUT = p46	D0(2) = c = OUTPUT = p32	D1(2) = g = OUTPUT = p23
DIR = Dip SW1 = INPUT = p52	D0(1) = b = OUTPUT = p35	D1(1) = f = OUTPUT = p25
OSC = OSC = INPUT = p127	D0(0) = a = OUTPUT = p40	D1(0) = e = OUTPUT = p27
CEO0 = L0 = OUTPUT = p70	CEO1 = L1 = OUTPUT = p77	COM = DIGIT1 = OUTPUT = p31

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "CEO0" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "CEO1" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "COM" LOC = "p31" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<0>" LOC = "p40" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<1>" LOC = "p35" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<2>" LOC = "p32" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<3>" LOC = "p30" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<0>" LOC = "p27" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<1>" LOC = "p25" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<2>" LOC = "p23" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<3>" LOC = "p20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DIR" LOC = "p52" | IOSTANDARD = LVCMOS33 ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;

```

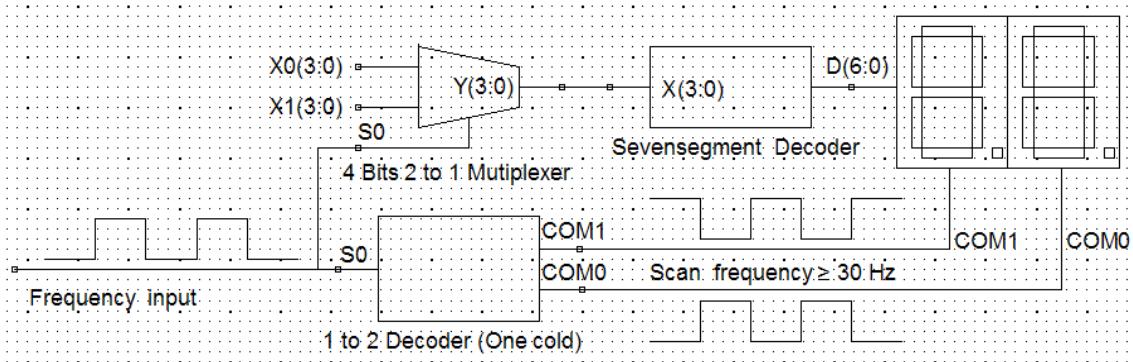
หลังจากโปรแกรมลง FPGA แล้ว ให้ทดสอบ ON Dip SW1 แล้วกดปุ่ม PB1 ไปเรื่อยๆ ลับกันการกดปุ่ม PB1 ค้างไว้ แล้วให้สังเกตดูผลที่ LED L0-L1 และเซ็นเซอร์menต์ DIGIT1 ว่าแต่ละเซกเมนต์ให้กลับเข้าตัวพุตเป็นไปตามทฤษฎีหรือไม่ หากนั้น OFF Dip SW1 แล้วทำการทดสอบซ้ำอีกครั้ง ทำการบันทึกผลการทดสอบ

4.3.6 การออกแบบวงจรนับขีน-ลง 2 หลัก (0-FF) ที่แสดงผลทางเลขฐานเทกเมนต์

อุปกรณ์ที่นำมาใช้เพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรนับขีน-ลง 4 บิต 2 หลักที่แสดงผลทางเลขฐานเทกเมนต์ด้วย CPLD

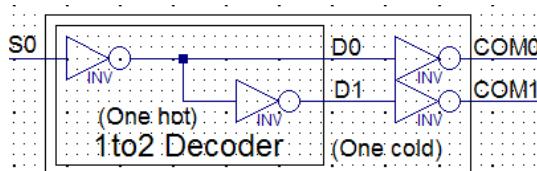
- a) วงจรสแกนและวงจรอุดรหัสตัวแสดงผลทางเลขฐานเทกเมนต์ (แบบแอดดิชันล์) 2 หลักแสดงดังรูปที่ 4.52 จะประกอบด้วย วงจรอุดรหัส (1 to 2 Decoder) แบบ Active low คือให้อ่าต์พูดเป็นโลจิก ‘0’ (One cold Decoder) แสดงดังรูปที่ 4.53 วงจร มัลติเพล็กซ์อร์ (2 to 1 Multiplexer) ขนาด 4 บิตดังรูปที่ 4.54 และวงจรอุดรหัสตัวแสดงผลทางเลขฐานเทกเมนต์ดังรูปที่ 4.55



รูปที่ 4.52 วงจรสแกนและอุดรหัสตัวแสดงผลทางเลขฐานเทกเมนต์ 2 หลัก

Input	Output			
	D0	D1	COM0	COM1
0	1	0	0	1
1	0	1	1	0

(a) ตารางความจริงวงจรอุดรหัส

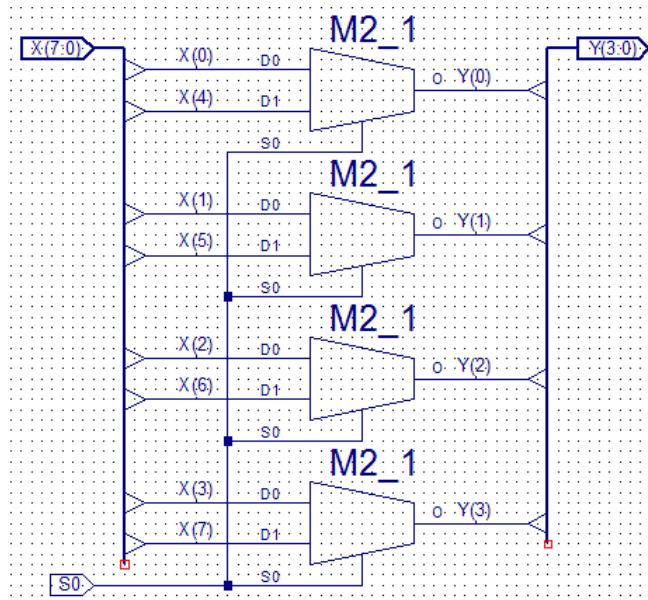


(b) ผังวงจรอุดรหัส (ยังไม่ลดรูปเพื่อทำให้เข้าใจง่าย)

รูปที่ 4.53 วงจรอุดรหัส (1 to 2 Decoder) แบบให้อ่าต์พูดเป็นโลจิก ‘0’

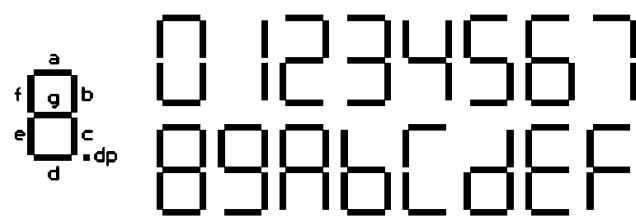
Control S0	Output
0	$Y(3:0) = X(3:0) = X_0(3:0) = Z(3:0)$
1	$Y(3:0) = X(7:4) = X_1(3:0) = Z(7:4)$

(a) ตารางความจริงของมัลติเพล็กเซอร์



(b) ผังวงจร มัลติเพล็กเซอร์

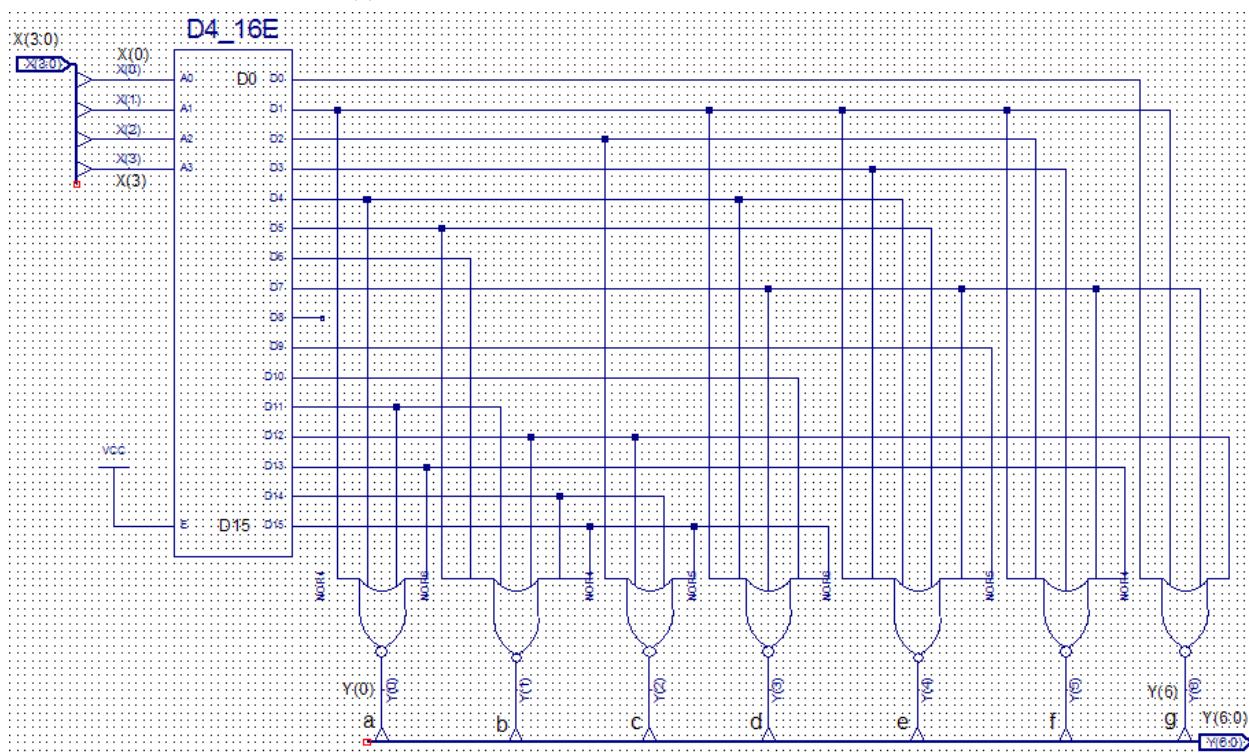
รูปที่ 4.54 วงจร มัลติเพล็กเซอร์ (2 to 1 Multiplexer) ขนาด 4 บิต



(a) Hexadecimal digit display format

No.	Input X(3:0)				Output Y(6:0)							
	X(3)	X(2)	X(1)	X(0)	Y(6)=g	Y(5)=f	Y(4)=e	Y(3)=d	Y(2)=c	Y(1)=b	Y(0)=a	
0	0	0	0	0	0	1	1	1	1	1	1	
1	0	0	0	1	0	0	0	0	1	1	0	
2	0	0	1	0	1	0	1	1	0	1	1	
3	0	0	1	1	1	0	0	1	1	1	1	
4	0	1	0	0	1	1	0	0	1	1	0	
5	0	1	0	1	1	1	0	1	1	0	1	
6	0	1	1	0	1	1	1	1	1	0	1	
7	0	1	1	1	0	0	0	0	1	1	1	
8	1	0	0	0	1	1	1	1	1	1	1	
9	1	0	0	1	1	1	0	1	1	1	1	
10 = A	1	0	1	0	1	1	1	0	1	1	1	
11 = b	1	0	1	1	1	1	1	1	1	0	0	
12 = C	1	1	0	0	0	1	1	1	0	0	1	
13 = d	1	1	0	1	1	0	1	1	1	1	0	
14 = E	1	1	1	0	1	1	1	1	0	0	1	
15 = F	1	1	1	1	1	1	1	0	0	0	1	

(b) ตารางความจริงวงจรอдрหัสตัวแสดงผลเลขฐานเซกเมนต์



(c) ผังวงจรอдрหัสตัวแสดงผลเลขฐานเซกเมนต์แบบแคนโอดร่วม

รูปที่ 4.55 วงจรอдрหัสตัวแสดงผลเลขฐานเซกเมนต์แบบแคนโอดร่วม (Hexadecimal digit display format)

หลักการทำงานของแกนเป็นดังนี้คือ (ให้พิจารณาครุภัที่ 4.56 ด้วย)

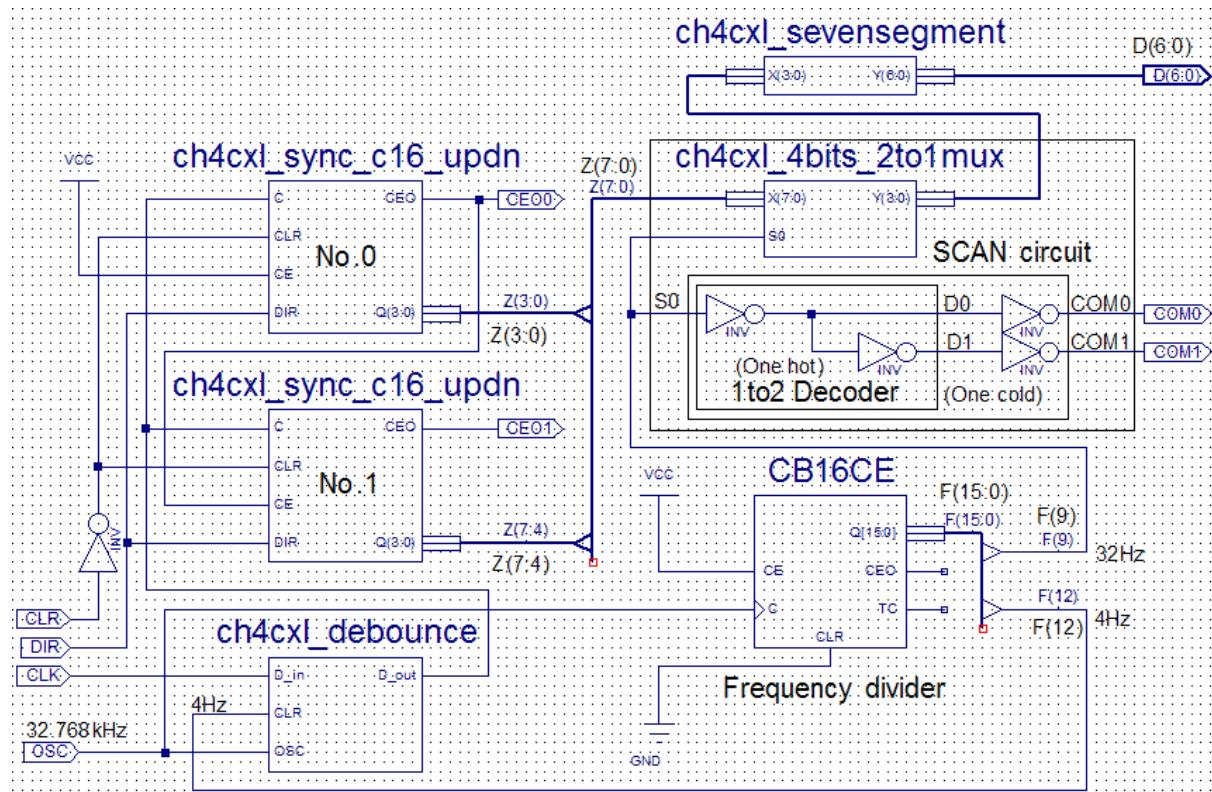
1) เมื่อ $S_0 = '0'$ วงจรแมตติเพลกิเซอร์จะเลือกสัญญาณ $X_0(3:0) = Z(3:0)$ ขนาด 4 บิตส่งไปที่วงจรกรหัสตัวแสดงผล เช่วนเซกเมนต์เพื่อไปขับตัวแสดงผลเช่วนเซกเมนต์ทั้ง 2 หลัก ในขณะเดียวกันวงจร One cold Decoder จะผลครหัสให้ $COM_0 = '0'$ และ $COM_1 = '1'$ ทำให้ตัวแสดงผลเช่วนเซกเมนต์หลักหน่วยติดสว่างเพียงหลักเดียว

2) เมื่อ $S_0 = '1'$ วงจรแมตติเพลกิเซอร์จะเลือกสัญญาณ $X_1(3:0) = Z(7:4)$ ขนาด 4 บิตส่งไปที่วงจรกรหัสตัวแสดงผล เช่วนเซกเมนต์เพื่อไปขับตัวแสดงผลเช่วนเซกเมนต์ทั้ง 2 หลัก ในขณะเดียวกันวงจร One cold Decoder จะผลครหัสให้ $COM_0 = '1'$ และ $COM_1 = '0'$ ทำให้ตัวแสดงผลเช่วนเซกเมนต์หลักสับติดสว่างเพียงหลักเดียว

3) ความถี่ที่ป้อนให้ S_0 หรือความถี่ที่ใช้แกนที่ขาเคนโดยร่วมของเช่วนเซกเมนต์แต่ละหลักต้องไม่น้อยกว่า 30 ครั้ง ต่อวินาทีจะทำให้ตัวแสดงผลเช่วนเซกเมนต์ติดสว่างทั้ง 2 หลักโดยมองไม่เห็นการกระพริบ

b) ขั้นตอนทำ Symbols ให้นำรูปที่ 4.54(b) และรูปที่ 4.55(b) มาสร้างไฟล์สำหรับทำ Symbols ไว้ใน Project Location ชื่อ ch4sch โดยกำหนด Project Name และ Source File ชื่อ ch4cxl_4bits_2to1mux และ ch4cxl_sevensegment ตามลำดับ

c) สร้างไฟล์วงจรรับขึ้น-ลงแบบชิงโกรนัส 4 บิต 2 หลักใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_3_6cxl และนำไฟล์วงจรโมโนสเตเบิล ชื่อ ch4cxl_debounce ไฟล์วงจรกรหัสตัวแสดงผลเช่วนเซกเมนต์ ชื่อ ch4cxl_sevensegment ไฟล์วงจรแมตติเพลกิเซอร์ (2 to 1 Multiplexer) ขนาด 4 บิต ชื่อ ch4cxl_4bits_2to1mux และไฟล์วงจรนับขึ้น-ลงแบบชิงโกรนัส 4 บิต ชื่อ ch4cxl_sync_c16_updn (ที่มีอยู่แล้ว) มาทำ Symbols จากนั้นวดผังวงจรดังรูปที่ 4.56 ซึ่งวงจรนี้จะเคลียร์ค่าต่ำสุดของวงจรรับขึ้นเมื่อ CLR = '0' และจะนับขึ้นเมื่อ DIR = '0' แต่ถ้า DIR = '1' จะเป็นการนับลง ขอให้สังเกตว่าที่วงจรโมโนสเตเบิลในการทดลองนี้จะมีสัญญาณความถี่ 4 Hz (จากการ Frequency divider) ป้อนที่ขา CLR เพื่อเคลียร์ค่าต่ำสุดของวงจรรับขึ้นเมื่อ CLR = '0' และจะนับขึ้นเมื่อ DIR = '0' แต่ถ้า DIR = '1' จะเป็นการนับลง 32Hz



รูปที่ 4.56 วงจรรับ FF (0-255) โดยใช้งานขึ้น-ลงแบบชิงโกรนัส 4 บิตจำนวน 2 หลัก

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB3 เป็นอินพุต LED3-LED4 และตัวแสดงผลเซเว่นเซกเมนต์ DIGIT1-DIGIT2 จึงต้องใช้ขาไอคอดร่วม COM0 และ COM1 ด้วย กล่าวคือ

CLK = PB1 = INPUT = p39	D(0) = a = OUTPUT = p27	COM0 = DIGIT1 = OUTPUT = p34
CLR = PB2 = INPUT = p40	D(1) = b = OUTPUT = p26	COM1 = DIGIT2 = OUTPUT = p33
OSC = OSC = INPUT = p5	D(2) = c = OUTPUT = p25	
DIR = PB3 = INPUT = p42	D(3) = d = OUTPUT = p24	
CEO0 = LED4 = OUTPUT = 35	D(4) = e = OUTPUT = p22	
CEO1 = LED3 = OUTPUT = p36	D(5) = f = OUTPUT = p20	
	D(6) = g = OUTPUT = p18	

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

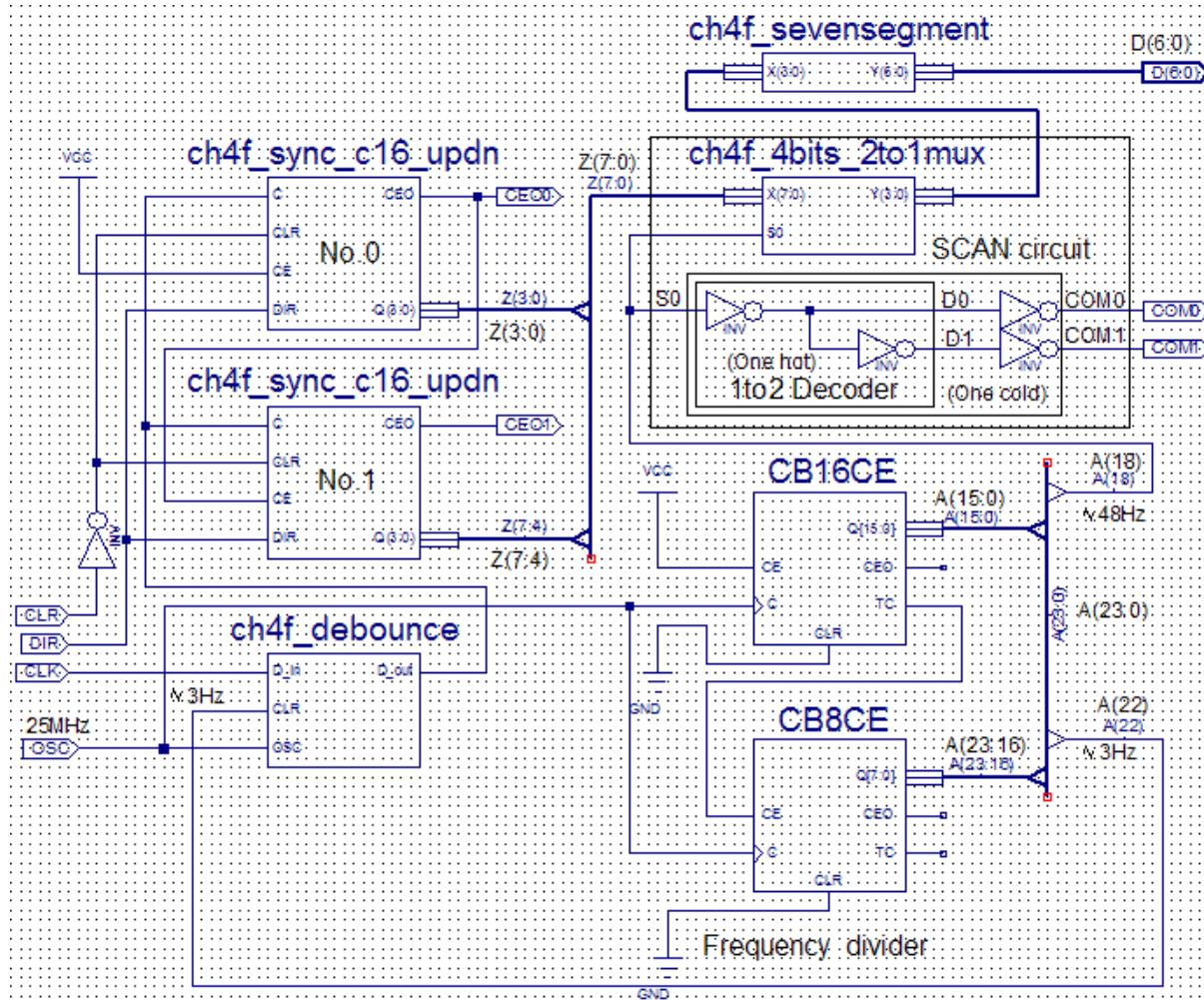
NET "CLK" LOC = "p39" ;
NET "CLR" LOC = "p40" ;
NET "CEO0" LOC = "p35" | SLEW = SLOW ;
NET "CEO1" LOC = "p36" | SLEW = SLOW ;
NET "COM0" LOC = "p34" | SLEW = SLOW ;
NET "COM1" LOC = "p33" | SLEW = SLOW ;
NET "D<0>" LOC = "p27" | SLEW = SLOW ;
NET "D<1>" LOC = "p26" | SLEW = SLOW ;
NET "D<2>" LOC = "p25" | SLEW = SLOW ;
NET "D<3>" LOC = "p24" | SLEW = SLOW ;
NET "D<4>" LOC = "p22" | SLEW = SLOW ;
NET "D<5>" LOC = "p20" | SLEW = SLOW ;
NET "D<6>" LOC = "p18" | SLEW = SLOW ;
NET "DIR" LOC = "p42" ;
NET "OSC" LOC = "p5" ;

```

หลังจากโปรแกรมลง CPLD แล้วให้กดลง ON Slide SW1 (PB3) แล้วกดปุ่ม PB1 ไปเรื่อยๆ สนับสนุนการกดปุ่ม PB1 ทางไฟแล้วให้สังเกตคุณลักษณะที่ LED3-LED4 เซเว่นเซกเมนต์ DIGIT1 และ DIGIT2 ว่าแต่ละเซกเมนต์ให้ล็อกเอาต์พุดเป็นไปตามทุยกฎหรือไม่ หากนั้นให้กด PB2 แล้วสังเกตเซเว่นเซกเมนต์ DIGIT1 และ DIGIT2 อีกครั้ง เสรีชแล้วให้ OFF Slide SW1 (PB3) แล้วทำการทดลองซ้ำอีกครั้ง ทำการบันทึกผลการทดลอง

2. สร้างวงจรนับขึ้น-ลง 4 บิต 2 หลักที่แสดงผลทางเซเว่นเซกเมนต์ด้วย FPGA

- a) นำผังวงจรในรูปที่ 4.54(b) และรูปที่ 4.55(c) มาสร้างไฟล์สำหรับทำ Symbols ไว้ใน Project Location ชื่อ ch4sch โดยกำหนด Project Name และ Source File ชื่อ ch4f_4bits_2to1mux และ ch4f_sevensegment ตามลำดับ
- b) สร้างไฟล์สำหรับทดสอบวงจรนับ FF (0-255) โดยใช้งานรับขึ้น-ลงแบบชิงโครนัส 4 บิตใน Project Location ชื่อ ch4sch จากนั้นกำหนด Project Name และ Source File ชื่อ ex4_3_6f และนำไฟล์วงจรโมโนสเตเบิลชื่อ ch4f_debounce ไฟล์วงจร มัลติเพล็กเซอร์ (2 to 1 Multiplexer) ขนาด 4 บิต ชื่อ ch4f_4bits_2to1mux ไฟล์วงจรอัตราตัวแสดงผลเซเว่นเซกเมนต์ ชื่อ ch4f_sevensegment และไฟล์วงจรนับขึ้น-ลงแบบชิงโครนัส 4 บิต ชื่อ ch4f_sync_c16_updn ที่มีอยู่แล้วมาทำ Symbols จากนั้น วัดผังวงจรดังรูปที่ 4.57 ซึ่งวงจรนี้จะเคลียร์ค่าเอาต์พุตวงจรนับเป็น 00 (Hex) เมื่อ CLR = '0' และจะนับขึ้นเมื่อ DIR = '0' ถ้า DIR = '1' จะเป็นการนับลง ขอให้สังเกตว่าจะมีสัญญาณความถี่ประมาณ 3 Hz ($25\text{MHz}/2^{23}$ จากวงจร Frequency divider) ป้อนที่ขา CLR เพื่อเคลียร์เอาต์พุตของวงจร โมโนสเตเบิล โดยอัตโนมัติกๆ $1/(3\text{Hz}) = 0.33$ วินาที



ຮູບທີ 4.57 ວົງຈຽນນັບ FF (0-255) ໂດຍໃຊ້ວັງຈຽນນັບຂຶ້ນ-ລົງແບນບົງໂຄຣນັສ 4 ບົດຈຳນວນ 2 ລັກ

ການກຳຫຼາດຂາສັ້ນຢູ່ມາຕ່າງໆ ຈະໃຊ້ອຟສີລິເລເຕେຣ໌ OSC = 25 MHz ປຸ່ມກົດ PB1-PB2 ແລະ Dip SW1 ເປັນອິນພູດ LED L0-L1 ແລະ ຕັ້ງແສດງຜົດເຊວນເຊັກເນັດ DIGIT1-DIGIT2 ເປັນອາຕີພູດ ຈຶ່ງຕ້ອງໃຫ້ຂາຄາໂຄດຮ່ວມ COM0 ແລະ COM1 ດັວກລ່າງຄືອ

CLK = PB1 = INPUT = p44

D(0) = a = OUTPUT = p40

COM0 = DIGIT1 = OUTPUT = p31

CLR = PB2 = INPUT = p46

D(1) = b = OUTPUT = p35

COM1 = DIGIT2 = OUTPUT = p33

OSC = OSC = INPUT = p127

D(2) = c = OUTPUT = p32

DIR = Dip SW1 = INPUT = p52

D(3) = d = OUTPUT = p30

CEO0 = L0 = OUTPUT = p70

D(4) = e = OUTPUT = p27

CEO1 = L1 = OUTPUT = p77

D(5) = f = OUTPUT = p25

D(6) = g = OUTPUT = p23

ໂດຍພິມພື້ນໃນ Edit Constraints (Text) ສຽງປັດ້ງນີ້

```

NET "CEO0" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "CEO1" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "COM0" LOC = "p31" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "COM1" LOC = "p33" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<0>" LOC = "p40" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<1>" LOC = "p35" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<2>" LOC = "p32" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<3>" LOC = "p30" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<4>" LOC = "p27" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<5>" LOC = "p25" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<6>" LOC = "p23" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DIR" LOC = "p52" | IOSTANDARD = LVCMOS33 ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;

```

หลังจากโปรแกรมลง FPGA แล้วให้ทดสอบ ON Dip SW1 แล้วกดปุ่ม PB1 ไปเรื่อยๆ ลักษณะการกดปุ่ม PB1 ค้างไว้แล้วให้สังเกตคุณลักษณะที่ LED L0-L1 เช่นเชกเมนต์ DIGIT1 และ DIGIT2 ว่าแต่ละเชกเมนต์ให้ออกจิกເອາຕ์พุดเป็นไปตามทฤษฎีหรือไม่ หากนั้นให้กด PB2 แล้วสังเกตเช่นเชกเมนต์ DIGIT1 และ DIGIT2 อีกครั้ง เสร็จแล้วให้ OFF Dip SW1 แล้วทำการทดสอบซ้ำอีกครั้ง ทำการบันทึกผลการทดสอบ

4.3.7 การออกแบบวงจรนับขึ้น-ลง 4 หลัก (0xFFFF) ที่แสดงทางเชวนเซกเมนต์

อุปกรณ์ที่ควรมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรนับขึ้น-ลง 4 หลักที่แสดงทางเชวนเซกเมนต์ด้วย CPLD

a) ออกแบบวงจรแกนและถอดรหัสตัวแสดงผลทางเชวนเซกเมนต์ 4 หลัก หลักการทำงานของวงจรแกนเพื่อแสดงผลตัวเลข 4 หลักทางเชวนเซกเมนต์ (แบบคอมมอนแคร็ต) ดังรูปที่ 4.58 โดยที่วงจรแกนประกอบด้วย วงจรถอดรหัส (2 to 4 Decoder) แบบให้อ่าต์พุตเป็นล็อกจิก '0' (One cold Decoder) วงรัมลติเพล็กเซอร์ (4 to 1 Multiplexer) ขนาด 4 บิต และวงจรถอดรหัสตัวแสดงผลทางเชวนเซกเมนต์ รายละเอียดผังวงจรถอดรหัสและวงรัมลติเพล็กเซอร์แสดงดังรูปที่ 4.59(b) และรูปที่ 4.60(b) ตามลำดับ โดยหลักการทำงานวงจรแกนเป็นดังนี้คือ

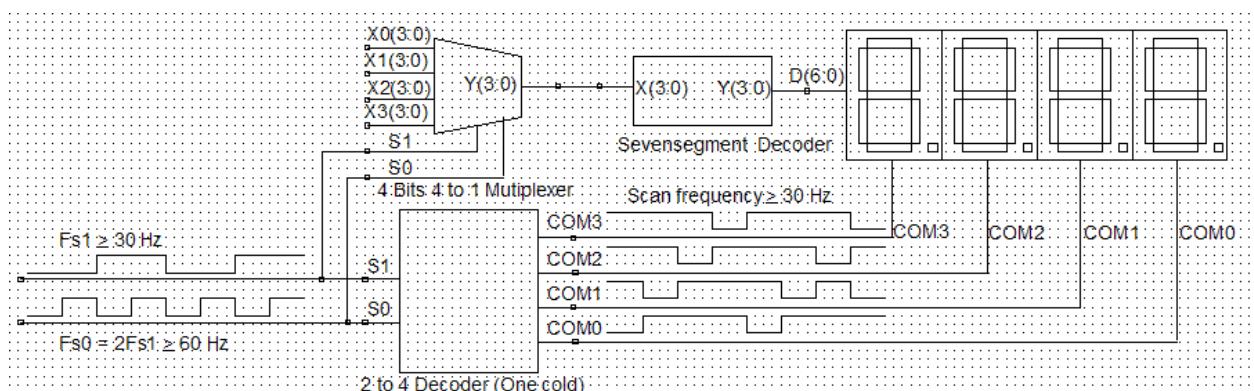
เมื่อ $S1S0 = "00"$ วงรัมลติเพล็กเซอร์จะเลือกสัญญาณ $X0(3:0)$ ขนาด 4 บิตไปที่อ่าต์พุต และขา COM0 = '0'

เมื่อ $S1S0 = "01"$ วงรัมลติเพล็กเซอร์จะเลือกสัญญาณ $X0(7:4)$ ขนาด 4 บิตไปที่อ่าต์พุต และขา COM1 = '0'

เมื่อ $S1S0 = "10"$ วงรัมลติเพล็กเซอร์จะเลือกสัญญาณ $X0(11:8)$ ขนาด 4 บิตไปที่อ่าต์พุต และขา COM2 = '0'

เมื่อ $S1S0 = "11"$ วงรัมลติเพล็กเซอร์จะเลือกสัญญาณ $X0(15:12)$ ขนาด 4 บิตไปที่อ่าต์พุต และขา COM3 = '0'

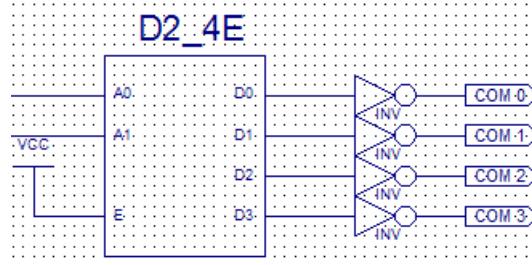
ความถี่ที่ใช้สแกนที่ขาคอมมอน (COM) ของแต่ละหลัก (เป็นล็อกจิก '0') เพื่อทำให้เชวนเซกเมนต์ติดสว่างโดยไม่เห็นการกระพริบนั้นต้องไม่น้อยกว่า 30 ครั้งต่อวินาที ดังนั้นความถี่ที่ป้อนให้ $S1$ และ $S0$ จึงต้องไม่น้อยกว่า 30 และ 60 ครั้งต่อวินาทีตามลำดับ จากนั้นรูปที่ 4.60 มาสร้างไฟล์สำหรับทำ Symbols ไว้ใน Project Location ชื่อ ch4sch โดยกำหนด Project Name และ Source File ชื่อ ch4cxl_4bits_4to1mux



รูปที่ 4.58 วงจรแกนและถอดรหัสตัวแสดงผลทางเชวนเซกเมนต์ 4 หลัก

S1 = A1	S0 = A0	COM0	COM1	COM2	COM3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

(a) ตารางความจริงของ decoder 2 to 4 (One cold Decoder)

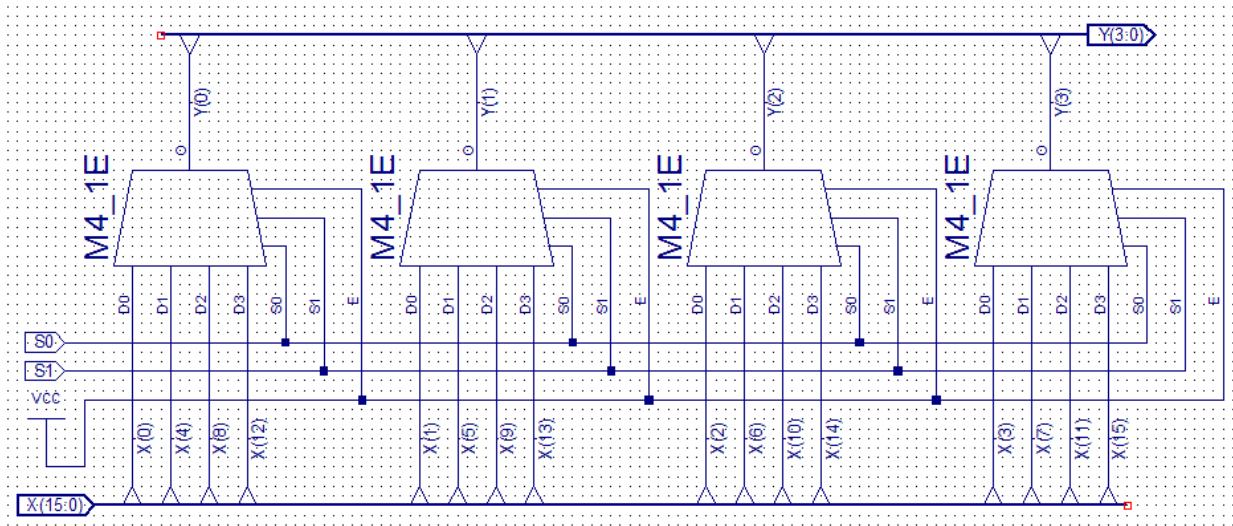


(b) ผังวงจร decoder 2 to 4 (One cold Decoder)

รูปที่ 4.59 วงจร decoder 2 to 4 (2 to 4 Decoder) แบบให้อเอต์พุดเป็นลอจิก ‘0’ (One cold Decoder)

S1	S0	Output
0	0	$Y(3:0) = X(3:0) = X0(3:0) = Z(3:0)$
0	1	$Y(3:0) = X(7:4) = X1(3:0) = Z(7:4)$
1	0	$Y(3:0) = X(11:8) = X2(3:0) = Z(11:8)$
1	1	$Y(3:0) = X(15:12) = X3(3:0) = Z(15:12)$

(a) ตารางความจริงของมัลติเพล็กเซอร์ 4 to 1 Multiplexer) ขนาด 4 บิต

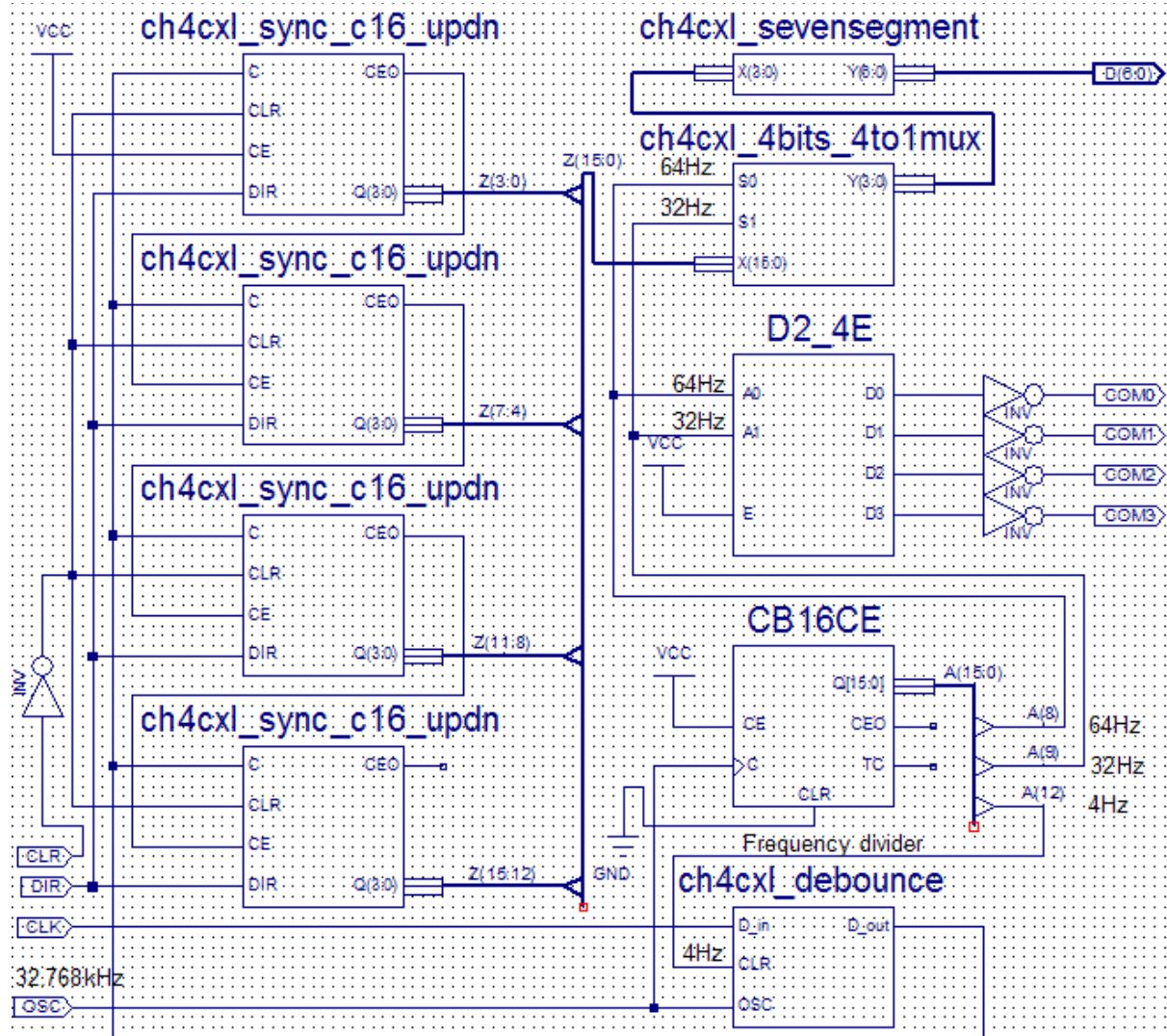


(b) ผังวงจร

รูปที่ 4.60 วงจร มัลติเพล็กเซอร์ (4 to 1 Multiplexer) ขนาด 4 บิต

b) สร้างไฟล์สำหรับทดสอบวงจรนับ FFFF โดยใช้วงจรนับขึ้น-ลงแบบซิงโกรนัส 4 บิต 4 หลักใน Project Location ชื่อ ch4sch จากนั้นกำหนด Project Name และ Source File ชื่อ ex4_3_7cxl และนำไฟล์ของวงจรโน้มโนสเกเบิล ชื่อ ch4cxl_debounce ไฟล์ของวงจรdecoder และวงจร มัลติเพล็กเซอร์ ชื่อ ch4cxl_4bits_4to1mux และไฟล์ของวงจรนับขึ้น-ลงแบบซิงโกรนัส 4 บิต ชื่อ ch4cxl_sync_c16_updn มาทำ Symbols จากนั้นกดผังวงจรดังรูปที่ 4.61 ซึ่งวงจรนี้จะเคลียร์ค่าเอต์พุดวงจรนับเป็น 0000

(Hex) เมื่อ CLR = ‘0’ และจะนับขึ้นเมื่อ DIR = ‘0’ ถ้า DIR = ‘1’ จะเป็นการนับลง ขอให้สังเกตว่าที่วงจรไมโครคอนโทรลเลอร์นี้มีสัญญาณความถี่ 4 Hz (จากวงจร Frequency divider) ป้อนที่ขา CLR เพื่อเคลียร์เอาต์พุตของวงจรไมโครคอนโทรลเลอร์โดยอัตโนมัติ ทุกๆ $1 / (4 \text{ Hz}) = 0.25$ วินาที สำหรับความถี่ที่ใช้สแกนแต่ละหลักจะใช้ความถี่ 32 Hz (อย่างน้อย 30Hz เพื่อไม่ให้เห็นการกระพริบ)



รูปที่ 4.61 วงจรนับ FFFF โดยใช้วงจรนับขึ้น-ลงแบบซิง โครนัส 4 บิตจำนวน 4 หลัก

การกำหนดขาสัญญาณต่างๆ ใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB3 เป็นอินพุต มีเชลล์เซกเม้นต์ DIGIT1-DIGIT4 ในการแสดงผลผ่านทาง D(6:0) โดยใช้เชลล์เซกเม้นต์ a, b, c, d, e, f และ g ดังนี้ในผังวงจรจึงต้องต่อขาค่าโอดร์ว์ว DIGIT1(SEG4)-DIGIT4(SEG1) ในการสแกน กันไว้คือ

CLK = PB1 = INPUT = p39

D(0) = a = OUTPUT = p27

COM0 = DIGIT1 = OUTPUT = p34

CLR = PB2 = INPUT = p40

D(1) = b = OUTPUT = p26

COM1 = DIGIT2 = OUTPUT = p33

OSC = OSC = INPUT = p5

D(2) = c = OUTPUT = p25

COM2 = DIGIT2 = OUTPUT = p29

DIR = PB3 = INPUT = p42

D(3) = d = OUTPUT = p24

COM3 = DIGIT2 = OUTPUT = p28

D(4) = e = OUTPUT = p22

D(5) = f = OUTPUT = p20

D(6) = g = OUTPUT = p18

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "CLK" LOC = "p39" ;
NET "CLR" LOC = "p40" ;
NET "COM0" LOC = "p34" | SLEW = SLOW ;
NET "COM1" LOC = "p33" | SLEW = SLOW ;
NET "COM2" LOC = "p29" | SLEW = SLOW ;
NET "COM3" LOC = "p28" | SLEW = SLOW ;
NET "D<0>" LOC = "p27" | SLEW = SLOW ;
NET "D<1>" LOC = "p26" | SLEW = SLOW ;
NET "D<2>" LOC = "p25" | SLEW = SLOW ;
NET "D<3>" LOC = "p24" | SLEW = SLOW ;
NET "D<4>" LOC = "p22" | SLEW = SLOW ;
NET "D<5>" LOC = "p20" | SLEW = SLOW ;
NET "D<6>" LOC = "p18" | SLEW = SLOW ;
NET "DIR" LOC = "p42" ;
NET "OSC" LOC = "p5" ;

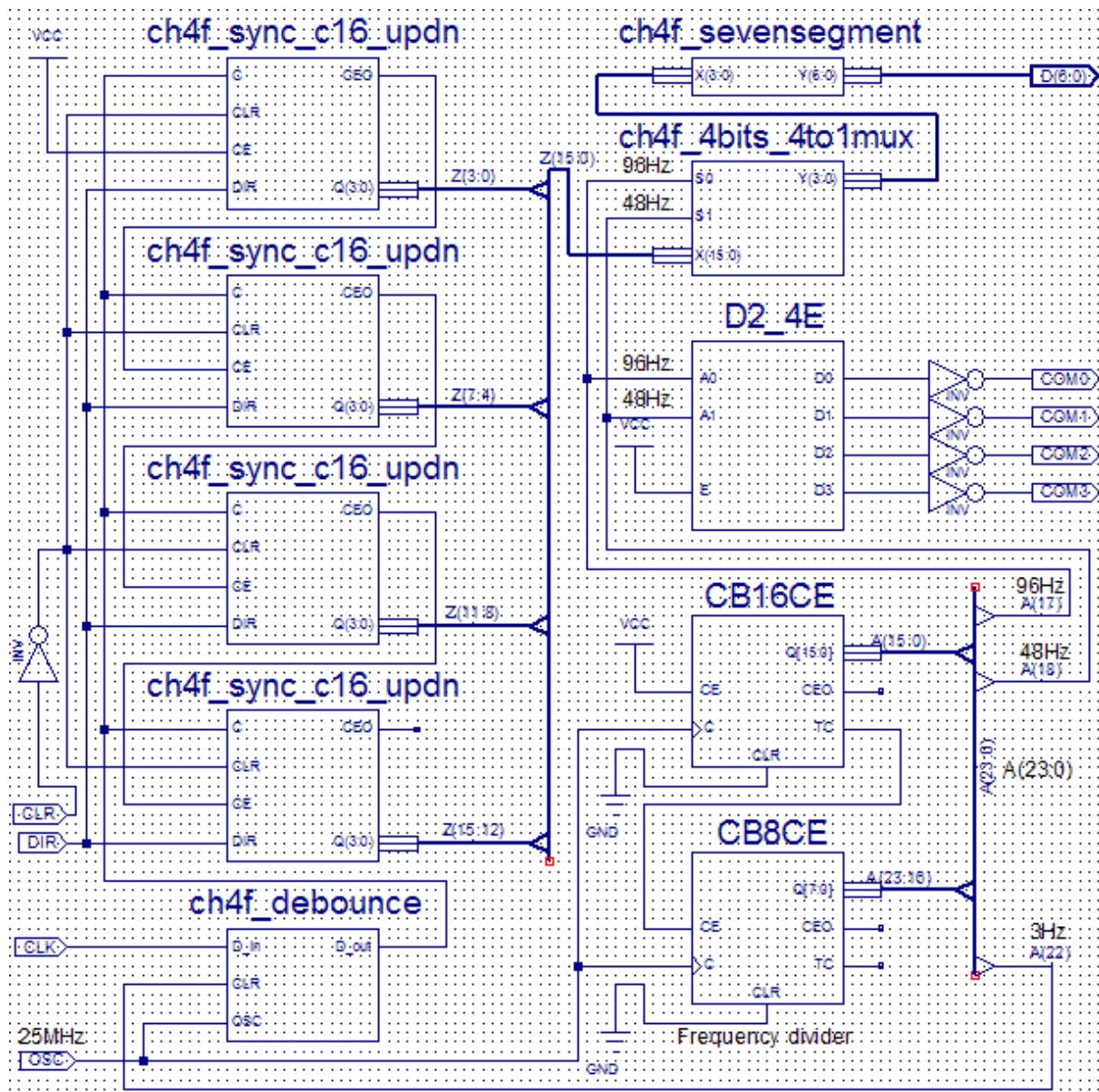
```

หลังจากโปรแกรมวงจรที่ออกแบบด้วย CPLD แล้วให้ทดลอง ON Slide SW1 (PB3) แล้วกดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกดปุ่ม PB1 ถ้าจะไว้แล้วให้สังเกตคุณลักษณะที่ LED3-LED4 เท่านั้นจะเป็น DIGIT1-DIGIT4 ว่าแต่ละเซกเมนต์ติดสว่างโดยให้ลองจิกເອົາຕີພຸດເປັນໄປຕາມທຸກໆຢູ່ໃໝ່ จากนັ້ນให้กด PB2 แล้วสังเกตເວລາເວລາທີ່ເວລາທີ່ ຈຳກັດ PB2 แล้วสังเกตເວລາເວລາທີ່ DIGIT1-DIGIT4 ອີກຮັງ ເສື່ອງແລ້ວໃຫ້ OFF Slide SW1 (PB3) แล้วทำการทดสอบซ้ำອີກຮັງ ทำการบันທຶກผลการทดสอบ

2. สร้างวงจรນับขึ้น-ลง 4 หลักที่แสดงทางເຊວນເຊກມີນຕໍ່ໄວ່ FPG

a) ออกแบบวงจรສະແກນແລະດອດຮ້າສັດຕິແສດງພາກເຊວນເຊກມີນຕໍ່ 4 หลัก หลักการทำงานของวงຈາກສະແກນເພື່ອແສດງພົດຕົວເລີບ 4 หลักທາງເຊວນເຊກມີນຕໍ່ (ແບນຄອມນອນແຕໂໂଡດ) ດັງລູບປີ່ 4.58 ໂດຍທີ່ຈະສະແກນປະກອບດ້ວຍ ວຈຈາກດອດຮ້າສ (2 to 4 Decoder) ແບນໃຫ້ອາດີພຸດເປັນລອຈິກ ‘0’ (One cold Decoder) ວຈຈາກມັດຕີເພີ້ກໍເຊອວ໌ (4 to 1 Multiplexer) ຫາມາດ 4 ບົດ ແລະວົງຈາກດອດຮ້າສ ຕົວແສດງພາກເຊວນເຊກມີນຕໍ່ ຮາຍລະເອີຍດັ່ງວົງຈາກດອດຮ້າສແລະວົງຈາກມັດຕີເພີ້ກໍເຊອວ໌ແສດງດັງລູບປີ່ 4.59(b) ແລະລູບປີ່ 4.60(b) ຕາມລຳດັບ ຈາກນັ້ນນຳລູບປີ່ 4.60(b) ມາສ້າງໄຟລ໌ສໍາຫັນທຳ Symbols ໃວ້າໃນ Project Location ຊື່ອ ch4sch ໂດຍກຳຫັນດ Project Name ແລະ Source File ຊື່ອ ch4f_4bits_4to1mux

b) ສ້າງໄຟລ໌ສໍາຫັນທົດສອບວົງຈາກນັ້ນ FFFF ໂດຍໃຫ້ວົງຈາກນັ້ນ-ลงແບນໃຈໂຄຣນັສ 4 ບົດ 4 หลักໃນ Project Location ຊື່ອ ch4sch ຈາກນັ້ນກຳຫັນດ Project Name ແລະ Source File ຊື່ອ ex4_3_7f ແລ້ວນຳໄຟລ໌ວົງຈາກໂມໂນສເຕເບີລ໌ຂື້ອ ch4f_debounce ໄຟລ໌ວົງຈາກດອດຮ້າສແລະມັດຕີເພີ້ກໍເຊອວ໌ຂື້ອ ch4f_4bits_4to1mux ແລະ ໄຟລ໌ວົງຈາກນັ້ນ-ลงແບນໃຈໂຄຣນັສ 4 ບົດຂື້ອ ch4f_sync_c16_updn ທີ່ມີອູ້ແລ້ວມາທຳ Symbols ຈາກນັ້ນວັດຜົງຈະດັງລູບປີ່ 4.62 ຜົ່ງຈາກນັ້ນຈະເຄີຍຮ່າກ່າວ່າເອົາຕີພຸດຈະກົດ 0000 (Hex) ເມື່ອ CLR = ‘0’ ແລະຈະນັບຂຶ້ນເມື່ອ DIR = ‘0’ ຢ້າ DIR = ‘1’ ຈະເປັນການນັບລົງ ພ້ອມໃຫ້ສັງເກດວ່າທີ່ວົງຈາກໂມໂນສເຕເບີລ໌ຈະມີສັງນູາມຄວາມຄືປະມານ 3 Hz (25MHz/2³) ຈາກວົງຈາກ Frequency divider ປື້ອນທີ່ໆ CLR ເພື່ອເຄີຍຮ່າກ່າວ່າພຸດຂອງວົງຈາກໂມໂນສເຕເບີລ໌ໂດຍອັດໂນມັດຖຸກໆ ປະມານ 1 / (3 Hz) = 0.33 ວິນາທີ່ ສໍາຫັນຄວາມຄືທີ່ໃຊ້ສະແກນແຕ່ລະຫັກຈະໃຫ້ຄວາມຄືປະມານ 48 Hz (ອ່າງນູ້ຂອງ 30Hz ເພື່ອໄໝໃຫ້ເກີນກາງກະປະປົບ)



ຮູບທີ 4.62 ວິຈນັນ FFFF ໂດຍໃຊ້ວິຈນັນບັນຫຼິນ-ລົງແນບບິນໂຄຣນັສ 4 ບົດຈຳນວນ 4 ລັກ

ການກໍາເໜີນຂາສົ່ງສູານຕ່າງໆ ຈະໃຊ້ອອສືລເລເຕອຣ໌ OSC= 25 MHz ປຸ່ມກົດ PB1-PB2 ແລະ Dip SW1 ເປັນອິນພູດ ແລະ ນີ້ເຫັນເຊັກມົນຕີ DIGIT1-DIGIT4 ໃນການແສດງຜົດຜ່ານທາງ D(6:0) ໂດຍໃຊ້ເຊັກມົນຕີ a, b, c, d, e, f ແລະ g ຕັ້ງນັ້ນໃນຜັງງາງຈຽງ ດັ່ງຕ້ອງຕ່ອງກາໂຄດຮ່ວມ DIGIT1-DIGIT4 ໃນການສັກເນ ກ່າວເກືອ

CLK = PB1 = INPUT = p44	D(0) = a = OUTPUT = p40	COM0 = DIGIT1 = OUTPUT = p31
CLR = PB2 = INPUT = p46	D(1) = b = OUTPUT = p35	COM1 = DIGIT2 = OUTPUT = p33
OSC = OSC = INPUT = p127	D(2) = c = OUTPUT = p32	COM2 = DIGIT2 = OUTPUT = p36
DIR = Dip SW1 = INPUT = p52	D(3) = d = OUTPUT = p30	COM3 = DIGIT2 = OUTPUT = p41
	D(4) = e = OUTPUT = p27	
	D(5) = f = OUTPUT = p25	
	D(6) = g = OUTPUT = p23	

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "COM0" LOC = "p31" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "COM1" LOC = "p33" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "COM2" LOC = "p36" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "COM3" LOC = "p41" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<0>" LOC = "p40" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<1>" LOC = "p35" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<2>" LOC = "p32" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<3>" LOC = "p30" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<4>" LOC = "p27" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<5>" LOC = "p25" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<6>" LOC = "p23" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DIR" LOC = "p52" | IOSTANDARD = LVCMOS33 ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;

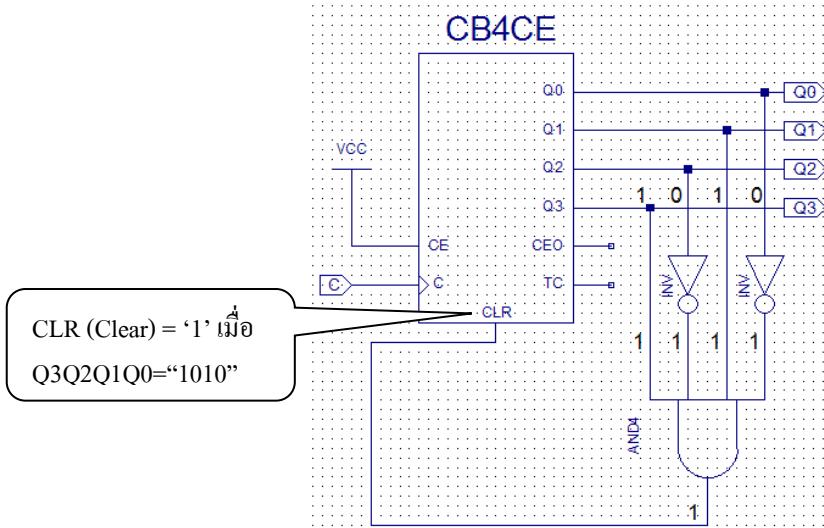
```

หลังจากโปรแกรมลง FPGA แล้วให้ทดสอบ ON Dip SW1 แล้วกดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกดปุ่ม PB1 ค้างไว้ แล้วให้สังเกตดูผลที่เซเว่นเซกเมนต์ DIGIT1-DIGIT4 ว่าแต่ละเซกเมนต์ให้กลับอิเล็กทรอนิกส์เป็นไปตามทฤษฎีหรือไม่ หากนั้นให้กด PB2 แล้วสังเกตเซเว่นเซกเมนต์ DIGIT1-DIGIT4 อีกครั้ง เสรีจแล้วให้ OFF Dip SW1 แล้วทำการทดสอบซ้ำอีกครั้ง ทำการบันทึกผลการทดลอง

