

ออกแบบ ไอซีดิจิตอล
ด้วย FPGA และ CPLD ภาคปฏิบัติ
โดยใช้ภาษา

VHDL

ซอฟต์แวร์ทูล **ISE WebPACK**

นาย ณรงค์ ทองฉิน

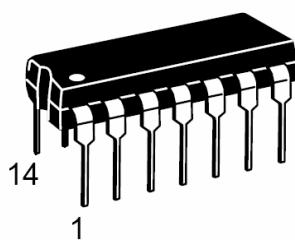
นายเจริญ วงศ์ชุมเย็น

บทที่ 1

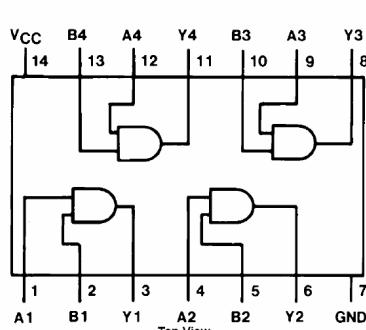
การออกแบบวงจรดิจิตอลด้วย FPGA และ CPLD

1.1 ไอซีมาตรฐาน

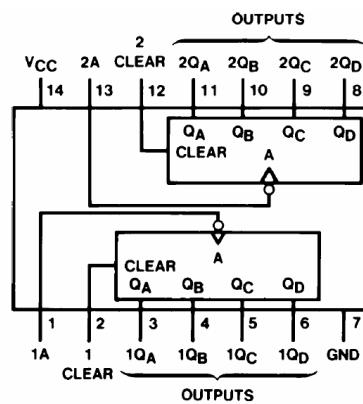
การออกแบบวงจรดิจิตอลขนาดเล็กโดยปกติจะนิยมใช้ชิพหรือไอซีมาตรฐาน เช่น ไอซี CMOS (อ่านว่าซีมอส) ตรรกะ 4000 และ 74HC00 หรือไอซี TTL (อ่านว่าทีทีแอล) ตรรกะ 74LS00 เป็นต้น ตัวอย่างแพคเกจไอซีแสดงดังรูปที่ 1.1a) ไอซี TTL เช่น เบอร์ 74LS08 (แอนด์เกต 2 อินพุต 4 ตัว) แสดงดังรูปที่ 1.1b) และ CMOS เช่น เบอร์ 74HC393 (วงจรนับไบนาเรียหรือฐานสอง 4 บิต 2 ตัว) แสดงดังรูปที่ 1.1c) จะเห็นได้ว่า ไอซีสำเร็จรูปเหล่านี้จะมีฟังก์ชันทำงานทางลốiจิกแบบตายตัวและเป็นวงจรขนาดเล็กอยู่ภายในเพียงไม่กี่ตัว จึงไม่เหมาะสมกับงานออกแบบวงจรขนาดใหญ่หรือความถี่สูงเนื่องจากเกิดเวลาล่าช้า (Delay) ขึ้นภายในตัวไอซีและในสายสัญญาณ โดยความเร็วของสัญญาณต่างๆ ในลายเส้นทองแดงของ PCB (ชนิด FR4) หรือสายสัญญาณนั้นจะมีความเร็ว (ค่ากลางๆ) ประมาณครึ่งหนึ่งของความเร็วแสงหรือ 15-18 เซนติเมตรต่อนาโนวินาที หากข้อจำกัดดังกล่าวทำให้การออกแบบแผงวงจรขนาดใหญ่ที่ใช้ความถี่สูงหลายสิบเมกะเฮิร์ตซ์มีความยุ่งยากมากขึ้นและอาจทำไม่ได้



1.1a) แพคเกจไอซี



1.1b) รายละเอียดของไอซีที่เป็นแอนด์เกต



1.1c) รายละเอียดของไอซีที่เป็นวงจรนับ

รูปที่ 1.1 ตัวอย่าง ไอซีตรรกะ TTL และ CMOS

ต่อมาได้มีการคิดค้น ไอซีหรือชิปดิจิตอลonen กประสงค์ที่โปรแกรมได้เพื่อให้มีฟังก์ชันการทำงานตามที่ต้องการเป็นผลสำเร็จประมาณปี 1970 โดยเรียกว่า Programmable Logic Device = PLD (อ่านว่าพีแอลดี) ซึ่งภายในจะบรรจุวงจรล็อกอิจิพื้นฐานที่มีฟังก์ชันทำงานแบบไม่ตายตัวไว้เป็นจำนวนมาก ปัจจุบันได้มีการออกแบบวงจรขนาดใหญ่โดยใช้ชิปดิจิตอลonen กประสงค์แทนการออกแบบด้วย ไอซีมาตรฐานตรรกะ TTL หรือ CMOS กันมากขึ้น น้อยครั้งเราจะพบชิป FPGA (Field Programmable Gate Array = เอฟพีจีเอ) และ/หรือ CPLD (Complex Programmable Logic Device = ซีพีแอลดี) เป็นส่วนประกอบที่ติดตั้งอยู่บนแผงวงจร เช่น ทางด้านสื่อสาร การแพทย์ การทหาร ระบบเครือข่าย หรือ เครื่องมือวัดต่างๆ เป็นต้น

การที่เราออกแบบวงจรย่อมส่วนต่างๆ รวมไว้ใน FPGA และ/หรือ CPLD เพียงตัวเดียวหรือเพียงไม่กี่ตัว ได้นั้นจะทำให้ แผงวงจร มีขนาดลดลงอย่างมาก ทำให้ได้วงจรที่ทำงานเร็วขึ้น เพราะสายสัญญาณต่างๆ สั้นลงและสายสัญญาณส่วนใหญ่จะอยู่ภายในชิป ทำให้ได้ผลิตภัณฑ์ที่มีขนาดเล็กกะทัดรัด

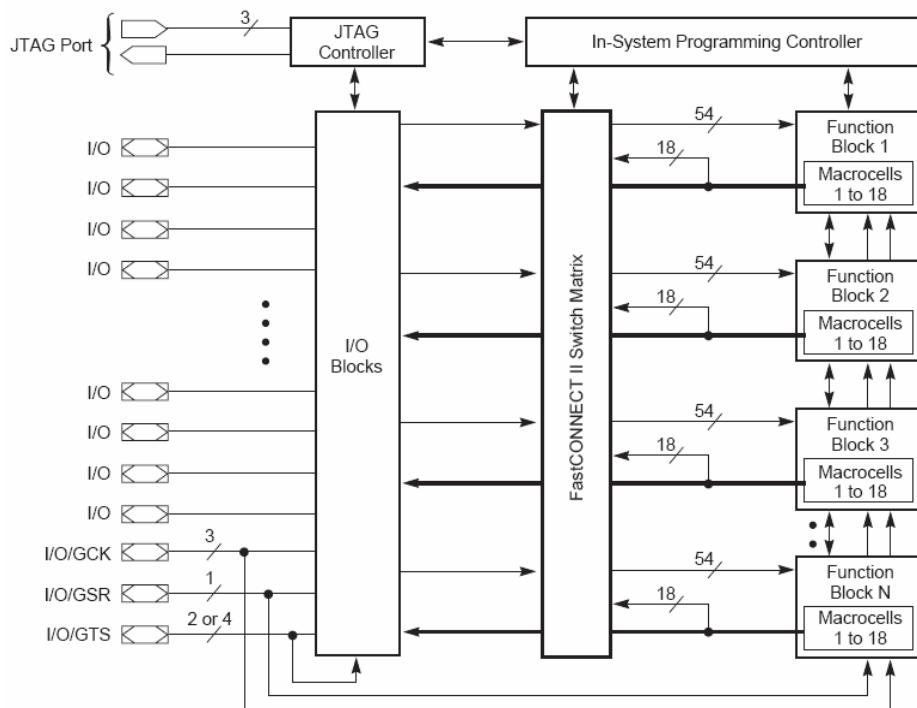
หนังสือเล่มนี้จะอธิบายการออกแบบสร้างวงจรดิจิตอลโดยใช้ FPGA และ CPLD ของบริษัท Xilinx เป็นหลัก ซึ่งผู้ใช้งานเริ่มต้นอาจไม่มีความจำเป็นต้องศึกษารายละเอียดโครงสร้างปลีกย่อยที่อยู่ภายในชิพมากนัก การออกแบบวงจรในลักษณะนี้ มีความจำเป็นต้องใช้คอมพิวเตอร์และซอฟต์แวร์ทุกตัว (Software Tool) ช่วยในการออกแบบ

1.2 CPLD

CPLD เป็นไอซีประเภท PLD ซึ่งเป็นชิปดิจิตอลนักประดิษฐ์ชนิดหนึ่งที่สามารถโปรแกรมให้มีฟังก์ชันทำงานตามที่ต้องการได้ ตัวอย่าง CPLD เบอร์ XC9536XL ที่มีความจุวงจร 800 เกตและมี 34 อินพุตเอาต์พุต (I/O) แสดงดังรูปที่ 1.2 โครงสร้างภายในชิป CPLD ตระกูล XC9500XL แสดงดังรูปที่ 1.3

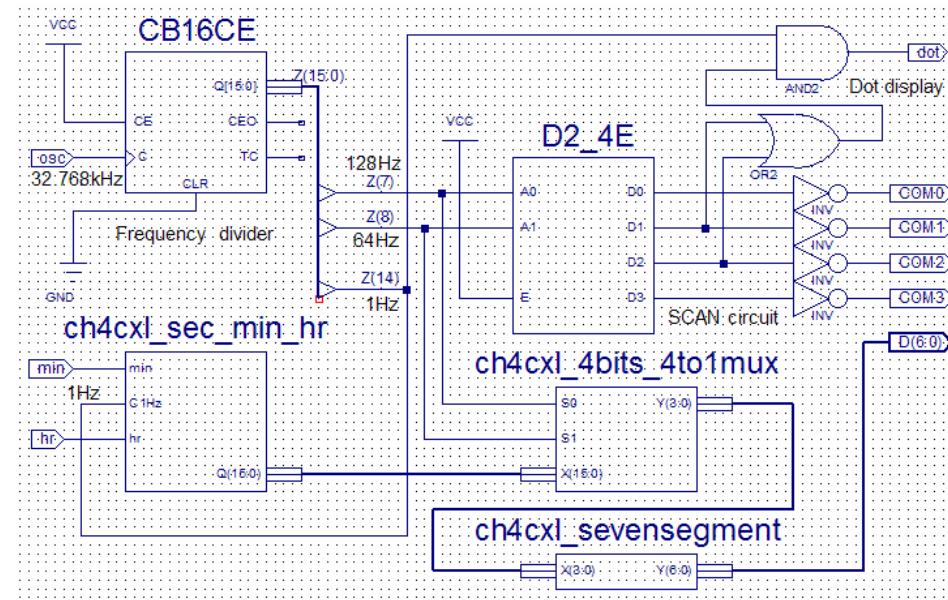


รูปที่ 1.2 ตัวอย่าง CPLD เบอร์ XC9536XL ที่มีความจุวงจร 800 เกต



รูปที่ 1.3 โครงสร้างภายในของ CPLD ตระกูล XC9500XL

โครงสร้าง CPLD จะประกอบด้วย Function Block (FB) และ I/O Block (IOB) ที่สามารถเชื่อมต่อถึงกันด้วย Switch matrix ภายใน Function Block ประกอบด้วยலอกิกพื้นฐานต่างๆ ที่สามารถโปรแกรมได้ I/O Block จะทำหน้าที่เป็นบันเฟอร์ที่อินพุตหรือเอาต์พุต วงจร In-system programming ใช้สำหรับโปรแกรม CPLD ผ่านทางพอร์ต JTAG (อ่านว่าแท็ก) โดยมี JTAG Controller เป็นตัวควบคุม ตัวอย่างวงจรนาฬิกาดิจิตอลดังรูปที่ 1.4 นั้นอาจสร้างได้ด้วยไอซีมาตรฐานตระกูล 7400 ไม่น้อยกว่า 10 ตัวแต่เมื่อสร้างวงจรนี้ด้วย CPLD เบอร์ XC9572XL จะกินความจุวงจรสูงสุดเพียง 70% เท่านั้นดังรายละเอียดแสดงในรูปที่ 1.5 CPLD มีความจุวงจรต่ำมากเมื่อเทียบกับ FPGA ซึ่งจะอธิบายในหัวข้อต่อไป โดยทั่วไป CPLD จะมีความจุวงจรไม่เกิน 10,000 เกต ข้อจำกัดของ CPLD คือใช้เวลาในการโปรแกรมค่อนข้างนานเนื่องจากมีโครงสร้างตัวเก็บข้อมูลภายในแบบ EEPROM หรือ Flash และราคาแพง (ราคายอดเยี่ยวกว่า FPGA) แต่มีโปรแกรมงานไว้ใน CPLD แล้วข้อมูลวงจรนั้นก็จะคงอยู่แม้ว่าจะไม่มีไฟเลี้ยงตัวชิปแล้วก็ตาม การโปรแกรมสามารถทำได้หลายครั้ง



รูปที่ 1.4 ตัวอย่างวงจรนาฬิกาดิจิตอล

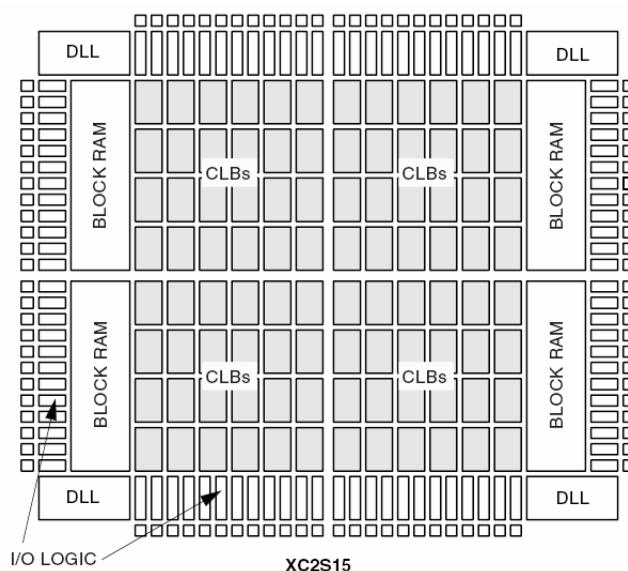
RESOURCES SUMMARY

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 เป็นชิปประเภท PLD เป็นชิปอนेकประสงค์ที่สามารถโปรแกรมให้เป็นวงจรคิดิจิตอลได้ตามที่ต้องการ แต่จะมีโครงสร้างภายในแตกต่างจาก CPLD อย่างลึกซึ้งและซับซ้อนกว่ามาก ตัวอย่างโครงสร้างภายในของ FPGA แสดงดังรูปที่ 1.6



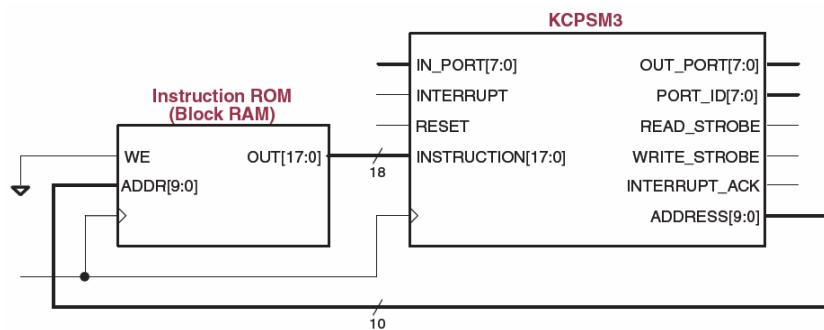
รูปที่ 1.6 โครงสร้างของ FPGA Spartan-II เบอร์ XC2S15

โครงสร้างภายใน FPGA ประกอบด้วย 2 ส่วนหลักๆ คือ Configurable Logic Block (CLB) ต่างๆ และ I/O LOGIC เพื่อใช้โปรแกรมให้เป็นวงจรคิดต่อตามที่ต้องการได้ ภายในมีอุปกรณ์ที่ทำหน้าที่เฉพาะงาน เช่น Delay-Locked Loop (DLL) และหน่วยความจำ (Block RAM) เพื่ออำนวยความสะดวกในการออกแบบวงจร FPGA มีความจุวงจรสูงมากตั้งแต่ระดับ 10,000 เกตถึงเป็นหลักร้อยล้านเกต ซึ่งขึ้นอยู่กับเทคโนโลยีที่ใช้ในการผลิต การโปรแกรม FPGA สามารถทำได้โดยโหลดข้อมูลวงจร (Configuration data) ลงไปเก็บที่เซลล์หน่วยความจำแบบ RAM (คงคล่องตัวกับ Block RAM) ที่อยู่ภายใน FPGA ดังนั้น FPGA จึงไม่มีข้อจำกัดในการโปรแกรมชั้นต่ำ และสามารถโปรแกรมข้อมูลวงจรลงไปได้เร็ว แต่มีข้อเสียที่สำคัญคือข้อมูลวงจรจะสูญหายหากไม่มีไฟเลี้ยง จึงต้องใช้หน่วยความจำภายนอกที่สามารถเก็บข้อมูลวงจรอยู่ได้แม้ว่าจะไม่มีไฟเลี้ยงตัวชิป เช่น Serial PROM เป็นต้น เพื่อใช้เก็บข้อมูล และจะมีการโหลดข้อมูลจาก Serial PROM ลงชิป FPGA โดยอัตโนมัติทุกครั้งที่มีการจ่ายไฟเลี้ยง

1.4 ข้อเปรียบเทียบระหว่าง FPGA และ CPLD กับไมโครคอนโทรลเลอร์

นักออกแบบวงจรคิดต่อส่วนใหญ่จะคุ้นเคยกับการใช้งานไมโครคอนโทรลเลอร์กันดีอยู่แล้ว เช่น ตรรกะ MCS-51 และตรรกะ PIC เป็นต้น ซึ่งมีการออกแบบสถาปัตยกรรมและชุดคำสั่งไว้ภายในเรียบร้อยแล้ว จึงสามารถนำชุดคำสั่งต่างๆ มาโปรแกรมให้ในไมโครคอนโทรลเลอร์ทำงานได้ตามที่ต้องการ ซึ่งต่างจาก FPGA และ CPLD ที่จะนำไปใช้ในงานออกแบบวงจรคิดต่อขนาดใหญ่หรือความซับซ้อนสูงๆ การออกแบบวงจรคิดต่อไว้ใน FPGA จะทำให้แรงงานมีขนาดเล็กลงมาก วงจรจะทำงานได้เร็วขึ้นและมีความเชื่อถือได้สูง เพราะสายสัญญาณต่างๆ สั้นลงและมีจุดเชื่อมวงจรต่อ กับวงจรภายนอกน้อยมาก

ปัจจุบันได้มีการออกแบบไมโครคอนโทรลเลอร์แบบฝังตัว (Embedded microcontroller) ดังรูปที่ 1.7 รวมไว้ภายใน FPGA ทำให้ได้วงจรที่ทำงานได้เร็วและมีความยืดหยุ่นสูง โดยไมโครคอนโทรลเลอร์จะกินพื้นที่วงจรใน FPGA ประมาณ 4-8% และสามารถ RUN ได้ที่ความถี่สูงมาก ซึ่งไมโครคอนโทรลเลอร์ที่สร้างจาก FPGA จะมีความเร็วสูงกว่าไมโครคอนโทรลเลอร์ทั่วไปที่มีขายในห้องทดลอง 2-20 เท่า และสร้างไว้ใน FPGA ได้พร้อมกันหลายตัว



รูปที่ 1.7 ตัวอย่างการออกแบบไมโครคอนโทรลเลอร์ตระกูลพิโคเบลซ์ฟิ๊ฟตัวใน FPGA ตระกูล Spartan-3

1.5 กระบวนการออกแบบวงจร

กระบวนการออกแบบวงจรโดยทั่วไปมักจะมีขั้นตอนหรือการทำงานเป็นขั้นตอนดังนี้

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

กระบวนการทำ System requirement คือ การหาความต้องการว่าเราต้องการสร้างอะไร งงานทำงานอย่างไร และมีอินพุตเอาต์พุตอะไรบ้าง กระบวนการทำ System design คือ การออกแบบและหาวิธีแก้ไขปัญหาจากขั้นตอนแรก จนนั้นจึงเป็นการทำ System implementation เพื่อสร้างวงจรตามที่ได้ออกแบบไว้ แล้วจึงทำการเพื่อทดสอบ (Testing) และหากมีปัญหา ก็ทำการแก้ไข (Debugging) จนนั้นจึงทำ 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 เกือบทั้งหมดมีอนกับ ISE 10.1i แต่ ISE 10.1i จะรองรับ FPGA ตระกูลล่าสุดได้แก่ Vertex-5 และ Spartan-3A เป็นต้น รวมทั้งคุณสมบัติใหม่ๆ บางประการที่เราแทบจะไม่ได้ใช้ และ ISE10.1i จะใช้หน่วยความจำบนชาร์ดดิสก์มากกว่า ISE8.1i ประมาณ 2 เท่า ISE 8.1i มี 2 ตัวด้วยกัน 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 บนระบบปฏิบัติการต่างๆ ได้ดังตารางที่ 1.1 และตารางที่ 1.2 โดยคระกูลชิพที่ใช้ได้กับ ISE 8.1i สรุปดังตารางที่ 1.3 และขนาด RAM อย่างน้อยที่สุดของคอมพิวเตอร์ที่แนะนำให้ใช้กับ WebPACK 8.1i และ Foundation 8.1i สรุปได้ดังตารางที่ 1.4 คือต้องไม่น้อยกว่า 256-512 MB ส่วน ISE10.1i มี 2 ตัว เช่นกัน 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 อย่างน้อยที่สุดของคอมพิวเตอร์ควรไม่น้อยกว่า 256-512 MB แต่อย่างไรก็ตามในทางปฏิบัติมักมีการเปิดใช้โปรแกรมอื่นรวมอยู่ด้วย ดังนั้นทั้ง ISE8.1i และ ISE10.1i ควรใช้ RAM ไม่น้อยกว่า 1-2 GB

ตารางที่ 1.1 ระบบปฏิบัติการที่ใช้กับ ISE Foundation 8.1i

ISE Foundation™
Microsoft Windows 2000 / XP
Sun Solaris 2.8 or 2.9
Red Hat Enterprise Linux 3 (32 & 64 bit)

ตารางที่ 1.2 ระบบปฏิบัติการที่ใช้กับ ISE WebPACK 8.1i

ISE WebPACK™
Microsoft Windows 2000 / XP
Red Hat Enterprise Linux 3 (32 bit)

ตารางที่ 1.3 ตระกูลของชิพ FPGA และ CPLD ที่ซอฟต์แวร์ทุกรองรับ

Family	ISE WebPack (MS Windows & Linux)	ISE Foundation (MS Windows, Linux, Linux64, & Solaris)
Virtex devices	Up to V600	All
Virtex-E devices	Up to V600E	All
Virtex-II devices	Up to 2V500	All
Virtex-II Pro devices	2VP2, 2VP4, 2VP7	All
Virtex-II Pro X devices	No	All

ตารางที่ 1.3 ตระกูลของชิพ FPGA และ CPLD ที่รองรับ (ต่อ)

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

ตารางที่ 1.4 ขนาด RAM อย่างน้อยที่สุดของคอมพิวเตอร์ที่แนะนำให้ใช้กับซอฟต์แวร์ทุล

Xilinx Device	RAM
<ul style="list-style-type: none"> • XC9500™/XL/XV • Automotive 9500XL • CoolRunner™/ CoolRunner™II • Automotive CoolRunner™-II • Spartan™-II XC2S15 through XC2S200 • Spartan™-IIE XC2S50E through XC2S600E • Automotive Spartan™-IIE AC2S50E through AC2S300E • Spartan™-3 XC3S50 through XC3S1000 • Automotive Spartan™-3 A3S50 through X3S1000 • Spartan™-3E XC3S100E through XC3S1200E • Virtex™XCV50 through XCV800 • Virtex™-E XCV50E through XCV600E • Virtex™-II XC2V40 through XC2V1000 • Virtex™-II PRO XC2VP2 through XC2VP7 • Virtex™-4 <ul style="list-style-type: none"> ♦ XC4VLX15 through XC4VLX25 ♦ XC4VFX12 through XC4VFX20 ♦ XC4VSX25 	256-512 Megabytes
<ul style="list-style-type: none"> • Spartan™-3 XC3S1500 through XC3S5000 • Virtex™ XCV1000 • Virtex™-E XCV1000E through XCV3200E • Virtex™-II XC2V1500 through XC2V6000 • Virtex™-II PRO XC2VP20 through XC2VP70 • Virtex™-II PRO X XC2VPX20 through XC2VPX70 • Virtex™-4 <ul style="list-style-type: none"> ♦ XC4VLX40 through XC4VLX100 ♦ XC4VFX40 through XC4VFX100 ♦ XC4VSX35 through XC4VSX55 	1-2 Gigabyte

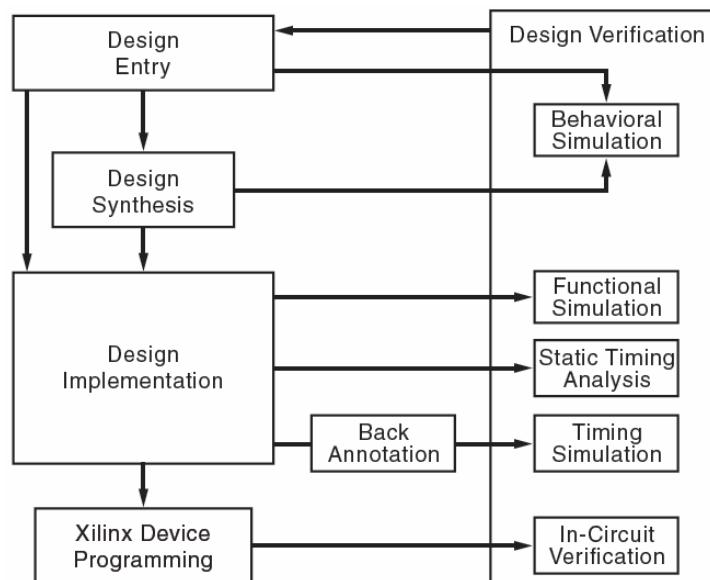
แม้ว่า WebPACK 8.1i จะเป็น Limited version ของ Foundation 8.1i ตามที่สรุปในตารางที่ 1.3 และตารางที่ 1.5 นั้น ถือว่าเพียงพอสำหรับการอุดกแบบวงจรขนาดใหญ่ทั่วๆ ไป โดยมีความสามารถที่เพิ่มเข้ามาคือ ISE Simulator, CORE Generator และ FPGA editor

ตารางที่ 1.5 คุณสมบัติและความสามารถที่ซอฟต์แวร์ทุกร่องรับ

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 ขั้นตอนการออกแบบวงจรดิจิตอลด้วย FPGA หรือ CPLD

ขั้นตอนการออกแบบวงจรดิจิตอลด้วย FPGA หรือ CPLD มีการทำงานหลายขั้นตอน รายละเอียดแสดงดังรูปที่ 1.8

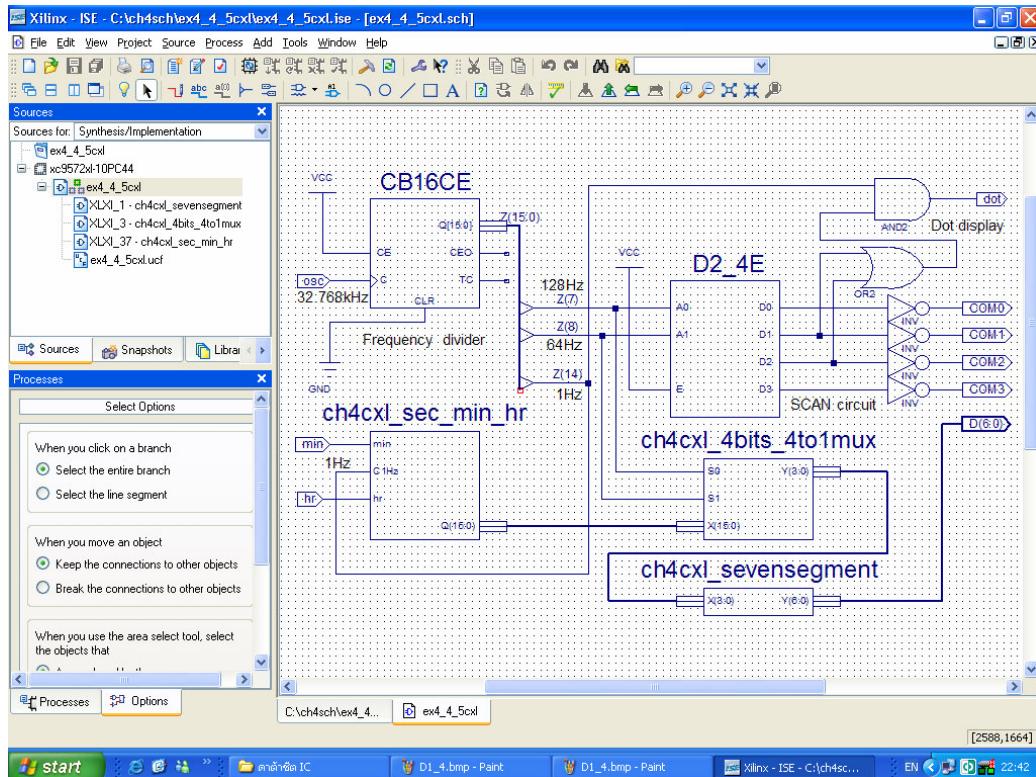


รูปที่ 1.8 ขั้นตอนการออกแบบวงจรดิจิตอลด้วย FPGA หรือ CPLD

1) การออกแบบวงจร

การออกแบบวงจร (Design entry) นั้นสามารถเลือกออกแบบด้วยวิธีการต่างๆ ได้แก่ วิธีวาดผังวงจร (Schematic) ดังรูปที่ 1.9 หรือออกแบบด้วยภาษาาระดับสูงที่เรียกว่า HDL (Hardware Description Language) เช่น VHDL (หรือ Verilog) และแสดงดังรูปที่ 1.10 หรือ อาจออกแบบด้วยวิธีการเขียนแผนภูมิสถานะ (State diagram) เพื่อให้ได้วงจรตามที่ต้องการ

การเขียนโค้ดของวงจรที่ซับซ้อนนั้นปกติจะเขียนโค้ดในระดับ RTL (Register transfer level) หรือในระดับที่ต่ำกว่าเพื่อให้สามารถสังเคราะห์ง่ายได้ ในการเขียนโค้ดระดับ RTL นั้นเราจะแบ่งวงจรที่ซับซ้อนออกเป็นบล็อกของวงจรย่อยๆ ตาม และเพื่อให้มองเห็นการไหลเวียนข้อมูลระหว่างรีจิสเตอร์โดยผ่านทางวงจรคอมมิเนชัน ซึ่งจะอธิบายในข้อที่ 2.23 ในบทที่ 2



รูปที่ 1.9 การออกแบบวงจรด้วยวิธีวาดผังวงจร (Schematic)

Xilinx - ISE - C:\ch4v\IC100UP\IC100UP.ise - [C100UP.vhd]

File Edit View Project Source Process Window Help

Sources Sources for: Synthesis/Implementation

C100UP
 x9572xl-10PC44
 C100UP - Behavioral
 C100UP.ucf

Processes

Add Existing Source Create New Source Design Utilities User Constraints Implement Design

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

entity C100UP is
    Port ( OSC : in STD_LOGIC;
           PB1 : in STD_LOGIC;
           CLR : in STD_LOGIC;
           DOT : out STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (6 downto 0);
           COM : out STD_LOGIC_VECTOR (1 downto 0));
end C100UP;

architecture Behavioral of C100UP is
    signal C,S,C_DB,CLR_DB : STD_LOGIC;
    signal Q_temp : STD_LOGIC_VECTOR (14 downto 0);
    signal Q0,Q1,A : STD_LOGIC_VECTOR (3 downto 0);
begin
    DB : process(OSC,PB1,CLR_DB)
        begin
            if CLR_DB='0' then
                C <= '0';
            elsif C_DB'event and C_DB='1' then
                if PB1='0' then
                    C <= '1';
                end if;
            end if;
        end process;
    C_DB <= not C and OSC;
    F_DIV : process (OSC)
        begin
            if OSC'event and OSC='1' then

```

รูปที่ 1.10 การออกแบบจรด้วยภาษา VHDL

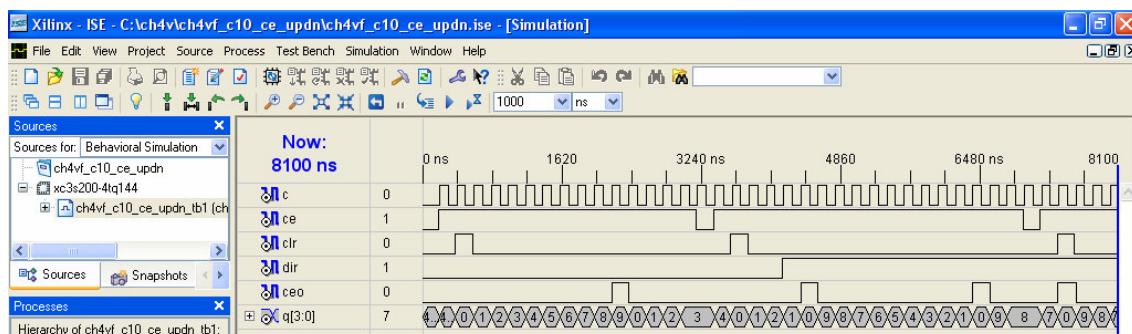
2) การสังเคราะห์วงจร

การสังเคราะห์วงจร (Design synthesis) คือ ขั้นตอนการแปลงโค้ดวงจรที่เขียนด้วยภาษาาระดับสูงให้เป็นวงจรระดับเกต (Gate level) ที่เขียนเป็น Text file ที่เรียกว่า Netlist file ถ้าใช้ Xilinx Synthesis Tool หรือ XST ก็จะได้ NGC file โค้ดที่นำไปสังเคราะห์วงจรนี้โดยปกติจะเป็นโค้ดในระดับ RTL ส่วนการออกแบบโดยใช้วิธีคาดผังวงจรซึ่งเป็นระดับเกตอยู่แล้วนั้นจะถูกเขียนในรูป Netlist file เช่นกัน โดยที่ NGC file จะประกอบด้วยไฟล์วงจรลอจิกที่ออกแบบ (Logical design data file) และเงื่อนไขมังคบ (Constraints file) เพื่อบอกให้ซอฟต์แวร์ทุกทำตามเงื่อนไขที่ผู้ออกแบบต้องการ ถ้าใช้ซอฟต์แวร์ทุกจากผู้พัฒนาซอฟต์แวร์ทุก หรือ Vendor รายอื่นก็จะเป็น EDIF file (อ่านว่า อีดีเอฟ ไฟล์) ซึ่งเป็น Industry standard file ใน การสังเคราะห์วงจรนี้เราอาจจะต้องมีการระบุว่าเป็น FPGA หรือ CPLD เพื่อป้อนค่าพารามิเตอร์ต่างๆ ให้กับทุกสังเคราะห์วงจร (Synthesis Tool) และในการสังเคราะห์วงจรนี้เราสามารถเลือกได้ว่าจะเน้นความเร็ว (Speed) หรือประหยัดจำนวนเกต (Area)

3) การตรวจสอบความถูกต้องของวงจร

การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design verification) เป็นการนำโค้ด HDL ของวงจรที่ออกแบบไปตรวจสอบความถูกต้องด้วยโปรแกรมจำลองการทำงาน (Simulation) โดยใช้ซอฟต์แวร์ทุก เช่น ISE Simulator หรือ ModelSim ตัวอย่างสัญญาณที่ได้จากการจำลองการทำงานแสดงดังรูปที่ 1.11 การจำลองการทำงานโดยทั่วไปมี 3 ระดับคือ

- Behavioral simulation เป็นการจำลองเดินพัฒนากรรมของวงจร โดยยังไม่คิดถึงโครงสร้างของวงจรภายในเพื่อให้ได้แบบจำลองการทำงานเบื้องต้น ซึ่งโค้ด HDL ที่เขียนนี้อาจจะนำไป Synthesis ไม่ได้แต่เขียนโค้ดได้รวดเร็ว
- Functional simulation เป็นการนำโค้ดในระดับ RTL ซึ่งเป็นโค้ดที่สามารถนำไปสังเคราะห์วงจรได้ไปจำลองการทำงานเพื่อตรวจสอบความถูกต้องของวงจรที่ออกแบบ
- Timing simulation เป็นการจำลองการทำงานที่ใกล้เคียงกับ Hardware จริงมากที่สุด เนื่องจากเป็นการนำข้อมูล Timing ที่เกิดขึ้นใน FPGA มาใช้ ทำให้สามารถตัดสินใจเลือกเบอร์ FPGA ได้ถูกต้องก่อนตัดสินใจซื้อ



รูปที่ 1.11 สัญญาณที่ได้จากการจำลองการทำงานของวงจรที่ออกแบบ

ในหนังสือเล่มนี้เราจะใช้ซอฟต์แวร์ทุก ISE Simulator ที่ติดตั้งมาพร้อมกับ ISE 8.1i-10.1i ไปใช้ในการจำลองการทำงาน โดยที่การจำลองการทำงานในเบื้องต้นนี้จะใช้ Waveform Editor ซึ่งจะอธิบายและมีตัวอย่างในข้อ 2.16.2 ข้อ 2.17.2 และในบทที่ 3 ถึงบทที่ 5 ส่วนการจำลองการทำงานที่มีความซับซ้อนนั้นจำเป็นต้องเขียน Testbench ซึ่งจะอธิบายในบทที่ 6

ISE Simulator ที่ติดตั้งมาพร้อมกับ ISE WebPACK นั้นจะเป็น Limit version ที่รองรับการเปลี่ยนโค้ดต่างๆ รวมกันได้ประมาณ 50,000 บรรทัด หากเกินจากนี้การคำนวณต่างๆ จะทำได้ช้ามาก

4) Design implementation

Design implementation ของ FPGA นั้นจะแปลง Netlist file เป็น Physical file มีขั้นตอนดังนี้

- Translate เป็นขั้นตอนรวมไฟล์วงจรโลจิกที่ออกแบบชิ่งอยู่ในรูปไฟล์ Netlists และ Constraints เข้าด้วยกัน
- Map เป็นขั้นตอนเลือกอุปกรณ์จากไฟล์ที่ได้จากขั้นตอน Translate หรือ NGD file ไปวางภายในหรือจับคู่กับอุปกรณ์พื้นฐานต่างๆ ที่มีอยู่ภายใน FPGA โดยจะเขียนอยู่ใน NCD file
- Place and Route เป็นขั้นตอนคำนวณหาตำแหน่งที่เหมาะสมเพื่อนำมาวางจริงๆ (Place) โดยจะขึ้นอยู่กับ Timing constraints และเมื่อวางเรียบร้อยแล้วจึงเชื่อมต่อสัญญาณต่างๆ เข้าด้วยกัน (Route)
- Programming file generation เป็นขั้นตอนสร้างไฟล์ Bitstream เพื่อให้เหมาะสมสำหรับการดาวน์โหลดลง FPGA

ส่วนกรณี CPLD นั้นมีความซับซ้อนน้อยกว่า FPGA มาก จึงเรียกรวมขั้นตอนหลังจากการแปลง (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) ไปดาวน์โหลดลงชิพโดยใช้เครื่องโปรแกรมหรือใช้ดาวน์โหลดเคเบิลก็ได้

1.7 การติดตั้งซอฟต์แวร์ทุก

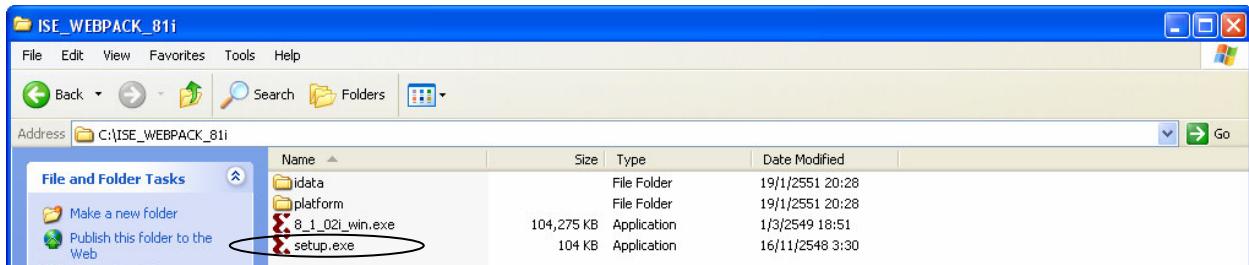
1) ขั้นตอนการขอและดาวน์โหลดซอฟต์แวร์

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 รวมทั้งไฟล์บทที่ 2 ของหนังสือเล่มนี้ที่เป็นภาพถ่ายเพื่อความสะดวกในการอ่านโดยไฟล์เกี่ยวกับอรรถทดลองรุ่นต่างๆ และข้อมูลไอซีที่เกี่ยวข้อง

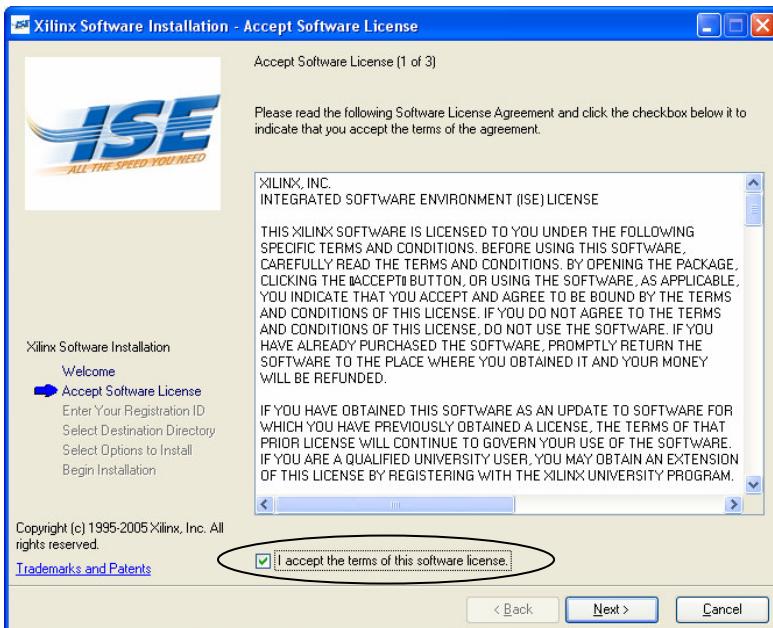
สำหรับซอฟต์แวร์ทุก ISE WebPACK 10.1i หรือเวอร์ชันล่าสุดสามารถดาวน์โหลดได้ฟรีที่ <http://www.xilinx.com> โดยจะต้องลงทะเบียนก่อน จากนั้นทาง Xilinx จะขึ้นชื่อ User name และ Pass word โดยล็อกมาทางอีเมล์ แล้วนำไป Sign in เพื่อดาวน์โหลดซอฟต์แวร์ทุก ถ้ามีไฟล์ ISE WebPACK 10.1i แล้วก็อาจพานา Registration ID ไปใช้เฉพาะตอนติดตั้งซอฟต์แวร์ทุก

2) วิธีการติดตั้ง ISE WebPACK 8.1i

- 1) ทำการ Uninstall ซอฟต์แวร์เวอร์ชันก่อน (ถ้าเคยติดตั้งมาแล้ว) ทิ้ง จากนั้น Copy ไฟล์ใน Folder ชื่อ ISE_WebPACK_81i จากแผ่น DVD ไปไว้ในไดร์ฟ C เมื่อต้นเบิลคลิก 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 แล้วจะได้หน้าต่างคลิกไป (ทำซ้ำจนครบ 3 ครั้ง) เสร็จแล้วคลิกปุ่ม Next ไปอีก 2 ครั้งและคลิกปุ่ม Install แล้วโปรแกรมจะเริ่มติดตั้งไฟล์ต่างๆ ลงในคอมพิวเตอร์ ดังรูปที่ 1.14



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

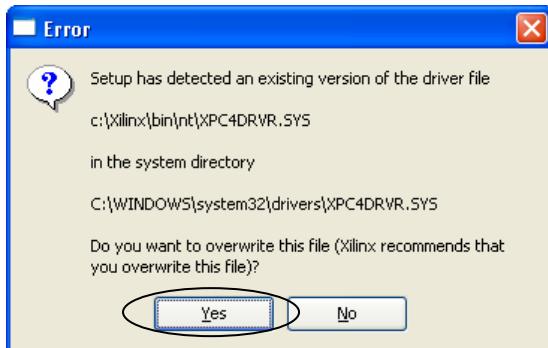


รูปที่ 1.13 หน้าต่าง Accept Software License (มี 3 หน้า)



รูปที่ 1.14 เริ่มติดตั้งซอฟต์แวร์

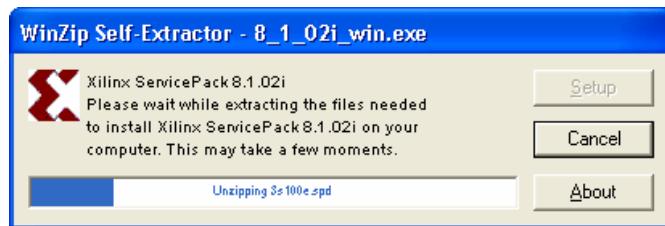
3) ในการนี้ที่คอมพิวเตอร์เคยติดตั้งซอฟต์แวร์ไว้แล้วก่อนหน้านี้ในระหว่างติดตั้งอาจมีหน้าต่าง Error ปรากฏขึ้นดังรูปที่ 1.15 ให้คลิก Yes แล้วจะได้หน้าต่างกดไปเพื่อทำการ Setup เมื่อ Setup แล้วเสร็จให้คลิก OK ก็จะถือว่าติดตั้งซอฟต์แวร์แล้วเสร็จ



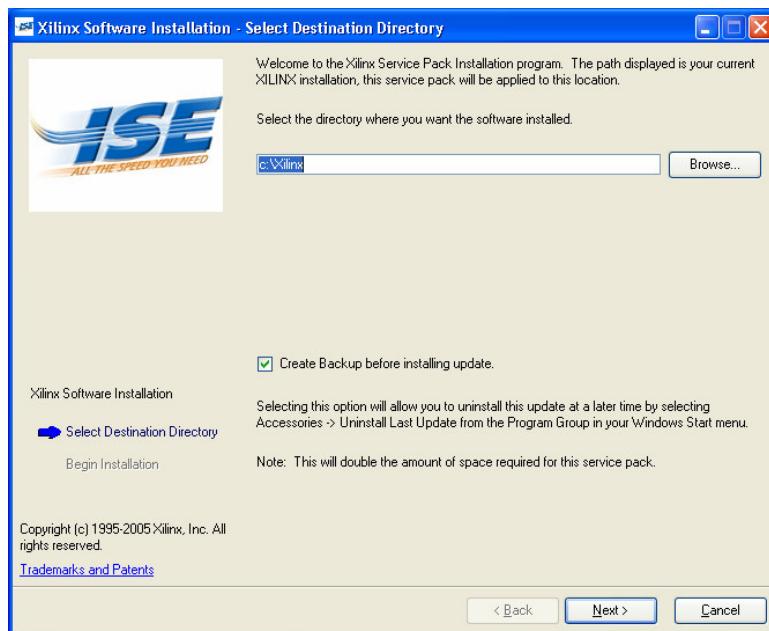
รูปที่ 1.15 ยกเลิกไฟล์ Driver ของซอฟต์แวร์เวอร์ชันเก่า

3) วิธีการติดตั้ง Service Pack 2

3.1) ในรูปที่ 1.12 จะมีไฟล์ Service Pack 2 (ชื่อ 8_1_02i_win.exe) อยู่ จากนั้นค้นเบิลคลิกไฟล์ 8_1_02i_win.exe แล้วจะได้ดังรูปที่ 1.16 แล้วโปรแกรมจะทำการแตกไฟล์ .exe ออก โดยอัตโนมัติ 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 เมื่อบэкอัพไฟล์เสร็จแล้วให้คลิก OK จากนั้นคลิก Install เพื่อเริ่มติดตั้งซอฟต์แวร์ดังรูปที่ 1.18 ไปจนแล้วเสร็จ คลิก OK ก็จะถือว่าการติดตั้งซอฟต์แวร์แล้วเสร็จสมบูรณ์

หมายเหตุ การใช้ซอฟต์แวร์ ISE WebPACK 8.1i นั้นมีความจำเป็นต้องติดตั้งซอฟต์แวร์ชื่อ Adobe Reader 7 ขึ้นไปควบคู่กันไปด้วยเพื่อใช้สำหรับอ่านไฟล์ .pdf ดังนั้นหากยังไม่ได้ติดตั้งก็ให้ผู้ใช้ติดตั้งซอฟต์แวร์ชื่อ Adobe Reader นี้ด้วย

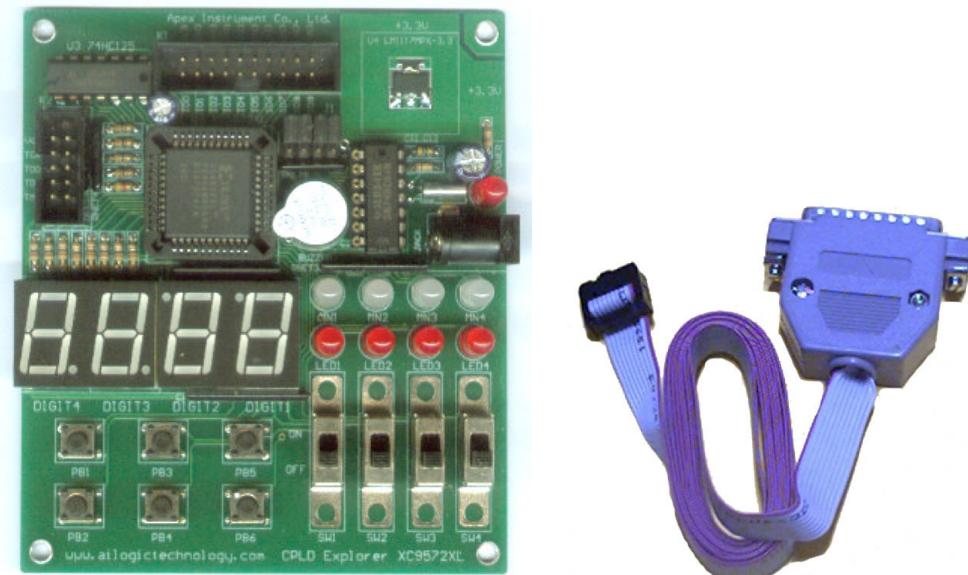


รูปที่ 1.18 หน้าต่าง Begin Installation ขณะกำลังติดตั้ง

1.8 บอร์ดทดลอง

1.8.1 CPLD Explorer XC9572XL

บอร์ดทดลองonen กประสังค์รุ่น CPLD Explorer XC9572XL เป็นบอร์ดทดลองที่เหมาะสมสำหรับผู้เริ่มต้นศึกษาการออกแบบวงจรดิจิตอลด้วย FPGA และ CPLD มีรายละเอียดแสดงดังรูปที่ 1.19 คุณสมบัติทั่วๆไปของบอร์ดทดลองเป็นดังนี้



รูปที่ 1.19 CPLD Explorer XC9572XL พร้อมสายดาวน์โหลด

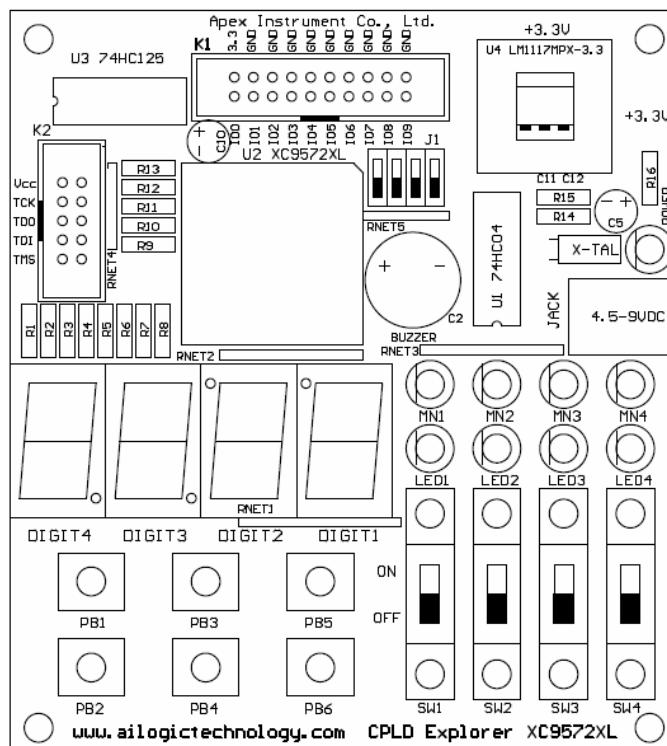
1.1) คุณสมบัติทั่วไป

- CPLD เบอร์ XC9572XL (1,600 เกต) แบบ PLCC 44 ขา (PC44) Speed Grade -10
- เซเว่นเซกเมนต์ 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

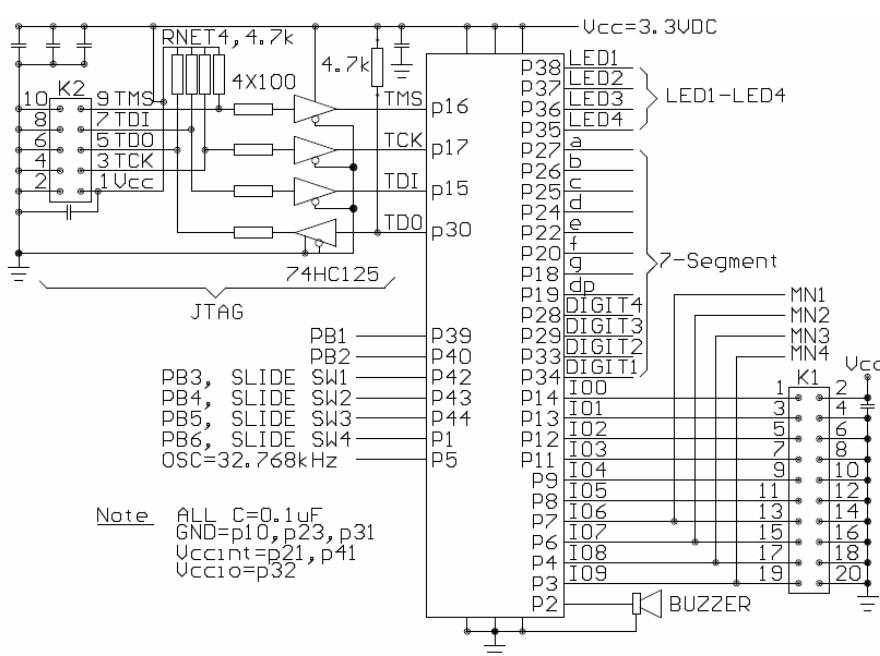
- Push button switch 6 ตัว และ Slide switch 4 ตัว (ใช้ร่วมกับ Push button switch)
- พอร์ต K1 เป็น I/O 10 บิต ซึ่งสามารถใช้กับ I/O 5 V.(ตระกูล TTL, 74HCT00,...) และ 3.3 V. ได้
- Onboard Oscillator 32.768 kHz

1.2) หลักการทำงานของบอร์ด

รายละเอียดการจัดวงจรปั๊มนอร์ดเป็นดังรูปที่ 1.20 โดยที่การจัดวง I/O ต่างๆ แสดงดังรูปที่ 1.21 และตารางที่ 1.6



รูปที่ 1.20 การจัดวงจรปั๊มนอร์ด CPLD Explorer XC9572XL



รูปที่ 1.21 การจัดวง I/O ต่างๆ ของ CPLD Explorer XC9572XL

ตารางที่ 1.6 แสดงตำแหน่งขาของชิป CPLD ที่ต่ออยู่กับอาร์ดแวร์ภายนอกที่อยู่บนบอร์ด

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 สтанะ 4 ดวง คือ LED1-LED4 โดยต่อขา cathode (Cathode) ลงกราว์ดและต่อขา anode (Anode) เข้ากับ I/O ของ CPLD โดยมีตัวด้านหน้า RNET2 470 ไอห์มต่ออนุกรมอยู่เพื่อจำกัดกระแสไฟฟ้าเดสก์ดังรูปที่ 1.22a) ดังนั้นถ้า CPLD ส่งลอจิก ‘1’ จะทำให้ LED ดวงนั้นติดสว่าง

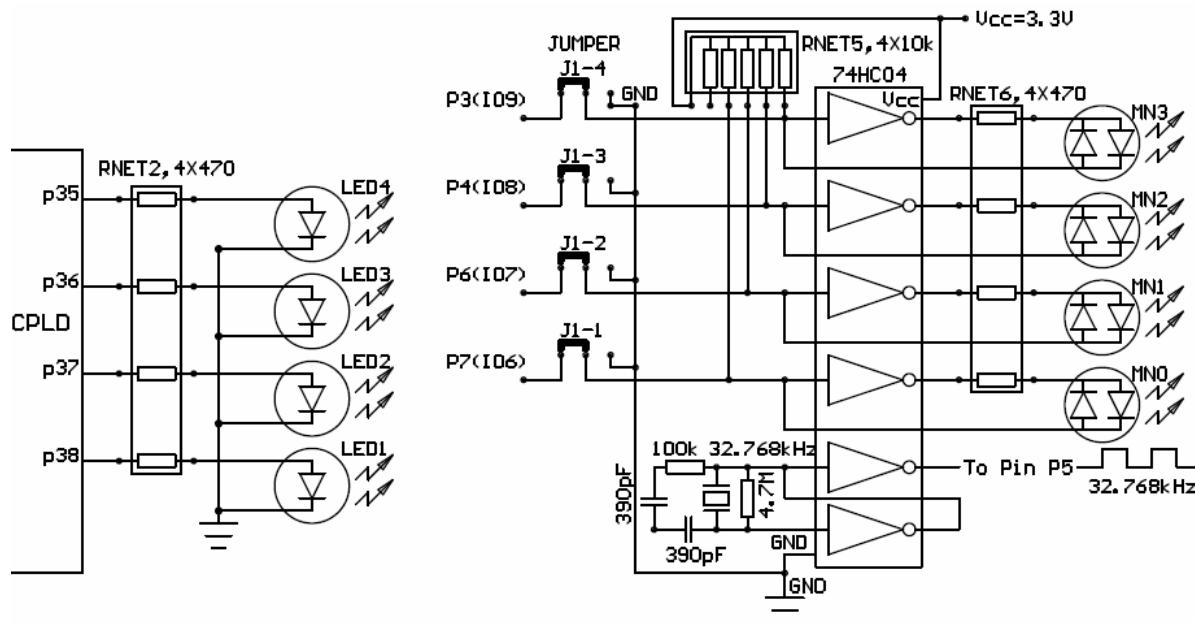
2) Logic monitor เป็น LED แสดงผล 3 สтанะ 4 ดวง คือ MN1-MN4 (J1-1 ถึง J1-4) ต่อพ่วงกับ I/O ของ CPLD ที่คอนเนกเตอร์ K1 (I/O6-I/O9) โดยใช้ Jumper J1 ดังรูปที่ 1.22b) ถ้า I/O ของ CPLD เป็นลอจิก ‘1’ LED ติดเป็นสีเหลือง แต่ถ้าเป็น ‘0’ จะติดเป็นสีแดง แต่ถ้าเป็น High impedance ก็จะดับ (ดับเกือบสนิทเนื่องจากมี RNET5 10 kΩ ไอห์มต่อพูลอัพเข้ากับ Vcc)

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

4) ออก (Buzzer) จำนวน 1 ตัว ซึ่งต่อขาบล๊อกกราว์ดและต่อขาอีกข้างซึ่งเป็นขั้วบวกกับขาอินพุตเอ้าต์พุต (I/O) p2 ของ CPLD ดังนั้นถ้า CPLD ส่งลอจิก ‘1’ จะทำให้ออกดัง หากจะให้ออกดับก็ส่งเป็นลอจิก ‘0’

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

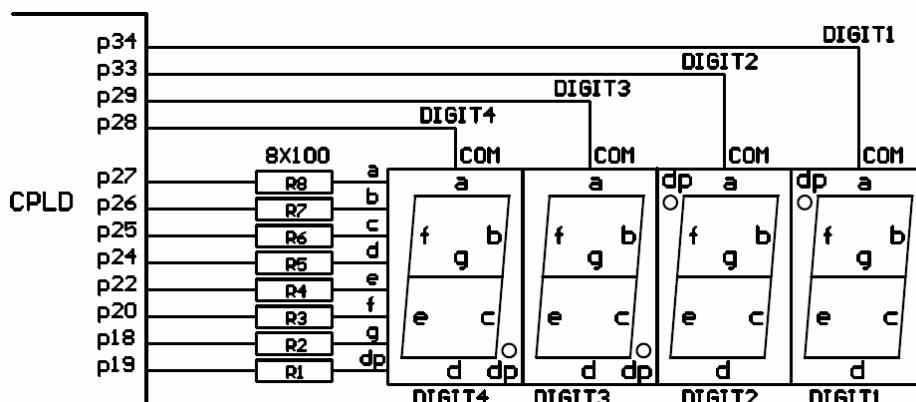
ไม่น้อยกว่า 30 ครั้ง \times 4 หลัก = ความถี่ 120 Hz จึงจะไม่เห็นการกระพริบ ตัวต้านทาน R1-R8 ใช้ในการจำกัดกระแส ส่วนที่ตัวแสดงผล DIGIT2 และ DIGIT1 จะมีการออกแบบกลับหัวให้จุด (dp) ขึ้นไปอยู่ด้านบนเพื่อการแสดงเครื่องหมาย ":" (Colon) ในการทำงานพิเศษหรือแสดงผลของค่าของอุณหภูมิ แต่การแสดงตัวเลขต่างๆ ยังคงใช้สายสัญญาณเดียวกับ DIGIT3 และ DIGIT4



1.22a) การต่อ LED เข้ากับขาของ CPLD

1.22b) วงจร LED Logic monitor และ 32.768kHz Oscillator

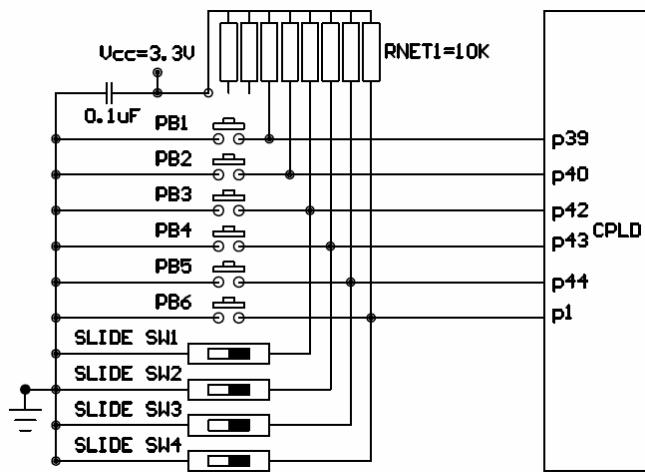
รูปที่ 1.22 การต่อ LED เข้ากับขาของ CPLD, วงจร Logic trainer และ 32.768kHz Oscillator



รูปที่ 1.23 ตัวแสดงผลเซเว่นเซกเมนต์ (7-Segment) จำนวน 4 หลัก

ด้านอินพุตของบอร์ดทดลองเป็นดังนี้

- สวิตช์กดติดปล่อยดับ (Push button Switch) อยู่ 6 ตัวคือ PB1-PB6 ต่ออยู่กับขาของ CPLD และมีสไลด์สวิตช์ (Slide Switch) อีก 4 ตัว คือ Slide SW1-Slide SW4 ต่อพ่วงอยู่กับ PB3-PB6 ตามลำดับแสดงดังรูปที่ 1.24 ซึ่งหากไม่กดสวิตช์จะให้ลอจิก ‘1’ แต่ถ้ากดจะให้ลอจิก ‘0’ เมื่อจากมีตัวต้านทานพูลอัพ RNET1 ต่ออยู่ และหากสไลด์สวิตช์ไปอยู่ที่ OFF จะให้ลอจิก ‘1’ แต่ถ้า ON จะให้ลอจิก ‘0’ การกด PB3 ค้างไว้จะเหมือนกับการเลื่อนสไลด์สวิตช์ SW1 ขึ้นไปที่ ON

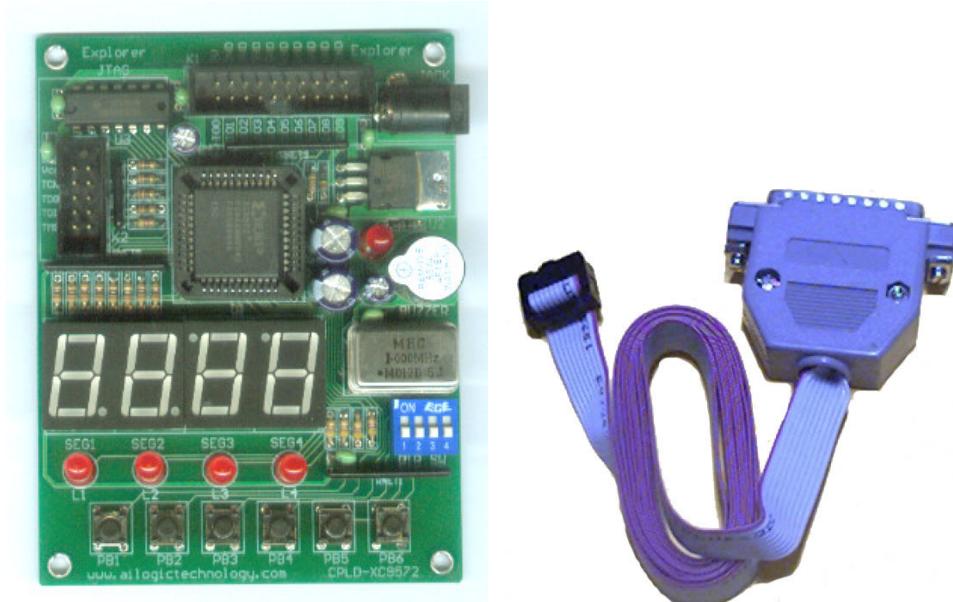


รูปที่ 1.24 สวิตչ์กดติดปล่อยดับและสไลด์สวิตช์

2) JTAG คอนเนกเตอร์ K2 ใช้เพื่อโปรแกรมข้อมูลงาน (Configuration data) ลง CPLD ผ่านทางพอร์ตขบวน (Parallel Port) ของคอมพิวเตอร์ โดยขาสัญญาณทุกเส้นจะต่อผ่านบีฟเฟอร์เบอร์ 74HC125 เพื่อป้องกันการรบกวนในสาย JTAG ส่วน R10–R13 ต่อໄໄว์เพื่อลดสัญญาณสะท้อนในสาย JTAG และก่อนโปรแกรมงานลงชิพจะต้องมีการทำกำหนดขา CPLD เป็นตามตารางที่ 1.6 จากนั้นต่อสาย JTAG และไฟเลี้ยงเข้าบอร์ด ในกรณีที่ใช้บอร์ดทดลองนี้เป็นเครื่องโปรแกรมนั้นจะใช้ได้กับเบอร์ XC9536XL และ XC9572XL ที่มีขาแบบ PLCC 44 ขา

1.8.2 CPLD Explorer XC9572

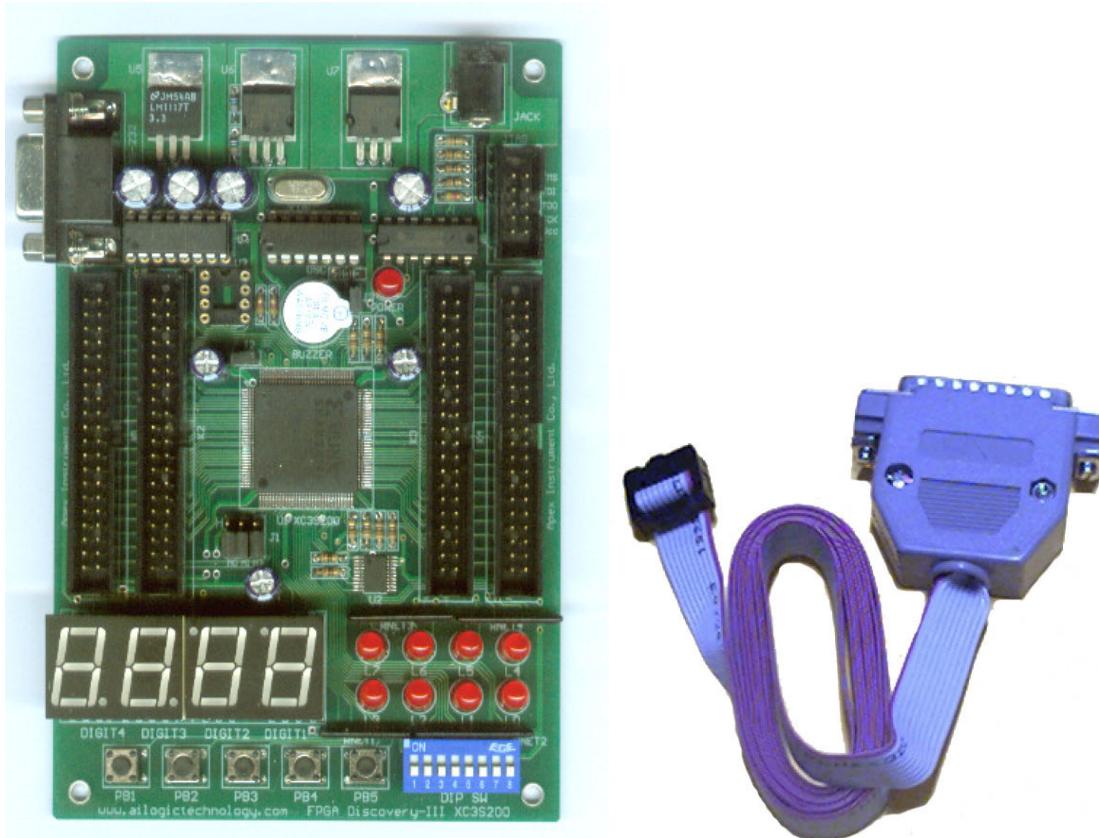
บอร์ดทดลองonenกประสงค์ CPLD Explorer XC9572 แสดงดังรูปที่ 1.25 เป็นบอร์ดทดลองที่มีคุณสมบัติทั่วไปเหมือนกับบอร์ด CPLD Explorer XC9572XL และมีรายการ I/O ตรงกันทุกประการตามตารางที่ 1.6 แต่จะแตกต่างตรงที่จะไม่มี Logic monitor 4 ดวงและมี I/O เป็นระบบ 5V ใช้ DIP Switch แทน Slide Switch และใช้ Oscillator ความถี่ 1 MHz แทน 32.768 kHz และสามารถใช้เป็นเครื่องโปรแกรม CPLD ได้เช่นเดียวกันคือเบอร์ 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 บอร์ดมีอุปกรณ์อำนวยความสะดวกที่เพียงพร้อมด้วยอุปกรณ์อินพุตเอาต์พุต อ่าย่างครบครัน จึงเหมาะสมสำหรับ LAB ออกแบบวงจรดิจิตอลและออกแบบ ไอซีขั้นสูง



รูปที่ 1.26 บอร์ดทดลองonenกประสงค์ FPGA Discovery-III XC3S200

1.1) ຄູນສົມບັດທິ່ວໄປ

บอร์ดทดลองonenกประสงค์ FPGA Discovery-III XC3S200 มີ 2 ຮຸນຕ້າຍກັນເຄືອ

- FPGA Discovery-III XC3S200F (ຮຸນ 200,000 ເກຕ)
- FPGA Discovery-III XC3S200F4 (ຮຸນ 400,000 ເກຕ)

บอร์ด FPGA Discovery-III XC3S200F และ FPGA Discovery-III XC3S200F4 จะມີລັກຂະພະເໜີອັນກັນທຸກປະກາດ ແຕ່ທີ່ແຕກຕ່າງກັນເຄືອ FPGA Discovery-III XC3S200F จะໃຊ້ FPGA ຕະຫຼອດ Spartan-3 ຂອງ Xilinx ເບອ້ XC3S200-4TQ144C ຊົ່ງເປັນ FPGA ພາດຄວາມຈຸງຈາກ 200,000 ເກຕ, Package ແບບ TQ144, Speed Grade:4 ແລະ Platform Flash PROM ເບອ້ XCF01SVO20C (ທີ່ສາມາດໂປຣແກຣມຂໍ້ອມຸນຸງຈາກຫຳໄດ້ ສິ່ງ 20,000 ຄຽ້ງ) ໃນຂະນະທີ່ FPGA Discovery-III XC3S200F4 ຈະໃຊ້chip FPGA ເບອ້ XC3S400-4TQ144C ຊົ່ງເປັນ FPGA ພາດ 400,000 ເກຕ Package ແບບ TQ144, Speed Grade:4 ແລະ Platform Flash PROM ເບອ້ XCF02SVO20C ບອດທິດສົມບັດ FPGA Discovery-III XC3S200 ມີຄູນສົມບັດທິ່ວໄປນີ້

- 7-Segment จำนวน 4 หลัก (ใช้ร่วมกับ Expansion ports และสามารถอ่านออกได้)
- 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 บิต
- 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)

1.2) คุณสมบัติที่สำคัญของชิป FPGA ตระกูล Spartan-3 เบอร์ XC3S200

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

1.3) คุณสมบัติที่สำคัญของชิป FPGA ตระกูล Spartan-3 เบอร์ XC3S400

- ความจุวงจร 400,000 เกต
- 18Kb block RAM จำนวน 16 ชุด (รวม 288Kb)
- 18x18 hardware multiplier จำนวน 16 ชุด
- Digital Clock Manager (DCM) จำนวน 4 ชุด

18Kb block RAM เป็นหน่วยความจำที่มีความเร็วสูงมาก (โดยประมาณ) 200 MHz ที่มีอยู่ในชิพสามารถรองรับได้เป็น RAM หรือ ROM ที่มีขนาดต่างๆ กัน ได้รวมทั้งทำเป็น FIFO ได้ รูปที่ 1.27 แสดงขนาดของ RAM ที่สร้างจาก Block RAM

Organization	Memory depth	Data width	Parity width
512x36	512	32	4
1Kx18	1K	16	2
2Kx9	2K	8	1
4Kx4	4K	4	-
8Kx2	8K	2	-
16Kx1	16K	1	-

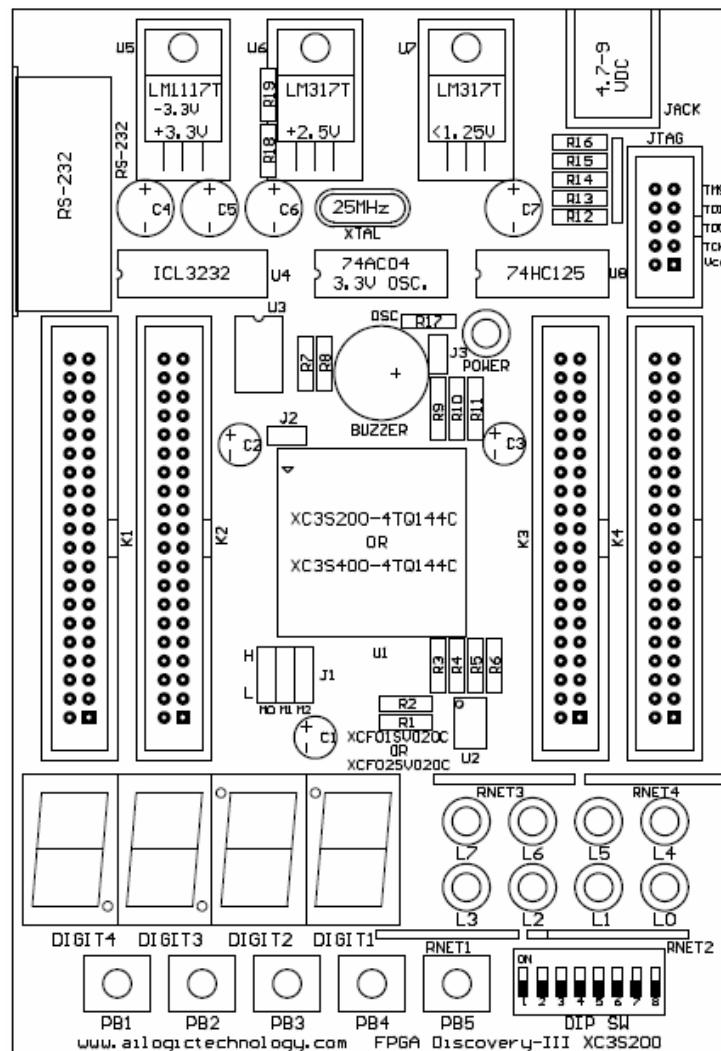
รูปที่ 1.27 RAM แบบ Single Port ขนาดต่างๆ ที่สร้างจาก Block RAM และลักษณะ

Digital Clock Manager (DCM) เป็นวงจรที่ช่วยจัดการเกี่ยวกับสัญญาณนาฬิกาและเลื่อนเฟสเมื่อจำนวน 4 ชุด ทำให้การออกแบบวงจรจ่ายขึ้นเนื่องจากสามารถสร้างความถี่ต่างๆ ได้จากอสซิลเลเตอร์ภายนอกเพียงชุดเดียว และอาจต้องต่อสัญญาณของอสซิลเลเตอร์เข้ากับ DCM ที่ทำงานในหน้าที่ดังต่อไปนี้

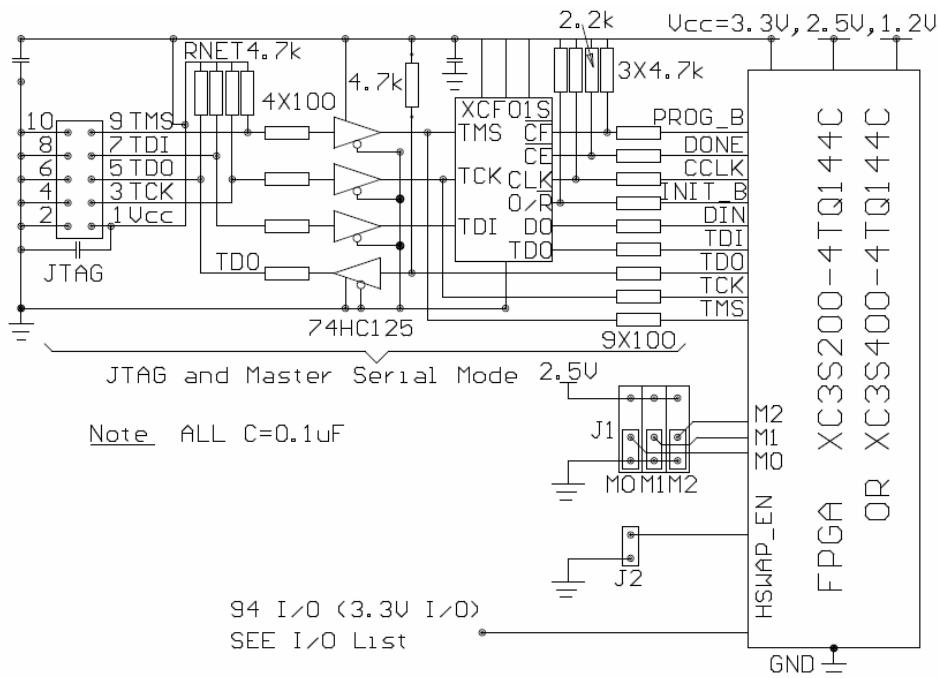
- หารความถี่ (Clock Divider) เป็นวงจรหารซึ่งจะให้ความถี่เอาต์พุตเท่ากับความถี่อินพุตหารด้วยตัวเลขดังต่อไปนี้ คือ 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 9, 10, 11, 12, 13, 14, 15, หรือ 16 ตามลำดับ
- สร้างความถี่สองเท่า (Clock Doubler) เป็นวงจรซึ่งจะให้ความถี่ที่เอาต์พุตจะเป็น 2 เท่าของความถี่อินพุต
- Digital Frequency Synthesizer (DFS) เป็นวงจรซึ่งสามารถกำหนดให้ความถี่เอาต์พุตเท่ากับผลคูณของความถี่อินพุต กับอัตราส่วนของ M/D โดยที่ $M=2$ ถึง 32 และ $D=1$ ถึง 32 เช่น ต้องการความถี่ 66.6666 MHz แต่ต้องกำหนดให้ $M=8$ และ $D=3$ บนบอร์ดนี้คือ 25 MHz เป็นสัญญาณอินพุตให้กับ DFS ดังนั้นต้องกำหนดให้ $M=8$ และ $D=3$
- Quadrant Phase Shift เป็นวงจรเลื่อนเฟส 90°, 180° และ 270° องศา ตามลำดับ
- Fine Phase Shift เป็นวงจรใช้ในการเลื่อนเฟสอย่างละเอียด มีความละเอียดอยู่ที่ 1/255 เท่าของความถี่

1.4) หลักการทำงานของบอร์ดอเนกประสงค์