4.4 การออกแบบวงจรนับ N แบบอะซิงโกรนัสและแบบชิงโกรนัส

4.4.1 การออกแบบวงจรนับ N แบบอะซิงโกรนัส

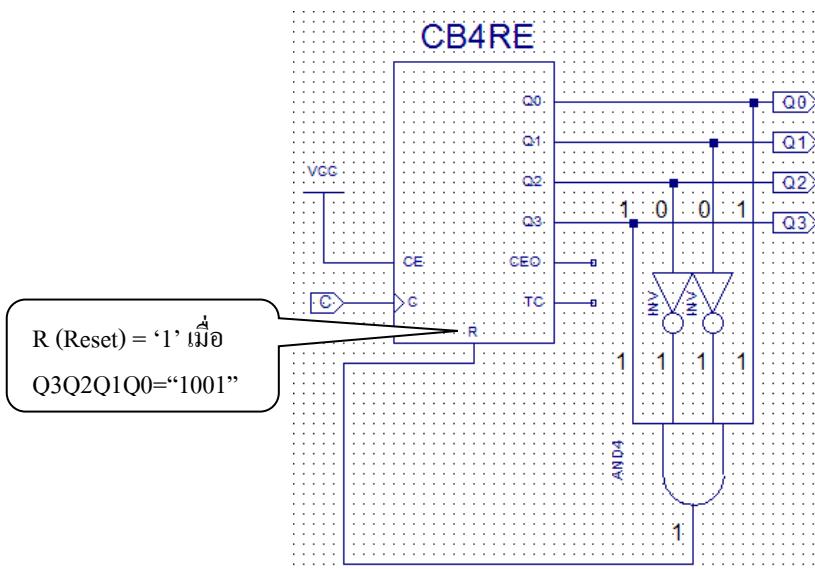
การออกแบบวงจรนับ N หรือหารความถี่ N แบบอะซิงโกรนัสนี้เราจะใช้ค่า N เป็นเงื่อนไขในการเคลียร์วงจรนับตัวอย่างวงจรนับ 10 (Modulo-10 counter) ดังรูปที่ 4.63 จะใช้ค่า $N = 10$ คือ $Q_3Q_2Q_1Q_0 = "1010"$ ในการเคลียร์เอาต์พุตเพื่อเริ่มต้นนับค่าใหม่ ข้อเสียของวงจนี้คือใช้งานได้ที่ความถี่ต่ำและอาจทำให้วงจรที่เกี่ยวข้องทำงานผิดพลาดเนื่องจากเอาต์พุตมีสถานะ “1010” ชั่วครู่หรือเกิดพัลส์แคบๆ (Glitch) การแก้ไขปัญหานี้ทำได้โดยใช้งานนับ N หรือหารความถี่ N แบบชิงโกรนัสโดยใช้วิธีเดียวกัน



รูปที่ 4.63 วงจรนับ 10 หรือหารความถี่ $N = 10$ แบบอะซิงโกรนัส

4.4.2 การออกแบบวงจรนับ N หรือหารความถี่ N แบบชิงโกรนัส

วงจรนับ N หรือหารความถี่ N แบบชิงโกรนัสดังรูปที่ 4.64 จะใช้ $N-1$ เป็นเงื่อนไขในการรีเซ็ตวงจรนับ เมื่อ $Q_3Q_2Q_1Q_0 = "1001"$ ($N=9$) แล้วทำให้ R (Reset) = '1' จากนั้นต้องรอสัญญาณนาฬิกาลูกคัดไปทริกวงจรนับจึงจะรีเซ็ตเอาต์พุต $Q_3Q_2Q_1Q_0 = "0000"$ เพื่อเริ่มต้นนับใหม่ ทำให้การออกแบบวงจรนับ N แบบชิงโกรนัสโดยวิธีนี้ย่างมาก



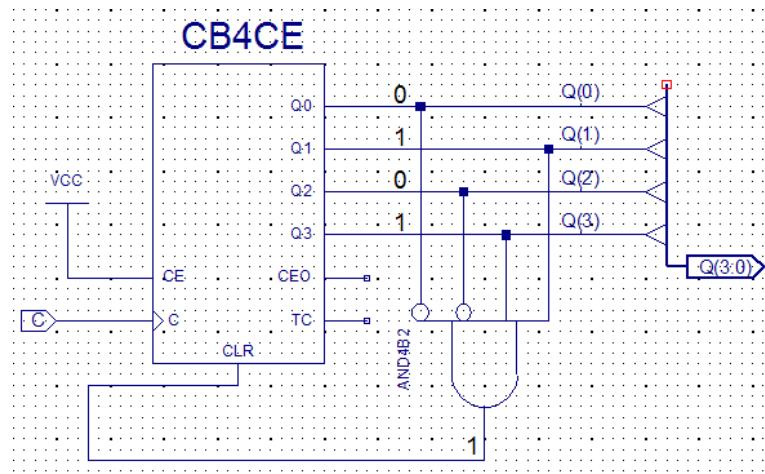
รูปที่ 4.64 วงจรนับ 10 หรือหารความถี่ N แบบชิงโกรนัส

4.4.1 การออกแบบวงจรนับ 10 แบบนับขึ้นแบบอะซิงโกรนัส

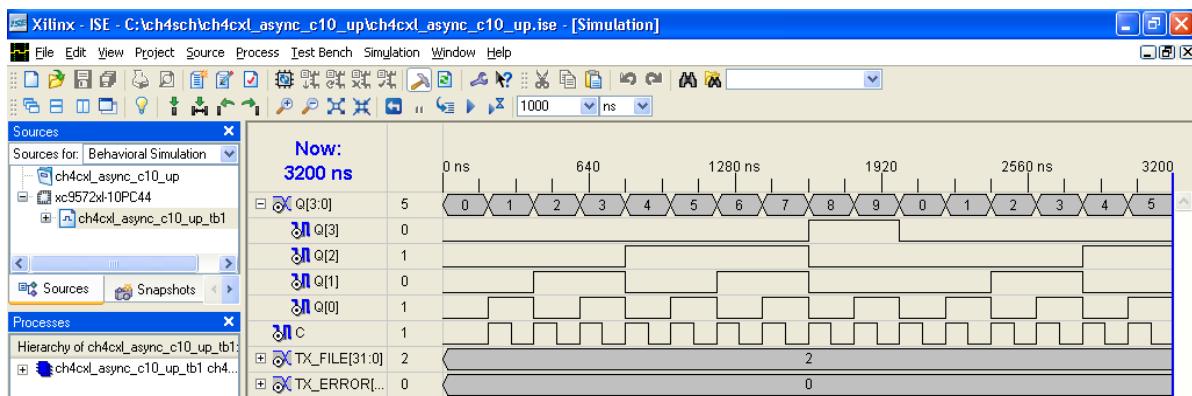
บอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรนับ 10 แบบนับขึ้นแบบอะซิงโกรนัสด้วย CPLD

- a) สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4cxl_async_c10_up และวัดผังวงจรดังรูปที่ 4.65 ทำ Behavioral simulation โดยใช้ไฟล์ชื่อ ch4cxl_async_c10_up_tb1 และพิจารณาผล Behavioral simulation ในรูปที่ 4.66 ว่าเป็นไปตามทฤษฎีหรือไม่

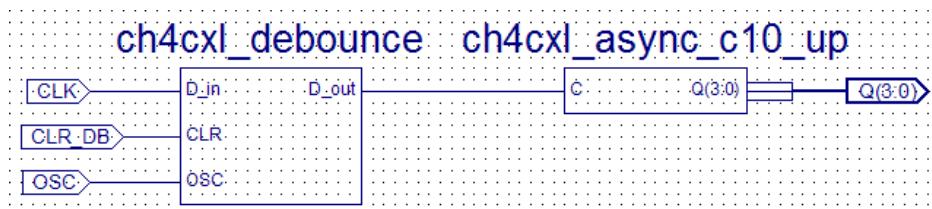


รูปที่ 4.65 ผังวงจรนับขึ้นแบบบริบีกเกิล



รูปที่ 4.66 ผล Behavioral simulation ของวงจรนับ 10 แบบนับขึ้นแบบอะซิงโกรนัส

- b) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_4_1cxl จากนั้นนำไฟล์วงจรโนนิสต์เบิลชื่อ ch4cxl_debounce และไฟล์วงจรนับชื่อ ch4cxl_async_c10_up มาทำ Symbols จากนั้นวัดผังวงจรดังรูปที่ 4.67



รูปที่ 4.67 วงจรทดสอบวงจรนับ 10 แบบนับขึ้นแบบอะซิงโกรนัส

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB2 เป็นอินพุต LED1-LED4 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{lll} \text{CLK} = \text{PB1} = \text{INPUT} = \text{p39} & \text{Q}(3) = \text{LED1} = \text{OUTPUT} = \text{p38} & \text{Q}(1) = \text{LED3} = \text{OUTPUT} = 36 \\ \text{CLR_DB} = \text{PB2} = \text{INPUT} = \text{p40} & \text{Q}(2) = \text{LED2} = \text{OUTPUT} = \text{p37} & \text{Q}(0) = \text{LED4} = \text{OUTPUT} = \text{p35} \\ \text{OSC} = \text{OSC} = \text{INPUT} = \text{p5} & & \end{array}$$

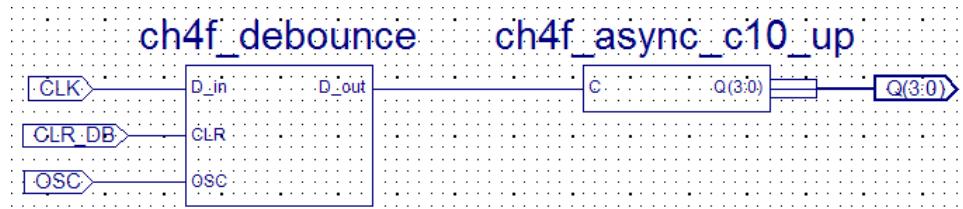
โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "CLK" LOC = "p39" ;
NET "CLR_DB" LOC = "p40" ;
NET "OSC" LOC = "p5" ;
NET "Q<0>" LOC = "p35" | SLEW = SLOW ;
NET "Q<1>" LOC = "p36" | SLEW = SLOW ;
NET "Q<2>" LOC = "p37" | SLEW = SLOW ;
NET "Q<3>" LOC = "p38" | SLEW = SLOW ;
```

หลังจากโปรแกรมลง CPLD แล้วให้ทดสอบกับปุ่ม PB1 และ PB2 ลับกันไปเรื่อยๆ แล้วให้สังเกตคูลท์ที่ LED2-LED4 ว่าให้ออกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

2. สร้างวงจรนับ 10 แบบนับขึ้นแบบอะซิงโกรันด์ด้วย FPGA

- a) สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4f_async_c10_up และวัดผังวงจรดังรูปที่ 4.65 จากนั้นทำ Behavioral simulation โดยใช้ไฟล์ชื่อ ch4f_async_c10_up_tb1 และวิเคราะห์ผล Behavioral simulation แล้วเขียนเดียวกับในรูปที่ 4.66 ว่าเป็นไปตามทฤษฎีหรือไม่
- b) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_4_1f จากนั้นนำไฟล์วงจรโนนิสเตเบิลชื่อ ch4f_debounce และไฟล์วงจรนับขึ้นแบบบริบiper ไปลิ๊กชื่อ ch4f_async_c10_up มาทำ Symbols จากนั้นวัดผังวงจรดังรูปที่ 4.68



รูปที่ 4.68 วงจรทดสอบวงจรนับขึ้น 3 บิต

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25MHz และปุ่มกด PB1-PB2 เป็นอินพุต LED L0-L3 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{lll} \text{CLK} = \text{PB1} = \text{INPUT} = \text{p44} & \text{Q}(3) = \text{L3} = \text{OUTPUT} = \text{p76} & \text{Q}(1) = \text{L1} = \text{OUTPUT} = \text{p77} \\ \text{CLR_DB} = \text{PB2} = \text{INPUT} = \text{p46} & \text{Q}(2) = \text{L2} = \text{OUTPUT} = \text{p69} & \text{Q}(0) = \text{L0} = \text{OUTPUT} = \text{p70} \\ \text{OSC} = \text{OSC} = \text{INPUT} = \text{p127} & & \end{array}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR_DB" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;
NET "Q<0>" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q<1>" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q<2>" LOC = "p69" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q<3>" LOC = "p76" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
```

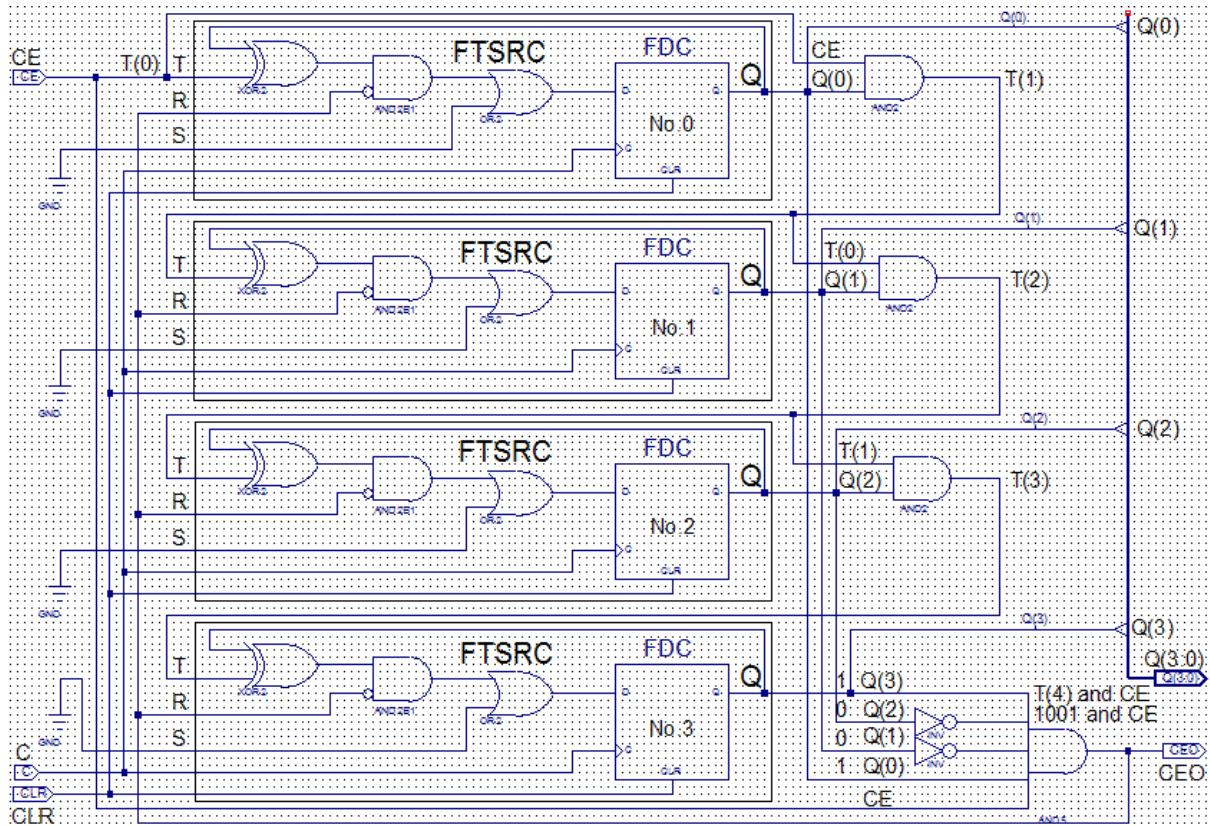
หลังจากโปรแกรมลง FPGA แล้วให้ทดสอบกดปุ่ม PB1 และ PB2 แล้วกันไปเรื่อยๆ แล้วให้สังเกตคุณผลที่ LED L0-L3 ว่าให้หลอด桔เอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

4.4.2 การออกแบบวงจรนับ 10 แบบนับขึ้นแบบชิงโครนัส

อุปกรณ์ที่ความมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

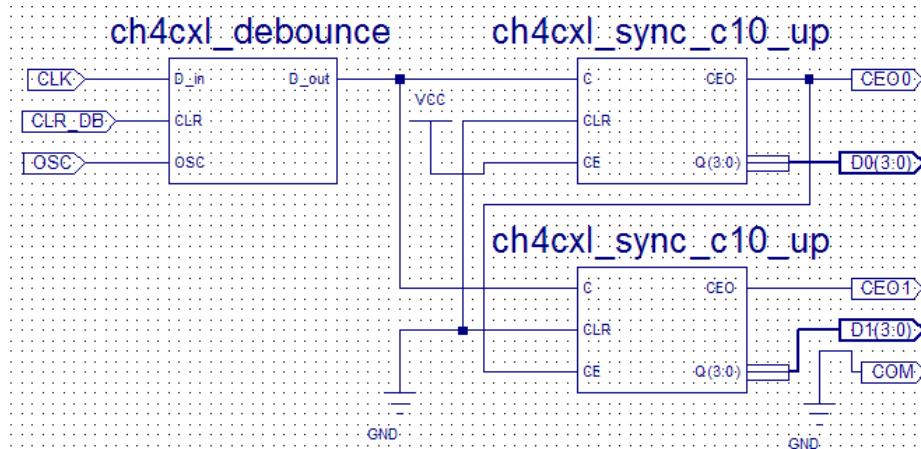
1. สร้างวงจรนับ 10 แบบนับขึ้นแบบชิงโครนัสด้วย CPLD

a) ออกแบบวงจรนับ 10 แบบนับขึ้นแบบชิงโครนัส ซึ่งเราใช้งาน T Flip-Flop ที่มีขาเคลียร์ (FDC) มาออกแบบเพิ่มเติมให้มีชิงโครนัสเซต (S) และรีเซ็ต (R) อยู่ในกรอบสี่เหลี่ยม (FTSRC) มาสร้างวงจรนับดังรูปที่ 4.69 ซึ่งใช้หลักการเดียวกับวงจรนับขึ้นแบบชิงโครนัส 4 บิต แต่เมื่อนับถึง 9 (1001) แล้วจะทำให้ขา CEO (Clock enable output) = '1' และเมื่อมี Clock ลูกคลักไปทrig กจะทำให้ T Flip-Flop ทุกตัวรีเซ็ต เอาท์พุตจึงมีค่าเป็น 0 ("0000") เพื่อเริ่มต้นนับค่าใหม่อีกครั้ง จากนั้นสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ch4cxl_sync_c10_up และวิเคราะห์ผังวงจรดังรูปที่ 4.69 และทำ Behavioral simulation โดยใช้ไฟล์ชื่อ ch4cxl_sync_c10_up_tb1 และพิจารณาผลว่าเป็นตามทฤษฎีหรือไม่



รูปที่ 4.69 ผังวงจรนับ 10 แบบนับขึ้นแบบชิงโครนัสและ Clock enable input/output

b) สร้างไฟล์สำหรับทดสอบวงจรนับ 100 โดยใช้งานวงจรนับ 10 แบบนับขึ้นแบบชิงโครนัสจำนวน 2 หลักใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_4_2cxl และนำไฟล์ลงในโน๊ตบุ๊คชื่อ ch4cxl_debounce และไฟวงจรนับ 10 แบบนับขึ้นแบบชิงโครนัสชื่อ ch4cxl_sync_c10_up มาทำ Symbols จากนั้นวัดผังวงจรดังรูปที่ 4.70



รูปที่ 4.70 วงจรนับ 100 โดยใช้งานนับ 10 แบบนับแบบชิงโกรนัสจำนวน 2 หลัก

การกำหนดขาสัญญาณต่างๆ ใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB2 เป็นอินพุต LED3-LED4 และเซเวนเซกเมนต์ DIGIT1 ใน การแสดงผล D0(3:0) โดยใช้เซกเมนต์ a, b, c และ d และ D1(3:0) จะใช้เซกเมนต์ e, f, g และ dp ดังนี้ในผังวงจร จึงต้องต่อขาโถดร่วม (COM) ลงกราวด์ด้วย กันล่างคือ

CLK = PB1 = INPUT = p39	D0(3) = d = OUTPUT = p24	D1(3) = dp = OUTPUT = p19
CLR_DB = PB2 = INPUT = p40	D0(2) = c = OUTPUT = p25	D1(2) = g = OUTPUT = p18
OSC = OSC = INPUT = p5	D0(1) = b = OUTPUT = p26	D1(1) = f = OUTPUT = p20
COM = DIGIT1 = OUTPUT = p34	D0(0) = a = OUTPUT = p27	D1(0) = e = OUTPUT = p22
CEO0 = LED4 = OUTPUT = p35	CEO1 = LED3 = OUTPUT = p36	

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

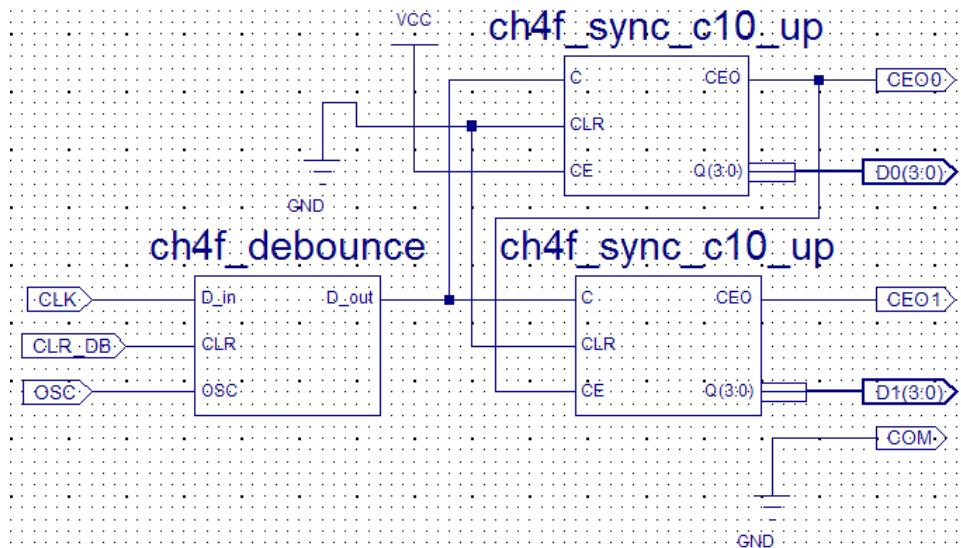
NET "CEO0" LOC = "p35" | SLEW = SLOW ;
NET "CEO1" LOC = "p36" | SLEW = SLOW ;
NET "CLK" LOC = "p39" ;
NET "CLR_DB" LOC = "p40" ;
NET "COM" LOC = "p34" | SLEW = SLOW ;
NET "DO<0>" LOC = "p27" | SLEW = SLOW ;
NET "DO<1>" LOC = "p26" | SLEW = SLOW ;
NET "DO<2>" LOC = "p25" | SLEW = SLOW ;
NET "DO<3>" LOC = "p24" | SLEW = SLOW ;
NET "D1<0>" LOC = "p22" | SLEW = SLOW ;
NET "D1<1>" LOC = "p20" | SLEW = SLOW ;
NET "D1<2>" LOC = "p18" | SLEW = SLOW ;
NET "D1<3>" LOC = "p19" | SLEW = SLOW ;
NET "OSC" LOC = "p5" ;

```

หลังจากโปรแกรมวงจรที่ออกแบบลง CPLD แล้วให้กดปุ่ม PB1 และ PB2 สลับกันไปเรื่อยๆ แล้วดูผลที่ LED3-LED4 และเซเวนเซกเมนต์ DIGIT1 ว่าแต่ละเซกเมนต์ให้ล้อจิกເອาร์พຸດเป็นไปตามทฤษฎีหรือไม่ ทำการบันทึกผลการทดลอง

2. สร้างวงจรนับ 10 แบบนับขั้นแบบชิงโกรนัสด้วย FPGA

- สร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ch4f_sync_c10_up และวิเคราะห์ผังวงจร ดังรูปที่ 4.69 แล้วทำ Behavioral simulation โดยใช้ไฟล์ชื่อ ch4f_sync_c10_up_tb1 และให้พิจารณาผลว่าเป็นตามทฤษฎีหรือไม่
- สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนดให้ Project Name และ Source File ชื่อ ex4_4_2f จากนั้นนำไฟล์ วงจร โนโนสเดเบิลชื่อ ch4f_debounce และไฟล์วงจรนับชื่อ ch4f_sync_c10_up มาทำ Symbols และวิเคราะห์ผังวงจรดังรูปที่ 4.71



รูปที่ 4.71 วงจรนับ 100 โดยใช้งานนับ 10 แบบนับแบบบิจ์โกรนัสจำนวน 2 หลัก

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25 MHz ปุ่มกด PB1-PB2 เป็นอินพุต และมี LED L0-L1 และเซเว่นเซกเมนต์ DIGIT1 ในการแสดงผล D0(3:0) โดยใช้เซกเมนต์ a, b, c และ d และ D1(3:0) จะใช้เซกเมนต์ e, f, g และ dp ดังนี้ในผังวงจร จึงต้องต่อขาค่าโดยร่วม (COM) ลงกราวด์ด้วย ก่อนวิธี

CLK = PB1 = INPUT = p44	D0(3) = d = OUTPUT = p30	D1(3) = dp = OUTPUT = p20
CLR_DB = PB2 = INPUT = p46	D0(2) = c = OUTPUT = p32	D1(2) = g = OUTPUT = p23
OSC = OSC = INPUT = p127	D0(1) = b = OUTPUT = p35	D1(1) = f = OUTPUT = p25
COM = DIGIT1 = OUTPUT = p31	D0(0) = a = OUTPUT = p40	D1(0) = e = OUTPUT = p27
CEO0 = L0 = OUTPUT = p70	CEO1 = L1 = OUTPUT = p77	

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "CEO0" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "CEO1" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR_DB" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "COM" LOC = "p31" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<0>" LOC = "p40" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<1>" LOC = "p35" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<2>" LOC = "p32" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DO<3>" LOC = "p30" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<0>" LOC = "p27" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<1>" LOC = "p25" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<2>" LOC = "p23" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D1<3>" LOC = "p20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;

```

หลังจากโปรแกรมวงจรที่ออกแบบลง FPGA แล้วให้กดปุ่ม PB1 และ PB2 สลับกันไปเรื่อยๆ และวัดผลที่ LED L0-L1 และเซเว่นเซกเมนต์ DIGIT1 ว่าแต่ละเซกเมนต์ให้ล้อจิกເອດพุดเป็นไปตามทฤษฎีหรือไม่ ทำการบันทึกผลการทดลอง

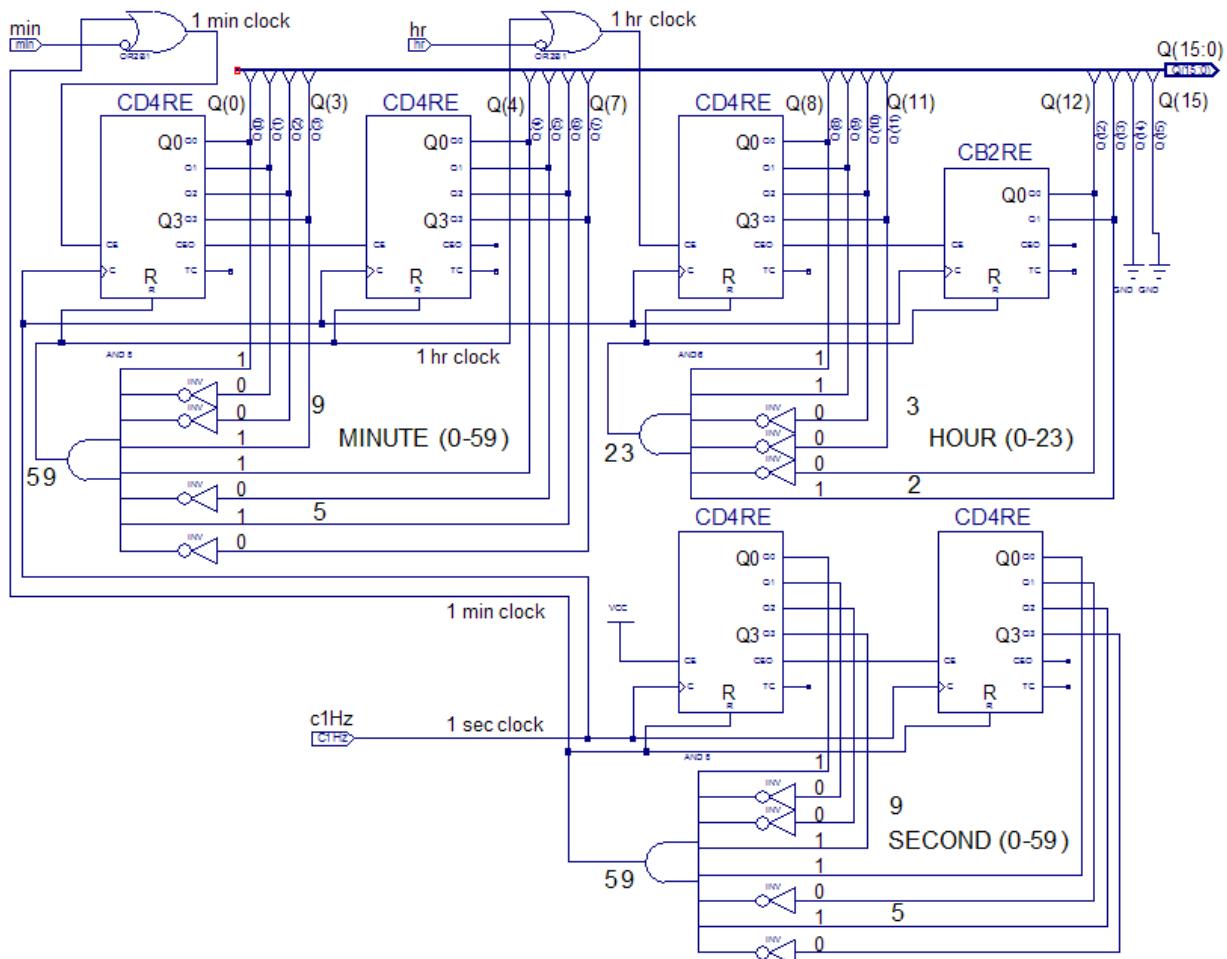
4.4.3 การออกแบบวงจรนาฬิกาดิจิตอลแบบตั้งเวลาหลักนาทีและชั่วโมงได้

อุปกรณ์ที่ควรมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรนาฬิกาดิจิตอลด้วย CPLD

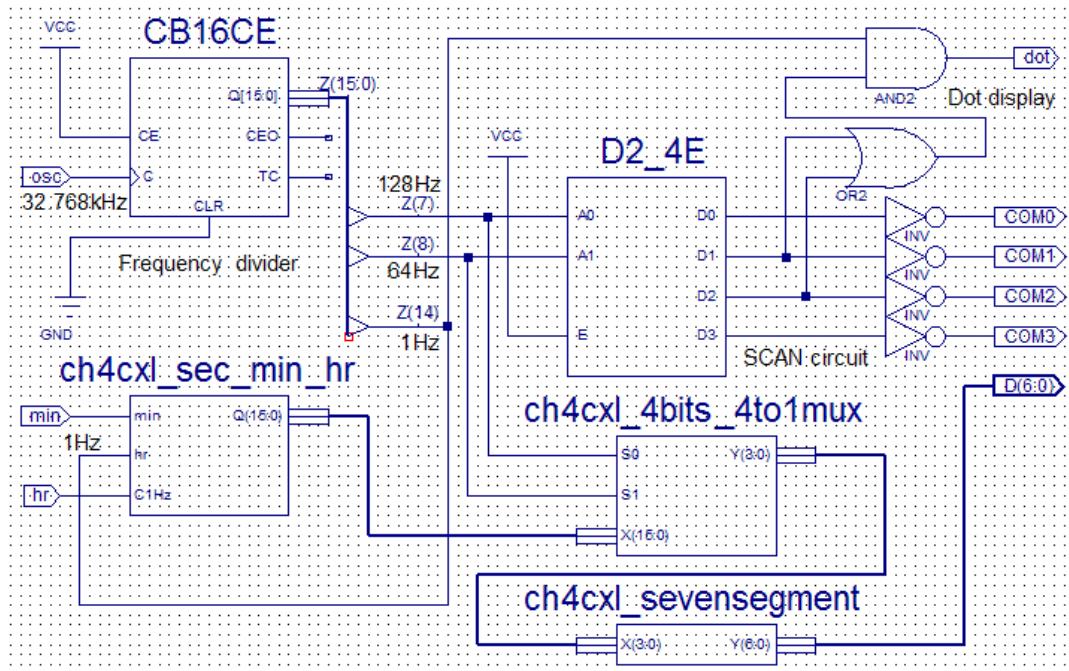
การสังเคราะห์วงจร

a) ออกแบบวงจรนับ 60 และวงจนับ 24 เพื่อทำเป็นวงจรแสดงผลเวลาวินาที นาที และ ชั่วโมง โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4cxl_sec_min_hr และทำการดาวน์โหลดในรูปที่ 4.72



รูปที่ 4.72 ผังวงจรนับ 60 และวงจนับ 24 เพื่อทำเป็นวงจรแสดงผลเวลาวินาที นาที และชั่วโมง

b) สร้างไฟล์วงจรนาฬิกาดิจิตอลใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_4_3cxl และนำไปไฟล์วงจนับ 60 และวงจนับ 24 เพื่อทำเป็นวงจรแสดงผลเวลาวินาที นาที และชั่วโมง ชื่อ ch4cxl_sec_min_hr ไฟล์วงจรดอร์หัสตัวแสดงผลทางเซเว่นเซกเมนต์ ชื่อ ch4cxl_sevensegment และไฟล์วงจรแมตติเพล็กซ์อร์ (4 to 1 Multiplexer)ขนาด 4 บิต ชื่อ ch4cxl_4bits_4to1mux มาทำเป็น Symbols จากนั้นนำผังวงจรดังรูปที่ 4.73 โดยเพิ่มความถี่สแกนเป็น 2 เท่า



ຮູບທີ 4.73 ຜັງຈຽນພິກາດິຈິດອອລ໌

ການກໍາໜາດ້າສ້າງຢ້ານຕ່າງໆ ຈະໃຊ້ອົບສົມເລເຕອർ OSC = 32.768 kHz ແລະ ບຸນກົດ PB1-PB2 ເປັນອິນພູດ ໂດຍມີເອົາຕົວ
ເປັນເຊວນເຊກມັນຕີ DIGIT1-DIGIT4 ໃນການແສດງຜລ ກລ່າວຄື່ອງ

Hr = PB1 = INPUT = p39	D(0) = a = OUTPUT = p27	D(4) = e = OUTPUT = p22
min = PB2 = INPUT = p40	D(1) = b = OUTPUT = p26	D(5) = f = OUTPUT = p20
OSC = OSC = INPUT = p5	D(2) = c = OUTPUT = p25	D(6) = g = OUTPUT = p18
COM0 = DIGIT1 = OUTPUT = p34	D(3) = d = OUTPUT = p24	dot = dp = OUTPUT = p19
COM1 = DIGIT2 = OUTPUT = p33		
COM2 = DIGIT3 = OUTPUT = p29		
COM3 = DIGIT4 = OUTPUT = p28		

ໂດຍພິມພື້ນໃນ Edit Constraints (Text) ສະບັບດັ່ງນີ້

```

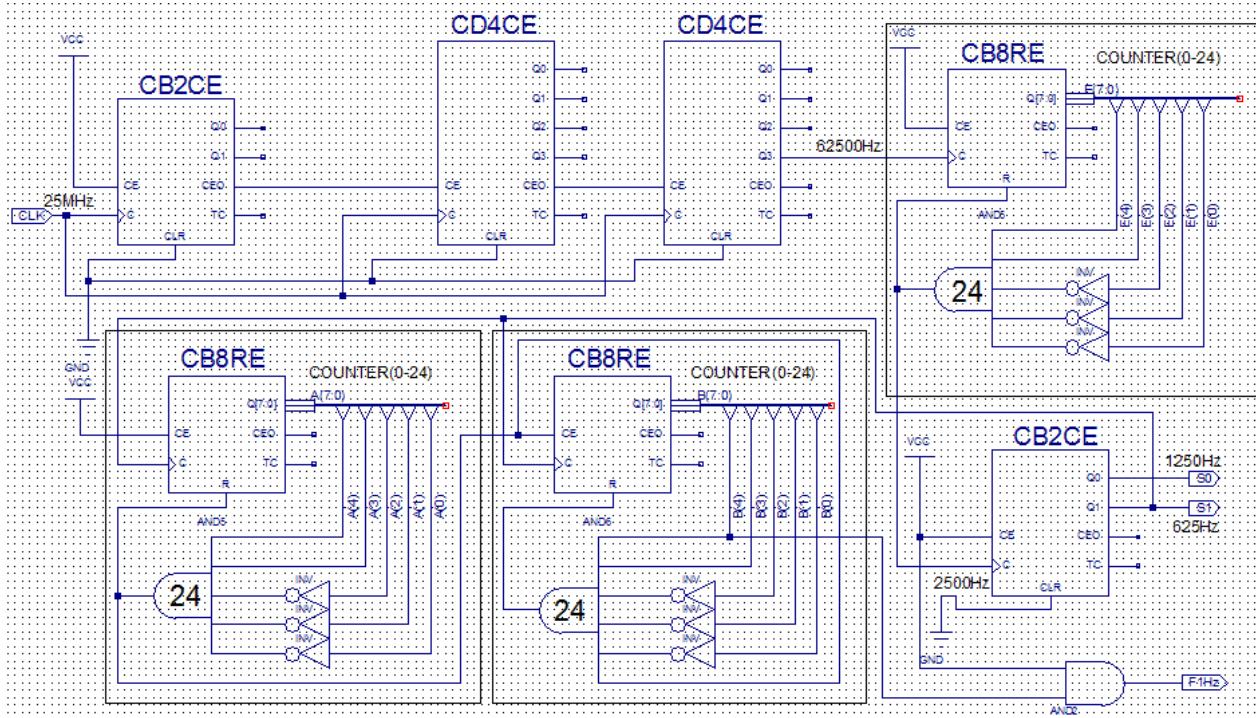
NET "COM0" LOC = "p34" | SLEW = SLOW ;
NET "COM1" LOC = "p33" | SLEW = SLOW ;
NET "COM2" LOC = "p29" | SLEW = SLOW ;
NET "COM3" LOC = "p28" | SLEW = SLOW ;
NET "D<0>" LOC = "p27" | SLEW = SLOW ;
NET "D<1>" LOC = "p26" | SLEW = SLOW ;
NET "D<2>" LOC = "p25" | SLEW = SLOW ;
NET "D<3>" LOC = "p24" | SLEW = SLOW ;
NET "D<4>" LOC = "p22" | SLEW = SLOW ;
NET "D<5>" LOC = "p20" | SLEW = SLOW ;
NET "D<6>" LOC = "p18" | SLEW = SLOW ;
NET "dot" LOC = "p19" | SLEW = SLOW ;
NET "hr" LOC = "p39" ;
NET "min" LOC = "p40" ;
NET "OSC" LOC = "p5" ;

```

ໜັງຈາກໂປຣແກຣມລົງ CPLD ແລ້ວໃຫ້ທົດລອງກົດປຸ່ມ PB2 ເພື່ອຕັ້ງເວລາໜັກນາທີ (min) ຈາກນັ້ນໃຫ້ກົດປຸ່ມ PB1 ເພື່ອຕັ້ງເວລາ
ໜັກຂ້າວໂນງ ເສົ່າງແລ້ວໃຫ້ຄູ່ພລຂອງເວລາວ່າໄປຕາມທຸກຢູ່ໃຫຍ່ໄວ່ ຈາກນັ້ນທຳການບັນທຶກການທົດລອງ

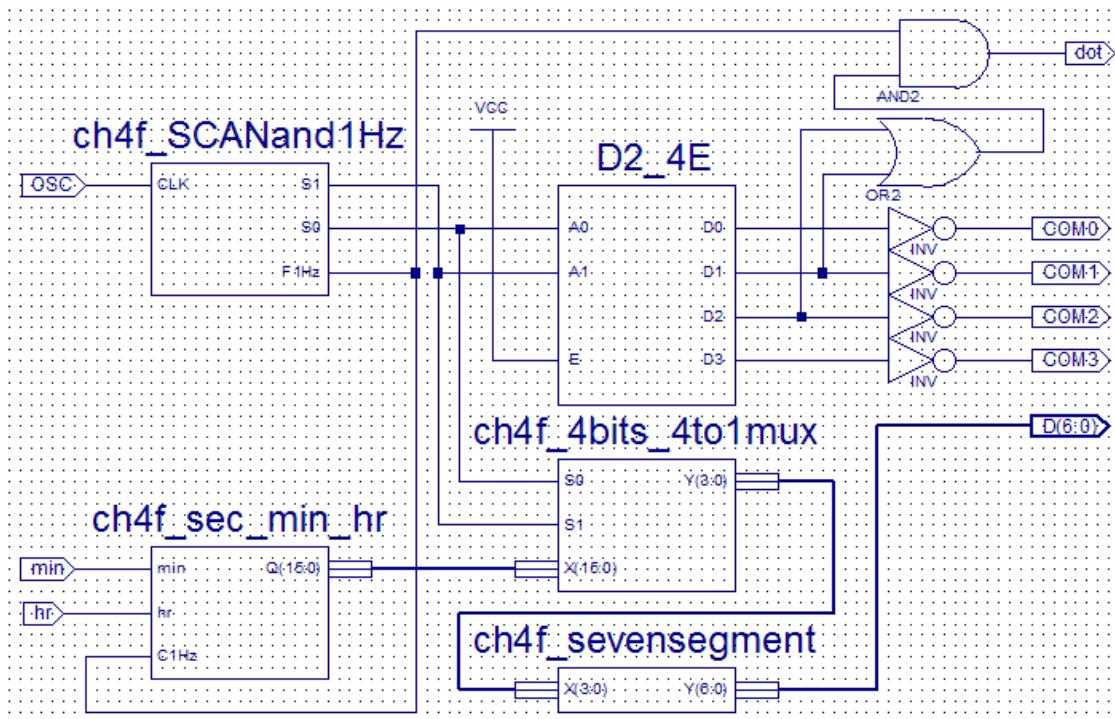
2. สร้างวงจรนาฬิกาดิจิตอลด้วย FPGA

- a) ออกแบบวงจรนับ 60 และวงจนับ 24 เพื่อทำเป็นวงจรแสดงผลเวลาวินาที นาที และชั่วโมง เช่นเดียวกับ CPLD โดยภาคผังวงจรดังรูปที่ 4.72 ไว้ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4f_sec_min_hr
- b) แล้วทำการออกแบบวงจนับ 100 (ประกอบด้วยวงจนับ 10 (CD4CE) 2 ชุด) วงจนับ 4 (CB2CE) 2 ชุด และวงจนับ 25 (3 ชุด) เพื่อนำมาสร้างความถี่ 1 Hz และสร้างความถี่สำหรับวงจรสแกนแต่ละหลักเท่ากับ 625Hz (มากกว่า 30 ครั้งต่อวินาที) โดยใช้ความถี่ 625Hz และ 1,250Hz จากความถี่อินพุต 25MHz แสดงดังรูปที่ 4.74 ซึ่งวงจนับ 100 สร้างจาก Symbols ชื่อ CD4CE (วงจนับ 10) จำนวน 2 ตัว จำนวนวงจนับ 25 จะตัดแปลงจาก Symbols ชื่อ CB8RE (วงจนับ 256) และขอให้สังเกตว่าเอาต์พุตของแอนด์เกต 6 อินพุตของวงจนับ 25 ตัวสุดท้าย แม้จะให้ความถี่เอาต์พุต 1 Hz เช่นเดียวกับที่ B(4) แต่จะมี Duty cycle แคบกว่ามาก ดังนั้นเพื่อให้เห็นการประวิบของจุดที่ใช้วนเชกเม้นต์ขั้ดขืน เราจึงเลือกความถี่เอาต์พุต 1 Hz จาก B(4) ไปจ่ายให้ขา F1Hz จากนั้นจึงทำการคาดผังวงจรสร้างความถี่ 1 Hz และวงจรสร้างความถี่สำหรับวงจรสแกนดังรูปที่ 4.74 ไว้ใน Project Location ชื่อ ch4sch โดยกำหนดให้ Project Name และ Source File ชื่อ ch4f_SCANand1Hz



รูปที่ 4.74 วงจรสร้างความถี่ 1 Hz และวงจรสร้างความถี่สำหรับวงจรสแกนชื่อ ch4f_SCANand1Hz

- c) สร้างไฟล์วงจรนาฬิกาดิจิตอลใน Project Location ชื่อ ch4sch แล้วกำหนดให้ Project Name และ Source File ชื่อ ex4_4_3f แล้วนำไฟล์วงจนับ 60 และวงจนับ 24 เพื่อทำเป็นวงจรแสดงผลเวลาวินาที นาที และชั่วโมง ชื่อ ch4f_sec_min_hr ไฟล์วงจรดูครับสัตว์แสดงผลทาง uten เชกเม้นต์ ชื่อ ch4f_sevensegment ไฟล์วงจรสร้างความถี่ 1 Hz และสร้างความถี่สำหรับวงจรสแกนชื่อ ch4f_SCANand1Hz และไฟล์วงจรมัลติเพล็กซ์อร์ชื่อ ch4f_4bits_4to1mux มาทำเป็น Symbols จากนั้นคาดผังวงจรดังรูปที่ 4.75



รูปที่ 4.75 ผังวงจรนาฬิกาดิจิตอล

การกำหนดขาสัญญาณต่างๆ ใช้ออสซิลเลเตอร์ OSC = 25 MHz และปุ่มกด PB1-PB2 เป็นอินพุต มี DIGIT1-DIGIT4 ในการแสดงผล กล่าวคือ

Hr = PB1 = INPUT = p44	D(0) = a = OUTPUT = p40	D(4) = e = OUTPUT = p27
min = PB2 = INPUT = p46	D(1) = b = OUTPUT = p35	D(5) = f = OUTPUT = p25
OSC = OSC = INPUT = p127	D(2) = c = OUTPUT = p32	D(6) = g = OUTPUT = p23
COM0 = DIGIT1 = OUTPUT = p31	D(3) = d = OUTPUT = p30	dot = dp = OUTPUT = p20
COM1 = DIGIT2 = OUTPUT = p33		
COM2 = DIGIT3 = OUTPUT = p36		
COM3 = DIGIT4 = OUTPUT = p41		