บอร์ดทดลองอเนกประสงค์รุ่น FPGA Discovery-III XC3S200 มีการจัดวางอุปกรณ์ดังรูปที่ 1.28 รายละเอียดการจัดวาง I/O (โดยย่อ) แสดงดังรูปที่ 1.29 และอุปกรณ์ที่ต่ออยู่กับบอร์ด FPGA (I/O List) ตามตารางที่ 1.7



รูปที่ 1.28 การจัดวางตำแหน่งการวางอุปกรณ์ด้านบน



รูปที่ 1.29 การจัดวาง I/O ต่างๆของ FPGA Discovery-III XC3S200 (โดยย่อ) และวงจรสำหรับดาวน์โหลด

ตารางที่ 1.7 รายละเอียดของอุปกรณ์ที่ต่ออยู่กับขา FPGA (I/O List)

7-Segment	FPGA Pinout	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	DIGIT1 , COMMON CATHODE
DIGIT2	p33	DIGIT2 , COMMON CATHODE
DIGIT3	p36	DIGIT3 , COMMON CATHODE
DIGIT4	p41	DIGIT4 , COMMON CATHODE

Push button	FPGA Pinout	Descriptions
PB1	p44	Push button No. 1
PB2	p46	Push button No. 2
PB3	p47	Push button No. 3
PB4	p50	Push button No. 4
PB5	p51	Push button No. 5

EEPROM	FPGA Pinout	Descriptions
I2C-SCL	p128	24LCXX
I2C-SDA	p129	24LCXX

RS232	FPGA Pinout	Descriptions
TX	p131	ICL3232CP
RX	p132	ICL3232CP

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	25MHz , GCLK6

BUZZER	FPGA Pinout	Descriptions
BUZZER	p125	BUZZER

ตารางที่ 1.7 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 1.7 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 1.7 รายละเอียดของอุปกรณ์ที่ต่ออยู่กับขา FPGA (I/O List) (ต่อ)

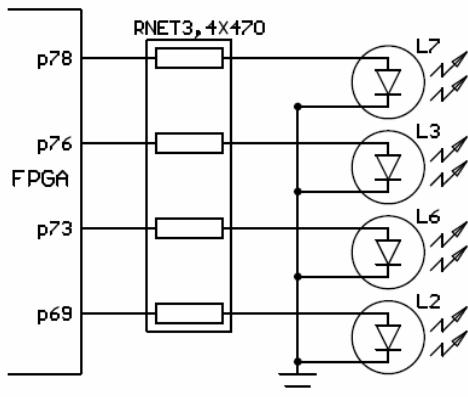
K3 CONNECTOR					
Descriptions	FPGA Pinout	K3 Pinout	K3 Pinout	FPGA Pinout	Descriptions
I/O , BUZZER	p124	40	39	-	GND
I/O	p122	38	37	-	GND
I/O	p118	36	35	-	GND
I/O	p113	34	33	-	GND
I/O	p108	32	31	-	GND
I/O	p105	30	29	-	GND
I/O	p103	28	27	-	GND
I/O	p100	26	25	-	GND
I/O	p98	24	23	-	GND
I/O	P96	22	21	-	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 1.7 รายละเอียดของอุปกรณ์ที่ต่ออยู่กับขา FPGA (I/O List) (ต่อ)

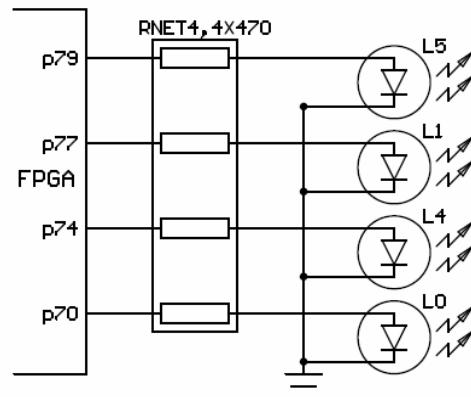
K4 CONNECTOR					
Descriptions	FPGA Pinout	K4 Pinout	K4 Pinout	FPGA Pinout	Descriptions
I/O	p125	40	39	-	GND
I/O	p123	38	37	-	GND
I/O	p119	36	35	-	GND
I/O	p116	34	33	-	GND
I/O	p112	32	31	-	GND
I/O	p107	30	29	-	GND
I/O	p104	28	27	-	GND
I/O	p102	26	25	-	GND
I/O	p99	24	23	-	GND
I/O	p97	22	21	-	GND
I/O	p95	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

ด้านเอาต์พุตของทดลองเป็นดังนี้

1) LED จำนวน 8 ดวง คือ LED0–LED7 โดยต่อขาเออโนนเดเข้ากับ I/O ของ FPGA โดยมี $R = 470$ โอห์ม คือ RNET3 และ RNET4 จำกัดกระแสสูงสุดที่ 1.30 LED ติดเมื่อเป็น ‘1’ เนื่อง I/O ของ LED และบาง I/O ของคอนเนคเตอร์ K3 และ K4 ต่อพ่วงกันอยู่ ดังนั้นหากไม่ต้องการใช้ LED ก็สามารถหักด้าบทาบนแบบเนตเวิร์กออกได้



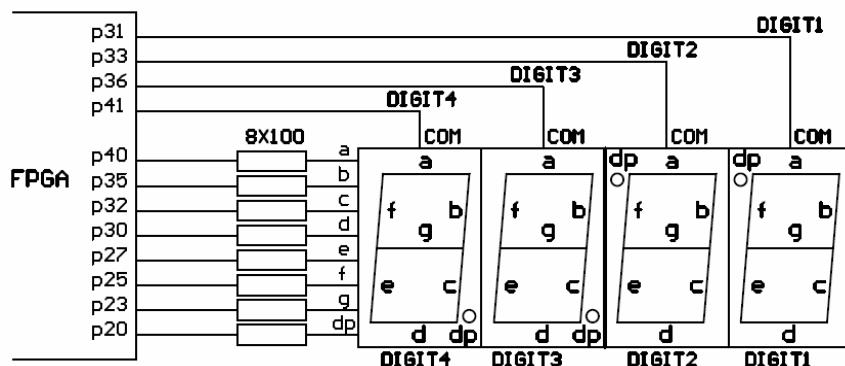
1.30a) การต่อ LED เข้ากับ FPGA



1.30b) การต่อ LED เข้ากับ FPGA

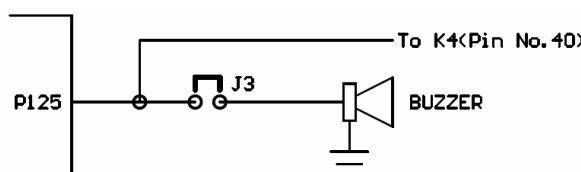
รูปที่ 1.30 การต่อ LED เข้ากับขาต่างๆของ FPGA

2) ตัวแสดงผลเซเว่นเซกเมนต์ (7-Segment) 4 หลัก คือ DIGIT4-DIGIT1 โดยที่ทุกหลักจะใช้สายสัญญาณข้อมูลร่วมกันแต่สาย公共 (Common) แต่ละหลักจะแยกออกจากกันแสดงดังรูปที่ 1.31 การแสดงตัวเลขบนตัวแสดงผลทั้ง 4 ตัวพร้อมกันจะใช้เทคนิคการแสดงความเร็วไม่น้อยกว่า 30 ครั้งx4 หลัก = 120 ครั้งต่อวินาที หากต้องการให้เซกเมนต์ไดติดกีต้องส่ง ‘1’ ที่สายสัญญาณพร้อมกับส่ง ‘0’ ที่ขา COM ของหลักนั้น ตัวแสดงผล DIGIT2 และ DIGIT1 ลูกหมุนให้จุด (dp) ขึ้นไปอยู่ด้านบนเพื่อการแสดง “:” (Colon) ของนาฬิกาหรือแสดงอุณหภูมิเป็นองศาโดยยังคงใช้สายสัญญาณเดียวกับสองตัวแรก



รูปที่ 1.31 ตัวแสดงผลเซเว่นเซกเมนต์ (7-Segment) จำนวน 4 หลัก

3) Buzzer จำนวน 1 ตัว ต่อขั้ว梧กับ I/O ของ FPGA ดังรูปที่ 1.32 ถ้า FPGA ส่งลอจิก ‘1’ จะทำให้ Buzzer ดัง และเนื่อง I/O ที่ใช้ขับ Buzzer ต่อพ่วงกับ I/O ขาที่ 40 ของคอนเนคเตอร์ K3 ดังนั้นหากไม่ใช้ Buzzer ก็ให้หัก Jumper J3 ออกได้

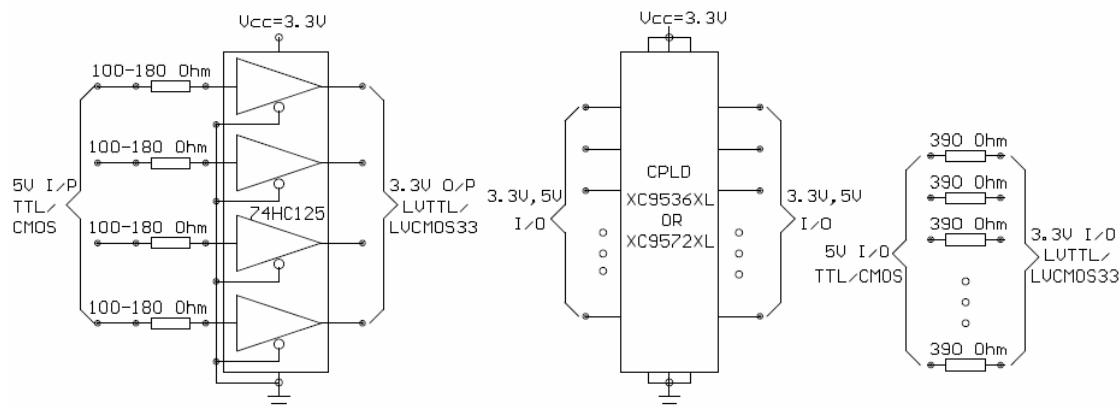


รูปที่ 1.32 การต่อ Buzzer กับขา FPGA

4) อินพุต/เอาต์พุตพอร์ต (I/O) ที่ค่อนเนนค酮อร์ K1-K4 มีรายละเอียด I/O ดังตารางที่ 1.7 เนื่องจากไฟเลี้ยง V_{CCO} ของ I/O ทั้งหมดเท่ากับ 3.3V จึงสามารถเลือก I/O ได้เพียง 2 ชนิด คือ LVCMOS33 หรือ LVTTL ดังตารางที่ 1.8 ซึ่งจะเป็น I/O ระบบ 3.3V ดังนั้นถ้า I/O ของ FPGA เป็นเอาต์พุตก็จะสามารถขับอินพุตของระบบ 3.3V และระบบ 5V ได้ (เช่น ตระกูล 74LS00, 74HCT) แต่ถ้า I/O ของ FPGA เป็นอินพุตก็จะใช้ได้เฉพาะระบบ 3.3V เท่านั้น การเรื่อมต่อระบบ 5V เข้ากับอินพุตของ FPGA มีตัวอย่างแสดงดังรูปที่ 1.33a ถึงรูปที่ 1.33c ซึ่งในรูปที่ 1.33c นั้นจะใช้ความต้านทานประมาณ 390 โอห์ม (ควรเลือกใช้ตัวต้านทานที่มีค่าอินเด็กデンซ์ไฟฟ้าในตัว เช่น R +/- 5% 1/W) เพื่อจำกัดกระแสไฟให้เข้ากับอินพุตของ FPGA ถ้าค่าของความต้านทานต่ำก็อาจทำให้อินพุตเสียหายได้ ซึ่งปกติกระแสไม่ควรเกิน 5 mA (ควรดูรายละเอียดจาก Data sheet ของ FPGA)

ตารางที่ 1.8 Single-ended I/O standards (Values in volts)

Signal Standard	V _{CCO}		V _{REF} for Inputs	Board Termination Voltage (V _{TT})
	For Outputs	For Inputs		
GTL	$\geq V_{TT}$	$\geq V_{TT}$	0.8	1.2
GTL	$\geq V_{TT}$	$\geq V_{TT}$	1	1.5
HSTL_I	1.5	-	0.75	0.75
HSTL_III	1.5	-	0.9	1.5
HSTL_I_18	1.8	-	0.9	0.9
HSTL_II_18	1.8	-	0.9	0.9
HSTL_III_18	1.8	-	1.1	1.8
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	-	-



1.33a) ต่อผ่านบีฟเฟอร์ 3.3V

1.33b) ต่อผ่าน CPLD ที่มี I/O 3.3V

1.33c) ต่อผ่านตัวต้านทานอนุกรม

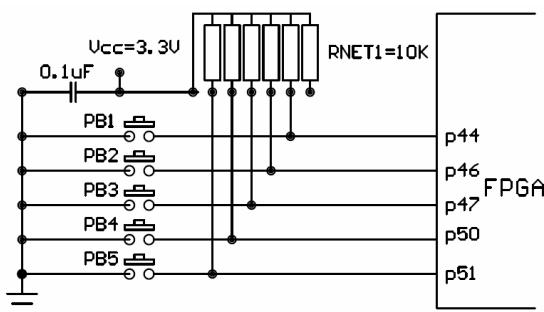
รูปที่ 1.33 ตัวอย่างการต่อเอาต์พุต 5 V กับอินพุตของ FPGA

เนื่องจากค่อนเนนค酮อร์ K1-K4 ถูกออกแบบให้มีสายสัญญาณและกราวด์อยู่ต่ำกว่าไฟเลี้ยงเพื่อแก้ปัญหาการรบกวนข้ามช่องสัญญาณ (Cross talk) ดังนั้นการต่อสายสัญญาณ I/O ออกจากค่อนเนนค酮อร์ K1-K4 นั้นถ้าต้องการใช้งานที่ความถี่สูงๆ ควรใช้สายแพร์ (Flat Cable) โดยที่ความยาวสูงสุด (ของเส้นสายทองแดงของ PCB และสายแพร์รวมกัน) ที่ไม่เกิดการสะท้อน (หรือ Transmission Line effect) $\leq (2''/\text{nS}) \times \text{ช่วงเวลาขึ้น} (\text{Rise time})$ เมื่อ O/P ของ FPGA เป็น Fast slew rate มี Rise time $\leq 1 \text{ nS}$ และ Slow slew rate จะมี Rise time $\leq 3 \text{ nS}$ ดังนั้น $(2''/\text{nS}) \times 3 \text{ nS} = 6''$ (ประมาณ 15 เซนติเมตร) ถ้าที่สายแพร์ยาวกว่านี้จะต้อง

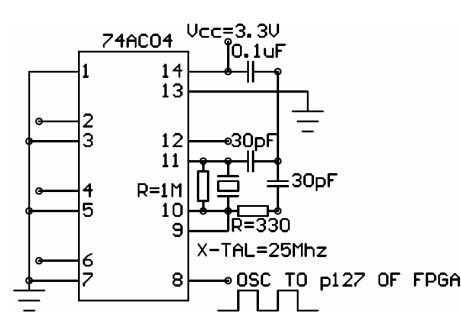
Terminate គ້າວິທີທີ່ເໜາະສມ ກາຮຕ່ອນອົບທົດລອງເຂົ້າກັນອົບທົດກາຍນອກນັ້ນຄ້າສາຍແພຣມີ V_{CC} ຮວມອູ່ຄ້ວຍຈະຕັ້ງຕ່ອດວັກນິປະປະຈຸ 0.1 μF ແລະ 10nF ແນບມັດຕິເລີຍອົບຫົວ Chip capacitor ເຂົ້າກັບຂົ້ວ V_{CC} ແລະ ກາວົດ (ກາວົດເພລນ) ທີ່ບໍ່ອົບນັ້ນຄ້ວຍເພື່ອໃຫ້ສາຍ V_{CC} ມີຄຸນສົນບັດຖາງໄຟຟ້າແນບ AC ເສມືອນວ່າເປັນກາວົດ ຄ້າເກີດອອສົມເລເຕໃນສາຍຂອງ I/O ກີ່ໃຊ້ຕ້ານທານຄ່າ 56-220 ໂອໜີມຕ່ອງອຸນຸກຮມເຂົ້າກັນ I/O ເພື່ອໜ່ວງອອສົມເລເຕ ຄ້າຄ່າຄວາມຕ້ານທານສູງເກີນໄປຈະທຳໃໝ່ງຈະທຳການຂ້າລົງ

ທາງຕ້ານອິນພຸດນອົບທົດລອງເປັນດັນນີ້

- 1) ສວິທັກົດຕິດປ່ອຍດັນ (Push button Switch) 5 ຕັວ ອື່ນ PB1-PB5 ຕ່ອກັນຂາຂອງ FPGA ລາກໄໝກົດສວິທັກົດຈະເປັນລອຈິກ ‘1’ ແຕ່ຫາກົດຈະເປັນລອຈິກ ‘0’ ເນື່ອງຈາກມີຕ້ານທານ RNET1 ຕ່ອພູລອພອູ່ດັງຮູປ໌ 1.34
- 2) ອອສົມເລເຕອ່ຽ (Oscillator) ບນບອົບກວາມຄໍ 25 Mhz ມີ 1 ຜຸດດັງຮູປ໌ 1.35 ໂດຍຕ່ອກັນຂາ p127 (GCLK6) ສາມາຮັດສ້າງກວາມຄໍເອົ້ນໆ ໂດຍໃໝ່ DCM ໄດ້ ສາມາຮັດໄສ່ອອສົມເລເຕອ່ຽ 3.3 ໂວລດກວາມຄໍທີ່ຕ່ອງການໄດ້ໂຄຍຄອດໄອົບ 74AC04 ອອກ

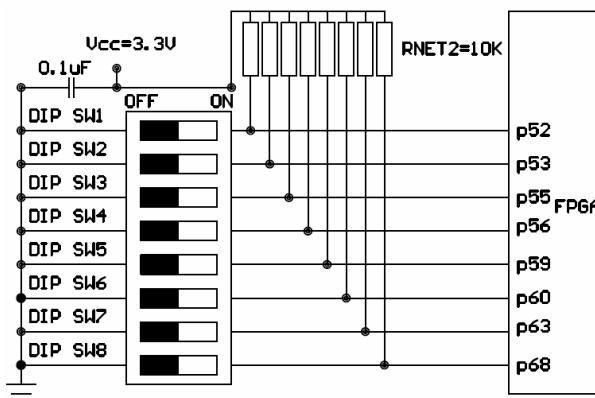


ຮູປ໌ 1.34 ສວິທັກົດຕິດປ່ອຍດັນ



ຮູປ໌ 1.35 ວົງຈຽອອສົມເລເຕອ່ຽ

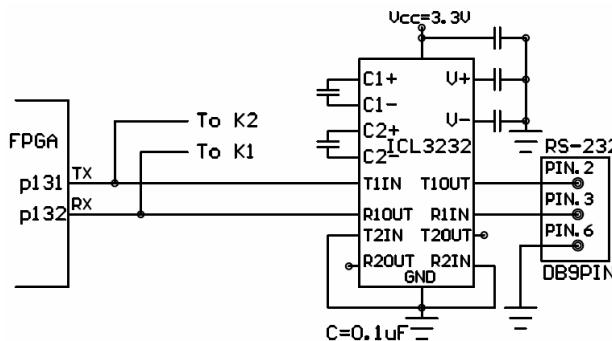
- 3) ດີພລວິທີ (Dip Switch) ບນບອົບກວາມຄໍນີ້ມີ 8 ສວິທີ ໂດຍປົກທາກເລື່ອນສວິທີສົ່ງໄປທີ່ OFF ຈະທຳໃໝ່ໄດ້ລອຈິກ ‘1’ ແລະ ຫາກເລື່ອນສວິທີເປັນໄປທີ່ ON ຈະທຳໃໝ່ໄດ້ລອຈິກ ‘0’ ເນື່ອງຈາກມີຕ້ານທານ RNET2 ຕ່ອພູລອພອູ່ດັງຮູປ໌ 1.36



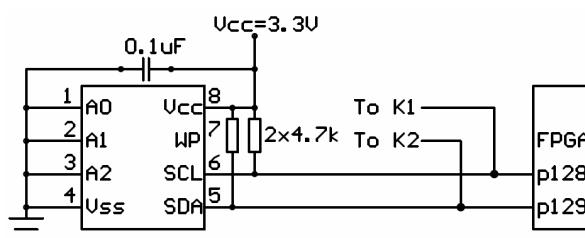
ຮູປ໌ 1.36 ວົງຈຽອອດີພລວິທີ

- 4) ພອ້ຽຕອນຸກຮມ (RS-232C) ໃຊ້ໄອົບເບອ່ຽ ICL3232CP (U4) ຈຶ່ງເປັນຮະບນ 3.3V ເພື່ອໃຫ້ສາມາຮັດຕ່ອກັນ I/O ຂອງ FPGA ໄດ້ໂຄຍຕຽງໂດຍມີວົງຈແສດງດັງຮູປ໌ 1.37 ແລະ ເນື່ອງຈາກ I/O ຂອງພອ້ຽຕອນຸກຮມຕ່ອງພ່ວງອູ່ກັບຫາທີ່ 35 ຂອງ K1 ແລະ ຫາທີ່ 39 ຂອງ K2 ດັນນັ້ນຄ້າຕໍ່ອັນການໃຊ້ຂາດັ່ງກ່າວຂອງ K1 ແລະ K2 ເພື່ອວັດຖຸປະສົງກໍເອີ້ນກີ່ສາມາຮັດທຳໄດ້ໂຄຍຄອດໄອົບ ICL3232CP ອອກໄກ້

- 5) ຂອຄເກີ້ດ 8 ຂາ (U3) ລຳທັບໄສໄອົບ EEPROM ແນບ I²C ເຊັ່ນ ເບອ່ຽ 24LC256 ເພື່ອເກັ່ນຂໍ້ມູນລັ້ນຈະຕ່ອດພ່ວງກັບຫາທີ່ 39 ຂອງ K1 ແລະ K2 ຕາມຄໍາດັບ ແສດງດັງຮູປ໌ 1.38 ມີຕ້ານທານ R7 ແລະ R8 ບනາດ 4.7 ກິໂລໂໜ້ມຕ່ອງ Pull up ຈຶ່ງຄ້າຕໍ່ອັນການໃຊ້ຂາທີ່ 39 ຂອງ K1 ແລະ K2 ກີ່ທຳໄດ້ໂຄຍຄອດໄອົບອອກ (ແຕ່ບ້ານຄົງສກາພ Pull up ລາກໄໝ່ຕັດ R7 ແລະ R8 ອອກໄປ)



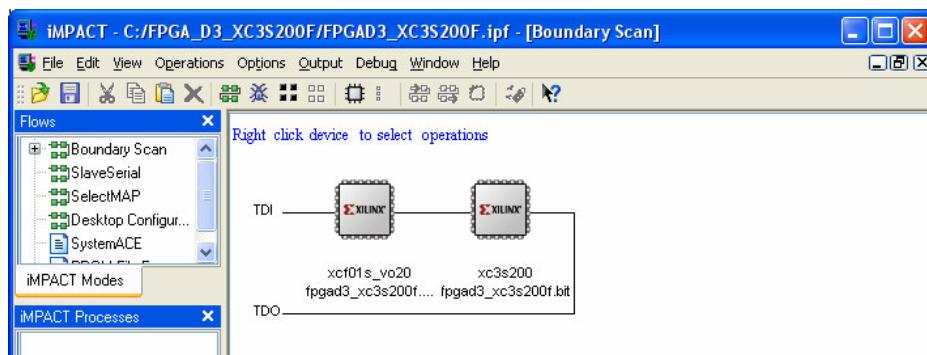
รูปที่ 1.37 พортอนุกรม (RS-232C)

รูปที่ 1.38 พортอนุกรมแบบ I²C

6) JTAG คอนนคเตอร์ใช้สำหรับต่อสายดาวน์โหลด (JTAG Cable) เข้ากับพอร์ตบนาน (Printer Port) ของคอมพิวเตอร์ เพื่อโปรแกรมข้อมูลวงจร (Configuration data) ลง FPGA มีรายละเอียดวงจรดังรูปที่ 1.29 ซึ่งได้ถูกออกแบบให้โปรแกรม Serial Flash PROM และ FPGA ได้ทั้งใน Master Serial Mode และ JTAG Mode (Boundary Scan Mode) ซึ่งในขั้นตอนดาวน์โหลดนี้ที่จอกомพิวเตอร์จะประภากู้ชิพ Platform Flash PROM และ FPGA พร้อมกันแสดงดังรูปที่ 1.39 เพราะมีการออกแบบให้ โหมด JTAG ต่อถึงกันและวนแบบลูปโซ่ เราจึงสามารถดาวน์โหลดข้อมูลลงชิพทั้งสองตัวหรือตัวใดตัวหนึ่งก็ได้ และเนื่องจาก ในตารางที่ 1.9 นั้น Jumper J1 ถูกเซตใน Master Serial Mode คือ M0, M1, M2 = "000" ดังนั้นเพื่อป้องกันการโปรแกรม FPGA ข้อผิดพลาดเนื่องจากการ โปรแกรมผิดโหมด (JTAG) ทาง Xilinx แนะนำใช้ไฟล์ใน Serial Flash PROM ที่ส่งก่อนทุกครั้ง หากไฟล์ใน Serial Flash PROM เป็นคนละไฟล์ (ไฟล์ไม่เหมือนกัน) กับไฟล์ที่จะโปรแกรมลง FPGA

ตารางที่ 1.9 รายการเซตโหมดในการโปรแกรม FPGA

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



รูปที่ 1.39 ขั้นตอนดาวน์โหลดที่จอกомพิวเตอร์จะประภากู้ชิพ Flash PROM และ FPGA พร้อมกัน

1.8.4 บอร์ดทดลองรุ่นอื่นๆ

สำหรับบอร์ดทดลองรุ่นอื่นๆ นั้นผู้ใช้ควรจะต้องศึกษาข้อมูลจากคู่มือบอร์ดทดลองรุ่นนั้นๆ การออกแบบวงจรและการโปรแกรมวงจรที่ออกแบบลงบอร์ดทดลองจะมีลักษณะเหมือนกันทุกประการ หากไฟล์ที่ใช้ในการทดลองไม่ตรงกับบอร์ดทดลองรุ่นที่กำหนดก็ให้ทำการแก้ไข Implementation constraints file (ไฟล์.ucf) เพื่อให้อุปกรณ์ต่างๆ ที่อยู่บนบอร์ดตรงกับขาของ FPGA หรือ CPLD ของบอร์ดรุ่นนั้นๆ

บทที่ 2

ภาษา VHDL

การออกแบบวงจรดิจิตอลโดยใช้ FPGA และ CPLD สามารถทำได้หลายวิธี ได้แก่ วิธีคาดผังวงจร (Schematic) วิธีเขียนบรรยายพุทธิกรรมการทำงานของวงจรด้วยภาษา HDL (Hardware Description Language) และ วิธีเขียนแผนภูมิสถานะ (State diagram) การออกแบบวงจรด้วยวิธีคาดผังวงจรไม่เหมาะสมสำหรับการออกแบบวงจรขนาดใหญ่ เพราะต้องใช้เวลาตรวจสอบวงจรนาน และตรวจสอบข้อผิดพลาดได้ยาก การออกแบบด้วยภาษา HDL ซึ่งเป็นภาษาระดับสูง เช่น ภาษา VHDL และ Verilog จึงเป็นทางเลือกที่ดีกว่า การออกแบบด้วยวิธีนี้ stemmed เป็นการใช้เครื่องทุ่นแรงชั้นเยี่ยมช่วยในการออกแบบวงจร เพราะเป็นการใช้ความสามารถของซอฟต์แวร์ทุกช่วยในการออกแบบโดยไม่จำเป็นต้องคิดรายละเอียดของวงจรในระดับล็อกิกเกตแต่อย่างใด

หนังสือเล่มนี้จะอธิบายการออกแบบวงจรดิจิตอลด้วยภาษา VHDL โดยใช้ซอฟต์แวร์ทุกของบริษัท Xilinx ที่สามารถดาวน์โหลดได้ฟรี คือ ISE WebPACK 8.1i และ ISE WebPACK 10.1i (สำหรับคนที่ใช้ Microsoft Vista Business (32-bit)) โดยที่เนื้อหาส่วนใหญ่ในบทที่ 2 นี้จะเน้นการเขียนโค้ด (Code) VHDL ที่สามารถนำไปสังเคราะห์เป็นวงจรได้ (Synthesizable)

2.1 บทนำ

VHDL ย่อมาจาก VHSIC Hardware Description Language (VHSIC = Very High Speed Integrated Circuit) VHDL เป็นภาษาระดับสูงที่ใช้ในการออกแบบระบบและวงจรดิจิตอลโดยมีรูปแบบคล้ายกับภาษา Pascal ภาษา VHDL เป็นภาษาที่อ่านเข้าใจง่ายและสามารถออกแบบวงจรในระดับต่างๆ ได้หลายระดับ โดยจะอยู่ในรูปแบบของฟังก์ชันท่านนั้น การเปลี่ยนแปลงแก้ไขวงจรหรือนำกลับมาใช้ใหม่จึงทำได้ง่าย สามารถใช้ภาษาระดับสูงในการเขียนโค้ดเพื่อช่วยตรวจสอบความถูกต้อง (Verification) ของวงจรที่ออกแบบได้ง่ายด้วยวิธีการจำลองการทำงาน (Simulation) จึงทำให้ไม่จำเป็นต้องทดลองกับวงจรจริง

VHDL ถูกจัดตั้งโดยสถาบัน IEEE ในปี 1987 (VHDL87) และได้ปรับปรุงรีโอยมาเป็น IEEE Std 1076-1993 (VHDL93), IEEE Std 1076-2000 และ IEEE Std 1076-2002 ตามลำดับ แต่ยังไม่ได้มาตรฐาน IEEE Std 1076 นั้นไม่ครอบคลุมล็อกิก บางสถานะ เช่น High impedance และสถานะอื่นๆ ที่ใช้จำลองการทำงาน ดังนั้น IEEE จึงได้นิยามชนิดข้อมูล std_logic (และ std_ulogic) ไว้ในมาตรฐาน IEEE Std 1164-1993 เพื่อแก้ไขปัญหาดังกล่าว แต่จากการที่ IEEE Std 1164-1993 ไม่ได้นิยามตัวดำเนินการ (Operator) ที่ใช้สำหรับเขียนโค้ด VHDL เพื่อการสังเคราะห์วงจรไว้อย่างเพียงพอ ทำให้ผู้พัฒนาซอฟต์แวร์ทุก (Software tool) สังเคราะห์วงจรแต่ละรายต้องสร้าง Package เพื่อนิยามตัวดำเนินการและชนิดข้อมูลแบบใหม่ขึ้นมาใช้เอง ซึ่งไม่เป็นมาตรฐานเดียวกัน ด้วยเหตุนี้ IEEE จึงได้นิยาม VHDL Synthesis packages ไว้ในมาตรฐาน IEEE Std 1076.3-1997 เพื่อใช้แทน Package ที่ไม่เป็นมาตรฐานเดียวกัน และช่วยให้โค้ด VHDL ที่เขียนขึ้นนั้นสามารถเข้ากันได้กับซอฟต์แวร์ทุกตัวๆ

2.2 VHDL พื้นฐาน

องค์ประกอบของภาษา VHDL (VHDL Structural elements) ตาม IEEE Std 1076 มีหน่วยออกแบบ (Design unit) ได้แก่

- Entity declaration คือ ส่วนที่เป็น Interface กล่าวคือ เป็นหน่วยออกแบบที่บอกรายละเอียดว่าอินพุต/เอาต์พุตพอร์ต (I/O Port) ของระบบดิจิตอลที่ออกแบบนั้นมีการเชื่อมต่อ (Interface) กับวงจรภายนอกอย่างไรบ้าง

- Architecture หรือ Architecture body คือ ส่วนที่เป็นหน่วยของแบบที่อธิบายความสัมพันธ์ระหว่างอินพุตและเอาต์พุต (บรรยายพฤติกรรมการทำงาน) ของระบบดิจิตอลที่ออกแบบ
- Package เป็นหน่วยของแบบที่เก็บรวมรวมโค้ดย่อยต่างๆ ซึ่งเป็นของส่วนกลางที่โค้ดต่างๆ เรียกใช้ร่วมกัน เช่น ค่าคงที่ โปรแกรมย่อย นิยามชนิดข้อมูล และ ประกาศใช้คอมโพnenet เป็นต้น เพื่ออำนวยความสะดวกในการเขียนโค้ด
- Configuration เป็นหน่วยของแบบที่ใช้ในการจับคู่หน่วยของแบบที่เป็น Entity declaration เข้ากับ Architecture ซึ่งใน การเขียนโค้ดนั้นอาจมีหลาย Architecture

โดยทั่วไปวงจรที่ออกแบบด้วยภาษา VHDL จะประกอบด้วยหน่วยของแบบพื้นฐานอย่างน้อย 2 หน่วย คือ ประกาศใช้เอนติตี้ (Entity declaration) หรือเอนติตี้ (Entity) และ อารชิเทกเชอร์บอดี้ (Architecture body) หรือ อารชิเทกเชอร์ (Architecture)

ตัวอย่างที่ 2.1 วงจรอตอครหัสเข้า 2 ออก 4 (2 to 4 Decoder) มีตารางความจริงแสดงดังรูปที่ 2.1a) และสมการบูลีน (สมการ SOP หรือ Sum of product) และแสดงดังรูปที่ 2.1b) วงจรอตอครหัสนี้มีผังวงจรแสดงดังรูปที่ 2.1c) ซึ่งอินพุต/เอาต์พุตในรูปที่ 2.1d) จะนำไปใช้ในการเขียนโค้ด (Code) ในส่วนของ Entity declaration ในขณะที่วงจรในรูปที่ 2.1e) จะนำไปใช้ในการเขียนโค้ดในส่วนของ Architecture body ซึ่งโค้ด VHDL ที่เขียนโดยใช้รูปแบบ VHDL87 และ VHDL93 และแสดงดังรูปที่ 2.2a) และรูปที่ 2.2b) 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

2.1a) ตารางความจริงของวงจร 2 to 4 Decoder

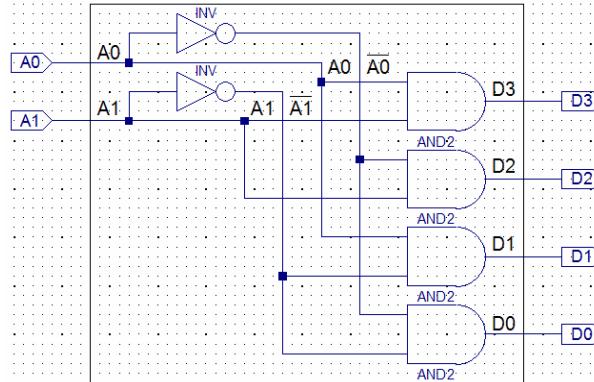
$$D0 = \overline{A1} \cdot \overline{A0}$$

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

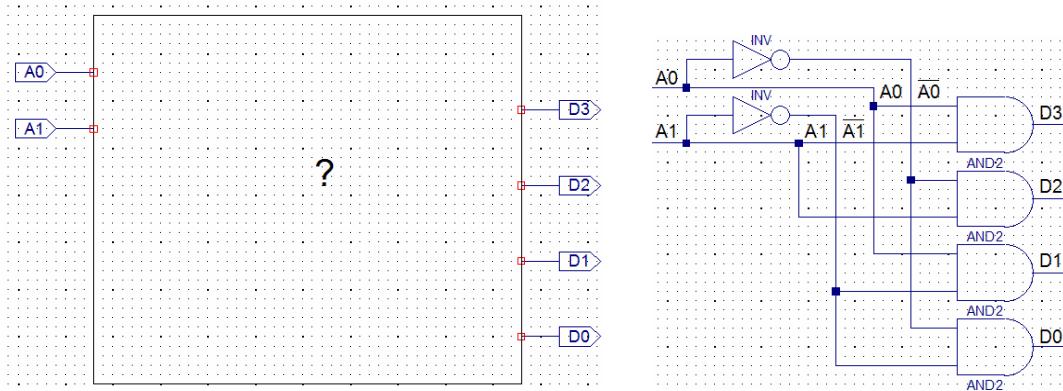
$$D2 = A1 \cdot \overline{A0}$$

$$D3 = A1 \cdot A0$$

2.1b) สมการบูลีนของวงจร 2 to 4 Decoder



2.1c) ผังวงจร 2 to4 Decoder



2.1d) ส่วนที่ใช้อธิบายประการใช้เอนดิตี้

2.1e) ส่วนที่ใช้อธิบายอาชีเทกเซอร์

รูปที่ 2.1 รายละเอียดต่างๆ 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
11 architecture BEHAVIORAL of DECODER2TO4 is
12 begin
13   D0 <= (not A1) and (not A0);
14   D1 <= (not A1) and A0;
15   D2 <= A1 and (not A0);
16   D3 <= A1 and A0;
17 end BEHAVIORAL;

```

บรรทัดที่ 2-9 เป็นประการใช้เอนดิตี้ ซึ่งส่วนนี้จะบอกว่าจะมีอินพุตและเอาต์พุตอะไรบ้าง

บรรทัดที่ 11-17 เป็นอาชีเทกเจอร์วันดี ซึ่งส่วนนี้จะบอกว่าจะทำงานอย่างไร

Operator เช่น and และ not

2.2a) 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;
10
11 architecture BEHAVIORAL of DECODER2TO4 is
12 begin
13   D0 <= (not A1) and (not A0);
14   D1 <= (not A1) and A0;
15   D2 <= A1 and (not A0);
16   D3 <= A1 and A0;
17 end architecture BEHAVIORAL;

```

บรรทัดที่ 9 อาจจะใช้ end entity แทนคำว่า end

บรรทัดที่ 17 อาจจะใช้ end architecture แทนคำว่า end

2.2b) โค้ด VHDL ที่เขียนโดยใช้รูปแบบใน VHDL93

รูปที่ 2.2 โค้ด VHDL ของวงจร 2 to 4 Decoder (เลขบรรทัดมีไว้เพื่ออำนวยความสะดวก ไม่ใช่ส่วนของโค้ด)

ให้ผู้อ่านเปิดไฟล์บทที่ 2 ของหนังสือเล่มนี้ในแผ่น DVD ซึ่งจะพบว่าในรูปที่ 2.2 นั้นนอกจากเราจะเขียนคำส่วน (Reserve words) ด้วยตัวพิมพ์เล็กแล้วซอฟต์แวร์ทูล ISE WebPACK จะแสดงคำส่วนด้วยตัวอักษรที่เป็นสีต่างๆ อ้างอิงจากเงื่อนไขดังนี้

เราจะเห็นได้ว่าให้คีย์เบิร์ดที่เขียนด้วยภาษา VHDL นั้นสามารถอ่านเข้าใจง่ายแม้บังคนจะไม่เคยรู้เกี่ยวกับภาษา VHDL มา ก่อนก็ตาม วงจร 2 to 4 Decoder นี้มีเงื่อนไขดังนี้ ชื่อ DECODER2TO4 และอาชีทธเชอร์ ชื่อ BEHAVIORAL มี A0 และ A1 เป็น อินพุต (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 = $\overline{A_1} \cdot \overline{A_0}$
	D1 <= (not A1) and A0;	มีความหมายว่า	D1 = $\overline{A_1} \cdot A_0$
	D2 <= A1 and (not A0);	มีความหมายว่า	D2 = $A_1 \cdot \overline{A_0}$
	D3 <= A1 and A0;	มีความหมายว่า	D3 = $A_1 \cdot A_0$

เราเรียก not และ and ว่าเป็นตัวดำเนินการหรือ Operator จากรูปที่ 2.2a) และรูปที่ 2.2b) นั้น A0, A1 และ D0-D3 มี สถานะเป็นลอจิก ‘1’ หรือ ‘0’ เท่านั้น เราจึงใช้ชนิดข้อมูล (Data type หรือ Type) ของอินพุตและเอาต์พุตเป็น “ชนิดข้อมูล bit”

คำสั่งที่อยู่ในอาชีทธเชอร์บรรทัดที่ 13-16 จะทำงานพร้อมกัน (Signal concurrency) โดยไม่สนใจลำดับก่อนและหลัง ซึ่งเราเรียกว่าคำสั่งคอนแครร์เรนต์ (Concurrent statement) การเขียนคำสั่งเหล่านี้ลับที่กันดังรูปที่ 2.3 จะให้ผลลัพธ์เหมือนเดิม

```

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 2.3 ตัวอย่าง Statement ใน VHDL ที่วางแผนบรรทัดกัน

จากรูป 2 to 4 Decoder ในตัวอย่างที่ 2.1 หากเราต้องการเขียนโค้ด VHDL ในส่วนของประกาศใช้เงื่อนไขให้กระชับ ขึ้นนี้เราสามารถใช้เครื่องหมาย “,” กันระหว่างอินพุตหรือเอาต์พุตต่างๆ ได้ดังรูปที่ 2.4

```

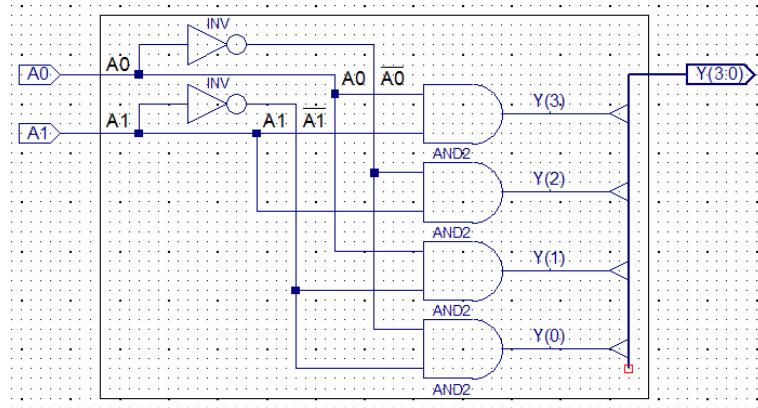
2 entity DECODER2TO4 is
3     port ( A0,A1 : in  bit;
4             D0,D1,D2,D3 : out bit);
5 end DECODER2TO4;
6
7 architecture BEHAVIORAL of DECODER2TO4 is
8 begin
9     D0 <= (not A1)and(not A0);
10    D1 <= (not A1)and A0;
11    D2 <= A1 and(not A0);
12    D3 <= A1 and A0;
13 end BEHAVIORAL;

```

การเขียนอินพุตหรือเอาต์พุตหลายตัวจะต้อง มี “,” กันระหว่างอินพุตหรือเอาต์พุตทุกตัว

รูปที่ 2.4 โค้ด VHDL ของวงจร 2 to 4 Decoder ที่เขียนให้กระชับขึ้น

จากรูป 2 to 4 Decoder ในตัวอย่างที่ 2.1 หากเราต้องการเขียนโค้ด VHDL โดยเขียนเอาต์พุตให้อยู่ในรูปของบัสก์ สามารถทำได้เช่นกัน โดยมีรายละเอียดของผังวงจรและโค้ด VHDL แสดงดังรูปที่ 2.5a) และรูปที่ 2.5b) ตามลำดับ



2.5a) 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;

```

2.5b) โค้ด VHDL ของวงจร 2 to 4 Decoder ที่มีเอาต์พุตเป็นบัสขนาด 4 บิต

รูปที่ 2.5 ผังวงจรและโค้ด VHDL ของวงจร 2 to 4 Decoder ที่มีเอาต์พุตเป็นบัสขนาด 4 บิต

จากรูปที่ 2.5a) เอาต์พุตของวงจร 2 to 4 Decoder เป็นบัส (Bus) Y ขนาด 4 บิต โดยมีสามารถแต่ละบิตของบัส Y ประกอบด้วย $Y(3)$, $Y(2)$, $Y(1)$ และ $Y(0)$ (เป็นค่าตัวอักษร Y_3 , Y_2 , Y_1 และ Y_0) โค้ด VHDL ในบรรทัดที่ 4 รูปที่ 2.5b) 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

2.3 กฏเกณฑ์ทั่วไป

โค้ด VHDL นั้นสามารถเขียนได้ทั้งตัวพิมพ์ใหญ่หรือพิมพ์เล็กหรือเขียนปุ่นกันก็มีความหมายเดียวกัน ซึ่งมีลักษณะที่เรียกว่า Case insensitive การเขียนคำสั่ง (Statement) แต่ละคำสั่งจะจบด้วยเครื่องหมาย ";" (Semicolon) ซึ่งการเขียนคำสั่งแต่ละคำสั่งอาจเขียนไว้ในบรรทัดเดียวกัน หรือในบรรทัดเดียวกันอาจมีหลายๆ คำสั่ง หรือบางคำสั่งอาจต้องเขียนหลายบรรทัดก็ได้ เช่นกัน การกำหนดค่า (Assign) จะใช้เครื่องหมายน้อยกว่าหรือเท่ากับ “`<=`” (Signal assignment)

การเขียนคำอธิบาย (Comment) ในภาษา VHDL นั้นก็เพื่อทำให้การอ่านโค้ดเข้าใจง่ายขึ้น โดยจะใช้ “-” (Hyphen) 2 ตัวติดกัน คือ “- -” แล้วเขียนตามด้วยคำอธิบาย เมื่อคอมไපเลอร์ตรวจพบ “- -” ก็จะละเว้นโดยไม่สนใจสิ่งที่เขียนไว้ต่อจากนั้นที่อยู่ในบรรทัดเดียวกัน ตัวอย่างการเขียนคำอธิบายแสดงดังรูปที่ 2.6

```

2 ----- Example -----
3 ----- VHDL code of 2 to 4 decoder -----
4 -----
5 -----
6
7 entity DECODER2TO4 is
8   port ( A0,A1 : in bit;          -- A0,A1=Input
9         D0,D1,D2,D3 : out bit); -- D0-D3=Output
10 end DECODER2TO4;
11
12 architecture BEHAVIORAL of DECODER2TO4 is
13 begin
14   D0 <= (not A1) and (not A0);
15   D1 <= (not A1) and A0;
16   D2 <= A1 and (not A0);
17   D3 <= A1 and A0;
18 end BEHAVIORAL;

```

รูปที่ 2.6 ตัวอย่างการเขียนคำอธิบายในโค้ด VHDL

2.4 การตั้งชื่อ

การตั้งชื่อ (Identifiers) ตาม VHDL87 นั้นอาจจะประกอบด้วย พัฒนา (Letter) ตัวเลข (Number) และ เครื่องหมาย จีดเส้นใต้ (Underscore) โดยที่

- ต้องเป็นต้นตัวพัฒนา
- ต้องเขียนติดกันโดยไม่เว้นช่องว่าง (White space)
- ห้ามใช้เครื่องหมายจีดเส้นใต้ติดกันหลายตัวและห้ามจบชื่อค้วยเครื่องหมายจีดเส้นใต้
- ห้ามใช้คำส่วน (Reserved words) ซึ่งคำส่วนตาม VHDL93 มีดังในรูปที่ 2.7

abs	bus	function	literal	others	return	transport
access	case	generate	loop	out	rol	type
after	component	generic	map	package	ror	unaffected
alias	configuration	group	mod	port	select	units
all	constant	guarded	nand	postponed	severity	until
and	disconnect	if	new	procedure	signal	use
architecture	downto	impure	next	process	shared	variable
array	else	in	nor	pure	sla	wait
assert	elsif	inertial	not	range	sll	when
attribute	end	inout	null	record	sra	while
begin	entity	is	of	register	srl	with
block	exit	label	on	reject	subtype	xnor
body	file	library	open	rem	then	xor
buffer	for	linkage	or	report	to	

รูปที่ 2.7 คำส่วนในภาษา VHDL

ตัวอย่างการตั้งชื่อ	ARCHI, aRChi , archi	ถูก เพราะเป็นชื่อเดียวกัน
	HALF_ADDER	ถูก
	HALF__ADDER	ผิด เพราะ underscore ติดกัน 2 ตัว
	HALF ADDER_	ผิด เพราะเกินช่องว่างและจบชื่อค่วยเครื่องหมายขีดเดือนได้
	74LS00	ผิด เพราะเป็นต้นคำยังตัวเลข
	open	ผิด เพราะใช้คำสงวน (ดู VHDL Reserved words ในรูปที่ 2.7)

2.5 ประเภทของพอร์ต

ประเภทของพอร์ต (Port mode หรือ Mode) ที่ใช้ในประกาศใช้เอนติตี้ (Entity declaration) ได้แก่

- in กือ อินพุต (Input) เป็นพอร์ตรับสัญญาณจากวงจรภายนอกเข้ามาในวงจรที่ออกแบบ
- out กือ เอาต์พุต (Output) เป็นพอร์ตส่งสัญญาณจากวงจรที่ออกแบบออกไปยังวงจรภายนอก
- inout กือ อินพุต/เอาต์พุต (I/O) เป็นพอร์ตรับ/ส่งสัญญาณเข้าหรือออกที่ใช้คิดต่อกับวงจรภายนอก
- buffer กือ เป็นเอาต์พุตพอร์ตที่สามารถอ่านค่ากลับเข้ามายังในวงจรที่ออกแบบได้ เช่น วงจรนับ ซึ่งจะอธิบายในตัวอย่างที่ 2.26 และตัวอย่างที่ 2.27 แต่อย่างไรก็ตามในการออกแบบวงจรในทางปฏิบัตินั้นเราสามารถเลือกเลี่ยงการใช้ Mode ที่เป็น buffer ได้อยู่แล้ว ดังนั้น Xilinx Synthesis Tool หรือ XST หรือซอฟต์แวร์สังเคราะห์วงจรที่มีอยู่ใน ISE WebPACK จึงแนะนำให้เลือกเลี่ยงการใช้ Mode ที่เป็น buffer

2.6 การเขียนโค้ด VHDL เมื่อต้นโดยใช้ชนิดข้อมูล std_logic และ std_logic_vector

จากตัวอย่างในข้อ 2.2 เราเขียนโค้ดโดยใช้ชนิดข้อมูล bit และ bit_vector ซึ่งมีสถานะโลจิก '0' และ '1' เท่านั้น ซึ่งในทางปฏิบัติบางวงจรอาจจะมี Tri-state buffer รวมอยู่ด้วย ซึ่งมีสถานะโลจิกได้ทั้ง '0', '1' และ High-impedance ดังนั้นชนิดข้อมูล bit จึงไม่ครอบคลุมสถานะ High-impedance ด้วยเหตุนี้เราจึงมีความจำเป็นต้องใช้ชนิดข้อมูล (Type) ใหม่ กือ std_logic (หรือ std_ulogic) แทนชนิดข้อมูล bit ซึ่งมีได้ 9 สถานะดังนี้

- 'U' มีสถานะโลจิก Uninitialized
- 'X' มีสถานะโลจิก Unknown
- '0' มีสถานะโลจิก Low (โลจิก'0')
- '1' มีสถานะโลจิก High (โลจิก'1')
- 'Z' มีสถานะโลจิก High impedance
- 'W' มีสถานะโลจิก Weak unknown
- 'L' มีสถานะโลจิก Weak low
- 'H' มีสถานะโลจิก Weak high
- '-' มีสถานะโลจิก Don't care

ชนิดข้อมูล std_logic จะเป็นชนิดข้อมูลย่อย (Subtype) ของชนิดข้อมูล std_ulogic ซึ่งมีสถานะโลจิกเหมือนกันแต่ชนิดข้อมูล std_logic จะมี Resolution function ในการแก้ไขปัญหาในกรณีที่เอาต์พุตของวงจรนั้นหรือต่อถึงกันหลายเอาต์พุต เช่น บัสของข้อมูล (Data bus) เป็นต้น ชนิดข้อมูล std_logic (และ std_ulogic) รวมทั้ง std_logic_vector (และ std_ulogic_vector) นั้นไม่ได้เป็นชนิดข้อมูลที่กำหนดล่วงหน้าไว้แล้วหรือ Predefined type ในมาตรฐาน IEEE Std 1076 ดังนั้นก่อนใช้งานจำเป็นต้อง

เรียกใช้ Package ชื่อ std_logic_1164 ที่ถูกคอมไพล์เก็บไว้ในไลบรารี (Library) ชื่อ ieee โดยจะต้องเขียนไว้ที่บรรทัดบนสุดของ โค๊ด ซึ่งเราสามารถเขียนโค๊ดโดยใช้ชนิดข้อมูล std_logic แทนชนิดข้อมูล bit ในรูปที่ 2.4 ได้ดังรูปที่ 2.8a) และในทำนองเดียวกันเราสามารถที่จะใช้ชนิดข้อมูล std_logic และ std_logic_vector !!แทนชนิดข้อมูล bit และ bit_vector ในรูปที่ 2.5b) ได้ดังรูปที่ 2.8b) ซึ่งการเรียกใช้ Package ชื่อ std_logic_1164 จะอยู่ในบรรทัดที่ 2 และบรรทัดที่ 3

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DECODER2TO4 is
6     port ( A0,A1 : in STD_LOGIC;
7             DO,D1,D2,D3 : out STD_LOGIC);
8 end DECODER2TO4;
9
10 architecture BEHAVIORAL of DECODER2TO4 is
11 begin
12     DO <= (not A1)and(not A0);
13     D1 <= (not A1)and A0;
14     D2 <= A1 and(not A0);
15     D3 <= A1 and A0;
16 end BEHAVIORAL;

```

2.8a) โค๊ด VHDL ของวงจร 2 to 4 Decoder ที่เขียนโดยใช้ชนิดข้อมูล std_logic

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DECODER2TO4 is
6     port ( A0,A1 : in STD_LOGIC;
7             Y : out STD_LOGIC_VECTOR(3 downto 0));
8 end DECODER2TO4;
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;

```

2.8b) โค๊ด VHDL ของวงจร 2 to 4 Decoder ที่มีเอาต์พุตเป็นชนิดข้อมูล std_logic_vector

รูปที่ 2.8 โค๊ด VHDL ของวงจร 2 to 4 Decoder ที่เขียนโดยใช้ชนิดข้อมูล std_logic และ std_logic_vector

2.7 ออกแบบ

ออกแบบ (Object) ในภาษา VHDL มี 4 ประเภท คือ ค่าคงที่ (Constant) สัญญาณ (Signal) ตัวแปร (Variable) และไฟล์ (File) อยากให้ผู้อ่านลองทำความเข้าใจในเบื้องต้นเกี่ยวกับออกแบบที่เป็นสัญญาณ (Signal) ดังตัวอย่างต่อไปนี้

ตัวอย่างที่ 2.2 วงจรมาตราقيเพล็กซ์เจอร์แบบ 2 อินพุต 1 เอาต์พุต (2 to 1 Multiplexer) มีผังวงจรแสดงดังรูปที่ 2.9 กล่าวคือ มี A และ B เป็นอินพุต มี C เป็นเอาต์พุตและ S เป็นขาควบคุม (อินพุต) เมื่อ S = '0' แล้วจะเลือกอินพุต A ไปที่เอาต์พุต C และเมื่อ S = '1' จะเลือกอินพุต B ไปที่เอาต์พุต C ซึ่งเราสามารถเขียนโค๊ด VHDL ได้ดังรูปที่ 2.10a) หรือรูปที่ 2.10b)

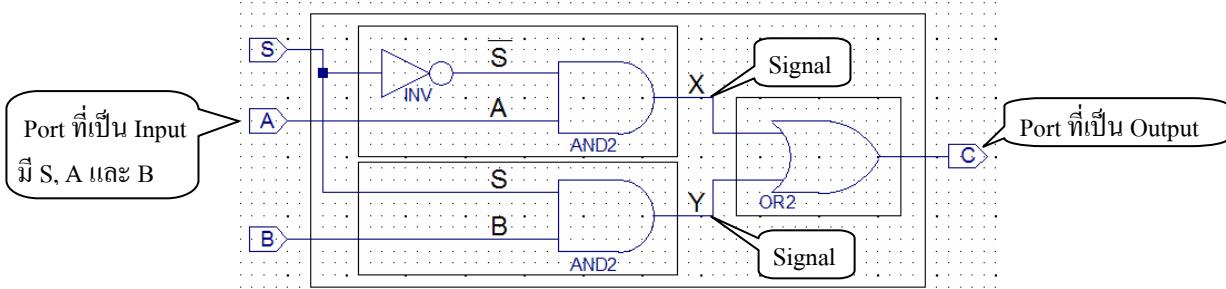
วงจรในรูปที่ 2.9 จะประกอบด้วยวงจรย่อย 3 วงจร โดยมี X และ Y เป็นสายสัญญาณหรือสายไฟเชื่อมต่อระหว่าง วงจรย่อยที่อยู่ภายในวงจรมาตราقيเพล็กซ์เจอร์ เราสามารถเขียนสมการบูลีน ได้ดังนี้

$$X = \overline{S} \cdot A$$

$$Y = S \cdot B$$

$$C = X + Y \quad \text{หรือ} \quad C = \overline{S} \cdot A + S \cdot B$$

ในภาษา VHDL นั้นจะเรียก X หรือ Y นี้ว่า Signal หรือสัญญาณ ซึ่งชนิดข้อมูลของ Signal X และ Signal Y ในกรณีนี้ เรากำหนดให้เป็นชนิดข้อมูล std_logic การประกาศใช้ Signal (Signal declaration) นั้นเราจะประกาศใช้ในบริเวณที่อยู่ระหว่าง architecture และ begin ดังรูปที่ 2.10a) ซึ่งเราจะประกาศใช้ Signal ก็ต่อเมื่อชื่อของ Signal นั้นไม่มีชื่อปรากฏใน port ของ entity



รูปที่ 2.9 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 นั้นเราจะประกาศใช้ ก็ต่อเมื่อชื่อของ Signal นั้นไม่มีชื่อปรากฏใน port ของ entity

2.10a) โค้ด VHDL วงจร 2 to 1 Multiplexer ที่เขียนโดยประกาศใช้ Signal (มีเรามองเห็นเป็นวงจรช่อง 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;

```

2.10b) โค้ด VHDL วงจร 2 to 1 Multiplexer ที่เขียนโดยไม่ประกาศใช้ Signal (มีเรามองเห็นเป็นวงจรใหญ่เพียงวงจรเดียว)

รูปที่ 2.10 โค้ด VHDL ของวงจร 2 to 1 Multiplexer ที่เขียนทั้งแบบประกาศใช้และไม่ประกาศใช้ Signal

จากรูปที่ 2.9 จะเห็นได้ว่า Signal จะเป็นสมือนสายไฟที่เชื่อมต่อ กันภายในวงจร และจากรูปที่ 2.10a) จะเห็นได้ว่า คำสั่งข้อบ 1 คำสั่ง (ซึ่งเป็นคำสั่งคอนแคร์เรนต์) จะแทนวงจรข้อบ 1 วงจร ดังนั้น Signal ในภาษา VHDL จึงทำหน้าที่ส่งผ่าน หรือแลกเปลี่ยนข้อมูลระหว่างคำสั่งคอนแคร์เรนต์ต่างๆ

ความหมายและรูปแบบการเขียนออบเจ็คต์ เป็นดังนี้ (โดยที่ ออบเจ็คต์ประเภท File จะอธิบายในบทที่ 6 ข้อ 6.2)

1) Constant

Constant ใช้ในการกำหนดค่าคงที่ รูปแบบการประกาศใช้ (Constant declaration) เป็นดังนี้

```
constant CONSTANT_NAME : TYPE [ := VALUE ] ;
```

```
เช่น constant WIDTH : integer := 16 ;
```

```
constant PI : real := 3.141592 ;
```

2) Signal

Signal เป็นสมือนสายสัญญาณ (สายไฟ) ที่ต่ออยู่กับภายในวงจร ใช้ในการส่งผ่านหรือแลกเปลี่ยนข้อมูลระหว่างคำสั่ง คอนแคร์เรนต์ต่างๆ การกำหนดค่าให้ Signal จะใช้เครื่องหมาย “`<=`” (Signal assignment) และต้องเป็นชนิดข้อมูลเดียวกันที่จะ กำหนดค่าให้กันและกันໄได้ เมื่อประกาศใช้ Signal แล้ว Signal จะสามารถมองเห็นหมดตลอดทั้งโค้ดหรือ “Global” และเมื่อ Signal อยู่ภายในคำสั่งชีวนิช (Sequential statement) นั้นการอัปเดต (Update) ค่าของ Signal จะทำได้ก็ต่อเมื่อจบการทำงาน ของคำสั่ง Process, Procedure และ Function แล้วเท่านั้น ดูตัวอย่างที่ 2.26 ตัวอย่างที่ 2.27 และตัวอย่างที่ 2.29

รูปแบบการประกาศใช้ Signal (Signal declaration) เป็นดังนี้

```
signal SIGNAL_NAME : TYPE [ := INITIAL_VALUE ] ;
```

```
เช่น signal Q_tmp : integer range 0 to 9 ;
```

```
signal AX : integer range 0 to 15 := 0 ;
```

```
signal B : std_logic_vector (0 to 31) ; - - B(0) = MSB, B(31) = LSB
```

```
signal C : std_logic_vector (3 downto 0) := "0000" ; - - C(3) = MSB, C(0) = LSB
```

```
signal X_CLK : std_logic ;
```

3) Variable

Variable จะคล้ายกับ Signal แต่จะถูกประกาศใช้เฉพาะภายในคำสั่งชีวนิช (Sequential statement) ได้แก่ คำสั่ง Process, Procedure และ Function เท่านั้น จึงมองเห็นໄได้เฉพาะบางพื้นที่หรือ “Local” Variable สามารถอัปเดต (Update) ค่าได้ทันที นั่นก็หมายความว่าเมื่อกำหนดค่าให้ Variable แล้ว ໄได้ที่อยู่ในคำสั่งดังสามารถนำค่ามานำไปใช้ได้ทันที Variable ใช้ได้เฉพาะภายในคำสั่งชีวนิชเท่านั้น การแลกเปลี่ยนค่าข้อมูลโดยตรงกับคำสั่งชีวนิชก็ทำไม่ได้ จึงต้องแลกเปลี่ยน ค่าข้อมูลโดยจะต้องกำหนดค่า Variable ให้กับ Signal หรือ Port ก่อนในขณะที่ยังคงอยู่ภายในคำสั่งชีวนิช ดูตัวอย่างที่ 2.26

ตัวอย่างที่ 2.27 และ ตัวอย่างที่ 2.29 ดังนั้น Variable จึงเป็นแค่พิธีที่เก็บค่าชั่วคราวในระหว่างที่กำลังทำการคำสั่งอยู่ภายในคำสั่งซึ่consequently เท่านั้น การกำหนดค่าให้กับ Variable จะใช้เครื่องหมาย “:=” (Variable assignment) และต้องเป็นชนิดข้อมูลเดียวกัน จึงจะกำหนดค่าให้กันและกันได้

รูปแบบการประกาศใช้ Variable (Variable declaration) เป็นดังนี้

```
variable VARIABLE_NAME : TYPE [ := INITIAL_VALUE ] ;
```

```
เช่น variable A_IN : integer range 0 to 9 ;
variable B : std_logic_vector (3 downto 0) := "0000"; -- B(3) = MSB, B(0) = LSB
variable QX : std_logic_vector (0 to 7); -- QX(0) = MSB, QX(7) = LSB
variable X : std_logic;
variable Y : std_logic := '0';
```

2.8 ชนิดข้อมูลและชนิดข้อมูลย่อย

ในการเขียนโค้ด VHDL ให้มีประสิทธิภาพนั้น ผู้เขียนจะต้องรู้เรื่องเกี่ยวกับชนิดข้อมูลและตัวดำเนินการ (Operator) ที่เกี่ยวข้องได้ดีพอสมควร ซึ่งได้แก่ ชนิดข้อมูลตามนิยามใน “Package standard” ของ IEEE Std 1076 ชนิดข้อมูลตามนิยามใน Std_logic_1164 package ของ IEEE Std 1164 และชนิดข้อมูลตามนิยามใน VHDL Synthesis packages ของ IEEE Std 1076.3

Package เหล่านี้จะนิยามชนิดข้อมูลต่างๆ พร้อมกับตัวดำเนินการที่เกี่ยวข้องมาให้ด้วย ตัวอย่างในรูปที่ 2.11a) และรูปที่ 2.11b) จะเป็นการนิยามชนิดข้อมูล bit และ integer ที่อยู่ในส่วนของประกาศใช้ Package (Package declaration) ชื่อ standard ของ IEEE Std 1076 ซึ่งสิ่งที่ใส่คำอธิบายไว้ (Comment) จะบอกว่าชนิดข้อมูล bit และ integer สามารถใช้กับตัวดำเนินการอะไรได้บ้าง โดยที่ในมาตรฐานจะนิยามตัวดำเนินการเหล่านี้ในรูปฟังก์ชัน (Function) เราจะเห็นได้ว่าชนิดข้อมูล bit ไม่สามารถใช้กับตัวดำเนินการคณิตศาสตร์ เช่น +, - และ * เป็นต้น ในขณะที่ชนิดข้อมูล integer ไม่สามารถใช้กับตัวดำเนินการตรรกะ เช่น and, or และ nand เป็นต้น นั่นก็หมายความว่า “ตัวดำเนินการชนิดหนึ่งจะใช้ได้กับชนิดข้อมูลบางชนิดเท่านั้น” นี่คือข้อจำกัดที่ผู้ออกแบบต้องทราบและน้อมถวิลที่ผู้เรียนดูมักจะเจอบัญหา

```
type BIT is ('0', '1');
-- The predefined operators for this type are as follows:
-- function "and" (anonymous, anonymous: BIT) return BIT;
-- function "or" (anonymous, anonymous: BIT) return BIT;
-- function "nand" (anonymous, anonymous: BIT) return BIT;
-- function "nor" (anonymous, anonymous: BIT) return BIT;
-- function "xor" (anonymous, anonymous: BIT) return BIT;
-- function "xnor" (anonymous, anonymous: BIT) return BIT;

-- function "not" (anonymous: BIT) return BIT;

-- function "=" (anonymous, anonymous: BIT) return BOOLEAN;
-- function "/=" (anonymous, anonymous: BIT) return BOOLEAN;
-- function "<" (anonymous, anonymous: BIT) return BOOLEAN;
-- function "<=" (anonymous, anonymous: BIT) return BOOLEAN;
-- function ">" (anonymous, anonymous: BIT) return BOOLEAN;
-- function ">=" (anonymous, anonymous: BIT) return BOOLEAN;
```

ตัวอย่างการนิยามชนิดข้อมูล bit ใน IEEE Std 1076
โดยการประกาศใช้ชนิดข้อมูล (Type declaration)

2.11a) การประกาศใช้ชนิดข้อมูล bit (ส่วนที่ไม่อธิบายใน Comment)

```

type INTEGER is range implementation_defined;
-- The predefined operators for this type are as follows:
-- function "*" (anonymous: universal_integer; anonymous: INTEGER)
--     return universal_integer;
-- function "/" (anonymous: universal_real; anonymous: INTEGER)
--     return universal_real;
-- function "=" (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function "/=" (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function "<" (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function "<=" (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function ">" (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function ">=" (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function "+" (anonymous: INTEGER) return INTEGER;
-- function "-" (anonymous: INTEGER) return INTEGER;
-- function "abs"(anonymous: INTEGER) return INTEGER;
-- function "+" (anonymous, anonymous: INTEGER) return INTEGER;
-- function "?" (anonymous, anonymous: INTEGER) return INTEGER;
-- function "*" (anonymous, anonymous: INTEGER) return INTEGER;
-- function "/" (anonymous, anonymous: INTEGER) return INTEGER;
-- function "mod"(anonymous, anonymous: INTEGER) return INTEGER;
-- function "rem"(anonymous, anonymous: INTEGER) return INTEGER;
-- function "***" (anonymous: INTEGER; anonymous: INTEGER) return INTEGER;

```

ตัวอย่างการประกาศใช้ชนิดข้อมูล integer
ใน IEEE Std 1076 เช่น type INTEGER is
range -2147483647 to 2147483647;

2.11b) การประกาศใช้ชนิดข้อมูล integer (ส่วนที่ไม่อ่านใน Comment)

รูปที่ 2.11 การประกาศใช้ชนิดข้อมูล bit และ integer ตามมาตรฐาน IEEE Std 1076

2.8.1 ชนิดข้อมูลตามมาตรฐาน IEEE Std 1076

มาตรฐาน IEEE Std 1076 ได้แบ่งชั้น (Class) ของชนิดข้อมูลออกเป็น 4 ชั้นดังนี้ คือ scalar types, composite types, access types และ file types โดยที่ซอฟต์แวร์ทุกสังเคราะห์จะรองรับการใช้ (Supported) เลขพาระ scalar types และ composite types เท่านั้น ส่วน file types ใช้ในการอ่านหรือเขียนไฟล์ข้อมูลสำหรับจำลองการทำงาน หรือ กำหนดค่าเริ่มต้นให้หน่วยความจำ

2.8.1.1 ชนิดข้อมูล Scalar

ชนิดข้อมูล Scalar (Scalar types) ประกอบด้วย Enumeration types, integer types, physical types และ floating point types โดยที่เราเรียกว่า Enumeration types และ integer types ว่า “discrete types” และจะเรียกว่า integer types, floating point types และ physical types ว่า “Numeric types” ผู้อ่านควรจำชื่อชนิดข้อมูลเหล่านี้ไว้เพื่อจะได้ไม่สับสนเมื่อต้องการค้นคว้าเพิ่มเติมจากตำราหรืออ่านจากมาตรฐานเกี่ยวกับ VHDL ที่เป็นภาษาต่างประเทศ

1) ชนิดข้อมูล Enumeration

ชนิดข้อมูล Enumeration หรือ Enumeration types ที่กำหนดล่วงหน้าไว้แล้ว (Predefined) ในมาตรฐาน IEEE Std 1076 (กล่าวคือ ชนิดข้อมูลนี้มีอยู่แล้วในมาตรฐาน IEEE Std 1076) โดยนิยามไว้ใน Package ชื่อ Standard ได้แก่

- ชนิดข้อมูล character เป็นชนิดข้อมูลที่เป็นตัวอักษรต่างๆ ตาม ISO 8859-1:1987 มีทั้ง 256 ตัว
- ชนิดข้อมูล bit มีค่าเป็นລອຈິກ ‘0’ และ ‘1’
- ชนิดข้อมูล boolean มีค่าเป็น เท็จ (หรือ FALSE) และ จริง (หรือ TRUE)
- ชนิดข้อมูล SEVERITY_LEVEL มี NOTE, WARNING, ERROR และ FAILURE
- ชนิดข้อมูล FILE_OPEN_KIND มี READ_MODE, WRITE_MODE และ APPEND_MODE
- ชนิดข้อมูล FILE_OPEN_STATUS มี OPEN_OK, STATUS_ERROR, NAME_ERROR, MODE_ERROR

การนิยามชนิดข้อมูล Enumeration โดยผู้ใช้งาน (User defined) หรือนิยามโดยกำหนดล่วงหน้าไว้แล้ว (Predefined) ใน Package ต่างๆ ในมาตรฐาน IEEE จะมีรูปแบบการประกาศใช้ชนิดข้อมูลเหมือนกัน ดังต่อไปนี้

```
type BOOLEAN is (FALSE, TRUE);
type BIT is ('0', '1');                                (ดูตัวอย่างในรูปที่ 2.11a))
type COLOR is (BLUE, GREEN, YELLOW, RED);
type LOGIC_LEVEL is ('0', '1', 'Z');                  -- Base type
subtype BIT_LOGIG is LOGIC_LEVEL range '0' to '1'; -- Subtype
```

การนิยามเป็นชนิดข้อมูลย่อย (Subtype) นั้นมีข้อดีกว่าการนิยามเป็นชนิดข้อมูล (Type) ใหม่ กดล้วนคือ เราสามารถทำ Operation ต่างๆ โดยใช้ตัวดำเนินการ (Operator) เดียวกันกับ Base type ได้ แต่ถ้านิยามเป็นชนิดข้อมูลใหม่ซึ่งเป็นคนละชนิด ข้อมูลกันนั้นจะไม่สามารถใช้ตัวดำเนินการต่างๆ ร่วมกันได้และอาจจำเป็นต้องนิยามตัวดำเนินการขึ้นมาใช้เอง