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "COM0" LOC = "p31" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "COM1" LOC = "p33" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "COM2" LOC = "p36" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "COM3" LOC = "p41" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<0>" LOC = "p40" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<1>" LOC = "p35" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<2>" LOC = "p32" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<3>" LOC = "p30" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<4>" LOC = "p27" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<5>" LOC = "p25" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<6>" LOC = "p23" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "dot" LOC = "p20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "hr" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "min" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;

```

หลังจากโปรแกรมวงจรที่ออกแบบลงชิป FPGA แล้วให้ทดลองกดปุ่ม PB2 เพื่อตั้งเวลาหลักนาที (min) จากนั้นแล้วให้กดปุ่ม PB1 เพื่อตั้งเวลาหลักชั่วโมง เสร็จแล้วให้คุณลองเวลาว่าไปตามทฤษฎีหรือไม่ จากนั้นทำการบันทึกผลการทดลอง

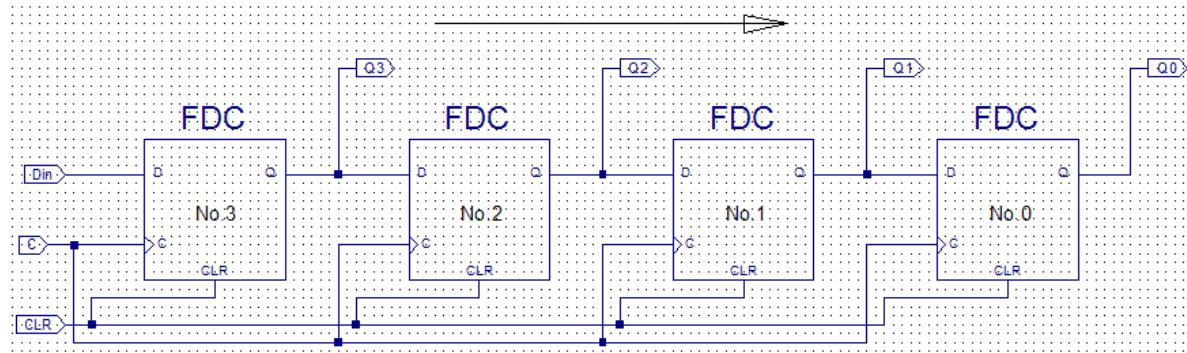
4.5 ชิฟต์รีจิสเตอร์

ชิฟต์รีจิสเตอร์เป็นวงจรซีเควนเชียล (Sequential) ที่มีการทำงานเป็นลำดับขั้น ซึ่งสามารถแบ่งตามวิธีการรับข้อมูลเข้า และส่งข้อมูลออก ได้เป็นดังนี้

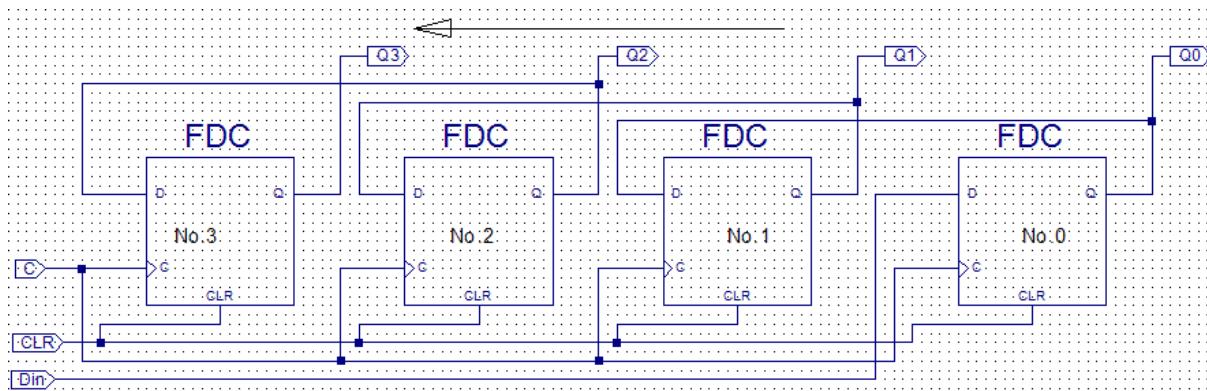
- 1) เข้าแบบอนุกรม-ออกแบบอนุกรม (Serial in–Serial out) และ/หรือขนาน (Serial in–Parallel out)
- 2) เข้าแบบขนาน-ออกแบบอนุกรม (Parallel in–Serial out) และ/หรือขนาน (Parallel in–Parallel out)

4.5.1 การออกแบบวงจรชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรม-ออกแบบอนุกรมและ/หรือแบบขนาน

ชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรม-ออกแบบอนุกรมและ/หรือแบบขนานแสดงดังรูปที่ 4.76 โดยที่ข้อมูลจะเข้าแบบอนุกรมที่อินพุต Din และจะเลื่อนข้อมูลไปทางขวา (Shift right) คือ เลื่อนบิตจาก MSB = Q3 ไป LSB = Q0 แบบอนุกรมไปออกที่ Q0 = Qout หรือเลื่อนข้อมูลออกแบบขนานที่ Q3, Q2, Q1 และ Q0 ตามจังหวะการทริกค้ายกอนบาก (หรืออนขึ้น) ของสัญญาณนาฬิกา ส่วนในรูปที่ 4.77 จะเลื่อนข้อมูลไปทางซ้าย (Shift left คือเลื่อนบิตจาก LSB ไป MSB)



รูปที่ 4.76 ชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรม-ออกแบบอนุกรมและแบบขนานแบบ Shift right

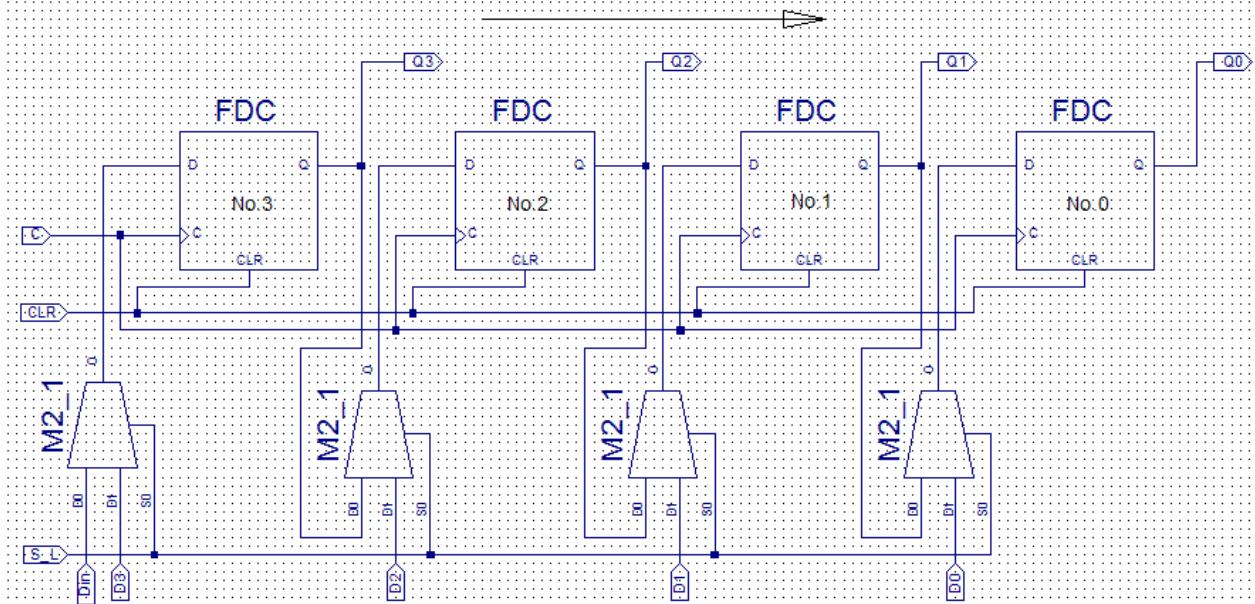


รูปที่ 4.77 ชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรม-ออกแบบอนุกรมและแบบขนานแบบ Shift left

4.5.2 การออกแบบวงจรชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรมหรือขนาน-ออกแบบอนุกรมและ/หรือขนาน

ชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรมหรือขนาน-ออกแบบอนุกรมและ/หรือขนานแสดงดังรูปที่ 4.78 ถ้า S_L เป็นลอจิก ‘0’ ข้อมูลจะเข้าทางอินพุต Din แล้วจึงเลื่อนข้อมูลออกทางเอต์พุตแบบอนุกรมหรือแบบขนานได้ทาง Q3-Q0 หรือ Qout

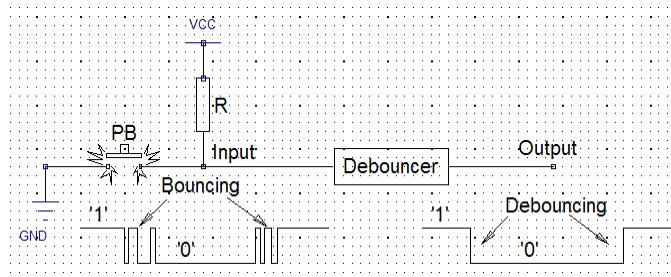
= Q0 ตามลำดับตามจังหวะของสัญญาณนาฬิกา แต่ถ้าขา S_L เป็นลอจิก ‘1’ ก็จะเป็นการโหลดข้อมูลแบบขนาน (Parallel access) เข้าทาง D3-D0 และจะเลื่อนออกทางเอาต์พุตแบบขนานหรือแบบอนุกรมตามลำดับตามจังหวะของสัญญาณนาฬิกา



รูปที่ 4.76 ชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรมหรือขนาน-ออกแบบอนุกรมและ/หรือขนาน

4.5.3 การออกแบบวงจรดีเบาเซอร์

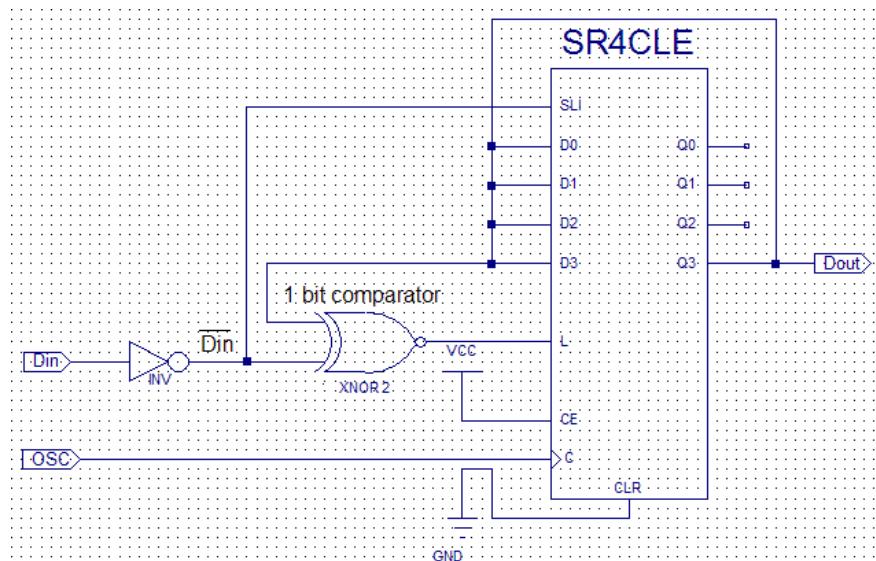
ในการกดคีย์บอร์ดหรือปุ่มกดจะเกิดเบาซ์ (Bouncing) ในช่วงที่หน้าสัมผัสแตกหักจากกันไม่สนิท เอาต์พุตจึงมีค่าไม่แน่นอนและจะให้พลั๊ส์ออกมากลายลุกทึ่งๆ ที่กดคีย์บอร์ดเพียงครั้งเดียว ปัญหานี้แก้ไขได้โดยใช้งานวงจรดีเบาเซอร์ (Debouncer) หรือโมโนโนสเตเบิล ตัวอย่างอินพุตและเอาต์พุตของวงจรดีเบาเซอร์แสดงดังรูปที่ 4.79 วงจรดีเบาเซอร์ที่ทำงานเดียวนแบบໄไอซี MC14490 แสดงรูปที่ 4.80 และรูปที่ 4.81 วงจรดีเบาเซอร์ที่ให้ความกว้างพลั๊ส์เอาต์พุตคงที่แสดงรูปที่ 4.82 และรูปที่ 4.83



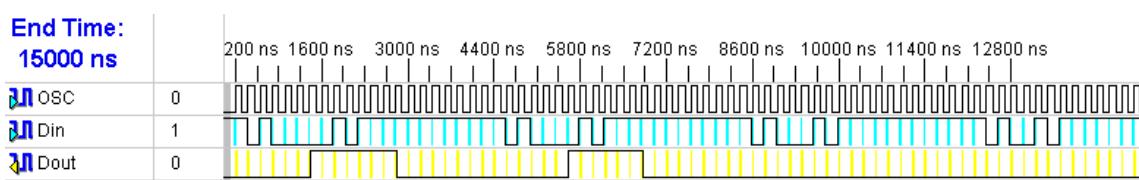
รูปที่ 4.79 ตัวอย่างอินพุตและเอาต์พุตของวงจรดีเบาเซอร์ขณะกดคีย์บอร์ดหรือปุ่มกด

รูปที่ 4.80 วงจรหลักประกอบด้วย ชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift left (SR4CLE) และ วงจรเปรียบเทียบขนาด 1 บิต (1 bit comparator) มีหลักการทำงานดังนี้คือ ถ้าไม่กดคีย์บอร์ด Din = ‘1’ (Active low) แล้ว \overline{Din} = ‘0’ ถ้าเอาต์พุต Dout = ‘0’ ก็จะทำให้เอาต์พุตของ 1 Bit comparator มีค่า = ‘1’ ดังนั้นขา L ของชิฟต์รีจิสเตอร์ = ‘1’ ทำให้ชิฟต์รีจิสเตอร์โหลดค่า Dout = ‘0’ ไปที่ D0-D3 ทำให้ Q0-Q3 = ‘0’ ด้วย ทำให้เอาต์พุต Dout = ‘0’ อญ্ত์ลดเวลา แต่ถ้ากดคีย์บอร์ด Din = ‘0’ แล้ว \overline{Din} = ‘1’ ทำให้เอาต์พุตของ 1 Bit comparator มีค่า = ‘0’ ดังนั้นขา L ของชิฟต์รีจิสเตอร์ = ‘0’ ทำให้ชิฟต์รีจิสเตอร์หยุดโหลดค่าแบบขนาน และจะทำการเลื่อนค่า \overline{Din} = ‘1’ แบบอนุกรมเข้าทางขา SLI ตามจังหวะของสัญญาณนาฬิกาจากอสซิลเลเตอร์ (C = OSC) ซึ่งหากหน้าสัมผัสแตกหักกันสนิทต่อเนื่องไม่น้อยกว่า 4 ภาพของสัญญาณนาฬิกาทำให้ Q0-Q3 = Dout = ‘1’ และเมื่อหน้าสัมผัสจาก

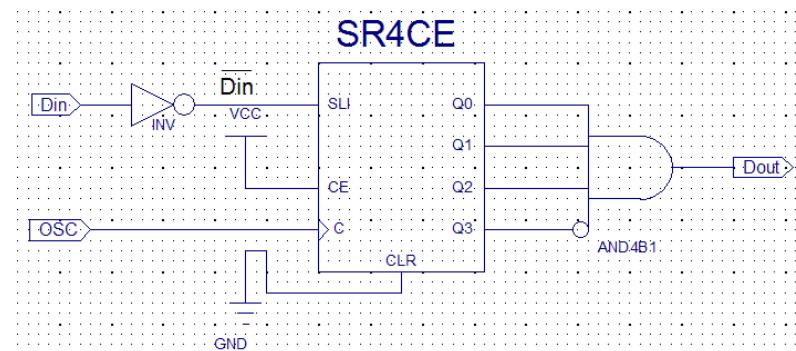
กันสนิทต่อเนื่องไม่น้อยกว่า 4 คาบของสัญญาณนาฬิกาทำให้ $D_{out} = 0$ โดยมีโคลั่งแกร์มเวลาแสดงดังรูปที่ 4.81 ซึ่งปกติการเกิดเบ่าจะกินเวลาอย่างต่อเนื่องกว่า 20 มิลลิวินาที ดังนั้นความถี่ของอสซิลเลเตอร์จึง $\frac{1}{(20\text{มิลลิวินาที}) / 4} = 200 \text{ Hz}$



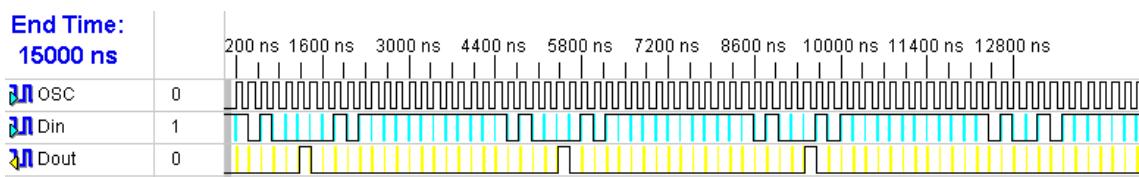
รูปที่ 4.80 ผังวงจรดิจิเตลเซอร์ที่ให้ความไว้ทางพัลส์เอาต์พุตขึ้นกับเวลาที่เกิดคิบบอร์ด



รูปที่ 4.81 ໄດ້ອະແກນເວລາຂອງງາງຈົດເບາເຊື່ອທີ່ໃຫ້ຄວາມກໍວັງພັດສີເຈົ້າພູ້ພົນໆກັບເວລາທີ່ກົດຄືບອົບຮົດ



รูปที่ 4.82 ผังวงจรวงจรดีเบาเซอร์ที่ให้ความกว้างพัลส์คงที่



รูปที่ 4.83 ໄດ້ອະແກນเวลาຂອງງරີເບາເຊອຣ໌ທີ່ໃຫ້ຄວາມກວ້າງພັດສົກທີ່

วงจรดีเบาเซอร์ที่ให้ความกว้างพัลส์คงที่ดังรูปที่ 4.82 นั้นเดินเรียนด้วยภาษา VHDL ไว้ในโค้ดตัวอย่างที่นำมาใน ISE WebPACK ซึ่งใช้ชิฟต์รีจิสเตอร์ 3 บิต แต่ผู้เขียนได้เพิ่มเป็น 4 บิตเพื่อให้มีประสิทธิภาพใกล้เคียงกับวงจรในรูปที่ 4.80 วงจนี้จะอักษะหลักการเลื่อนค่าເອດตามจังหวะสัญญาณนาฬิกาจากอสัจฉิจลเดอว์ ($C = OSC$) และจะให้อาต์พต $Dout = '1'$ (ความ

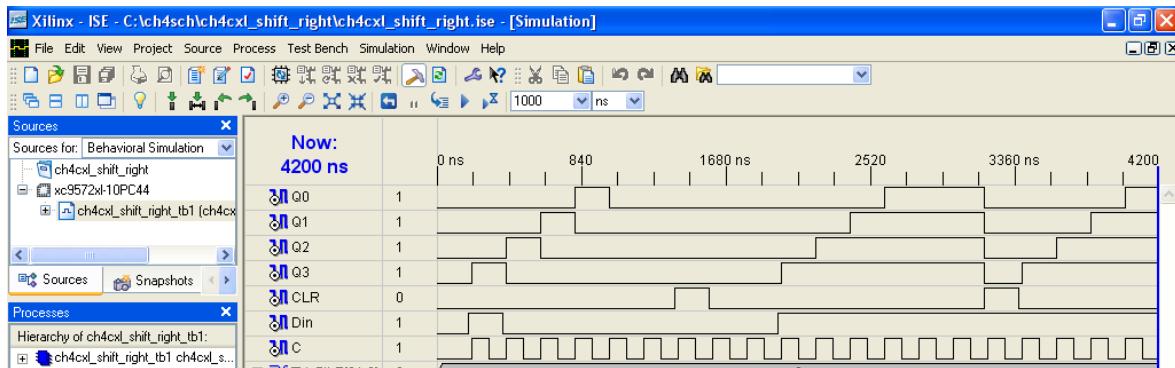
กว้างคงที่ 1 คาม) เมื่อหน้าสัมผัสแตะกันสนิทต่อเนื่องไม่น้อยกว่า 3 คามดังรูปที่ 4.83 ดังนั้นความถี่อสูตรจึงไม่ควรเกิน $1 / (20\text{มิลลิวินาที} / 3) = 150 \text{ Hz}$ วงจรดีเบาเซอร์ในรูปที่ 4.80 และรูปที่ 4.82 มีประสิทธิภาพดีกว่าวงจรดีเบาเซอร์ที่ได้อธิบายไปแล้วในการทดลองที่ 4.2.2 แต่อย่างไรก็ตามวงจรในการทดลองที่ 4.2.2 นั้นมีโครงสร้างที่สามารถทำความเข้าใจได้ง่าย

4.5.1 การออกแบบวงจรชิฟต์รีจิสเตอร์ 4 บิต

อุปกรณ์ที่ควรมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

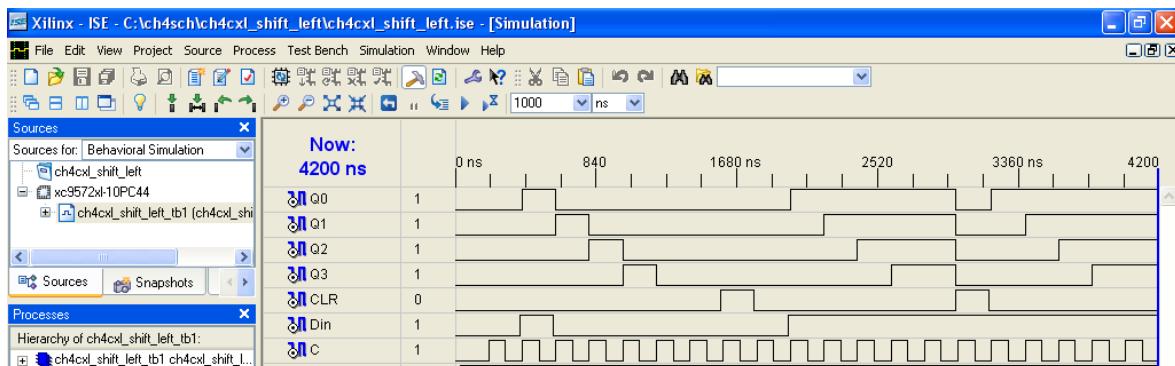
1. สร้างวงจรชิฟต์รีจิสเตอร์ 4 บิตด้วย CPLD

a) วัดผังวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบเลื่อนข้อมูลไปทางขวา (Shift right) ดังรูปที่ 4.76 โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ch4cxl_shift_right และทำการ simulation โดยใช้ไฟล์ชื่อ ch4cxl_shift_right_tb1 และพิจารณาผล Behavioral simulation ในรูปที่ 4.84 ว่าเป็นไปตามทฤษฎีหรือไม่



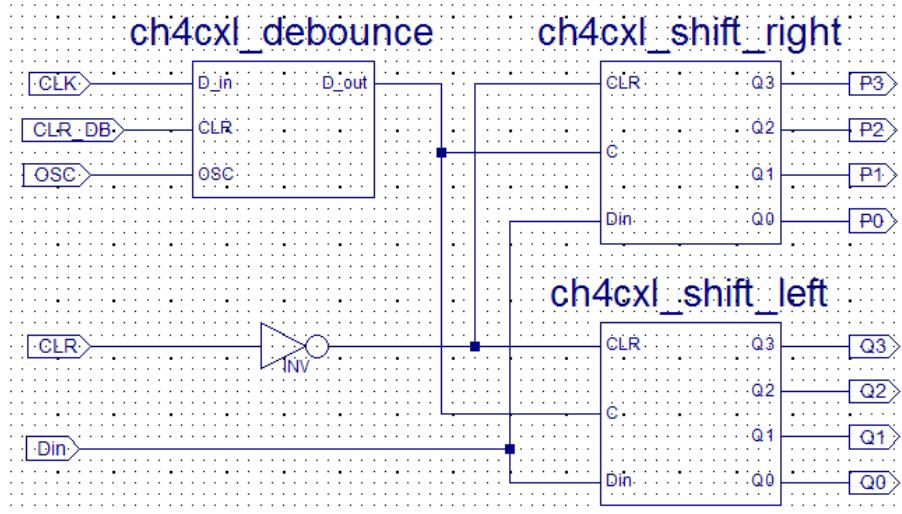
รูปที่ 4.84 ผล Behavioral simulation ของวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวา (Shift right)

b) วัดผังวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบเลื่อนข้อมูลไปทางซ้าย (Shift left) ดังรูปที่ 4.77 โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ch4cxl_shift_left และทำการ simulation โดยใช้ไฟล์ชื่อ ch4cxl_shift_left_tb1 และพิจารณาผล Behavioral simulation ในรูปที่ 4.85 ว่าเป็นไปตามทฤษฎีหรือไม่



รูปที่ 4.85 ผล Behavioral simulation ของวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางซ้าย (Shift left)

c) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_5_1cxl จากนั้นนำไฟล์ผังจริงมาใส่ในสแตบิลชื่อ ch4cxl_debounce หากวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวา ชื่อ ch4cxl_shift_right และไฟล์ผังจริงชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางซ้าย ชื่อ ch4cxl_shift_left มาทำ Symbols จากนั้นวัดผังวงจรดังรูปที่ 4.86



รูปที่ 4.86 วงจรทดสอบของชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวาและเลื่อนข้อมูลไปทางซ้าย

การกำหนดขาสัญญาณ จะใช้ OSC = 32.768kHz ปุ่มกด PB1-PB4 เป็นอินพุต LED1-LED4 เป็นเอาต์พุต ก่อว่าคือ

CLK = PB1 = INPUT = p39	Q0 = LED4 = OUTPUT = p35	P0 = MN4 = OUTPUT = p3
CLR_DB = PB2 = INPUT = p40	Q1 = LED3 = OUTPUT = p36	P1 = MN3 = OUTPUT = p4
OSC = OSC = INPUT = p5	Q2 = LED2 = OUTPUT = p37	P2 = MN2 = OUTPUT = p6
Din = PB3 = INPUT = p42	Q3 = LED1 = OUTPUT = p38	P3 = MN1 = OUTPUT = p7
CLR = PB4 = INPUT = p43		

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

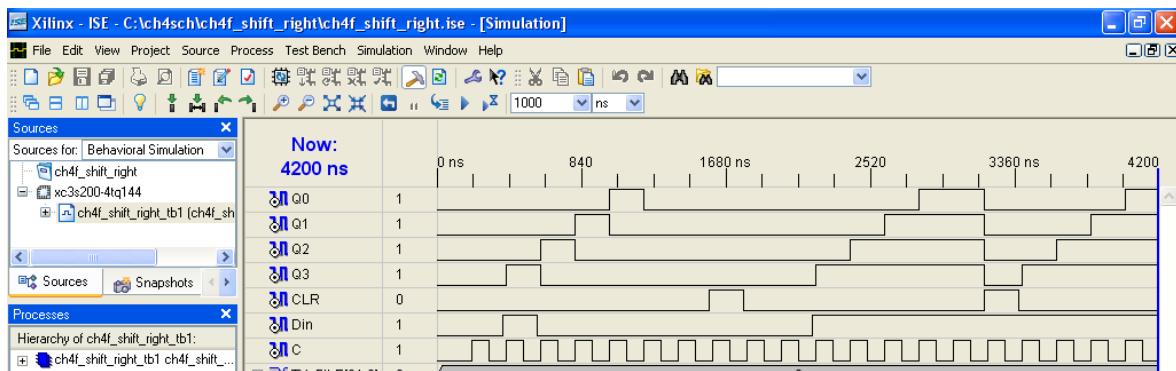
NET "CLK" LOC = "p39" ;
NET "CLR" LOC = "p43" ;
NET "CLR_DB" LOC = "p40" ;
NET "Din" LOC = "p42" ;
NET "OSC" LOC = "p5" ;
NET "P0" LOC = "p3" | SLEW = SLOW ;
NET "P1" LOC = "p4" | SLEW = SLOW ;
NET "P2" LOC = "p6" | SLEW = SLOW ;
NET "P3" LOC = "p7" | SLEW = SLOW ;
NET "Q0" LOC = "p35" | SLEW = SLOW ;
NET "Q1" LOC = "p36" | SLEW = SLOW ;
NET "Q2" LOC = "p37" | SLEW = SLOW ;
NET "Q3" LOC = "p38" | SLEW = SLOW ;

```

หลังจากโปรแกรมวงจรที่ออกแบบลง CPLD แล้วให้ OFF Slide SW1 และ Slide SW2 ทดลองกดปุ่ม PB1 และกดปุ่ม PB2 สลับกันไปอีก 4 ครั้งแล้วสังเกตคุณลักษณะที่ LED1-LED4 ว่าคิดสว่างโดยให้ล็อกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จำนวนน้ำหนักปุ่ม PB4 แล้วกดปุ่ม PB1 และกดปุ่ม PB2 สลับกันไปอีก 4 ครั้ง กดปุ่ม PB3 แล้วค้างไว้ (ON Slide SW1) แล้วกดปุ่ม PB1 และกดปุ่ม PB2 สลับกันไปอีก 4 ครั้งแล้วให้สังเกตคุณลักษณะที่ LED1-LED4 จำนวนน้ำหนักบันทึกผลการทดลอง

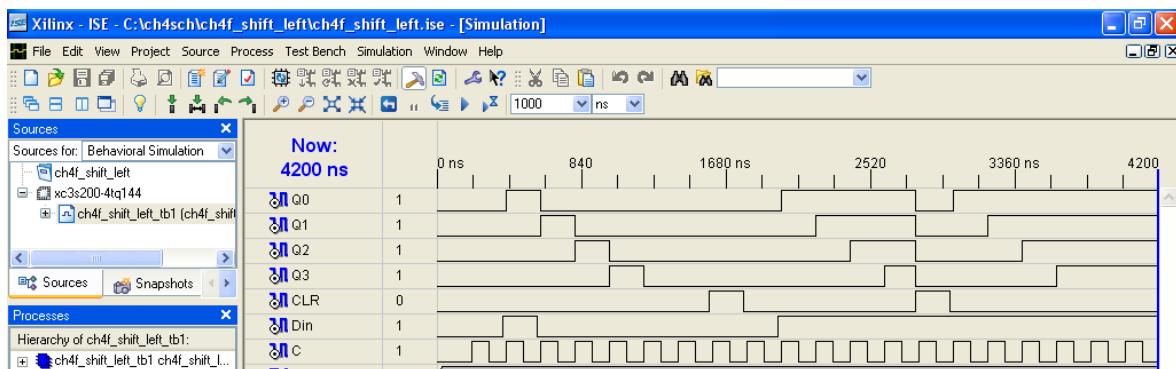
2. สร้างวงจรชิฟต์รีจิสเตอร์ 4 บิตด้วย FPGA

a) ภาคผังวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบเลื่อนข้อมูลไปทางขวา (Shift right) ดังรูปที่ 4.76 โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ch4f_shift_right เสร็จแล้วทำ Behavioral simulation โดยใช้ไฟล์ ชื่อ ch4f_shift_right_tb1 และพิจารณาผล Behavioral simulation ในรูปที่ 4.87 ว่าเป็นไปตามทฤษฎีหรือไม่



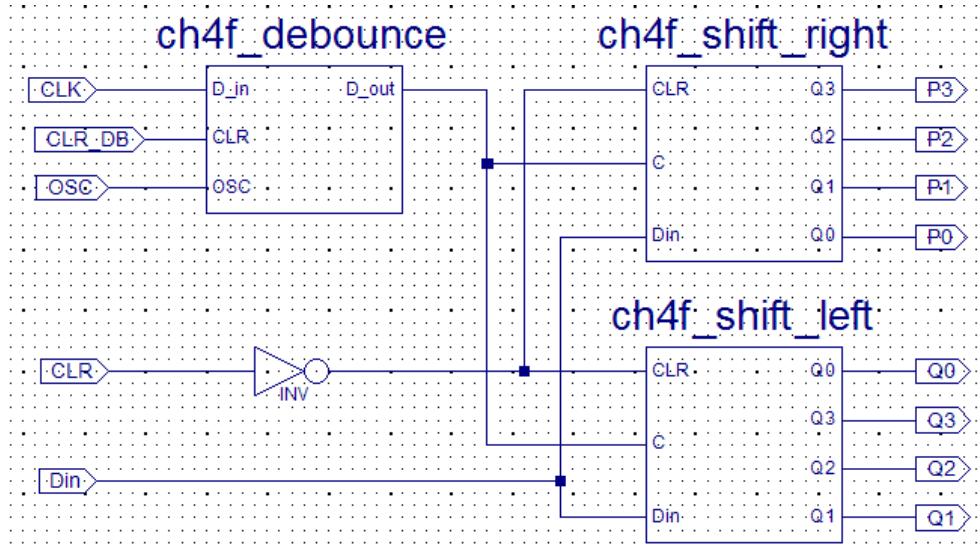
รูปที่ 4.87 ผล Behavioral simulation ของวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวา (Shift right)

b) ภาคผังวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบเลื่อนข้อมูลไปทางซ้าย (Shift left) ดังรูปที่ 4.77 โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนดให้ Project Name และ Source File ชื่อ ch4f_shift_left เสร็จแล้วทำ Behavioral simulation โดยใช้ไฟล์ ชื่อ ch4f_shift_left_tb1 และพิจารณาผล Behavioral simulation ในรูปที่ 4.88 ว่าเป็นไปตามทฤษฎีหรือไม่



รูปที่ 4.88 ผล Behavioral simulation ของวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางซ้าย (Shift left)

c) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนดให้ Project Name และ Source File ชื่อ ex4_5_1f จากนั้นนำไฟล์ วงจรโน้มโนเตเบิล ชื่อ ch4f_debounce ไฟล์วงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวา ชื่อ ch4f_shift_right และไฟล์วงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางซ้าย ชื่อ ch4f_shift_left มาทำ Symbols จากนั้นภาคผังวงจรดังรูปที่ 4.89



รูปที่ 4.89 วงจรทดสอบของชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวาและเลื่อนข้อมูลไปทางซ้าย

การกำหนดขาสัญญาณจะใช้ OSC = 25MHz ปุ่มกด PB1-PB3 และ Dip SW1 เป็นอินพุต LED L0-L3 เป็นเอาต์พุต กล่าวคือ

CLK = PB1 = INPUT = p44	Q3 = L3 = OUTPUT = p76	P3 = L7 = OUTPUT = p78
CLR_DB = PB2 = INPUT = p46	Q2 = L2 = OUTPUT = p69	P2 = L6 = OUTPUT = p73
Din = Dip SW1 = INPUT = p52	Q1 = L1 = OUTPUT = p77	P1 = L5 = OUTPUT = p79
CLR = PB3 = INPUT = p47	Q0 = L0 = OUTPUT = p70	P0 = L4 = OUTPUT = p74
OSC = OSC = INPUT = p127		

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR" LOC = "p47" | IOSTANDARD = LVCMOS33 ;
NET "CLR_DB" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "Din" LOC = "p52" | IOSTANDARD = LVCMOS33 ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;
NET "P0" LOC = "p74" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "P1" LOC = "p79" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "P2" LOC = "p73" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "P3" LOC = "p78" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q0" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q1" LOC = "p77" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q2" LOC = "p69" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Q3" LOC = "p76" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;

```

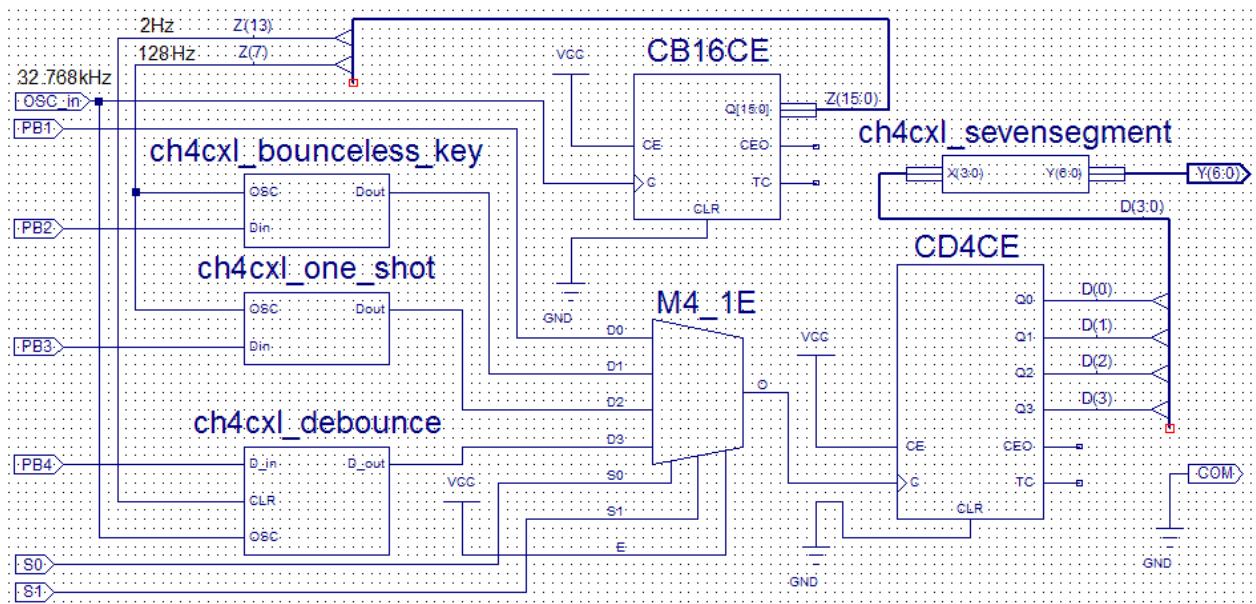
หลังจากโปรแกรมวงจรที่ออกแบบลง FPGA แล้วให้ OFF Dip SW1 ทดลองกดปุ่ม PB1 และกดปุ่ม PB2 สลับกันไปอีก 4 ครั้งแล้วสังเกตคุณลักษณะที่ LED L0-L3 ว่าคิดสว่างโดยให้ออกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นกดปุ่ม PB3 และกดปุ่ม PB1 และกดปุ่ม PB2 สลับกันไปอีก 4 ครั้ง ON Dip SW1 แล้วกดปุ่ม PB1 และกดปุ่ม PB2 สลับกันไปอีก 4 ครั้งแล้วให้สังเกตคุณลักษณะที่ LED L0-L3 จากนั้นจึงบันทึกผลการทดลอง

4.5.2 การออกแบบวงจรดีเบาเซอร์แบบต่างๆ

อุปกรณ์ที่ควรมีเพื่อทดลองคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรดีเบาเซอร์ด้วย CPLD

- ภาคผังวงจรดีเบาเซอร์ในรูปที่ 4.80 และรูปที่ 4.82 โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch จากนั้นกำหนด Project Name และ Source File ของวงจรรูปที่ 4.80 และรูปที่ 4.82 ชื่อ ch4cxl_bounceless_key และ ch4cxl_one_shot ตามลำดับ
- สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_5_2cxl จากนั้นนำไฟล์วงจรดอครหัสตัวแสดงผลทางเลขานเชกเมนต์ ชื่อ ch4cxl_sevensegment ไฟล์วงจรดีเบาเซอร์ (โนโนนสเตบิล) ชื่อ ch4cxl_debounce, ch4cxl_bounceless_key และ ch4cxl_one_shot มาทำ Symbols จากนั้นนำผังวงจรดังรูปที่ 4.90 ซึ่งจะนี้จะเป็นการเลือกสัญญาณจากปุ่มกดโดยไม่ผ่านวงจรดีเบาเซอร์และเลือกผ่านวงจรดีเบาเซอร์ทั้ง 3 แบบเพื่อส่งพัลส์ให้วงจรนับ 10



รูปที่ 4.90 วงจรทดสอบวงจรดีเบาเซอร์แต่ละชนิด

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB6 และ OSC เป็นอินพุต มีเข้าเรนเชกเมนต์ DIGIT1 ในการแสดงผลผ่านทาง Y(6:0) โดยใช้เชกเมนต์ a, b, c, d, e, f และ g และต้องต่อขาโอดรร์รัม DIGIT1 ลงกราวด์ ก่อนว่าคือ

PB1 = PB1 = INPUT = p39	Y(0) = a = OUTPUT = p27	Y(5) = f = OUTPUT = p20
PB2 = PB2 = INPUT = p40	Y(1) = b = OUTPUT = p26	Y(6) = g = OUTPUT = p18
PB3 = PB3 = INPUT = p42	Y(2) = c = OUTPUT = p25	COM = DIGIT1 = OUTPUT = p34
PB4 = PB4 = INPUT = p43	Y(3) = d = OUTPUT = p24	S1 = PB5 = INPUT = p44
OSC_in = OSC = INPUT = p5	Y(4) = e = OUTPUT = p22	S0 = PB6 = INPUT = p1

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

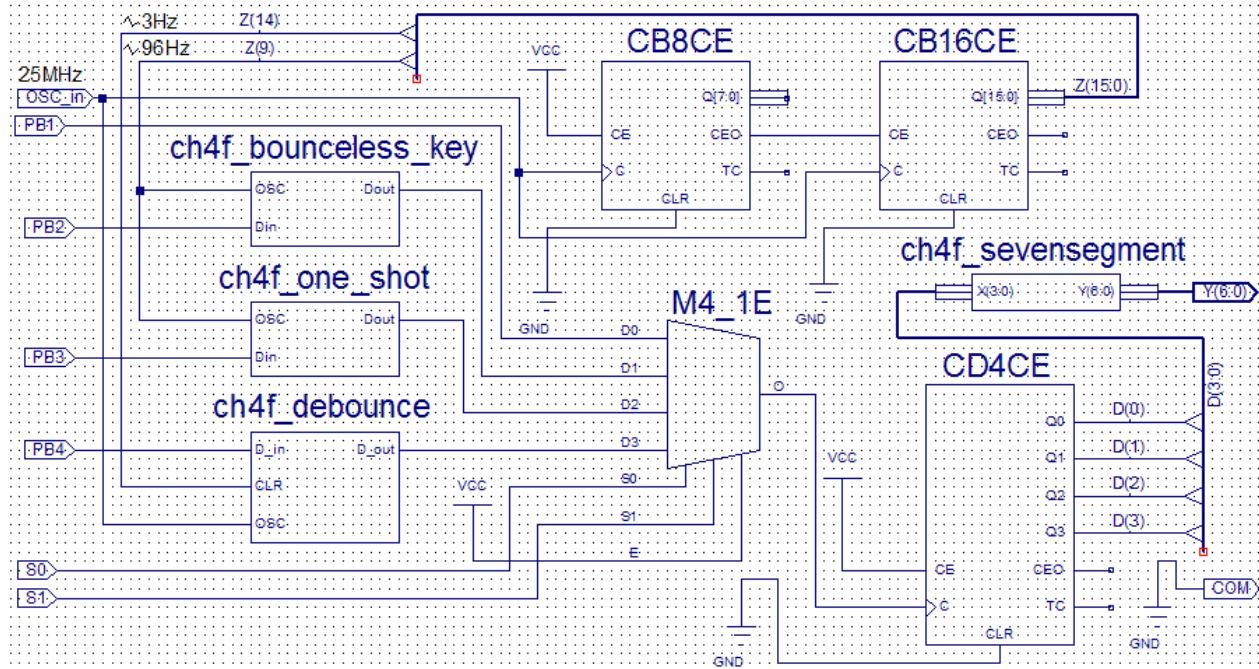
NET "PB1" LOC = "p39" ;
NET "PB2" LOC = "p40" ;
NET "PB3" LOC = "p42" ;
NET "PB4" LOC = "p43" ;
NET "S0" LOC = "p1" ;
NET "S1" LOC = "p44" ;
NET "OSC_in" LOC = "p5" ;
NET "Y<0>" LOC = "p27" | SLEW = SLOW ;
NET "Y<1>" LOC = "p26" | SLEW = SLOW ;
NET "Y<2>" LOC = "p25" | SLEW = SLOW ;
NET "Y<3>" LOC = "p24" | SLEW = SLOW ;
NET "Y<4>" LOC = "p22" | SLEW = SLOW ;
NET "Y<5>" LOC = "p20" | SLEW = SLOW ;
NET "Y<6>" LOC = "p18" | SLEW = SLOW ;
NET "COM" LOC = "p34" | SLEW = SLOW ;

```

หลังจากโปรแกรม CPLD แล้วให้เซต S1S0 = “00” (ON Slide SW3 หรือ PB5 และ ON Slide SW4 หรือ PB6) แล้วกดปุ่ม PB1 ไปเรื่อยๆ อย่างช้าๆ แล้วกดเร็วขึ้นเรื่อยๆ และกดค้างไว้ ให้สังเกตคุณที่เวลาเซกเมนต์ว่าติดสว่างโดยใช้หลอดจิกເອาດพุดเป็นตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง แล้วทดลองซ้ำโดยเซต S1S0 = “01”, “10”, และ “11” และกดปุ่ม PB2, PB3 และ PB4 ตามลำดับ

2. สร้างวงจรดิจิตอลด้วย FPGA

- ภาคผังวงจรดิจิตอลรุ่นที่ 4.80 และรุ่นที่ 4.82 โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch จากนั้นกำหนด Project Name และ Source File ของวงจรรุ่นที่ 4.80 และรุ่นที่ 4.82 ชื่อ ch4f_bounceless_key และ ch4f_one_shot ตามลำดับ
- สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_5_2f จากนั้นนำไฟล์วงจรดิจิตอลทั้งหมดแสดงผลทางเวลาเซกเมนต์ ชื่อ ch4f_sevensegment ไฟล์วงจรดิจิตอล (โมโนสเตเบิล) ชื่อ ch4f_debounce, ch4f_bounceless_key และ ch4f_one_shot มาทำ Symbols จากนั้นนำผังวงจรดังรุ่นที่ 4.91 ซึ่งจะชนที่จะเป็นการเลือกสัญญาณจากปุ่มกดโดยไม่ผ่านวงจรดิจิตอลและเลือกผ่านวงจรดิจิตอลทั้ง 3 แบบเพื่อส่งพัลส์ให้วงจรรับ 10



รุ่นที่ 4.91 วงจรดสอบวงจรดิจิตอลแต่ละชนิด

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB4 และ OSC เป็นอินพุต มีเซกเมนต์ DIGIT1 ในการแสดงผลผ่านทาง Y(6:0)โดยใช้เซกเมนต์ a, b, c, d, e, f และ g และต้องต่อขา COM ไปครร่วม DIGIT1 ลงกราวด์ กล่าวคือ

PB1 = PB1 = INPUT = p44	Y(0) = a = OUTPUT = p40	Y(5) = f = OUTPUT = p25
PB2 = PB2 = INPUT = p46	Y(1) = b = OUTPUT = p35	Y(6) = g = OUTPUT = p23
PB3 = PB3 = INPUT = p47	Y(2) = c = OUTPUT = p32	S1 = Dip SW1 = INPUT = p52
PB4 = PB4 = INPUT = p50	Y(3) = d = OUTPUT = p30	S0 = Dip SW2 = INPUT = p53
OSC_in = OSC = INPUT = p127	Y(4) = e = OUTPUT = p27	COM = DIGIT1 = OUTPUT = p31