2) ชนิดข้อมูล integer

ชนิดข้อมูล integer หรือ integer type ที่กำหนดล่วงหน้าไว้แล้ว (Predefined) ในมาตรฐาน IEEE Std 1076 โดยนิยามไว้ใน Package ชื่อ Standard มีค่าอย่างน้อยอยู่ในช่วง $-(2^{31}-1)$ หรือ -2147483647 ถึง $(2^{31}-1)$ หรือ 2147483647 ซึ่งโดยปกติจะถูกกำหนดไว้ล่วงหน้า (Default) เป็นเลขฐานสิบ เช่น -5, -1, 0, 8, 100 เป็นต้น

ชนิดข้อมูลย่อยหรือ Subtype ของชนิดข้อมูล integer (Base type) ที่กำหนดล่วงหน้าไว้แล้ว (Predefined) ได้แก่ natural (มีค่าอย่างน้อยอยู่ในช่วง 0 ถึง 2147483647) และ positive (มีค่าอย่างน้อยอยู่ในช่วง 1 ถึง 2147483647)

การนิยามชนิดข้อมูล integer โดยผู้ใช้งาน (User defined) หรือนิยามไว้ใน Package ต่างๆ จะมีรูปแบบการประกาศใช้ชนิดข้อมูลเหมือนกัน ดังต่อไปนี้

```
type INTEGER is range -2147483647 to 2147483647;      (ดูตัวอย่างในรูปที่ 2.11b))
subtype NATURAL is integer range 0 to 2147483647;
subtype POSITIVE is integer range 1 to 2147483647;
type WORD_INDEX is range 31 downto 0;
type BYTE_LENGTH is range 0 to 255;
subtype HIGH_BIT_LOW is BYTE_LENGTH range 0 to 127;
```

3) ชนิดข้อมูล physical

ชนิดข้อมูล physical หรือ physical types ที่กำหนดล่วงหน้าไว้แล้ว (Predefined) ในมาตรฐาน IEEE Std 1076 โดยนิยามไว้ใน Package ชื่อ Standard คือ ชนิดข้อมูล time มีค่าอย่างน้อยอยู่ในช่วง -2147483647 ถึง 2147483647 ตัวอย่างเช่น

```
type DURATION is range -2147483647 to 2147483647
units
    fs;                      -- femtosecond
    ps = 1000 fs;            -- picosecond
    ns = 1000 ps;            -- nanosecond
    us = 1000 ns;            -- microsecond
    ms = 1000 us;            -- millisecond
    sec = 1000 ms;           -- second
    min = 60 sec;             -- minute
end units;
```

ในการแก้ของชนิดข้อมูล time นั้น Xilinx Synthesis Tool หรือ XST ที่มีอยู่ในซอฟต์แวร์ทุก ISE WebPACK และ ISE Foundation จะละเว้น (Ignored) โดยจะไม่สนใจชนิดข้อมูลนี้

4) ชนิดข้อมูล floating point

ชนิดข้อมูล floating point (real) หรือ floating point types (real types) ที่กำหนดล่วงหน้าไว้แล้ว (Predefined) ใน มาตรฐาน IEEE Std 1076 โดยนิยามไว้ใน Package ชื่อ Standard มีค่าอย่างน้อยอยู่ในช่วง -1.0E38 to +1.0E38 ชนิดข้อมูล real เช่น 1.2, 10.0 เป็นต้น (ถ้าเขียนเป็น 10 จะถูกยกเว้นเป็นชนิดข้อมูล integer)

2.8.1.2 ชนิดข้อมูล Composite

ชนิดข้อมูล Composite (Composite types) ประกอบด้วย Array types และ Record types มีรายละเอียดดังนี้คือ

1) ชนิดข้อมูลอะเรย์

ชนิดข้อมูลอะเรย์ (Array) หรือ Array types เป็นชนิดข้อมูลที่มีสมาชิกหลายตัวที่เป็นชนิดข้อมูลเดียวกัน โดย Xilinx Synthesis Tool หรือ XST จะรองรับการใช้ชนิดข้อมูล Multi-dimentional array ได้สูงสุดถึง 3 มิติ ชนิดข้อมูลอะเรย์ที่กำหนดล่วงหน้าไว้แล้ว (Predefined) ใน มาตรฐาน IEEE Std 1076 โดยนิยามไว้ใน Package ชื่อ Standard คือ ชนิดข้อมูล string และ bit_vector โดยที่ชนิดข้อมูล string คือ ชนิดข้อมูลอะเรย์ 1 มิติ (One-dimensional arrays) ของชนิดข้อมูล character และชนิดข้อมูล bit_vector คือ ชนิดข้อมูลอะเรย์ 1 มิติของชนิดข้อมูล bit

การนิยามชนิดข้อมูลอะเรย์ 1 มิติโดยผู้ใช้งาน (User defined) หรือนิยามไว้ใน Package ต่างๆ จะมีรูปแบบการประกาศใช้ชนิดข้อมูลเหมือนกัน ทั้งแบบ Unconstrained array (ไม่จำกัดขอบเขต) และ Constrained array (จำกัดขอบเขต) ตัวอย่างเช่น

```

type BIT_VECTOR is array (NATURAL range <>) of BIT;      -- Unconstrained array
type STRING is array (POSITIVE range <>) of CHARACTER; -- Unconstrained array
type NIBBLE is array (3 downto 0) of BIT;                  -- Constrained array
subtype BYTE is BIT_VECTOR (7 downto 0);                  -- Constrained array

```

ตัวอย่างการนิยามชนิดข้อมูลอะเรย์ 1 มิติ x 1 มิติ เช่น

```
type DATA_ROM is array (4095 downto 0) of BIT_VECTOR (7 downto 0);
```

หรือถ้า尼ยาม subtype BYTE is BIT_VECTOR (7 downto 0); ไว้แล้วก็จะเขียนใหม่ได้ดังนี้

```
type DATA_ROM is array (4095 downto 0) of BYTE;
```

2) ชนิดข้อมูล record

ชนิดข้อมูล record หรือ record types จะเหมือนชนิดข้อมูลอะเรย์ แต่เป็นชนิดข้อมูลต่างชนิดกัน ตัวอย่างเช่น

```

type CONTROL_BUS is record
    R_W : bit;
    DATA_BUS : bit_vector (7 downto 0);
    ADDR_BUS : integer range 0 to 1023;
    CE : bit;
end record;

```

2.8.1.3 ชนิดข้อมูล access

ชนิดข้อมูล access นี้จะอธิบายในบทที่ 6 ข้อ 6.2

2.8.1.4 ชนิดข้อมูล file

ชนิดข้อมูล file นี้จะอธิบายในบทที่ 6 ข้อ 6.2 ซึ่งอาจนำไปใช้ในการใส่ค่าเริ่มต้น (Initial value) ให้กับ RAM ที่อ่านค่าจากไฟล์ที่อยู่ภายนอก หรือใช้เป็นไฟล์อินพุตที่สร้างสัญญาณทดสอบ (Stimulus) หรือ Test vector ให้กับ Testbench

2.8.2 ชนิดข้อมูลตามมาตรฐาน IEEE Std 1164-1993

วงจรดิจิตอลที่ใช้ในทางปฏิบัติอาจมีสถานะคลอกิกได้ทั้ง ‘0’, ‘1’ และ High-impedance ชนิดข้อมูล bit และ bit_vector จึงไม่ครอบคลุมสถานะคลอกิกทุกสถานะ ดังนั้น IEEE จึงได้นิยามชนิดข้อมูล std_logic (และ std_ulogic) เพื่อใช้แทน bit และนิยามชนิดข้อมูล std_logic_vector (และ std_ulogic_vector) เพื่อใช้แทน bit_vector โดยที่ std_logic จะมีสถานะคลอกิกเหมือนกับ std_ulogic ทุกประการ แต่ std_logic เป็นชนิดข้อมูลย่อย (Subtype) ที่เป็น Resolved type ของ std_ulogic (โดยที่ std_ulogic จะเป็น Unresolved type) ชนิดข้อมูล std_logic มีคุณสมบัติพิเศษ คือ ขอมให้อาต์พุตต่อถึงกันหรือชันกันได้หลายอาต์พุต เรียกว่า Multiple-drivers โดยใช้ Resolution function เพื่อหาค่าสถานะคลอกิกของอาต์พุต ซึ่งชนิดข้อมูลอื่นจะทำ Multiple-drivers ไม่ได้

ชนิดข้อมูล std_logic (และ std_ulogic) เป็น Scalar types และ Enumeration types IEEE Std 1164 ได้กำหนดชนิดข้อมูลนี้และชนิดข้อมูลย่อยต่างๆ ไว้ใน Std_logic_1164 package โดยที่ std_logic (และ std_ulogic) มีสถานะลอจิกได้ 9 สถานะเพื่อให้ครอบคลุมทุกสถานะที่ใช้จำลองการทำงาน (Simulation) และสังเคราะห์งงาน (Synthesis) มีรายละเอียดการนิยามเป็นดังนี้

```
type STD_ULOGIC is ('U', -- Uninitialized
                     'X', -- Forcing Unknown
                     '0', -- Forcing 0
                     '1', -- Forcing 1
                     'Z', -- High Impedance
                     'W', -- Weak Unknown
                     'L', -- Weak 0
                     'H', -- Weak 1
                     '-'); -- Don't care
```

และ subtype STD_LOGIC is resolved STD_ULOGIC;

```
subtype X01 is resolved STD_ULOGIC range 'X' to '1';      -- ('X', '0', '1')
subtype X01Z is resolved STD_ULOGIC range 'X' to 'Z';      -- ('X', '0', '1', 'Z')
subtype UX01 is resolved STD_ULOGIC range 'U' to '1';      -- ('U', 'X', '0', '1')
subtype UX01Z is resolved STD_ULOGIC range 'U' to 'Z';     -- ('U', 'X', '0', '1', 'Z')
```

โดยที่ Resolution function (ฟังก์ชัน resolved) ที่นิยามไว้ใน Std_logic_1164 package มีรายละเอียดดังในรูปที่ 2.12

```
-- resolution function
CONSTANT resolution_table : stdlogic_table := (
-- | U   X   0   1   Z   W   L   H   -   | |
-- |-----|-----|-----|-----|-----|-----|-----|-----|-----|
( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', -- | U |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', -- | X |
( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X', -- | 0 |
( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X', -- | 1 |
( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X', -- | Z |
( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X', -- | W |
( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X', -- | L |
( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X', -- | H |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X') -- | - |
```

รูปที่ 2.12 Resolution function ที่นิยามไว้ใน std_logic_1164 package

ในทำนองเดียวกัน IEEE Std 1164 ได้กำหนดชนิดข้อมูลของเรซิ่ล (One-dimensional arrays) ของ std_logic (และ std_ulogic) ซึ่งก็คือชนิดข้อมูล std_logic_vector (และ std_ulogic_vector) ไว้ใน Std_logic_1164 package ดังนี้

```
type STD_ULOGIC_VECTOR is array (natural range <>) of STD_ULOGIC; -- Unconstrained array
type STD_LOGIC_VECTOR is array (natural range <>) of STD_LOGIC; -- Unconstrained array
```

การนิยามชนิดข้อมูลօร์เดอร์ 1 มิติโดยผู้ใช้งาน (User defined) หรือนิยามไว้ใน Package ต่างๆ จะมีรูปแบบการประกาศใช้ชนิดข้อมูลเหมือนกัน ตัวอย่างเช่น

```
type NIBBLE is array (3 downto 0) of STD_LOGIC;
subtype BYTE is std_logic_vector (7 downto 0);
```

ตัวอย่างการนิยามชนิดข้อมูลօร์เดอร์ 1 มิติ x 1 มิติ เช่น

```
type DATA_ROM is array (4095 downto 0) of std_logic_vector (7 downto 0);
```

หรือถ้า尼ยาม subtype BYTE is std_logic_vector (7 downto 0); ไว้แล้วก็จะเขียนใหม่ได้ดังนี้

```
type DATA_ROM is array (4095 downto 0) of BYTE;
```

ตัวอย่างการนิยามชนิดข้อมูลօร์เดอร์ 2 มิติ เช่น

```
type DATA_2D is array (15 downto 0, 7 downto 0) of std_logic;
```

```
type A_DATA is array (0 to 255, 7 downto 0) of std_logic;
```

ตัวอย่างการนิยามชนิดข้อมูลօร์เดอร์ 3 มิติ เช่น

```
type DATA_3D is array (0 to 255, 15 downto 0, 7 downto 0) of std_logic;
```

นอกจากนี้แล้ว IEEE Std 1164 ยังได้นิยามเกี่ยวกับ Operator ต่างๆ ซึ่งจะอธิบายในข้อ 2.9 นิยามฟังก์ชันต่างๆ ได้แก่

- ฟังก์ชัน Edge detection เป็นดังนี้

rising_edge (CLK) มีความหมายเดียวกับ CLK'event and CLK = '1' คือ CLK ทริกตัวขอนบนขาขึ้น
falling_edge (CLK) มีความหมายเดียวกับ CLK'event and CLK = '0' คือ CLK ทริกตัวขอนบนขาลง

- ฟังก์ชันแปลงชนิดข้อมูลไปเป็นชนิดข้อมูลที่ต่างชนิดกัน (Conversion functions) มีรายละเอียดดังรูปที่ 2.13

Conversion	Function
std_ulogic to bit	to_bit (expression)
std_logic_vector to bit_vector	to_bitvector (expression)
std_ulogic_vector to bit_vector	to_bitvector (expression)
bit to std_ulogic	to_StdULogic (expression)
bit_vector to std_logic_vector	to_StdLogicVector (expression)
bit_vector to std_ulogic_vector	to_StdULogicVector (expression)
std_ulogic to std_logic_vector	to_StdLogicVector (expression)
std_logic to std_ulogic_vector	to_StdULogicVector (expression)

รูปที่ 2.13 ตารางแสดงฟังก์ชันแปลงชนิดข้อมูลไปเป็นชนิดข้อมูลที่ต่างชนิดกัน

2.8.3 ชนิดข้อมูลตามมาตรฐาน IEEE Std 1076.3-1997

ปัญหาในทางปฏิบัติของ IEEE Std 1076 และ IEEE Std 1164 คือ ชนิดข้อมูล bit, bit_vector, std_logic (std_ulogic) และ std_logic_vector (std_ulogic_vector) ไม่สามารถใช้กับตัวดำเนินการคณิตศาสตร์ได้ ซึ่งตาม IEEE Std 1076 นั้นตัวดำเนินการคณิตศาสตร์จะใช้ได้กับชนิดข้อมูล Numeric ได้แก่ integer, floating point และ physical เท่านั้น จึงทำให้ผู้ใช้งานต้องสร้าง Function ขึ้นมาใช้เองหรือใช้ Package ที่ไม่เป็นมาตรฐาน (Nonstandard package) ของผู้พัฒนาซอฟต์แวร์ทุกสังเคราะห์ วงจร ทำให้โค้ดที่เขียนขึ้นไม่สามารถเข้ากันได้กับซอฟต์แวร์ทุกจากผู้ผลิต (Vendor) รายอื่น เพื่อแก้ไขปัญหานี้ IEEE จึงได้ออกมาตรฐาน IEEE Std 1076.3 โดยนิยามชนิดข้อมูล unsigned และ signed (อยู่ในรูปฟอร์ม 2's complement) ซึ่งเป็น Numeric types ใหม่ไว้ใน Numeric_bit package และ Numeric_std package โดยเขียนในรูปฟอร์มของชนิดข้อมูล bit_vector และ std_logic_vector ตามลำดับ ซึ่งสามารถใช้กับตัวดำเนินการคณิตศาสตร์ได้รากับว่าเป็นชนิดข้อมูล integer โดยที่ Package นี้ จะนิยามชนิดข้อมูล unsigned และ signed คล้ายกับของ std_logic_arith package ของ Synopsys (Nonstandard package) การใช้ชนิดข้อมูลและตัวดำเนินการ (Operator) ต่างๆ ตาม IEEE Std 1076.3 นั้นจะต้องเรียกใช้ Numeric_bit หรือ Numeric_std package

ชนิดข้อมูล unsigned และ signed ที่นิยามไว้ใน Numeric_std package ของ IEEE Std 1076.3 นั้นจะเป็นชนิดข้อมูล อะเรย์ โดยที่สามารถแต่งตัวจะยังคงเป็นชนิดข้อมูล std_logic การประยุกต์ใช้ชนิดข้อมูล unsigned และ signed เป็นดังนี้

```
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC; -- Unconstrained array
type SIGNED is array (NATURAL range <>) of STD_LOGIC; -- Unconstrained array
```

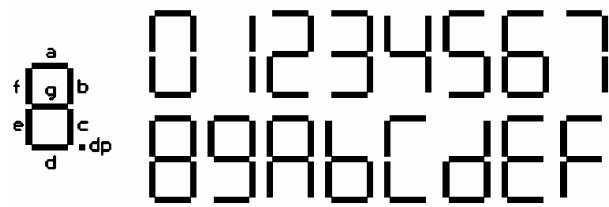
IEEE Std 1076.3 ได้นิยามเกี่ยวกับตัวดำเนินการ (Operator) ต่างๆ ซึ่งจะอธิบายในข้อ 2.9 ไว้อย่างครบถ้วน นอกจากนี้แล้ว IEEE Std 1076.3 ยังได้นิยามฟังก์ชันต่างๆ ได้แก่

- ฟังก์ชัน Edge detection ของ IEEE Std 1076.3 นั้นจะใช้ร่วมกับ IEEE Std 1164 ได้อยู่แล้ว เพราะว่าสามารถของชนิดข้อมูล unsigned และ signed เป็นชนิดข้อมูล std_logic
- ฟังก์ชันแปลง (Conversion functions) ชนิดข้อมูล unsigned และ signed เป็นชนิดข้อมูลอื่นๆ หรือกลับกันมีรายละเอียดในตารางรูปที่ 2.14 โดยที่ “*” หมายถึง Explicit type conversions ซึ่งจะอธิบายในข้อ 2.9.5

Conversion	Function (Numeric_std package)	Function (std_logic_arith package)
unsigned to integer	to_integer (expression)	conv_integer (expression)
signed to integer	to_integer (expression)	conv_integer (expression)
integer to unsigned	to_unsigned (expression, size)	conv_unsigned (expression, size)
integer to signed	to_unsigned (expression, size)	conv_signed (expression, size)
Resizing : unsigned	resize (expression, size)	conv_unsigned (expression, size)
Resizing : signed	resize (expression, size)	conv_signed (expression, size)
std_logic_vector to unsigned	unsigned (expression) *	conv_unsigned (expression)
std_logic_vector to signed	signed (expression) *	conv_signed (expression)
unsigned to std_logic_vector	std_logic_vector (expression) *	conv_std_logic_vector (expression)
signed to std_logic_vector	std_logic_vector (expression) *	conv_std_logic_vector (expression)

รูปที่ 2.14 ฟังก์ชันแปลงชนิดข้อมูลไปเป็นชนิดข้อมูลที่ต่างชนิดกัน

ตัวอย่างที่ 2.3 ฝึกเขียนโค้ด VHDL ของวงจรอุดรหัสตัวแสดงผลเซเว่นเซกเมนต์แบบคอมมอนแค็ปต์โดยใช้ ROM ขนาด 7 บิตในการเก็บข้อมูลของแต่ละเซกเมนต์ของเลข 0-F ในรูปแบบเลขฐานสิบหกแสดงดังรูปที่ 2.15a) ถึงรูปที่ 2.15f)



2.15a) รูปแบบเลขฐานสิบหกของตัวแสดงผลเซเว่นเซกเมนต์

No.	Input				Output						
	A(3)	A(2)	A(1)	A(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

2.15b) ตารางความจริงของตัวอุดรหัสตัวแสดงผลเซเว่นเซกเมนต์แบบแค็ปต์รวม

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;          เรียกใช้ numeric_std package ที่อยู่ใน Library ieee เพราะใช้คำสั่ง to_integer และใช้ unsigned
5
6 entity ROM1 is
7     Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
8            Y : out STD_LOGIC_VECTOR(6 downto 0));          ประกาศใช้ชนิดข้อมูล rom_type แบบ Constrained array
9 end ROM1;
10 architecture Behavioral of ROM1 is
11     type rom_type is array (15 downto 0) of std_logic_vector(6 downto 0);          -- gfedcba gfedcba
12     constant ROM : rom_type :=("1110001","1111001", --F,E
13                               "1011110","0111001", --d,C           Y(0)=a
14                               "1111100","1110111", --b,A           ---
15                               "1101111","1111111", --9,8 Y(5)=f | Y(1)=b
16                               "0000111","1111101", --7,6           --- Y(6)=g
17                               "1101101","1100110", --5,4 Y(4)=e | Y(2)=c
18                               "1001111","1011011", --3,2           ---
19                               "0000110","0111111");--1,0           Y(3)=d
20
21 signal N : integer range 15 downto 0;          ใช้คำสั่ง unsigned แปลงชนิดข้อมูล std_logic_vector เป็น unsigned และใช้ to_integer แปลงเป็น integer
22 begin
23     N <= to_integer(unsigned(A));
24     Y <= ROM(N);
25 end Behavioral;

```

2.15c) โค้ด VHDL ที่เขียนโดยประกาศใช้ชนิดข้อมูล std_logic_vector อะเรย์ 1 มิติ x 1 มิติแบบ Constrained array

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity ROM1 is
7     Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
8             Y : out STD_LOGIC_VECTOR(6 downto 0));
9 end ROM1;
10 architecture Behavioral of ROM1 is
11     type rom_type is array (integer range <>) of std_logic_vector(6 downto 0);
12         -- gfedcba gfedcba
13     constant ROM : rom_type(15 downto 0):= ("1110001", "1111001", --F,E
14                                         "1011110", "0111001", --d,C      Y(0)=a
15                                         "1111100", "1110111", --b,A      ---
16                                         "1101111", "1111111", --9,8 Y(5)=f|  Y(1)=b
17                                         "0000111", "1111101", --7,6      --- Y(6)=g
18                                         "1101101", "1100110", --5,4 Y(4)=e|  Y(2)=c
19                                         "1001111", "1011011", --3,2      --- 
20                                         "0000110", "0111111");--1,0      Y(3)=d
21     signal N : integer range 15 downto 0;
22 begin
23     N <= to_integer(unsigned(A));
24     Y <= ROM(N);
25 end Behavioral;

```

เรียกใช้ numeric_std package ที่อยู่ใน Library ieee เพราะใช้คำสั่ง to_integer และใช้ unsigned

ประกาศใช้ชนิดข้อมูล rom_type แบบ Unconstrained array

การประกาศใช้ Constant ที่มีการกำหนดค่าเริ่มต้น

ใช้คำสั่ง unsigned !! แปลงชนิดข้อมูล std_logic_vector เป็น unsigned และใช้ to_integer แปลงเป็น integer

2.15d) โค้ด VHDL ที่เขียนโดยประกาศใช้ชนิดข้อมูล std_logic_vector อะเรย์ 1 มิติ x 1 มิติแบบ Unconstrained array

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity ROM1 is
7     Port ( A : in unsigned(3 downto 0);
8             Y : out unsigned(6 downto 0));
9 end ROM1;
10 architecture Behavioral of ROM1 is
11     type rom_type is array (15 downto 0) of unsigned(6 downto 0);
12         -- gfedcba gfedcba
13     constant ROM : rom_type := ("1110001", "1111001", --F,E
14                                         "1011110", "0111001", --d,C      Y(0)=a
15                                         "1111100", "1110111", --b,A      ---
16                                         "1101111", "1111111", --9,8 Y(5)=f|  Y(1)=b
17                                         "0000111", "1111101", --7,6      --- Y(6)=g
18                                         "1101101", "1100110", --5,4 Y(4)=e|  Y(2)=c
19                                         "1001111", "1011011", --3,2      --- 
20                                         "0000110", "0111111");--1,0      Y(3)=d
21     signal N : integer range 15 downto 0;
22 begin
23     N <= to_integer(A);
24     Y <= ROM(N);
25 end Behavioral;

```

เรียกใช้ numeric_std package ที่อยู่ใน Library ieee เพราะใช้คำสั่ง to_integer และใช้ unsigned

ประกาศใช้ชนิดข้อมูล rom_type แบบ Constrained array

การประกาศใช้ Constant ที่มีการกำหนดค่าเริ่มต้น

ใช้คำสั่ง to_integer แปลงชนิดข้อมูล unsigned เป็นชนิดข้อมูล integer

2.15e) โค้ด VHDL ที่เขียนโดยประกาศใช้ชนิดข้อมูล unsigned อะเรย์ 1 มิติ x 1 มิติแบบ Constrained array

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity ROM1 is
7     Port ( A : in unsigned(3 downto 0);
8             Y : out unsigned(6 downto 0));
9 end ROM1;
10 architecture Behavioral of ROM1 is
11     type rom_type is array (integer range <>) of unsigned(6 downto 0);
12     -- gfedcba gfedcba
13     constant ROM : rom_type(15 downto 0) := (
14         "1110001", "1111001", --F,E
15         "1011110", "0111001", --d,C   Y(0)=a
16         "1111100", "1110111", --b,A   ---
17         "1101111", "1111111", --9,8 Y(5)=f | Y(1)=b
18         "0000111", "1111101", --7,6   --- Y(6)=g
19         "1101101", "1100110", --5,4 Y(4)=e | Y(2)=c
20         "1001111", "1011011", --3,2   --- Y(3)=d
21         "0000110", "0111111"); --1,0
22 begin
23     N <= to_integer(A);
24     Y <= ROM(N);
25 end Behavioral;

```

เรียกใช้ numeric_std package ที่อยู่ใน Library ieee เพราะใช้คำสั่ง to_integer และใช้ unsigned

ประการใช้ชนิดข้อมูล rom_type !! แบบ Unconstrained array

การประการใช้ Constant ที่มีการกำหนดค่าเริ่มต้น

ใช้คำสั่ง to_integer แปลงชนิดข้อมูล unsigned เป็นชนิดข้อมูล integer

2.15f) โค้ด VHDL ที่เปลี่ยนโดยประการใช้ชนิดข้อมูล unsigned อะเรย์ 1 มิติ x 1 มิติแบบ Unconstrained array

รูปที่ 2.15 โค้ด VHDL รูปแบบต่างๆ ของ ROM ที่ใช้เป็นตัวอครหัสตัวแสดงผลเซเว่นเซกเมนต์แบบโถดร่วม

รูปที่ 2.15c) และรูปที่ 2.15d) แสดงการประการใช้ชนิดข้อมูล std_logic_vector แบบ 1 มิติ x 1 มิติที่นิยามชนิดข้อมูลใหม่โดยผู้ใช้งาน ซึ่งจะเปลี่ยนชนิดข้อมูลเป็นแบบ Constrained array และ Unconstrained array ตามลำดับ โดยจะประการใช้ Constant ที่มีขนาด 7 บิตจำนวน 16 ค่าจากตารางในรูปที่ 2.15b) ไปเก็บไว้ใน ROM ตัวอย่าง เช่น ROM(0) = "0111111" (หรือแสดงผลเลข 0) โดยมีบิตที่ 6 ของ ROM(0) คือ ROM(0)(6) = '0' (บิตที่ 6 (MSB) ของ ROM(0) มีค่า = '0') เป็นต้น โค้ดในบรรทัดที่ 23 จะเป็นการแปลงชนิดข้อมูล std_logic_vector เป็น integer โดยทางอ้อม เนื่องจาก IEEE Std 1164 ไม่ได้นิยามฟังก์ชันแปลงชนิดข้อมูลนี้ไว้ ดังนั้นเราจึงแก้ปัญหานี้โดยแปลงชนิดข้อมูล std_logic_vector เป็น unsigned ด้วยคำสั่ง unsigned ก่อน จากนั้นจึงแปลงชนิดข้อมูล unsigned เป็น integer อีกครั้งด้วยคำสั่ง to_integer ซึ่งการใช้คำสั่ง to_integer และการใช้ชนิดข้อมูล unsigned นั้นจะต้องเรียกใช้ numeric_std package ที่ลูกคอมไฟล์เก็บไว้ใน Library ieee

จากตัวอย่างในรูปที่ 2.15e) และรูปที่ 2.15f) จะเป็นในทำนองเดียวกันกับตัวอย่างในรูปที่ 2.15c) และรูปที่ 2.15d) แต่จะประการใช้ชนิดข้อมูล unsigned ซึ่งโค้ดในบรรทัดที่ 23 จะเป็นการแปลงชนิดข้อมูล unsigned เป็น integer และที่สำคัญเราต้องไม่ลืมว่าสามารถแต่งตัวของชนิดข้อมูล unsigned นั้นเป็นชนิดข้อมูล std_logic

2.9 ตัวดำเนินการ

โอเปอเรเตอร์ (Operator) หรือตัวดำเนินการในภาษา VHDL ซึ่งตัวดำเนินการที่นิยามไว้ล่วงหน้าแล้ว (Predefined operator) ตาม IEEE Std 1076 โดยเราจะเรียงลำดับความสำคัญจากต่ำสุด (ด้านบน) ไปลำดับความสำคัญสูงสุด (ด้านล่าง) ได้แก่

- Logical operator ได้แก่ and , or , nand , nor , xor และ xnor
- Relational operator ได้แก่ = , /= , < , <= , > และ >=
- Shift operator ได้แก่ sll , srl , sla , sra , rol และ ror

- Adding operator ได้แก่ +, - และ &
- Sign ได้แก่ +, -
- Multiplying operator ได้แก่ *, /, mod และ rem
- Miscellaneous operator ได้แก่ **, abs และ not

ตัวดำเนินการหรือโอเปอเรเตอร์ (Operator) ต่างๆ ในภาษา VHDL ตามมาตรฐาน IEEE Std 1076, IEEE Std 1164 และ IEEE Std 1076.3 รวมทั้ง Synopsys packages (Nonstandard package) ที่ผู้อ่านควรทราบ ได้แก่

2.9.1 ตัวดำเนินการตรรกะ

ตัวดำเนินการตรรกะ (Logical operators) ได้แก่ and, or, nand, nor, xor และ xnor (รวม not) ซึ่งทุกตัวจะมีลำดับความสำคัญเท่ากัน ยกเว้น not จะมีลำดับความสำคัญสูงสุด ตัวดำเนินการนี้ถูกนิยามไว้ล่วงหน้าแล้วในมาตรฐานต่างๆ เป็นดังนี้

- ตาม IEEE Std 1076 ตัวดำเนินการตรรกะ (รวม not) จะใช้ได้กับชนิดข้อมูล bit, bit_vector และ boolean โดยที่ในกรณีที่เป็นชนิดข้อมูลของเรียนนั้นขนาดของเรย์ต้องเท่ากัน
- ตาม IEEE Std 1164 ตัวดำเนินการตรรกะนั้นจะใช้ได้กับชนิดข้อมูล std_logic (std_ulogic) และ std_logic_vector (std_ulogic_vector) ในกรณีที่ใช้ตัวดำเนินการตรรกะกับชนิดข้อมูลของเรียนนั้นขนาดของเรย์ต้องเท่ากัน การที่ตัวดำเนินการนี้ถูกขยายนิยามจากเดิมตาม IEEE Std 1076 เพื่อให้สามารถใช้ได้กับชนิดข้อมูล std_logic และ std_logic_vector ก็หมายความว่าตัวดำเนินการนี้ได้ทำ “เกิน” หรือ “Overload” หน้าที่จากนิยามเดิม ดังนั้นใน IEEE Std 1164 จึงรวมเรียกตัวดำเนินการเหล่านี้ว่า Overloaded logical operators และเรียก Function ที่ใช้นิยามตัวดำเนินการเหล่านี้ว่า Overloaded function ซึ่งต่อไปถ้าเจอกำกว่า “Overloading” หรือ “Overloaded” ก็จะมีความหมายในทำองนี้ กล่าวคือ ตัวดำเนินการที่มีชื่อหรือเครื่องหมายซ้ำกันแต่ทำหน้าที่เพิ่มขึ้นหรือมีความหมายแตกต่างไปจากเดิมที่นิยามไว้ใน IEEE Std 1076
- ตาม IEEE Std 1076.3 ตัวดำเนินการตรรกะใช้ได้กับชนิดข้อมูล unsigned และ signed โดยที่ขนาดของเรย์ต้องเท่ากัน ซึ่งตัวดำเนินการตรรกะในกรณีนี้จะเป็น Overloaded operators ! เช่นกัน

เนื่องจากตัวดำเนินการ not มีลำดับความสำคัญสูงกว่าตัวดำเนินการ and ดังนั้นโค้ด VHDL ของวงจร 2 to 4 Decoder ในรูปที่ 2.8a) นั้นสามารถเขียนโดยไม่ล่วงเดิมได้ดังรูปที่ 2.16 โดยโค้ด VHDL นี้จะบังคับให้ผลลัพธ์เหมือนเดิม

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DECODER2TO4 is
6     port ( A0,A1 : in STD_LOGIC;
7            D0,D1,D2,D3 : out STD_LOGIC);
8 end DECODER2TO4;
9
10 architecture BEHAVIORAL of DECODER2TO4 is
11 begin
12     D0 <= not A1 and not A0;
13     D1 <= not A1 and A0;
14     D2 <= A1 and not A0;
15     D3 <= A1 and A0;
16 end BEHAVIORAL;

```

รูปที่ 2.16 โค้ด VHDL ของวงจร 2 to 4 Decoder ที่เขียนโดยไม่ใช้งานเดิมที่คำสั่งต่างๆ

2.9.2 ตัวดำเนินการความสัมพันธ์

ตัวดำเนินการความสัมพันธ์ (Relational operators) ได้แก่ = (เท่ากับ), /= (ไม่เท่ากับ), >= (มากกว่าหรือเท่ากับ), > (มากกว่า), <= (น้อยกว่าหรือเท่ากับ) และ < (น้อยกว่า) โดยทุกตัวจะมีลำดับความสำคัญเท่ากัน ตัวดำเนินการนี้ใช้ได้กับชนิดข้อมูล Scalar และ Discrete array ตาม IEEE Std 1076 และชนิดข้อมูล signed และ unsigned ตาม IEEE Std 1076.3 แม้ว่า IEEE Std 1164 จะไม่ได้นิยามการใช้ตัวดำเนินการนี้กับชนิดข้อมูล std_logic (std_ulogic) และ std_logic_vector (std_ulogic_vector) ไว้ก็ตาม แต่ Xilinx Synthesis Tool หรือ XST ได้รองรับการใช้ชนิดข้อมูลนี้

ในกรณีที่ต้องการเปรียบเทียบชนิดข้อมูลของเรซึ่งมีขนาดของเรซึ่งไม่เท่ากันนั้น ตัวดำเนินการความสัมพันธ์จะกระทำโดยเริ่มจากสามาชิกของเรซึ่งด้านซ้ายสุดก่อน จากนั้นจะทยอยกระทำการกับสามาชิกตัวถัดไปทางด้านขวาท่าที่ของเรซึ่งมีอยู่เท่านั้น

2.9.3 ตัวดำเนินการรวมข้อมูล

ตัวดำเนินการรวมข้อมูลหรือ & (Concatenation operators) ใช้ในการรวมออบเจกต์ที่เป็นชนิดข้อมูลเดียวกันให้ได้เป็นชนิดข้อมูลของเรซึ่ง 1 มิติ ตัวดำเนินการนี้ใช้ได้กับชนิดข้อมูลทุกชนิดตาม IEEE Std 1076 แม้ IEEE Std 1076.3 และ IEEE Std 1164 จะไม่ได้นิยามการใช้ตัวดำเนินการนี้ แต่ Xilinx Synthesis Tool หรือ XST ได้รองรับการใช้ตัวดำเนินการนี้ ตัวอย่างเช่น

```
A <= "01" & "1101"; จะมีความหมายเดียวกับ A <= "011101";
B <= '0' & "111" & "0011"; จะมีความหมายเดียวกับ B <= "01110011";
C <= A & B; จะมีความหมายเดียวกับ C <= "01110101110011"
```

วิธีการรวมข้อมูลอีกรูปแบบหนึ่ง คือ Aggregate ซึ่งมีรูปแบบการเขียนดังนี้

```
Aggregate ::= ( [choice =>] identifier {, [choice =>] identifier} )
```

โดยที่ ::= มีความหมายว่า “สามารถถูกแทนที่ได้โดย”

ตัวอย่าง วิธีการรวมข้อมูลแบบ Aggregate เช่น ถ้ากำหนดให้ signal A : std_logic_vector (3 downto 0);

และให้	$A <= ('0', '1', '0', '0');$	- - Positional notation (กล่าวคือ ต้องเรียงตามตำแหน่ง)
หรือ	$A <= (0 => '0', 2 => '1', 1 => '0', 3 => '0');$	- - Named notation (กล่าวคือ ไม่ต้องเรียงตามตำแหน่ง)
ผลลัพธ์ คือ	$A <= "0100";$	

2.9.4 ตัวดำเนินการคณิตศาสตร์

ตัวดำเนินการคณิตศาสตร์ (Arithmetic operators) ได้แก่ +, -, *, / (หาร), ** (ยกกำลัง), MOD (Modulus), REM (Remainder) และ ABS (Absolute value) ตัวดำเนินการคณิตศาสตร์ที่นิยามไว้ล่วงหน้าแล้วในมาตรฐานต่างๆ เป็นดังนี้

- ตาม IEEE Std 1076 จะใช้ +, - และ ABS ได้กับชนิดข้อมูล Numeric (ได้แก่ชนิดข้อมูล integer, floating point และ physical) และจะใช้ * (คูณ), / (หาร), ** (ยกกำลัง) ได้กับชนิดข้อมูล integer types และ floating point types ยกเว้น

MOD (Modulus) และ REM (Remainder) จะใช้ได้กับชนิดข้อมูล integer types เราจะเห็นได้ว่าตัวดำเนินการคณิตศาสตร์นี้ไม่สามารถใช้กับชนิดข้อมูล bit และ bit_vector

- ตาม IEEE Std 1164 เราไม่สามารถใช้ตัวดำเนินการคณิตศาสตร์กับชนิดข้อมูล std_logic และ std_logic_vector ได้ เพราะไม่ได้นิยามไว้ การแก้ไขปัญหานี้ทำได้โดยเรียกใช้ Package ชื่อ std_logic_signed หรือ std_logic_unsigned ของ Synopsys (Nonstandard package) ที่คอมpile'ไว้ใน Library ieee ซึ่งจะยอมให้ทำการ +, -, * และ ABS ได้รวมกันว่า เป็นชนิดข้อมูล integer โดยที่ Xilinx Synthesis Tool หรือ XST จะรองรับการใช้ Package นี้ นอกจากนี้แล้ว std_logic_signed package หรือ std_logic_unsigned package ยังได้นิยามฟังก์ชันแปลง (Conversion functions) ชนิดข้อมูล std_logic_vector เป็น integer มีรายละเอียดดังตารางในรูปที่ 2.17 ในขณะที่การแปลงค่ากลับกันจาก integer เป็น std_logic_vector จะต้องเรียกใช้ std_logic_arith package

Conversion	Function
std_logic_vector to integer	conv_integer (expression)
integer to std_logic_vector	conv_std_logic_vector (expression, size)

รูปที่ 2.17 ฟังก์ชันแปลงชนิดข้อมูลระหว่าง std_logic และ integer

- ตาม IEEE Std 1076.3 จะสามารถใช้ตัวดำเนินการคณิตศาสตร์ได้กับชนิดข้อมูล unsigned และ signed รวมกันว่าเป็นชนิดข้อมูล integer โดยจะต้องเรียกใช้ Package ชื่อ Numeric_std ที่คอมpile'ไว้ใน Library ieee ซึ่งตัวดำเนินการคณิตศาสตร์นี้ได้แก่ +, -, *, / (หาร), MOD, REM และ ABS

2.9.5 การแปลงชนิดข้อมูล

เนื่องจากภาษา VHDL ไม่ยอมให้มีการทำ Operation หรือส่งผ่านค่าระหว่างออบเจกต์ที่มีชนิดข้อมูลต่างชนิดกัน ดังนั้น จึงต้องทำการแปลงชนิดข้อมูล (Type conversion) ให้เป็นชนิดเดียวกันเสียก่อน การแปลงชนิดข้อมูลมีรูปแบบดังนี้

- การแปลงชนิดข้อมูลโดยใช้ฟังก์ชัน (Type conversion function) มีรายละเอียดตามตารางในรูปที่ 2.13 รูปที่ 2.14 และ รูปที่ 2.17
- การแปลงชนิดข้อมูลที่มีความสัมพันธ์ใกล้ชิดกัน (Explicit type conversion) คือ การแปลงชนิดข้อมูลอย่างง่ายตามที่กำหนดไว้ใน IEEE Std 1076 โดยมีรูปแบบดังนี้

```
type_conversion ::= type_mark (expression)
```

โดยที่ 1 ::= มีความหมายว่า “สามารถถูกแทนที่ได้โดย”

2 type_mark คือ ชื่อชนิดข้อมูลที่ต้องการซึ่งเป็นผลลัพธ์ที่ได้จากการแปลงชนิดข้อมูลแล้ว

ชนิดข้อมูลที่สามารถใช้ได้กับการแปลงชนิดข้อมูลที่มีความสัมพันธ์ใกล้ชิด (Closely related) ได้แก่

- ชนิดข้อมูล integer หรือ floating point ที่เป็นชนิดข้อมูลเดียวกันแต่มีレンจ์ (Range) ไม่เท่ากัน หรือแปลงชนิดข้อมูล integer เป็น floating point หรือแปลงกลับกัน ซึ่งการแปลงลักษณะนี้อาจมีการปัดเศษขึ้นหรือลงเพื่อให้ได้ค่าที่ใกล้เคียง

ตัวอย่าง การแปลงชนิดข้อมูลนี้ เช่น ถ้าเราประกาศใช้ชนิดข้อมูลเป็นดังนี้

```
type X is range 0 to 100;
type Y is range 0 to 1000;
```

และถ้าเราประกาศใช้ Signal เป็นดังนี้

```
signal QT : X;
signal Q : Y;
```

เราสามารถเขียนโค้ดการแปลงชนิดข้อมูล (Explicit type conversion) จากชนิดข้อมูล X เป็น Y ได้ดังนี้คือ

```
Q <= Y(QT);
```

2) ชนิดข้อมูลของเรายังที่มีมิติเท่ากัน (ไม่สนใจทิศทาง downto หรือ to) โดยมีสมาชิกของอะเรย์แต่ละตัวต้องเป็นชนิดข้อมูลเดียวกัน เช่น การแปลงชนิดข้อมูล unsigned หรือ signed เป็น std_logic_vector หรือกลับกันมีรายละเอียดดังรูปที่ 2.18

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 2.18 ตัวอย่างการแปลงชนิดข้อมูล unsigned หรือ signed กับ std_logic_vector

ตัวอย่างการแปลงชนิดข้อมูลจาก unsigned เป็น std_logic_vector เช่น ถ้าเราประกาศใช้ Signal เป็นดังนี้

```
signal QT : unsigned (3 downto 0);
signal Q : std_logic_vector (3 downto 0);
```

เราสามารถเขียนโค้ดแปลงชนิดข้อมูล (Explicit type conversion) จาก unsigned เป็น std_logic_vector ได้ดังนี้ คือ

```
Q <= std_logic_vector (QT);
```

ซึ่งจะให้ผลลัพธ์เช่นเดียวกับการกำหนดค่าในลักษณะแบบตัวต่อตัวได้โดยตรง เนื่องจากสมาชิกของ unsigned และสมาชิกของ std_logic_vector เป็นชนิดข้อมูล std_logic เหมือนกัน กล่าวคือ

```
Q(0) <= QT(0);
Q(1) <= QT(1);
Q(2) <= QT(2);
Q(3) <= QT(3);
```

ตัวอย่างที่ 2.4 ฝึกการเขียนโค้ด VHDL ของวงจรบวก (แบบไม่คิดเครื่องหมายหรือ Unsigned) 2 บิตและเอาต์พุต 3 บิต โดยในรูปที่ 2.19a จะเป็นการเขียนโค้ดโดยเรียกใช้ Package ชื่อ std_logic_unsigned ของ Synopsys ซึ่งเป็น Nonstandard package รูปที่

2.19b) จะเป็นการเขียนโค้ดโดยเรียกใช้ Package ชื่อ Numeric_std แต่อินพุต/เอาต์พุตขังคงเป็นชนิดข้อมูล std_logic_vector ส่วนในรูปที่ 2.19c) จะเป็นการเขียนโค้ดโดยเรียกใช้ Package ชื่อ Numeric_std และอินพุต/เอาต์พุตเป็นชนิดข้อมูล unsigned

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;          틀องเรียกใช้ std_logic_unsigned package ใน Library ieee เพราะใช้ “+” กับชนิดข้อมูล std_logic_vector
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);          ใช้ Operator “+” กับชนิดข้อมูล std_logic_vector
15
16 end BEHAVIORAL;

```

2.19a) โค้ด VHDL ของวงจรบวกโดยเรียกใช้ Package ชื่อ std_logic_unsigned

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;          틀องเรียกใช้ Numeric_std package ใน Library ieee
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));          แปลงชนิดข้อมูลกลับไปเป็น std_logic_vector
15
16 end BEHAVIORAL;

```

2.19b) โค้ด VHDL ของวงจรบวกโดยเรียกใช้ Numeric_std package แต่อินพุต/เอาต์พุตขังคงเป็น std_logic_vector

```

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

```

2.19c) โค้ด VHDL ของวงจรบวกโดยเรียกใช้ Numeric_std package แต่อินพุต/เอาต์พุตเป็น unsigned

รูปที่ 2.19 โค้ด VHDL รูปแบบต่างๆ ของวงจรบวก (แบบ Unsigned) 2 บิตและเอาต์พุต 3 บิต

จากรูปที่ 2.19a) เป็นการเขียนโค้ด VHDL ของวงจรบวกที่มีการเรียกใช้ Package ชื่อ std_logic_unsigned การเขียนโค้ดลักษณะนี้จะมีความสะดวกและเขียนได้ง่ายที่สุดเนื่องจาก Package ชื่อ std_logic_unsigned นี้ยอมให้ชนิดข้อมูล std_logic หรือ std_logic_vector สามารถบวกกันได้โดยตรง แม้ว่า std_logic_unsigned เป็น Nonstandard package แต่ก็นิยมใช้กันอย่างแพร่หลาย และขอให้สังเกตว่าโค้ดในบรรทัดที่ 14 นั้นจะมีการทำ Concatenation ของ '0'&A และ '0'&B เพื่อบอกว่าอย่าง A และ B เป็นขนาด 3 บิตเพื่อไว้สำหรับตัวพารามิเตอร์ที่ผลบวกเกินจากค่า "011"

จากรูปที่ 2.19b) เป็นการเขียนโค้ด VHDL ของวงจรบวกที่มีการเรียกใช้ Package ชื่อ Numeric_std โดย Package นี้ยอมให้ชนิดข้อมูล unsigned สามารถบวกกันได้โดยตรง แต่เนื่องจากอินพุต/เอาต์พุตของวงจรเป็น std_logic_vector ดังนั้นโค้ดในบรรทัดที่ 14 นั้นก่อนการบวกจะต้องแปลงชนิดข้อมูล std_logic_vector เป็น unsigned เสียก่อน (หลังจากทำ Concatenation '0'&A และ '0'&B แล้ว) จากนั้นจึงแปลงชนิดข้อมูล unsigned ที่เป็นผลลัพธ์กลับมาเป็นชนิดข้อมูล std_logic_vector ตามเดิม

จากรูปที่ 2.19c) เป็นการเขียนโค้ด VHDL ของวงจรบวกที่มีการเรียกใช้ Package ชื่อ Numeric_std การเขียนโค้ดแบบนี้จะง่ายที่สุด เนื่องจาก Package นี้ยอมให้ชนิดข้อมูล unsigned สามารถบวกกันได้โดยตรง (หลังจากทำ Concatenation '0'&A และ '0'&B แล้ว) ดังนั้นอินพุต/เอาต์พุตจึงเป็นชนิดข้อมูล unsigned

Synthesis and Simulation Design Guide ของ Xilinx ได้แนะนำให้เขียนโค้ด VHDL ในระดับบนสุด (Top-level) นั้นจะต้องประกาศใช้พอร์ต (Top-level port declaration) เป็นชนิดข้อมูล std_logic และ std_logic_vector เท่านั้น เพราะซอฟต์แวร์ทุกจำลองการทำงานของ Xilinx ไม่รองรับการใช้อินพุต/เอาต์พุตที่เป็นชนิดข้อมูล unsigned และ signed (แม้ว่าสามารถแต่ละตัวในอะเรย์ของชนิดข้อมูล unsigned และ signed จะเป็นชนิดข้อมูล std_logic ก็ตาม) ดังนั้นในกรณีที่เป็นการเขียนโค้ดระดับบนสุดจึงต้องเขียนโค้ดดังรูปที่ 2.19a) หรือรูปที่ 2.19b) แต่อย่างไรก็ตามถ้าไม่ใช้การเขียนโค้ดระดับบนสุดก็สามารถเขียนได้ทั้ง 3 วิธี และหากต้องการเขียนในรูปแบบมาตรฐานก็ควรเขียนโดยใช้ VHDL ดังรูปที่ 2.19b) และรูปที่ 2.19c)

2.9.6 ตัวดำเนินการเลื่อน

ตัวดำเนินการเลื่อน (Shift operators) ที่นิยมไว้ล่วงหน้าแล้วในมาตรฐานต่างๆ เป็นดังนี้

- ตาม IEEE Std 1076 จะใช้ตัวดำเนินการเลื่อน ได้แก่ กับชนิดข้อมูลอะเรย์ 1 มิติของ bit (bit_vector) หรือ boolean โดยที่ตัวดำเนินการเลื่อนจะมีดังนี้

sll	คือ Shift left logical
srl	คือ Shift right logical
sla	คือ Shift left arithmetic
sra	คือ Shift right arithmetic
rol	คือ Rotate left
ror	คือ Rotate right

โดยที่ Shift logical และ Shift arithmetic จะมีความแตกต่างกันที่ตัวดำเนินการเลื่อน Shift logical จะเอา '0' ไปแทนที่บิตที่ถูกเลื่อน ในขณะที่ตัวดำเนินการเลื่อน Shift arithmetic นั้นจะเอาบิตสุดท้าย (LSB) หรือบิตแรก (MSB) ไปแทนที่บิตที่ถูกเลื่อน โดยจะขึ้นอยู่กับทิศทางการเลื่อนบิต

ตัวอย่างตัวดำเนินการเลื่อน เช่น ถ้ากำหนดให้ signal A : bit_vector (7 downto 0) := “01110001”;

ดังนี้ B <= A sll 2; คือ B <= A(5 downto 0) & “00”; จะได้ B <= “11000100”
 C <= A srl 2; คือ C <= “00” & A(7 downto 2); จะได้ C <= “00011100”
 D <= A sla 2; คือ B <= A(5 downto 0) & A(0) & A(0); จะได้ D <= “11000111”
 E <= A sra 2; คือ E <= A(7) & A(7) & A(7 downto 2); จะได้ E <= “00011100”

โดยที่ 1) A(5 downto 0) เป็น “Slices of array” ของ A มีความหมายเท่ากับ A(5)&A(4)&A(3)&A(2)&A(1)&A(0)
 2) A(7 downto 2) เป็น “Slices of array” ของ A มีความหมายเท่ากับ A(7)&A(6)&A(5)&A(4)&A(3)&A(2)

- ตาม IEEE Std 1164 เราไม่สามารถใช้ตัวดำเนินการเลื่อนกับชนิดข้อมูล std_logic_vector เพราะ IEEE Std 1164 ไม่ได้นิยามไว้ แต่อย่างไรก็ตามปัญหานี้แก้ไขได้โดยง่าย ตัวอย่างเช่น ถ้ากำหนดให้

signal A : std_logic_vector (7 downto 0) := “01110001”;

แล้วเราจะแทนคำสั่ง B <= A sll 2; ได้ด้วย B <= A(5 downto 0) & “00”; จะได้ B <= “11000100”

- ตาม IEEE Std 1076.3 จะใช้ตัวดำเนินการเลื่อนกับชนิดข้อมูล unsigned และ signed ได้แก่ SHIFT_LEFT, SHIFT_RIGHT, ROTATE_LEFT, ROTATE_RIGHT (โดยจำนวนครั้งที่เลื่อนเป็นชนิดข้อมูล natural) และ sll, srl, rol, ror (โดยจำนวนครั้งที่เลื่อนเป็นชนิดข้อมูล integer) ในกรณีใช้ SHIFT_RIGHT (หรือ srl) กับชนิดข้อมูล unsigned และ signed จะไม่เหมือนกัน โดยถ้าเป็นชนิดข้อมูล unsigned จะเดิม ‘0’ เข้าไปแทนที่บิตที่ถูกเลื่อนออก แต่ถ้าเป็นชนิดข้อมูล signed จะเดิมค่าล้อจิกของบิตเครื่องหมาย (Sign bit) เข้าไปแทนที่บิตที่ถูกเลื่อนออก โดยที่ Xilinx Synthesis Tool หรือ XST จะรองรับการใช้ตัวดำเนินการเลื่อนกับชนิดข้อมูล unsigned และ signed เช่น sll, srl, rol และ ror เท่านั้น แต่อย่างไรก็ตามเราจะยกตัวอย่างตัวดำเนินการเลื่อนทั้งหมดเพื่อให้ผู้อ่านทำความเข้าใจได้ดังนี้

```
เช่น signal U1 : UNSIGNED (7 downto 0) := “01101011”;
          signal U2 : UNSIGNED (7 downto 0) := “11101011”;
          signal S1 : SIGNED (7 downto 0) := “01101011”;
          signal S2 : SIGNED (7 downto 0) := “11101011”;
          signal N : NATURAL := 3 ;
```

ดังนี้ SHIFT_LEFT (U1, N) = “01011000”
 SHIFT_LEFT (S1, N) = “01011000”
 SHIFT_LEFT (U2, N) = “01011000”
 SHIFT_LEFT (S2, N) = “01011000”

SHIFT_RIGHT (U1, N) = “00001101”
 SHIFT_RIGHT (S1, N) = “00001101”
 SHIFT_RIGHT (U2, N) = “00011101”
 SHIFT_RIGHT (S2, N) = “11111101”

U1 sll N = “01011000”	U1 srl N = “00001101”
S1 sll N = “01011000”	S1 srl N = “00001101”
U2 sll N = “01011000”	U2 srl N = “00011101”
S2 sll N = “01011000”	S2 srl N = “11111101”

U1 rol N = “01011011”	U1 ror N = “01101101”
S1 rol N = “01011011”	S1 ror N = “01101101”
U2 rol N = “01011111”	U2 ror N = “01111101”
S2 rol N = “01011111”	S2 ror N = “01111101”

ตัวอย่างที่ 2.5 ฝึกเขียนโค้ด VHDL ของวงจรเลื่อนข้อมูล A (แบบ Unsigned) ไปทางซ้าย (Shift left logical) 1 บิต 2 บิตและ 3 บิต ตามลำดับ ซึ่ง A เป็นชนิดข้อมูล std_logic_vector โค้ดที่ได้แสดงดังรูปที่ 2.20a) และรูปที่ 2.20b)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_LEFT_A is
6     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
7             B,C,D : out STD_LOGIC_VECTOR (3 downto 0));
8 end SHIFT_LEFT_A;
9
10 architecture Behavioral of SHIFT_LEFT_A is
11 begin
12     B <= A(2 downto 0) & '0';
13     C <= A(1 downto 0) & "00";
14     D <= A(0) & "000";
15 end Behavioral;
```

2.20a) จะเป็นการเขียนโค้ดโดยใช้ “&” แทนการใช้ “sll” ที่ IEEE std 1164 ไม่ได้นิยามไว้

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.*;
5
6 entity SHIFT_LEFT_A is
7     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
8             B,C,D : out STD_LOGIC_VECTOR (3 downto 0));
9 end SHIFT_LEFT_A;
10
11 architecture Behavioral of SHIFT_LEFT_A is
12     signal X : UNSIGNED (3 downto 0);
13 begin
14     X <= UNSIGNED (A);
15     B <= STD_LOGIC_VECTOR ( X sll 1 );
16     C <= STD_LOGIC_VECTOR ( X sll 2 );
17     D <= STD_LOGIC_VECTOR ( X sll 3 );
18 end Behavioral;
```

ต้องเรียกใช้ Numeric_std package ใน Library ieee เพราะใช้ชนิดข้อมูล unsigned และใช้ “sll”

2.20b) จะเป็นการเขียนโค้ดโดยการใช้ “sll” ตาม IEEE Std 1076.3

รูปที่ 2.20 โค้ด VHDL วงจรเลื่อนข้อมูล(แบบไม่คิดเครื่องหมาย) ไปทางซ้าย (Shift left logical)

จากรูปที่ 2.20a) นั้นเป็นการเขียนโค้ดโดยใช้ “&” ส่วนรูปที่ 2.20b) นั้นจะเป็นการเขียนโค้ดโดยเรียกใช้ Package ชื่อ Numeric_std ซึ่งในการนี้เรามาจำเป็นต้องแปลงชนิดข้อมูลของ A จากชนิดข้อมูล std_logic_vector เป็น unsigned ก่อน เพื่อให้สามารถที่จะใช้กับ Operator “sll” ได้ เมื่อทำการเลื่อนข้อมูลแบบ “sll” เรียบร้อยแล้วจึงแปลงชนิดข้อมูลของผลลัพธ์กลับมาเป็น std_logic_vector อีกครั้ง

2.10 รูปแบบการเขียนโค้ด VHDL ทั่วไป

Xilinx Synthesis Tool (XST) หรือซอฟต์แวร์สังเคราะห์วงจรของบริษัท Xilinx ได้แนะนำให้ใช้ชนิดข้อมูล std_logic และ std_logic_vector สำหรับอินพุต/เอาต์พุตพอร์ต (I/O Port) ต่างๆ ที่เป็นการออกแบบในระดับบนสุด เพื่อให้ครอบคลุมทุกสถานะโลจิกที่ใช้ในซอฟต์แวร์จำลองการทำงาน รูปแบบการเขียนโค้ดที่ประกอบด้วยอย่างน้อย 3 ส่วนแสดงดังรูปที่ 2.21 คือ

- Using the package เป็นส่วนที่เรียกใช้ Package ที่เกี่ยวข้องกับการออกแบบ โค้ดที่อยู่ใน Package จะถูกคอมไพล์เป็นไฟล์ライบรารี (Library) ซึ่งในการออกแบบจะรู้ว่า สามารถเรียกใช้โค้ดจากส่วนนี้ได้ เช่นกัน (โค้ดที่ใช้ร่วมกัน)
- Entity declaration เป็นหน่วยของการออกแบบที่ใช้ในการกำหนดหรือประกาศใช้อินพุต/เอาต์พุตพอร์ต (I/O Port)
- Architecture body เป็นหน่วยของการออกแบบที่อธิบายพฤติกรรมการทำงานของระบบโดยติดต่อที่ออกแบบ

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DECODER2TO4 is
6     port ( A0,A1 : in STD_LOGIC;
7            Y : out STD_LOGIC_VECTOR(3 downto 0));
8 end DECODER2TO4;
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 2.21 ตัวอย่างแสดงรายละเอียดรูปแบบทั่วไปในการเขียนโค้ด VHDL (วงจร 2 to 4 Decoder)

2.10.1 การเรียกใช้ Package

การเรียกใช้ Package นั้นเราจะต้องเขียนไว้ที่ด้านบนสุดของโค้ด เช่น Package ชื่อ std_logic_1164 ที่อยู่ใน Library ชื่อ ieee ซึ่งการเรียกใช้ Library และ Package (Using the package) จะใช้คำสั่ง Use (Use clause) มีรูปแบบการเขียนทั่วไปดังนี้

```

library LIBRARY_NAME ;
use LIBRARY_NAME . PACKAGE_NAME . PACKAGE_PARTS ;

```

หมายเหตุ Package ชื่อ standard ที่อยู่ในライบรารีชื่อ std และ Package ชื่อ work ที่อยู่ในไลบรารีชื่อ work นั้นจะถูกเรียกใช้โดยอัตโนมัติถ้าหากการกำหนดล่วงหน้าไว้แล้ว (Default) จึงไม่ต้องเขียนส่วนที่เรียกใช้ทั้ง 2 Package ดังกล่าวแต่อย่างใด

2.10.2 ประกาศใช้เอนคิตตี้

ประกาศใช้เอนคิตตี้ (Entity Declaration) เป็นหน่วยของการออกแบบที่ใช้ในการกำหนดอินพุต/เอาต์พุตพอร์ต (I/O Port) ที่ใช้ติดต่อกับภายนอกของระบบโดยติดต่อที่เราออกแบบว่ามีอะไรบ้าง จากรูปที่ 2.21 ประกาศใช้เอนคิตตี้จะอยู่ในบรรทัดที่ 5 ถึงบรรทัดที่ 8 ซึ่งมีเฉพาะคำสั่ง Port (แต่ไม่มีคำสั่ง generic) โดยที่ประกาศใช้เอนคิตตี้จะมีรูปแบบการเขียนทั่วไปดังนี้

```
entity ENTITY_NAME is
    [ generic ( GENERIC DECLARATIONS ) ; ]
    [ port ( PORT DECLARATIONS ) ; ]
end [entity] [ENTITY_NAME];
```

โดยที่ 1) ประกาศใช้ Generic (Generic declarations) เป็นดังนี้

```
generic ( [GENERIC_NAME : TYPE [ := VALUE ] ] { ; GENERIC_NAME : TYPE [ := VALUE ] } );
```

2) ประกาศใช้ Port (Port declarations) เป็นดังนี้

```
port ( [ PORT_NAME : mode TYPE ] { ; PORT_NAME : mode TYPE } );
```

3) สัญลักษณ์ {...} หมายถึงจะมีหรือไม่มีก็ได้หรืออาจมีได้หลายตัว แต่ถ้าเป็น [...] หมายถึงจะมีหรือไม่มีก็ได้

4) การตั้งชื่อต่างๆ ต้องเป็นตามเกณฑ์ในข้อ 2.4

2.10.3 สถาปัตยกรรม

จากรูปที่ 2.21 สถาปัตยกรรม (Architecture) หรือสถาปัตยกรรมอ็อกซ์ (Architecture body) จะอยู่ในบรรทัดที่ 10 ถึงบรรทัดที่ 16 เป็นหน่วยออกแบบที่อธิบายฟังก์ชันหรือบรรยายพฤติกรรมการทำงานของวงจรที่ออกแบบ รูปแบบการเขียนทั่วไปเป็นดังนี้

```
architecture ARCHITECTURE_NAME of ENTITY_NAME is
    { DECLARATIVE_ITEM }
begin
    { CONCURRENT_STATEMENT }
end [architecture] [ARCHITECTURE_NAME];
```

โดยที่ 1) DECLARATIVE_ITEM ชื่นมี Declaration บางส่วนได้อธิบายไปแล้ว DECLARATIVE_ITEM เป็นดังนี้

- use clause (คู่รูปที่ 2.21 และข้อ 2.10.1)
- Subprogram declaration และ Subprogram body
- Type declaration และ/หรือ Subtype declaration
- Alias declaration เป็นการประกาศชื่อออบเจกต์ใหม่ที่เป็นส่วนย่อยของออบเจกต์เดิม มีรูปแบบดังนี้

```
alias ALIAS_NAME [ : SUBTYPE_INDICATION ] is ALIASED_OBJECT_NAME;
```

เช่น variable SIGN_NUMBER : std_logic_vector (0 to 31);

alias SIGN_BIT : std_logic is SIGN_NUMBER (0);

alias MAGNITUDE : std_logic_vector (30 downto 0) is SIGN_NUMBER (1 to 31);

- Constant declaration และ/หรือ Signal declaration
- Component declaration
- File declaration
- Attribute declaration และ/หรือ Attribute specification
- Configuration specification

2) CONCURRENT_STATEMENT คือ คำสั่งคอนแคร์เรนต์ ซึ่งจะอธิบายอย่างละเอียดในข้อที่ 2.13

3) สัญลักษณ์ {...} หมายถึงจะมีหรือไม่มีก็ได้หรืออาจมีได้หลายตัว แต่ถ้าเป็น [...] หมายถึงจะมีหรือไม่มีก็ได้

4) การตั้งชื่อ จะต้องเป็นตามเกณฑ์ในข้อ 2.4 ซึ่งซอฟต์แวร์ทุก XST ได้ Default ชื่อ Architecture เป็น BEHAVIORAL

ตัวอย่างที่ 2.6 ฝึกการเรียกใช้ Package และการใช้คำสั่ง generic ที่อยู่ใน Entity โดยนำโค้ด VHDL ของวงจรบวกในรูปที่ 2.19a) และรูปที่ 2.19b) ในตัวอย่างที่ 2.4 มาเขียนในรูปฟอร์มทั่วไปเป็นวงจรบวก N บิตแสดงดังรูปที่ 2.22a) และรูปที่ 2.22b)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;          // ต้องเรียกใช้ std_logic_unsigned package เพื่อใช้
                                            // Operator “+” กับชนิดข้อมูล std_logic_vector
5
6 entity ADDER_2BIT is
7     generic ( N : integer := 2);           // การใช้คำสั่ง generic ถ้า N=2 จะเป็นวงจรบวก 2 บิต
8     port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
9            C : out STD_LOGIC_VECTOR (N downto 0));
10 end ADDER_2BIT;
11
12 architecture BEHAVIORAL of ADDER_2BIT is
13 begin
14
15     C <= ('0' & A) + ('0' & B);          // ใช้ Operator “+” กับชนิดข้อมูล std_logic_vector
16
17 end BEHAVIORAL;

```

2.22a) โค้ด VHDL ของวงจรบวก 2 บิตที่มีการเรียกใช้ Package ชื่อ std_logic_unsigned และคำสั่ง generic

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;                // ต้องเรียกใช้ Numeric_std package เพื่อใช้ชนิดข้อมูล unsigned
5
6 entity ADDER_2BIT is
7     generic ( N : integer := 2);           // การใช้คำสั่ง generic ถ้า N=2 จะเป็นวงจรบวก 2 บิต
8     port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
9            C : out STD_LOGIC_VECTOR (N downto 0));
10 end ADDER_2BIT;
11
12 architecture BEHAVIORAL of ADDER_2BIT is
13 begin
14
15     C <= std_logic_vector(unsigned('0' & A) + unsigned('0' & B));      // แปลงชนิดข้อมูลกลับไปเป็น std_logic_vector
16
17 end BEHAVIORAL;

```

2.22b) โค้ด VHDL ของวงจรบวก 2 บิตที่มีการเรียกใช้ Numeric_std package และคำสั่ง generic

รูปที่ 2.22 โค้ด VHDL ของวงจรบวก 2 บิต โดยใช้คำสั่ง generic โดยการกำหนดให้ N = 2

รูปที่ 2.22 แสดงการเขียนโค้ดรูปแบบทั่วไปโดยใช้คำสั่ง generic เช่น ถ้าต้องการวงจรบวก 16 บิตก็สามารถทำได้โดยแก้ไขโค้ดเพียงเล็กน้อย โดยแก้ N = 16 (แก้ไขเฉพาะบรรทัดที่ 7 เพียงที่เดียว) ในรูปที่ 2.22a) หรือรูปที่ 2.22b) เท่านั้น

2.11 Attribute

Attribute เป็นคุณสมบัติของ VHDL ที่ยอมให้มีการเพิ่มรายละเอียดให้แก่องค์กร์ (Object) ที่เป็น Signal, Variable หรือชนิดข้อมูล เช่น การระบุขอบเขต การทริก (Trig) เป็นต้น ซึ่งการทริกมีประโยชน์ในการเขียนโค้ดวงจรซีเควล (Sequential circuit) และโค้ดสำหรับใช้จัดการทำงาน และส่วนของโปรแกรมย่อยต่างๆ Attribute มีรูปฟอร์มการเขียนดังนี้

```
X'attribute_name'
```

โดยที่ attribute_name คือชื่อ Attribute ของอบเจกต์หรือชนิดข้อมูล X และ สัญลักษณ์ “’” ให้ออกเสียงว่า ทิค (Tick)

ตัวอย่างโค้ดบางส่วนของ Package ชื่อ Std_logic_1164 ในรูปที่ 2.23 ที่แสดงตัวอย่างการใช้ Attribute คือ LENGTH, LOW และ RANGE โดยให้สังเกตที่สัญลักษณ์ “’” (ไม่ใช่ “”) โดยในขั้นตอนนี้ผู้อ่านยังไม่ต้องสนใจความหมายที่อยู่ในโค้ด

```
PACKAGE BODY Std_logic_1164 IS
-- local types
TYPE stdlogic_id IS ARRAY (std_ulogic) OF std_ulogic;
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;
-- resolution function
CONSTANT resolution_table : stdlogic_table := (
-- | U X O 1 Z W L H - | |
-- |
( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
( 'U', 'X', 'O', 'X', 'O', 'O', 'O', 'X' ), -- | O |
( 'U', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ) -- | - |
);
FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
VARIABLE result : std_ulogic := 'Z'; -- weakest state default
BEGIN
-- the test for a single driver is essential; otherwise, the
-- loop would return 'X' for a single driver of '-' and that
-- would conflict with the value of a single driver unresolved
-- signal.
IF (s'LENGTH = 1) THEN RETURN s(s'LOW);
ELSE
FOR i IN s'RANGE LOOP
result := resolution_table(result, s(i));
END LOOP;
END IF;
RETURN result;
```

รูปที่ 2.23 ตัวอย่างการใช้ Attribute ที่เขียนไว้ใน Package body ของ Std_logic_1164

Attribute แบ่งตามชั้นได้ 2 ชั้น คือ Attribute ที่กำหนดล่วงหน้าไว้แล้ว (Predefined attributes) ในมาตรฐาน IEEE ต่างๆ และ Attribute กำหนดโดยผู้ใช้งาน (User defined attributes) โดยที่ชนิดของ Attribute นั้นจะจำแนกตามการส่งค่ากลับ (Return) ว่าเป็น value, type, range, function หรือ signal

2.11.1 Predefined attributes

1) Type attribute

Type attribute เป็น Attribute ที่ใช้กับชนิดข้อมูล Enumeration, Integer และ Physical และเราเรียก Enumeration types และ Integer types ว่า “Discrete types” Type attribute ที่ผู้อ่านควรทราบ ได้แก่

Attribute	Returns	Kind of attribute
T'left	Left bound of T	Value
T'right	Right bound of T	Value
T'high	Upper bound of T	Value
T'low	Lower bound of T	Value
T'pos(X)	Position number of X in T	Function
T'val(X)	value in T of position X	Function
T'succ(X)	T'val(T'pos(X)+1)	Function
T'pred(X)	T'val(T'pos(X)-1)	Function
T'leftof(X)	T'pred(X) if T is ascending T'succ(X) if T is descending	Function
T'rightof(X)	T'succ(X) if T is ascending T'pred(X) if T is descending	Function
T'image(X)	String representing value of X	Function
T'value(X)	Value of string X	Function

ในกรณีที่เป็น Ascending range (เช่น 0 to 15) นั้น T'left = T'low, T'right = T'high, T'leftof(X) = T'pred(X) และ T'rightof(X) = T'succ(X) แต่ถ้าเป็น Decending range (เช่น 15 downto 0) นั้น T'left = T'high, T'right = T'low, T'leftof(X) = T'succ(X) และ T'rightof(X) = T'pred(X)

ตัวอย่างการหาค่าที่ Return ของ Type attribute เช่น ถ้าเราประกาศใช้ชนิดข้อมูล Integer และ Enumeration ดังนี้

```
type WORD is range 15 downto 0;      -- -15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0 (Decending range)
type ADDR is range -5 to 5          -- - - -5,-4,-3,-2,-1,0,1,2,3,4,5       (Ascending range)
type COLOR is (BL, G, Y, R);
```

จะได้ค่าที่ Return ดังนี้

WORD'left	= 15	WORD'right	= 0
WORD'high	= 15	WORD'low	= 0
ADDR'left	= -5	ADDR'right	= 5
ADDR'high	= 5	ADDR'low	= -5

COLOR'left	= BL	COLOR'right	= R
COLOR'high	= R	COLOR'low	= BL
COLOR'pos(BL)	= 0	COLOR'val(0)	= BL
COLOR'pos(R)	= 3	COLOR'value("BL")	= BL
COLOR'succ(G)	= Y	COLOR'pred(Y)	= G
COLOR'leftof(Y)	= G	COLOR'rightof(Y)	= R

2) Array attribute

Attribute ທີ່ໃຊ້ກັນອອນເຈັກຕໍ່ຮູ້ອໜີນິດຂໍ້ມູນລວະເຮັດ 1 ມີຕີ (ໄມ້ຕ້ອງຮະບຸ (N)) ແລະ ຂໍ້ມູນລວະເຮັດຫາຍົມຕີ (ຕ້ອງຮະບຸ (N)) ເຊັ່ນ

Attribute	Returns	Kind of attribute
A'left(N)	left bound of Nth index of A	Function
A'right(N)	right bound of Nth index of A	Function
A'high(N)	high bound of Nth index of A	Function
A'low(N)	low bound of Nth index of A	Function
A'range(N)	range of Nth index of A	Range
A'reverse_range(N)	reverse range of Nth index of A	Range
A'length(N)	number of values in Nth index of A	Value
A'ascending	true if array range ascending	Value

ຕ້ວອງຢ່າງການທາຄ່າທີ່ Return ຂອງ Array attribute ເຊັ່ນ ຕ້າງປະກາດໃຫ້ໜີນິດຂໍ້ມູນລວະເຮັດ 1 ດັ່ງນີ້