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "COM" LOC = "p31" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "OSC_in" LOC = "p127" | IOSTANDARD = LVCMOS33 ;
NET "PB1" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "PB2" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "PB3" LOC = "p47" | IOSTANDARD = LVCMOS33 ;
NET "PB4" LOC = "p50" | IOSTANDARD = LVCMOS33 ;
NET "S0" LOC = "p53" | IOSTANDARD = LVCMOS33 ;
NET "S1" LOC = "p52" | IOSTANDARD = LVCMOS33 ;
NET "Y<0>" LOC = "p40" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Y<1>" LOC = "p35" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Y<2>" LOC = "p32" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Y<3>" LOC = "p30" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Y<4>" LOC = "p27" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Y<5>" LOC = "p25" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "Y<6>" LOC = "p23" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;

```

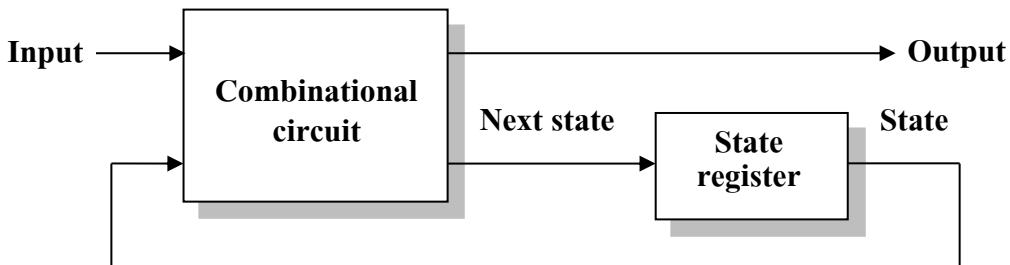
หลังจากโปรแกรม FPGA แล้วให้เซ็ต S1S0 = “00” (ON Dip SW1 และ ON Dip SW2) แล้วกดปุ่ม PB1 ไปเรื่อยๆ อย่างช้าๆ แล้วกดเร็วขึ้นเรื่อยๆ และกดค้างไว้ ให้สังเกตคุณที่ใช้เวนเชกเม้นต์ว่าติดสว่างโดยให้ลองเอาต์พุตเป็นตามทฤษฎีหรือไม่ หากนั้นจึงบันทึกผลการทดลอง แล้วทดลองซ้ำโดยเซ็ต S1S0 = “01”, “10”, และ “11” และกดปุ่ม PB2, PB3 และ PB4 ตามลำดับ

4.6 การออกแบบชีวนิรภัย

วงจรนับและชิฟตรีจิสเตอร์ที่ได้อธิบายไปแล้วคือเป็นวงจรซีเควนเชียล (Sequential circuit) อย่างง่าย ซึ่งในขอนี้เราจะอธิบายการออกแบบวงจรซีเควนเชียลโดยย่างเป็นระบบวิธี ซึ่งหมายความว่าการออกแบบวงจรที่มีความซับซ้อน วงจรซีเควนเชียลแบ่งเป็น 2 ชนิด คือ วงจรซีเควนเชียลแบบซิงโกรนัส (Synchronous sequential circuit) และวงจรซีเควนเชียลแบบอะซิงโกรนัส (Asynchronous sequential circuit) วงจรซีเควนเชียลแบบซิงโกรนัสนั้นจำเป็นต้องใช้สัญญาณนาฬิกา (Clock) จึงเรียกอีกอย่างหนึ่งว่า Clocked sequential machines และเนื่องจากจำนวน State ของวงจรซีเควนเชียลแบบซิงโกรนัสมีจำกัด ดังนั้นวงจรนี้อาจเรียกอีกอย่างหนึ่งว่า Finite state machine (FSM)

ในเบื้องต้นนี้เราจะอธิบายเฉพาะการออกแบบวงจรชีวคณ์เชิงลับแบบซิงโครนัสเท่านั้น ซึ่ง Xilinx synthesis tool หรือ XST จะรองรับการออกแบบวงจรชีวคณ์เชิงลับแบบซิงโครนัสนี้

วงจรซีเคนเนชีล (Sequential circuit หรือ Sequential machine) จะประกอบด้วย วงจรอคูมบินเด้น (Combinational circuit) และหน่วยความจำหรือรีจิสเตอร์ (State register) โดยที่เอาต์พุต (Output) หรือเอาต์พุตปัจจุบัน (Present output) ของวงจรจะขึ้นกับข้อมูลที่อยู่ในหน่วยความจำและอาจจะขึ้นกับอินพุต (Input) หรืออินพุตปัจจุบัน (Present input) อีกด้วย ซึ่งข้อมูลในหน่วยความจำนี้จะขึ้นกับอินพุตอดีต (Previous input) บล็อกไดอะแกรมของวงจรซีเคนเนชีลแสดงดังรูปที่ 4.92 ค่าของข้อมูลที่เก็บในหน่วยความจำ ปัจจุบัน เราเรียกว่า “State” หรือ “Present state” หรือ “Current state”

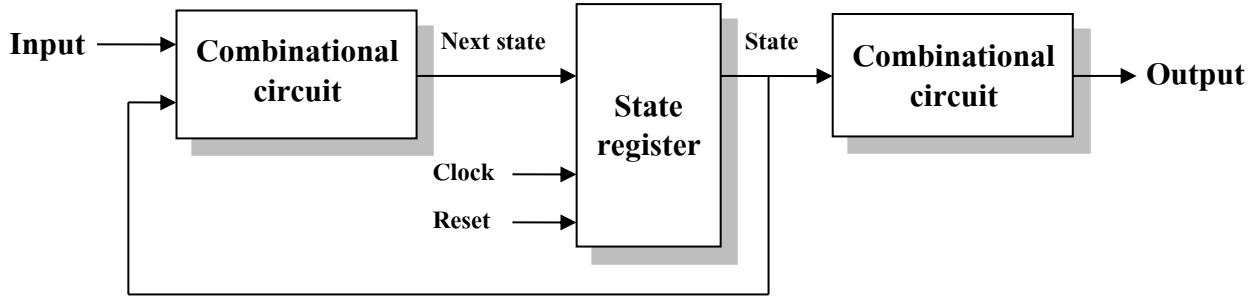


รูปที่ 4.92 บล็อกไกด์อะแกรมของวงจรซีเควนเชียลอย่างง่าย

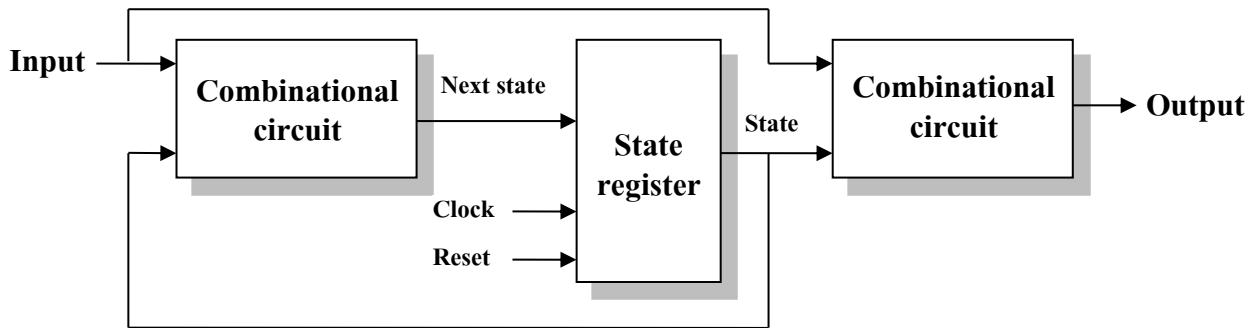
4.6.1 Finite state machine

Finite state machine (FSM) แบ่งตามชนิดของอาต์พุตได้ 2 ชนิด คือ มัวร์ (Moor-type FSM หรือ Moor machine) และ เมียลี (Mealy-type FSM หรือ Mealy machine) แสดงดังรูปที่ 4.93 และรูปที่ 4.94 ตามลำดับ โดยที่ Moor-type FSM นั้นอาต์พุต (ปัจจุบัน) จะขึ้นกับข้อมูลที่เก็บในหน่วยความจำ (State register) ณ ปัจจุบันหรือ State หรือ Present state เท่านั้น ในขณะที่ Mealy-type FSM นั้นอาต์พุต (ปัจจุบัน) จะขึ้นกับข้อมูลที่เก็บในหน่วยความจำ (State register) ณ ปัจจุบันและอินพุต (ปัจจุบัน)

ในรูปที่ 4.93 และรูปที่ 4.94 นั้นจะมี Clock และอาจมี Reset เพิ่มเข้ามาที่ State register ซึ่งในการออกแบบ FSM นั้น คำว่า Reset ในที่นี้จะมีความหมายเดียวกับ Clear หรือ Asynchronous clear ซึ่งเป็นข้อยกเว้นในหนังสือเล่มนี้เพื่อให้สอดคล้อง กับตำราของต่างๆ ที่มักจะใช้คำว่า น้อด เมื่อป้อนอินพุต (ถ้ามี) ให้กับ Finite state machine แล้วมีการทริก State register ด้วย Clock ก็จะทำให้วงจร มีการเปลี่ยน State เป็น State ใหม่ เอาต์พุตของ State ใหม่นี้จะถูกป้อนกลับไปที่อินพุตในส่วนของวงจร คอมบินेशัน (ด้านซ้าย) ทำให้ได้ค่า Next state ใหม่และค่าเอาต์พุตใหม่ จากนั้นวงจรจะวนกลับมาทำงานในลักษณะเป็นวัฏจักร ต่อไปเรื่อยๆ โดยที่ ไปแล้ว State register จะจะเป็น D Flip-flop หรือ JK Flip-flop หรือ Flip-flop แบบอื่นก็ได้ เช่นกัน



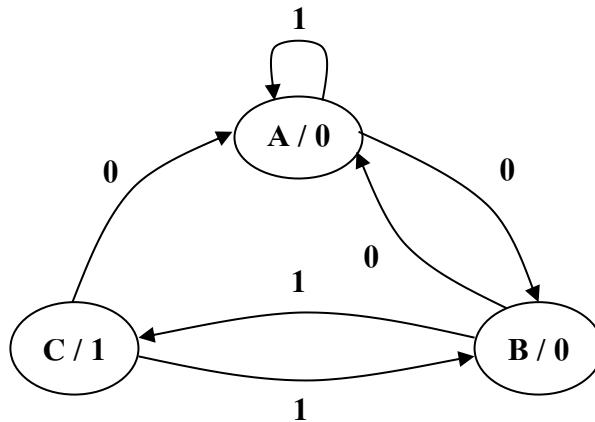
รูปที่ 4.93 บล็อกไซด์แกรมของ Moore-type FSM



รูปที่ 4.94 บล็อกไซด์แกรมของ Mealy-type FSM

4.6.2 State diagram และ State table ของ Moore-type FSM

State diagram และ State table ของ Moore-type FSM แสดงดังรูปที่ 4.95a) และรูปที่ 4.95b) ตามลำดับ ในรูปที่ 4.95a) นั้นด้วยการที่อยู่ภายในวงกลมเป็นชื่อ State และตัวเลขจะเป็นค่าเอต์พุต (ปัจจุบัน) เช่น State A วงจร มีเอต์พุต = '0' การเปลี่ยน State (State transition) จะเป็นตามทิศทางของลูกศร โดยทางลูกศรเป็น State ปัจจุบัน (Present state) และหัวลูกศรจะเป็น State ถัดไป (Next state) ตัวเลขที่กำกับใกล้เส้น (Transition) จะเป็นค่าอินพุต (ปัจจุบัน) หลักการทำงานของวงจร เช่น วงจรเริ่มต้นที่ State A และมีเอต์พุต Z = '0' ถ้าป้อนอินพุต X = '0' และมี Clock ไปทวิก วงจรจะเปลี่ยนไปที่ State B และที่ State B จะมีเอต์พุต Z = '0' ถ้าป้อนอินพุต X = '1' และมี Clock ไปทวิก วงจรจะเปลี่ยนไปที่ State C และที่ State C จะมีเอต์พุต Z = '1' ถ้าป้อนอินพุต X = '1' และมี Clock ไปทวิก วงจรจะเปลี่ยนกลับไปที่ State B ที่ State B จะมีเอต์พุต Z = '0'



(a) State diagram ของ Moor-type FSM

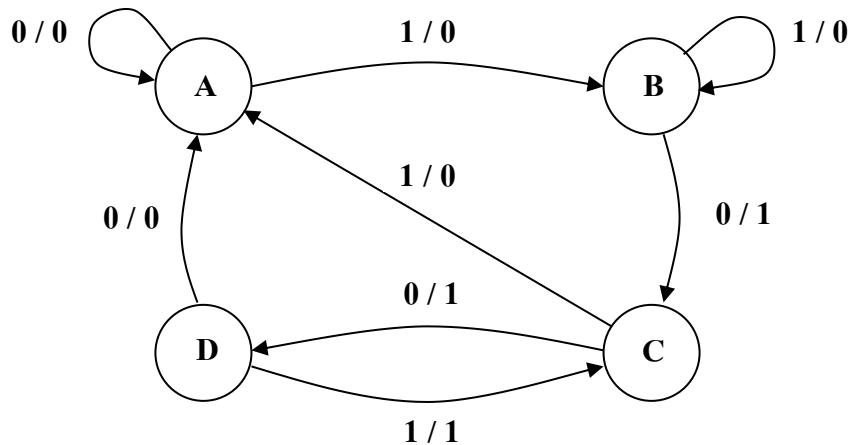
Present state	Next state		Output Z
	X = 0	X = 1	
A	B	A	0
B	A	C	0
C	A	B	1

(b) State table ของ Moor-type FSM

รูปที่ 4.95 State diagram และ State table ของ Moor-type FSM ที่มี X เป็นอินพุตและ Z เป็นเอาต์พุต

4.6.3 State diagram และ State table ของ Mealy-type FSM

การสร้าง State diagram และ State table ของ Mealy-type FSM ในการออกแบบวงจรซีกวนเชิงเดียว เพื่อให้เข้าใจง่ายขึ้น ให้ผู้อ่านพิจารณาตัวอย่างดังในรูปที่ รูปที่ 4.96(a) และรูปที่ 4.96(b) ตามลำดับ ในรูปที่ 4.96(a) นั้นจะมีวงกลมแสดง State ต่างๆ โดยมีตัวอักษรที่อยู่ภายในเป็นชื่อ State การเปลี่ยน State (State transition) จะเป็นตามทิศทางของหัวลูกศร โดยที่ ทางของลูกศรจะเป็น State ปัจจุบัน (Present state) ในขณะที่หัวของลูกศรจะเป็น State ถัดไป (Next state) ตัวเลขที่กำกับไว้ ใกล้เส้น(Transition) คือชัยะจะเป็นอินพุต (ปัจจุบัน) และค่าของชัยะจะเป็นเอาต์พุต (ปัจจุบัน) หลักการทำงานของวงจร เช่น เริ่มต้นที่ State A ถ้าป้อนอินพุต X = ‘1’ แล้วเอาต์พุต Z = ‘0’ และเมื่อมี Clock ไปทวิกรวงจรจะเปลี่ยนไปที่ State B ที่ State B ถ้า ป้อนอินพุต X = ‘0’ แล้วเอาต์พุต Z = ‘1’ และเมื่อมี Clock ไปทวิกรวงจรจะเปลี่ยนไปที่ State C ที่ State C ถ้าป้อนอินพุต X = ‘1’ แล้วเอาต์พุต Z = ‘0’ และเมื่อมี Clock ไปทวิกรวงจรจะเปลี่ยนไปที่ State A ที่ State A ถ้าป้อนอินพุต X = ‘0’ แล้วเอาต์พุต Z = ‘0’ และเมื่อมี Clock ไปทวิกรวงจรจะคงอยู่ที่ State A เราจะเห็นได้อย่างชัดเจนว่า Mealy-type FSM นั้นอินพุตจะมีผลต่อเอาต์พุต



(a) State diagram ของ Mealy-type FSM

Present State	Next state		Output Z	
	X = 0	X = 1	X = 0	X = 1
A	A	B	0	0
B	C	B	1	0
C	D	A	1	0
D	A	C	0	1

(b) State table ของ Mealy-type FSM

รูปที่ 4.96 State diagram และ State table ของ Mealy-type FSM ที่มี X เป็นอินพุตและ Z เป็นเอาต์พุต

4.6.4 ขั้นตอนการออกแบบ Finite state machine

ในหัวข้อนี้จะบททวนเกี่ยวกับขั้นตอนการออกแบบของ Finite state machine (FSM) ให้ผู้อ่านอีกครั้ง เพื่อให้คุณที่ไม่มีพื้นฐานการออกแบบของ FSM นี้จะได้เข้าใจในเบื้องต้น แต่ยังไรมีความลึกซึ้งเพิ่มเติม ขั้นตอนการออกแบบ Finite state machine สามารถสรุปได้ดังต่อไปนี้

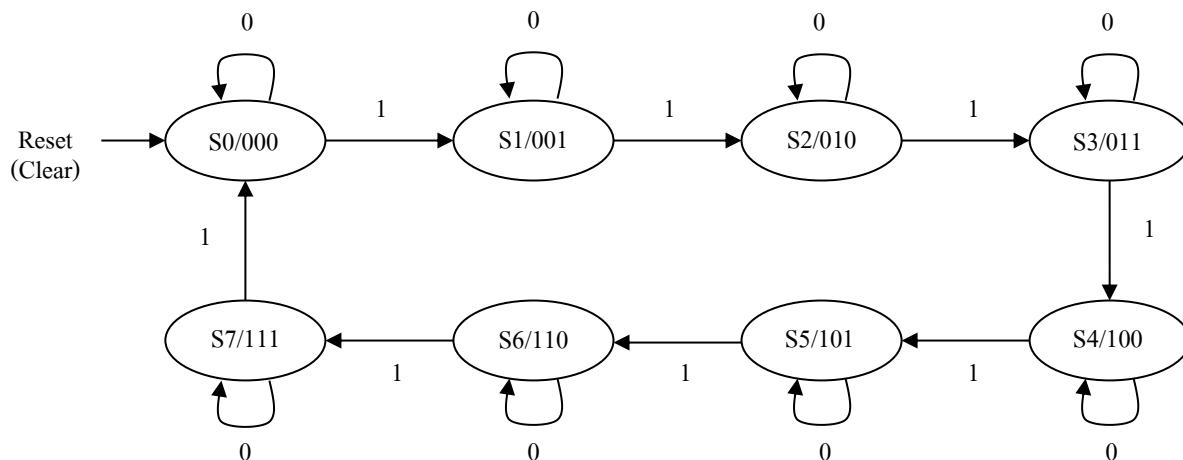
- 1) หากกำหนดหรือรายละเอียดของวงจรซีเควนเชียลแบบบิงโกรนัสหรือ Finite state machine (FSM) ให้ผู้อ่านอีกครั้ง เพื่อให้คุณที่ไม่มีพื้นฐานการออกแบบของ FSM นี้จะได้เข้าใจในเบื้องต้น แต่ยังไรมีความลึกซึ้งเพิ่มเติม ขั้นตอนการออกแบบ Finite state machine สามารถสรุปได้ดังต่อไปนี้
 - 1) หาข้อกำหนดหรือรายละเอียดของวงจรซีเควนเชียลแบบบิงโกรนัสที่ต้องการออกแบบ จากนั้นให้นำข้อมูลที่ได้ไปสร้าง State diagram และ/หรือ State table ซึ่งอาจจะเป็น Moore-type FSM หรือ Mealy-type FSM ในกรณีที่สามารถลดจำนวน State บาง State ที่ซ้ำซ้อนได้ (ถ้ามี) ก็ควรทำการลดจำนวน State (State reduction) ก่อน ซึ่งขั้นตอนการลด State สามารถอ่านได้จากหนังสือออกแบบดิจิตอลทั่วไปได้ ซึ่งการลด State ที่ซ้ำซ้อนออกไปอาจจะทำให้สามารถประยัดจำนวน Flip-flop และเกตต่างๆ ลงได้
 - 2) กำหนดค่าให้กับ State (State assignment หรือ State encoding) และ State ลงไว้ใน State table นั้น เราเรียกว่า State-assigned table จากนั้นจึงกำหนดชนิดและจำนวน Flip-flop ที่ใช้กับวงจรที่ออกแบบ และนำ State-assigned table ไปสร้างตารางความจริงเพื่อนำไปหาสมการบูลีนของ Next state ที่ใช้ป้อนให้กับอินพุตของ Flip-flop และหาสมการบูลีนของวงจรคอมบินेशันของ Next state ที่ได้มาเป็นข้อมูลในการสร้างเป็นวงจรตามที่เราต้องการออกแบบ
 - 3) ใช้สมการบูลีนที่ได้มาเป็นข้อมูลในการสร้างเป็นวงจรตามที่เราต้องการออกแบบ

การทำ State encoding นั้นจะต้องคำนึงถึงอุปกรณ์พื้นฐานภายใน CPLD และ FPGA ด้วย เพื่อใช้ทรัพยากรที่มีอยู่อย่างคุ้มค่าและวงจรคงมีสมรรถนะที่ดี (Speed) การประยัด Flip-flop อาจจะทำให้วงจรคอมบินेशันมีขนาดใหญ่และมีเวลาล่าช้า (Delay time) เพิ่มขึ้น โดยทั่วไป CPLD มีจำนวน Flip-flop จำกัด จึงการทำ State encoding เป็นรหัสไบนาเรีย (Binary) หรือเกรย์ (Gray) ซึ่งต่างจาก FPGA ที่มี Flip-flop เป็นจำนวนมาก จึงควรทำ State encoding แบบ “One-hot” เพื่อทำให้วงจรคอมบินेशันมีขนาดเล็กลงและวงจรมีสมรรถนะที่ดี (Speed)

ตัวอย่างที่ 4.1 ออกแบบวงจรนับ 8 แบบชิงโครนัสที่เป็นวงจรแบบนับขึ้นที่มีขา Clock enable input ($CE = X$) โดยมี Z เป็นเอาต์พุต (ประกอบด้วย Z_2, Z_1 และ Z_0) วงจรจะนับเมื่อ CE หรือ $X = '1'$ และ Clear (Reset) $Z = "000"$ เมื่อ Reset = ' 1 '

การสังเคราะห์วงจร

ขั้นตอนที่ 1 สร้าง State diagram และ State table ของวงจรนับ 8 แสดงดังรูปที่ 4.97(a) และรูปที่ 4.97(b) ซึ่งวงจรนับนี้ค่าเอาต์พุต Z จะขึ้นกับ State เท่านั้น จึงเป็น Moor-type FSM วงจรนับนี้ไม่มี State ที่ซ้ำซ้อน จึงไม่ต้องทำขั้นตอนลดจำนวน



(a) State diagram ของวงจรนับ 8 แบบนับขึ้นที่มีขา Clock enable input

Present state	Next state		Output Z
	X = 0	X = 1	
S0	S0	S1	000
S1	S1	S2	001
S2	S2	S3	010
S3	S3	S4	011
S4	S4	S5	100
S5	S5	S6	101
S6	S6	S7	110
S7	S7	S0	111

(b) State table ของวงจรนับ 8 แบบนับขึ้นที่มีขา Clock enable input

รูปที่ 4.97 State diagram และ State table ของวงจรนับ 8 แบบนับขึ้นที่มีขา Clock enable input (Moor-type FSM)

ขั้นตอนที่ 2 กำหนดค่าให้กับ State (State assignment หรือ State encoding) โดยในตัวอย่างนี้เราจะกำหนดเป็นเลขไบนาเรียหรือเลขฐานสองแสดงดังรูปที่ 4.98 ช่วงจrnabb 8 จะมี 8 State จึงใช้รีจิสเตอร์ 3 ตัว ($2^3 = 8$) โดยเราเลือกเป็น D Flip-flop ทั้ง 3 ตัว ถ้าให้อาต์พุตของ State หรือ Present state = $q_2q_1q_0$ และของ Next state = $q_2^*q_1^*q_0^*$ จากนั้นให้แทนค่า State (State assignment) ลงใน State table แล้วจะได้ State-assigned table ดังรูปที่ 4.99

ขอให้ผู้อ่านสังเกตว่าในกรณีที่ใช้ D Flip-flop เป็น State register นั้นอินพุตของ D Flip-flop คือ D ดังนั้นค่า Next state หรืออาต์พุตถัดไป (เมื่อทริกคือสัญญาณนาฬิกา) ของ D Flip-flop คือ q^* จะเป็นค่าๆ เดียวกับอินพุต D Flip-flop ดังนั้น $q^* = D$

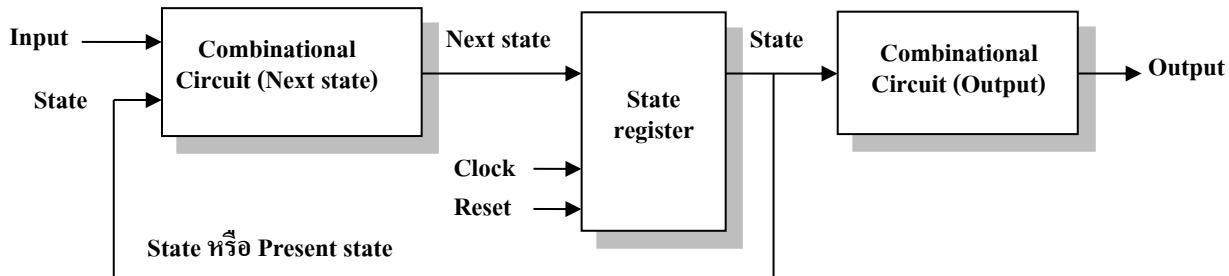
State	$q_2q_1q_0$
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110
S7	111

รูปที่ 4.98 State assignment ของชrnabb 8 แบบนับขึ้นที่มีขา Clock enable input

Present state = $q_2q_1q_0$	Next state = $q_2^*q_1^*q_0^*$		Output $Z=Z_2Z_1Z_0$
	X = 0	X = 1	
000	000	001	000
001	001	010	001
010	010	011	010
011	011	100	011
100	100	101	100
101	101	110	101
110	110	111	110
111	111	000	111

รูปที่ 4.99 State-assigned table ของชrnabb 8 แบบนับขึ้นที่มีขา Clock enable input

นำรูปที่ 4.93 มาเขียนใหม่ได้ดังรูปที่ 4.100 หลังจากที่เลือกรีจิสเตอร์ (State register) เป็น D Flip-flop ได้แล้ว สิ่งที่เราต้องทำต่อไป คือ หาสมการบูลีนของชrnabb 2 วงจรในรูปที่ 4.100 โดยที่สมการวงจรคอมบินेशันของ Next state ที่ใช้ป้อนให้อินพุตของ State register (ที่เป็น D Flip-flop 3 ตัว คือ $q_2^*q_1^*q_0^*$) นั้นจะมาจากการของอาต์พุต (คือ X) และมาจาก State (Present state คือ $q_2q_1q_0$) ส่วนสมการวงจรคอมบินेशันของอาต์พุต (Moor-type FSM) นั้นจะมาจากการของ State เพียงอย่างเดียว จากนั้นนำรูปที่ 4.99 มาเขียนใหม่ได้ดังรูปที่ 4.101(a) และรูปที่ 4.101(b)



รูปที่ 4.100 บล็อกไซด์ของ Moer-type FSM

Input & State	Next state
Xq2q1q0	q2*q1*q0*
0 0 0 0	0 0 0
0 0 0 1	0 0 1
0 0 1 0	0 1 0
0 0 1 1	0 1 1
0 1 0 0	1 0 0
0 1 0 1	1 0 1
0 1 1 0	1 1 0
0 1 1 1	1 1 1
1 0 0 0	0 0 1
1 0 0 1	0 1 0
1 0 1 0	0 1 1
1 0 1 1	1 0 0
1 1 0 0	1 0 1
1 1 0 1	1 1 0
1 1 1 0	1 1 1
1 1 1 1	0 0 0

(a) อินพุต-เอาต์พุตของจุดน้ำหนึ่งของ Next state

State q2q1q0	Output Z = Z2Z1Z0
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 0
1 1 1	1 1 1

(b) อินพุต-เอาต์พุตของจุดน้ำหนึ่งของเอต์พุต

รูปที่ 4.101 ตารางความจริงของจุดน้ำหนึ่งของ Next state และจุดน้ำหนึ่งของเอต์พุต

จากนี้จึงนำตารางความจริงในรูปที่ 4.101(a) และรูปที่ 4.101(b) ไปเขียนเป็น Excitation map และ Output map ดังนี้

- Excitation map ของ Next state q2* รูปที่ 4.102(a) ได้จากคอลัมน์ Input & State และช้ายสุดของ Next state ในรูปที่ 4.101(a)
- Excitation map ของ Next state q1* รูปที่ 4.102(b) ได้จากคอลัมน์ Input & State และกลางของ Next state ในรูปที่ 4.101(a)
- Excitation map ของ Next state q0* รูปที่ 4.102(c) ได้จากคอลัมน์ Input & State และขวาสุดของ Next state ในรูปที่ 4.101(a)
- Output map ของเอต์พุต Z2 รูปที่ 4.102(d) ได้จากคอลัมน์ State และจากคอลัมน์ช้ายสุดของ Output ในรูปที่ 4.101(b)
- Output map ของเอต์พุต Z1 รูปที่ 4.102(e) ได้จากคอลัมน์ State และจากคอลัมน์กลางของ Output ในรูปที่ 4.101(b)
- Output map ของเอต์พุต Z0 รูปที่ 4.102(f) ได้จากคอลัมน์ State และจากคอลัมน์ขวาสุดของ Output ในรูปที่ 4.101(b)

		Xq2				
		00	01	11	10	
q1q0		00	0	1	1	0
01	q1q0	0	1	1	0	
		11	0	1	0	
10	q1q0	0	1	1	0	

(a) Excitation map ของ Next state q2*

		Xq2				
		00	01	11	10	
q1q0		00	0	0	0	0
01	q1q0	0	0	1	1	
		11	1	1	0	
10	q1q0	1	1	1	1	

(b) Excitation map ของ Next state q1*

		Xq2				
		00	01	11	10	
q1q0		00	0	0	1	1
01	q1q0	1	1	0	0	
		11	1	0	0	
10	q1q0	0	0	1	1	

(c) Excitation map ของ Next state q0*

		Z2		
		0	1	
q1q0		00	0	1
01	q1q0	0	1	
		11	0	
10	q1q0	0	1	

(d) Output map ของเอาต์พุต Z2

		q2		
		0	1	
q1q0		00	0	0
01	q1q0	0	0	
		11	1	
10	q1q0	1	1	

(e) Output map ของเอาต์พุต Z1

		Z0		
		0	1	
q1q0		00	0	0
01	q1q0	1	1	
		11	1	
10	q1q0	0	0	

(f) Output map ของเอาต์พุต Z0

รูปที่ 4.102 Excitation map ของวงจร Next state และ Output map ของวงจรเอาต์พุต

จากรูปที่ 4.37(a) ถึงรูปที่ 4.37(f) สามารถเขียนเป็นสมการบูลีนได้ดังนี้

$$q2^* = X \cdot \overline{q2} \cdot q1 \cdot q0 + \overline{X} \cdot q2 + q2 \cdot \overline{q1} + q2 \cdot \overline{q0}$$

$$= X \cdot q0 \cdot q1 \cdot \overline{q2} + (\overline{X} + \overline{q1} + \overline{q0}) \cdot q2 = X \cdot q1 \cdot q0 \cdot \overline{q2} + \overline{X} \cdot q1 \cdot q0 \cdot q2 = (Xq1q0) \oplus q2$$

$$q1^* = X \cdot \overline{q1} \cdot q0 + \overline{X} \cdot q1 + q1 \cdot \overline{q0}$$

$$= X \cdot \overline{q1} \cdot q0 + (\overline{X} + \overline{q0}) \cdot q1 = X \cdot q0 \cdot \overline{q1} + \overline{X} \cdot q0 \cdot q1 = (X \cdot q0) \oplus q1$$

$$q0^* = \overline{X} \cdot q0 + X \cdot \overline{q0}$$

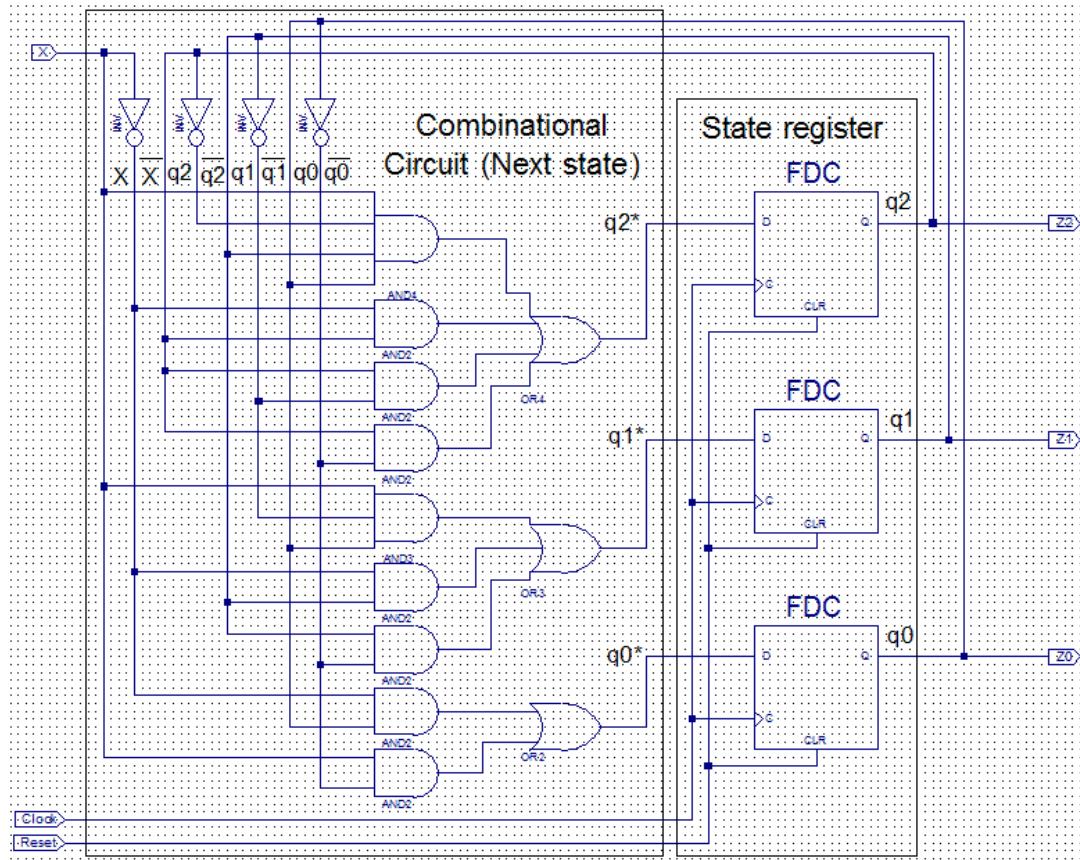
$$= X \oplus q0$$

$$Z2 = q2$$

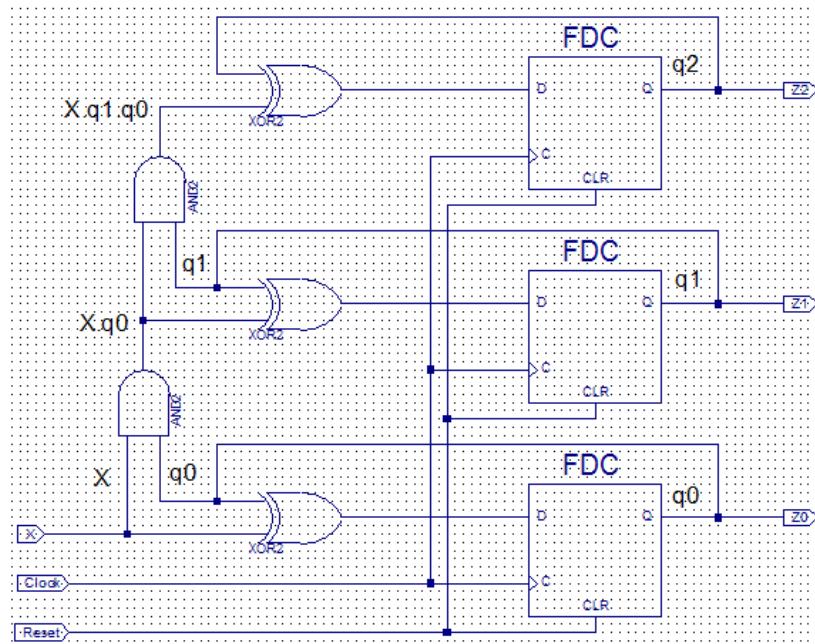
$$Z1 = q1$$

$$Z0 = q0$$

ບັນຄອນທີ 3 ໃຫ້ສ່າມການນູ້ຄືນວ່າຮຽກນົມເນັນຂັ້ນຂອງ Next state ແລະ ຂອງເອົາຕຸພຸດທີ່ໄດ້ປັບປຸງທີ່ 4.103 ແລະ ຮູບທີ່ 4.104 ດັ່ງນັ້ນການອອກແນບນັ້ນຍ່າງເປັນຮະບວນວິທີໃນຂໍ້ 4.6 ນີ້ຈະໃຫ້ຜູ້ລັບຜູ້ເຊື່ອເວັບວ່າງຈາກຮັນທີ່ອີນາຍໃນຂໍ້ 4.3 ຖຸກປະກາດ



ຮູບທີ່ 4.103 ຜັງຈຽນ 8 ແບນັ້ນຂຶ້ນທີ່ມີຂາ Clock enable input



ຮູບທີ່ 4.104 ຜັງຈຽນ 8 ແບນັ້ນຂຶ້ນທີ່ມີຂາ Clock enable input ໂດຍທີ່ XOR ແລະ D Flip-flop ກີ່ອີ້ນ T Flip-flop

ໃນບັນຄອນທີ່ 2 ລ້ານເລືອກ State register ເປັນ JK Flip-flop ແກນການໃຫ້ D Flip-flop ດັ່ງນັ້ນເຮົາຈຶ່ງເປັນຕົ້ນດັດແປງຕາງໆ ມາຮັດວຽກຮຽກນົມເນັນຂັ້ນຂອງ Next state ໃນຮູບທີ່ 4.101a) ເສີຍໄໝ່ ໂດຍເພີ່ມຄອລັນ Input ທີ່ຕົ້ນການຂອງ JK Flip-flop ແຕ່

ผลตัวที่ทำให้อาต์พุต Next state ของ JK Flip-flop ได้ผลลัพธ์เช่นเดียวกับ D Flip-flop โดย Input ที่ต้องการคือ อินพุต J และ K ของ Flip-flop และตัวจะต้องสอดคล้องกับ Excitation table ของ JK Flip-Flop ในรูปที่ 4.105 ส่วนวงจรคอมบินेशันของอาต์พุตจะยังคงเหมือนเดิมดังในรูปที่ 4.101(b) ตารางความจริงวงจรคอมบินेशันของ Next state และของอาต์พุตที่ได้แสดงดังรูปที่ 4.106

Transition at output	Present state q	Next state q*	Input ที่ต้องการ	
			J	K
0 → 0	0	0	0	d
0 → 1	0	1	1	d
1 → 0	1	0	d	1
1 → 1	1	1	d	0

รูปที่ 4.105 Excitation table ของ JK Flip-Flop

Input & State	Next state	Input ที่ต้องการ		
		J2K2	J1K1	J0K0
Xq2q1q0	q2*q1*q0*			
0 0 0 0	0 0 0	0 d	0 d	0 d
0 0 0 1	0 0 1	0 d	0 d	d 0
0 0 1 0	0 1 0	0 d	d 0	0 d
0 0 1 1	0 1 1	0 d	d 0	d 0
0 1 0 0	1 0 0	d 0	0 d	0 d
0 1 0 1	1 0 1	d 0	0 d	d 0
0 1 1 0	1 1 0	d 0	d 0	0 d
0 1 1 1	1 1 1	d 0	d 0	d 0
1 0 0 0	0 0 1	0 d	0 d	1 d
1 0 0 1	0 1 0	0 d	1 d	d 1
1 0 1 0	0 1 1	0 d	d 0	1 d
1 0 1 1	1 0 0	1 d	d 1	d 1
1 1 0 0	1 0 1	d 0	0 d	1 d
1 1 0 1	1 1 0	d 0	1 d	d 1
1 1 1 0	1 1 1	d 0	d 0	1 d
1 1 1 1	0 0 0	d 1	d 1	d 1

State q2q1q0	Output Z = Z2Z1Z0
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 0
1 1 1	1 1 1

(a) อินพุต-อาต์พุตวงจรคอมบินेशันของ Next state (b) อินพุต-อาต์พุตวงจรคอมบินेशันของอาต์พุต
รูปที่ 4.106 ตารางความจริงวงจรคอมบินेशันของ Next state และวงจรคอมบินेशันของอาต์พุต

จากนั้นนำตารางความจริงในรูปที่ 4.106(a) และรูปที่ 4.106(b) ไปเขียนเป็น Excitation map และ Output map ต่างๆ ดังนี้

- Excitation map ของ J2 รูปที่ 4.107(a) ได้จากการอ่าน Input & State และ J2 ในรูปที่ 4.106(a)
- Excitation map ของ K2 รูปที่ 4.107(b) ได้จากการอ่าน Input & State และ K2 ในรูปที่ 4.106(a)
- Excitation map ของ J1 รูปที่ 4.107(c) ได้จากการอ่าน Input & State และ J1 ในรูปที่ 4.106(a)
- Excitation map ของ K1 รูปที่ 4.107(d) ได้จากการอ่าน Input & State และ K1 ในรูปที่ 4.106(a)
- Excitation map ของ J0 รูปที่ 4.107(e) ได้จากการอ่าน Input & State และ J0 ในรูปที่ 4.106(a)
- Excitation map ของ K0 รูปที่ 4.107(f) ได้จากการอ่าน Input & State และ K0 ในรูปที่ 4.106(a)
- Output map ของอาต์พุต Z2 รูปที่ 4.107(g) ได้จากการอ่าน State และจากการอ่านชี้สุดของ Output ในรูปที่ 4.106(b)
- Output map ของอาต์พุต Z1 รูปที่ 4.107(h) ได้จากการอ่าน State และจากการอ่านชี้สุดของ Output ในรูปที่ 4.106(b)
- Output map ของอาต์พุต Z0 รูปที่ 4.107(i) ได้จากการอ่าน State และจากการอ่านชี้สุดของ Output ในรูปที่ 4.106(b)

		Xq2				
		00	01	11	10	
q1q0		00	0	d	d	0
		01	0	d	d	0
q1q0		11	0	d	d	1
		10	0	d	d	0

(a) Excitation map ของ Next state J2

		Xq2				
		00	01	11	10	
q1q0		00	d	0	0	d
		01	d	0	0	d
q1q0		11	d	0	1	d
		10	d	0	0	d

(b) Excitation map ของ Next state K2

		Xq2				
		00	01	11	10	
q1q0		00	0	0	0	0
		01	0	0	1	1
q1q0		11	d	d	d	d
		10	d	d	d	d

(c) Excitation map ของ Next state J1

		Xq2				
		00	01	11	10	
q1q0		00	d	d	d	d
		01	d	d	d	d
q1q0		11	0	0	1	1
		10	0	0	0	0

(d) Excitation map ของ Next state K1

		Xq2				
		00	01	11	10	
q1q0		00	0	0	1	1
		01	d	d	d	d
q1q0		11	d	d	d	d
		10	0	0	1	1

(e) Excitation map ของ Next state J0

		Xq2				
		00	01	11	10	
q1q0		00	d	d	d	d
		01	0	0	1	1
q1q0		11	0	0	1	1
		10	d	d	d	d

(f) Excitation map ของ Next state K0

		q2		
		0	1	
q1q0		00	1	
		01		1
q1q0		11		1
		10		1

(g) Output map ของเอาต์พุต Z2

		q2		
		0	1	
q1q0		00	0	0
		01	0	0
q1q0		11	1	1
		10	1	1

(h) Output map ของเอาต์พุต Z1

		q2		
		0	1	
q1q0		00	0	0
		01	1	1
q1q0		11	1	1
		10	0	0

(i) Output map ของเอาต์พุต Z0

รูปที่ 4.107 Excitation map ของวงจร Next state และ Output map ของวงจรเอาต์พุต

จากรูปที่ 4.107(a) ถึงรูปที่ 4.107(i) สามารถเปลี่ยนเป็นสมการบูลีนได้ดังนี้

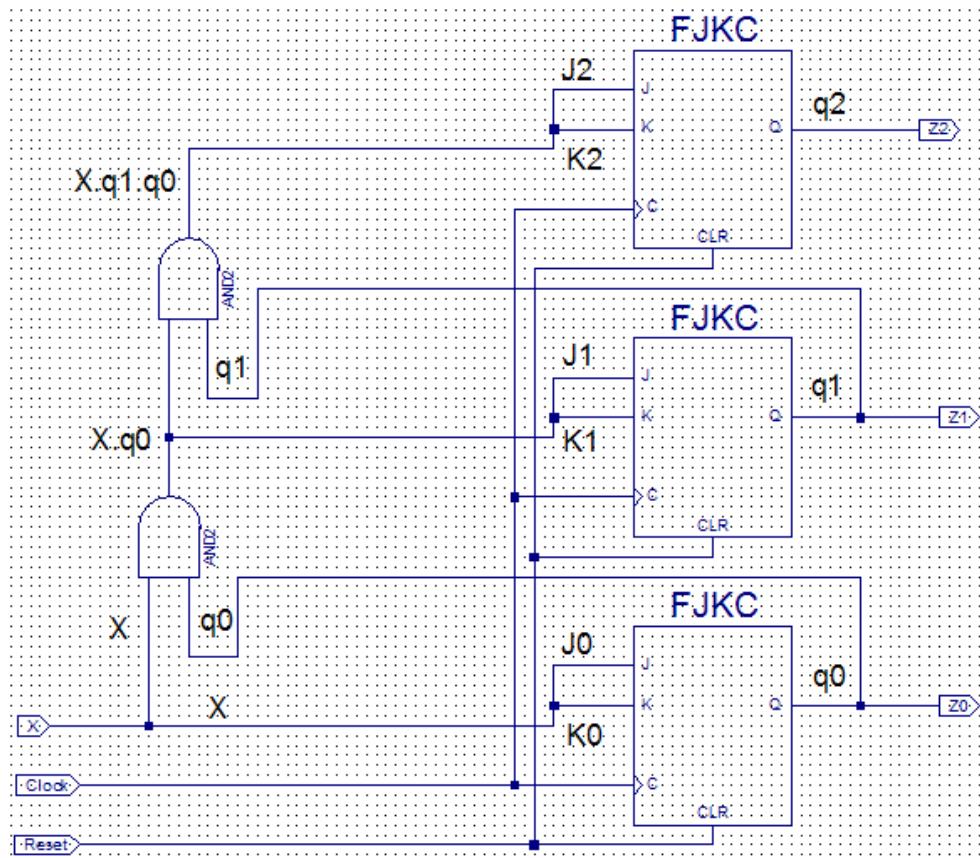
$$J_2 = X \cdot q_1 \cdot q_0 \quad \text{และ} \quad K_2 = X \cdot q_1 \cdot q_0$$

$$J_1 = X \cdot q_0 \quad \text{และ} \quad K_1 = X \cdot q_0$$

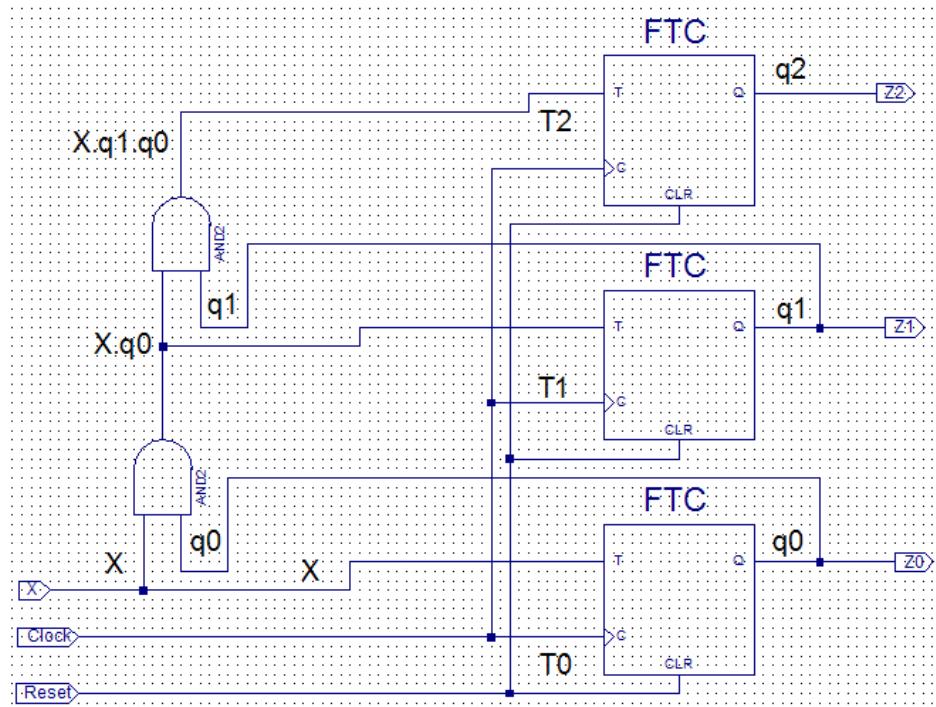
$$J_0 = X \quad \text{และ} \quad K_0 = X$$

$$Z_2 = q_2 \quad Z_1 = q_1 \quad \text{และ} \quad Z_0 = q_0$$

จากนั้นทำขั้นตอนที่ 3 ใช้สมการบูลีนของรากอนบินชันของ Next state และเอาต์พุตที่ได้ไปสร้างวงจรได้ดังรูปที่ 4.108 และรูปที่ 4.109 ที่เขียนในรูปของ T Flip-flop ซึ่งจะให้ผลลัพธ์เช่นเดียวกับวงจรนับที่อธิบายในข้อ 4.3 ทุกประการ

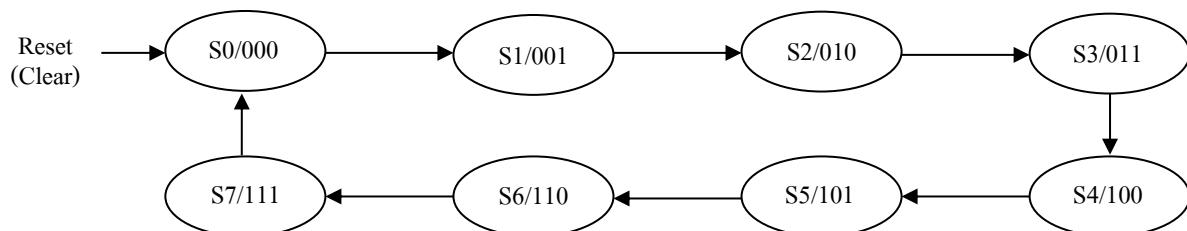


รูปที่ 4.108 ผังวงจรนับ 8 แบบบูลีนที่มีขา Clock enable input ซึ่งออกแบบโดยใช้ JK Flip-flop



รูปที่ 4.109 ผังวงจรนับ 8 แบบนับขึ้นที่มีขา Clock enable input ซึ่งออกแบบโดยใช้ T Flip-flop

ในทางปฏิบัติ้นว่างจรนับทั่วไปอาจจะมีเฉพาะเอาต์พุต Z ประกอบด้วย Z2, Z1 และ Z0 (โดยไม่มีอินพุต ดังนั้น State diagram และ State table จึงเป็นกรณีพิเศษที่ไม่มีอินพุต ซึ่ง State diagram และ State table ของวงจรนับแบบชิงโกรนั้นนับ 8 แบบนับขึ้นแสดงดังรูปที่ 4.110a) และรูปที่ 4.110b) จากนั้นจึงเปลี่ยนตารางความจริงของรวมบินเนชันของ Next state และของเอาต์พุต และรูปที่ 4.111a) และรูปที่ 4.111b) เพื่อนำไปหาสมการบูลีนสร้างวงจรที่ต้องการต่อไป



(a) State diagram ของวงจรนับ 8 แบบนับขึ้น

Present state	Next state		Output Z
	X = 1	X = 0	
S0	S1	S0	000
S1	S2	S1	001
S2	S3	S2	010
S3	S4	S3	011
S4	S5	S4	100
S5	S6	S5	101
S6	S7	S6	110
S7	S0	S7	111

(b) State table ของวงจรนับ 8 แบบนับขึ้น

รูปที่ 4.110 State diagram และ State table ของวงจรนับ 8 แบบนับขึ้น (Moor-type FSM)

State	Next state
q2q1q0	q2*q1*q0*
0 0 0	0 0 1
0 0 1	0 1 0
0 1 0	0 1 1
0 1 1	1 0 0
1 0 0	1 0 1
1 0 1	1 1 0
1 1 0	1 1 1
1 1 1	0 0 0

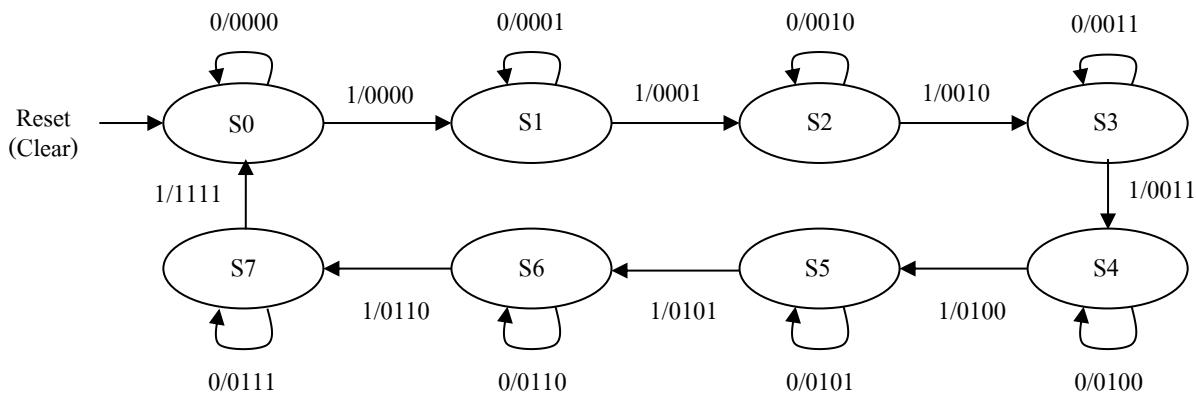
State	Output Z
q2q1q0	Z2Z1Z0
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 0
1 1 1	1 1 1

(a) อินพุต-เอาต์พุตของวงจรคอมบินेशันของ Next state (b) อินพุต-เอาต์พุตของวงจรคอมบินेशันของเอาต์พุต

รูปที่ 4.111 ตารางความจริงของวงจรคอมบินेशันของ Next state และวงจรคอมบินेशันของเอาต์พุต

ตัวอย่างที่ 4.2 ให้ออกแบบวงจรนับ 8 แบบซิงโครนัสที่เป็นวงจรแบบนับขึ้นที่มีขา Clock enable input (CE = X) และขา Clock enable output (CEO = Z3) โดยที่ Z เป็นเอาต์พุต (ประกอบด้วย Z3, Z2, Z1 และ Z0) วงจรจะนับเมื่อ CE หรือ X = '1' และจะ Clear (Reset) เอาต์พุตเมื่อ Reset = '1' และเมื่อวงจรนับถึงค่าเอาต์พุตสูงสุดแล้วถ้าขา CE = '1' ก็จะให้เอาต์พุตขา Z3 = '1'

ขั้นตอนที่ 1 สร้าง State diagram และ State table ของวงจรนับ 8 แสดงดังรูปที่ 4.112(a) และรูปที่ 4.112(b) ตามลำดับ ซึ่งในกรณีของวงจรนับนี้ค่าเอาต์พุต Z3 จะขึ้นกับ State และอินพุต ดังนั้นวงจรนี้จึงเป็น Mealy-type FSM



(a) State diagram ของวงจรนับ 8 มีขา Clock enable output

Present state	Next state		Output Z	
	X = 0	X = 1	X = 0	X = 1
S0	S0	S1	0000	0000
S1	S1	S2	0001	0001
S2	S2	S3	0010	0010
S3	S3	S4	0011	0011
S4	S4	S5	0100	0100
S5	S5	S6	0101	0101
S6	S6	S7	0110	0110
S7	S7	S0	0111	1111

(b) State table ของวงจรนับ 8 มีขา Clock enable output

รูปที่ 4.112 State diagram และ State table ของวงจรนับ 8 มีขา Clock enable output (Mealy-type FSM)

ข้อตอนที่ 2 กำหนดค่า State เป็นเลขไบนารีดังรูปที่ 4.113 และเลือก State register เป็น D Flip-flop แทนค่าในรูปที่ 4.112(b) แล้วจะได้ดังรูปที่ 4.114 จากนั้นเขียนใหม่ดังรูปที่ 4.115(a) และรูปที่ 4.115(b) เพื่อหาสมการบูลีนของรุ่นบินชัน Next state และเอาต์พุต

State	$q_2q_1q_0$
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110
S7	111

รูปที่ 4.113 State assignment ของวงจรนับ 8

Present state = $q_2q_1q_0$	Next state = $q_2^*q_1^*q_0^*$		Output Z = $Z_3Z_2Z_1Z_0$	
	X = 0	X = 1	X = 0	X = 1
000	000	001	0000	0000
001	001	010	0001	0001
010	010	011	0010	0010
011	011	100	0011	0011
100	100	101	0100	0100
101	101	110	0101	0101
110	110	111	0110	0110
111	111	000	0111	1111

รูปที่ 4.114 State-assigned table ของวงจรนับ 8

Input & State	Next state
$Xq_2q_1q_0$	$q_2^*q_1^*q_0^*$
0 0 0 0	0 0 0
0 0 0 1	0 0 1
0 0 1 0	0 1 0
0 0 1 1	0 1 1
0 1 0 0	1 0 0
0 1 0 1	1 0 1
0 1 1 0	1 1 0
0 1 1 1	1 1 1
1 0 0 0	0 0 1
1 0 0 1	0 1 0
1 0 1 0	0 1 1
1 0 1 1	1 0 0
1 1 0 0	1 0 1
1 1 0 1	1 1 0
1 1 1 0	1 1 1
1 1 1 1	0 0 0

(a) อินพุต-เอาต์พุตของรุ่นบินชันของ Next state

Input & State	Output Z
$Xq_2q_1q_0$	$Z_3Z_2Z_1Z_0$
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 0
0 0 1 1	0 0 1 1
0 1 0 0	0 1 0 0
0 1 0 1	0 1 0 1
0 1 1 0	0 1 1 0
0 1 1 1	0 1 1 1
1 0 0 0	0 0 0 0
1 0 0 1	0 0 0 1
1 0 1 0	0 0 1 0
1 0 1 1	0 0 1 1
1 1 0 0	0 1 0 0
1 1 0 1	0 1 0 1
1 1 1 0	0 1 1 0
1 1 1 1	1 1 1 1

(b) อินพุต-เอาต์พุตของรุ่นบินชันของเอาต์พุต

รูปที่ 4.115 ตารางความจริงของรุ่นบินชันของ Next state และรุ่นบินชันของเอาต์พุต

จากนั้นจึงนำตารางความจริงในรูปที่ 4.115(a) และรูปที่ 4.115(b) ไปเป็น Excitation map และ Output map ต่างๆ ดังนี้

- Excitation map จะเหมือนกับตัวอย่างที่ 4.1 ดังนั้นสมการบูลีน $q2^*, q1^*$ และ $q0^*$ เหมือนกับตัวอย่างที่ 4.1 ทุกประการ
- Output map ของเอาต์พุต $Z3, Z2, Z1$ และ $Z0$ ดังในรูปที่ 4.116 จะได้จากคอลัมน์ State และ Output ในรูปที่ 4.115(b)

		Xq2				
		00	01	11	10	
q1q0		00	0	0	0	0
00		00	0	0	0	0
01		01	0	0	0	0
11		11	0	0	1	0
10		10	0	0	0	0

(a) Output map ของเอาต์พุต $Z3$

		Xq2				
		00	01	11	10	
q1q0		00	0	1	1	0
00		00	0	1	1	0
01		01	0	1	1	0
11		11	0	1	1	0
10		10	0	1	1	0

(b) Output map ของเอาต์พุต $Z2$

		Xq2				
		00	01	11	10	
q1q0		00	0	0	0	0
00		00	0	0	0	0
01		01	0	0	0	0
11		11	1	1	1	1
10		10	1	1	1	1

(c) Output map ของเอาต์พุต $Z1$

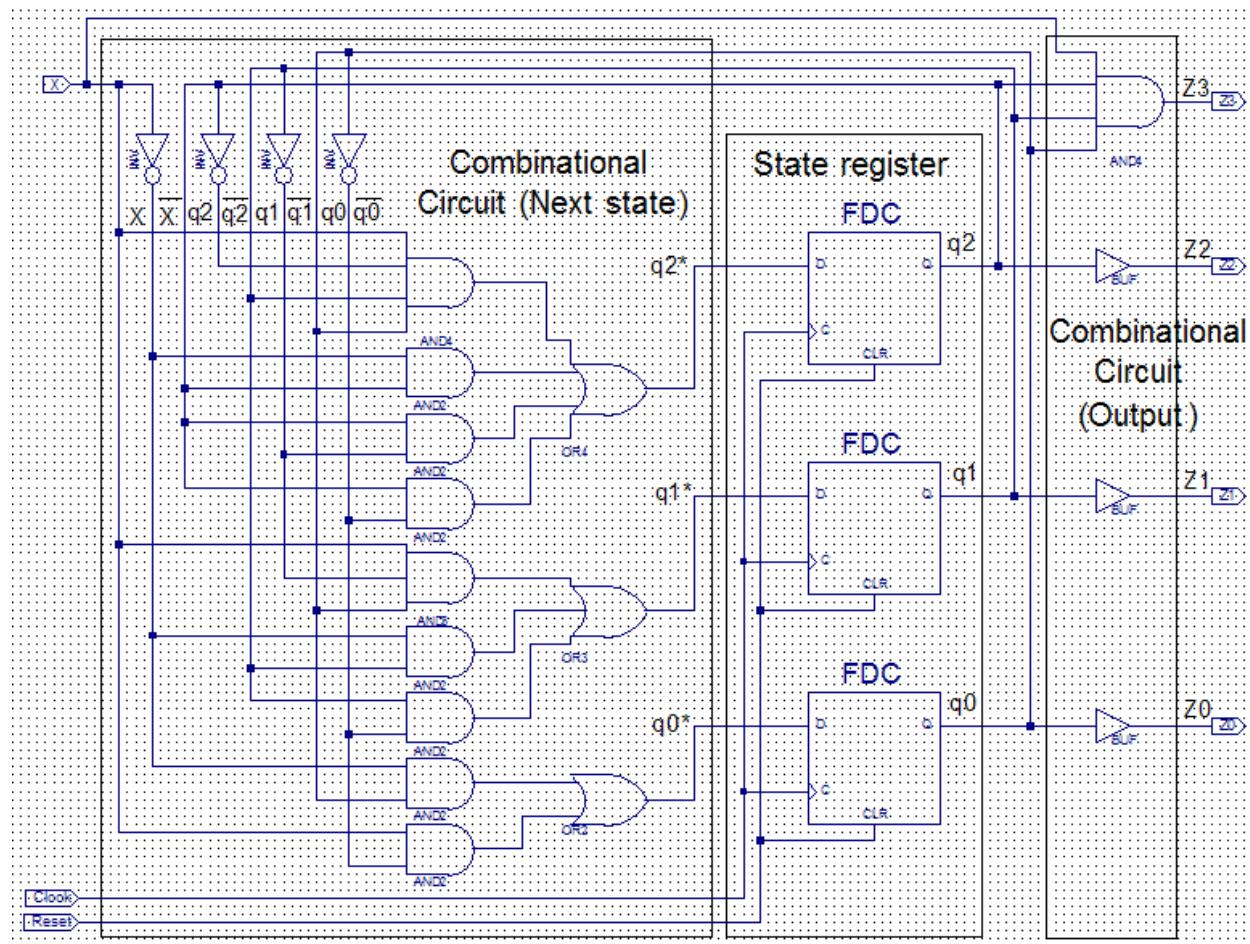
		Xq2				
		00	01	11	10	
q1q0		00	0	0	0	0
00		00	1	1	1	1
01		01	1	1	1	1
11		11	1	1	1	1
10		10	0	0	0	0

(d) Output map ของเอาต์พุต $Z0$

รูปที่ 4.116 Excitation map ของวงจร Next state และ Output map ของวงจรเอาต์พุต

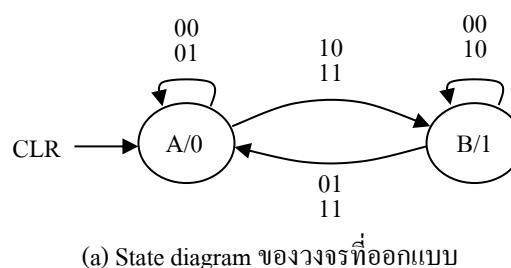
$$\text{จะได้ } Z3 = X.q2.q1.q0, \quad Z2 = q2, \quad Z1 = q1 \quad \text{และ} \quad Z0 = q0$$

ข้อตอนที่ 3 ใช้สมการบูลีนของ Next state และของเอาต์พุตที่ได้มานี้เป็นข้อมูลในการสร้างวงจรได้ดังรูปที่ 4.117



รูปที่ 4.117 ผังวงจรนับ 8 ที่มีขา Clock enable Output

ตัวอย่างที่ 4.3 ให้ออกแบบวงจรซิงโกรนัสที่มี State diagram และ State table ดังรูปที่ 4.118 มี J และ K เป็นอินพุต วงจรนี้จะ Clear $Q = '0'$ เมื่อ $CLR = '1'$ และจะเปลี่ยน State ทุกครั้งที่ทริกด้วยขอบนาฬิกของ Clock (JK)CLK(กำหนดให้อินพุต $X = JK$



(a) State diagram ของวงจรที่ออกแบบ

Present state	Next state				Output
	$X = 00$	$X = 01$	$X = 10$	$X = 11$	
A	A	A	B	B	0
B	B	A	B	A	1

(b) State table ของวงจรที่ออกแบบ

รูปที่ 4.118 State diagram และ State table ของวงจรที่ออกแบบ (Moor-type FSM)

จากนั้นกำหนดค่า State เป็นเลขไบนารีแสดงดังรูปที่ 4.119 ซึ่งวงจรมี 2 State จึงใช้รีจิสเตอร์ 1 ตัว ($2^1 = 2$) โดยเลือกเป็น D Flip-flop กำหนดให้อเอาต์พุตของ (D Flip-flop) State หรือ Present state = q_0 และของ Next state = q_0^* จากนั้นให้แทนค่า State ลงใน State table และจะได้ State-assigned table ดังรูปที่ 4.120

State	q0
S0	0
S1	1

รูปที่ 4.119 State assignment ของวงจรนับ 8

Present state	Next state				Output
	X = 00	X = 01	X = 10	X = 11	
0	0	0	1	1	0
1	1	0	1	0	1

รูปที่ 4.120 State-assigned table ของวงจรที่ออกแบบ

นำรูปที่ 4.120 มาเขียนใหม่ดังรูปที่ 4.121(a) และรูปที่ 4.121(b) แล้วหาสมการบูลีนของ Next state ที่ใช้ป้อนให้อินพุตของ State register ซึ่งมาจากอินพุตภายนอก (Input) และ State (Present state) ส่วนสมการของเอาต์พุตนั้นจะมาจากการ State เท่านั้น

Input & State	Next state
J K q0	q0*
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	1
1 0 1	1
1 1 0	1
1 1 1	0

(a) อินพุต-เอาต์พุตของวงจร Next state

State	Output
q0	Q
0	0
1	1

(b) อินพุต-เอาต์พุตของวงจรเอาต์พุต

รูปที่ 4.121 ตารางความจริงของวงจร Next state และวงจรเอาต์พุต

จากนั้นจึงนำตารางความจริงในรูปที่ 4.121(a) และรูปที่ 4.121(b) ไปเขียนเป็น Excitation map และ Output ดังนี้

- จากรูปที่ 4.121(b) จะได้ $Q = q0$
- Excitation map ของ Next state $q0^*$ ดังรูปที่ 4.122 นั้นจะหาได้จากการอ้อมน์ Input & State และ Next state ในรูปที่ 4.121(a)

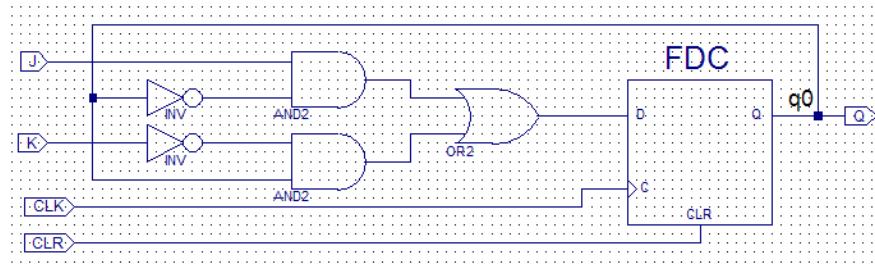
q0*		JK			
q0		00	01	11	10
0	0	0	1	1	
1	1	0	0	1	

รูปที่ 4.122 Excitation map ของวงจร Next state $q0^*$

จากรูปที่ 4.122 สามารถเขียนเป็นสมการบูลีนของ Next state ได้ดังนี้

$$q0^* = J \cdot \overline{q0} + \overline{K} \cdot q0$$

ใช้สมการบูลีนของ Next state และสมการบูลีนของเอาต์พุตที่ได้มาเป็นข้อมูลในการสร้างวงจรตามที่ออกแบบจะได้ดังรูปที่ 4.123 ซึ่งเป็น JK Flip-flop ที่เขียนอยู่ในรูปฟอร์มของ D Flip-flop นั้นเอง

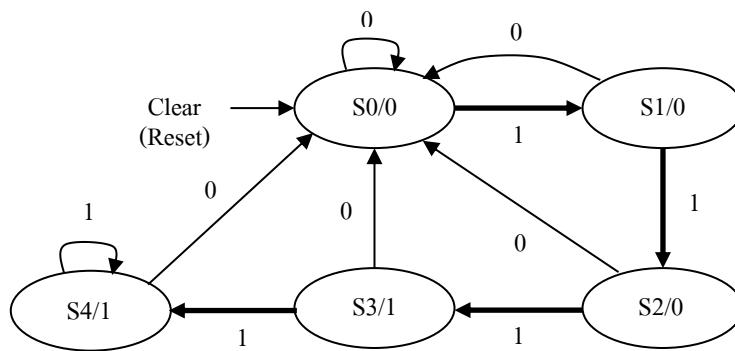


รูปที่ 4.123 ผังวงจร JK Flip-flop ที่เขียนอยู่ในรูปฟอร์มของ D Flip-flop

ตัวอย่างที่ 4.4 ออกแบบวงจรตรวจจับลำดับ (Sequence detector) เป็น Moor-type FSM มี X เป็นอินพุตและ Z เป็นเอาต์พุต โดยที่ วงจรนี้จะให้อาต์พุต Z = '1' ก็ต่อเมื่อตรวจจับคลอกิกอินพุต X = '1' จำนวน 4 ตัวติดต่อกัน

จากโจทย์เรามาระเบียน State diagram และ State table และรูปที่ 4.124 และรูปที่ 4.125 โดยมีหลักการคิดดังนี้

- 1) ถ้า S0 เป็น State เริ่มต้น เมื่อตรวจจับอินพุต X = '0' ได้ ก็ให้ State อยู่ที่ State S0 เช่นเดิม
- 2) ที่ State S0 เมื่อตรวจจับอินพุต X = '1' ตัวแรกได้แล้ว ให้วงจรเปลี่ยน State ไปที่ State S1
- 3) ที่ State S1 เมื่อตรวจจับอินพุต X = '1' ตัวที่ 2 ได้แล้ว ให้วงจรเปลี่ยน State ไปที่ State S2
- 4) ที่ State S2 เมื่อตรวจจับอินพุต X = '1' ตัวที่ 3 ได้แล้ว ให้วงจรเปลี่ยน State ไปที่ State S3
- 5) ที่ State S3 เมื่อตรวจจับอินพุต X = '1' ตัวที่ 4 ได้แล้ว ให้วงจรเปลี่ยน State ไปที่ State S4
- 6) จากนั้นให้เขียนทั้ง 5 State นี้ขึ้นมา ก่อนพร้อมเขียนส่วนที่มีลูกศรชี้ไปยัง State ตัดไป แล้วจึงเขียน State diagram และ State table ที่เหลือให้เป็นตามข้อกำหนดจนแล้วเสร็จ จะได้รูปที่ 4.124 และรูปที่ 4.125 ตามลำดับ



รูปที่ 4.124 State diagram ของวงจรตรวจขับลำดับ

Present state	Next state		Output Z
	X = 0	X = 1	
S0	S0	S1	0
S1	S0	S2	0
S2	S0	S3	0
S3	S0	S4	0
S4	S0	S4	1

รูปที่ 4.125 State table ของ Moor-type FSM ของวงจรตรวจขับลำดับ

ตัวอย่างการกำหนดค่า State (State assignment หรือ State encoding) แบบต่างๆ แสดงดังรูปที่ 4.126 ในตัวอย่างนี้เราจะยกตัวอย่างการเข้ารหัส State เป็น One-hot ดังตารางรูปที่ 4.127 โดยใช้จิสเกอร์เป็น D Flip-flop 5 ตัว จากนั้นแทนค่า State ในรูปที่ 4.127 ลงใน State table ในรูปที่ 4.125 แล้วจะได้ State-assigned table ดังรูปที่ 4.128 จากนั้นนำรูปที่ 4.128 มาเขียนใหม่ดังรูปที่ 4.129(a) และรูปที่ 4.129(b) โดยที่วงจร Next state นั้นจะไม่เขียนແລກที่ไม่ได้ใช้และให้เป็น Don't care จำนวน 54 ถ้า (จาก $26 = 64$ ถ้า) ส่วนวงจรเออเด็ฟตูละไม่เขียน 27 ถ้า (จาก $25 = 32$ ถ้า) จากนั้นจึงหาสมการบูลินของ Next state และของเออเด็ฟตูละ

No.	Binary	Gray	One-hot
0	000	000	00000001
1	001	001	00000010
2	010	011	00000100
3	011	010	00001000
4	100	110	00010000
5	101	111	00100000
6	110	101	01000000
7	111	100	10000000

รูปที่ 4.126 ตารางแสดงการเข้ารหัสแบบต่างๆ ที่ใช้ทำ State encoding

State	q4q3q2q1q0
S0	00001
S1	00010
S2	00100
S3	01000
S4	10000

รูปที่ 4.127 ตารางแสดงการเข้ารหัสแบบต่างๆ ที่ใช้ทำ State encoding

Present state	Next state		Output Z
	X = 0	X = 1	
00001	00001	00010	0
00010	00001	00100	0
00100	00001	01000	0
01000	00001	10000	0
10000	00001	10000	1

รูปที่ 4.128 State table ของ Moor-type FSM ของว งตรวจสอบลำดับ

Input & State	Next state
Xq4q3q2q1q0	q4*q3*q2*q1*q0*
0 0 0 0 0 1	0 0 0 0 1
0 0 0 0 1 0	0 0 0 0 1
0 0 0 1 0 0	0 0 0 0 1
0 0 1 0 0 0	0 0 0 0 1
0 1 0 0 0 0	0 0 0 0 1
1 0 0 0 0 1	0 0 0 1 0
1 0 0 0 1 0	0 0 1 0 0
1 0 0 1 0 0	0 1 0 0 0
1 0 1 0 0 0	1 0 0 0 0
1 1 0 0 0 0	1 0 0 0 0

(a) อินพุต-เอาต์พุตของวงจร Next state

State	Output
q4q3q2q1q0	Z
0 0 0 0 1	0
0 0 0 1 0	0
0 0 1 0 0	0
0 1 0 0 0	0
1 0 0 0 0	1

(b) อินพุต-เอาต์พุตของวงจรเอาต์พุต

รูปที่ 4.129 ตารางความจริงของวงจร Next state และวงจรเอาต์พุต

จากตารางความจริงของวงจร Next state ในรูปที่ 4.129(a) และวงจรเอาต์พุตในรูปที่ 4.129(b) นั้นสมการบูลีนจะลดรูปลงได้มาก เนื่องจากมี Don't care เป็นจำนวนมาก จะได้สมการบูลีนวงจร Next state และเอาต์พุตดังนี้

$$q4^* = X \cdot q4 + X \cdot q3$$

$$q3^* = X \cdot q2$$

$$q2^* = X \cdot q1$$

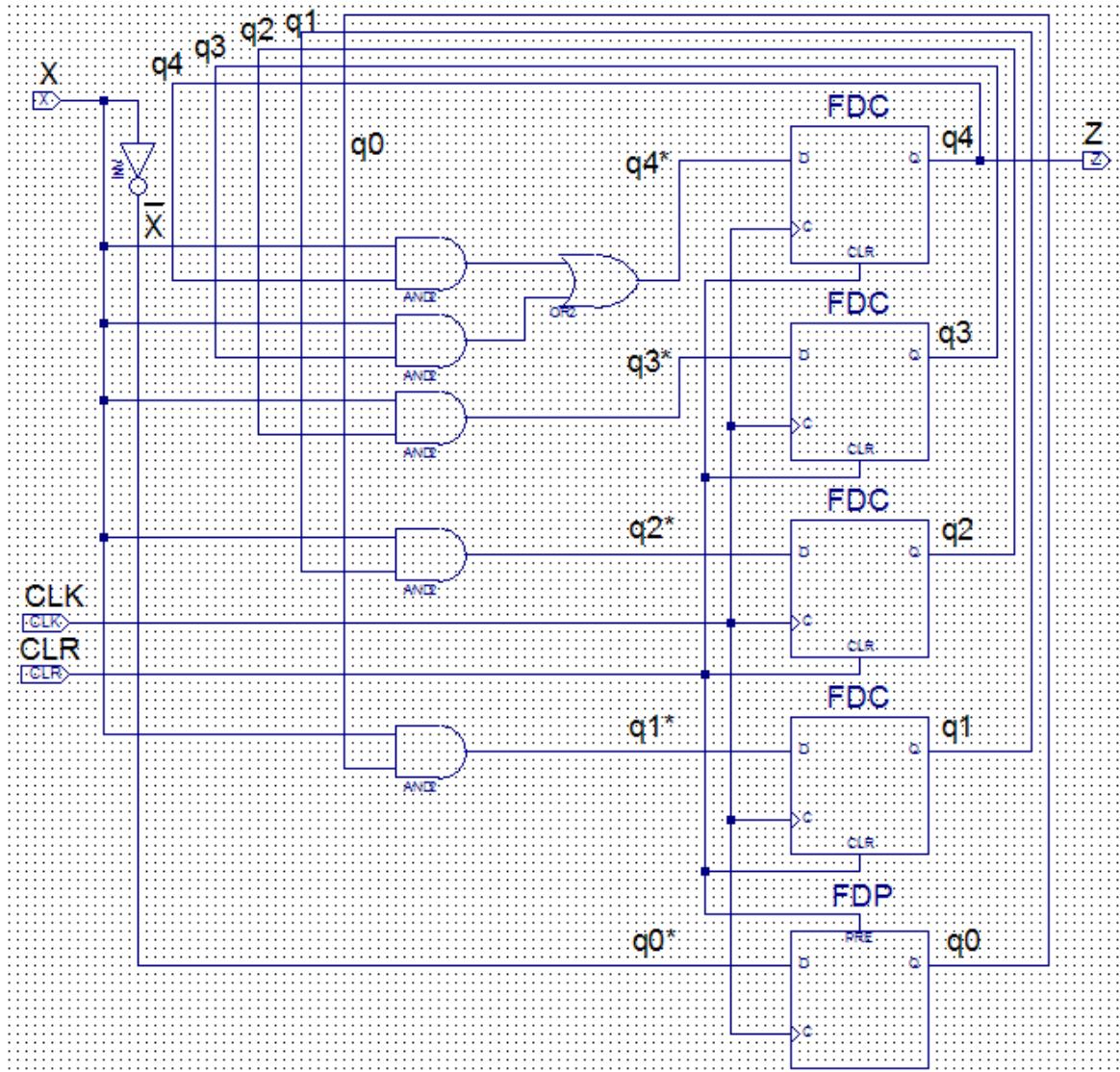
$$q1^* = X \cdot q0$$

$$q0^* = \overline{X} \cdot q4 + \overline{X} \cdot q3 + \overline{X} \cdot q2 + \overline{X} \cdot q3 + \overline{X} \cdot q0$$

$$= \overline{X} \cdot (q4 + q3 + q2 + q3 + q0) = \overline{X} \quad (q4 + q3 + q2 + q3 + q0 = 1 \text{ เพราะมีเพียงบิตเดียวเท่ากับ } 1)$$

$$Z = q4$$

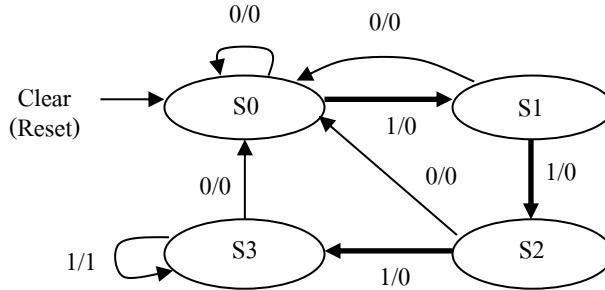
ใช้สมการบูลีนของ Next state และสมการบูลีนของเอาต์พุตที่ได้มาเป็นข้อมูลในการสร้างวงจรจะได้ดังรูปที่ 4.130



รูปที่ 4.130 วงจรตรวจจับลำดับแบบ Moor-type FSM

ในการถอดรหัสแบบตรวจจับลำดับ (Sequence detector) ในตัวอย่างที่ รูปที่ 4.4 เป็น Mealy-type FSM จากโจทย์ เราสามารถเขียน State diagram และ State table แสดงดังรูปที่ รูปที่ 4.131 และรูปที่ รูปที่ 4.132 โดยมีหลักการคิดดังนี้

- 1) ถ้า S_0 เป็น State เริ่มต้น เมื่อตรวจจับอินพุต $X = '0'$ ได้ ก็ให้ State อู๊ดที่ S_0 เช่นเดิม
- 2) เมื่อตรวจจับอินพุต $X = '1'$ ตัวแรกได้แล้ว ให้วางระเบียบ State ไปที่ State S_1
- 3) ที่ State S_1 เมื่อตรวจจับอินพุต $X = '1'$ ตัวที่ 2 ได้แล้ว ให้เปลี่ยน State ไปที่ State S_2
- 4) ที่ State S_2 เมื่อตรวจจับอินพุต $X = '1'$ ตัวที่ 3 ได้แล้ว ให้เปลี่ยน State ไปที่ State S_3
- 5) ที่ State S_3 เมื่อ $X = '1'$ แล้วเอาต์พุต $Z = '1'$ ทันที จากนั้นเขียน State diagram 4 State นี้ขึ้นมาก่อน แล้วจึงเขียน ส่วนที่เหลือให้เป็นตามข้อกำหนดงานแล้วเสร็จ แล้วจึงเขียน State table จะได้รูปที่ 4.131 และรูปที่ 4.132 ตามลำดับ



รูปที่ 4.131 State diagram ของวงจรตรวจขับลำดับ

Present state	Next state		Output Z	
	X = 0	X = 1	X = 0	X = 1
S0	S0	S1	S0	S1
S1	S0	S2	S0	S2
S2	S0	S3	S0	S3
S3	S0	S3	S0	S3

รูปที่ 4.132 State table ของ Moor-type FSM ของ จรวจขับลำดับ

ตัวอย่างนี้จะเข้ารหัส State เป็น One-hot ดังตารางรูปที่ 4.68 และใช้จิสเตอร์เป็น D Flip-flop แทนค่า State ในรูปที่ 4.133 ลงใน State table ในรูปที่ 4.132 จะได้ดังรูปที่ 4.134 ซึ่งเปียนใหม่ดังรูปที่ 4.135 โดยที่วงจร Next state และเอาต์พุตนั้นเรา จะไม่เปียนแล้วที่ไม่ได้ใช้และกำหนดเป็น Don't care 24 แล้ว (จาก $2^5 = 32$ แล้ว) จากนั้นทำการบูตเลนววงจร Next state และเอาต์พุต

State	$q_3q_2q_1q_0$
S0	0001
S1	0010
S2	0100
S3	1000

รูปที่ 4.133 ตารางแสดงการเข้ารหัสแบบต่างๆ ที่ใช้ทำ State encoding

Present state = $q_3q_2q_1q_0$	Next state = $q_3^*q_2^*q_1^*q_0^*$		Output Z	
	X = 0	X = 1	X = 0	X = 1
0001	0001	0001	0	0
0010	0001	0010	0	0
0100	0001	0100	0	0
1000	0001	1000	0	1

รูปที่ 4.134 State table ของ Moor-type FSM ของ จรวจขับลำดับ

Input & State	Next state
$Xq_3q_2q_1q_0$	$q_3^*q_2^*q_1^*q_0^*$
0 0 0 0 1	0 0 0 1
0 0 0 1 0	0 0 0 1
0 0 1 0 0	0 0 0 1
0 1 0 0 0	0 0 0 1
1 0 0 0 1	0 0 1 0
1 0 0 1 0	0 1 0 0
1 0 1 0 0	1 0 0 0
1 1 0 0 0	1 0 0 0

(a) อินพุต-เอาต์พุตของวงจร Next state

Input & State	Output
$Xq_3q_2q_1q_0$	Z
0 0 0 0 1	0
0 0 0 1 0	0
0 0 1 0 0	0
0 1 0 0 0	0
1 0 0 0 1	0
1 0 0 1 0	0
1 0 1 0 0	0
1 1 0 0 0	1

(b) อินพุต-เอาต์พุตของวงจรเอาต์พุต

รูปที่ 4.135 ตารางความจริงของวงจร Next state และวงจรเอาต์พุต

จากตารางความจริงของวงจร Next state ในรูปที่ 4.135(a) และวงจรเอาต์พุตในรูปที่ 4.135(b) นั้นสมการบูลีนจะลดรูปลงได้มาก เนื่องจากมี Don't care เป็นจำนวนมาก จะได้สมการบูลีนวงจร Next state ดังนี้

$$q_3^* = X \cdot q_3 + X \cdot q_1$$

$$q_2^* = X \cdot q_1$$

$$q_1^* = X \cdot q_0$$

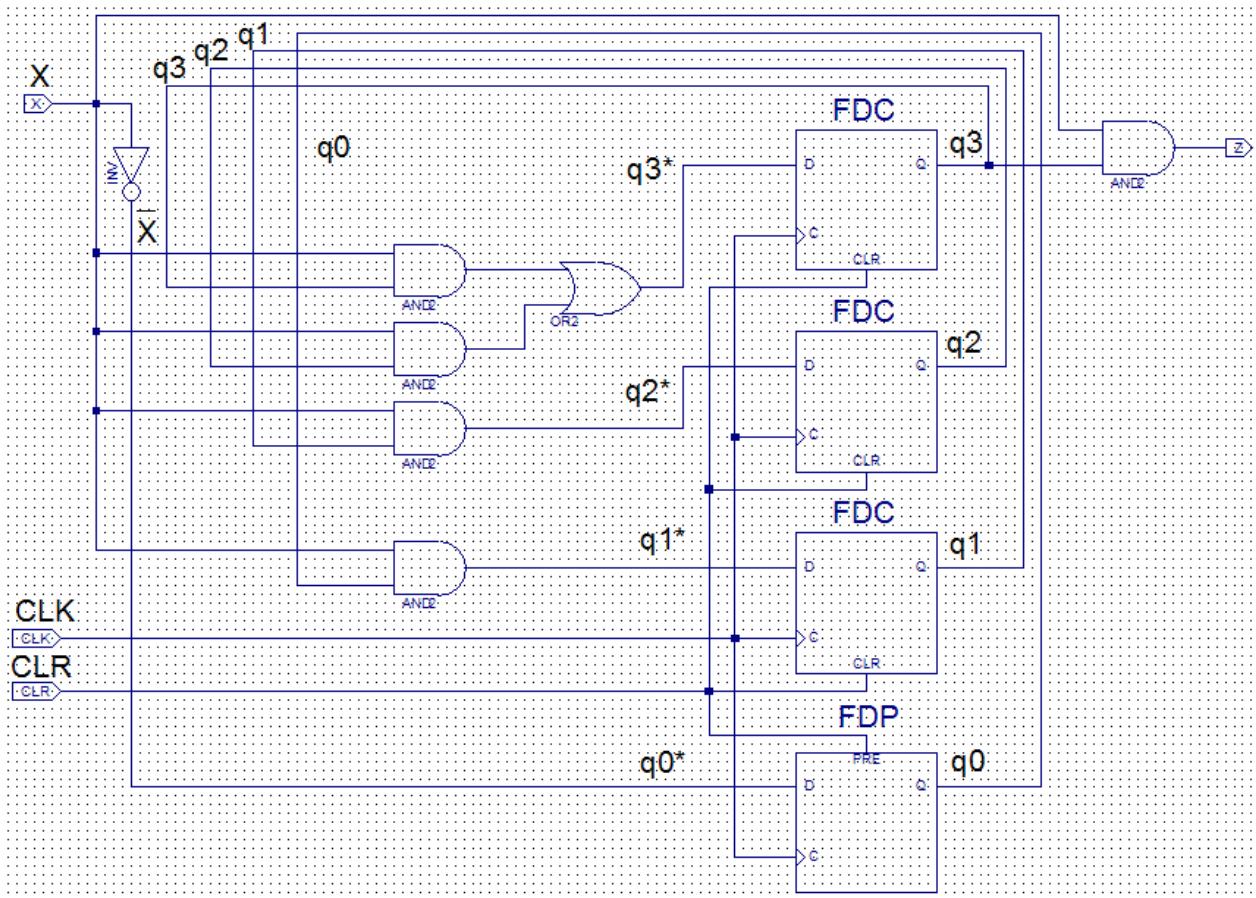
$$q_0^* = \overline{X} \cdot q_3 + \overline{X} \cdot q_2 + \overline{X} \cdot q_1 + \overline{X} \cdot q_0$$

$$= \overline{X} \cdot (q_3 + q_2 + q_3 + q_0) = \overline{X} \quad (q_3 + q_2 + q_3 + q_0 = 1 \text{ เพราะมีเพียงบิตเดียวเท่านั้นที่เป็นลอจิก '1'})$$

และจะได้สมการบูลีนวงจรเอาต์พุตคือ

$$Z = X \cdot q_3$$

จากนั้นจึงใช้สมการบูลีนของ Next state และสมการบูลีนของเอาต์พุตที่ได้นำไปเป็นสร้างวงจรตรวจจับลำดับแบบ Mealy-type FSM ดังรูปที่ 4.136



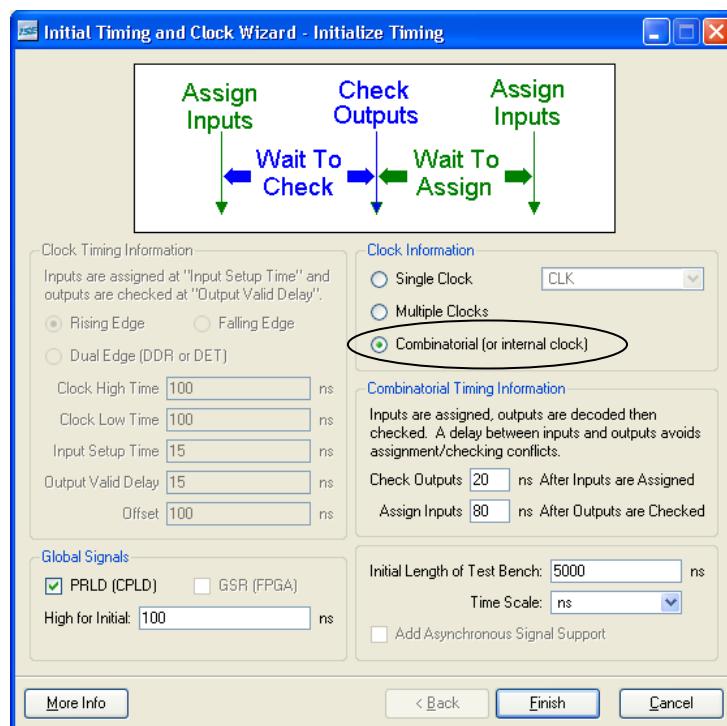
รูปที่ 4.136 วงจรตรวจจับลำดับแบบ Mealy-type FSM

4.6.1 การออกแบบวงจรตรวจจับลำดับแบบ Moor-type FSM

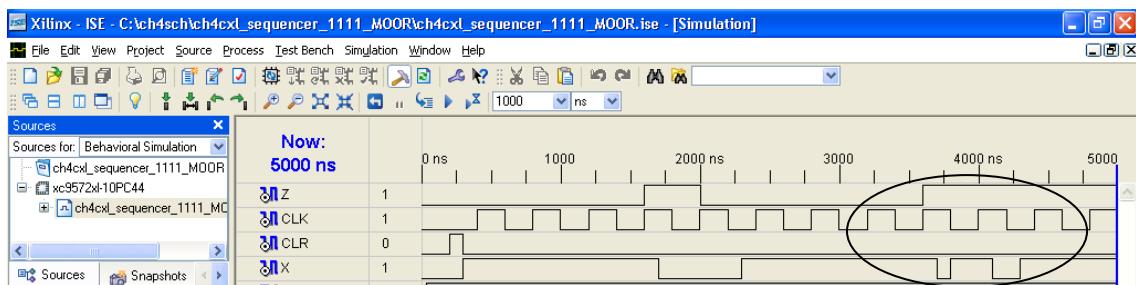
อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรตรวจจับลำดับด้วย CPLD

a) วิเคราะห์วงจรตรวจจับลำดับแบบ Moor-type FSM ในรูปที่ 4.130 วงจรนี้จะให้อาต์พุต $Z = '1'$ ก็ต่อเมื่อตรวจสอบอินพุต $X = '1'$ จำนวน 4 ตัวติดต่อกัน โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4cxl_sequencer_1111_MOOR จากนั้นทำ Simulation โดยใช้ไฟล์ชื่อ ch4cxl_Sequencer_1111_MOOR_tb1 และกำหนดค่าต่างๆ ดังรูปที่ 4.137 เสร็จแล้วพิจารณาผล Simulation ดังรูปที่ 4.138 ว่าเป็นไปตามทฤษฎีหรือไม่

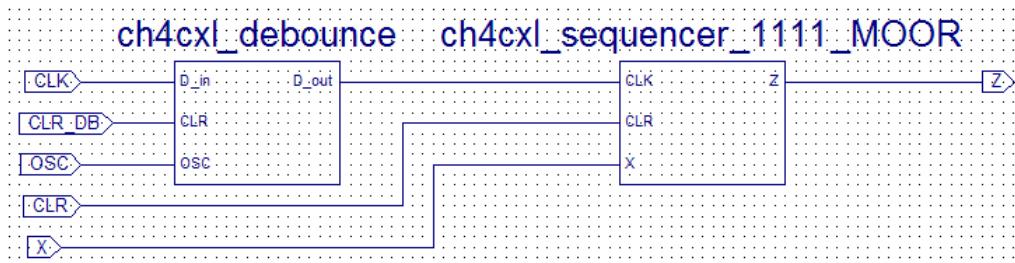


รูปที่ 4.137 การกำหนดค่าต่างๆ ที่ใช้ Simulation



รูปที่ 4.138 การกำหนดค่าต่างๆ ที่ใช้ Simulation (บริเวณวงกลมมี $X = '0'$ ในช่วงเวลาแคบๆ)

b) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_6_1cxl จากนั้นนำไฟล์วงจรโมโนสเตเบิลชื่อ ch4cxl_debounce และไฟล์ชื่อ ch4cxl_sequencer_1111_MOOR มาทำ Symbols และวิเคราะห์วงจรดังรูปที่ 4.139



รูปที่ 4.139 ผังวงจรทดสอบของวงจรตรวจจับลำดับแบบ Moor-type FSM

การกำหนดขาสัญญาณ ใช้ OSC = 32.768kHz, PB1-PB1, Slide SW1 และ Slide SW2 เป็นอินพุต LED4 เป็นเอาต์พุต

$$\begin{aligned} \text{CLK} &= \text{PB1} = \text{INPUT} = \text{p39} & \text{X} &= \text{PB3}(\text{Slide SW1}) = \text{INPUT} = \text{p42} & \text{Z} &= \text{LED4} = \text{OUTPUT} = \text{p35} \\ \text{CLR_DB} &= \text{PB2} = \text{INPUT} = \text{p40} & \text{CLR} &= \text{PB4}(\text{Slide SW2}) = \text{INPUT} = \text{p43} & \text{OSC} &= \text{OSC} = \text{INPUT} = \text{p5} \end{aligned}$$

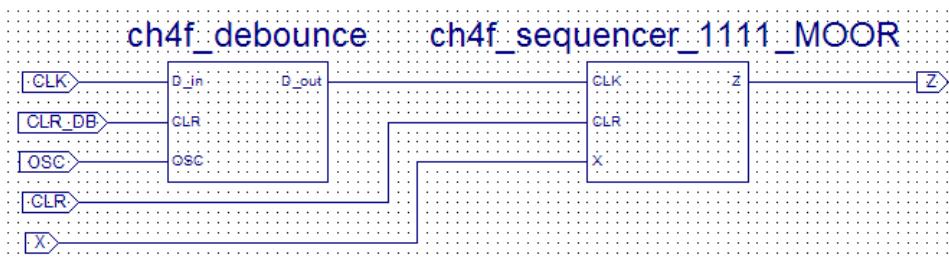
โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "CLK" LOC = "p39" ;
NET "CLR" LOC = "p43" ;
NET "CLR_DB" LOC = "p40" ;
NET "OSC" LOC = "p5" ;
NET "X" LOC = "p42" ;
NET "Z" LOC = "p35" | SLEW = SLOW ;
```