```
type A1D is array (3 downto 0) of std_logic;
type A2D is array (0 to 15, 7 downto 0) of std_logic;
signal Y : std_logic_vector (31 downto 0);
```

ຈະໄດ້ຄ່າທີ່ Return ດັ່ງນີ້

A1D'left	= 3	A1D'right	= 0
A1D'high	= 3	A1D'low	= 0
A1D'range	= 3 downto 0	A1D'reverse_range	= 0 to 3
A1D'length	= 4	A1D'ascending	= False
A2D'left(1)	= 0	A2D'right(1)	= 15
A2D'high(1)	= 15	A2D'low(1)	= 0
A2D'range(1)	= 0 to 15	A2D'reverse_range(1)	= 15 downto 0
A2D'length(1)	= 16	A2D'ascending(1)	= True
A2D'left(2)	= 7	A2D'right(2)	= 0
A2D'high(2)	= 7	A2D'low(2)	= 0
A2D'range(2)	= 7 downto 0	A2D'reverse_range(2)	= 0 to 7
A2D'length(2)	= 8	A2D'ascending(2)	= False
Y'left	= 31	Y'right	= 0
Y'high	= 31	Y'low	= 0
Y'range	= 31 downto 0	Y'reverse_range	= 0 to 31
Y'length	= 32	Y'ascending	= False

3) Signal attribute

Signal attribute เป็น Attribute ที่ใช้กับ Signal เช่น

Attribute	Returns	Kind of attribute
S'event	true if an event has just occurred on S in current cycle	Function
S'active	true if signal S is active in the current cycle	Function
S'last_value	value of S prior to latest change	Function
S'last_event	time at which S last changed	Function
S'last_active	time at which S last active	Function
S'delayed[(T)]	value of S delayed by T time units	Signal
S'stable[(T)]	true if no event on S over last T time units	Signal
S'quiet[(T)]	true if S quiet for T time units	Signal
S'transaction	bit value which toggles each time signal S changes	Signal

ในทางปฏิบัติ Attribute นี้ส่วนใหญ่กับโค้ดจำลองการทำงาน และมีบางตัวที่ใช้กับโค้ดสังเคราะห์วงจรเช่น “event”

เช่น C' event มีความหมายว่า การเปลี่ยนแปลง (ด้วยขอบขาขึ้นหรือขาลง) ของสัญญาณ C (Clock)

ถ้าเป็น C' event and C = '1' มีความเดียวกับ rising_edge (C) มีความหมายว่า ทริกด้วยขอบขาขึ้นของสัญญาณ C

C' event and C = '0' มีความเดียวกับ falling edge (C) มีความหมายว่า ทริกด้วยขอบขาลงของสัญญาณ C โดยที่ rising_edge และ falling_edge จะถูกนิยามไว้ใน IEEE Std 1164 (ดังนั้นจึงใช้กับ IEEE Std 1076.3 ได้เช่นกัน)

XST จะรองรับการใช้ Attribute ที่กำหนดล่วงหน้าไว้แล้ว (Predefined attributes) บางตัวเท่านั้น ได้แก่ high, low, left, right, range, reverse_range, length, pos, ascending, event และ last_value โค้ดที่ Attribute อื่นๆ นั้น XST จะละเว้น (Ignored)

2.11.2 User defined attributes

User defined attributes จะเป็น Attributes ที่กำหนดโดยผู้ใช้ จะมีประโยชน์ในการกำหนดเงื่อนไขบังคับ (Constraints) ต่างๆ ลงในโค้ด VHDL เพื่อให้การออกแบบวงจรเป็นไปอย่างมีประสิทธิภาพ ซึ่งจะมีตัวอย่างการใช้งานในบทที่ 4 ข้อ 4.6.6

2.12 คำสั่งที่ใช้ในการเขียนโค้ด VHDL

การออกแบบวงจรดิจิตอลหรือแบบจำลองคิจิตอล (Digital modeling) ด้วยการเขียนโค้ด VHDL สามารถทำได้รวดเร็ว โดยผู้ออกแบบอาจไม่จำเป็นต้องลงลึกในรายละเอียดการออกแบบระดับล็อกอิ吉เกต เราอาจเขียนโค้ดโดยใช้คำสั่งคอนแคร์เรนต์ (Concurrent statement) และ/หรือใช้คำสั่งชีเควนเชียล (Sequential statement) คำสั่งคอนแคร์เรนต์ทุกคำสั่งจะทำงานพร้อมกัน โดยไม่คำนึงถึงลำดับ (Order) ก่อนหลัง แต่คำสั่งชีเควนเชียลจะมีลำดับความสำคัญก่อนหลังและต้องเขียนไว้ในคำสั่งโปรดเซส (Process statement) Function หรือ Procedure คำสั่ง Process เป็นคำสั่งคอนแคร์เรนต์แต่ภายในจะเป็นคำสั่งชีเควนเชียล โค้ดของวงจรคอมบินेशัน (Combination circuit) อาจเขียนได้ด้วยคำสั่งคอนแคร์เรนต์และ/หรือคำสั่งชีเควนเชียลก็ได้ ในขณะที่โค้ดของวงจรชีเควนเชียล (Sequential circuit) จะต้องเขียนด้วยคำสั่งชีเควนเชียลเท่านั้น น้อยกว่าที่ผู้อ่านมักจะสับสนระหว่าง “คำสั่งชีเควนเชียล” กับ “วงจรชีเควนเชียล” เอาต์พุตของวงจรชีเควนเชียลจะขึ้นกับอินพุตปัจจุบันและอินพุตอดีต ซึ่งก็หมายความว่า วงจรลักษณะนี้มีความจำเป็นต้องใช้หน่วยความจำ ในขณะที่วงจรคอมบินेशันนั้นเอาต์พุตจะขึ้นกับอินพุตปัจจุบันเท่านั้น

โค้ด VHDL บางประเภทสามารถนำจำลองการทำงานได้ (Simulatable) แต่อาจจะไม่สามารถสังเคราะห์เป็นวงจรได้ (Synthesizable) ซึ่งทุกบอร์ดจะเน้นการเขียนโค้ดเพื่อสังเคราะห์เป็นวงจรได้เป็นหลัก ยกเว้นบทที่ 6 ที่เป็นการเขียน Testbench

2.13 คำสั่งคอนแคร์เรนต์

คำสั่งคอนแคร์เรนต์ได้แก่ Simple signal assignment statement, Selected signal assignment statement, Conditional signal assignment statement, Component instantiation statement, Generate statement, Block statement, Process statement และ Concurrent procedure calls เราเรียกโดยแบบคอนแคร์เรนต์อีกชื่อหนึ่งว่าเป็นแบบ Data flow (Dataflow descriptions)

2.13.1 Simple signal assignment 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

2.13.2 Selected signal assignment statement

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 2.4

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 และเราอาจกำหนดค่าว่า "UNAFFECTED" ใน EXPRESSION กรณีที่ไม่มีการทำอะไรเลย (หรือ No action ซึ่ง XST ดึงแต่ ISE WebPACK9.1 ที่นี่ไปจึงจะรองรับการใช้คำสั่งนี้)

เนื่องจากแต่ละ CHOICE ไม่มีลำดับความสำคัญ (Priority) ดังนั้นทุก CHOICE จะต้องไม่ซ้ำชื่อกันและต้องกำหนดให้ครบทุก CHOICE ที่เป็นไปได้

2.13.3 Conditional signal assignment statement

คำสั่ง Conditional signal assignment มีรูปแบบการเขียนดังนี้

```
[ label :] TARGET <= { EXPRESSION when CONDITION else }
                      EXPRESSION ;
```

- โดยที่
- 1) สัญลักษณ์ {...} หมายถึงจะมีหรือไม่มีก็ได้หรืออาจมีได้หลายตัว แต่ถ้าเป็น [...] หมายถึงจะมีหรือไม่มีก็ได้
 - 2) การตั้งชื่อต่างๆ ต้องเป็นตามเกณฑ์ในข้อ 2.4

คำสั่งนี้จะกำหนด EXPRESSION ให้กับ TARGET เมื่อตรวจสอบ CONDITION เป็นจริง (True) อันแรกสุดก่อน โดยจะไม่สนใจ CONDITION ที่เป็นจริงที่อยู่ในลำดับถัดไป (ซึ่งอาจมีหลาย CONDITION) ดังนั้นแต่ละ CONDITION ในคำสั่งนี้จึงมีลำดับความสำคัญ (Priority) และในการซึ่งไม่พบ CONDITION เป็นจริง (True) ที่จะเป็นการกำหนด EXPRESSION สุดท้ายให้กับ TARGET

ตัวอย่างที่ 2.7 ฝึกใช้คำสั่ง Selected signal assignment และ Conditional signal assignment ในการออกแบบวงจรแมติเพล็กเซอร์ 4 อินพุต 1 เอาต์พุต โดยใช้สัญญาณ Control S0 และ S1 เป็นตัวเลือกอินพุต ดังมีรายละเอียดตามตารางความจริงในรูปที่ 2.24a) ซึ่งสามารถเขียนโค้ด VHDL ได้ดังรูปที่ 2.24b) ถึงรูปที่ 2.24d)

Control		Output	Remark
S1	S0	O	
0	0	A	$O = A$
0	1	B	$O = B$
1	0	C	$O = C$
1	1	D	$O = D$

2.24a) ตารางความจริงของแมติเพล็กเซอร์ 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

การรวมชนิดข้อมูลเป็นแบบหลายบิต โดยใช้ "&" (Concatenation)

2.24b) โค้ดของแมติเพล็กเซอร์ที่เขียนด้วยคำสั่ง 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;

```

2.24c) โค้ด凰จรมัลติเพล็กเซอร์ที่เขียนด้วยคำสั่ง Conditional 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 begin
12     O <=  A when (S1='0' and S0='0') else
13         B when (S1='0' and S0='1') else
14         C when (S1='1' and S0='0') else
15         D;                                         --(S1='1' and S0='1')
16 end BEHAVIORAL;

```

2.24d) โค้ดที่เขียนด้วยคำสั่ง Conditional signal assignment แต่ไม่ใช้ตัวคำแนะนำการรวมข้อมูล

รูปที่ 2.24 โค้ด凰จรมัลติเพล็กเซอร์ 4 อินพุต 1 เอาต์พุต

จากโค้ดในรูปที่ 2.24b) 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 $S0 = '0'$) ก็ให้กำหนดค่าเอาต์พุต O จากอินพุต C และถ้า SEL มีสถานะลอจิกเป็นค่าที่เหลือทั้งหมด (when others) โดยรวมสถานะในกรณีที่ $SEL = "11"$ เช่นไปด้วย ก็ให้กำหนดค่าเอาต์พุต O จากอินพุต D การใช้คำว่า when others จึงสมมุติเป็นการเขียนเพื่อให้ได้ครอบคลุมทุกเงื่อนไขที่เป็นไปได้ทั้งหมด เพราะว่า $S1$ และ $S0$ เป็นชนิดข้อมูล std_logic แต่ละตัวจึงมีสถานะลอจิกได้ 9 สถานะ

จากโค้ดในรูปที่ 2.24c) มีความหมายว่า เมื่อ $SEL = "00"$ ($S1 = '0'$ และ $S0 = '0'$) เป็นจริงแล้วให้กำหนดค่าเอาต์พุต O จากอินพุต A แต่ถ้า $SEL = "00"$ เป็นเท็จจะไปทำ CONDITION ไปถัดไป ถ้า $SEL = "01"$ ($S1 = '0'$ และ $S0 = '1'$) เป็นจริงแล้วให้กำหนดค่าเอาต์พุต O จากอินพุต B แต่ถ้า $SEL = "01"$ เป็นเท็จจะไปทำ CONDITION ถัดไป และถ้า $SEL = "10"$ ($S1 = '1'$ และ $S0 = '0'$) เป็นจริงแล้วให้กำหนดค่าเอาต์พุต O จากอินพุต C ถ้าไม่พบ CONDITION ที่เป็นจริง (โดยรวมกรณีที่ $SEL = "11"$ เช่นไปด้วยแล้ว) ก็ให้กำหนดค่าเอาต์พุต O จาก EXPRESSION สุดท้าย คือ อินพุต D

ในรูปที่ 2.24d) เป็นการเขียนโค้ดโดยไม่ใช้ตัวคำแนะนำการรวมข้อมูลหรือ & (Concatenation operators) ของ $S0$ และ $S1$

ตัวอย่างที่ 2.8 ฝึกใช้คำสั่ง Selected signal assignment และ Conditional signal assignment ในการออกแบบวงจรแมตติเพล็กเซอร์ 4 อินพุต 1 เอาต์พุต โดยใช้สัญญาณ Control S เป็นบันধนาด 2 บิต (คือ S(1) และ S(0)) เป็นตัวเลือกอินพุต ดังมีรายละเอียดตามตารางความจริงในรูปที่ 2.25a) ซึ่งสามารถเขียนโค้ด VHDL ได้ดังรูปที่ 2.25b) และรูปที่ 2.25c)

Control S	Output O	Remark
“00”	A	O = A
“01”	B	O = B
“10”	C	O = C
“11”	D	O = D

2.25a) ตารางความจริงของวงจรแมตติเพล็กเซอร์ 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 : in STD_LOGIC;
7         S : in STD_LOGIC_VECTOR(1 downto 0);
8         O : out STD_LOGIC);
9 end MUX4TO1;
10
11 architecture BEHAVIORAL of MUX4TO1 is
12 begin
13   with S select
14     O <=  A when "00",
15           B when "01",
16           C when "10",
17           D when others;  -- S="11"
18 end BEHAVIORAL;

```

2.25b) โค้ดของวงจรแมตติเพล็กเซอร์ที่เขียนด้วยคำสั่ง Selected signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity MUX4TO1 is
6   port ( A,B,C,D : in STD_LOGIC;
7         S : in STD_LOGIC_VECTOR(1 downto 0);
8         O : out STD_LOGIC);
9 end MUX4TO1;
10
11 architecture BEHAVIORAL of MUX4TO1 is
12 begin
13   O <=  A when S="00" else
14           B when S="01" else
15           C when S="10" else
16           D;                      -- S="11"
17 end BEHAVIORAL;

```

2.25c) โค้ดของวงจรแมตติเพล็กเซอร์ที่เขียนด้วยคำสั่ง Conditional signal assignment

รูปที่ 2.25 โค้ดของวงจรแมตติเพล็กเซอร์ 4 อินพุต 1 เอาต์พุต

ตัวอย่างที่ 2.9 ฝึกใช้คำสั่ง Selected signal assignment และ Conditional signal assignment ในการออกแบบวงจรบวกและลบ (unsigned) 4 บิต นิ้ว A และ B เป็นอินพุตและ RESULT เป็นเอาต์พุต 5 บิต โดยมี OPER ควบคุมการบวกหรือลบ โดยโค้ดต่อๆ ที่ได้แสดงดังรูปที่ 2.26a) ถึงรูปที่ 2.26f)

หมายเหตุ เมื่อ $A < B$ แล้วถ้า $A-B$ ก็จะทำให้ได้ผลลัพธ์แบบ 2's complement

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;          틉ອງເຮັດໃຫ້ std_logic_unsigned package ເພຣະໃຫ້
5                                         Operator “+”ແລະ “-” ກັບໜົນດີຂໍ້ມູນ std_logic_vector
6 entity ADD_SUB_4BIT is
7     Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
8             OPER : in STD_LOGIC;
9             RESULT : out STD_LOGIC_VECTOR (4 downto 0));
10 end ADD_SUB_4BIT;
11
12 architecture Behavioral of ADD_SUB_4BIT is
13 begin
14     with OPER select
15         RESULT <= ('0' &A)+('0' &B) when '0',
16                         ('0' &A)-('0' &B) when others;          ໃຫ້ Operator “+” ແລະ “-” ກັບໜົນ
17 end Behavioral;                        ຂໍ້ມູນ std_logic_vector

```

2.26a) ໂຄສ່ວງຈະບວກແລະລວບອ່າງຈ່າຍທີ່ເປີຍນຳວຍຄໍາສັ່ງ Selected signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;          틉ອງເຮັດໃຫ້ std_logic_unsigned package ເພຣະໃຫ້
5                                         Operator “+”ແລະ “-” ກັບໜົນດີຂໍ້ມູນ std_logic_vector
6 entity ADD_SUB_4BIT is
7     Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
8             OPER : in STD_LOGIC;
9             RESULT : out STD_LOGIC_VECTOR (4 downto 0));
10 end ADD_SUB_4BIT;
11
12 architecture Behavioral of ADD_SUB_4BIT is
13 begin
14     RESULT <= ('0' &A)+('0' &B) when OPER = '0' else          ໃຫ້ Operator “+” ແລະ “-” ກັບ
15                         ('0' &A)-('0' &B);                      ຜົນດີຂໍ້ມູນ std_logic_vector
16 end Behavioral;

```

2.26b) ໂຄສ່ວງຈະບວກແລະລວບອ່າງຈ່າຍທີ່ເປີຍນຳວຍຄໍາສັ່ງ Conditional signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;                틉ອງເຮັດໃຫ້ numeric_std package ເພຣະໃຫ້ Operator
5                                         “+” ແລະ “-” ກັບໜົນດີຂໍ້ມູນ unsigned
6 entity ADD_SUB_4BIT is
7     Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
8             OPER : in STD_LOGIC;
9             RESULT : out STD_LOGIC_VECTOR (4 downto 0));
10 end ADD_SUB_4BIT;
11
12 architecture Behavioral of ADD_SUB_4BIT is
13 begin
14     with OPER select
15         RESULT <= STD_LOGIC_VECTOR(unsigned('0' &A) + unsigned('0' &B)) when '0',
16                         STD_LOGIC_VECTOR(unsigned('0' &A) - unsigned('0' &B)) when others;          ໃຫ້ “+” ແລະ “-” ກັບ std_logic_vector ໄມ໌ໄດ້ ຈຶ່ງຕໍ່ອັງ
17 end Behavioral;                        ແປລ່ງເປັນ unsigned ເມື່ອເສົ່ວ່ຈແລ້ວຈຶ່ງແປລ່ງກັບຄືນ

```

2.26c) ໂຄສ່ວງຈະບວກແລະລວບອ່າງຈ່າຍທີ່ເປີຍນຳວຍຄໍາສັ່ງ Selected signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;           틉องเรียกใช้ numeric_std package !พรະใช้ Operator
5                                         “+” และ “-” กับชนิดข้อมูล unsigned
6
7 entity ADD_SUB_4BIT is
8     Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
9             OPER : in STD_LOGIC;
10            RESULT : out STD_LOGIC_VECTOR (4 downto 0));
11
12 end ADD_SUB_4BIT;
13
14 architecture Behavioral of ADD_SUB_4BIT is
15 begin
16     RESULT <= STD_LOGIC_VECTOR(unsigned('0'&A) + unsigned('0'&B)) when OPER='0' else
17                 STD_LOGIC_VECTOR(unsigned('0'&A) - unsigned('0'&B));           ใช้ “+” และ “-” กับ std_logic_vector ไม่ได้ จึงต้อง
18                                         แปลงเป็น unsigned เมื่อเสร็จแล้วจึงแปลงกลับคืน
19
20 end Behavioral;

```

2.26d) โค้ดวงจรบวกและลบอย่างง่ายที่เขียนด้วยคำสั่ง Conditional signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;           틉องเรียกใช้ numeric_std package !พรະใช้ Operator
5                                         “+” และ “-” กับชนิดข้อมูล unsigned
6
7 entity ADD_SUB_4BIT is
8     Port ( A,B : in unsigned (3 downto 0);
9             OPER : in STD_LOGIC;
10            RESULT : out unsigned (4 downto 0));
11
12 end ADD_SUB_4BIT;
13
14 architecture Behavioral of ADD_SUB_4BIT is
15 begin
16     with OPER select
17         RESULT <= ('0'&A) + ('0'&B) when '0',
18                         ('0'&A) - ('0'&B) when others;
19
20 end Behavioral;

```

2.26e) โค้ดวงจรบวกและลบอย่างง่ายที่เขียนด้วยคำสั่ง Selected signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;           틉องเรียกใช้ numeric_std package !พรະใช้ Operator
5                                         “+” และ “-” กับชนิดข้อมูล unsigned
6
7 entity ADD_SUB_4BIT is
8     Port ( A,B : in unsigned (3 downto 0);
9             OPER : in STD_LOGIC;
10            RESULT : out unsigned (4 downto 0));
11
12 end ADD_SUB_4BIT;
13
14 architecture Behavioral of ADD_SUB_4BIT is
15 begin
16     RESULT <= ('0'&A) + ('0'&B) when OPER='0' else
17                 ('0'&A) - ('0'&B);
18
19 end Behavioral;

```

2.26f) โค้ดวงจรบวกและลบอย่างง่ายที่เขียนด้วยคำสั่ง Conditional signal assignment

รูปที่ 2.26 โค้ดวงจรบวกและลบอย่างง่าย

ตัวอย่างที่ 2.10 ฝึกใช้คำสั่ง Selected signal assignment และ Conditional signal assignment ในการออกแบบวงจรกดรหัสเข้า 2 ออก 4 (2 to 4 Decoder) โดยอินพุตเป็นบัส 2 บิตและเอาต์พุตเป็นบัส 4 บิตดังรูปที่ 2.27

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DECODER2TO4 is
6     port ( A : in STD_LOGIC_VECTOR(1 downto 0);
7            Y : out STD_LOGIC_VECTOR(3 downto 0));
8 end DECODER2TO4;
9
10 architecture BEHAVIORAL of DECODER2TO4 is
11 begin
12 with A select
13     Y <= "0001" when "00", -- Y(3)='0',Y(2)='0',Y(1)='0',Y(0)='1'
14     "0010" when "01", -- Y(3)='0',Y(2)='0',Y(1)='1',Y(0)='0'
15     "0100" when "10", -- Y(3)='0',Y(2)='1',Y(1)='0',Y(0)='0'
16     "1000" when others;-- Y(3)='1',Y(2)='0',Y(1)='0',Y(0)='0'
17 end BEHAVIORAL;

```

2.27a) โค้ดวงจร 2 to 4 Decoder ที่เขียนด้วยคำสั่ง Selected signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DECODER2TO4 is
6     port ( A : in STD_LOGIC_VECTOR(1 downto 0);
7            Y : out STD_LOGIC_VECTOR(3 downto 0));
8 end DECODER2TO4;
9
10 architecture BEHAVIORAL of DECODER2TO4 is
11 begin
12     Y <= "0001" when A="00" else -- Y(3)='0',Y(2)='0',Y(1)='0',Y(0)='1'
13     "0010" when A="01" else -- Y(3)='0',Y(2)='0',Y(1)='1',Y(0)='0'
14     "0100" when A="10" else -- Y(3)='0',Y(2)='1',Y(1)='0',Y(0)='0'
15     "1000" ;                -- Y(3)='1',Y(2)='0',Y(1)='0',Y(0)='0'
16 end BEHAVIORAL;

```

2.27b) โค้ดวงจร 2 to 4 Decoder ที่เขียนด้วยคำสั่ง Conditional signal assignment

รูปที่ 2.27 โค้ด VHDL ของวงจร 2 to 4 Decoder

จากตัวอย่างในรูปที่ 2.27 นั้นบัส A มีขนาด 2 บิตประกอบด้วย A(1) และ A(0) ในขณะที่บัส Y มีขนาด 4 บิตประกอบด้วย Y(3), Y(2), Y(1) และ Y(0) ดังนั้นการเขียนโค้ดแบบนี้จึงมองໄດ້ทั้งที่เป็นในลักษณะของบัสและในลักษณะของแต่ละบิตໄດ້เช่นกัน ขอให้ผู้อ่านทำความเข้าใจการเขียนโค้ดในรูปแบบที่เป็นบัสและเป็นแต่ละบิตให้ดี ซึ่งจะเห็นໄດ້อย่างชัดเจนว่าการเขียนโค้ดในรูปแบบที่เป็นบัสจะเขียนได้สะดวกกว่าและง่ายกว่ามาก

ตัวอย่างที่ 2.11 ฝึกใช้คำสั่ง Selected signal assignment และ Conditional signal assignment ในการออกแบบวงจรลดครั้งที่สอง แสดงผลเริ่มแรกเมื่อคุณนับไป โดยใช้รูปแบบเลขฐานสิบหกแสดงดังรูปที่ 2.15a) และมีตารางความจริงแสดงดังรูปที่ 2.15b) ในตัวอย่างที่ 2.3 โค้ดของวงจรที่ได้แสดงดังรูปที่ 2.28a) และ 2.28b)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity HEX_TO_7SEGMENT is
6     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
7             Y : out STD_LOGIC_VECTOR (6 downto 0));
8 end HEX_TO_7SEGMENT;
9
10 architecture Behavioral of HEX_TO_7SEGMENT is
11 begin
12     with A select
13     -- gfedcba
14         Y <= "1110001" when "1111", --F
15         "1111001" when "1110", --E      Y(0)=a
16         "1011110" when "1101", --d      ---
17         "0111001" when "1100", --C      Y(5)=f|  Y(1)=b
18         "1111100" when "1011", --b      --- Y(6)=g
19         "1101111" when "1010", --A      Y(4)=e|  Y(2)=c
20         "1101111" when "1001", --9      ---
21         "1111111" when "1000", --8      Y(3)=d
22         "0000111" when "0111", --7
23         "1111101" when "0110", --6
24         "1101101" when "0101", --5
25         "1100110" when "0100", --4
26         "1001111" when "0011", --3
27         "1011011" when "0010", --2
28         "0000110" when "0001", --1
29         "0111111" when others; --0
30 end Behavioral;

```

2.28a) โค้ดควบจัดการหัสตัวแสดงผลเลขฐานเซกเมนต์ที่เขียนด้วยคำสั่ง Selected signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity HEX_TO_7SEGMENT is
6     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
7             Y : out STD_LOGIC_VECTOR (6 downto 0));
8 end HEX_TO_7SEGMENT;
9
10 architecture Behavioral of HEX_TO_7SEGMENT is
11 begin --gfedcba
12     Y <= "1110001" when A="1111" else --F
13     "1111001" when A="1110" else --E      Y(0)=a
14     "1011110" when A="1101" else --d      ---
15     "0111001" when A="1100" else --C      Y(5)=f|  Y(1)=b
16     "1111100" when A="1011" else --b      --- Y(6)=g
17     "1101111" when A="1010" else --A      Y(4)=e|  Y(2)=c
18     "1101111" when A="1001" else --9      ---
19     "1111111" when A="1000" else --8      Y(3)=d
20     "0000111" when A="0111" else --7
21     "1111101" when A="0110" else --6
22     "1101101" when A="0101" else --5
23     "1100110" when A="0100" else --4
24     "1001111" when A="0011" else --3
25     "1011011" when A="0010" else --2
26     "0000110" when A="0001" else --1
27     "0111111";                      --0
28 end Behavioral;

```

2.28b) โค้ดควบจัดการหัสตัวแสดงผลเลขฐานเซกเมนต์ที่เขียนด้วยคำสั่ง Conditional signal assignment

รูปที่ 2.28 โค้ด VHDL ของวงจรจัดการหัสตัวแสดงผลเลขฐานเซกเมนต์

ตัวอย่างที่ 2.12 ฝึกใช้คำสั่ง Selected signal assignment และ Conditional signal assignment ในการออกแบบวงจรจดจำรหัสตัวแสดงผลเซกเมนต์แบบคอมมอนแค็ปต์โดยเมื่อตัวเลขที่ 2.11 แต่จะจดจำรหัสได้เฉพาะเลข 0-9 เท่านั้น และเมื่ออินพุตวงจรจดจำรหัสเป็นเลข 10-15 (1010,1011,1100,1101,1110 และ 1111) แล้วเอาต์พุตของตัวจดจำรหัสนี้จะแสดงเลข 0 ໂກ็คของวงจรที่ได้แสดงดังรูปที่ 2.29a) และ 2.29b)

```

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   with A select
13   --      gfedcba
14     Y <= "1101111" when "1001", --9
15           "1111111" when "1000", --8      Y(0)=a
16           "0000111" when "0111", --7      ---
17           "1111101" when "0110", --6 Y(5)=f|  Y(1)=b
18           "1101101" when "0101", --5      --- Y(6)=g
19           "1100110" when "0100", --4 Y(4)=e|  Y(2)=c
20           "1001111" when "0011", --3      ---
21           "1011011" when "0010", --2      Y(3)=d
22           "0000110" when "0001", --1
23           "0111111" when others; --0,a,b,c,d,E,F
24 end Behavioral;
```

2.29a) ໂກ็ควงจรจดจำรหัสตัวแสดงผลเซกเมนต์ที่เขียนด้วยคำสั่ง Selected signal assignment

```

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;
```

2.29b) ໂກ็ควงจรจดจำรหัสตัวแสดงผลเซกเมนต์ที่เขียนด้วยคำสั่ง Conditional signal assignment

รูปที่ 2.29 ໂກ็ค VHDL ของวงจรจดจำรหัสตัวแสดงผลเซกเมนต์ได้เฉพาะเลข 0-9

ตัวอย่างที่ 2.13 ฝึกใช้คำสั่ง Conditional signal assignment ในการออกแบบวงจรเข้ารหัสที่มีลำดับความสำคัญ (Priority encoder) ที่ใช้แปลงสัญญาณปุ่มกดแบบ Active low (กล่าวคือ เมื่อกดปุ่มจะให้ดอจิก ‘0’) เป็นรหัส BCD (หรือเลขฐานสิบองหนาเลข 0-9) และเมื่อกดปุ่มพร้อมกันหลายปุ่มวงจรนี้จะเลือกเข้ารหัสโดยให้อาร์พุตเป็นของปุ่มกดที่มีค่าตัวเลขสูงสุดเท่านั้น และถ้าไม่มีการกดปุ่มเดียวก็ให้อาร์พุตเป็น “1111” ซึ่งมีตารางความจริงดังรูปที่ 2.30a) โดยที่ “-” คือ Don't care และโภคของวงจรเข้ารหัสที่ได้แสดงดังรูปที่ 2.30b)

จากรูปที่ 2.30b) จะเห็นได้ว่าความสามารถออกแบบวงจรเข้ารหัสที่มีลำดับความสำคัญ (Priority encoder) โดยใช้คำสั่ง Conditional signal assignment ได้ง่าย ให้ผู้อ่านทดสอบออกแบบวงจรที่เขียนคำสั่ง Selected signal assignment ดูน้ำหนึ่งจากนั้นจึงคุ้นเคยที่เขียนโดยใช้คำสั่ง Selected signal assignment ในรูปที่ 2.30c) ซึ่งวิธีนี้จะต้องเขียนให้ครบถ้วนเงื่อนไขซึ่งบ่งบอกกว่าวิธีแรก และขอให้สังเกตว่าในกรณีของชนิดข้อมูล integer นั้นความสามารถเขียนโดยรวมทุกเงื่อนไข จึงอาจไม่จำเป็นต้องใช้ “when others” ในบรรทัดที่ 14 ถ้าเราแปลงชนิดข้อมูล integer โดยเรียกใช้ Numeric_std package จะต้องใช้คำสั่ง to_integer

No.	Input A											Output B			
	A(9)	A(8)	A(7)	A(6)	A(5)	A(4)	A(3)	A(2)	A(1)	A(0)	B(3)	B(2)	B(1)	B(0)	
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	0	-	0	0	0	1	1
2	1	1	1	1	1	1	1	0	-	-	0	0	1	0	0
3	1	1	1	1	1	1	0	-	-	-	0	0	1	1	1
4	1	1	1	1	1	0	-	-	-	-	0	1	0	0	0
5	1	1	1	1	0	-	-	-	-	-	0	1	0	1	1
6	1	1	1	0	-	-	-	-	-	-	0	1	1	0	0
7	1	1	0	-	-	-	-	-	-	-	0	1	1	1	1
8	1	0	-	-	-	-	-	-	-	-	1	0	0	0	0
9	0	-	-	-	-	-	-	-	-	-	1	0	0	1	1

2.30a) ตารางความจริงของวงจรเข้ารหัสที่มีลำดับความสำคัญที่ใช้แปลงสัญญาณปุ่มกดแบบ Active low เป็นรหัส BCD

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity PRI_ENCODER_BCD is
6   Port ( A : in STD_LOGIC_VECTOR (9 downto 0);
7         B : out STD_LOGIC_VECTOR (3 downto 0));
8 end PRI_ENCODER_BCD;
9
10 architecture Behavioral of PRI_ENCODER_BCD is
11 begin
12   B <= "1001" when A(9)='0' else --Key pad No.9
13   "1000" when A(8)='0' else --Key pad No.8
14   "0111" when A(7)='0' else --Key pad No.7
15   "0110" when A(6)='0' else --Key pad No.6
16   "0101" when A(5)='0' else --Key pad No.5
17   "0100" when A(4)='0' else --Key pad No.4
18   "0011" when A(3)='0' else --Key pad No.3
19   "0010" when A(2)='0' else --Key pad No.2
20   "0001" when A(1)='0' else --Key pad No.1
21   "0000" when A(0)='0' else --Key pad No.0
22   "1111";
23 end Behavioral;

```

2.30b) โภคของวงจรเข้ารหัสที่มีลำดับความสำคัญที่เขียนโดยใช้คำสั่ง Conditional signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;          티องเรียกใช้ std_logic_unsigned package
5                                         เพราะมีการใช้คำสั่ง conv_integer
6 entity PRI_ENCODER_BCD is
7     Port ( A : in STD_LOGIC_VECTOR (9 downto 0);
8            B : out STD_LOGIC_VECTOR (3 downto 0));
9 end PRI_ENCODER_BCD;
10
11 architecture Behavioral of PRI_ENCODER_BCD is
12     signal X : integer range 0 to 1023;
13 begin
14     X <= conv_integer(A);
15     with X select
16         B <= "1001" when 0 to 511,      --Key pad No.9
17             "1000" when 512 to 767,      --Key pad No.8
18             "0111" when 768 to 895,      --Key pad No.7
19             "0110" when 896 to 959,      --Key pad No.6
20             "0101" when 960 to 991,      --Key pad No.5
21             "0100" when 992 to 1007,      --Key pad No.4
22             "0011" when 1008 to 1015,      --Key pad No.3
23             "0010" when 1016 to 1019,      --Key pad No.2
24             "0001" when 1020 to 1021,      --Key pad No.1
25             "0000" when 1022,              --Key pad No.0
26             "1111" when 1023;           --Key pad No.0
27 end Behavioral;

```

2.30c) โค้ดของวงจรเข้ารหัสที่มีลำดับความสำคัญที่เขียนโดยใช้คำสั่ง Selected signal assignment

รูปที่ 2.30 ตารางความจริงและ โค้ดของวงจรเข้ารหัสที่มีลำดับความสำคัญที่ใช้แปลงสัญญาณปุ่มกดเป็นรหัส BCD

2.13.4 Component instantiation statement

Component instantiation statement เป็นคำสั่งแบบคอมโคร์เรนต์ ซึ่งจะใช้ในการออกแบบวงจรที่มีความซับซ้อน เช่น ไมโครโปรเซสเซอร์ เป็นต้น หรือใช้เขียน Netlist เพื่อแยกแจงว่า วงจรที่ออกแบบนั้นมีอุปกรณ์พื้นฐานต่อ กันอย่างไร ข้าง คำสั่งนี้ ใช้เขียนโดยแบบลำดับชั้น (Hierarchy model) หรือแบบโครงสร้าง (Structural model หรือ Structural descriptions) ซึ่งเป็นการ ออกแบบวงจรหลัก ไว้ในระดับบนสุด (Top level) จากวงจรย่อยหรือคอมโพเนนต์ (Component) ต่างๆ ที่เคยออกแบบไว้แล้ว หรือ มีอยู่แล้วมาใช้ในการออกแบบ คอมโพเนนต์จึงเป็นโค้ดของวงจรย่อยนั้นเอง ซึ่งก่อนใช้งานจะต้องประกาศใช้คอมโพเนนต์ (Component declaration) จากนั้นจึงใช้คำสั่งใช้คอมโพเนนต์ (Component instantiation statement) เพื่อนำคอมโพเนนต์ไปใช้

1) การประกาศใช้คอมโพเนนต์

ก่อนใช้งานต้องประกาศใช้คอมโพเนนต์ (Component declaration) ใน Architecture หรือ Package โดยมีรูปแบบดังนี้

```

component COMPONENT_NAME
    [ generic (GENERIC_DECLARATIONS) ]
    [ port (PORT_DECLARATIONS) ]
end component ;

```

โดยที่ GENERIC_DECLARATIONS และ PORT_DECLARATIONS จะมีรูปแบบเดียวกันที่ใช้ในประกาศใช้ Entity

2) คำสั่งใช้คอมโพเนนต์

คำสั่งใช้คอมโพเนนต์ (Component instantiation statement) เป็นคำสั่งคอนเคนเต้นต์ โดยมีรูปแบบดังนี้

```
INSTANCE_NAME : COMPONENT_NAME
[ generic map ( GENERIC_NAME => VALUE { , GENERIC_NAME => VALUE } ) ]
port map ( [PORT_NAME =>] SIGNAL_NAME { , [PORT_NAME =>] SIGNAL_NAME } );
```

โดยที่ [PORT_NAME =>] จะเป็นชื่อ อินพุต/เอาต์พุตของคอมโพเนนต์ การเขียนคำสั่งใช้คอมโพเนนต์ที่ไม่มี [PORT_NAME =>] จะเรียกว่าแบบ Positional association ซึ่งต้องวางตำแหน่ง SIGNAL_NAME เรียงตามลำดับ แต่ถ้ามี [PORT_NAME =>] ก็จะเรียกว่าแบบ Named association ซึ่งจะไม่สนใจการเรียงตามลำดับของ SIGNAL_NAME

ตัวอย่างที่ 2.14 โค้ดของวงจรบวก 3 บิตที่แสดงผลทางเลขฐานเทอม 1 หลักแสดงดังรูปที่ 2.31a) ให้ทำการเขียนโค้ดใหม่โดยใช้คอมโพเนนต์จากโค้ดของวงจรบวก 2 บิตในรูปที่ 2.31b) และโค้ดของวงจรรถอหัสตัวแสดงผลทางเลขฐานเทอมที่ในรูปที่ 2.31c) เพื่อใช้แทนโค้ดของวงจรบวก 3 บิตในรูปที่ 2.31a) โดยรายละเอียดของโค้ดใหม่ที่ได้แสดงดังรูปที่ 2.31d) ซึ่งชื่อ IC1 และ IC2 ในบรรทัดที่ 23 และบรรทัดที่ 27 ในรูปที่ 2.31d) จะเป็น INSTANCE_NAME นั้นเอง

```
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.all;
4 use IEEE.STD_LOGIC_UNSIGNED.all; ต้องเรียกใช้ std_logic_unsigned package เพราะใช้ Operator “+” กับชนิดข้อมูล std_logic_vector
5
6 entity ADDER3BIT_1DIGIT is
7   Port ( A,B : in STD_LOGIC_VECTOR (2 downto 0);
8         Y : out STD_LOGIC_VECTOR (6 downto 0));
9 end ADDER3BIT_1DIGIT;
10
11 architecture Behavioral of ADDER3BIT_1DIGIT is
12   signal C : STD_LOGIC_VECTOR (3 downto 0); การประกาศใช้ Signal จะประกาศใช้ เมื่อชื่อของ Signal นั้นไม่มีชื่อปรากฏ ใน port ของ entity ซึ่ง Signal C ใช้ต่อจากเอาต์พุต IC1 ไปที่อินพุต IC2
13 begin
14   -----IC1=ADDER3BIT-----
15
16   C <= ('0' & A) + ('0' & B); ใช้ Operator “+” กับชนิดข้อมูล std_logic_vector
17
18   -----IC2=7SEGMENT DECODER-----
19   --      gfedcba
20   Y <= "1110001" when C="1111" else --F
21     "1111001" when C="1110" else --E      Y(0)=a
22     "1011110" when C="1101" else --d      ---
23     "0111001" when C="1100" else --C Y(5)=f|  Y(1)=b
24     "11111100" when C="1011" else --b      --- Y(6)=g
25     "1110111" when C="1010" else --A Y(4)=e|  Y(2)=c
26     "1101111" when C="1001" else --9      ---
27     "1111111" when C="1000" else --8      Y(3)=d
28     "0000111" when C="0111" else --7
29     "1111101" when C="0110" else --6
30     "1101101" when C="0101" else --5
31     "1100110" when C="0100" else --4
32     "1001111" when C="0011" else --3
33     "1011011" when C="0010" else --2
34     "0000110" when C="0001" else --1
35     "0111111";                         --0
36 end Behavioral;
```

2.31a) โค้ด VHDL ของวงจรบวกขนาด 3 บิตที่แสดงผลทางเลขฐานเทอม 1 หลัก

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL; ต้องเรียกใช้ std_logic_unsigned package เพราะใช้ Operator “+” กับชนิดข้อมูล std_logic_vector
5
6 entity ADDER_2BIT is
7     generic ( N : integer := 2);
8     port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
9            C : out STD_LOGIC_VECTOR (N downto 0));
10 end ADDER_2BIT;
11
12 architecture BEHAVIORAL of ADDER_2BIT is
13 begin
14
15     C <= ('0' & A) + ('0' & B); ใช้ Operator “+” กับชนิดข้อมูล std_logic_vector
16
17 end BEHAVIORAL;