หลังจากโปรแกรม CPLD แล้วให้ OFF Slide SW1 (X = '1') และ ON Slide SW2 (CLR = '0') แล้วกดปุ่ม PB1 สลับกับ PB2 ไปเรื่อยๆ จะกระตุ้น LED4 ติดสว่างแล้วกดปุ่ม PB2 จากนั้น ON Slide SW1 (X = '0') กดปุ่ม PB1 แล้วกดปุ่ม PB2 จากนั้น OFF Slide SW1 (X = '1') แล้วกดปุ่ม PB1 สลับกับ PB2 ไปเรื่อยๆ จะกระตุ้น LED4 ติดสว่างอีกครั้ง แล้ว OFF Slide SW2 (CLR = '1') แล้ว ON Slide SW2 (CLR = '0') อีกครั้ง แล้วทำการทดลองซ้ำ จนกว่าจะได้สูตรผลและบันทึกผลการทดลอง

2. สร้างวงจรตรวจจับลำดับด้วย FPGA

- a) วาดผังวงจรตรวจจับลำดับแบบ Moor-type FSM ในรูปที่ 4.130 วงจรนี้จะให้อาต์พุต Z = '1' ก็ต่อเมื่อตรวจจับอินพุต X = '1' จำนวน 4 ตัวติดต่อกัน โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4cf_sequencer_1111_MOOR จากนั้นทำ Simulation โดยใช้ไฟล์ชื่อ ch4cf_Sequencer_1111_MOOR_tb1 และกำหนดค่าต่างๆ และป้อนรูปคลื่นอินพุตทำงานองค์ประกอบในรูปที่ 4.137 และรูปที่ 4.138 แล้วพิจารณาผล Simulation ว่าเป็นตามทฤษฎีหรือไม่
- b) สร้างไฟล์ใน Project Location ชื่อ ch4sch และกำหนด Project Name และ Source File ชื่อ ex4_6_1f จากนั้นนำไฟล์วางลงในสเปคบิลชื่อ ch4cf_debounce และไฟล์ชื่อ ch4cf_sequencer_1111_MOOR มาทำ Symbols แล้ววาดผังวงจรดังรูปที่ 4.140



รูปที่ 4.140 ผังวงจรทดสอบของวงจรตรวจจับลำดับแบบ Moor-type FSM

การกำหนดขาสัญญาณ ใช้ OSC = 25MHz, PB1-PB1, Dip SW1 และ Dip SW2 เป็นอินพุต LED L0 เป็นเอาต์พุต

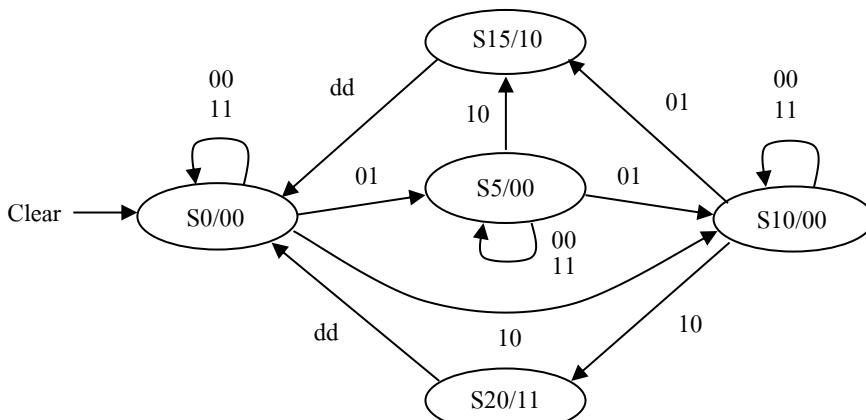
$$\begin{aligned} \text{CLK} &= \text{PB1} = \text{INPUT} = \text{p44} & \text{X} &= \text{Dip SW1} = \text{INPUT} = \text{p52} & \text{Z} &= \text{LED0} = \text{OUTPUT} = \text{p35} \\ \text{CLR_DB} &= \text{PB2} = \text{INPUT} = \text{p46} & \text{CLR} &= \text{Dip SW2} = \text{INPUT} = \text{p53} & \text{OSC} &= \text{OSC} = \text{INPUT} = \text{p127} \end{aligned}$$

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```
NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR" LOC = "p53" | IOSTANDARD = LVCMOS33 ;
NET "CLR_DB" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;
NET "X" LOC = "p52" | IOSTANDARD = LVCMOS33 ;
NET "Z" LOC = "p70" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
```

หลังจากโปรแกรม FPGA แล้วให้ OFF Dip SW1 ($X = 1$) และ ON Dip SW2 ($CLR = 0$) แล้วกดปุ่ม PB1 สลับกับ PB2 ไปเรื่อยๆ จนกระทั่ง LED L0 ติดสว่างแล้วกดปุ่ม PB2 จากนั้น ON Dip SW1 ($X = 0$) กดปุ่ม PB1 และกดปุ่ม PB2 จากนั้น OFF Dip SW1 ($X = 1$) แล้วกดปุ่ม PB1 สลับกับ PB2 ไปเรื่อยๆ จนกระทั่ง LED L0 ติดสว่างอีกรั้ง แล้ว OFF Dip SW2 ($CLR = 1$) แล้ว ON Dip SW2 ($CLR = 0$) อีกรั้ง แล้วทำการทดลองซ้ำ จากนั้นให้สรุปผลและบันทึกผลการทดลอง

วงจรหน่วยควบคุม (Control unit) เครื่องขายน้ำอัดลมอัตโนมัติ (Vending machine) ซึ่งรับสัญญาณอินพุตจากการรับเหรียญ 5 บาทและ 10 บาทที่จะต้องหยอดเหรียญครั้งละเหรียญเท่านั้น เมื่อหยอดเหรียญครบ 15 บาทเครื่องจะปล่อยน้ำอัดลมออกมา 1 กระป๋อง ถ้าใส่เหรียญเกินรวมเป็น 20 บาทเครื่องจะปล่อยน้ำอัดลมออกมา 1 กระป๋องพร้อมเงินทอน 5 บาท ซึ่งเราสามารถเขียน State diagram และ State table ได้ดังรูปที่ 4.141 และรูปที่ 4.142 ตามลำดับ กำหนดให้ X และ Z เป็นอินพุตและเอาต์พุต แบบอะเรย์ โดยที่ $X(1)$ และ $X(0)$ เป็นอินพุตที่ใช้รับเหรียญ 10 บาทและ 5 บาทตามลำดับ ในขณะที่ $Z(1)$ และ $Z(0)$ จะเป็นเอาต์พุตของวงจรปล่อยน้ำอัดลมและวงจรทอนเงิน 5 บาทตามลำดับ โดยที่ d คือ Don't care



รูปที่ 4.141 State diagram (Moor-type FSM) วงจรหน่วยควบคุมของเครื่องขายน้ำอัดลมอัตโนมัติ (อย่างง่าย)

Present state	Next state				Output Z
	$X = 00$	$X = 01$	$X = 10$	$X = 11$	
S0	S0	S5	S10	S0	00
S5	S5	S10	S15	S5	00
S10	S10	S15	S20	S10	00
S15	S0	S0	S0	S0	10
S20	S0	S0	S0	S0	11

รูปที่ 4.142 State table วงจรหน่วยควบคุมของเครื่องขายน้ำอัดลมอัตโนมัติ (อย่างง่าย)

4.6.2 การออกแบบตรวจสอบจับลำดับแบบ Mealy-type FSM

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1. สร้างวงจรตรวจจับลำดับด้วย CPLD

ทำการทดลอง CPLD เมื่อcionกับการทดลองที่ 4.6.1 ทุกประการ แต่ใช้วงจรตรวจจับลำดับแบบ Mealy-type FSM ในรูปที่ 4.136 วงจนี้จะให้อาตพุต $Z = '1'$ ก็ต่อเมื่อตรวจจับอินพุต $X = '1'$ จำนวน 4 ตัวติดต่อกัน โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4cxl_sequencer_1111_MEALY จากนั้นทำ Simulation โดยใช้ไฟล์ชื่อ ch4cxl_Sequencer_1111_MEALY_tb1 เสร็จแล้วพิจารณาผล Simulation ว่าเป็นไปตามทฤษฎีหรือไม่ และสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ex4_6_2cx1 จากนั้นนำไฟล์วงจรโน้นสเกนเบิลชื่อ ch4cxl_debounce และไฟล์ชื่อ ch4cxl_sequencer_1111_MEALY มาทำ Symbols แล้วดูว่าดังวงจรตรวจจับลำดับ

การกำหนดขาสัญญาณต่างๆ จะเหมือนกับการทดลอง CPLD ในการทดลองที่ 4.6.1 ทุกประการ หลังจากโปรแกรม CPLD แล้วให้ OFF Slide SW1 ($X = '1'$) และ ON Slide SW2 ($CLR = '0'$) แล้วกดปุ่ม PB1 สลับกับ PB2 ไปเรื่อยๆ จนกระทั่ง LED4 ติดสว่างแล้วกดปุ่ม PB2 จากนั้น ON Slide SW1 ($X = '0'$) กดปุ่ม PB1 แล้วกดปุ่ม PB2 จากนั้น OFF Slide SW1 ($X = '1'$) แล้วกดปุ่ม PB1 สลับกับ PB2 ไปเรื่อยๆ จนกระทั่ง LED4 ติดสว่างอีกครั้ง จากนั้นทดลอง ON-OFF Slide SW1 สลับกันไปเรื่อยๆ และสังเกตดูผล LED4 ไปพร้อมๆ กัน และ OFF Slide SW2 ($CLR = '1'$) แล้ว ON Slide SW2 ($CLR = '0'$) อีกครั้งแล้วทำการทดลองซ้ำ จากนั้นให้สรุปผลและบันทึกผลการทดลอง

2. สร้างวงจรตรวจจับลำดับด้วย FPGA

ทำการทดลอง FPGA เมื่อcionกับการทดลองที่ 4.6.1 ทุกประการ แต่ใช้วงจรตรวจจับลำดับแบบ Mealy-type FSM ในรูปที่ 4.136 วงจนี้จะให้อาตพุต $Z = '1'$ ก็ต่อเมื่อตรวจจับอินพุต $X = '1'$ จำนวน 4 ตัวติดต่อกัน โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ch4f_sequencer_1111_MEALY จากนั้นทำ Simulation โดยใช้ไฟล์ชื่อ ch4f_Sequencer_1111_MEALY_tb1 เสร็จแล้วพิจารณาผล Simulation ว่าเป็นไปตามทฤษฎีหรือไม่ และสร้างไฟล์ใน Project Location ชื่อ ch4sch กำหนด Project Name และ Source File ชื่อ ex4_6_2f จากนั้นนำไฟล์วงจรโน้นสเกนเบิลชื่อ ch4f_debounce และไฟล์ชื่อ ch4f_sequencer_1111_MEALY มาทำ Symbols แล้วดูว่าดังวงจรตรวจจับลำดับ

การกำหนดขาสัญญาณต่างๆ จะเหมือนกับการทดลอง FPGA ในการทดลองที่ 4.6.1 ทุกประการ หลังจากโปรแกรม FPGA แล้วให้ OFF Dip SW1 ($X = '1'$) และ ON Dip SW2 ($CLR = '0'$) แล้วกดปุ่ม PB1 สลับกับ PB2 ไปเรื่อยๆ จนกระทั่ง LED L0 ติดสว่างแล้วกดปุ่ม PB2 จากนั้น ON Dip SW1 ($X = '0'$) กดปุ่ม PB1 แล้วกดปุ่ม PB2 จากนั้น OFF Dip SW1 ($X = '1'$) แล้วกดปุ่ม PB1 สลับกับ PB2 ไปเรื่อยๆ จนกระทั่ง LED L0 ติดสว่างอีกครั้ง จากนั้นทดลอง ON-OFF Dip SW1 สลับกันไปเรื่อยๆ และสังเกตดูผล LED L0 ไปพร้อมๆ กัน และ OFF Dip SW2 ($CLR = '1'$) แล้ว ON Dip SW2 ($CLR = '0'$) อีกครั้งแล้วทำการทดลองซ้ำ จากนั้นให้สรุปผลและบันทึกผลการทดลอง

4.7 วิธี Schematic โดยใช้ Symbols ที่เป็นไฟล์ภาษา VHDL

เนื่องจากไฟล์วงจรดิจิตอลที่เขียนด้วยภาษาระดับสูงนั้นสามารถดาวน์โหลดได้ทางอินเตอร์เน็ต หรือดูได้จากหนังสือทั่วไป ซึ่งอาจจะเขียนด้วยภาษาระดับสูง เช่น VHDL หรือ Verilog เราสามารถนำไฟล์เหล่านั้นมาใช้ได้โดยการสร้าง Symbols ซึ่งในประเทศไทยนั้นส่วนใหญ่จะออกแบบด้วยภาษา VHDL เราจะอธิบายเฉพาะการนำไฟล์ VHDL ไปสร้าง Symbols เท่านั้น

VHDL [6-12] ย่อมาจาก VHSIC Hardware Description Language (VHSIC = Very High Speed Integrated Circuit) VHDL เป็นภาษาระดับสูงที่ใช้ในการออกแบบระบบดิจิตอล มีรูปแบบคล้ายกับภาษา Pascal ถูกปรับเปลี่ยนโดยกระทรวงกลาโหมของสหรัฐอเมริกา ต่อมา IEEE ได้รับภารานี้เข้ามาปรับปรุงเป็นมาตรฐาน IEEE Std 1076-1987 (VHDL87) และได้ปรับปรุงเป็น IEEE Std 1076-1993 (VHDL93) ตามลำดับ นอกจากนี้แล้ว IEEE ยังได้นิยามชนิดข้อมูล std_logic ไว้ใน มาตรฐาน IEEE Std 1164-1993 เพื่อครอบคลุมลักษณะการทำงาน เช่น High impedance และสถานะอื่นๆ ที่ใช้สำรองการทำงาน

4.7.1 VHDL พื้นฐาน

โดยทั่วไปว่าจะที่ออกแบบด้วยภาษา VHDL จะประกอบด้วยออกแบบพื้นฐานอย่างน้อย 2 หน่วย คือ ประกาศใช้ เอนติตี้ (Entity declaration) หรือเอนติตี้ (Entity) และ อะชิเทกเชอร์บอดี้ (Architecture body) หรือ อะชิเทกเชอร์ (Architecture)

- Entity declaration คือ ส่วนที่เป็น Interface กล่าวคือ เป็นหน่วยออกแบบที่บอกรายละเอียดว่าอินพุต/เอาต์พุตพอร์ต (I/O Port) ของระบบดิจิตอลที่ออกแบบนั้นมีการเชื่อมต่อ (Interface) กับวงจรภายในอย่างไรบ้าง
- Architecture หรือ Architecture body คือ ส่วนที่เป็นหน่วยออกแบบที่อธิบายความสัมพันธ์ระหว่างอินพุตและเอาต์พุต (บรรยายพฤติกรรมการทำงาน) ของระบบดิจิตอลที่ออกแบบ

ตัวอย่างที่ 4.5 วงจรคอร์หัสเข้า 2 ออก 4 (2 to 4 Decoder) มีตารางความจริงดังรูปที่ 4.143(a) มีสมการบูลีนดังรูปที่ 4.143(b) และผังวงจรแสดงดังรูปที่ 4.143(c) เราจะนำ I/O ที่อยู่ภายนอกกรอบสี่เหลี่ยมในรูปที่ 4.143(c) ไปเขียนในส่วนประกาศใช้เอนติตี้ (ที่อยู่ในบรรทัดที่ 2-9 ในรูปที่ 4.144) และนำวงจรภายในกรอบสี่เหลี่ยมในรูปที่ 4.144(c) ไปเขียนในส่วนอะชิเทกเชอร์บอดี้ (ที่อยู่ในบรรทัดที่ 11-17 ในรูปที่ 4.144) โดยที่เขียนโดยใช้รูปแบบ VHDL87 และ VHDL93 และรูปที่ 4.144(a) และรูปที่ 4.144(b) ตามลำดับ

Input		Output			
A1	A0	D3	D2	D1	D0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

(a) ตารางความจริงของวงจร 2 to 4 Decoder

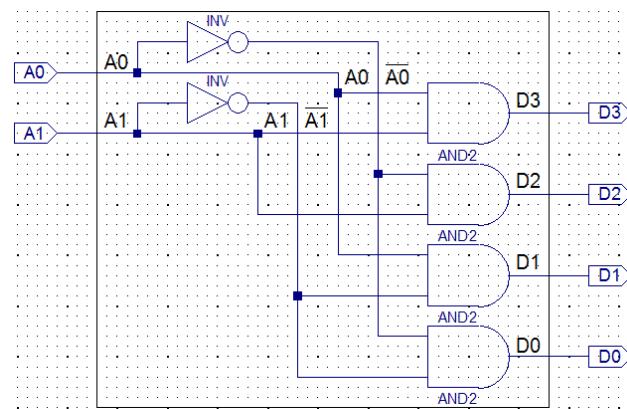
$$D0 = \overline{A1} \cdot \overline{A0}$$

$$D1 = \overline{A1} \cdot A0$$

$$D2 = A1 \cdot \overline{A0}$$

$$D3 = A1 \cdot A0$$

(b) สมการบูลิ่นของวงจร 2 to 4 Decoder



(c) ผังวงจร 2 to 4 Decoder

รูปที่ 4.143 รายละเอียดค่างๆ ของวงจร 2 to 4 Decoder ที่จะนำไปเขียนด้วยภาษา VHDL

```

2 entity DECODER2TO4 is
3   port ( A0 : in bit;
4         A1 : in bit;
5         D0 : out bit;
6         D1 : out bit;
7         D2 : out bit;
8         D3 : out bit);
9 end DECODER2TO4;
10 architecture BEHAVIORAL of DECODER2TO4 is
11 begin
12   DO <= (not A1) and (not A0); บรรทัดที่ 11-17 เป็นอาชีтекขอร์บันดี้ ซึ่งส่วนนี้จะบอกว่า จริงๆ แล้ว
13   D1 <= (not A1) and A0;        เอาต์พุตจะ “ไร้บ้าง” อินพุตคือ A0,A1 และเอาต์พุตคือ D1,D2,D3,D4
14   D2 <= A1 and (not A0);      บรรทัดที่ 2-9 เป็นประการใช้ออนดิตี้ ซึ่งส่วนนี้จะบอกว่า จริงๆ แล้ว
15   D3 <= A1 and A0;          เอาต์พุตจะ “ไร้บ้าง” อินพุตคือ A0,A1 และเอาต์พุตคือ D1,D2,D3,D4
16
17 end BEHAVIORAL;

```

(a) โค้ด VHDL ที่เขียนโดยใช้รูปแบบ VHDL87

```

2 entity DECODER2TO4 is
3   port ( A0 : in bit;
4         A1 : in bit;
5         D0 : out bit;
6         D1 : out bit;
7         D2 : out bit;
8         D3 : out bit);
9 end entity DECODER2TO4; บรรทัดที่ 9 อาจจะใช้ end entity แทนคำว่า end
10
11 architecture BEHAVIORAL of DECODER2TO4 is
12 begin
13   DO <= (not A1) and (not A0); บรรทัดที่ 17 อาจจะใช้ end architecture แทนคำว่า end
14   D1 <= (not A1) and A0;
15   D2 <= A1 and (not A0);
16   D3 <= A1 and A0;
17 end architecture BEHAVIORAL;

```

(b) โค้ด VHDL ที่เขียนโดยใช้รูปแบบใน VHDL93

รูปที่ 4.144 โค้ด VHDL ของวงจร 2 to 4 Decoder (เลขบรรทัดมีไว้เพื่ออำนวยความสะดวก ไม่ใช่ส่วนของโค้ด)

จากรูปที่ 4.144 วงจร 2 to 4 Decoder มีอ่อนดิตี้ ชื่อ DECODER2TO4 และอาชีтекขอร์ ชื่อ BEHAVIORAL มี A0 และ A1 เป็นอินพุต (A0 และ A1 มี Mode เป็น Input หรือ in) มี D0 ถึง D3 เป็นเอาต์พุต (D0 ถึง D3 มี Mode เป็น Output หรือ out)

โดยที่ คำสั่ง (Statement)	D0 <= (not A1) and (not A0);	มีความหมายว่า	$D0 = \overline{A1} \cdot \overline{A0}$
	D1 <= (not A1) and A0;	มีความหมายว่า	$D1 = \overline{A1} \cdot A0$
	D2 <= A1 and (not A0);	มีความหมายว่า	$D2 = A1 \cdot \overline{A0}$
	D3 <= A1 and A0;	มีความหมายว่า	$D3 = A1 \cdot A0$

เราเรียก not และ and ว่าเป็นตัวดำเนินการ หรือ Operator จากรูปที่ 4.144a) และรูปที่ 4.144b) นั้น A0, A1 และ D0-D3 มีสถานะเป็นล็อกิก ‘1’ หรือ ‘0’ เท่านั้น เราจึงใช้ชนิดข้อมูล (Data type หรือ Type) ของอินพุตและเอาต์พุตเป็น “ชนิดข้อมูล bit”

คำสั่งที่อยู่ในอาชีтекขอร์บรรทัดที่ 13-16 จะทำงานพร้อมกัน (Signal concurrency) โดยไม่สนใจลำดับก่อนและหลัง ซึ่งเราเรียกว่า คำสั่งคอนแครร์เรนต์ (Concurrent statement) การเขียนคำสั่งเหล่านี้กลับที่กันดังรูปที่ 4.145 จะให้ผลลัพธ์เหมือนกัน

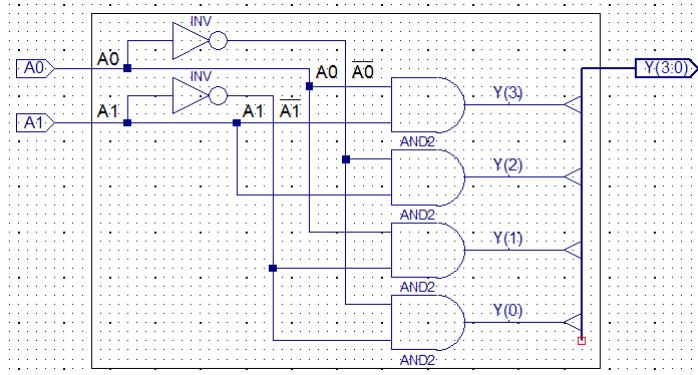
```

11 architecture BEHAVIORAL of DECODER2TO4 is
12 begin
13     D3 <= A1 and A0;
14     D2 <= A1 and (not A0);
15     D1 <= (not A1) and A0;
16     D0 <= (not A1) and (not A0);
17 end architecture BEHAVIORAL;

```

รูปที่ 4.145 ตัวอย่าง Statement ใน VHDL ที่วางแผนลับบนรหัสกัน

เราสามารถเขียนเอาต์พุตของวงจร 2 to 4 Decoder ให้อยู่ในรูปของบัสได้ เช่นกัน แสดงดังรูปที่ 4.146(a) และรูปที่ 4.146(b)



(a) ผังวงจร 2 to 4 Decoder ที่มีเอาต์พุตเป็นบัสขนาด 4 บิต

```

2 entity DECODER2TO4 is
3     port ( A0, A1 : in bit;
4             Y : out bit_vector(3 downto 0));
5 end DECODER2TO4;
6
7 architecture BEHAVIORAL of DECODER2TO4 is
8 begin
9     Y(0) <= (not A1) and (not A0);
10    Y(1) <= (not A1) and A0;
11    Y(2) <= A1 and (not A0);
12    Y(3) <= A1 and A0;
13 end BEHAVIORAL;

```

การเปลี่ยนอินพุตหรือเอาต์พุตหลายตัวไว้ด้วยกันจะต้องมี “,” กันระหว่างอินพุตหรือเอาต์พุตทุกด้วย

(b) โค้ด VHDL ของวงจร 2 to 4 Decoder ที่มีเอาต์พุตเป็นบัสขนาด 4 บิต

รูปที่ 4.146 ผังวงจรและโค้ด VHDL ของวงจร 2 to 4 Decoder ที่มีเอาต์พุตเป็นบัสขนาด 4 บิต

จากรูปที่ 4.146(a) เอาต์พุตของวงจร 2 to 4 Decoder เป็นบัส (Bus) Y ขนาด 4 บิต โดยมีสมาชิกแต่ละบิตของบัส Y ประกอบด้วย Y(3), Y(2), Y(1) และ Y(0) (เป็นค่าตัวบัญชี Y3, Y2, Y1 และ Y0) โค้ด VHDL ในบรรทัดที่ 4 รูปที่ 4.146(b) จะใช้ชนิดข้อมูล bit_vector ซึ่งเป็นชนิดข้อมูลแบบอะเรย์ (Array) ของชนิดข้อมูล bit โดยจะเขียนเป็น bit_vector (3 downto 0) แทนความหมายของบัส Y ที่มีขนาด 4 บิต ซึ่งการเขียนในลักษณะนี้ Y(3) จะเป็นบิต MSB และ Y(0) จะเป็นบิต LSB

4.7.2 การตั้งชื่อ

การตั้งชื่อในภาษา VHDL นั้นอาจจะประกอบด้วย พยัญชนะ ตัวเลข และเครื่องหมายขีดเด้นใต้ (Underscore) โดยที่

- ต้องเขียนต้นคำด้วยพยัญชนะ
- ต้องเขียนติดกันโดยไม่มีว่างช่องว่าง (White space)
- ห้ามใช้เครื่องหมายขีดเด้นใต้ติดกันหลายตัวและห้ามจบชื่อด้วยเครื่องหมายขีดเด้นใต้
- ห้ามใช้คำส่วน (Reserved words) ซอฟต์แวร์ทุก ISE WebPACK จะแสดงคำส่วนค้ำยตัวอักษรที่เป็นสีต่างๆ

ตัวอย่างการตั้งชื่อ ARCHI, aRChi , archi	ถูก เพราะเป็นชื่อเดียวกัน
HALF_ADDER	ถูก
HALF__ADDER	ผิด เพราะ underscore ติดกัน 2 ตัว
HALF ADDER_	ผิด เพราะเว้นช่องว่างและจบชื่อด้วยเครื่องหมายขีดเส้นใต้
74LS00	ผิด เพราะชื่นต้นด้วยตัวเลข
open	ผิด เพราะใช้คำสlang

4.7.3 ประเภทของพอร์ต

ประเภทของพอร์ต (Port mode หรือ Mode) ได้แก่ in กือ อินพุต (Input) out กือ เอาต์พุต (Output) inout กือ อินพุต/เอาต์พุต (I/O) buffer กือ เป็นเอาต์พุตพอร์ตที่สามารถอ่านค่ากลับเข้ามาภายในวงจรที่ออกแบบได้ ดูตัวอย่างที่ 4.11

4.7.4 ชนิดข้อมูล

เนื่องจาก“ตัวดำเนินการ (Operator) ชนิดหนึ่งจะใช้ได้กับชนิดข้อมูล (Type หรือ Data type) บางชนิดเท่านั้น” สิ่งนี้กือ ข้อจำกัดที่ผู้ออกแบบต้องทราบและป้องครรังที่ผู้เริ่มต้นมักจะเจอบัญหา นี่ ชนิดข้อมูลที่ควรทราบมีดังนี้

- ชนิดข้อมูล bit มีค่าเป็นโลจิก '0' และ '1'
- ชนิดข้อมูล integer มีค่าอย่างน้อยอยู่ในช่วง $-(2^{31}-1)$ หรือ -2147483647 ถึง $(2^{31}-1)$ หรือ 2147483647 ซึ่งโดยปกติจะถูกกำหนดไว้ล่วงหน้า (Default) เป็นเลขฐานสิบ เช่น -5, -1, 0, 8, 100 เป็นต้น
- ชนิดข้อมูล boolean มีค่าเป็นเท็จ (หรือ FALSE) และ จริง (หรือ TRUE)
- ชนิดข้อมูล std_logic ที่ใช้แทนชนิดข้อมูล bit ซึ่งมีได้ 9 สถานะดังนี้

'U' มีสถานะโลจิก Uninitialized	'W' มีสถานะโลจิก Weak unknown
'X' มีสถานะโลจิก Unknown	'L' มีสถานะโลจิก Weak low
'0' มีสถานะโลจิก Low (โลจิก'0')	'H' มีสถานะโลจิก Weak high
'1' มีสถานะโลจิก High (โลจิก'1')	'-' มีสถานะโลจิก Don't care
'Z' มีสถานะโลจิก High impedance	

ชนิดข้อมูลอาร์เรย์ (Array) หรือ Array types เป็นชนิดข้อมูลเดียวกันที่มีสมาชิกหลายตัว ชนิดข้อมูลที่ควรทราบได้แก่ ชนิดข้อมูล bit_vector เป็นชนิดข้อมูลอาร์เรย์ 1 มิติของ bit ชนิดข้อมูล std_logic_vector เป็นชนิดข้อมูลอาร์เรย์ 1 มิติของ std_logic ชนิดข้อมูล unsigned และ signed เป็นชนิดข้อมูลอาร์เรย์ 1 มิติของ std_logic โดยที่ชนิดข้อมูล unsigned และ signed (อยู่ในรูปฟอร์ม 2's complement) จะเขียนในฟอร์ม std_logic_vector และมีคุณสมบัติพิเศษกว่า std_logic_vector กือ สามารถใช้กับตัวดำเนินการทางคณิตศาสตร์ได้

เราสามารถเขียนโค้ด VHDL ของวงจร 2 to 4 Decoder โดยใช้ชนิดข้อมูล std_logic และชนิดข้อมูล std_logic_vector แทนชนิดข้อมูล bit และ bit_vector ได้ดังได้ดังรูปที่ 4.147

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.all; //บรรทัดที่ 2 และ 3 เป็นการเรียกใช้ Package ชื่อ std_logic_1164 ที่อยู่ใน
4 //ไลบรารี ชื่อ ieee เพื่อบอกคอมไไฟเลอร์ให้รู้จักชนิดข้อมูล std_logic
5 entity DECODER2TO4 is
6     port ( A0, A1 : in STD_LOGIC;
7            Y : out STD_LOGIC_VECTOR(3 downto 0)); //ชนิดข้อมูล std_logic
8 end DECODER2TO4; //และ std logic vector
9
10 architecture BEHAVIORAL of DECODER2TO4 is
11 begin
12     Y(0) <= (not A1) and (not A0);
13     Y(1) <= (not A1) and A0;
14     Y(2) <= A1 and (not A0);
15     Y(3) <= A1 and A0;
16 end BEHAVIORAL;

```

รูปที่ 4.147 โค้ด VHDL ของวงจร 2 to 4 Decoder ที่ใช้ชนิดข้อมูล std_logic และชนิดข้อมูล std_logic_vector

4.7.5 โอเปอเรเตอร์

ตัวดำเนินการหรือ โอเปอเรเตอร์ (Operator) ค่าๆ ในภาษา VHDL ตามมาตรฐาน IEEE Std 1076, IEEE Std 1164 และ IEEE Std 1076.3 รวมทั้ง Synopsys packages (Nonstandard package) ที่ผู้อ่านควรทราบได้แก่

- ตัวดำเนินการตรรกะ (Logical operators) ได้แก่ and, or, nand, nor, xor และ xnor (รวม not) ซึ่งทุกตัวจะมีลำดับความสำคัญเท่ากัน ยกเว้น not จะมีลำดับความสำคัญสูงสุด ตัวดำเนินการตรรกะ (รวม not) จะใช้ได้กับชนิดข้อมูล bit, bit_vector, boolean, std_logic, std_logic_vector, unsigned และ signed โดยที่ชนิดข้อมูลของเรย์ขนาดของเรย์ต้องเท่ากัน
- ตัวดำเนินการความสัมพันธ์ (Relational operators) ได้แก่ = (เท่ากัน), /= (ไม่เท่ากัน), > (มากกว่า), >= (มากกว่าหรือเท่ากัน), <= (น้อยกว่าหรือเท่ากัน) และ < (น้อยกว่า) โดยทุกตัวจะมีลำดับความสำคัญเท่ากัน ในกรณีที่ต้องการเปรียบเทียบชนิดข้อมูลของเรย์ที่มีขนาดของเรย์ไม่เท่ากัน ตัวดำเนินการความสัมพันธ์จะกระทำการโดยรีเมิร์ฟจากสามาชิกของอะเรย์ด้านซ้ายสุดก่อน
- ตัวดำเนินการรวมข้อมูลหรือ & (Concatenation operators) ใช้ในการรวมออบเจกต์ที่เป็นชนิดข้อมูลเดียวกันให้เป็นชนิดข้อมูลของเรย์ 1 มิติ ตัวอย่างเช่น

A <= "01" & "1101"; จะมีความหมายเดียวกับ A <= "011101";

B <= '0' & "111" & "0011"; จะมีความหมายเดียวกับ B <= "01110011";

C <= A & B; จะมีความหมายเดียวกับ C <= "01110101110011"

- ตัวดำเนินการคณิตศาสตร์ (Arithmetic operators) ได้แก่ +, -, *, / (คูณ), / (หาร), ** (ยกกำลัง), MOD (Modulus), REM (Remainder) และ ABS (Absolute value) ตัวดำเนินการคณิตศาสตร์สามารถใช้ได้กับชนิดข้อมูล unsigned และ signed รวมกับว่าเป็นชนิดข้อมูล integer โดยจะต้องเรียกใช้ Package ชื่อ Numeric_std ที่คอมไไฟล์ไว้ใน Library ieee ส่วนชนิดข้อมูล std_logic และ std_logic_vector นั้นไม่สามารถใช้กับตัวดำเนินการคณิตศาสตร์ได้ การแก้ไขปัญหานี้ทำได้โดยเรียกใช้ Package ชื่อ std_logic_signed หรือ std_logic_unsigned ของ Synopsys (Nonstandard package) ที่คอมไไฟล์ไว้ใน Library ieee ซึ่งจะยอมให้ทำการ +, -, * และ ABS ได้รวมกับว่าเป็นชนิดข้อมูล integer นอกจากนี้ std_logic_signed package หรือ std_logic_unsigned package ได้ทำฟังก์ชันแปลง (Conversion functions) ชนิดข้อมูล std_logic_vector เป็น integer ไว้ออกด้วยดังตารางในรูปที่ 4.148 การแปลงค่ากลับกันจาก integer เป็น std_logic_vector จะต้องเรียกใช้ std_logic_arith package และเนื่องจากภาษา VHDL ไม่ยอมให้มีการทำ Operation

หรือส่งผ่านค่าระหว่างชนิดข้อมูลต่างชนิดกัน ดังนี้จึงต้องแปลงชนิดข้อมูล (Type conversion) ให้เป็นชนิดเดียวกัน ก่อน การแปลงชนิดข้อมูล unsigned หรือ signed เป็น std_logic_vector หรือกลับกัน มีรายละเอียดดังรูปที่ 4.149

Conversion	Function
std_logic to integer	conv_integer(expression)
integer to std_logic	conv_std_logic(expression, size)

รูปที่ 4.148 ฟังก์ชันแปลงชนิดข้อมูลระหว่าง std_logic และ integer

Conversion	Explicit type conversion
std_logic_vector to unsigned	unsigned(expression)
std_logic_vector to signed	signed(expression)
unsigned to std_logic_vector	std_logic_vector(expression)
signed to std_logic_vector	std_logic_vector(expression)

รูปที่ 4.149 ตัวอย่างการแปลงชนิดข้อมูล unsigned หรือ signed กับ std_logic_vector

ตัวอย่างที่ 4.6 เขียนโค้ด VHDL ของรบวก (แบบ Unsigned) 2 บิตและเอาต์พุต 3 บิต โดยเรียกใช้ Package ชื่อ std_logic_unsigned ดังรูปที่ 4.150(a) และเรียกใช้ Numeric_std ดังรูปที่ 4.150(b) โดยที่โค้ดในบรรทัดที่ 14 นั้นจะมีการทำ Concatenation ของ ‘0’&A และ ‘0’&B เพื่อขยับขยายเรียบร้อย A และ B ให้เท่ากับ C ที่มีขนาด 3 บิต เพื่อไว้สำหรับตัวทดในการนับจากค่า “011”

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ADDER_2BIT is
7     port ( A,B : in STD_LOGIC_VECTOR (1 downto 0);
8             C : out STD_LOGIC_VECTOR (2 downto 0));
9 end ADDER_2BIT;
10
11 architecture BEHAVIORAL of ADDER_2BIT is
12 begin
13
14     C <= ('0' & A) + ('0' & B);
15
16 end BEHAVIORAL;

```

(a) โค้ด VHDL ของรบวกโดยเรียกใช้ Package ชื่อ std_logic_unsigned

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity ADDER_2BIT is
7     port ( A,B : in STD_LOGIC_VECTOR (1 downto 0);
8             C : out STD_LOGIC_VECTOR (2 downto 0));
9 end ADDER_2BIT;
10
11 architecture BEHAVIORAL of ADDER_2BIT is
12 begin
13
14     C <= std_logic_vector(unsigned('0' & A) + unsigned('0' & B));
15
16 end BEHAVIORAL;

```

(b) โค้ด VHDL ของรบวกโดยเรียกใช้ Numeric_std package

รูปที่ 4.150 โค้ด VHDL รูปแบบต่างๆ ของรบวก (แบบ Unsigned) 2 บิตและเอาต์พุต 3 บิต

4.7.6 ออกแบบ

ออกแบบ(Design) ในภาษา VHDL มี 4 ประเภท คือ ค่าคงที่ (Constant) สัญญาณ (Signal) ตัวแปร (Variable) และไฟล์ (File) หากให้ผู้อ่านลองทำความเข้าใจในเบื้องต้นเกี่ยวกับออกแบบที่เป็นสัญญาณ (Signal) ดังตัวอย่างด่อไปนี้

ตัวอย่างที่ 4.7 วงจรแมลติเพลก์เซอร์แบบ 2 อินพุต 1 เอาต์พุต (2 to 1 Multiplexer) มีผังวงจรแสดงดังรูปที่ 4.151 กล่าวคือ มี A และ B เป็นอินพุต มี C เป็นเอาต์พุต และ S เป็นขาควบคุม (อินพุต) เมื่อ S = '0' แล้วจะเลือกอินพุต A ไปที่เอาต์พุต C และ เมื่อ S = '1' จะเลือกอินพุต B ไปที่เอาต์พุต C ซึ่งความสามารถเขียนໂຄด์ VHDL ได้ดังรูปที่ 4.152a) หรือรูปที่ 4.152b)

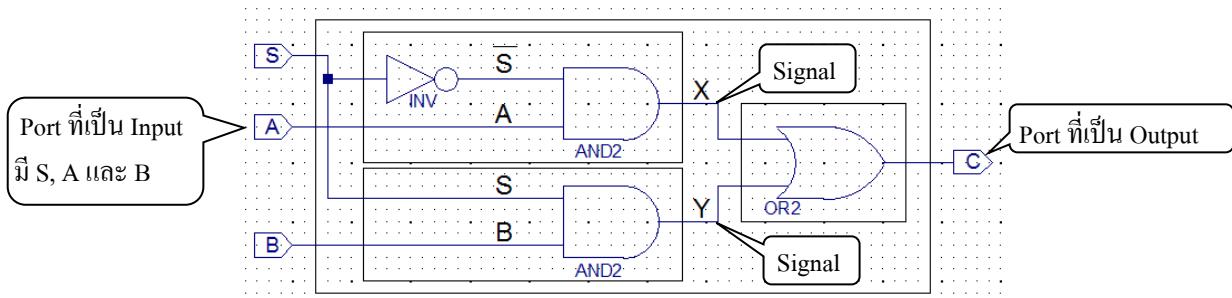
วงจรในรูปที่ 4.151 จะประกอบด้วยวงจรชุด 3 วงจร โดยมี X และ Y เป็นสายสัญญาณ (Signal) หรือสายไฟเชื่อมต่อที่อยู่ภายในวงจรแมลติเพลก์เซอร์หรือเชื่อมต่อระหว่างวงจรชุดที่อยู่ภายนอก ความสามารถเขียนสมการบูลีนได้ดังนี้

$$X = \overline{S} \cdot A$$

$$Y = S \cdot B$$

$$C = X + Y \quad \text{หรือ} \quad C = \overline{S} \cdot A + S \cdot B$$

ในภาษาVHDL นั้นจะเรียก X หรือ Y นี้ว่า Signal ซึ่งชนิดข้อมูลของ Signal X และ Signal Y ในกรณีนี้เราทำหน้าที่เป็นชนิดข้อมูล std_logic การประกาศใช้ Signal (Signal declaration) นั้นเราจะประกาศใช้ในบริเวณที่อยู่ระหว่าง architecture และ begin ดังรูปที่ 4.152(a) ซึ่งเราจะประกาศใช้ Signal ก็ต่อเมื่อชื่อของ Signal นั้นไม่มีชื่อปรากฏใน port ของ entity และจากรูปที่ 4.152(a) จะเห็นได้ว่าคำสั่งบอย 1 คำสั่ง (ซึ่งเป็นคำสั่งคอนโทรลเรนต์) จะแทนวงจรชุด 1 วงจร ดังนั้น Signal ในภาษา VHDL จึงทำหน้าที่ส่งผ่านหรือแลกเปลี่ยนข้อมูลระหว่างคำสั่งคอนโทรลเรนต์ต่างๆ



รูปที่ 4.151 ผังวงจร 2 to 1 Multiplexer

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity MUX2TO1 is
6     port ( A,B : in STD_LOGIC;
7             S : in STD_LOGIC;
8             C : out STD_LOGIC);
9 end MUX2TO1;
10
11 architecture BEHAVIORAL of MUX2TO1 is
12     signal X,Y : STD_LOGIC;
13 begin
14     X <= (not S)and A;
15     Y <= S and B;
16     C <= X or Y;
17 end BEHAVIORAL;

```

การประกาศใช้ Signal X และ Signal Y นี้เราจะประกาศใช้ก็ต่อเมื่อชื่อของ Signal นั้นไม่มีชื่อปรากฏใน port ของ entity

(a) โค้ด VHDL วงจร 2 to 1 Multiplexer ที่เขียนโดยประกาศใช้ Signal (เมื่อเรามองเห็นเป็นวงจรย่อย 3 วงจร)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity MUX2TO1 is
6     port ( A,B : in STD_LOGIC;
7             S : in STD_LOGIC;
8             C : out STD_LOGIC);
9 end MUX2TO1;
10
11 architecture BEHAVIORAL of MUX2TO1 is
12 begin
13     C <= ((not S)and A)or(S and B);
14 end BEHAVIORAL;

```

(b) โค้ด VHDL วงจร 2 to 1 Multiplexer ที่เขียนโดยไม่ประกาศใช้ Signal (เมื่อเรามองเห็นเป็นวงจรใหญ่เพียงวงจรเดียว)

รูปที่ 4.152 โค้ด VHDL ของวงจร 2 to 1 Multiplexer ที่เขียนทั้งแบบประกาศใช้และไม่ประกาศใช้ Signal

4.7.7 Attribute

Attribute เป็นคุณสมบัติของ VHDL ที่ขอมให้มีการเพิ่มรายละเอียดให้แก่องเจ็คต์ (Object) เช่น

C' event มีความหมายว่า การเปลี่ยนแปลง (ด้วยขอบขาขึ้นหรือขาลง) ของสัญญาณ C (Clock)

C' event and C = '1' มีความหมายว่า ทริกค์ขาขึ้นของสัญญาณ C

C' event and C = '0' มีความหมายว่า ทริกค์ขาลงของสัญญาณ C

4.7.8 คำสั่งคอนเคอร์เรนซ์

คำสั่งคอนเคอร์เรนซ์ที่ควรทราบ ได้แก่ Simple signal assignment statement, Selected signal assignment statement, Conditional signal assignment statement และ Process statement

- คำสั่ง Simple signal assignment จะถูกใช้ในการกำหนดค่าให้กับสัญญาณหรือพอร์ต โดยมีรูปแบบการเขียนดังนี้

```
[ label : ] TARGET <= EXPRESSION ;
```

ตัวอย่างคำสั่ง Simple signal assignment เช่น

```
C <= A and B ;
```

```
X <= "00000000" ;
```

เราเรียก “=<” ว่า Signal assignment และเราอาจใช้ X <= (others => '0') แทน X <= "00000000" ได้ ซึ่งเป็นการกำหนดค่า '0' ให้ X จนครบทุกบิต ในทำนองเดียวกันเราอาจใช้ X <= (others => '1') แทน X <= "11111111" ได้เช่นกัน

- คำสั่ง Selected signal assignment มีรูปแบบการเขียนเป็นดังนี้

```
[ label : ] with CHOICE_EXPRESSION select
TARGET <= { EXPRESSION when CHOICE { | CHOICE}, }
           EXPRESSION when CHOICE { | CHOICE};
```

โดยที่ 1) สัญลักษณ์ {...} หมายถึงจะมีหรือไม่มีก็ได้หรืออาจมีได้หลายตัว แต่ถ้าเป็น [...] หมายถึงจะมีหรือไม่มีก็ได้
2) การตั้งชื่อต่างๆ ต้องเป็นตามเกณฑ์ในข้อ 4.7.2

คำสั่งนี้จะกำหนดค่า EXPRESSION ให้กับ TARGET เมื่อ CHOICE มีค่าตรงกับค่าใน CHOICE_EXPRESSION โดยที่ค่าของ CHOICE อาจอยู่ในรูปของค่าที่แน่นอน (Static expression เช่น 3) หรือ มีค่าเป็นช่วง (Static range เช่น 7 downto 4) หรือ ค่าที่ไม่มีความต่อเนื่องหลายค่า เช่น "0001" | "0101" | "0111" เป็นต้น (โดยที่ | หมายถึง OR) และอาจใช้คำว่า "OTHERS" แทน CHOICE เหลือทั้งหมดที่เป็นไปได้ และเนื่องจากแต่ละ CHOICE ไม่มีลำดับความสำคัญ (Priority) ดังนั้นทุก CHOICE จะต้องไม่ซ้ำกันและต้องกำหนดให้ครบถ้วน CHOICE ที่เป็นไปได้

- คำสั่ง Conditional signal assignment มีรูปแบบการเขียนดังนี้

```
[ label : ] TARGET <= { EXPRESSION when CONDITION else }
                           EXPRESSION ;
```

โดยที่ 1) สัญลักษณ์ {...} หมายถึงจะมีหรือไม่มีก็ได้หรืออาจมีได้หลายตัว แต่ถ้าเป็น [...] หมายถึงจะมีหรือไม่มีก็ได้
2) การตั้งชื่อต่างๆ ต้องเป็นตามเกณฑ์ในข้อ 4.7.2

คำสั่งนี้จะกำหนด EXPRESSION ให้กับ TARGET เมื่อตรวจสอบ CONDITION เป็นจริง (True) อันแรกสุดก่อน โดยจะไม่สนใจ CONDITION ที่เป็นจริงที่อยู่ในลำดับถัดไป (ซึ่งอาจมีหลาย CONDITION) ดังนั้นแต่ละ CONDITION ในคำสั่งนี้จึงมีลำดับความสำคัญ (Priority) และในกรณีที่ไม่พบ CONDITION เป็นจริงก็ให้กำหนด EXPRESSION สุดท้ายให้กับ TARGET

ตัวอย่างที่ 4.8 ฝึกใช้คำสั่ง Selected signal assignment และ Conditional signal assignment ในการออกแบบวงจรแมตติเพล็กเซอร์ 4 อินพุต 1 เอาต์พุต โดยใช้สัญญาณ Control S0 และ S1 เป็นตัวเลือกอินพุต ดังมีรายละเอียดตามตารางความจริงในรูปที่ 4.153a) ซึ่งสามารถเขียนโค้ด VHDL ได้ดังรูปที่ 4.153(b) ถึงรูปที่ 4.153(c)

Control		Output	Remark
S1	S0	O	
0	0	A	$O = A$
0	1	B	$O = B$
1	0	C	$O = C$
1	1	D	$O = D$

(a) ตารางความจริงวงจรแมตติเพล็กเซอร์ 4 อินพุต 1 เอาต์พุต

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity MUX4TO1 is
6     port ( A,B,C,D,S1,S0 : in STD_LOGIC;
7             O : out STD_LOGIC);
8 end MUX4TO1;
9
10 architecture BEHAVIORAL of MUX4TO1 is
11     signal SEL : STD_LOGIC_VECTOR(1 downto 0);
12 begin
13     SEL <= S1&S0;           -- Concatenation
14     with SEL select
15         O <= A when "00",
16             B when "01",
17             C when "10",
18             D when others; -- SEL="11"
19 end BEHAVIORAL;

```

ต้องประกาศใช้ signal เนื่องจาก SEL ไม่มีชื่อใน port ของ entity

การรวมชนิดข้อมูลเป็นแบบหลายบิตโดยใช้ "&" (Concatenation)

--" หรือ Comment ใส่คำอธิบายตามข้อความนี้

(b) โค้ดวงจรแมตติเพล็กเซอร์ที่เขียนด้วยคำสั่ง Selected signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity MUX4TO1 is
6     port ( A,B,C,D,S1,S0 : in STD_LOGIC;
7             O : out STD_LOGIC);
8 end MUX4TO1;
9
10 architecture BEHAVIORAL of MUX4TO1 is
11     signal SEL : STD_LOGIC_VECTOR(1 downto 0);
12 begin
13     SEL <= S1&S0;           -- Concatenation
14     O <= A when SEL="00" else
15         B when SEL="01" else
16         C when SEL="10" else
17         D;                  -- SEL="11"
18 end BEHAVIORAL;

```

ต้องประกาศใช้ signal เนื่องจาก SEL ไม่มีชื่อใน port ของ entity

การรวมชนิดข้อมูลเป็นแบบหลายบิตโดยใช้ "&" (Concatenation)

--" หรือ Comment ใส่คำอธิบายตามข้อความนี้

(c) โค้ดวงจรแมตติเพล็กเซอร์ที่เขียนด้วยคำสั่ง Conditional signal assignment

รูปที่ 4.153 โค้ดวงจรแมตติเพล็กเซอร์ 4 อินพุต 1 เอาต์พุต

จากโค้ดในรูปที่ 4.153b) เมื่อ SEL = "00" (คือ S1 = '0' และ S0 = '0') ให้กำหนดค่าเอาต์พุต O จากอินพุต A แต่ถ้า SEL = "01" (S1 = '0' และ S0 = '1') ให้กำหนดค่าเอาต์พุต O จากอินพุต B ถ้า SEL = "10" (S1 = '1' และ S0 = '0') ให้กำหนดค่าเอาต์พุต O จากอินพุต C และถ้า SEL มีสถานะล็อกิกเป็นค่าที่เหลือทั้งหมด (when others) โดยรวมสถานะในกรณีที่

SEL = “11” เป้าไปด้วย ก็ให้กำหนดค่าเอาต์พุต O จากอินพุต D การใช้คำว่า when others จึงสมมุติเป็นการเขียนเพื่อให้ได้ครอบคลุมทุกเงื่อนไขที่เป็นไปได้ทั้งหมด เพราะว่า S1 และ S0 เป็นชนิดข้อมูล std_logic แต่ละตัวจึงมีสถานะลงจิ๊กได้ 9 สถานะ

จากໄก์ในรูปที่ 4.153(c) มีความหมายว่า เมื่อ SEL = “00” ($S1 = '0'$ และ $S0 = '0'$) เป็นจริงแล้วให้กำหนดค่าเอาต์พุต O จากอินพุต A แต่ถ้า SEL = “00” เป็นเท็จก็จะไปทำ CONDITION ไข้ดังไป ถ้า SEL = “01” ($S1 = '0'$ และ $S0 = '1'$) เป็นจริงแล้วให้กำหนดค่าเอาต์พุต O จากอินพุต B แต่ถ้า SEL = “01” เป็นเท็จก็จะไปทำ CONDITION ถัดไป และถ้า SEL = “10” ($S1 = '1'$ และ $S0 = '0'$) เป็นจริงแล้วให้กำหนดค่าเอาต์พุต O จากอินพุต C ถ้าไม่พบ CONDITION ที่เป็นจริง (โดยรวมกรณีที่ SEL = “11” แล้ว) ก็ให้กำหนดค่าเอาต์พุต O จาก EXPRESSION สุดท้าย คือ อินพุต D

ตัวอย่างที่ 4.9 ออกแบบวงจรเข้ารหัสที่มีลำดับความสำคัญ (Priority encoder) ที่ใช้แปลงสัญญาณปุ่มกดแบบ Active low เป็นเลขไบナรี ถ้ากดปุ่ม A(0) จะได้อเอาต์พุต B = “00” ถ้ากดปุ่ม A(1) จะได้ B = “01” ถ้ากดปุ่ม A(2) จะได้ B = “10” ถ้าไม่กดปุ่ม จะได้ B = “11” และเมื่อกดปุ่มพร้อมกันหลายปุ่มเอาต์พุตของปุ่มกดที่มีค่าสูงสุด โค้ดของวงจรเข้ารหัสแสดงดังรูปที่ 4.154

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity KEYPAD_PRIORITY_ENCODER is
6     Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
7             B : out STD_LOGIC_VECTOR (1 downto 0));
8 end KEYPAD_PRIORITY_ENCODER;
9
10 architecture Behavioral of KEYPAD_PRIORITY_ENCODER is
11 begin
12     B <= "10" when A(2) = '0' else --KEYPAD No.2 : B(1)='1',B(0)='0'
13         "01" when A(1) = '0' else --KEYPAD No.1 : B(1)='0',B(0)='1'
14         "00" when A(0) = '0' else --KEYPAD No.0 : B(1)='0',B(0)='0'
15         "11";                      --                                B(1)='1',B(0)='1'
16 end Behavioral;

```

รูปที่ 4.154 โค้ดของวงจรเข้ารหัสที่มีลำดับความสำคัญที่เขียนโดยใช้คำสั่ง Conditional signal assignment

- คำสั่งโพรเซส (Process statement) เป็นคำสั่งคอนแคร์เรนต์แต่ภายในจะเป็นคำสั่งซึ่งควบเชิขล ซึ่งเราจะอธิบายละเอียด ในข้อ 4.7.9 คำสั่ง Process มีรูปแบบการเขียนดังนี้

```

[ label : ] process [ (SENSITIVITY LIST) ]
    {PROCESS_DECLARATIVE_PART}
    begin
        {SEQUENTIAL_STATEMENT}
    end process [ label ];

```

โดยที่ PROCESS_DECLARATIVE_PART ที่ควรทราบในเบื้องต้นได้แก่ ประกาศใช้ตัวแปร (Variable declaration) ซึ่ง Variable จะต่างจาก Signal ตรงที่ใช้ได้เฉพาะภายใน Process เท่านั้นและสามารถ Update ค่าได้ทันที ในขณะที่ Signal จะ Update ค่าได้เมื่อจบ Process

- 2) SENSITIVITY LIST คือ อินพุตทุกตัวมีผลต่อการเปลี่ยนแปลงของ Process โดยตรง (ไม่ใช่โดยทางอ้อม)
- 3) SEQUENTIAL_STATEMENT คือ สั่งซึ่งควบเชิขล

4.7.9 คำสั่งชีวนิรเมชย์ล

คำสั่งชีวนิรเมชย์ล (Sequential statement) จะมีลำดับความสำคัญก่อน-หลัง โดยจะต้องเขียนไว้ภายในคำสั่ง Process โดยที่คำสั่ง Process แต่ละคำสั่งจะเป็นคำสั่งคอนโทรลเรนต์ ยกเว้นคำสั่งที่อยู่ภายใต้ Process จะเป็นคำสั่งชีวนิรเมชย์ล คำสั่งที่ควรทราบได้แก่ Signal assignment statement, Variable assignment statement, If statement และ Case statement

- คำสั่ง Signal assignment ใช้ในการกำหนดค่าให้กับสัญญาณ (Signal) หรือพอร์ต (Port) มีรูปแบบการเขียนดังนี้

```
TARGET <= EXPRESSION ;
```

- คำสั่ง Variable assignment ใช้ในการกำหนดค่าให้กับ Variable ที่อยู่ในคำสั่ง Process เท่านั้น มีรูปแบบการเขียนดังนี้

```
TARGET := EXPRESSION ;
```

- คำสั่ง if เป็นคำสั่งชีวนิรเมชย์ล มีรูปแบบการเขียนคำสั่งดังนี้

```
if CONDITION then {SEQUENTIAL_STATEMENT}
{ elsif CONDITION then {SEQUENTIAL_STATEMENT} }
[ else {SEQUENTIAL_STATEMENT} ]
end if;
```

คำสั่ง if จะคล้ายๆ กับคำสั่ง Conditional signal assignment แต่คำสั่ง if จะต้องเขียนไว้ในคำสั่ง Process เมื่อตรวจ CONDITION (ทางบูลีน) และแล้วพบว่าเป็นจริง (True) ก็จะทำการ SEQUENTIAL_STATEMENT (ซึ่งอาจมีหลายคำสั่ง) ที่อยู่หลัง then แต่ถ้าเป็นเท็จ (False) ก็จะทำการตรวจสอบ CONDITION อีกไปจนกว่าจะพบ CONDITION ที่เป็นจริงแล้วจึงทำการ SEQUENTIAL_STATEMENT นั้นโดยไม่สนใจ CONDITION ที่เหลือ ดังนั้นแต่ละ CONDITION จึงมีลำดับความสำคัญ (Priority) ถ้าไม่มี CONDITION ที่เป็นจริงแต่มีคำสั่ง else ก็จะทำการ SEQUENTIAL_STATEMENT ที่อยู่หลัง else แต่ถ้าไม่มี CONDITION ที่เป็นจริงเลยและไม่มีคำสั่ง else อยู่ด้วยก็ไม่ต้องทำการ SEQUENTIAL_STATEMENT ใดๆ ซึ่งก็หมายความว่าสถานะล็อกของเอาต์พุตก่อนหน้านั้นถูกคงค่า (Hold) ไว้หรือ Latch ค่าล็อกนั้นไว้ นั่นก็แสดงว่ามีวงจร “Latch” เกิดขึ้น

- คำสั่ง case เป็นคำสั่งชีวนิรเมชย์ล มีรูปแบบการเขียนเป็นดังนี้

```
case EXPRESSION is
when CHOICES => {SEQUENTIAL_STATEMENT}
{ when CHOICES => {SEQUENTIAL_STATEMENT} }
end case;
```

คำสั่ง case จะคล้ายๆ กับคำสั่ง Selected signal assignment แต่คำสั่ง case จะต้องเขียนไว้ในคำสั่ง Process เมื่อค่าของ CHOICES ตรงกับค่าใน EXPRESSION ก็จะทำการ SEQUENTIAL_STATEMENT (อาจมีหลายๆ คำสั่ง) โดยที่ค่าของ CHOICES อาจเป็นค่าที่แน่นอน (Static expression เช่น 3) หรือ เป็นช่วง (Static range เช่น 7 downto 4) หรือ ค่าที่ไม่ต่อเนื่องหลายค่า เช่น “0001” | “0101” | “0111” เป็นต้น (โดยที่ | หมายถึง OR) เราอาจใช้ “OTHERS” แทน CHOICES เพื่อทั้งหมด

และเนื่องแต่ละ CHOICES ไม่มีการลำดับความสำคัญ (Priority) ดังนั้น CHOICES จะต้องไม่ซ้อนกันและต้องกำหนดให้ครบทุก CHOICES

ตัวอย่างที่ 4.10 ทำความเข้าใจเกี่ยวกับ Sensitivity list โดยเขียนโค้ดงาน D Flip-flop แสดงดังรูปที่ 4.155 โดยที่ C คือ สัญญาณนาฬิกา (Clock) ที่ทริกด้วยขอบขาขึ้นหรือขอบลง รูปที่ 4.155(a) เป็น D Flip-flop แบบ Asynchronous clear กล่าวคือ เมื่อขา CLR = '1' แล้วเอาต์พุต Q = '0' (ทันที) โดยไม่ต้องรอการทริกด้วยสัญญาณนาฬิกา ส่วนรูปที่ 4.155(b) เป็น D Flip-flop แบบ Synchronous reset กล่าวคือ เมื่อขา RESET = '1' แล้วจะรีเซ็ตทันที แต่จะต้องรอสัญญาณนาฬิกา (Clock) ทริกก่อนแล้วจึงรีเซ็ตเอาต์พุต Q = '0'

จากโค้ดในรูปที่ 4.155(a) ถ้าอินพุต C และ CLR เป็นสถานะกี่จะมีผลต่อเอาต์พุต Q ทันที จึงต้องเขียนอินพุต C และ CLR ไว้ใน Sensitivity list แต่ถ้าเป็นโค้ดในรูปที่ 4.155(b) นั้นเอาต์พุต Q เป็นสถานะกี่ต่อเมื่อมีการทริกด้วยอินพุต C (สัญญาณนาฬิกา) เท่านั้น เราจึงเขียนอินพุต C เพียงตัวเดียวไว้ใน Sensitivity list

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DFF_CLR is
6   port ( D,C,CLR : in STD_LOGIC;
7         Q : out STD_LOGIC);
8 end DFF_CLR;
9
10 architecture BEHAVIORAL of DFF_CLR is
11 begin
12 process(C,CLR)
13 begin
14   if CLR='1' then Q <= '0';
15   elsif (C'event and C='1') then Q <= D;
16   end if;
17 end process;
18 end BEHAVIORAL;

```

การเปลี่ยนแปลงของอินพุต C และ CLR นั้นจะมีผลต่อเอาต์พุต Q โดยตรง (ทันที) จึงต้องเขียน C และ CLR ไว้ใน Sensitivity list

C'event and C='1' คือ C ทริกด้วยขอบขาขึ้น หรือขอบลง ซึ่งจะไส่่งเล็บหรือไม่ไส่่ก็ได้

(a) โค้ดงาน D Flip-flop แบบ Asynchronous clear

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DFF_RESET is
6   port ( D,C,RESET : in STD_LOGIC;
7         Q : out STD_LOGIC);
8 end DFF_RESET;
9
10 architecture BEHAVIORAL of DFF_RESET is
11 begin
12 process(C)
13 begin
14   if (C'event and C='1') then
15     if RESET='1' then Q <= '0';
16     else Q <= D;
17   end if;
18 end if;
19 end process;
20 end BEHAVIORAL;

```

การ RESET ไม่มีผลต่อเอาต์พุต Q โดยตรง (ทันที) เพราะต้องรอการทริกจาก C แต่การทริกด้วย C จะมีผลต่อเอาต์พุต Q โดยตรง จึงเขียน C เพียงตัวเดียวใน Sensitivity list

C'event and C='1' คือ C ทริกด้วยขอบขาขึ้น หรือขอบลง ซึ่งจะไส่่งเล็บหรือไม่ไส่่ก็ได้

(b) โค้ดงาน D Flip-flop แบบ Synchronous reset

รูปที่ 4.155 โค้ดงาน D Flip-flop

ตัวอย่างที่ 4.11 ฟิกการเขียนโค้ดวงจรนับ 16 (4 บิต) แบบนับขึ้นในรูปแบบต่างๆ แสดงดังรูปที่ 4.156(a) ถึงรูปที่ 4.156(e)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity COUNTER4BIT is
7     port ( C,CLR : in STD_LOGIC;
8             Q : buffer STD_LOGIC_VECTOR (3 downto 0));
9 end COUNTER4BIT;
10
11 architecture Behavioral of COUNTER4BIT is
12 begin
13     process(C,CLR)
14     begin
15         if CLR='1' then Q <= "0000";
16         elsif C'event and C='1' then Q <= Q + 1;
17         end if;
18     end process;
19 end Behavioral;

```

เรียกใช้ Package ชื่อ std_logic_unsigned เพื่อให้ชนิดข้อมูล std_logic_vector สามารถใช้กับ “+” ได้

Port Q สามารถอ่านค่ากลับได้ดังนี้จึงต้องใช้ Mode “buffer”

เป็นการอ่านค่าของ Port Q กลับเข้ามาเพื่อเพิ่มค่าครึ่งละ 1 แล้วส่งค่าใหม่ไปที่ Port Q อีกครึ่ง นั่นก็แสดงว่า Port Q เป็น Port ที่อ่านค่ากลับได้หรือ “buffer”

(a) โค้ดวงจรนับ 16 แบบนับขึ้นที่มีเอาต์พุต Q เป็นชนิดข้อมูล std_logic_vector และมี Mode เป็น “buffer”

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity COUNTER4BIT is
7     port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end COUNTER4BIT;
10
11 architecture Behavioral of COUNTER4BIT is
12     signal Q_temp : STD_LOGIC_VECTOR (3 downto 0);
13 begin
14     process(C,CLR)
15     begin
16         if CLR='1' then Q_temp <= "0000";
17         elsif C'event and C='1' then Q_temp <= Q_temp + 1;
18         end if;
19     end process;
20     Q <= Q_temp;
21 end Behavioral;

```

เรียกใช้ Package ชื่อ std_logic_unsigned เพื่อให้ชนิดข้อมูล std_logic_vector สามารถใช้กับ “+” ได้

signal Q_temp ไม่ใช่ Port จึงไม่มีความจำเป็นต้องใช้ Mode “buffer”

กำหนดค่า (Assign) Q_temp ซึ่งเป็น Signal ให้กับ Q นั้นต้องทำภายนอก Process เนื่องจาก การ Update ค่าของ Signal จะกระทำเมื่อจบ Process

(b) โค้ดวงจรนับ 16 แบบนับขึ้นที่มีเอาต์พุต Q เป็นชนิดข้อมูล std_logic_vector และประกาศใช้ Signal

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity COUNTER4BIT is
7     port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end COUNTER4BIT;
10
11 architecture Behavioral of COUNTER4BIT is
12 begin
13     process(C,CLR)
14     variable Q_var : STD_LOGIC_VECTOR (3 downto 0);
15     begin
16         if CLR='1' then Q_var := "0000";
17         elsif C'event and C='1' then Q_var := Q_var + 1;
18         end if;
19         Q <= Q_var;
20     end process;
21 end Behavioral;

```

เรียกใช้ Package ชื่อ std_logic_unsigned เพื่อให้ชนิดข้อมูล std_logic_vector สามารถใช้กับ “+” ได้

Variable Q_var ไม่ใช่ Port จึงไม่มีความจำเป็นต้องใช้ Mode “buffer”

ต้องประกาศใช้ Variable ภายใน Process เท่านั้น

Variable assignment

Variable ใช้ได้เฉพาะภายใน Process เท่านั้น ดังนั้นการส่งผ่านค่าออกนอก Process จึงต้อง Assign ค่า Q_var ให้กับ Q (Port หรือ Signal) ก่อน

(c) โค้ดวงจรนับ 16 แบบนับขึ้นที่มีเอาต์พุต Q เป็นชนิดข้อมูล std_logic_vector และประกาศใช้ Variable

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity COUNTER4BIT is
7     port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end COUNTER4BIT;
10
11 architecture Behavioral of COUNTER4BIT is
12     signal Q_temp : unsigned (3 downto 0);
13 begin
14     process(C,CLR)
15     begin
16         if CLR='1' then Q_temp <= "0000";
17         elsif C'event and C='1' then Q_temp <= Q_temp + 1;
18         end if;
19     end process;
20     Q <= std_logic_vector ( Q_temp );
21 end Behavioral;

```

เรียกใช้ Package ชื่อ numeric_std เพื่อให้ชนิดข้อมูล unsigned สามารถใช้กับ “+” ได้

signal Q_temp ไม่ใช่ Port จึงไม่มีความจำเป็นต้องใช้ Mode “buffer”

แปลงชนิดข้อมูลของ Q_temp ซึ่งเป็น unsigned ไปเป็น std_logic_vector แล้วกำหนดค่าให้กับ Q

(d) โค้ดวงจรนับ 16 แบบนับขึ้นที่มีเอาต์พุต Q เป็นชนิดข้อมูล std_logic_vector และ Signal เป็น unsigned

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity COUNTER4BIT is
7     port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end COUNTER4BIT;
10
11 architecture Behavioral of COUNTER4BIT is
12 begin
13     process(C,CLR)
14     variable Q_var : unsigned (3 downto 0);
15     begin
16         if CLR='1' then Q_var := "0000";
17         elsif C'event and C='1' then Q_var := Q_var + 1;
18         end if;
19         Q <= std_logic_vector ( Q_var );
20     end process;
21 end Behavioral;

```

เรียกใช้ Package ชื่อ numeric_std เพื่อให้ชนิดข้อมูล unsigned สามารถใช้กับ “+” ได้

Variable Q_var ไม่ใช่ Port จึงไม่มีความจำเป็นต้องใช้ Mode “buffer”

ต้องประกาศใช้ Variable ก่อนใน Process เท่านั้น

แปลงชนิดข้อมูลของ Q_var จาก unsigned ไปเป็น std_logic_vector แล้วกำหนดค่าให้กับ Q

(e) โค้ดวงจรนับ 16 แบบนับขึ้นที่มีเอาต์พุต Q เป็นชนิดข้อมูล std_logic_vector และ Variable เป็น unsigned

รูปที่ 4.156 โค้ดวงจรนับ 16 แบบนับขึ้น

ในบรรทัดที่ 16 รูปที่ 4.156(a) $Q \leq Q + 1$ คือ การอ่านค่า Port Q ที่เป็นเอาต์พุตกลับเข้ามาในวงจรเพื่อเพิ่มค่าครึ่งละ 1 แล้วส่งค่าใหม่ออกไปอีกครึ่ง แสดงว่า Port Q เป็นเอาต์พุตที่อ่านค่ากลับได้ จึงต้องใช้ Port mode เป็น “buffer”

รูปที่ 4.156(b) ถึงรูปที่ 4.156(e) เป็นการเขียนโค้ดเพื่อหลีกเลี่ยงการใช้ “buffer” เมื่อเรียกใช้ std_logic_unsigned package แล้วจะทำให้ $Q \leq Q + 1$ ในโค้ดรูปที่ 4.156(a) ถึงรูปที่ 4.156(c) ให้ผลเซ็นเดียวกับ $Q \leq Q + "0001"$ หรือ $Q \leq Q + '1'$ และเมื่อเรียกใช้ numeric_std package แล้วจะทำให้ $Q \leq Q + 1$ ในโค้ดรูปที่ 4.156(d) ถึงรูปที่ 4.156(e) ให้ผลเซ็นเดียวกับ $Q \leq Q + "0001"$ หรือ $Q \leq Q + "1"$ (“1” เป็นชนิดข้อมูลแบบอะเรย์ Variable จะใช้ได้เฉพาะใน Process การสั่งค่าออกภายนอก Process จะต้อง Assign ค่าให้กับ Port หรือ Signal ดังรูปที่ 4.156(c) และรูปที่ 4.156(e) Variable จะ Update ค่าได้ทันทีซึ่งต่างจาก Signal ที่จะ Update ค่าได้ทีต่อเมื่อจบ Process ดังนั้นจึงต้อง Assign ค่า Signal ภายนอก Process ดังรูปที่ 4.156(b) และรูปที่ 4.156(d)

ถ้าเราให้ค่าด้วยจรนับ 16 รูปที่ 4.156(b) ถึงรูปที่ 4.156(e) ไป Simulation จะได้ค่าเอาต์พุตเริ่มต้นเป็น 'U' (Uninitialized) แล้วค่าจะเปลี่ยนเป็น 'X' (Unknown) และเมื่อ Clear (CLR) = '1' แล้วจึงนับค่าปกติ การแก้ไขปัญหานี้ Xilinx Synthesis Tool หรือ XST ของให้ใส่ค่าเริ่มต้นดังตัวอย่างในรูปที่ 4.157 ผล Simulation แสดงดังรูปที่ 4.158(a) และรูปที่ 4.158(b)

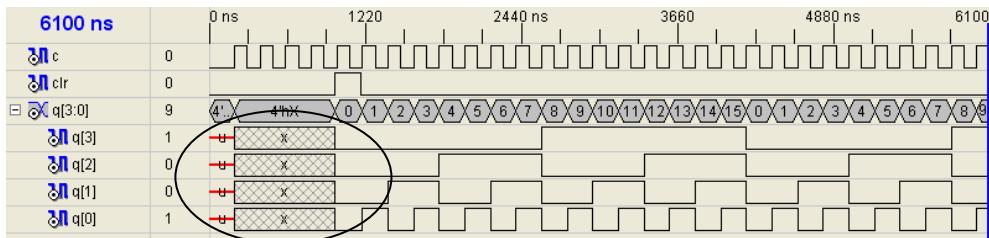
```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity COUNTER4BIT is
7     port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end COUNTER4BIT;
10
11 architecture Behavioral of COUNTER4BIT is
12     signal Q_temp : STD_LOGIC_VECTOR (3 downto 0):="0000";
13 begin
14     process(C,CLR)
15     begin
16         if CLR='1' then Q_temp <= "0000";
17         elsif C'event and C='1' then Q_temp <= Q_temp + 1;
18         end if;
19     end process;
20     Q <= Q_temp;
21 end Behavioral;

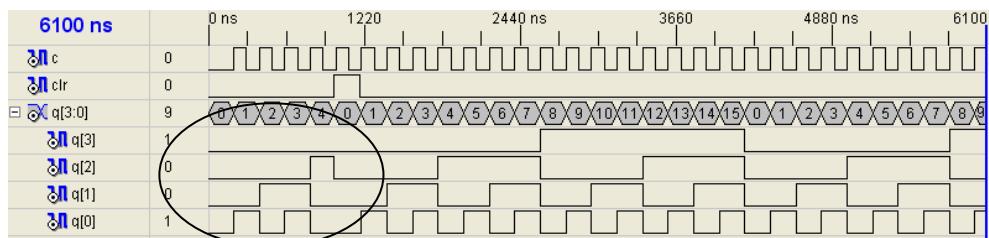
```

การใส่ค่าเริ่มต้น Q_temp
ให้มีค่าเป็น "0000"

รูปที่ 4.157 ตัวอย่างโค้ดวงจรนับ 16 ที่ใส่ค่าเริ่มต้นเป็น "0000"



(a) ผล Simulation โค้ดวงจรนับ 16 ที่ไม่ได้ใส่ค่าเริ่มต้นในโค้ดวงจรนับ 16



(b) ผล Simulation โค้ดวงจรนับ 16 ที่ใส่ค่าเริ่มต้นเป็น "0000" ในโค้ดวงจรนับ 16

รูปที่ 4.158 ผล Simulation ของโค้ดวงจรนับ 16

ตัวอย่างที่ 4.12 ฝึกเขียนโค้ดวงจรนับ 10 (4 บิต) แบบนับขึ้นแสดงดังรูปที่ 4.159(a) และรูปที่ 4.159(b) ซึ่งโค้ดบรรทัดที่ 18 ควรเขียนเป็น $Q \geq 9$ แทน $Q = 9$ เพื่อป้องกันการนับนอก Range กีอ 10-15 (1010, 1011, 1100, 1101, 1110, 1111) ซึ่งอาจเกิดขึ้นขณะเริ่มจ่ายไฟเลี้ยง ถ้าเริ่มต้นวงจรนับนอก Range กีจะนับต่อไปจนถึง 15 แล้วจึงกลับมาต้นใน Range ตามปกติ (0-9) อีกครั้งในรูปที่ 4.159(a) และรูปที่ 4.159(b) ถ้าเราเขียนใช้ std_logic_unsigned Package แล้ว $QT \geq 9$ จะให้ผลเหมือนกับ $QT \geq "1001"$

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity COUNTER10UP is
7     Port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR(3 downto 0));
9 end COUNTER10UP;
10
11 architecture Behavioral of COUNTER10UP is
12     signal QT : STD_LOGIC_VECTOR (3 downto 0);
13 begin
14     process(C,CLR)
15     begin
16         if CLR='1' then QT <= "0000";
17         elsif C'event and C='1' then
18             if QT >= 9 then QT <= "0000"; เมื่อค่า QT >= 9 แล้วมีสัญญาณนาฬิกา C ทริกค์  
ด้วยขอบขาขึ้นจะทำให้ QT มีค่าเป็น "0000"
19             else QT <= QT + 1;
20             end if;
21         end if;
22     end process;
23     Q <= QT;
24 end Behavioral;

```

(a) โค้ดควบจรนับ 10 แบบนับขึ้นที่มีอادر์พุต Q เป็นชนิดข้อมูล std_logic_vector และประกาศใช้ Signal

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity COUNTER10UP is
7     Port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR(3 downto 0));
9 end COUNTER10UP;
10
11 architecture Behavioral of COUNTER10UP is
12 begin
13     process(C,CLR)
14     variable QT : STD_LOGIC_VECTOR (3 downto 0);
15     begin
16         if CLR='1' then QT := "0000";
17         elsif C'event and C='1' then
18             if QT >= 9 then QT := "0000"; เมื่อค่า QT >= 9 (หรือ Q >= "1001")  
แล้วมีสัญญาณนาฬิกา C ทริกด้วยขอบ  
ขาขึ้นจะทำให้ QT มีค่าเป็น "0000"
19             else QT := QT + 1;
20             end if;
21         end if;
22         Q <= QT;
23     end process;
24 end Behavioral;

```

(b) โค้ดควบจรนับ 10 แบบนับขึ้นที่มีอادر์พุต Q เป็นชนิดข้อมูล std_logic_vector และประกาศใช้ Variable

รูปที่ 4.159 โค้ดควบจรนับ 10 แบบนับขึ้น

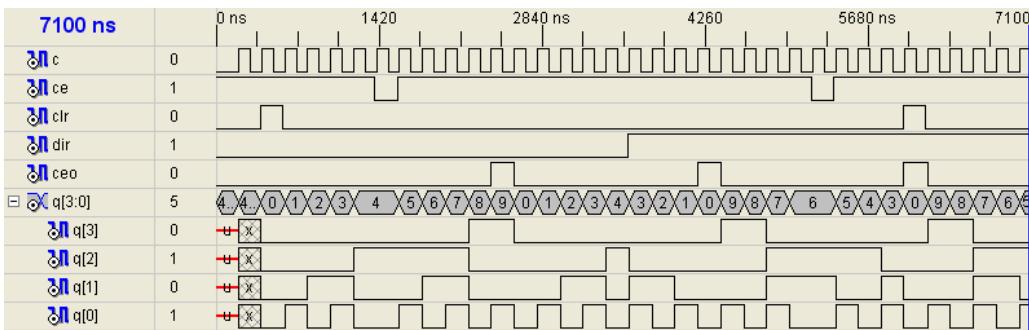
ตัวอย่างโค้ดควบจรนับ 10 แบบนับขึ้น-นับลงแบบคาสเคด (Cascade) แสดงรูปที่ 4.160(a) และมีผล Simulation แสดงดังรูปที่ 4.160(b) วงจรจะนับเมื่อ Clock Enable input (CE) = '1' และหยุดนับเมื่อ CE = '0' วงจนับขึ้นเมื่อ Direction (DIR) = '0' และเมื่อนับถึง 9 แล้ว Clock Enable output (CEO) = '1' วงจนับลงเมื่อ DIR = '1' และเมื่อนับถึง 0 และ CEO = '1'

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity C10UPDN_CE is
7     Port ( C,CE,CLR,DIR : in STD_LOGIC;
8             CEO : out STD_LOGIC;           --Clock enable output
9             Q : out STD_LOGIC_VECTOR (3 downto 0));
10 end C10UPDN_CE;
11
12 architecture Behavioral of C10UPDN_CE is
13     signal QT : STD_LOGIC_VECTOR (3 downto 0);
14 begin
15 process(C,CLR)
16     begin
17         if CLR='1' then QT <= "0000";
18         elsif (C'event and C='1') then
19             if CE='1' then
20                 if DIR='0' then          -- Count up
21                     if QT>=9 then QT <= "0000";
22                     else QT <= QT + 1;
23                     end if;
24                 else                      -- Count down
25                     if (QT=0 or QT>9) then QT <= "1001";
26                     else QT <= QT - 1;
27                     end if;
28                 end if;
29             end if;
30         end if;
31     end process;
32     Q <= QT;                         -- Counter output
33     CEO <= CE when (DIR ='0' and QT=9) else --CEO-count up
34         CE when (DIR ='1' and QT=0) else --CEO-count down
35         '0';
36 end Behavioral;

```

(a) ໄດ້គົດຂອງງາງຈຽນ 10 ແບບນັບຂຶ້ນ-ນັບลงທີ່ມີຂາ CE ແລະ CEO (ງາງຈຽນນີ້ມີ 3 ຄຳສັ່ງຄອນເກອຮີເຣນຕີ)



(b) ຜຸດໍາລອງການທຳມານຂອງງາງຈຽນ 10 ແບບນັບຂຶ້ນ-ນັບลงທີ່ມີຂາ CE ແລະ CEO

ຮູບປີ 4.160 ວັງຈຽນ 10 ແບບນັບຂຶ້ນ-ນັບลงທີ່ມີຂາ CE ແລະ CEO

ໃນການອີເວັກນາຮັກສາມາຮັດເປີຍນໂດ້ຄວງຈຽນນັບອື່ນໆ ໄດ້ເຊັ່ນກັນ ໂດຍຮູບປີ 4.161 ເປັນໂດ້ຄວງຈຽນ 24 (ນັບ 0-23) ທີ່ແສດງພລເປັນຮ້າສ BCD 2 ລັກ ຜຶ່ງວັງຈຽນຈະເປັນການນຳເອງວັງຈຽນ 100 (ນັບ 0-99) ມາດັດແປລັດເປັນວັງຈຽນ 24 ໂດຍໃຫ້ຫຼັກການ Synchronous reset ເຊັ່ນເດືອກນັບທີ່ໄດ້ອື່ນຍາຍໄປແລ້ວໃນຫຼຸດ 4.4.2 ໂດຍຮາຈະໃໝ່ 23 ໄປເຮັດວຽກຈຽນ ກລ່າວເມື່ອນັບຄົງ 23 ແລ້ວສໍານິ່ງ Clock ລູກຄົດໄປເຫົ້າໄປທົກ ທຳໄໝວງຈຽນມີຄ່າເປັນ 00 ວັງຈຽນຈະນັບເມື່ອ CE = '1' ແລະ ຈາກຮູບປີ 4.161 ດ້ວຍການກັບໄປໄວ ໂດຍຈຳກວງຈຽນ 24 (0-23) ເປັນວັງຈຽນ 60 (ນັບ 0-59) ກີ່ໃຫ້ແກ້ໄຂເຄີຍໄວໃນບຣທັດທີ່ 19 ໂດຍແກ້ຈາກ 23 ເປັນ 59

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity C24D_UP is
7     Port ( C,CE,CLR : in STD_LOGIC;
8             Q0,Q1 : out STD_LOGIC_VECTOR (3 downto 0));
9 end C24D_UP;
10
11 architecture Behavioral of C24D_UP is
12     signal Q0T,Q1T : STD_LOGIC_VECTOR (3 downto 0);
13 begin
14 process(C,CLR)
15 begin
16     if CLR='1' then Q1T <= "0000"; Q0T <= "0000";--Clear
17     elsif (C'event and C='1') then
18         if CE='1' then                                --Clock enable
19             if (Q1T>=2 and Q0T>=3) then          --Synchronous reset:N>=23
20                 Q1T <= "0000";
21                 Q0T <= "0000";
22             else
23                 if (Q0T>=9) then                  --Counter:N=0-99
24                     Q0T <= "0000";
25                     if (Q1T>=9) then Q1T <= "0000";
26                     else Q1T <= Q1T + 1;
27                 end if;
28                 else Q0T <= Q0T + 1;
29             end if;
30         end if;
31     end if;
32 end process;
33     Q0 <= Q0T;  Q1 <= Q1T;
34 end Behavioral;
35

```

รูปที่ 4.161 โค้ดของวงจรนับ 24 (0-23) แบบนับขึ้นที่แสดงผลเป็นรหัส BCD หรือเลขฐานสิบ 2 หลัก

ตัวอย่างโค้ด VDHL ที่มักจะนำไปใช้บ่อย คือ โค้ดของวงจรกดรหัส BCD เป็นเซเว่นเซกเมนต์ (BCD to 7-Segment Decoder) !! แสดงดังรูปที่ 4.162

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DECODER_7SEGMENT is
6     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
7             Y : out STD_LOGIC_VECTOR (6 downto 0));
8 end DECODER_7SEGMENT;
9
10 architecture Behavioral of DECODER_7SEGMENT is
11 begin
12     gfedcba
13     Y <= "1101111" when A = "1001" else --9
14     "1111111" when A = "1000" else --8      Y(0)=a
15     "0000111" when A = "0111" else --7      ---
16     "1111101" when A = "0110" else --6 Y(5)=f | Y(1)=b
17     "1101101" when A = "0101" else --5      --- Y(6)=g
18     "1100110" when A = "0100" else --4 Y(4)=e | Y(2)=c
19     "1001111" when A = "0011" else --3      ---
20     "1011011" when A = "0010" else --2      Y(3)=d
21     "0000110" when A = "0001" else --1
22     "0111111" ;                      --0,A,b,C,d,E,F
23 end Behavioral;

```

รูปที่ 4.162 โค้ดของวงจรกดรหัส BCD เป็นเซเว่นเซกเมนต์

จากหลักการที่อธิบายมาทั้งหมดนี้จะทำให้ผู้อ่านสามารถนำความรู้เบื้องต้นนี้ไปใช้ได้ สามารถแก้ไขโค้ดที่ดาวน์โหลดทางอินเตอร์เน็ตหรือโค้ดที่ได้จากหนังสือโดยทั่วไป เพื่อนำไปประยุกต์ใช้งาน โดยสร้าง Symbols ตามที่ได้อธิบายไปแล้ว

4.7.1 การออกแบบวงจรนับบีน-ลง 4 หลัก (0-9999) ที่แสดงทางเทเวนเซกเมนต์

อุปกรณ์ที่ควรมีเพื่อทดสอบคือบอร์ด CPLD Explorer XC9572XL และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

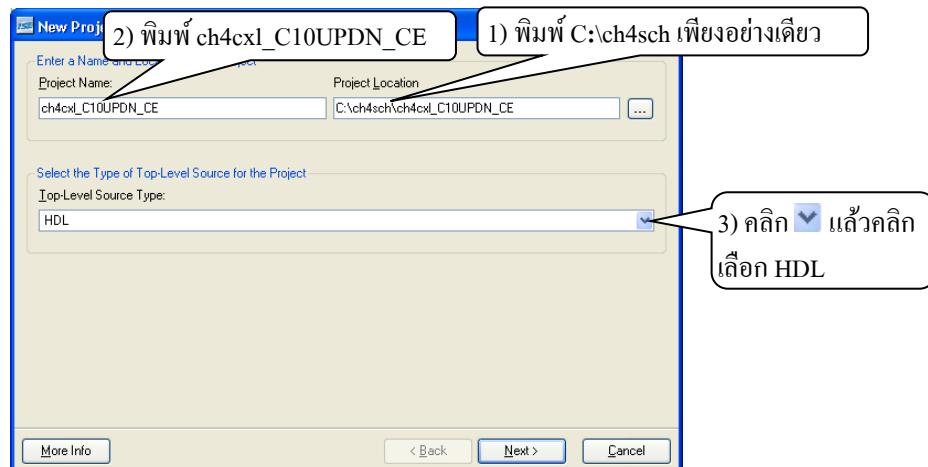
1. สร้างวงจรนับบีน-ลง 4 หลัก (0-9999) ที่แสดงทางเทเวนเซกเมนต์ด้วย CPLD

ในการทดลองนี้ให้ออกแบบวงจรนับบีน-ลง 4 หลัก (0-9999) แทนวงจรออกแบบวงจรนับบีน-ลง 4 หลัก (0-FFFF) ที่ได้ทำการทดลองไปแล้วในการทดลองที่ 4.3.7 แต่จะสร้าง Symbols จากโค้ด VHDL ของวงจรนับ 10 แบบบีนบีนลงในรูปที่ 4.89(a) เพื่อนำไปสร้างเป็นวงจรนับ 0-9999 และจากโค้ด VHDL ของจรดอครหัส BCD เป็นเทเวนเซกเมนต์ในรูปที่ 4.162

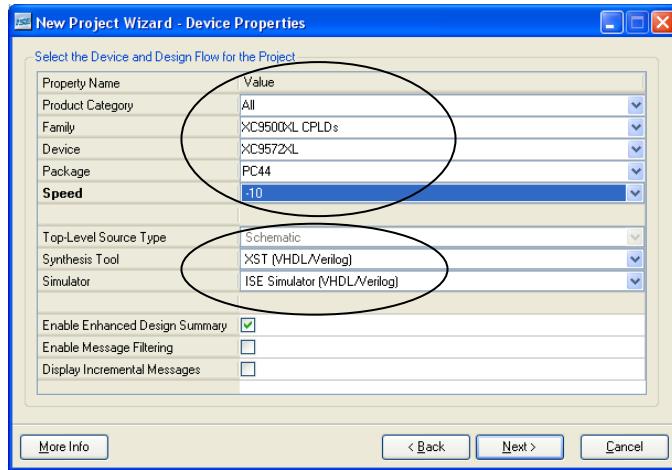
a) ขั้นตอนการสร้าง Symbols จากโค้ด VHDL วงจรนับ 10 แบบบีนบีนลงในรูปที่ 4.160(a) กี คือ ขั้นตอนการออกแบบวงจร (Design entry) นั่นเอง ซึ่งจะแตกต่างจากการออกแบบด้วยวิธีวิชาพัฒนาระบบเดียวกันนี้ มีรายละเอียดดังนี้

a.1) ขั้นตอนการสร้างไฟล์ Symbols เริ่มที่จอกомพิวเตอร์ คลิกปุ่ม Start -> Programs -> Xilinx ISE 8.1i -> Project Navigator หรือดับเบิลคลิกที่ แล้วจะได้หน้าต่าง Xilinx-ISE (ถ้ามีหน้าต่าง) Tip of the Day ช่องขึ้นมาให้คลิก OK คลิกที่ (File -> New Project) แล้วจะได้หน้าต่าง (หรือ) Dialog box (New Project Wizard-Create New Project) จากนั้นสร้างโปรเจกต์ไฟล์ Project File ใหม่โดยพิมพ์ชื่อ (ch4sch) มี Folder นือยู่แล้ว ลงในช่องว่างของ (Project Location) ก่อน แล้วพิมพ์ชื่อ ch4cxl_C10UPDN_CE ลงในช่องว่างของ Project Name คลิกที่ Top-Level Source Type เป็น HDL ดังรูปที่ 4.163 คลิก Next แล้วจะได้หน้าต่าง New Project Wizard-Device Properties แล้วคลิกเลือกดังรูปที่ 4.164

หมายเหตุ การตั้งชื่อจะใช้กฎเกณฑ์ที่กำหนดในภาษา VHDL ในข้อ 2.4 คือ จะต้องเป็นตัวอักษร A-Z, a-z และตัวอักขระภาษาไทย A-Z, a-z, 0-9 หรือ Underscores (_) แต่ห้ามจบด้วย (_) และห้ามเริ่มช่องว่างระหว่างตัวอักษร

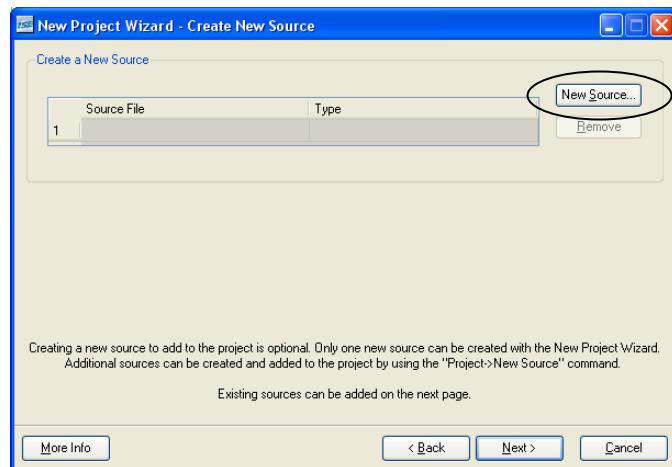


รูปที่ 4.163 หน้าต่าง New Project Wizard-Create New Project

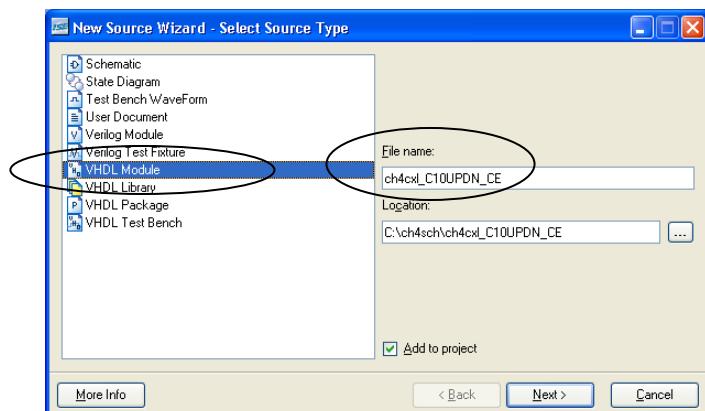


รูปที่ 4.164 หน้าต่าง New Project Wizard–Device Properties ในกรณีที่ใช้บอร์ด CPLD Explorer XC9572XL

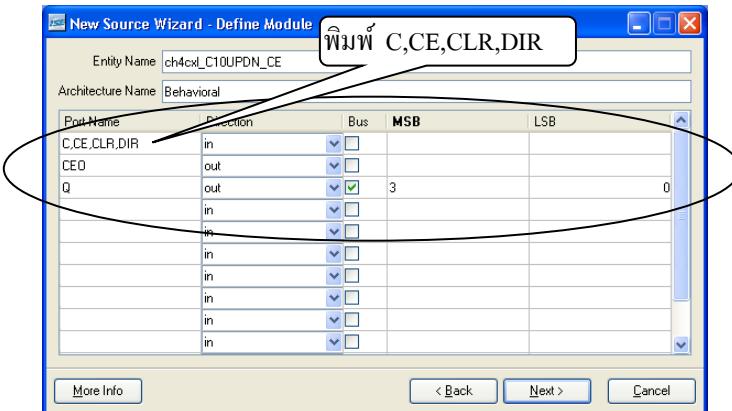
a.2) คลิก Next ในรูปที่ 4.164 แล้วจะได้หน้าต่างดังรูปที่ 4.165 คลิกปุ่ม New Source แล้วจะได้หน้าต่างถัดไป งานนี้พิมพ์ชื่อ Source File ชื่อ ch4cxl_C10UPDN_CE ลงในช่อง File Name แล้วคลิกที่ VHDL Module ดังรูปที่ 4.166 คลิก Next จะได้หน้าต่างถัดไป เมื่อกำหนดอินพุต A,B และเอาต์พุต C เรียบร้อยแล้วจะได้ดังรูปที่ 4.167 คลิก Next 1 ครั้ง คลิก Finish 1 ครั้ง แล้วคลิก Next อีก 1 ครั้งก็จะได้หน้าต่าง New Project Wizard–Add Existing Source ดังรูปที่ 4.168



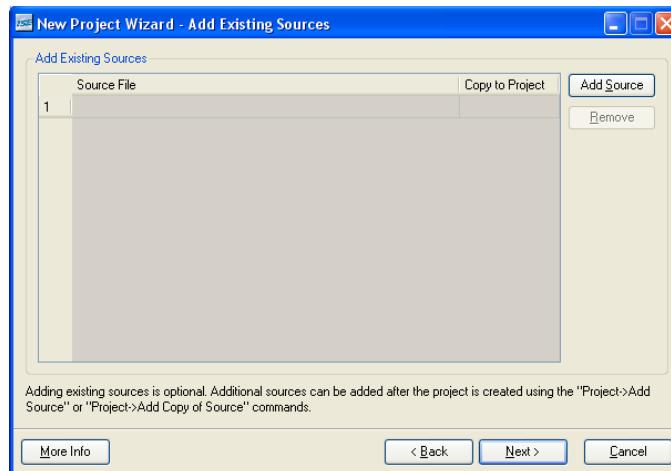
รูปที่ 4.165 หน้าต่าง New Project Wizard–Create New Source



รูปที่ 4.166 หน้าต่าง New Source Wizard–Select Source Type

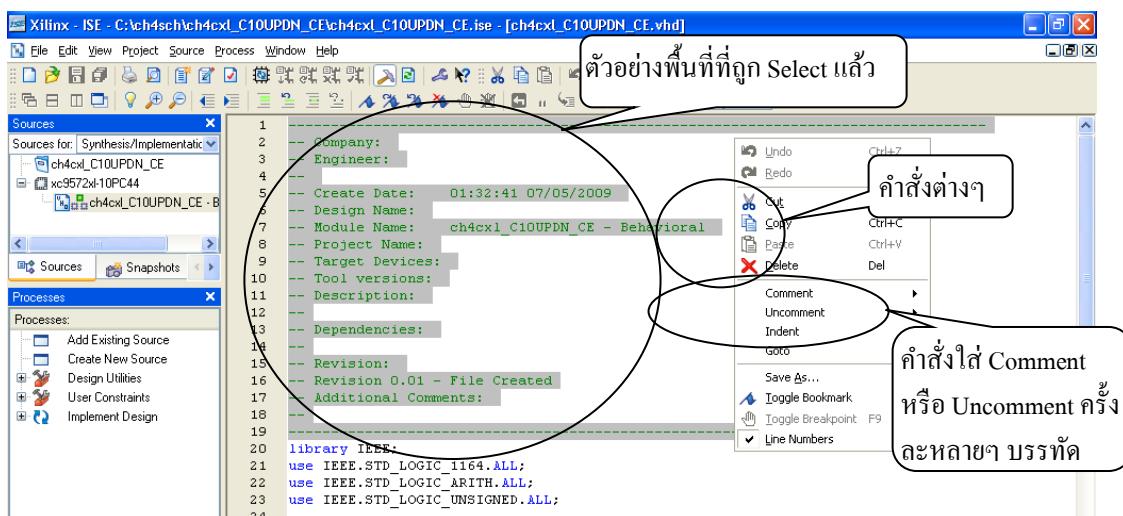


รูปที่ 4.167 หน้าต่าง New Source Wizard–Define Module ซึ่งกรณีเป็นบัสให้คลิก “√” ที่คอลัมน์ Bus ด้วย



รูปที่ 4.168 หน้าต่าง New Project Wizard-Add Existing Source

a.3) ในรูปที่ 4.168 คลิก Next 1 ครั้ง คลิก Finish อีก 1 ครั้งแล้วจะได้หน้าต่าง Xilinx–ISE สำหรับเขียนโค้ด VHDL (Text Editor) จากนั้นให้คลิกซ้ายค้างไว้แล้วลากเมาส์เพื่อเลือกพื้นที่ แล้วคลิกขวาเมาส์ที่หน้าต่างหลัก (Workspace) ก็จะพบว่ามีคำสั่งต่างๆ ที่เราคุ้นเคยกันดีอยู่แล้วดังในรูปที่ 4.169



รูปที่ 4.169 หน้าต่างหลัก (Workspace)

a.4) ทำการกัดลอกโค้ดในรูปที่ 4.160(a) เสร็จแล้วจะได้ดังรูปที่ 4.170 ซึ่งจะเห็นว่าซอฟต์แวร์ทูลเป็นคนจัดการให้เรางานไปทั้งหมด โดยที่เราจะเขียนเฉพาะส่วนของโค้ดที่อยู่ภายใต้ชื่อ模块ที่กำหนดใน Architecture เท่านั้น จากนั้นคลิก บันทึกไฟล์

```

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

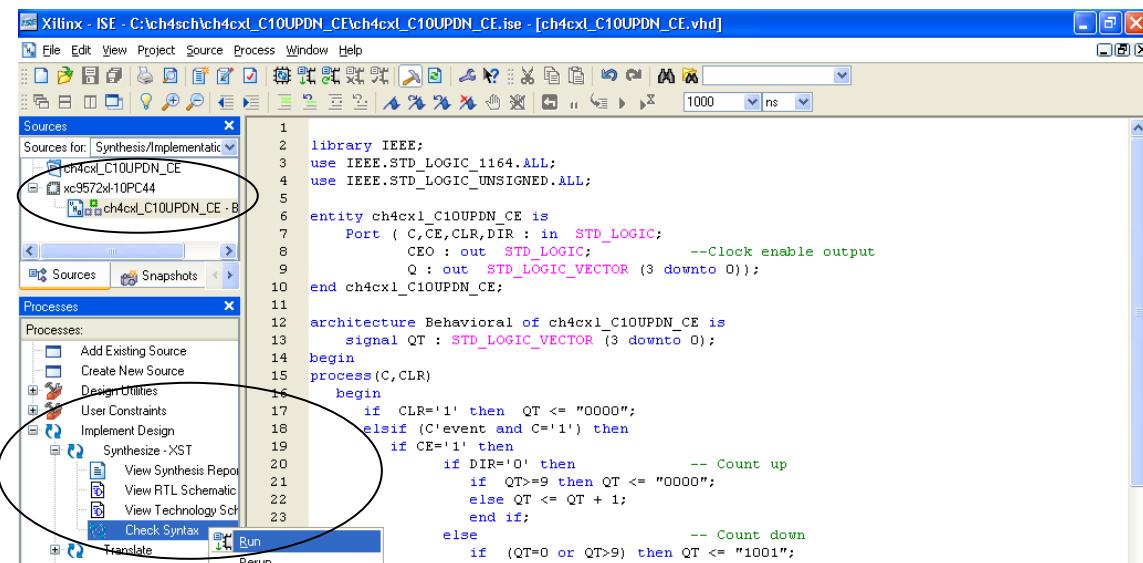
entity ch4cxl_C10UPDN_CE is
    Port ( C,CE,CLR,DIR : in STD_LOGIC;
           CEO : out STD_LOGIC; --Clock enable output
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end ch4cxl_C10UPDN_CE;

architecture Behavioral of ch4cxl_C10UPDN_CE is
    signal QT : STD_LOGIC_VECTOR (3 downto 0);
begin
process(C,CLR)
begin
    if CLR='1' then QT <= "0000";
    elsif (C'event and C='1') then
        if CE='1' then
            if DIR='0' then -- Count up
                if QT>=9 then QT <= "0000";
                else QT <= QT + 1;
                end if;
            else -- Count down
                if (QT=0 or QT>9) then QT <= "1001";
                else QT <= QT - 1;
                end if;
            end if;
        end if;
    end if;
end process;
Q <= QT; -- Counter output
CEO <= CE when (DIR ='0' and QT=9) else --CEO-count up
      CE when (DIR ='1' and QT=0) else --CEO-count down
      '0';
end Behavioral;

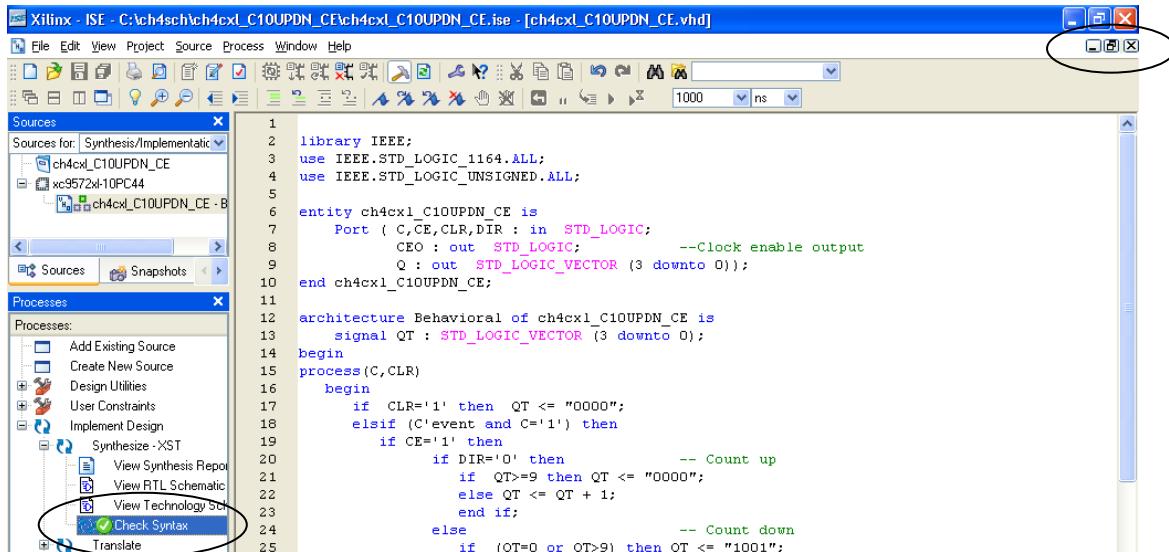
```

รูปที่ 4.170 โค้ด VHDL ของวงจรนับ 10 แบบนับขึ้น-นับลงที่เขียนเสร็จเรียบร้อยแล้ว

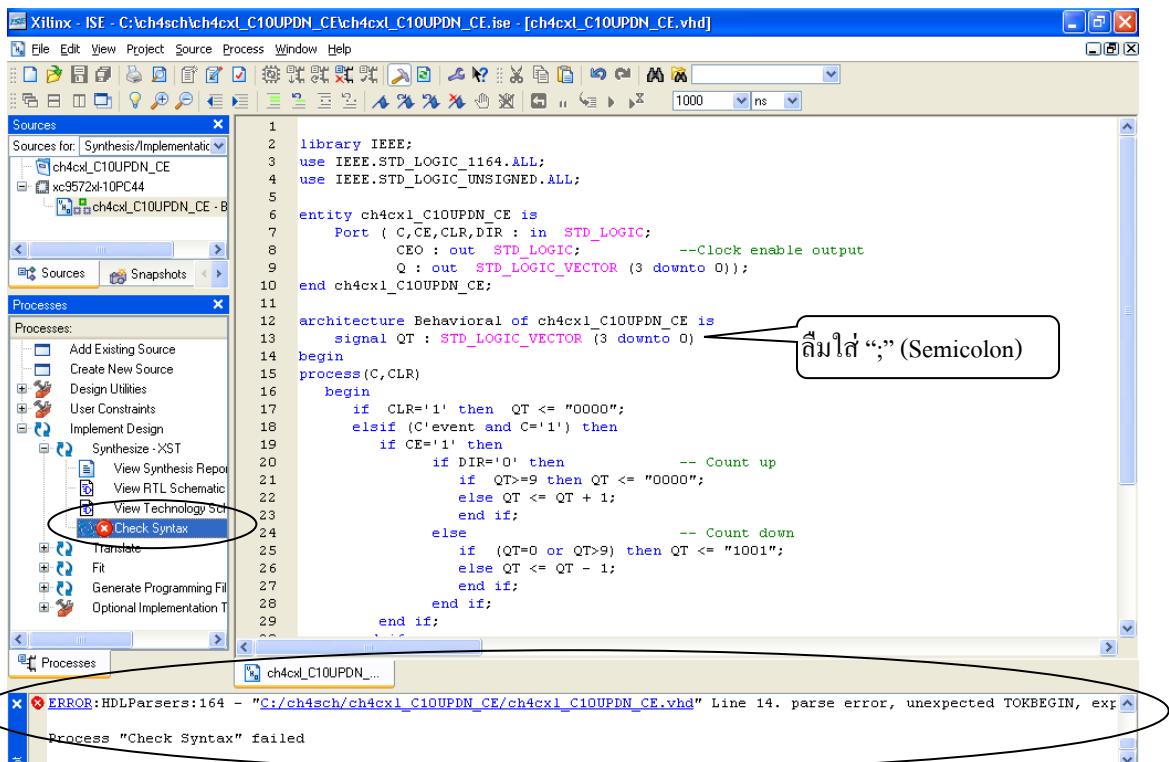
a.5 การตรวจสอบความถูกต้อง (Syntax) ของโค้ด คลิกที่ชื่อไฟล์ ch4cxl_C10UPDN_CE ในหน้าต่าง Source แล้วคลิกที่ “+” หน้า Implement design และหน้า Synthesize-XST ในหน้าต่าง Process จนเป็น “-” คลิกขวาที่ Check Syntax แล้วคลิก Run ดังรูปที่ 4.171 ถ้าได้ ✓ หน้า Check Syntax ดังในรูปที่ 4.172 ถือว่า Check Syntax ผ่านและถือว่าโค้ดถูก Compiled เรียบร้อยแล้ว แต่ถ้ามีข้อผิดพลาดจะได้ ✗ หน้า Check Syntax เช่น ในรูปที่ 4.173 บอกว่า Line 14 ซึ่งเกิดจากผลสืบเนื่องจากการลืมใส่ “;” (Semicolon) ตอนจบบรรทัดที่ 13 เมื่อแก้ไขแล้วให้คลิก □ บันทึกไฟล์และตรวจสอบความถูกต้องซ้ำอีกครั้ง ถ้าไม่มีข้อผิดพลาดก็ถือว่าขั้นตอน Design entry นี้แล้วเสร็จ คลิก X (สีดำ) เพื่อปิด Text Editor และกลับไปที่หน้าต่าง Xilinx-ISE ดังรูปที่ 4.174 เพื่อทำการ Simulation (ซึ่งขั้นตอนจะเหมือนเดิมทุกประการ) หรือคลิก X (สีแดง) เพื่อออกจาก ISE WebPACK 8.1i



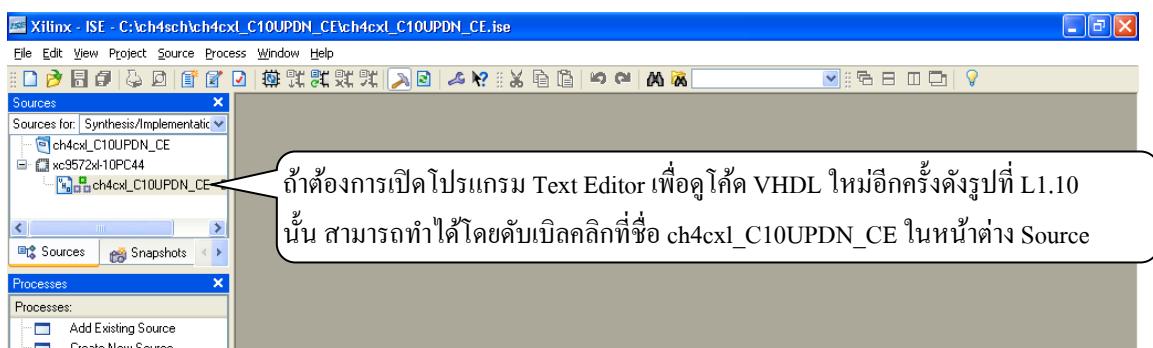
รูปที่ 4.171 การตรวจสอบความถูกต้อง



รูปที่ 4.172 หน้าต่าง Xilinx-ISE เมื่อ Check syntax ผ่าน



รูปที่ 4.173 การเขียนโค้ดที่มีข้อผิดพลาด



รูปที่ 4.174 หน้าต่าง Xilinx-ISE

b) สร้างไฟล์สำหรับทำ Symbols จากโค้ด VHDL วงจรดอครหัส BCD เป็นเซเวนเซกเมนต์ในรูปที่ 4.162 ไว้ใน Project Location ชื่อ ch4sch จากนั้นกำหนด Project Name และ Source File ชื่อ ch4cxl_DECODER_7SEGMENT เสิร์จแล้วจะได้ดังรูปที่ 4.175

```

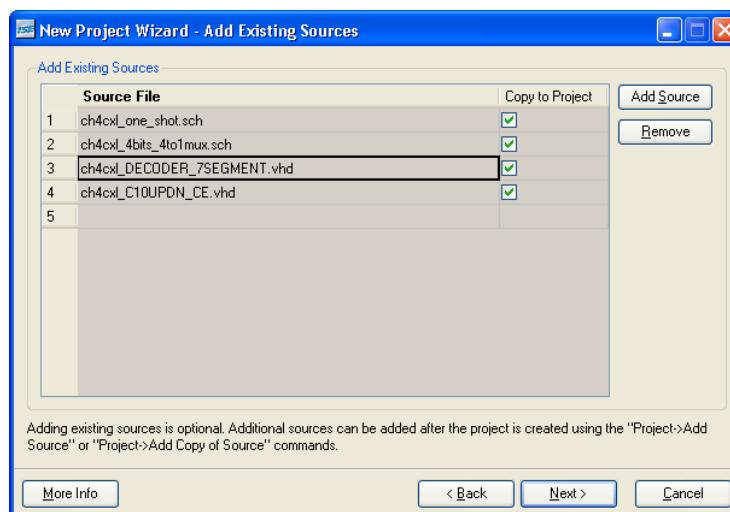
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ch4cxl_DECODER_7SEGMENT is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (6 downto 0));
end ch4cxl_DECODER_7SEGMENT;

architecture Behavioral of ch4cxl_DECODER_7SEGMENT is
begin
    -- gfedcba
    Y <= "1101111" when A = "1001" else --9
        "1111111" when A = "1000" else --8      Y(0)=a
        "0000111" when A = "0111" else --7      ---
        "1111101" when A = "0110" else --6 Y(5)=f | Y(1)=b
        "1101101" when A = "0101" else --5      --- Y(6)=g
        "1100110" when A = "0100" else --4 Y(4)=e | Y(2)=c
        "1001111" when A = "0011" else --3      ---
        "1011011" when A = "0010" else --2      Y(3)=d
        "0000110" when A = "0001" else --1      ---
        "0111111" ;                           --0,A,b,C,d,E,F
end Behavioral;

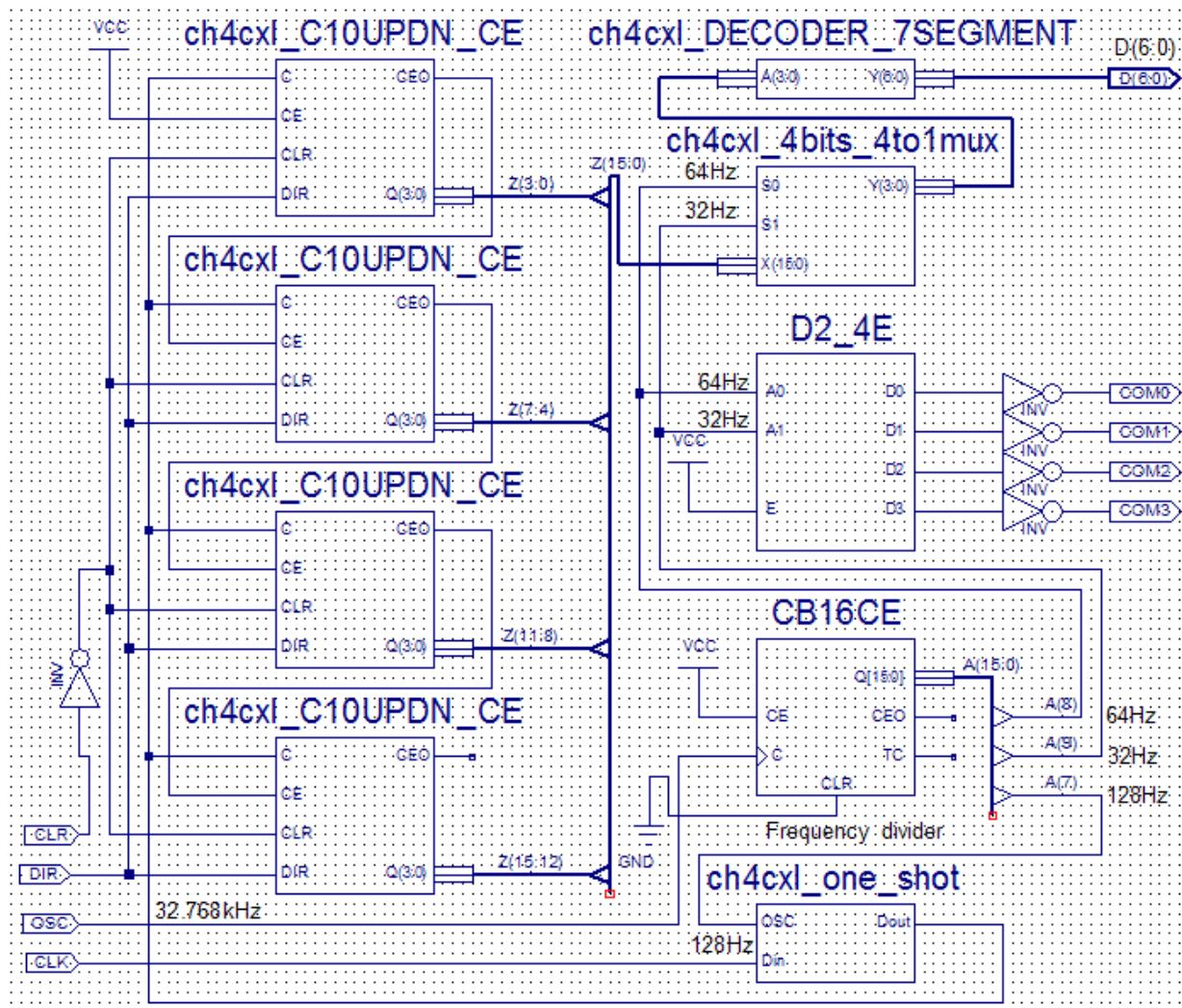
```

รูปที่ 4.175 โค้ด VHDL วงจรดอครหัส BCD เป็นเซเวนเซกเมนต์ที่เขียนเสร็จเรียบร้อยแล้ว

c) สร้างวงจรนับขีน-ลงแบบ 4 หลัก (0-9999) ไว้ใน Project Location ชื่อ ch4sch จากนั้นกำหนด Project Name และ Source File ชื่อ ex4_7_1cxl แล้วนำไฟล์วงจรโน�สเตรเบิล ชื่อ ch4cxl_one_shot (ใช้แทน ch4cxl_debounce) ไฟล์วงจรมัดดิเพล็กเซอร์ ชื่อ ch4cxl_4bits_4to1mux ไฟล์วงจรดอครหัส BCD เป็นเซเวนเซกเมนต์ ชื่อ ch4cxl_DECODER_7SEGMENT และไฟล์วงจรนับ 10 แบบนับขีน-ลง ชื่อ ch4cxl_C10UPDN_CE มาทำ Symbols จากนั้นทำการ Add Source เพื่อให้เพิ่มไฟล์ที่จะนำไปสร้างเป็น Symbols ซึ่งขั้นตอน Add Source และขั้นตอนการสร้าง Symbols นั้นไม่ว่าจะเป็นไฟล์ (Source file) ที่ได้จากวิธี Schematic หรือไฟล์ VHDL ก็จะใช้วิธีการเหมือนกันทุกประการ โดยที่หน้าต่าง New Project Wizard-Add Existing Source หลังจากเลือกไฟล์ที่จะนำไปสร้างเป็น Symbols จะครบแล้วแสดงดังรูปที่ 4.176 หลังจากสร้าง Symbols เรียบร้อยแล้วให้วาดผังวงจรแสดงดังรูปที่ 4.177 ซึ่งวงจรนับนี้จะเคลียร์เอาต์พุตเป็น “0000” เมื่อ CLR = ‘0’ โดยจะนับขึ้นเมื่อ DIR = ‘0’ และถ้า DIR = ‘1’ จะเป็นการนับลง ขอให้สังเกตว่าความถี่ที่ป้อนให้วงจรโน�สเตรเบิล = 128 Hz (จากวงจร Frequency divider) สำหรับความถี่ที่ใช้สแกนแต่ละหลัก = 32 Hz (อย่างน้อย 30Hz เพื่อไม่ให้เกิดการกระพริบ)



รูปที่ 4.176 หน้าต่าง New Project Wizard-Add Existing Source หลังจากเลือกไฟล์ที่จะนำไปสร้างเป็น Symbols ครบแล้ว



ຮູບທີ 4.177 ວົງຈານບັນຫຼິນ-ລົງແບບ 4 ລັກ (0-9999)

ການກໍາທຳນຳສັງເກດ ໃຊ້ OSC = 32.768 kHz, PB1-PB3 ເປັນອິນພຸດ ແລະ ໃຊ້ DIGIT1-DIGIT4 ແລ້ວກໍາສະເກນ

CLK = PB1= INPUT=p39	D(0) = a= OUTPUT= p27	D(4) = e= OUTPUT= p22	COM0 = DIGIT1= OUTPUT= p34
CLR = PB2= INPUT=p40	D(1) = b= OUTPUT= p26	D(5) = f= OUTPUT= p20	COM1 = DIGIT2= OUTPUT= p33
OSC= OSC= INPUT= p5	D(2) = c= OUTPUT= p25	D(6) = g= OUTPUT= p18	COM2 = DIGIT3= OUTPUT= p29
DIR = PB3= INPUT= p42	D(3) = d= OUTPUT= p24		COM3 = DIGIT4= OUTPUT= p28

ໂດຍພິມພື້ນໃນ Edit Constraints (Text) ສຽງດັ່ງນີ້

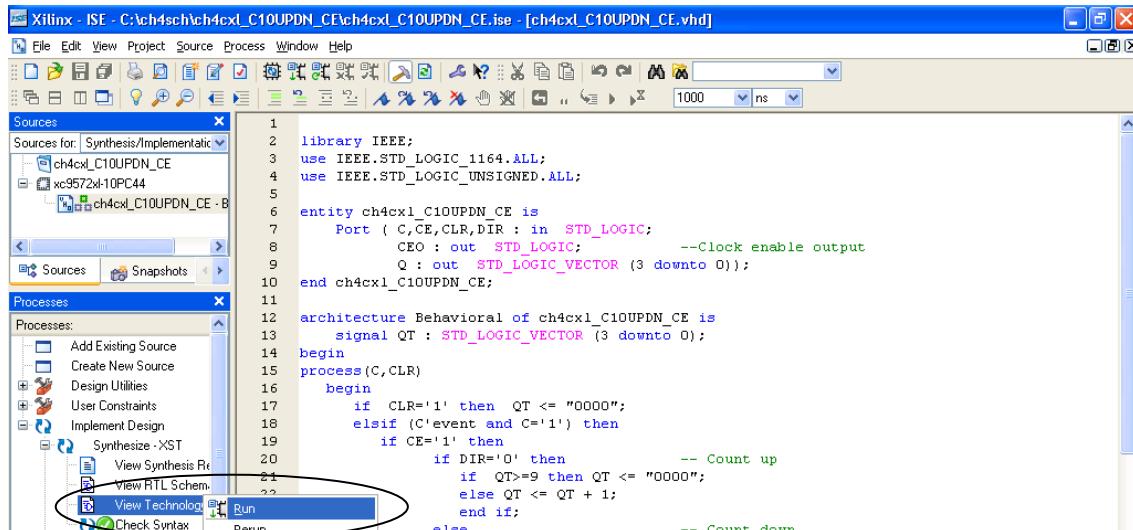
```

NET "CLK" LOC = "p39" ;
NET "CLR" LOC = "p40" ;
NET "COM0" LOC = "p34" | SLEW = SLOW ;
NET "COM1" LOC = "p33" | SLEW = SLOW ;
NET "COM2" LOC = "p29" | SLEW = SLOW ;
NET "COM3" LOC = "p28" | SLEW = SLOW ;
NET "D<0>" LOC = "p27" | SLEW = SLOW ;
NET "D<1>" LOC = "p26" | SLEW = SLOW ;
NET "D<2>" LOC = "p25" | SLEW = SLOW ;
NET "D<3>" LOC = "p24" | SLEW = SLOW ;
NET "D<4>" LOC = "p22" | SLEW = SLOW ;
NET "D<5>" LOC = "p20" | SLEW = SLOW ;
NET "D<6>" LOC = "p18" | SLEW = SLOW ;
NET "DIR" LOC = "p42" ;
NET "OSC" LOC = "p5" ;

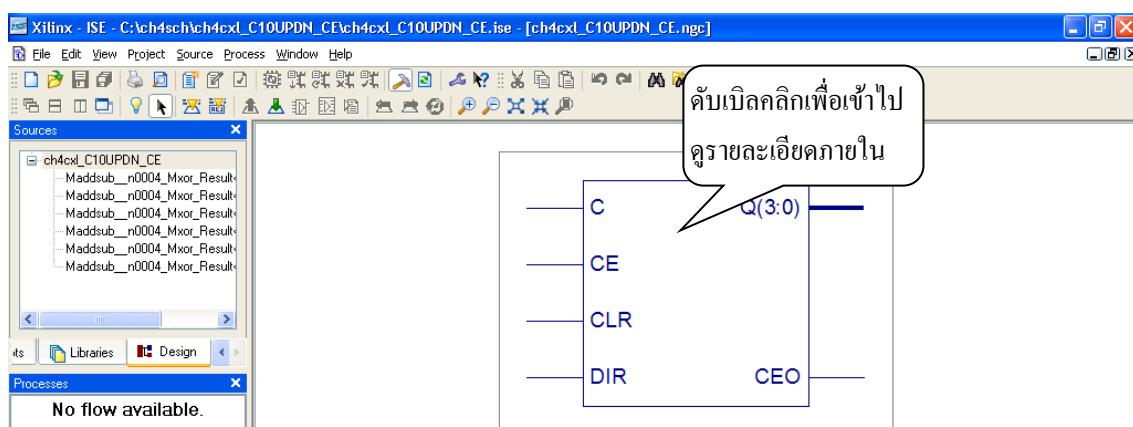
```

หลังจากโปรแกรม CPLD แล้วให้ ON Slide SW1 (PB3) และกดปุ่ม PB1 ไปเรื่อยๆ ลับกันการกดปุ่ม PB1 ค้างไว้แล้ว ให้คุณที่ LED3-LED4 และที่ DIGIT1-DIGIT4 ว่าแต่ละเซกเมนต์ติดสว่างเป็นตามทฤษฎีหรือไม่ จากนั้นให้กด PB2 แล้วให้คุณที่ DIGIT1-DIGIT4 อิกครึ่ง เสร็จแล้วให้ OFF Slide SW1 (PB3) และทำการทดลองซ้ำอิกครึ่ง ทำการบันทึกผลการทดลอง

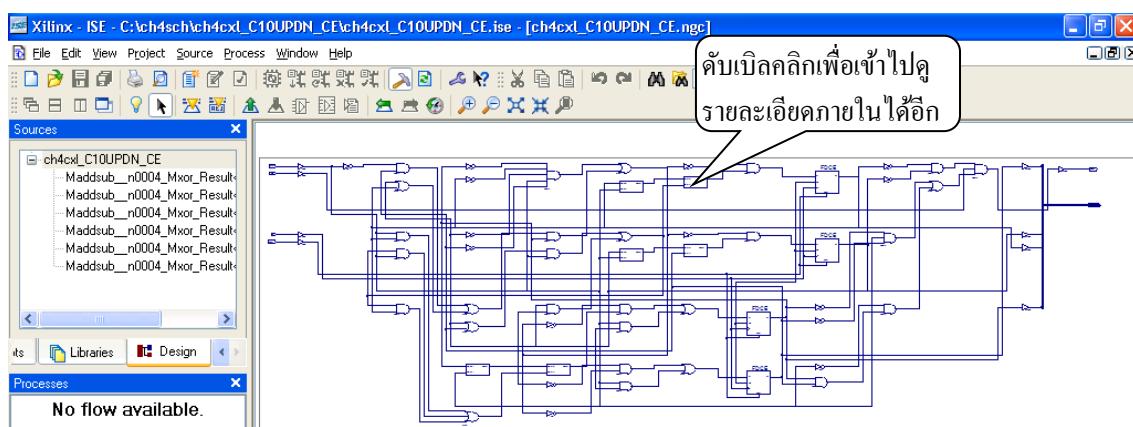
หมายเหตุ ถ้าต้องการดู Schematic หรือผังวงจรของโค้ดวงจรนับ 10 ในรูปที่ 4.170 ที่สังเคราะห์วงจรแล้ว ให้คลิกขวาที่ View Technology Shematic และคลิก Run ดังรูปที่ 4.178 และจะได้รูปที่ 4.179 และเมื่อบันเบิกคลิก จะได้ผังวงจรดังรูปที่ 4.180



รูปที่ 4.178 ขั้นตอนท่า View Technology Shematic



รูปที่ 4.179 View Technology Shematic ของวงจรนับ 10 แบบนับขึ้น-นับลง

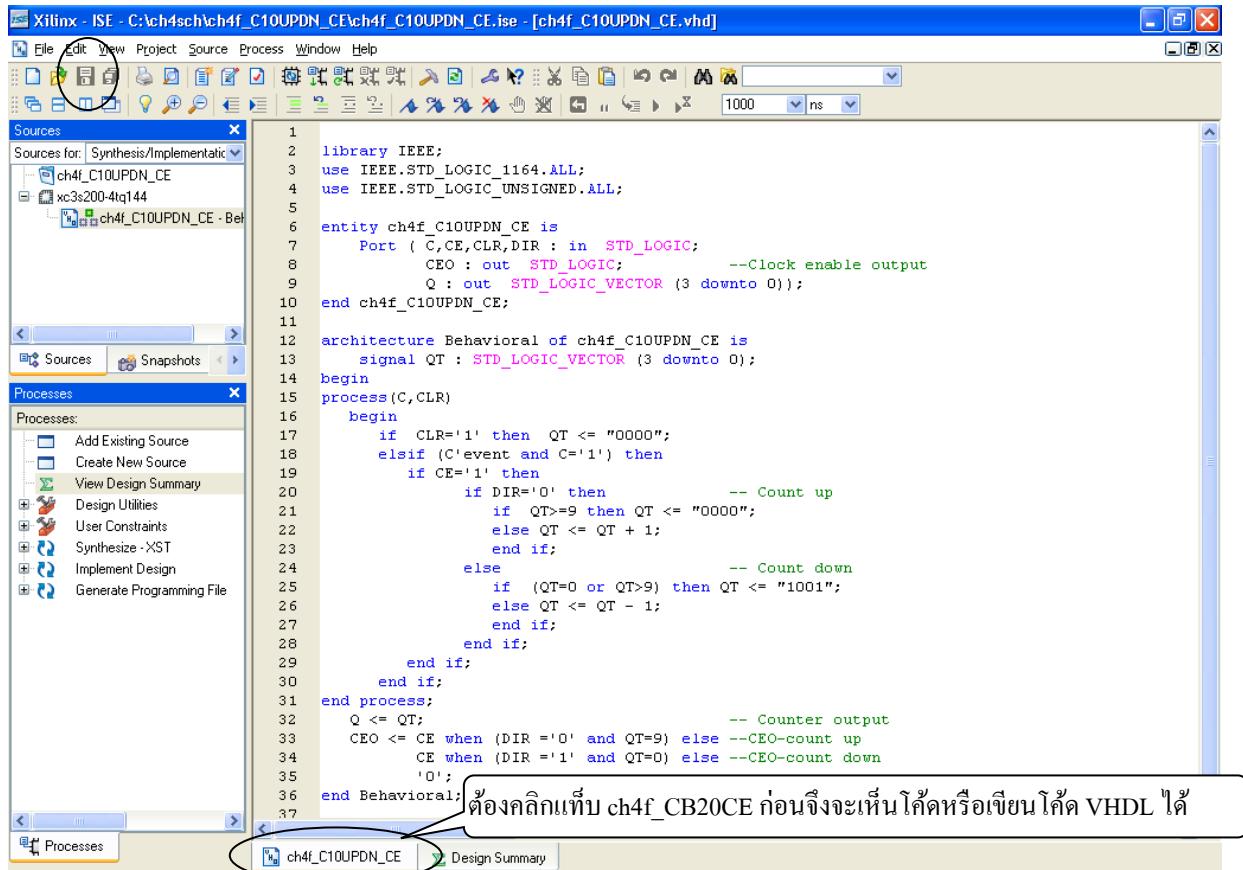


รูปที่ 4.180 รายละเอียดภายใน View Technology Shematic ของวงจรนับ 10 แบบนับขึ้น-นับลงที่เขียนด้วยโค้ด VHDL

2. สร้างวงจรนับขึ้น-ลง 4 หลัก (0-9999) ที่แสดงทางเขเวนเซกเมนต์ด้วย FPGA

ในการทดลองนี้ให้ออกแบบวงจรนับขึ้น-ลง 4 หลัก (0-9999) เหมือนกับการทดลองโดยใช้ CPLD ในข้อ 1 ทุกประการ ซึ่งใช้แทนวงจรนับ (0xFFFF) ใน การทดลองที่ 4.3.7 แต่จะสร้าง Symbols จากโค้ด VHDL วงจรนับ 10 แบบนับขึ้น-นับลงในรูปที่ 4.89(a) เพื่อนำไปสร้างเป็นวงจรนับ 0-9999 และจากโค้ด VHDL วงจรอุดรหัส BCD เป็นเขเวนเซกเมนต์ในรูปที่ 4.91

a) ขั้นตอนการสร้าง Symbols จากโค้ด VHDL วงจรนับ 10 แบบนับขึ้น-นับลงในรูปที่ 4.160(a) ซึ่งก็คือ ขั้นตอนการออกแบบวงจร (Design entry) นั่นเอง ซึ่งจะเหมือนกับการออกแบบโดย CPLD ในข้อ 1 ทุกประการ โดยสร้างไฟล์ใน Project Location ชื่อ ch4sch แล้วกำหนด Project Name และ Source File ชื่อ ch4f_C10UPDN_CE ก่อนเขียนโค้ด VHDL ด้วยคลิกแท็บ ch4f_CB20CE ที่ด้านล่างของหน้าต่างหลักก่อนจึงจะเห็นหน้าต่างหลักที่ใช้เขียนโค้ด VHDL เสร็จแล้วจะได้ดังรูปที่ 4.181



รูปที่ 4.181 โค้ด VHDL วงจรนับ 10 แบบนับขึ้น-นับลงที่เขียนเสร็จเรียบร้อยแล้ว

b) สร้างไฟล์สำหรับทำ Symbols จากโค้ด VHDL วงจรอุดรหัส BCD เป็นเขเวนเซกเมนต์ในรูปที่ 4.162 ไว้ใน Project Location ชื่อ ch4sch จากนั้นกำหนด Project Name และ Source File ชื่อ ch4f_DECODR_7SEGMENT เสร็จแล้วจะได้ดังรูปที่ 4.182

c) สร้างไฟล์สำหรับทำ Symbols จากโค้ด VHDL วงจรนับ 20 บิตแบบไบนารี โดยใช้ความรู้เบื้องต้นที่อธิบายไปแล้วในการเขียนโค้ด VHDL เพื่อนำไปสร้างวงจร Frequency divider โดยเขียนโค้ดไว้ใน Project Location ชื่อ ch4sch จากนั้นกำหนด Project Name และ Source File ชื่อ ch4f_CB20CE เสร็จแล้วจะได้ดังรูปที่ 4.183

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ch4f_DECODER_7SEGMENT is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (6 downto 0));
end ch4f_DECODER_7SEGMENT;

architecture Behavioral of ch4f_DECODER_7SEGMENT is
begin
    -- gfedcba
    Y <= "1101111" when A = "0001" else --9
        "1111111" when A = "1000" else --8      Y(0)=a
        "0000111" when A = "0111" else --7      ---
        "1111101" when A = "0110" else --6 Y(5)=f | Y(1)=b
        "1101101" when A = "0101" else --5      --- Y(6)=g
        "1100110" when A = "0100" else --4 Y(4)=e | Y(2)=c
        "1001111" when A = "0011" else --3      ---
        "1011011" when A = "0010" else --2      Y(3)=d
        "0000110" when A = "0001" else --1      ---
        "0111111" ;                            --0,A,b,C,d,E,F
end Behavioral;

```

รูปที่ 4.182 โค้ด VHDL วงจรดอครหัส BCD เป็นเซเวนเซกเมนต์ที่เขียนเสร็จเรียบร้อยแล้ว

```

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

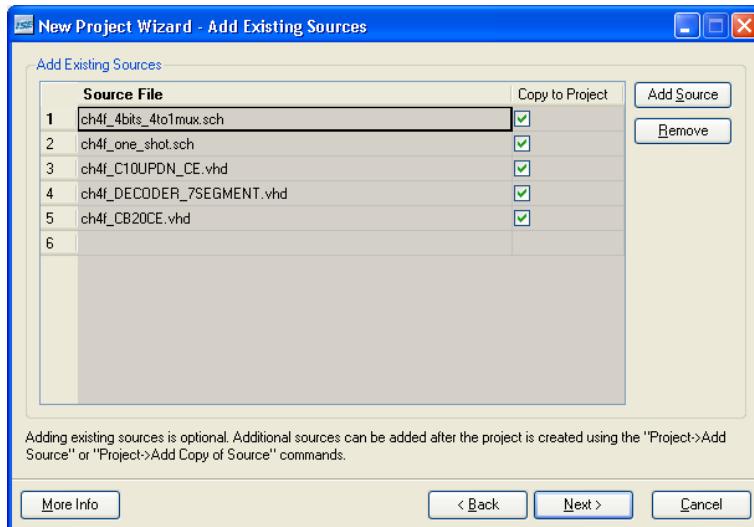
entity ch4f_CB20CE is
    Port ( C,CE,CLR : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (19 downto 0));
end ch4f_CB20CE;

architecture Behavioral of ch4f_CB20CE is
    signal QT : STD_LOGIC_VECTOR (19 downto 0);
begin
    process(C,CLR)
    begin
        if      CLR = '1' then QT <= (others => '0');
        elsif C'event and C= '1' then
            if CE = '1' then QT <= QT + 1;
            end if;
        end if;
    end process;
    Q <= QT;
end Behavioral;

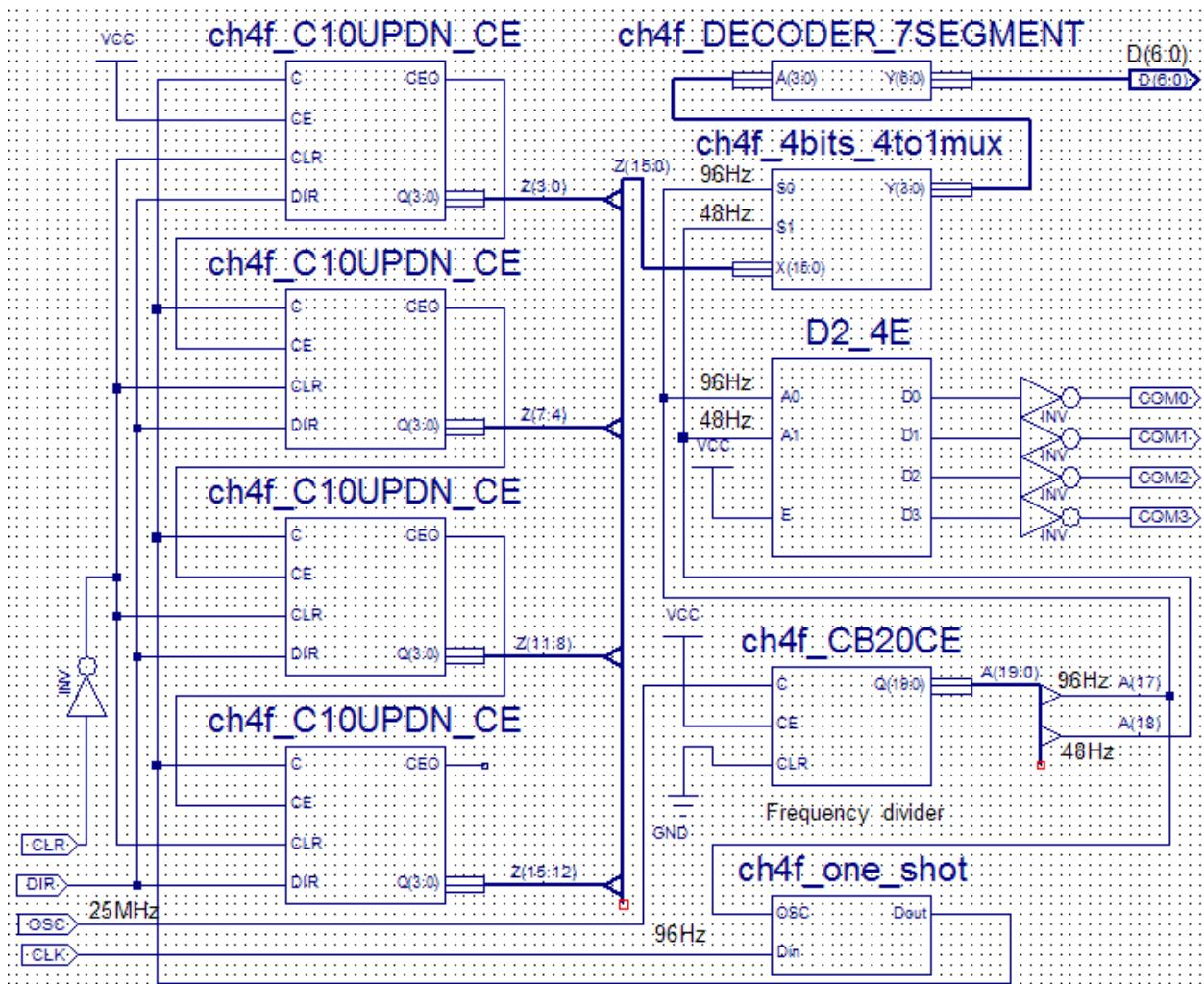
```

รูปที่ 4.183 โค้ด VHDL วงจรนับ 20 บิตแบบใบหนารีที่เขียนเสร็จเรียบร้อยแล้ว

d) สร้างวงจรนับขึ้น-ลงแบบ 4 หลัก (0-9999) ไว้ใน Project Location ชื่อ ch4sch จากนั้นกำหนด Project Name และ Source File ชื่อ ex4_7_1f และนำไฟล์วงจรโนโนสเตเบิล ชื่อ ch4f_one_shot (ใช้แทน ch4f_debounce) ไฟล์วงจรแมตติเพล็กเซอร์ ชื่อ ch4f_4bits_4to1mux ไฟล์วงจรดอครหัส BCD เป็นเซเวนเซกเมนต์ ชื่อ ch4f_DECODR_7SEGMENT ไฟล์วงจรนับ 10 แบบ นับขึ้น-นับลง ชื่อ ch4f_C10UPDN_CE และไฟล์วงจรนับ 20 บิตแบบใบหนารี ชื่อ ch4f_CB20CE มาทำ Symbols จากนั้นทำการ Add Source เพื่อใช้เพิ่มไฟล์ที่จะนำไปสร้างเป็น Symbols ซึ่งขั้นตอน Add Source และสร้าง Symbols นี้ไม่ว่าจะเป็นไฟล์ (Source file) ที่ได้จากการดาวน์โหลด (Schematic) หรือไฟล์ VHDL ก็จะใช้วิธีการเหมือนกันทุกประการ โดยที่หน้าต่าง New Project Wizard-Add Existing Source หลังจากเลือกไฟล์ที่จะนำไปสร้างเป็น Symbols แล้วจะได้ดังรูปที่ 4.184 หลังจากสร้าง Symbols เรียบร้อยแล้วให้วาดผังวงจรดังรูปที่ 4.185 ซึ่งวงจนับนี้จะเก็บเอาต์พุต “0000” เมื่อ CLR = ‘0’ โดยจะนับขึ้นเมื่อ DIR = ‘0’ และถ้า DIR = ‘1’ จะเป็นการนับลง ขอให้สังเกตว่าเราใช้วงจรนับ 20 บิตแบบใบหนารีเพื่อทำเป็นวงจร Frequency divider สำหรับในการสร้างความถี่ประมาณ = 96 Hz (25 Mhz/2¹⁸) ป้อนให้วงจรโนโนสเตเบิล และความถี่ที่ใช้ในการสแกนแต่ละหลัก = 48 Hz(25Mhz/2¹⁹) และ = 96 Hz ซึ่งปกติความถี่สแกนต้องน้อย 30 Hz เพื่อไม่ให้เห็นการกระพริบ



รูปที่ 4.184 หน้าต่าง New Project Wizard-Add Existing Source หลังจากเลือกไฟล์ที่จะนำไปสร้างเป็น Symbolsแล้ว



รูปที่ 4.185 วงจรนับขึ้น-ลงแบบ 4 หลัก (0-9999)

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC= 25 MHz, ปุ่มกด PB1-PB2 และ Dip SW1 เป็นอินพุต และใช้ DIGIT1-DIGIT4 แสดงผลด้วยการสแกน ก้าวคืบ

CLK = PB1 = INPUT = p44	D(0) = a = OUTPUT = p40	COM0 = DIGIT1 = OUTPUT = p31
CLR = PB2 = INPUT = p46	D(1) = b = OUTPUT = p35	COM1 = DIGIT2 = OUTPUT = p33
OSC = OSC = INPUT = p127	D(2) = c = OUTPUT = p32	COM2 = DIGIT2 = OUTPUT = p36
DIR = Dip SW1 = INPUT = p52	D(3) = d = OUTPUT = p30	COM3 = DIGIT2 = OUTPUT = p41
	D(4) = e = OUTPUT = p27	
	D(5) = f = OUTPUT = p25	
	D(6) = g = OUTPUT = p23	

โดยพิมพ์ใน Edit Constraints (Text) สรุปดังนี้

```

NET "CLK" LOC = "p44" | IOSTANDARD = LVCMOS33 ;
NET "CLR" LOC = "p46" | IOSTANDARD = LVCMOS33 ;
NET "COM0" LOC = "p31" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "COM1" LOC = "p33" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "COM2" LOC = "p36" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "COM3" LOC = "p41" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<0>" LOC = "p40" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<1>" LOC = "p35" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<2>" LOC = "p32" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<3>" LOC = "p30" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<4>" LOC = "p27" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<5>" LOC = "p25" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "D<6>" LOC = "p23" | IOSTANDARD = LVCMOS33 | SLEW = SLOW ;
NET "DIR" LOC = "p52" | IOSTANDARD = LVCMOS33 ;
NET "OSC" LOC = "p127" | IOSTANDARD = LVCMOS33 ;

```

หลังจากโปรแกรมลง FPGA แล้วให้ทดลอง ON Dip SW1 แล้วกดปุ่ม PB1 ไปเรื่อยๆ ลักษณะการกดปุ่ม PB1 ค้างไว้ แล้วให้สังเกตคุณลักษณะที่ เช่นเวลาเชกเมนต์ DIGIT1-DIGIT4 ว่าแต่ละเชกเมนต์ให้ล้อจิกເອົາຕີພຸດເປັນໄປຕາມທຸນຢູ່ໃຫ້ມາ ຈາກນີ້ໃຫ້ กด PB2 แล้วสังเกตเวลาเชกเมนต์ DIGIT1-DIGIT4 ອີກຮັງ ເສື່ອງແລ້ວໃຫ້ OFF Dip SW1 แล้วทำการทดสอบຫຼຳອີກຮັງ ทำการບັນທຶກผลการทดลอง

สรุป

วงจรແລດ໌ມີພື້ນຖານນາຈາກການໃຊ້ແນນດີເກຫຍານຂອງເກມາຕ່ອໄຂວັນ ທຳໃຫ້ສາມາດຄອງຄ້າງສະຕານະຂອງສັນຍາມ ເຂົ້າທີ່ພຸດໄດ້ ແລະສາມາດຄ່ອຍໂດຍໄປສ້າງຟິລິປິຟລອປ່ຽນແບນຕ່າງໆ ອາທີເຫັນ ດີຟິລິປິຟລອປ່ຽນທີ່ເຂົ້າທີ່ພຸດຈະເປັນແປງຕາມອິນພຸດກີ ຕ່ອມື່ອມີສັນຍາມນາພິກາເຂົ້າມາ ຩີ້ອີ້ນຟິລິປິຟລອປ່ຽນທີ່ຈະສລັບສັນຍາມເຂົ້າທີ່ພຸດທຸກໆ ຮັງທີ່ມີສັນຍາມນາພິກາເຂົ້າມາ ຮົມທີ່ເຈັກຟິລິປິຟລອປ່ຽນທີ່ມີສະຕານະການທຳມາທີ່ໄມ່ປໍລິ່ນແປງ ເຊື້ອີເຊື້ອ ແລະທີ່ອັກເກີດ ຜົ່ງສາມາດລັນດຳໄປປະຍຸກຕີໃຫ້ສ້າງວົງຈົ່ງເຄວານເຊີຍແບນ ຕ່າງທີ່ມີການທຳມາສລັບສ້າງມາເຂົ້າມາ ວົງຈານນັບແບນອະຈິງ ໂຄຣນັດທີ່ໄມ່ຈຳເປັນຕ້ອງໃຫ້ຟິລິປິຟລອປ່ຽນທຸກໆ ຕ້າທຳມາພ້ອມໆ ກັນ ຕ່າງຈາກວົງຈານນັບແບນອະຈິງ ໂຄຣນັດທີ່ຟິລິປິຟລອປ່ຽນທຸກໆ ຕ້າຕ້ອງທຳມາທຸກໆ ຮັງທີ່ມີສັນຍາມນາພິກາເຂົ້າມາ ຜົ່ງສາມາດອອກແບນໄທ ເປັນໄດ້ທີ່ວົງຈານນັບ ວົງຈານນັບລົງ ແລະວົງຈານນັບນັບລົງ ສ່ວນການສ້າງນາພິກາດິຈິຕອລະເປັນການນໍາວົງຈານນັບແຕ່ລະຮູບແບນນາ ພສມຮ່ວມກັນເພື່ອໃຫ້ນັບຄ່າຮັບວິນາທີ່ນາທີ່ ແລະຂ້າວົງຕາມລຳດັບ

ການອອກແບນວົງຈົ່ງເຄວານເຊີຍລົມກັຈນີ້ອູ່ສອງຮູບແບນທີ່ນີ້ມີກັນຄືການໃຊ້ມັວງແລະເມີຍລື່ມໂມເຄລ ໂດຍມັວງໂມເຄລສັນຍາມເຂົ້າທີ່ພຸດຈະເຂົ້າທີ່ພຸດທຸກໆ ແລະສັນຍາມອິນພຸດ ສ່ວນໂປຣແກຣມກາຍາ VHDL ຈະເປັນຖຸລົກທີ່ຂ່າຍໃຫ້ການອອກແບນວົງຈົ່ງເດີຕອລ່າຍເຂົ້າ ໂດຍຈະສາມາດໃຊ້ກະບວນການເປີຍໂປຣແກຣມຄ້າຍໆ ກາຍຮະດັບສູງ ທ້ຳກັງໄປໃນການສ້າງວົງຈົ່ງແທນກາວດັບວິນາທີ່ນາທີ່ ຢື່ນຮູ້ໄດ້ຈ່າຍເຂົ້າ

คำถามท้ายบท

- ให้อธิบายการทำงานของวงจรอุดรหัสตัวแสดงผลเซเว่นเซกเมนต์แบบడิจิตอลร่วม (Hexadecimal digit display format) (คำตอบ จะทำงานตรงข้ามกับวงจรอุดรหัสตัวแสดงผลเซเว่นเซกเมนต์ที่ได้อธิบายในบทที่ 3 โดยวงจรนี้จะออกแบบเป็นวงจรอุดรหัสตัวแสดงผลเซเว่นเซกเมนต์แบบออโนดร่วมก่อน โดยเริ่มจากการนำอินพุต X(3:0) มาอุดรหัสเป็น 0-15 (D0-D15) โดยใช้งาน 4 to 16 Decoder จากนั้นให้ไปพิจารณาตัวเลข 0-F ว่าตัวเลขแต่ละตัวมีเซกเมนต์ใดบ้างที่ต้อง เช่น เซกเมนต์ a (แบบออโนดร่วม) ของเลข 1, 4, 6 และ d จะต้อง เป็นตัวนี้ ดังนั้น a (แบบออโนดร่วม) = D1 + D4 + D11 + D13 (โดยที่ + คือ OR) จากนั้นจึงนำเซกเมนต์ที่ได้มาไปผ่านอินเวอร์เตอร์ก็จะได้ a (แบบออโนดร่วม) ซึ่งในรูปที่ 4.55c จะใช้นอร์เกตแทนออร์เกตที่นำมาต่อ กับอินเวอร์เตอร์ สำหรับการออกแบบเซกเมนต์อื่นๆ ก็จะใช้หลักการเดียวกัน)
- ต้องสแกนความถี่ที่ขา A โอดร่วมของตัวแสดงผลเซเว่นเซกเมนต์แต่ละหลักประมาณเท่าใดจึงจะไม่เห็นการกระพริบ (คำตอบ ไม่น้อยกว่า 30 Hz โดยให้นักศึกษาทดลองเปลี่ยนความถี่ที่ป้อนเข้าหา S0 ของวงจรสแกน (Scan circuit) โดยในกรณีของ CPLD ให้แก้ไขวงจรในรูปที่ 4.56 ที่ละ 1 ค่าของความถี่จาก F(9) หรือ 32 Hz เป็น F(10) หรือ 16 Hz, F(11) หรือ 8 Hz และ F(8) หรือ 64 Hz ซึ่งนักศึกษาจะทราบได้เองว่าความถี่ต่ำสุดที่เริ่มไม่เห็นการกระพริบของตัวแสดงผลเซเว่นเซกเมนต์เป็นความถี่อย่างน้อยเท่าใด ส่วนในกรณีที่เป็น FPGA ให้ทดลองแก้ที่ละค่าตั้งแต่ A(17) หรือ 96 Hz ถึง Z(20) หรือ 12 Hz
- ต้องป้อนความถี่ที่สร้างจากวงจร Frequency divider (ในรูปที่ 4.50 หรือรูปที่ 4.51) ประมาณเท่าใดที่ขาเคลียร์ (CLR) เพื่อให้วงจร โมโนสเตเบิลที่ใช้งานได้ (คำตอบ อยู่ในช่วงประมาณ 2-4 Hz โดยให้นักศึกษาทดลองเปลี่ยนความถี่ที่ป้อนที่ขาเคลียร์ (CLR) ของวงจร โมโนสเตเบิล โดยในกรณีของ CPLD ให้แก้ไขวงจรในรูปที่ 4.50 ที่ละ 1 ค่าของความถี่ที่ออกจากบัส A คือ A(13) หรือ 2 Hz เป็น A(11) หรือ 8 Hz, A(12) หรือ 4 Hz และ A(14) หรือ 1 Hz โดยให้ทดลองกดปุ่ม PB1 ไปซัก 4-5 ครั้งทุกๆ ค่าความถี่ตั้งแต่ 1-8 Hz ซึ่งนักศึกษาจะทราบได้เองว่าความถี่ช่วงใดที่เหมาะสม ส่วนในกรณีที่เป็น FPGA ให้แก้ไขวงจรในรูปที่ 4.51 ที่ละ 1 ค่าของความถี่ที่ออกจากบัส Z คือ Z(20) ถึง Z(23) หรือ หรือประมาณ 0.75 Hz ถึง 6 Hz
- ให้นำ Symbols ชื่อ cb4ce ที่อยู่ในโปรแกรมวาดผังวงจร (Schematic editor หรือ ECS) ไปสร้างเป็นวงจรนับขึ้นแบบซิงโครนัส 4 บิต (0-15) และทดลองดูว่าให้ผลตามทฤษฎีหรือไม่
- ให้นำ Symbols ชื่อ cb4cled ที่อยู่ในโปรแกรมวาดผังวงจร (Schematic editor หรือ ECS) มาสร้างเป็นวงจรนับ ขึ้น-ลง แบบซิงโครนัส 4 บิต (0-15) และทดลองดูว่าให้ผลตามทฤษฎีหรือไม่
- ถ้าต้องการแปลง Jenkeflip flop ไปให้กลายเป็นคีฟลิปฟล็อกและทีฟลิปฟล็อกต้องต่อวงจรเพิ่มเติมอย่างไรบ้าง
- จงออกแบบวงจรนับแบบซิงโครนัส โดยให้นับค่าต่างๆ ดังนี้ 2, 3, 4, 5, 6, 7 แล้ววนไป 2 ใหม่
- จงออกแบบวงจรนับแบบอะซิงโครนัส Mod 5
- จงออกแบบวงจรนับขึ้นบันลุงแบบซิงโครนัสที่มีค่าการนับอยู่ระหว่าง 1 – 6 แต่เพียงเท่านั้น
- จงออกแบบวงจรนาฬิกาดิจิตอลที่สามารถแสดง ชั่วโมง นาที วินาที ได้ และกำหนดเวลาดังกล่าวได้ด้วย
- จงออกแบบวงจรารความถี่ 500
- จงเขียนโปรแกรมภาษา VHDL เพื่อสร้างวงจร Jenkeflip flop 。
- จงเขียนโปรแกรมภาษา VHDL เพื่อสร้างชิฟфаร์จิสเตอร์บนาค 8 บิต ที่มีการทำงานแบบ SIPO

บทที่ 5 การลดรูปสมการบูลีนด้วยวิธีการของ Quine McCluskey

กล่าวนำ

ในกระบวนการออกแบบวงจรดิจิตอลโดยทั่วไป จะเริ่มจากการสร้างตารางความจริง เก็บสมการบูลีน และลดรูปด้วยวิธีการของพีชคณิตบูลีน หรือคอนอร์เมฟ ซึ่งเป็นวิธีที่นิยมกันโดยทั่วไป แต่นอกเหนือจากทั้ง 2 วิธีที่กล่าวมา ยังมีวิธีการลดรูปอีกวิธีหนึ่งที่เป็นที่นิยมใช้กันคือวิธีการของ Quine McCluskey ซึ่งมีวิธีการและกระบวนการที่ซับซ้อนมากขึ้น ไม่จำเป็นต้องอาศัยประสบการณ์หรือความสามารถส่วนบุคคลของผู้ออกแบบมากนัก ทำให้ผู้ออกแบบที่มีประสบการณ์อยู่ก็สามารถออกแบบวงจรที่มีขนาดที่เหมาะสมที่สุดได้หากทำการและวิธีการที่ได้วางไว้ อีกทั้งวิธีการของ Quine McCluskey ยังเหมาะสมกับวงจรดิจิตอลที่มีเข้าท์พุทธหลายๆ ตัว เนื่องจากสามารถรวมพจน์บางพจน์เพื่อใช้ร่วมกันได้ ทำให้สามารถประยุกต์วงจรได้มากกว่าเดิม

5.1 Quine McCluskey

วิธีการลดรูปสมการบูลีนโดยทั่วไปที่ใช้สมการบูลีนในการลดรูป มักจะมีปัญหาเกิดขึ้นที่ขาดประสบการณ์ในการทำโจทย์มาบ้างไม่มากพอ เนื่องจากมองไม่ออกว่าควรจะรวมพจน์ใดเข้าด้วยกันบ้าง ทำให้ไม่สามารถลดรูปสมการได้น้อยที่สุด เท่าที่ควรจะเป็น แต่วิธีการลดรูปของ Quine McCluskey สามารถให้อาท์พุทที่ใช้เกตน้อยที่สุดได้ โดยมีหลักการที่ซับซ้อน เป็นขั้นเป็นตอน ซึ่งมีพื้นฐานมาจากกระบวนการลดรูปด้วยสมการบูลีนที่ว่า ในพจน์สองพจน์ที่มีตัวแปรต่างกันเพียง 1 บิต สามารถตัดตัวแปรนั้นทิ้งได้ เช่น ($\overline{ABC} + ABC = AB$)

หลักการและขั้นตอนในการลดรูปด้วยวิธีการของ Quine McCluskey

1. จัดกลุ่มพจน์ตามจำนวนเลข 1 ในพจน์นั้นๆ
2. เปรียบเทียบสมาชิกของกลุ่มที่อยู่ติดกันทุกตัวเข้าด้วยกัน ไปเรื่อยๆ จนหมด
3. สร้างตาราง PI (Prime Implicant) และเลือก PI ที่น้อยที่สุด

1. การจัดกลุ่มพจน์ตามจำนวนเลข 1 ในพจน์นั้นๆ

โดยปกติแล้วสมการที่อยู่ในรูปของ Sum of Product (SOP) หรือ Minterm จะได้มาจากการที่เอาท์พุทเป็น 1 หลายๆ พจน์มา OR กัน ในขั้นตอนนี้ให้ทำการแบ่งกลุ่มพจน์ทึ้งหมดออกเป็นกลุ่มๆ โดยแยกตามจำนวนของบิตที่เป็น 1 ในแต่ละพจน์ เช่น กลุ่มที่มีเลข 1 เป็นจำนวน 0 ตัว, กลุ่มที่มีเลข 1 เป็นจำนวน 1 ตัว, กลุ่มที่มีเลข 1 เป็นจำนวน 2 ตัว 3 ตัว 4 ตัว ไปเรื่อยๆ ตัวอย่างเช่น หากมีโจทย์ว่า

$$F(A,B,C,D) = \sum m(2, 4, 6, 8, 9, 10, 12, 13, 15)$$

จะเขียนเป็นตารางความจริงได้ดังนี้

Truth table					
Minterms	Input				Output Y
	A	B	C	D	
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

สามารถแบ่งกลุ่มได้ดังนี้

กลุ่มที่ 1 กลุ่มที่มีจำนวนบิทที่เป็น 1 เพียงตัวเดียว คือ พจน์ 2(0010), 4(0100), 8(1000)

กลุ่มที่ 2 กลุ่มที่มีจำนวนบิทที่เป็น 1 จำนวน 2 ตัว คือ พจน์ 6(0110), 9(1001), 10(1010), 12(1100)

กลุ่มที่ 3 กลุ่มที่มีจำนวนบิทที่เป็น 1 จำนวน 3 ตัว คือ พจน์ 13(1101)

กลุ่มที่ 4 กลุ่มที่มีจำนวนบิทที่เป็น 1 จำนวน 4 ตัว คือ พจน์ 15(1111)

หลังจากนี้ให้นำพจน์ที่แบ่งกลุ่มเรียบร้อยแล้วมาเขียนเป็นตาราง List 1 ดังนี้

List1		
Minterms	ABCD	
2	0010	กลุ่มที่ 1 (มีเลข 1 จำนวน 1 ตัว)
4	0100	
8	1000	
6	0110	กลุ่มที่ 2 (มีเลข 1 จำนวน 2 ตัว)
9	1001	
10	1010	
12	1100	
13	1101	กลุ่มที่ 3 (มีเลข 1 จำนวน 3 ตัว)
15	1111	กลุ่มที่ 4 (มีเลข 1 จำนวน 4 ตัว)

2. เปรียบเทียบสมาชิกของกลุ่มที่อยู่ติดกันทุกตัวเข้าด้วยกัน ไปเรื่อยๆ จนหมด

2.1 หลังจากสร้างตาราง List 1 เรียบร้อยแล้วให้ทำการเปรียบเทียบข้อมูลในแต่ละกลุ่มเฉพาะกลุ่มที่อยู่ติดกัน เช่น กลุ่มที่ 1 กับกลุ่มที่ 2, กลุ่มที่ 2 กับ กลุ่มที่ 3 และ กลุ่มที่ 3 กับ กลุ่มที่ 4 โดยให้เปรียบเทียบสมาชิกทุกตัวในกลุ่มนั้นๆ กับสมาชิกทุกตัวของอีกกลุ่มหนึ่ง เช่นการเปรียบเทียบกลุ่มที่ 1 กับกลุ่มที่ 2 (กลุ่มที่ 1 มีพจน์ 2, 4, 8 และกลุ่มที่ 2 มีพจน์ 6, 9, 10, 12) จะมีการเปรียบเทียบทั้งหมด 12 ชุด คือ

- ชุด 1 - 4 เปรียบเทียบ 2 กับ 6, เปรียบเทียบ 2 กับ 9, เปรียบเทียบ 2 กับ 10 และ เปรียบเทียบ 2 กับ 12
- ชุด 5 - 8 เปรียบเทียบ 4 กับ 6, 4 กับ 9, 4 กับ 10 และ 4 กับ 12
- ชุด 9 - 12 เปรียบเทียบ 8 กับ 6, 8 กับ 9, 8 กับ 10 และ 8 กับ 12

เพื่อนำไปสร้างเป็นตาราง List 2

2.2 การเปรียบข้อมูลในแต่ละคู่นั้นให้ดูว่าข้อมูลทั้ง 2 ตัวต่างกันกี่บิท หากต่างกันเพียงแค่ 1 บิท ถือว่าจับคู่กันได้ เช่น 2 กับ 6 คือ 0010 กับ 0110 จะเห็นว่าข้อมูลต่างกันเพียงแค่บิทเดียวคือ บิทที่ 2 จากซ้าย แสดงว่า 2 กับ 6 จับคู่กันได้ ให้นำไปเก็บในตาราง List 2 เป็น 0-10 แต่หากเป็นการเปรียบกันระหว่าง 2 กับ 9 คือ 0010 กับ 1001 จะเห็นว่าข้อมูลต่างกัน 3 บิท คือ บิทแรก บิทที่ 3 และบิทที่ 4 จากซ้ายเมื่อ ถือว่าสองตัวนี้จับคู่กันไม่ได้ เนื่องจากเราต้องการเฉพาะที่ต่างกันเพียง 1 บิทเท่านั้น ดังนั้นจะได้ผลลัพธ์การเปรียบเทียบแต่ละกลุ่มดังนี้

กลุ่มที่ 1 เปรียบเทียบกับกลุ่มที่ 2 จะได้ผลลัพธ์ที่จับคู่ได้ดังนี้

2 กับ 6	0-10	ต่างกันเฉพาะบิตที่ 2 จากซ้าย
2 กับ 10	-010	ต่างกันเฉพาะบิตที่ 1 จากซ้าย
4 กับ 6	01-0	ต่างกันเฉพาะบิตที่ 3 จากซ้าย
4 กับ 12	-100	ต่างกันเฉพาะบิตที่ 1 จากซ้าย
8 กับ 9	100-	ต่างกันเฉพาะบิตที่ 4 จากซ้าย
8 กับ 10	10-0	ต่างกันเฉพาะบิตที่ 3 จากซ้าย
8 กับ 12	1-00	ต่างกันเฉพาะบิตที่ 2 จากซ้าย

กลุ่มที่ 2 เปรียบเทียบกับกลุ่มที่ 3 จะได้ผลลัพธ์ที่จับคู่ได้ดังนี้

9 กับ 13	1-01	ต่างกันเฉพาะบิตที่ 2 จากซ้าย
12 กับ 13	110-	ต่างกันเฉพาะบิตที่ 4 จากซ้าย

กลุ่มที่ 3 เปรียบเทียบกับกลุ่มที่ 4 จะได้ผลลัพธ์ที่จับคู่ได้ดังนี้

13 กับ 15	11-1	ต่างกันเฉพาะบิตที่ 3 จากซ้าย
-----------	------	------------------------------

List 1			List 2		
Minterms	ABCD		Minterms	ABCD	
2	0010		2,6	0-10	
4	0100		2,10	-010	
8	1000		4,6	01-0	
6	0110		4,12	-100	
9	1001		8,9	100-	
10	1010		8,10	10-0	
12	1100		8,12	1-00	
13	1101		9,13	1-01	
15	1111		12,13	110-	
			13,15	11-1	

นำผลการเปรียบเทียบทั้งหมดไปเขียนเป็นตาราง List 2 โดยแบ่งเป็นกลุ่มตามผลลัพธ์ที่เปรียบเทียบระหว่างกลุ่มในตาราง List 1 เหมือนเดิม คือ

ผลของการเปรียบเทียบระหว่างกลุ่มที่ 1 กับกลุ่มที่ 2 ของตาราง List 1 ซึ่งมีผลลัพธ์เป็น 2 กับ 6, 2 กับ 10, 4 กับ 6, 4 กับ 12, 8 กับ 9, 8 กับ 10 และ 8 กับ 12 ให้ถือเป็นกลุ่มที่ 1 ของตาราง List 2

ผลของการเปรียบเทียบระหว่างกลุ่มที่ 2 กับกลุ่มที่ 3 ของตาราง List 1 ซึ่งมีผลลัพธ์เป็น 9 กับ 13 และ 12 กับ 13 ให้ถือเป็นกลุ่มที่ 2 ของตาราง List 2

ผลของการเปรียบเทียบระหว่างกลุ่มที่ 3 กับกลุ่มที่ 4 ของตาราง List 1 ซึ่งมีผลลัพธ์เป็น 13 กับ 15 ให้ถือเป็นกลุ่มที่ 3 ของตาราง List 2

หลังจากได้ตาราง List 2 แล้วให้เปรียบเทียบระหว่างกลุ่มที่อยู่ติดกันดังเดิมเพื่อสร้างตาราง List 3 โดยใช้หลักการเดิมคือเปรียบเทียบเฉพาะตัวที่ต่างกันเพียง บิตเท่านั้น โดยหากเป็น – เมื่อมองกัน ให้ถือว่าข้อมูลเหมือนกัน เพราะจะนับผลลัพธ์ที่ได้ของตาราง List 3 ซึ่งเกิดจากการเปรียบเทียบของตาราง List 2 ในแต่ละกลุ่มจะได้เป็นคู่ของ 8, 8 กับคู่ของ 12, 13 เพียงชุดเดียวเท่านั้นที่จับคู่กัน ได้แก่ 8 กับ 9 เป็น 100- และคู่ของ 12 กับ 13 เป็น 110- ผลลัพธ์ที่ได้จะเป็น 8, 9, 12, 13 = 1-0- (ทั้ง 2 ชุด ต่างกันเพียงบิตที่ 2 จากซ้ายเท่านั้น)

List 1			List 2			List 3		
Minterms	ABCD		Minterms	ABCD		Minterms	ABCD	
2	0010		2,6	0-10		8,9,12,13	1-0-	
4	0100		2,10	-010				
8	1000		4,6	01-0				
6	0110		4,12	-100				
9	1001		8,9	100-				
10	1010		8,10	10-0				
12	1100		8,12	1-00				
13	1101		9,13	1-01				
15	1111		12,13	110-				
			13,15	11-1				

หลังจากนั้นนำผลลัพธ์ที่ได้ไปเขียนเป็นตาราง List 3 แต่ถ้าหากว่าในตาราง List 3 มีผลลัพธ์ที่ได้มากกว่า 1 กลุ่มก็ให้ทำการเปรียบเทียบสมาชิกในทุกตัวของทุกกลุ่มไปเรื่อยๆ เพื่อสร้างตาราง List 4, List 5, List 6 ต่อไป

3. สร้างตาราง PI (Prime Implicant) และเลือก PI ที่น้อยที่สุด

3.1 จากผลของการเปรียบเทียบในทุกๆ ชุดให้คูณกันในตาราง List 3 ดูด้วยสามารถลดแทนพจน์ได้ได้บ้าง เช่นในตัวอย่าง ตาราง List 3 มีพจน์ 8, 9, 12, 13 = 1-0- และคงว่าพจน์นี้สามารถลดแทนพจน์ของกลุ่มที่ 1 ในตาราง List 2 คือ พจน์ 8, 9 และพจน์ 8, 12

อีกทั้งสามารถลดแทนพจน์ของกลุ่มที่ 2 ในตาราง List 2 คือ พจน์ 9, 13 และพจน์ 12, 13 โดยให้ใส่เครื่องหมายถูกไว้ท้ายตาราง List 2 ในพจน์ดังกล่าว

รวมทั้งสามารถลดแทนพจน์ของกลุ่มที่ 1 ในตาราง List 1 คือ พจน์ 8

และสามารถลดแทนพจน์ของกลุ่มที่ 2 ในตาราง List 1 คือ พจน์ 9 กับ 12

และสามารถลดแทนพจน์ของกลุ่มที่ 3 ในตาราง List 1 คือ พจน์ 13

แล้วให้พจน์ 8, 9, 12, 13 ในตาราง List 3 เป็น PI₁

List 1			List 2			List 3		
Minterms	ABCD		Minterms	ABCD		Minterms	ABCD	
2	0010		2,6	0-10		8,9,12,13	1-0-	PI ₁
4	0100		2,10	-010				
8	1000	✓	4,6	01-0				
6	0110		4,12	-100				
9	1001	✓	8,9	100-	✓			
10	1010		8,10	10-0				
12	1100	✓	8,12	1-00	✓			
13	1101	✓	9,13	1-01	✓			
15	1111		12,13	110-	✓			
			13,15	11-1				

หลังจากได้ PI₁ และให้มองข้อมูลนี้ไปยังตาราง List 2 จะเห็นได้ว่า [พจน์ 2, 6], [พจน์ 2, 10], [พจน์ 4, 6], [พจน์ 4, 12], [พจน์ 8, 10] และ [พจน์ 13, 15] ซึ่งไม่มีเครื่องหมายถูกดังนั้นจึงให้พจน์ดังกล่าวเป็น PI ลำดับถัดๆ ไปดังนี้

2, 6 เป็น PI₂

2, 10 เป็น PI₃

4, 6 เป็น PI₄

4, 12 เป็น PI₅

8, 10 เป็น PI₆

13, 15 เป็น PI₇

List 1			List 2			List 3		
Minterms	ABCD		Minterms	ABCD		Minterms	ABCD	
2	0010	✓	2,6	0-10	PI ₂	8,9,12,13	1-0-	PI ₁
4	0100	✓	2,10	-010	PI ₃			
8	1000	✓	4,6	01-0	PI ₄			
6	0110	✓	4,12	-100	PI ₅			
9	1001	✓	8,9	100-	✓			
10	1010	✓	8,10	10-0	PI ₆			
12	1100	✓	8,12	1-00	✓			
13	1101	✓	9,13	1-01	✓			
15	1111	✓	12,13	110-	✓			
			13,15	11-1	PI ₇			

จากพจน์ 2, 6 เป็น PI_2 สามารถลดแทนพจน์ 2 และ 6 ในตาราง List 1 กลุ่มที่ 1 และ 2 ตามลำดับ ดังนั้นให้ใส่เครื่องหมายถูกไว้ท้ายพจน์ 2 และพจน์ 6 ในตาราง List 1 หลังจากนั้นให้พิจารณา PI_3, PI_4, PI_5 ไปเรื่อยๆ จะเห็นได้ว่าพจน์ในตาราง List 1 ทุกตัวสามารถลดแทนได้ด้วยพจน์ในตาราง List 2 จึงไม่มีพจน์ใดที่จะต้องถูกเลือกมาเป็น PI เพิ่มเติมอีก สุดท้ายจะได้ PI ทั้งหมด 7 ตัวคือ

PI_1 ทดแทนพจน์ 8, 9, 12, 13

PI_2 ทดแทนพจน์ 2, 6

PI_3 ทดแทนพจน์ 2, 10

PI_4 ทดแทนพจน์ 4, 6

PI_5 ทดแทนพจน์ 4, 12

PI_6 ทดแทนพจน์ 8, 10

PI_7 ทดแทนพจน์ 13, 15

แล้วนำ PI ทั้งหมดไปเขียนเป็นตารางโดยให้แนวนอนเป็น PI แนวตั้ง เป็นพจน์ของโจทย์และใส่เครื่องหมายกาหนาที่ในแนวนอนของ PI ที่ PI นั้นทดแทนพจน์นั้นๆ ได้ เช่น PI_1 ทดแทนพจน์ 8, 9, 12, 13 ให้ใส่กาหนาที่ในแนวนอนของ PI_1 ที่ตรงกับคอลัมน์ 8, 9, 12, 13 ส่วน PI_2 ทดแทนพจน์ 2, 6 ให้ใส่กาหนาที่ในแนวนอนของ PI_2 ที่ตรงกับคอลัมน์ 2 และ 6 แล้วทำไปให้ครบทุกๆ PI

	2	4	6	8	9	10	12	13	15
PI_1				X	X		X	X	
PI_2	X		X						
PI_3	X					X			
PI_4		X	X						
PI_5		X					X		
PI_6				X		X			
PI_7								X	X

3.2 การเลือก PI ไปใช้งานมีหลักการดังนี้

1. ให้เลือก PI รวมกันทั้งหมดน้อยที่สุด
2. PI ทุกตัวที่เลือกมาจะต้องครอบคลุมทุกๆ พจน์ของโจทย์ทั้งหมด

โดยปกติแล้วจะมี PI บางตัวที่จำเป็นจะต้องเลือกมาใช้งานซึ่งเป็น PI ที่ทดแทนพจน์บางพจน์ของโจทย์ โดยไม่มี PI ตัวอื่นสามารถแทนพจน์นั้นๆ ได้เลย ดูได้จากว่าในแนวตั้งใดๆ ก็ตามจะมีเครื่องหมายกาหนาทั้งหมดที่อยู่ในแนวนอน เช่น PI_1 และ PI_7 ซึ่งทดแทนพจน์ 9 และ 15 ตามลำดับ ที่ไม่มี PI ได้สามารถทดแทนพจน์ทั้งสองตัวนี้ได้ เราเรียก PI นี้ว่า Essential PI ซึ่งเป็น PI ที่จำเป็นต้องเลือกไปใช้งาน

เมื่อเราจำเป็นต้องเลือก PI_1 และ PI_7 ไปใช้งานให้ดูว่า PI_1 และ PI_7 ครอบคลุมพจน์ใดบ้าง โดยให้ใส่เครื่องหมายถูกไว้ด้านบนของพจน์นั้นๆ นั่นก็คือพจน์ 8, 9, 12, 13 และ 15

	2	4	6	8	✓	✓	10	✓	12	13	✓
** PI ₁				X	(X)			X	X		
PI ₂	X		X								
PI ₃	X						X				
PI ₄		X	X								
PI ₅		X							X		
PI ₆				X			X				
** PI ₇									X	(X)	

สุดท้ายแล้วจะยังคงเหลือพจน์ที่ไม่ครอบคลุมอีก 4 พจน์คือ 2, 4, 6 และ 13 ให้นำไปสร้างเป็นตาราง PI อันใหม่

	2	4	6	10
PI ₂	X		X	
* PI ₃	X			X
* PI ₄		X	X	
PI ₅		X		
PI ₆				X

จากตารางข้างบนจะต้องเลือก PI จำนวนน้อยที่สุดที่สามารถครอบคลุมได้ทั้ง 4 พจน์ โดยที่เราสามารถเลือก

PI₂ กับ PI₅ และ PI₆ เป็นชุดที่ 1

PI₃ และ PI₄ เป็นชุดที่ 2

ดังนั้นเราควรจะเลือกชุดที่ 2 เพราะใช้ PI เพียงแค่ 2 ตัว

สรุปแล้ว PI ที่เราเลือกมาใช้งานคือ PI₁ และ PI₇ ซึ่งเป็น Essential PI รวมกับ PI₃ และ PI₄

เมื่อเราเลือก PI ได้แล้ว ก็สามารถเขียนเป็นสมการซึ่งลครูปได้ดังนี้

PI₁ ทดสอบพจน์ 8, 9, 12, 13 = 1-0- จะเขียนเป็นสมการบูลีนได้ว่า A⁻C

(1-0- เป็น A⁻C เพราะ 1 ตัวแรกแทนตัวแปร A และ 0 แทนตัวแปร C)

PI₇ ทดสอบพจน์ 13, 15 = 11-1 จะเขียนเป็นสมการบูลีนได้ว่า ABD

PI₃ ทดสอบพจน์ 2, 10 = -010 จะเขียนเป็นสมการบูลีนได้ว่า BCD

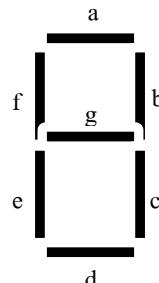
PI₄ ทดสอบพจน์ 4, 6 = 01-0 จะเขียนเป็นสมการบูลีนได้ว่า ABC

สุดท้ายแล้วจากโจทย์ที่ว่า F(A,B,C,D) = $\sum m(2, 4, 6, 8, 9, 10, 12, 13, 15)$

สามารถลครูปสมการบูลีนเหลือเป็น A⁻C + ABD + BCD + ABC ซึ่งเป็นสมการที่น้อยที่สุดไม่สามารถลครูปได้อีกแล้ว

5.2 ตัวอย่างการสร้างวงจรผลลัพธ์ตัวแสดงผลเจ็ดส่วนด้วยวิธีการของ Quine McCluskey

การสร้างตัวผลลัพธ์ตัวแสดงผลเจ็ดส่วนจะเป็นการสร้างวงจรที่มีหลายเอาท์พุท ซึ่งสามารถใช้งานร่วมกันได้ทำให้ประมวลจรรยาการทำงาน ดังนี้จึงควรใช้วิธีการของ Quine McCluskey เนื่องจากสามารถออกแบบโดยดูวงจรที่ใช้ร่วมกันได้



ตัวแสดงผลเจ็ดส่วนคือแอลอีดี(LED) ที่นำมาต่อขาฝากหนึ่งรวมเข้าด้วยกันเป็นขาร่วม(Common) ในที่นี้จะใช้แบบค่าโอลร่วมนี้คือขาข้อมูล a – g ต้องเป็นโลจิก ”1” แอลอีดีจึงจะสว่าง โดยมีลักษณะการวางแต่ละรางความจริงดังนี้

Truth table									
	Input				Output				
Minterms	A	B	C	D	g	f	e	d	c
0	0	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	1	0	1	1	0
3	0	0	1	1	1	0	0	1	1
4	0	1	0	0	1	1	0	0	1
5	0	1	0	1	1	1	0	1	0
6	0	1	1	0	1	1	1	1	0
7	0	1	1	1	0	0	0	0	1
8	1	0	0	0	1	1	1	1	1
9	1	0	0	1	1	1	0	1	1

จากตารางความจริงนำมาเขียนเป็นฟังก์ชันในรูปแบบมินเทอม(Minterm) ได้ดังนี้

$$F_a(A,B,C,D) = \sum m(0, 2, 3, 5, 6, 7, 8, 9)$$

$$F_b(A,B,C,D) = \sum m(0, 1, 2, 3, 4, 7, 8, 9)$$

$$F_c(A,B,C,D) = \sum m(0, 1, 3, 4, 5, 6, 7, 8, 9)$$

$$F_d(A,B,C,D) = \sum m(0, 2, 3, 5, 6, 8, 9)$$

$$F_e(A,B,C,D) = \sum m(0, 2, 6, 8)$$

$$F_f(A,B,C,D) = \sum m(0, 4, 5, 6, 8, 9)$$

$$F_g(A,B,C,D) = \sum m(2, 3, 4, 5, 6, 8, 9)$$

วิธีการลดรูปโดยใช้ Quine-McCluskey แบบหลายอ่าท์พุทนั้นมีข้อแตกต่างจากวิธีธรรมดัดงี้

1. ในทุกๆ พจน์ของตาราง List แต่ละอันจำเป็นต้องมีสถานะ(Flag) เพื่อบอกว่ามาจากอ่าท์พุตตัวใดบ้าง
2. การเปรียบเทียบอ่าท์พุตเพื่อสร้างอ่าท์พุตใหม่ที่สามารถรวมกันได้ จะรวมได้เฉพาะที่มีสถานะร่วมกันเท่านั้น
3. ทุกๆ พจน์ที่เป็น PI แล้วกดแทนพจน์ใดๆ ได้นั้น พจน์ที่แทนจะต้องมีสถานะเหมือนกับพจน์ที่เป็น PI นั้นๆ เท่านั้น
4. การเลือก PI ไปใช้งานเป็นวงจรต้องดูตามอ่าท์พุตเป็นชุดๆ ไป

1. ในทุกๆ พจน์ของตาราง List แต่ละอันจำเป็นต้องมีสถานะ(Flag) บอกว่ามาจากอ่าท์พุตตัวใดบ้าง

List 1			
Minterms	ABCD	Flags	
0	0000	abcdef	กลุ่มที่ 1 มีจำนวนเลข 1 จำนวน 0 ตัว
1	0001	bc	
2	0010	abdeg	กลุ่มที่ 2 มีจำนวนเลข 1
4	0100	bcfg	จำนวน 2 ตัว
8	1000	abcdefg	
3	0011	abcdg	
5	0101	acdfg	กลุ่มที่ 3 มีจำนวนเลข 1
6	0110	acdefg	จำนวน 3 ตัว
9	1001	abcdfg	
7	1111	abc	กลุ่มที่ 4 มีจำนวนเลข 1 จำนวน 4 ตัว

2. การเปรียบเทียบอ่าท์พุตเพื่อสร้างอ่าท์พุตใหม่ที่สามารถรวมกันได้ จะรวมได้เฉพาะที่มีสถานะร่วมกันเท่านั้น เช่น พจน์ 0 และ 1 รวมกันได้ เนื่องจากมี สถานะ bc ทึ้งคู่ ได้เป็น พจน์ 0, 1 ผลรวมเป็น 000- สถานะ bc

พจน์ 2 และ 3 รวมกันได้ เนื่องจากมี สถานะ abdg ทึ้งคู่ ได้เป็น พจน์ 2, 3 ผลรวมเป็น 001- สถานะ abdg ถึงแม้ว่า พจน์ 2 จะมีสถานะเป็น abdeg และพจน์ 3 มีสถานะเป็น abcdg แต่ที่ซ้ำกันมีเพียง abdg เท่านั้น

List 1			List 2		
Minterms	<i>ABCD</i>	Flags	Minterms	<i>ABCD</i>	Flags
0	0000	abcdef	0,1	000-	bc
1	0001	bc	0,2	00-0	abde
2	0010	abdeg	0,4	0-00	bcf
4	0100	bcfg	0,8	-000	abcdef
8	1000	abcdefg	1,3	00-1	bc
3	0011	abcdg	1,5	0-01	c
5	0101	acdfg	1,9	-001	bc
6	0110	acdefg	2,3	001-	abdg
9	1001	abcdfg	2,6	0-10	adeg
7	1111	abc	4,5	010-	cfg
			4,6	01-0	cfg
			8,9	100-	abcdfg
			3,7	0-11	abc
			5,7	01-1	ac
			6,7	011-	ac

หลังจากได้ตาราง List 2 ให้สร้างตาราง List 3 โดยยังคงใช้หลักการเดิมคือมองเฉพาะที่มีสถานะร่วมกันเท่านั้น

List 1			List 2			List 3		
Minterms	ABCD	Flags	Minterms	ABCD	Flags	Minterms	ABCD	Flags
0	0000	abcdef	0,1	000-	bc	0,1,2,3	00--	b
1	0001	bc	0,2	00-0	abde	0,1,4,5	0-0-	c
2	0010	abdeg	0,4	0-00	bcf	0,1,8,9	-00-	bc
4	0100	bcfg	0,8	-000	abcdef	1,3,5,7	0- -1	c
8	1000	abcdefg	1,3	00-1	bc	2,3,6,7	0-1-	a
3	0011	abcdg	1,5	0-01	c	4,5,6,7	01--	c
5	0101	acdfg	1,9	-001	bc			
6	0110	acdefg	2,3	001-	abdg			
9	1001	abcdfg	2,6	0-10	adeg			
7	1111	abc	4,5	010-	cfg			
			4,6	01-0	cfg			
			8,9	100-	abcdfg			
			3,7	0-11	abc			
			5,7	01-1	ac			
			6,7	011-	ac			

หลังจากได้ตาราง List 3 จะเห็นว่าไม่สามารถสร้างตาราง List 4 ได้เนื่องจากไม่พจน์คู่ใดที่มีสถานะร่วมกันและต่างกันเพียงแค่ 1 บิตได้

3. ทุกๆ พจน์ที่เป็น PI แล้วทดสอบพจน์ใดๆ ได้นั้น พจน์ที่แทนจะต้องมีสถานะเหมือนกับพจน์ที่เป็น PI นั้นๆ เท่านั้น ในการระบุว่าพจน์ชุดใดเป็น PI บ้าง ให้เริ่มจากมองที่ตาราง List 3 ให้ทุกๆ พจน์เป็น PI เรียงตามลำดับเป็น $PI_1 - PI_6$

โดย PI_3 จะสามารถทดสอบพจน์ 0, 1 ได้เนื่องจากมีสถานะ bc เมื่อเทียบกับ

PI_2 และ PI_5 จะสามารถทดสอบพจน์ 1, 5 ได้เนื่องจากมีสถานะ c เมื่อเทียบกับ

ส่วนพจน์อื่นๆ ไม่สามารถทดสอบพจน์ใดๆ ได้เนื่องจากสถานะไม่ตรงกันทั้งหมด

List 1			List 2			List 3		
Minterms	ABCD	Flags	Minterms	ABCD	Flags	Minterms	ABCD	Flags
0	0000	abcdef	0,1	000-	bc	0,1,2,3	00--	b PI ₁
1	0001	bc	0,2	00-0	abde	0,1,4,5	0-0-	c PI ₂
2	0010	abdeg	0,4	0-00	bcf	0,1,8,9	-00-	bc PI ₃
4	0100	bcfg	0,8	-000	abcdef	1,3,5,7	0—1	c PI ₄
8	1000	abcdefg	1,3	00-1	bc	2,3,6,7	0-1-	a PI ₅
3	0011	abcdg	1,5	0-01	c	4,5,6,7	01--	c PI ₆
5	0101	acdfg	1,9	-001	bc			
6	0110	acdefg	2,3	001-	abdg			
9	1001	abcdfg	2,6	0-10	adeg			
7	1111	abc	4,5	010-	cfg			
			4,6	01-0	cfg			
			8,9	100-	abcdfg			
			3,7	0-11	abc			
			5,7	01-1	ac			
			6,7	011-	ac			

หลังจากนั้นในตาราง List 2 ทุกๆ พจน์ที่เหลืออยู่จะถูกลายเป็น PI₇ – PI₁₉

List 1			List 2			List 3		
Minterms	ABCD	Flags	Minterms	ABCD	Flags	Minterms	ABCD	Flags
0	0000	abcdef \vee	0,1	000-	bc \vee	0,1,2,3	00--	b PI ₁
1	0001	bc \vee	0,2	00-0	abde PI ₇	0,1,4,5	0-0-	c PI ₂
2	0010	abdeg	0,4	0-00	bcf PI ₈	0,1,8,9	-00-	bc PI ₃
4	0100	bcfg	0,8	-000	abcdef PI ₉	1,3,5,7	0—1	c PI ₄
8	1000	abcdefg	1,3	00-1	bc PI ₁₀	2,3,6,7	0-1-	a PI ₅
3	0011	abcdg	1,5	0-01	c \vee	4,5,6,7	01--	c PI ₆
5	0101	acdfg	1,9	-001	bc PI ₁₁			
6	0110	acdefg	2,3	001-	abdg PI ₁₂			
9	1001	abcdfg \vee	2,6	0-10	adeg PI ₁₃			
7	1111	abc \vee	4,5	010-	cfg PI ₁₄			
			4,6	01-0	cfg PI ₁₅			
			8,9	100-	abcdfg PI ₁₆			
			3,7	0-11	abc PI ₁₇			
			5,7	01-1	ac PI ₁₈			
			6,7	011-	ac PI ₁₉			

โดย PI₁₉ จะสามารถถูกแทนพจน์ 0 ได้เนื่องจากมีสถานะ abcdef เหมือนกัน
 PI₃, PI₁₀ และ PI₁₁ จะสามารถถูกแทนพจน์ 1 ได้เนื่องจากมีสถานะ bc เหมือนกัน
 PI₁₆ จะสามารถถูกแทนพจน์ 9 ได้เนื่องจากมีสถานะ abcdgf เหมือนกัน
 PI₁₇ จะสามารถถูกแทนพจน์ 7 ได้เนื่องจากมีสถานะ abc เหมือนกัน
 ส่วนพจน์อื่นๆ ไม่สามารถถูกแทนพจน์ได้ ได้เนื่องจาก สถานะไม่ตรงกันทั้งหมด

List 1				List 2				List 3			
Minterms	ABCD	Flags		Minterms	ABCD	Flags		Minterms	ABCD	Flags	
0	0000	abcdef	✓	0,1	000-	bc	✓	0,1,2,3	00--	b	PI ₁
1	0001	bc	✓	0,2	00-0	abde	PI ₇	0,1,4,5	0-0-	c	PI ₂
2	0010	abdeg		0,4	0-00	bcf	PI ₈	0,1,8,9	-00-	bc	PI ₃
4	0100	bcfg		0,8	-000	abcdef	PI ₉	1,3,5,7	0—1	c	PI ₄
8	1000	abcdefg		1,3	00-1	bc	PI ₁₀	2,3,6,7	0-1-	a	PI ₅
3	0011	abcdg		1,5	0-01	c	✓	4,5,6,7	01--	c	PI ₆
5	0101	acdfg		1,9	-001	bc	PI ₁₁				
6	0110	acdefg		2,3	001-	abdg	PI ₁₂				
9	1001	abcdfg	✓	2,6	0-10	adeg	PI ₁₃				
7	1111	abc	✓	4,5	010-	cfg	PI ₁₄				
				4,6	01-0	cfg	PI ₁₅				
				8,9	100-	abcdefg	PI ₁₆				
				3,7	0-11	abc	PI ₁₇				
				5,7	01-1	ac	PI ₁₈				
				6,7	011-	ac	PI ₁₉				

ดังนั้นพจน์ที่เหลือทั้งหมดจะถูกยกไปเป็น PI₂₀ - PI₂₅

List 1			List 2			List 3					
Minterms	ABCD	Flags	Minterms	ABCD	Flags	Minterms	ABCD	Flags			
0	0000	abcdef	✓	0,1	000-	bc	✓	0,1,2,3	00--	b	PI ₁
1	0001	bc	✓	0,2	00-0	abde	PI ₇	0,1,4,5	0-0-	c	PI ₂
2	0010	abdeg	PI ₂₀	0,4	0-00	bcf	PI ₈	0,1,8,9	-00-	bc	PI ₃
4	0100	bcfg	PI ₂₁	0,8	-000	abcdef	PI ₉	1,3,5,7	0—1	c	PI ₄
8	1000	abcdefg	PI ₂₂	1,3	00-1	bc	PI ₁₀	2,3,6,7	0-1-	a	PI ₅
3	0011	abcdg	PI ₂₃	1,5	0-01	c	✓	4,5,6,7	01--	c	PI ₆
5	0101	acdfg	PI ₂₄	1,9	-001	bc	PI ₁₁				
6	0110	acdefg	PI ₂₅	2,3	001-	abdg	PI ₁₂				
9	1001	abcdfg	✓	2,6	0-10	adeg	PI ₁₃				
7	1111	abc	✓	4,5	010-	cfg	PI ₁₄				
				4,6	01-0	cfg	PI ₁₅				
				8,9	100-	abcdfg	PI ₁₆				
				3,7	0-11	abc	PI ₁₇				
				5,7	01-1	ac	PI ₁₈				
				6,7	011-	ac	PI ₁₉				

4. การเลือก PI ไปใช้งานเป็นวงจรต้องดูตามเอาท์พุทเป็นชุดๆ ไป

เช่น เอาท์พุท a จะดูเฉพาะ PI ที่มีสถานะเป็น a เท่านั้นที่เราสนใจคือ PI₅, PI₇, PI₉, PI₁₂, PI₁₃, PI₁₆, PI₁₇, PI₁₈, PI₁₉, PI₂₀, PI₂₂, PI₂₃, PI₂₄ และ PI₂₅ เท่านั้น แต่ในขณะเดียวกัน PI ก็จะทดแทนได้เฉพาะพจน์ที่ a มีเท่านั้นคือ 0, 2, 3, 5, 6, 7, 8, 9

	a								
	0	2	3	5	6	7	8	9	
PI₅		X	X		X	X			
PI₇	X	X							
PI₉	X						X		
PI₁₂		X	X						
PI₁₃		X			X				
** PI₁₆							X	(X)	
PI₁₇			X			X			
PI₁₈				X		X			
PI₁₉					X	X			
PI₂₀		X							
PI₂₂							X		
PI₂₃			X						
PI₂₄				X					
PI₂₅					X				

ส่วนการเลือก PI ไปใช้งานก็เลือกเหมือนปกติคือ หา Essential PI ก่อน แล้วค่อยเลือก PI ที่เลือกให้มีจำนวนรวมน้อยที่สุดแต่ครอบคลุมทุกๆ พจน์ของ a เช่น

จากตารางจะเห็นว่า PI₁₆ เป็น Essential PI เนื่องจากมี PI เดียวที่ครอบคลุมพจน์ 9

และเมื่อตัดพจน์ที่ครอบคลุมโดย PI₁₆ ไปแล้วจะเหลือดังนี้

	a						
	0	2	3	5	6	7	
PI₅		X	X		X	X	
PI₇	X	X					
PI₉	X						
PI₁₂		X	X				
PI₁₃		X			X		
** PI₁₆							
PI₁₇			X			X	
PI₁₈				X		X	
PI₁₉					X	X	
PI₂₀		X					
PI₂₂							
PI₂₃			X				
PI₂₄				X			
PI₂₅					X		

ส่วนการเลือก PI ที่เหลือจำเป็นต้องคุ้ว่าตัวใดใช้ร่วมกับเอาท์พุตอันอื่นๆ ได้บ้าง จะทำให้ประหัดวงจรได้มากขึ้น จึงควรเปียนตารางให้มีทุกๆ เอาท์พุตที่เดียวจะได้ดูได้ง่าย เช่นหากครุ่รวมกัน 3 เอาท์พุตคือ e, f และ g

	e				f					g							
	0	2	6	8	0	4	5	6	8	9	2	3	4	5	6	8	9
PI₇	X	X															
PI₈					X	X											
PI₉	X			X	X				X								
PI₁₂											X	X					
PI₁₃		X	X								X					X	
PI₁₄					X	X							X	X			
PI₁₅					X		X						X		X		
**PI₁₆								X	(X)						X	(X)	
PI₂₀		X									X						
PI₂₁					X								X				
PI₂₂			X						X							X	
PI₂₃												X					
PI₂₄						X							X				
PI₂₅			X					X						X			

จากตารางจะเห็นได้ว่า **PI₁₆** ยังคงเป็น Essential PI อยู่เนื่องจากเป็นตัวเดียวที่ครอบคลุมพจน์ 9 ดังนั้นมีตัด **PI₁₆** ซึ่งครอบคลุมพจน์ 8 และ 9 ออกไปแล้วจะได้ตาราง PI ดังนี้

	e				f				g				
	0	2	6	8	0	4	5	6	2	3	4	5	6
	PI₇	X	X										
PI₈					X	X							
PI₉	X			X	X								
PI₁₂									X	X			
PI₁₃		X	X						X				X
PI₁₄						X	X				X	X	
PI₁₅						X		X			X		X
**PI₁₆													
PI₂₀		X							X				
PI₂₁						X					X		
PI₂₂				X									
PI₂₃										X			
PI₂₄							X						X
PI₂₅			X					X					X

จากตารางจะเห็นได้ว่า

เอาท์พุท e สามารถเลือกชุด PI คี่ที่สูดคือ **PI₉** และ **PI₁₃**

เอาท์พุท f สามารถเลือกชุด PI คี่ที่สูดคือ **PI₈** + (**PI₁₄** หรือ **PI₂₄**) + (**PI₁₅** หรือ **PI₂₅**)

หรือ **PI₉** + **PI₁₄** + (**PI₁₅** หรือ **PI₂₅**)

หรือ **PI₉** + **PI₁₅** + **PI₂₄**

เอาท์พุท g สามารถเลือกชุด PI คี่ที่สูดคือ **PI₁₂** + **PI₁₄** + (**PI₁₃** หรือ **PI₁₅** หรือ **PI₂₅**)

หรือ **PI₁₃** + **PI₁₄** + **PI₂₃**

จะเห็นได้ว่าถ้าเราเลือก PI ที่ใช้ช้ากันได้ดีที่สุดสำหรับเอาท์พุท e, f และ g คือ

e คือ **PI₉** และ **PI₁₃**

f คือ **PI₉**, **PI₁₄** และ **PI₁₅**

g คือ **PI₁₃**, **PI₁₄** และ **PI₂₃**

จะเห็นได้ว่ามี **PI₉**, **PI₁₃** และ **PI₁₄** ที่สามารถใช้ร่วมกันได้

ในเอาท์พุทที่เหลือก็มีหลักการเดียวกัน

สรุป

กระบวนการลดรูปสมการบูลีนโดยวิธีการของ Quine McCluskey จะมีขั้นตอนหลักๆ 3 ขั้นตอนคือ

1. จัดกลุ่มพจน์ตามจำนวนเลข 1 ในพจน์นั้นๆ
2. เปรียบเทียบสมาชิกของกลุ่มที่อยู่ติดกันทุกตัวเข้าด้วยกัน ไปเรื่อยๆ จนหมด
3. สร้างตาราง PI (Prime Implicant) และเลือก PI ที่น้อยที่สุด

ซึ่งในขั้นตอนที่ 1 จะเป็นการแบ่งกลุ่มของอินพุตตามจำนวนบิตเลข 1 ที่มีเพื่อสร้าง List ที่ 1 หลังจากนั้นในขั้นตอนที่ 2 จะเป็นการเปรียบเทียบอินพุตในกลุ่มที่อยู่ติดกันในทุกๆ อินพุตเพื่อหาคู่อินพุตที่แตกต่างกันเพียง 1 บิต เพื่อสร้าง List ที่ 2 และทำซ้ำเพื่อสร้าง List ที่ 3 4 ต่อไปเรื่อยๆ จนไม่มีคู่อินพุตใดๆ ที่แตกต่างกันเพียง 1 บิตอีกต่อไป แล้วจึงเป็นการสร้างตาราง PI เพื่อหากลุ่มของ PI เพื่อที่น้อยที่สุดที่ครอบคลุมทุกๆ อินพุตทั้งหมดที่มี โดยมี Essential PI หรือ PI ที่มีเพียงตัวเดียวในอินพุทนั้นๆ ที่จำเป็นต้องเลือก โดยสามารถเลือกใช้ PI ซ้ำกันในแต่ละເ好象ท์พุทได้ จะสามารถทำให้ประหยัดเวลาและลดขนาดของวงจรได้มากขึ้นเนื่องจากสามารถใช้งานบางส่วนร่วมกันได้

คำถามท้ายบท

1. กระบวนการลดรูปโดยใช้วิธีการของ Quine McCluskey จะสามารถลดรูปได้มากกว่า การใช้พีชคณิตบูลีนและคนอร์เมเนียนอย่างไร
2. การลดรูปที่มีอินพุตจำนวน 5 บิต สามารถใช้วิธีการของ Quine McCluskey ได้หรือไม่ จงอธิบาย
3. จงครุปสมการจาก $F(A,B,C,D) = \sum m(1, 4, 7, 8, 9, 10, 12, 14, 15)$
4. จำนวน List สูงสุดที่สามารถสร้างได้จากอินพุตขนาด 4 บิต หรือ 5 List ใช้หรือไม่ จงอธิบาย
5. ทุกครั้งของการลดรูปด้วยวิธีการของ Quine McCluskey ของต้องมี Essential PI อย่างน้อย 1 ตัวเสมอใช่หรือไม่ เพราะเหตุใด
6. จำนวน PI สูงสุดที่เป็นไปได้ของการลดรูปโดยใช้วิธีการของ Quine McCluskey ที่มี 4 อินพุตคือจำนวนเท่าไหร่
7. เพราะเหตุใด จึงต้องเลือกจำนวน PI ให้น้อยที่สุดที่ครอบคลุมทุกๆ อินพุต
8. เพราะเหตุใด การเปรียบเทียบคู่ของอินพุตแต่ละตัวต้องคุณภาพคู่ที่อินพุตต่างกันเพียงแค่ 1 บิตเท่านั้น

== หน้าว่าง ==

បរຮណານຸກຮມ

- [1] Charles H. Roth, Jr., Fundamentals of Logic Design,2006.
- [2] Alan B. Marcovitz, Introduction to logic, Second edition, McGRAW-HILL International edition, 2005.
- [3] XST User Guide 8.1i, Xilinx Inc.
- [4] ISE 8.2 In-Depth Tutorial, Xilinx Inc.
- [5] Synthesis and Simulation Design Guide 8.1i, Xilinx Inc.
- [6] Stephen Brown, Zvonko Vranesic, Fundamentals of Digital Logic with VHDL Design, McGraw-Hill Professional, 2004.
- [7] IEEE Std 1076-1993 : IEEE Standard VHDL Language Reference Manual.
- [8] IEEE Std 1076-1987 : IEEE Standard VHDL Language Reference Manual.
- [9] IEEE Std 1164-1993 : IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164).
- [10] IEEE Std 1076.3-1997 : IEEE Standard VHDL Synthesis Packages.
- [11] FPGA Compiler II / FPGA Express, VHDL Reference Manual Version 1999.05, Synopsys Inc., May 1999.
- [12] VHDL Language Reference, Summary Technical Reference, TR0114 (v1.2), Altium Ltd., September 20, 2005.

== หน้าว่าง ==

ดictionar

&

& 261

.

.bit 21

.jed 21

.pdf 23

2

2 to 1 Multiplexer 147, 150, 263

2 to 4 Decoder 132, 135, 256

2's complement 260

4

4 to 1 Multiplexer 147, 151

4 to 2 Binary Encoder 145, 146

8

8 to 1 Multiplexer 152

A

ABS 261

Absolute value 261

ACE 75

Active high 114

Active Low 114

Add Bus tap 46

Add I/O Maker 46

Add Symbols 46

Add Wire 46

Adobe Reader 7 23

Architecture 256

Arithmetic operators 261

Array 260

Array types 260

Assign New Configuration File 80

Asynchronous clear 269

Asynchronous counter 170

Asynchronous preset and clear 162

Asynchronous sequential circuit 228

Attribute 264

B

BCD 132

Behavioral simulation 20, 85

Binary counter 170

Binary up-counter 170

bit 260

bit_vector 260

Bitstream 21

block RAM 31

Block RAM 17

boolean 260

Boolean algebra 13

boolean expression 113

Bouncing 95, 105, 167, 218

Boundary Scan Mode 36

Buffer 113

Buzzer 25, 30

C

Carry 127

Carry-in 128

Carry-out 128

Cascadable counter 172

Case statement 268

Chip capacitor 35

CLB 17

Clear 162

Clock 157, 161, 171

Clock enable input 172

Clock enable output 172

Clocked sequential machines 228

CLR 162

CMOS 13

Combinational circuit 228

Combinational logic circuits 127

Common 27

Common Anode 133

Common cathode 133

Concatenation operators 261

Conditional signal assignment 265, 266

Configurable Logic Block 17

Configuration data 17, 36

Configuration data) 29

Configure Device 60, 79

Constant 263

Constraints 21

Control unit 254

Copy 47

CORE Generator 18

CPLD 14, 17, 19, 24, 29, 57, 83

CPLD Explorer XC9572 29

CPLD Explorer XC9572XL 24

Create a self-checking test bench 86

Create schematic symbol 99

Cross talk.....	35
Current state	228

D

D Flip-flop.....	269
D Flip-Flop	83, 161, 165
Data type.....	260
DCM	31
Debouncer	167, 218
Delay time	20
Delay-Locked Loop	17
Demultiplexer	153
DEMUX.....	153
Design entry	19
Design Entry	39, 64, 115
Design Implementation.....	21, 55, 70, 88, 119
Design synthesis.....	20, 54, 69, 88, 118
Design verification	20
Design Verification	85
Device Programming.....	21
Digital Clock Manager	31
Digital Frequency Synthesizer	31
DIP Switch	29, 30
Discovery-III XC3S200.....	30
DLL	17
Documentation	17
Don't care.....	260

E

Edge triggered.....	157
Editor toolbar	45
EEPROM	15, 31, 36
Embedded microcontroller	17
Entity.....	256
Entity declaration.....	256
Essential PI	295, 305
Excitation map	234, 237
Expansion ports.....	31

F

Fast slew rate	35
FB	15
File	263
Finite state machine.....	228
Flash PROM.....	78
Flat Cable	35
Flip-Flop.....	157
FPGA.....	16, 17, 19, 21, 30, 36, 64, 78
FPGA editor	18
FSM	228
Full adder	127, 131
Function Block.....	15

G

Gate level	20
Generate Programming File	59, 74
Generate PROM	75
Gray.....	232

H

Half adder	127, 130
Hardware Description Language	19, 256
hardware multiplier	31
HDL.....	19, 20
High.....	260
High impedance	113, 260
Hold.....	157

I

I/O Block	15
I/O LOGIC	17
I/O Marker	51
I ² C	31, 36
IEEE	256
If statement	268
iMPACT	60, 79
Implement Design.....	54, 59, 73
Implementation constraints file	55
Initial Timing and Clock Wizard.....	85
integer.....	260
Interface.....	256
IOB	15
ISE Simulator	18, 20, 41
ISE WebPACK 10.i.....	21, 56
ISE WebPACK 8.i.....	18

J

JK Flip-Flop	162, 167
JTAG	15, 29, 30, 36, 60, 78
JTAG File.....	75
JTAG Mode.....	36

K

K-map.....	13
------------	----

L

Latch	157
LED	30, 297
LED แบบ 2 สีตามสี	27
Logic gates	113
Logic monitor	27, 29
Logic trainer	28
Logical operators.....	261

Low.....	260
LVC MOS33	35
LV TTL.....	35

M

Map.....	21
Master Serial Mode.....	36
Master serial	81
Mealy machine.....	228
Mealy-type FSM.....	228, 230
Memory	157
Minterm	289
MOD.....	261
Modulus	261
Monostable	167
Moor machine.....	228
Moor-type FSM	228, 229, 246
MUX	147

N

NAND Gate.....	157
NAND Gate latch	159
Netlist file.....	21
New Project Wizard	40
New Source	41
Next state.....	229, 230
No change	157
Nonstandard package	261
NOR Gate	157
NOR Gate latch.....	159

O

Object.....	263
Object Properties	47
Onboard Oscillator	25
Operater.....	260
Operator	261
Oscillator	29, 31
Output map.....	234, 237

P

Parallel in–Parallel out	217
Parallel in–Serial out	217
Parallel Port.....	29
Paste	47
Physical file	21
PI	289, 293
Place and Route	21
Platform Flash PROM	30
Port mode	260
Positive edge-triggered	161
Present input.....	228
Present output	228
Present state	228, 229, 230
PRESET	162

Previous input	228
Prime Implicant.....	289, 293
Printer Port	36
Priority encoder	267
Process statement	267
Process status icon	55
Processes window	45
Programming file generation	21
Project Navigator	39
Pull down	114
Pull up	114
Pulse.....	167
Push button switch	25
Push button Switch	28
Push Button Switch	31

Q

Quine McCluskey	13, 289, 298
-----------------------	--------------

R

rchitecture body	256
Registration ID	21
Relational operators	261
REM.....	261
Remainder.....	261
Restore Default Layout	42
Ripple counter	170
Rise time	35
Rotate	46
Route	21
RS-232C	36
RS-232C Port	31

S

Schematic	19, 41, 64
Segment	28, 30
Select	45
Selected signal assignment	266
Selected signal assignment	265
Sensitivity list	269
Sequence detector	246
Sequential	217
Sequential circuit	228
Sequential machine	228
Sequential statement	268
Serial Flash PROM	36
Serial in–Parallel out	217
Serial in–Serial out	217
Serial PROM	17
Signal	263
Signal assignment statement	268
signed	260
Simple signal assignment statement	265
Simulation	20
Simulator	41
Slide switch	25
Slide Switch	28

Slow slew rate	35
SOP	289
Sources window	45
Standard toolbar	45
State	228
State assignment	231
State diagram	19, 229, 230, 231
State encoding	231
State reduction	231
State register	228, 233
State table	229, 230, 231
State transition	229, 230
State-assigned table	231
std_logic	260
std_logic_vector	260
Sum	127
Sum of Product	289
Switch matrix	15
Symbols	91, 99, 256
Synchronous counter	171
Synchronous reset	162, 269
Synchronous sequential circuit	228
Synopsys	261
Synopsys packages	261
Syntax	52
Synthesis Tool	40
Synthesize	54
System design	17
System implementation	17
System requirement	17

T

T Flip-Flop	163
Terminate	35
Testing and debugging	17
Timing constraints	21
Timing simulation	20
Toolbar	45
Top-Level Source Type	40
Transcript window	45
Transition	229, 230
Translate	21
Transmission Line effect	35
Tri-state buffer	113
TTL	13

U

UCF	55
Uninitialized	260
Unknown	260
unsigned	260
User constraints file	55

V

Variable	263
Variable assignment statement	268
Vending machine	254

Verilog	19
VHDL	19, 256
VHSIC	256

W

Waveform Editor	20
Weak high	260
Weak low	260
Weak unknown	260
when others	267
Workspace	45

X

Xilinx Synthesis Tool	20
XST	20, 40

Z

Zoom Full View	45
Zoom In	45
Zoom Out	45
Zoom To Box	45

I

เก้แมพ	13
เกร็งขาเนื้อ อัดลม อัดโนมัติ	254
เช่วนเชกเม้นต์	132
เทคนิคการสแกน	27
เอนดิต	256
เอาด์ฟูลปีจุบัน	228

II

แคลใจคร่วม	133
แผนภูมิสกานะ	19
แคลช์	157
แอโนดิคร่วม	133
แอนด์เกต	120
แอลอีดี	297

III

ไมโนเตเบิล	83, 105, 167, 218
ไอเปอร์เรคอร์	261

ๆ

ไคร-สเดคบ์ไฟอร์	113, 127
ไฟล์	263
ไมโครคอนโทรลเลอร์	17
ไมโครคอนโทรลเลอร์แบบผึ่งด้า	17

ก

กราฟเพลน	35
การเปิดไฟล์ที่เก็บออกแบบໄว้แล้วมาใช้งาน	62
การโปรแกรมข้อมูลวงจรลงชิป	21, 59, 74
การจำลองการทำงาน	20
การจำลองการทำงานเชิงพุทธิกรรม	83
การตรวจสอบความถูกต้อง	52
การตรวจสอบความถูกต้องของวงจร	20, 85
การติดตั้งซอฟต์แวร์ทุกๆ	21
การตั้งเคราะห์วงจร	20, 54, 69, 88, 118, 128, 141, 147
การออกแบบวงจร	19, 39
การออกแบบวงจรซีเคนซีชีบล	228
การออกแบบวงจรรับ N แบบอะจังไกรนั้ส	206

ข

ขอบข่าย	161
ขั้นตอนการโปรแกรม Flash PROM	78
ขั้นตอนการขอและดาวน์โหลดซอฟต์แวร์	21
ขั้นตอนการออกแบบ Finite state machine	231
ขั้นตอนการออกแบบวงจร	64, 115
ขั้นตอนสร้างไฟล์ PROM	75

ค

คินิแมกเลสติก	13
ค่าโคลร์วัม	27
ค่าคงที่	263
ค่าถ่วงก่อนคอนโทรลเลอร์เรนต์	265
ค่าสั่งซีเคนซีชีบล	268

จ

จัมป์อร์	78
----------	----

ย

ชนิดข้อมูล	260
------------	-----

ชนิดข้อมูลของเรซ	260
ชิฟต์รีจิสเตอร์	217

ๆ

ชิ้นส่วนสำเร็จ	162
ชิมอส	13

ต

ตัวแปร	263
ตัวดำเนินการ	260
ตัวดำเนินการคณิตศาสตร์	261
ตัวดำเนินการความสัมพันธ์	261
ตัวดำเนินการตรวจสอบ	261
ตัวดำเนินการรวมข้อมูล	261
ตัวทด	127
ตัวทดเช้า	128
ตัวทดออก	128

ห

หิฟเฟอต์	13
----------	----

น

นาฬิกาดิจิตอล	213
---------------	-----

บ

บอร์ดทดลอง	24
บัฟไฟอร์	113, 127
บัฟไฟอร์สองทิศทาง	127

ย

ประเภทของพอร์ต	260
----------------	-----

ผ

ผู้ร่วม	127
---------	-----

พ

พอร์ตขนาด	29, 36
พอร์ตคณิตกรรม.....	36
พัลส์	167
พิกอเบลดช..	17
พิชคณิตคูลิน	13
พูลดาวน์	114
พูลอัพ	114

ฟ

ฟลิปฟลอป	157, 161
----------------	----------

ร

รีเซ็ตเตอร์.....	228
------------------	-----

ส

ล็อกอินเกต.....	113, 115, 125
-----------------	---------------

จ

วงจรเร้าหัส	143
วงจรเร้าหัสที่มีคำดับความสำาคัญ.....	267
วงจรเบรย์บีบ	141
วงจรแก๊บเนาซ์	95, 105
วงจรคอมบินเนชัน	127, 228
วงจรซีเคแวนซีซิล	217, 228
วงจรซีเคแวนซีซิลแบบบิชิงโกรนัส	228
วงจรซีเคแวนซีซิลแบบอะชิงโกรนัส	228
วงจรคีเบแซชอร์.....	167, 218
วงจรคีมัคติเพล็กซ์ชอร์.....	153
วงจรตรวจล้ำด้าน	246, 252
วงจรกรองหัส.....	132
วงจรกรองหัส BCD เป็นเซ่านเชกเม้นต์.....	137, 138
วงจรกรองหัสเท้า 2 ออก 4.....	256
วงจรกรองหัสตัวแสดงผลเจ็ดส่วน	297
วงจรนับ 10.....	207, 272
วงจรนับ 16.....	270
วงจรนับ 4	95, 105
วงจรนับ 4 แบบบิชิงโกรนัส.....	39, 64
วงจรนับ 8 แบบบิชิงโกรนัส	232
วงจรนับ FF	196

วงจรนับเลขฐานสอง.....	170
วงจรนับแบบบิชิงโกรนัส	171
วงจรนับแบบอะชิงโกรนัส.....	170
วงจรนับเข็มแบบแล็บไบนาเรี่ย.....	170
วงจรนับเข็มแบบบิชิงโกรนัส	181
วงจรนับเข็ม 4 หลัก.....	200, 276, 284
วงจรนับเข็ม-ลงแบบบิชิงโกรนัส	188
วงจรนับลงแบบบิชิงโกรนัส	185
วงจรนับลงแบบบิบีด	178
วงจรนับลงแบบอะชิงโกรนัส	170
วงจรบวก	127, 262
วงจรแมกติเพล็กซ์ชอร์	147
วงจรแมกติเพล็กซ์ชอร์ 4 อินพุต 1 เอาต์พุต	266
วงจรแมกติเพล็กซ์ชอร์แบบ 2 อินพุต 1 เอาต์พุต.....	150, 263
วงจรแมกติเพล็กซ์ชอร์แบบ 4 อินพุต 1 เอาต์พุต.....	151
วงจรนำ่วยความคุณ	254
วงจรหารความถี่	206
วิธีการติดตั้ง ISE WebPACK 8.1i	21
วิธีการติดตั้ง Service Pack 2.....	23
วิธีวิเคราะห์วงจร	19

ศ

สไลด์สวิตช์	28
สมการบูลีน	113
สวิตซ์กังคิดปล่อยดับ	28
สัญญาณ	263
สัญญาณนาฬิกา.....	157, 161, 171
สายแพร์.....	35

ห

หน่วยความจำ.....	157, 228
------------------	----------

อ

ออด	25
ອອນເຈັກຕີ	263
ອອർເກຕ	120
อะเรບີ.....	260
อะชิงโกรนัสພວິເສດແລະເກີຍີ.....	162
ອາຊີເທັກເຂອງ	256
ອາຊີເທັກເຂອງບອດີ	256
ອືນໄວຣີເຄອງ	120
ອືນພຸບຈຸ້ານ	228
ອືນພຸດອົດ	228

ออกแบบไอซีดิจิตอลด้วย FPGA และ CPLD ภาคปฏิบัติโดยใช้วิธี Schematic

FPGA และ CPLD เข้ามายืนหนาทอ่าย่างมากทางด้านการออกแบบไอซี (IC Design) ในปัจจุบัน ซึ่งอุปกรณ์ที่ใช้เทคโนโลยีชั้นสูงไม่ว่าจะเป็น เครื่องมือทางพาร์ การแพทช์ ระบบเครือข่าย และ สื่อสาร ในปัจจุบันนี้จะมีบนาคเล็กและประสมทิศภาพสูง บอยครั้งจะพบว่ามี FPGA หรือ CPLD อยู่บนแผงวงจร หนังสือเล่มนี้อธิบายเกี่ยวกับกระบวนการออกแบบ ไอซีดิจิตอลด้วย FPGA และ CPLD โดยใช้วิธี Schematic โดยยกตัวอย่างการออกแบบวงจรทั้งภาคทฤษฎีและภาคปฏิบัติ และสามารถทดลองได้จริงทุกการทดลอง ได้แก่ โลจิกเกต แลตซ์ ฟลิปฟล็อป วงจรคอมบินेशัน และ วงจรซีเคเวนเซียล เนื่องจาก วิธี Schematic เป็นวิธีคาดผังวงจรโดยใช้คอมพิวเตอร์แล้วนำไปดาวน์โหลดลง FPGA หรือ CPLD จึงทำได้รวดเร็วกว่าการนำไอซี เช่น ตรรกะ TTL ต่างๆ ไปต่อวงจรบนแผงต่อวงจรหรือโปรโทบอร์ด จึงสามารถลดความยุ่งยากในการต่อวงจรและการตรวจสอบลงไปได้มาก ทำให้สามารถออกแบบสร้างวงจรที่มีขนาดใหญ่ขึ้นและซับซ้อนกว่าได้ สามารถลดเวลาในการเรียนการสอนวิชาดิจิตอลทั้งภาคทฤษฎีและภาคปฏิบัติลงไปได้มาก หนังสือเล่มนี้จึงเหมาะสมสำหรับนักศึกษา อาจารย์ และ ผู้สนใจทางด้านการออกแบบวงจรดิจิตอลระดับพื้นฐานจนถึงระดับปานกลาง โดยมีในงานการทดลองต่างๆ มากถึง 30 การทดลอง

ประวัติผู้เขียน

นายเจริญ วงศ์ชุมย์ เป็น สำเร็จการศึกษาวิศวกรรมศาสตรบัณฑิต (เกียรตินิยมอันดับ 1) สาขาวิศวกรรมคอมพิวเตอร์ ปี 2541 และ วิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมไฟฟ้า ปี 2544 จากสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง เป็นอาจารย์ประจำ ภาควิชาศึกษาศาสตร์คอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง เป็นอาจารย์พิเศษ ภาควิชาศึกษาศาสตร์หลักสถาบัน มีผลงานด้านวิชาการระดับนานาชาติจำนวน 5 บทความ และมีประสบการณ์ในการออกแบบและตรวจสอบความถูกต้อง (Design verification) ไมโครโปรเซสเซอร์ ARM7 ที่เขียนด้วยภาษา VHDL

E-mail : kvocharo@kmitl.ac.th

นาย ณรงค์ ทองนิม สำเร็จการศึกษาวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมไฟฟ้า มหาวิทยาลัยสงขลานครินทร์ ปี 2529 และ วิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมไฟฟ้ากำลัง จุฬาลงกรณ์มหาวิทยาลัย ปี 2533 ปัจจุบันเป็น กรรมการผู้จัดการ บริษัท เอเพก อินสตรูเม้นต์ จำกัด มีผลงานวิชาการทางด้าน FPGA ในต่างประเทศจำนวน 3 บทความ งานทางด้านวิชาการ เป็นอดีตอาจารย์พิเศษ ภาควิชาศึกษาศาสตร์ มหาวิทยาลัยธรรมศาสตร์ และ เป็นอดีตนักวิจัย ศูนย์เชี่ยวชาญพิเศษเฉพาะด้านเทคโนโลยีไฟฟ้ากำลัง จุฬาลงกรณ์มหาวิทยาลัย

E-mail : support@ailogictechnology.com