```

2.31b) โค้ด VHDL ของวงจรบวกขนาด 2 บิต

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity HEX_TO_7SEGMENT is
6     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
7            Y : out STD_LOGIC_VECTOR (6 downto 0));
8 end HEX_TO_7SEGMENT;
9
10 architecture Behavioral of HEX_TO_7SEGMENT is
11 begin --gfedcba
12     Y <= "1110001" when A="1111" else --F
13         "1111001" when A="1110" else --E      Y(0)=a
14         "1011110" when A="1101" else --d      ---
15         "0111001" when A="1100" else --C Y(5)=f|  Y(1)=b
16         "1111100" when A="1011" else --b      --- Y(6)=g
17         "1110111" when A="1010" else --A Y(4)=e|  Y(2)=c
18         "1101111" when A="1001" else --9      ---
19         "1111111" when A="1000" else --8      Y(3)=d
20         "0000111" when A="0111" else --7
21         "1111101" when A="0110" else --6
22         "1101101" when A="0101" else --5
23         "1100110" when A="0100" else --4
24         "1001111" when A="0011" else --3
25         "1011011" when A="0010" else --2
26         "0000110" when A="0001" else --1
27         "0111111";                      --0
28 end Behavioral;

```

2.31c) โค้ดของวงจรจดจำรหัสตัวแสดงผลเซเว่นเซกเมนต์ที่เปลี่ยนด้วยคำสั่ง Conditional signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ADDER3BIT_1DIGIT is
6     Port ( A,B : in STD_LOGIC_VECTOR (2 downto 0);
7             Y : out STD_LOGIC_VECTOR (6 downto 0));
8 end ADDER3BIT_1DIGIT;
9
10 architecture Behavioral of ADDER3BIT_1DIGIT is
11     component ADDER_2BIT
12         generic ( N : integer := 2);
13         port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
14                 C : out STD_LOGIC_VECTOR (N downto 0));
15     end component;
16     component HEX_TO_7SEGMENT
17         Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
18                 Y : out STD_LOGIC_VECTOR (6 downto 0));
19     end component;
20     signal C : STD_LOGIC_VECTOR (3 downto 0);
21 begin
22     -----
23     IC1 : ADDER_2BIT generic map (N => 3)      --ADDER_2BIT=>ADDER_3BIT
24           port map (A,B,C);                  --Positional association
25     INSTANCE_NAME
26     -----
27     IC2 : HEX_TO_7SEGMENT port map (A => C, Y => Y);--Named association
28
29 end Behavioral;

```

บรรทัด 11-19 เป็นการประกาศใช้คอมโพเนนต์

Signal C ใช้ต่อจากเอาต์พุต IC1 ไปที่อินพุต IC2

บรรทัดที่ 23-24 และ 27 เป็นคำสั่งใช้คอมโพเนนต์

2.31d) โค้ด VHDL ของวงจรบวกขนาด 3 บิตที่แสดงผลทางเข温นเซกเมนต์จำนวน 1 หลักที่ใช้ Component

รูปที่ 2.31 โค้ด VHDL ของวงจรบวกขนาด 3 บิตที่แสดงผลทางเข温นเซกเมนต์จำนวน 1 หลัก

ขอให้ผู้อ่านสังเกตว่าโค้ด VHDL ของวงจรบวก 3 บิตที่แสดงผลทางเข温นเซกเมนต์จำนวน 1 หลักในรูปที่ 2.31a) นั้นประกอบด้วยคำสั่งคอนเคอร์เรนต์ 2 คำสั่งหรือ 2 วงจรย่อย กือ โค้ดของวงจรบวก 3 บิต (บรรทัดที่ 16) และโค้ดของวงจรบวกหัสตัวแสดงผลทางเข温นเซกเมนต์ (บรรทัดที่ 20-35) โดยใช้ Signal C ที่เป็นสมือนสายไฟเชื่อมต่อจากเอาต์พุตของวงจรบวก 3 บิตไปยังอินพุตของวงจรบวกหัสตัวแสดงผลทางเข温นเซกเมนต์ (Signal C ใช้ในการแยกเปลี่ยนของมูลระหว่างคำสั่งคอนเคอร์เรนต์ทั้ง 2 คำสั่งนี้) ซึ่งวงจรบวกขนาด 3 บิตที่แสดงผลทางเข温นเซกเมนต์จำนวน 1 หลักในรูปที่ 2.31a) จะให้ผลลัพธ์เหมือนกับวงจรในรูปที่ 2.31d) ทุกประการ เราจะใช้หลักการออกแบบทั้ง 2 วิธีนี้ในการออกแบบวงจรขนาดใหญ่ หรือเราอาจออกแบบวงจรขนาดใหญ่โดยใช้หลักการทั้ง 2 วิธีนี้ปันกันก็ได้

จากรูปที่ 2.31d) ในบรรทัดที่ 24 สามารถเขียนแบบ Named association ได้เป็น port map (A => A, B => B, C => C) และในบรรทัดที่ 27 สามารถเขียนแบบ Positional association ได้เป็น port map (C, Y)

เพื่อให้เข้าใจง่ายขึ้นเราสามารถเขียนโค้ด VHDL ในรูปที่ 2.31a) ใหม่โดยเพิ่ม Label ชื่อ IC1 และ IC2 เข้าไปด้วยแสดงดังรูปที่ 2.32 ซึ่งสามารถนำไปใช้ยืนยันกับโค้ดในรูปที่ 2.31d) ได้โดยตรง ซึ่งมองสมือนว่าวงจรนี้ประกอบด้วยวงจรย่อยที่เป็นไอชี 2 ตัว กือ IC1 และ IC2 โดยที่การประกาศใช้คอมโพเนนต์ (Component declaration) เป็นสมือนการสร้างช่องเก็ต (Socket) ของตัวไอชีที่อยู่บนแผงวงจร (PCB) ส่วนคำสั่งใช้คอมโพเนนต์ (Component instantiation statement) เป็นสมือนการนำเอาไอชีที่เป็นเบอร์ตามชื่อของ Component ไปเสียบลงบนช่องเก็ต

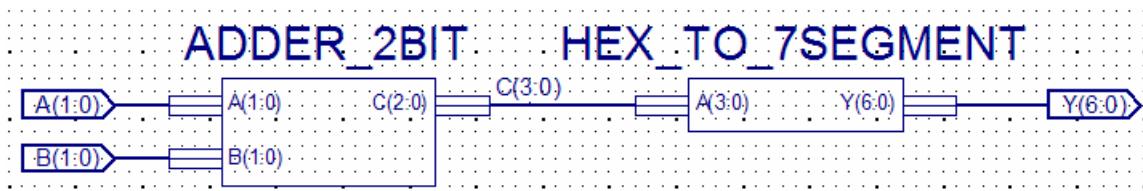
ในกรณีที่เรานำ Component ไปสร้างเป็น Symbols เพื่อใช้ในการออกแบบด้วยวิธีคาดผังวงจรนั้นเราจะเห็นได้อย่างชัดเจนว่าชื่อของ Component กือชื่อของ Symbols นั้นเอง ซึ่งแสดงดังรูปที่ 2.33

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity ADDER3BIT_1DIGIT is
7      Port ( A,B : in STD_LOGIC_VECTOR (2 downto 0);
8             Y : out STD_LOGIC_VECTOR (6 downto 0));
9  end ADDER3BIT_1DIGIT;
10
11 architecture Behavioral of ADDER3BIT_1DIGIT is
12     signal C : STD_LOGIC_VECTOR (3 downto 0);
13 begin
14     -----IC1=ADDER3BIT-----
15
16 IC1 : C <= ('0' & A) + ('0' & B);
17
18     -----IC2=7SEGMENT DECODER-----
19     gfedcba
20 IC2 : Y <= "1110001" when C="1111" else --F
21         "1111001" when C="1110" else --E      Y(0)=a
22         "1011110" when C="1101" else --d      ---
23         "0111001" when C="1100" else --C Y(5)=f | Y(1)=b
24         "1111100" when C="1011" else --b      --- Y(6)=g
25         "1110111" when C="1010" else --A Y(4)=e | Y(2)=c
26         "1101111" when C="1001" else --9      ---
27         "1111111" when C="1000" else --8      Y(3)=d
28         "0000111" when C="0111" else --7
29         "1111101" when C="0110" else --6
30         "1101101" when C="0101" else --5
31         "1100110" when C="0100" else --4
32         "1001111" when C="0011" else --3
33         "1011011" when C="0010" else --2
34         "0000110" when C="0001" else --1
35         "0111111";                      --0
36 end Behavioral;

```

รูปที่ 2.32 โค้ด VHDL ของวงจรบวกขนาด 3 บิตที่แสดงผลทางเข็มแทนต์จำนวน 1 หลัก



รูปที่ 2.33 วงจรบวก 3 บิต ($N = 3$) ที่แสดงผลทางเข็มแทนต์จำนวน 1 หลักที่ออกแบบด้วยวิธีการผังวงจร

2.13.5 Generate statement

คำสั่ง Generate มี 2 แบบ คือ for/generate และ if/generate โดยที่คำสั่ง for/generate เหมาะกับการทำคำสั่งซ้ำๆ หรือสร้างวงจรย่อยๆ กันหลายชุด ส่วนคำสั่ง if/generate นั้นจะทำการทำตามคำสั่งก็ต่อเมื่อเงื่อนไข (CONDITION) เป็นจริง ซึ่งรูปแบบการเขียนคำสั่งเป็นดังนี้

```
generate_label : for INDEX in DISCRETE_RANGE generate
    {CONCURRENT_STATEMENT}
end generate [ generate_label ] ;
```

และ

```
generate_label : if CONDITION generate
    {CONCURRENT_STATEMENT}
end generate [ generate_label ] ;
```

ตัวอย่างที่ 2.15 ฝึกใช้คำสั่ง for/generate โดยเขียนโค้ดวงจรบวก 4 บิตจาก Component ของวงจรบวกแบบ Full adder 1 บิตที่มีอินพุตคือ A, B, ตัวทดเข้า (Carry-in) หรือ Cin, และเอาต์พุตผลรวมหรือ Sum และ ตัวทดออก (Carry-out) หรือ Cout ดังตารางความจริงในรูปที่ 2.34a) และ โค้ดของวงจรบวกแบบ Full adder ขนาด 1 บิตแสดงดังรูปที่ 2.34b) โค้ดของวงจรบวก 4 บิตที่ได้แสดงดังรูปที่ 2.34c) หรือรูปที่ 2.34d) โค้ดที่ได้ในรูปที่ 2.34d) จะเป็นการใช้ Attribute ช่วยในการเขียนโค้ดให้อยู่ในรูปฟอร์มทั่วไป

Input			Output	
A	B	Cin	Sum	Cout
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

2.34a) ตารางความจริงของวงจร Full adder ขนาด 1 บิต

```
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity FULL_ADDER_1BIT is
6     Port ( A,B,Cin : in STD_LOGIC;
7             Cout,Sum : out STD_LOGIC);
8 end FULL_ADDER_1BIT;
9
10 architecture Behavioral of FULL_ADDER_1BIT is
11     signal X : STD_LOGIC_VECTOR(2 downto 0);
12 begin
13     X <= A&B&Cin;
14     Sum <= '1' when (X="001" or X="010" or X="100" or X="111") else
15             '0';
16     Cout <= '1' when (X="011" or X="101" or X="110" or X="111") else
17             '0';
18 end Behavioral;
```

2.34b) โค้ดของวงจร Full adder ขนาด 1 บิตที่จะนำไปทำเป็น Component

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ADDN_FOR_GEN is
6     Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
7             Cin : in STD_LOGIC;
8             Cout : out STD_LOGIC;
9             Sum : out STD_LOGIC_VECTOR (3 downto 0));
10 end ADDN_FOR_GEN;
11
12 architecture Behavioral of ADDN_FOR_GEN is
13     component FULL_ADDER_1BIT
14         Port ( A,B,Cin : in STD_LOGIC;
15                 Cout,Sum : out STD_LOGIC);
16     end component;
17     signal C_tmp : STD_LOGIC_VECTOR (4 downto 0);
18 begin
19     C_tmp(0) <= Cin; generate_label ชี้งการตั้งชื่อให้สื่อความหมายด้วย
20
21 ADDN_GEN : for I in 3 downto 0 generate
22     ADDN_BIT : FULL_ADDER_1BIT
23         port map(A(I),B(I),C_tmp(I),C_tmp(I+1),Sum(I));
24     end generate;
25
26
27     Cout <= C_tmp(4);
28
29 end Behavioral;

```

2.34c) โค้ดของวงจรบวกขนาด 4 บิตที่สร้างจาก Component วงจร Full adder ขนาด 1 บิตจำนวน 4 ชุด

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ADDN_FOR_GEN is
6     generic (N : integer := 4);
7     Port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
8             Cin : in STD_LOGIC;
9             Cout : out STD_LOGIC;
10            Sum : out STD_LOGIC_VECTOR (N-1 downto 0));
11 end ADDN_FOR_GEN;
12
13 architecture Behavioral of ADDN_FOR_GEN is
14     component FULL_ADDER_1BIT
15         Port ( A,B,Cin : in STD_LOGIC;
16                 Cout,Sum : out STD_LOGIC);
17     end component;
18     signal C_tmp : STD_LOGIC_VECTOR (N downto 0);
19 begin
20     C_tmp(0) <= Cin;
21
22 ADDN_GEN : for I in A'range generate --A'range = N-1 downto 0
23     ADDN_BIT : FULL_ADDER_1BIT
24         port map(A(I),B(I),C_tmp(I),C_tmp(I+1),Sum(I));
25     end generate;
26
27     Cout <= C_tmp(C_tmp'high);      --C_tmp'high = N
28 end Behavioral;

```

2.34d) โค้ดของวงจรบวกขนาด 4 บิตที่สร้างจาก Component วงจร Full adder ขนาด 1 บิต

รูปที่ 2.34 โค้ดของวงจรบวกขนาด 4 บิต

ขอให้ผู้อ่านสังเกตว่าโค้ดในรูปที่ 2.34b) นั้นอาจเขียนใหม่ด้วยคำสั่ง Selected signal assignment ได้ดังรูปที่ 2.35

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity FULL_ADDER_1BIT is
6     Port ( A,B,Cin : in STD_LOGIC;
7             Cout,Sum : out STD_LOGIC);
8 end FULL_ADDER_1BIT;
9
10 architecture Behavioral of FULL_ADDER_1BIT is
11     signal X : STD_LOGIC_VECTOR(2 downto 0);
12 begin
13     X <= A&B&Cin;
14     with X select
15         Sum <= '1' when "001" | "010" | "100" | "111",
16         '0' when others;
17     with X select
18         Cout <= '1' when "011" | "101" | "110" | "111",
19         '0' when others;
20 end Behavioral;
```

รูปที่ 2.35 โค้ดของวงจร Full adder ขนาด 1 บิตที่เขียนโดยคงรั้นคำสั่ง Selected signal assignment

ตัวอย่างที่ 2.16 ฝึกใช้คำสั่ง if/generate โดยนำเอาโค้ดในตัวอย่างที่ 2.15 มาเขียนโดยใหม่โดยมีการระบุขอบเขตว่าจะใช้โค้ดที่เขียนในรูปฟอร์มทั่วไปได้เฉพาะในช่วง $N \geq 2$ ถึง $N \leq 32$ เท่านั้น โดยที่โค้ดของวงจรบวก 4 บิตที่ได้แสดงดังรูปที่ 2.36

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ADDN_IF_GEN is
6     generic (N : integer := 4);
7     Port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
8             Cin : in STD_LOGIC;
9             Cout : out STD_LOGIC;
10            Sum : out STD_LOGIC_VECTOR (N-1 downto 0));
11 end ADDN_IF_GEN;
12
13 architecture Behavioral of ADDN_IF_GEN is
14     component FULL_ADDER_1BIT
15         Port ( A,B,Cin : in STD_LOGIC;
16                 Cout,Sum : out STD_LOGIC);
17     end component;
18     signal C_tmp : STD_LOGIC_VECTOR (N downto 0);
19 begin
20     C_tmp(0) <= Cin;
21     --IF/generate loop-----
22 ADDN_2TO32 : if (N>=2 and N<=32) generate
23     --For/generate loop-----
24         ADDN_GEN : for I in A'range generate --A'range = N-1 downto 0
25             ADDN_BIT : FULL_ADDER_1BIT
26                 port map(A(I),B(I),C_tmp(I),C_tmp(I+1),Sum(I));
27             end generate;
28         --END for/generate loop-----
29     end generate;
30     --END if/generate loop-----
31     Cout <= C_tmp(C_tmp'high); --C_tmp'high = N
32 end Behavioral;
```

รูปที่ 2.36 โค้ดของวงจรบวกขนาด 4 บิตที่เขียนในรูปฟอร์มทั่วไปโดยใช้ได้เฉพาะในช่วง $N \geq 2$ ถึง $N \leq 32$ เท่านั้น

2.13.6 Block statement

คำสั่ง Block (Block statement) ใช้สำหรับการแบ่ง (Partitioning) ໂຄດของกลุ่มคำสั่งคอนแคร์เรนต์ขนาดใหญ่ออกเป็นส่วนๆ เพื่อสะดวกในการบริหารจัดการและตรวจสอบ ภายในໂຄດอาจมีคำสั่ง Block ได้หลายคำสั่งและอาจมีคำสั่ง Block อาจซ้อนกัน (Nested blocks) ได้อีกด้วย ในกรณีที่มีคำสั่ง Block ซ้อนกันอยู่ภายในและหากมีออบเจ็คท์ซึ่งซ่อนอยู่ในคำสั่ง Block ที่อยู่ภายในจะใช้ออบเจ็คท์ภายในคำสั่ง Block นี้ก่อน โดยจะยกเลิกออบเจ็คท์ที่อยู่ภายนอกคำสั่ง Block คำสั่ง Block แบ่งออกเป็น Simple block และ Guarded block โดยที่ XST จะรองรับเฉพาะ Simple block เท่านั้น Simple block มีรูปแบบการเขียนดังนี้

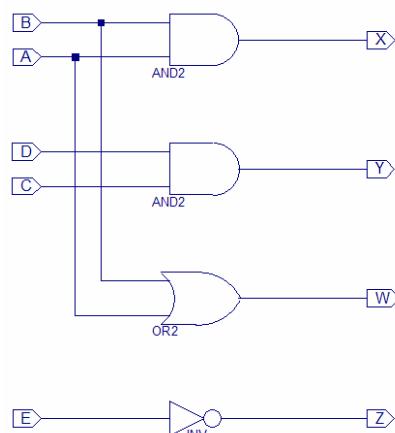
```
label : block
  {BLOCK_DECLARATIVE_PART}
  begin
    {CONCURRENT_STATEMENT}
  end block [ label ];
```

โดยที่ 1) BLOCK_DECLARATIVE_PART จะมีขอบเขตการใช้งานเฉพาะภายใน Block เท่านั้น ซึ่งได้แก่

- use clause
- Subprogram declaration และ Subprogram body
- Type declaration และ/หรือ Subtype declaration
- Constant declaration
- Signal declaration
- Component declaration

2) CONCURRENT_STATEMENT คือ คำสั่งคอนแคร์เรนต์

ตัวอย่างที่ 2.17 ฝึกการเขียนໂຄດ VHDL ของวงจรที่ประกอบด้วยเกตต่างๆ แสดงดังรูปที่ 2.37a) โดยใช้คำสั่ง Block (คำสั่ง Block ที่ซ้อนกันหลายคำสั่ง) ซึ่งวงจรนี้เป็นของໂຄດที่แสดงดังรูปที่ 2.37b)



2.37a) ผังวงจรที่ประกอบด้วยเกตต่างๆ

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity EX_BLOCK is
6     Port ( A,B,C,D,E : in STD_LOGIC;
7             W,X,Y,Z : out STD_LOGIC);
8 end EX_BLOCK;
9
10 architecture Behavioral of EX_BLOCK is
11 begin
12
13     Z <= not E;
14
15     B1: block
16         signal S: STD_LOGIC;      --Declaration of signal S in block B1
17     begin
18         S <= A and B;           --signal S from B1
19         B2: block
20             signal S: STD_LOGIC; --Declaration of signal S in block B2
21             begin
22                 S <= C and D;       --signal S from B2 (Declaration of S from B1 is overrided.)
23                 B3: block
24                     begin
25                         Y <= S;          --signal S from B2 (Declaration of S from B1 is overrided.)
26                     end block B3;
27                 end block B2;
28                 X <= S;           --signal S from B1
29             end block B1;
30
31         W <= A or B;
32
33     end Behavioral;

```

2.37b) การเขียน logic ด้วยคำสั่ง Block

รูปที่ 2.37 ว่าจะที่ประกอบด้วยเกตต่างๆ และการเขียน logic ด้วยคำสั่ง Block

2.13.7 Process statement

คำสั่งโพร์เชส (Process statement) เป็นคำสั่งคอนโทรลเรนต์แต่กายนั้นจะเป็นคำสั่งซึ่คนเขียน ซึ่งเราจะอธิบาย ละเอียดในข้อ 2.14 คำสั่ง Process มีรูปแบบการเขียนดังนี้

```

[ label : ] process [ (SENSITIVITY_LIST) ]
    {PROCESS_DECLARATIVE_PART}
    begin
        {SEQUENTIAL_STATEMENT}
    end process [ label ];

```

โดยที่ 1) PROCESS_DECLARATIVE_PART ได้แก่

- use clause
- Subprogram declaration และ Subprogram body
- Type declaration และ/หรือ Subtype declaration

- Constant declaration
- Variable declaration

2) SENSITIVITY LIST คือ อินพุตทุกตัวมีผลต่อการเปลี่ยนแปลงของ Process โดยตรง (ไม่ใช่โดยทางอ้อม)

3) SEQUENTIAL_STATEMENT คือ สั่งซีเควนเชียล

2.13.8 Concurrent procedure calls

Concurrent procedure calls เป็นการเรียกใช้โปรแกรมย่อยที่ใช้ภายใน Architecture หรือคำสั่ง Block ซึ่งรายละเอียดจะอธิบายในภายหลัง รูปแบบการเขียน Concurrent procedure calls เป็นดังนี้

```
PROCEDURE_NAME [ ( [ NAME => ] EXPRESSION { , [ NAME => ] EXPRESSION } ) ] ;
```

2.14 คำสั่งซีเควนเชียล

คำสั่งซีเควนเชียล (Sequential statement) จะมีลำดับความสำคัญก่อนและหลัง โดยจะต้องเขียนไว้ภายในคำสั่ง Process, Function หรือ Procedure เท่านั้น คำสั่ง Process แต่ละคำสั่งจะเป็นคำสั่งคอนโทรลレンต์ ยกเว้นคำสั่งที่อยู่ภายใน Process จะเป็นคำสั่งซีเควนเชียล คำสั่งซีเควนเชียลได้แก่ Signal assignment statement, Variable assignment statement, If statement, Case statement, Wait statement และ Loop statement รวมทั้งการเขียนโปรแกรมย่อย (Subprograms) การเขียนโดยแบบซีเควนเชียลนี้จะเรียกอีกอย่างว่าเป็นการเขียนโค้ดแบบบรรยายพฤติกรรม (Behavioral descriptions)

2.14.1 Signal assignment statement

คำสั่ง Signal assignment ใช้ในการกำหนดค่าให้กับสัญญาณ (Signal) หรือพอร์ต (Port) โดยมีรูปแบบการเขียนดังนี้

```
TARGET <= EXPRESSION ;
```

2.14.2 Variable assignment statement

คำสั่ง Variable assignment ใช้ในการกำหนดค่าให้กับ Variable ที่อยู่ในคำสั่ง Process, Function หรือ Procedure เท่านั้น โดยมีรูปแบบการเขียนดังนี้

```
TARGET := EXPRESSION ;
```

2.14.3 if statement

คำสั่ง if เป็นคำสั่งซึ่งความเชิงลด 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 ถัดไปจนกว่าจะพบ CONDITION ที่เป็นจริงแล้วจึงทำการ Sequential Statement นั้นโดยไม่สนใจ CONDITION ที่เหลือ ดังนั้นแต่ละ CONDITION จึงมีลำดับความสำคัญ (Priority) ถ้าไม่มี CONDITION ที่เป็นจริงแต่มีคำสั่ง else ก็จะทำการ Sequential Statement ที่อยู่หลัง else แต่ถ้าไม่มี CONDITION ที่เป็นจริงเลยและไม่มีคำสั่ง else อู้ดีวก็ไม่ต้องทำการ Sequential Statement ใดๆ ซึ่งก็หมายความว่าสถานะล็อกของเอาต์พุตก่อนหน้านั้นถูกคงค่า (Hold) ไว้หรือ Latch ค่าลอดิกนั้นไว้ นั่นก็แสดงว่าเมืองจร “Latch” เกิดขึ้น

2.14.4 case statement

คำสั่ง case เป็นคำสั่งซึ่งความเชิงลด 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 เหลือทั้งหมด และอาจใช้ “NULL” แทน Sequential Statement ในกรณีที่ไม่ทำอะไรมาก (No action) และเนื่องแต่ละ CHOICES ไม่มีการลำดับความสำคัญ (Priority) ดังนั้น CHOICES จะต้องไม่ซ้ำซ้อนกันและต้องกำหนดให้ครบถ้วน CHOICES

ตัวอย่างที่ 2.18 ฝึกใช้คำสั่ง if และ case ในการออกแบบวงจรแมติเพล็กซ์อร์ 4 อินพุต 1 เอาต์พุต ในตัวอย่างที่ 2.7 ซึ่งสามารถเขียนโดย VHDL ได้ดังรูปที่ 2.38a) ถึงรูปที่ 2.38c)

โค้ดในรูปที่ 2.38a) มีความหมายว่า เมื่อ SEL = “00” ($S1 = '0'$ และ $S0 = '0'$) เป็นจริงแล้วให้กำหนดค่าเอาต์พุต O จาก อินพุต A แต่ถ้า SEL = “00” เป็นเท็จก็จะไปทำ CONDITION ถัดไป ถ้า SEL = “01” ($S1 = '0'$ และ $S0 = '1'$) เป็นจริงแล้วให้

กำหนดค่าเอาต์พุต O จากอินพุต B แต่ถ้า SEL = “01” เป็นเท็จก็จะไปทำ CONDITION ต่อไป และถ้า SEL = “10” (S1 = ‘1’ และ S0 = ‘0’) เป็นจริงแล้วให้กำหนดค่าเอาต์พุต O จากอินพุต C แต่ถ้าไม่พบ CONDITION ที่เป็นจริง (รวมกรณีที่ SEL = “11” แล้ว) และมี else อัญคิรักษ์จะทำงาน SEQUENTIAL_STATEMENT ที่อยู่หลัง else คือ ให้กำหนดค่าเอาต์พุต O จากอินพุต D

รูปที่ 2.38b) เป็นการเขียนโค้ดโดยไม่ใช้ตัวดำเนินการรวมข้อมูลหรือ & (Concatenation operators) ของ S0 และ S1

โค้ดในรูปที่ 2.38c) 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’) ก็ให้กำหนดค่าเอาต์พุต O จากอินพุต C และถ้า SEL มีสถานะลอกิกเป็นค่าที่เหลือทั้งหมด (when others) โดยรวมสถานะที่ SEL = “11” เข้าไปด้วย ก็ให้กำหนดค่าเอาต์พุต O จากอินพุต D การใช้คำว่า when others จึง stemmed ในการเขียนเพื่อให้ได้ครอบคลุมทุกเงื่อนไขที่เป็นไปได้ทั้งหมด เพราะว่า S1 และ S0 เป็นชนิดข้อมูล std_logic และตัวจึงมีสถานะลอกิกได้ 9 สถานะ

```

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     process(A,B,C,D,SEL)
15         begin
16             if      SEL="00" then O <= A;
17             elsif SEL="01" then O <= B;
18             elsif SEL="10" then O <= C;
19             else               O <= D;    -- SEL="11"
20             end if;
21         end process;
22 end BEHAVIORAL;

```

2.38a) โค้ดวงจรแมติเพล็กเซอร์ที่เขียนด้วยคำสั่ง if

```

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 begin
12     process(A,B,C,D,S1,S0)
13         begin
14             if      (S1='0' and S0='0') then O <= A;
15             elsif (S1='0' and S0='1') then O <= B;
16             elsif (S1='1' and S0='0') then O <= C;
17             else               O <= D;    -- (S1='1' and S0='1')
18             end if;
19         end process;
20 end BEHAVIORAL;

```

2.38b) โค้ดวงจรแมติเพล็กเซอร์ที่เขียนด้วยคำสั่ง if แต่ไม่ใช้ตัวดำเนินการรวมข้อมูล

```

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     process(A,B,C,D,SEL)
15     begin
16         case SEL is
17             when "00" => O <= A;
18             when "01" => O <= B;
19             when "10" => O <= C;
20             when others => O <= D; -- SEL="11"
21         end case;
22     end process;
23 end BEHAVIORAL;

```

2.38c) โค้ดวงจรมัลติเพล็กเซอร์ที่เขียนด้วยคำสั่ง Case

รูปที่ 2.38 โค้ดวงจรมัลติเพล็กเซอร์ 4 อินพุต 1 เอาต์พุต

หมายเหตุ SENSITIVITY LIST กือ อินพุตทุกตัวที่มีผลต่อการเปลี่ยนแปลงของ Process โดยตรงทันที (ไม่ใช่โดยทางอ้อม)

ตัวอย่างที่ 2.19 ฝึกใช้คำสั่ง if และ case ใน การออกแบบวงจร มัลติเพล็กเซอร์ 4 อินพุต 1 เอาต์พุต ในตัวอย่างที่ 2.8 ซึ่งสามารถเขียนโค้ด VHDL ได้ดังรูปที่ 2.39a) และรูปที่ 2.39b)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity MUX4TO1 is
6     port ( A,B,C,D : in STD_LOGIC;
7             S : in STD_LOGIC_VECTOR(1 downto 0);
8             O : out STD_LOGIC);
9 end MUX4TO1;
10
11 architecture BEHAVIORAL of MUX4TO1 is
12 begin
13     process(A,B,C,D,S)
14     begin
15         if      S="00" then O <= A;
16         elsif S="01" then O <= B;
17         elsif S="10" then O <= C;
18         else            O <= D;   -- S="11"
19     end if;
20     end process;
21 end BEHAVIORAL;

```

2.39a) โค้ดวงจร มัลติเพล็กเซอร์ที่เขียนด้วยคำสั่ง if

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity MUX4TO1 is
6     port ( A,B,C,D : in STD_LOGIC;
7             S : in STD_LOGIC_VECTOR(1 downto 0);
8             O : out STD_LOGIC);
9 end MUX4TO1;
10
11 architecture BEHAVIORAL of MUX4TO1 is
12 begin
13     process(A,B,C,D,S)
14     begin
15         case S is
16             when "00" => O <= A;
17             when "01" => O <= B;
18             when "10" => O <= C;
19             when others => O <= D; -- S="11"
20         end case;
21     end process;
22 end BEHAVIORAL;

```

2.39b) โค้ด凰จรมัลติเพล็กเซอร์ที่เขียนด้วยคำสั่ง case

รูปที่ 2.39 โค้ด凰จรมัลติเพล็กเซอร์ 4 อินพุต 1 เอาต์พุต

จากตัวอย่างในรูปที่ 2.39 นั้นบัส S มีขนาด 2 บิตประกอบด้วย S(1) และ S(0) ดังนั้นการเขียนโค้ดแบบนี้จึงมองได้ทั้งที่เป็นรูปแบบของบัสและรูปแบบของแต่ละบิตได้อยู่แล้ว ขอให้ผู้อ่านทำความเข้าใจการเขียนโค้ดในรูปแบบที่เป็นบัสและเป็นแต่ละบิตให้ดี ซึ่งจะเห็นได้อย่างชัดเจนว่าการเขียนโค้ดในรูปแบบที่เป็นบัสจะสะดวกและง่ายกว่าตัวอย่างในรูปที่ 2.38

ตัวอย่างที่ 2.20 ฟิกใช้คำสั่ง if และ case ในการออกแบบวงจรบวกและลบ (Unsigned) ขนาด 4 บิตในตัวอย่างที่ 2.9 ที่มี A และ B เป็นอินพุต และมี RESULT เป็นเอาต์พุตขนาด 5 บิต โดยมี OPER ควบคุมการบวกหรือลบ โดยโค้ดต่างๆ ที่ได้แสดงดังรูปที่ 2.40a) ถึงรูปที่ 2.40f)

หมายเหตุ เมื่อ $A < B$ แล้วถ้าเรา $A-B$ จะทำให้ได้ผลลัพธ์แบบ 2's complement

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ADD_SUB_4BIT is
7     Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
8             OPER : in STD_LOGIC;
9             RESULT : out STD_LOGIC_VECTOR (4 downto 0));
10 end ADD_SUB_4BIT;
11
12 architecture Behavioral of ADD_SUB_4BIT is
13     begin
14 process(A,B,OPER)
15     begin
16         if OPER='0' then RESULT <= ('0' &A) + ('0' &B);
17         else
18             RESULT <= ('0' &A) - ('0' &B);
19         end if;
20     end process;
21 end Behavioral;

```

ต้องเรียกใช้ std_logic_unsigned package เพราะใช้ Operator “+” และ “-” กับชนิดข้อมูล std_logic_vector

ใช้ Operator “+” และ “-” กับชนิดข้อมูล std_logic_vector

2.40a) โค้ด凰จรบวกและลบบอต่ายที่เขียนด้วยคำสั่ง if

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ADD_SUB_4BIT is
7     Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
8             OPER : in STD_LOGIC;
9             RESULT : out STD_LOGIC_VECTOR (4 downto 0));
10 end ADD_SUB_4BIT;
11
12 architecture Behavioral of ADD_SUB_4BIT is
13     begin
14 process(A,B,OPER)
15     begin
16         case OPER is
17             when '0'    => RESULT <= ('0' &A) + ('0' &B);
18             when others => RESULT <= ('0' &A) - ('0' &B);
19         end case;
20     end process;
21 end Behavioral;

```

ต้องเรียกใช้ std_logic_unsigned package เพราะใช้ Operator “+” และ “-” กับชนิดข้อมูล std_logic_vector

ใช้ Operator “+” และ “-” กับชนิดข้อมูล std_logic_vector

2.40b) โค้ดวงจรบวกและลบอย่างง่ายที่เขียนด้วยคำสั่ง Case

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity ADD_SUB_4BIT is
7     Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
8             OPER : in STD_LOGIC;
9             RESULT : out STD_LOGIC_VECTOR (4 downto 0));
10 end ADD_SUB_4BIT;
11
12 architecture Behavioral of ADD_SUB_4BIT is
13     begin
14 process(A,B,OPER)
15 begin
16     if OPER='0' then RESULT <= STD_LOGIC_VECTOR(unsigned('0' &A) + unsigned('0' &B));
17     else                 RESULT <= STD_LOGIC_VECTOR(unsigned('0' &A) - unsigned('0' &B));
18     end if;
19 end process;
20 end Behavioral;

```

ต้องเรียกใช้ numeric_std package เพราะใช้ Operator “+” และ “-” กับชนิดข้อมูล unsigned

ใช้ “+” และ “-” กับ std_logic_vector ไม่ได้ จึงต้องแปลงเป็น unsigned เมื่อเสร็จแล้วจึงแปลงกลับคืน

2.40c) โค้ดวงจรบวกและลบอย่างง่ายที่เขียนด้วยคำสั่ง if และใช้ชนิดข้อมูลใช้ชนิดข้อมูล std_logic_vector

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity ADD_SUB_4BIT is
7     Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
8             OPER : in STD_LOGIC;
9             RESULT : out STD_LOGIC_VECTOR (4 downto 0));
10 end ADD_SUB_4BIT;
11
12 architecture Behavioral of ADD_SUB_4BIT is
13     begin
14 process(A,B,OPER)
15 begin
16     case OPER is
17         when '0'    => RESULT <= STD_LOGIC_VECTOR(unsigned('0' &A) + unsigned('0' &B));
18         when others => RESULT <= STD_LOGIC_VECTOR(unsigned('0' &A) - unsigned('0' &B));
19     end case;
20 end process;
21 end Behavioral;

```

ต้องเรียกใช้ numeric_std package เพราะใช้ Operator “+” และ “-” กับชนิดข้อมูล unsigned

ใช้ “+” และ “-” กับ std_logic_vector ไม่ได้ จึงต้องแปลงเป็น unsigned เมื่อเสร็จแล้วจึงแปลงกลับคืน

2.40d) โค้ดวงจรบวกและลบอย่างง่ายที่เขียนด้วยคำสั่ง case และใช้ชนิดข้อมูล std_logic_vector

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity ADD_SUB_4BIT is
7     Port ( A,B : in  unsigned (3 downto 0);
8             OPER : in  STD_LOGIC;
9             RESULT : out  unsigned (4 downto 0));
10 end ADD_SUB_4BIT;
11 architecture Behavioral of ADD_SUB_4BIT is
12 begin
13 process(A,B,OPER)
14 begin
15     if OPER='0' then RESULT <= ('0'&A) + ('0'&B);
16     else                 RESULT <= ('0'&A) - ('0'&B);
17     end if;
18 end process;
19 end Behavioral;

```

ต้องเรียกใช้ numeric_std package เพราะใช้ Operator
“+” และ “-” กับชนิดข้อมูล unsigned

2.40e) โค้ดวงจรบวกและลบอย่างง่ายที่เขียนด้วยคำสั่ง if และใช้ชนิดข้อมูล unsigned

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity ADD_SUB_4BIT is
7     Port ( A,B : in  unsigned (3 downto 0);
8             OPER : in  STD_LOGIC;
9             RESULT : out  unsigned (4 downto 0));
10 end ADD_SUB_4BIT;
11 architecture Behavioral of ADD_SUB_4BIT is
12 begin
13 process(A,B,OPER)
14 begin
15     case OPER is
16         when '0'      => RESULT <= ('0'&A) + ('0'&B);
17         when others => RESULT <= ('0'&A) - ('0'&B);
18     end case;
19 end process;
20 end Behavioral;

```

ต้องเรียกใช้ numeric_std package เพราะใช้ Operator
“+” และ “-” กับชนิดข้อมูล unsigned

2.40f) โค้ดวงจรบวกและลบอย่างง่ายที่เขียนด้วยคำสั่ง case และใช้ชนิดข้อมูล unsigned

รูปที่ 2.40 โค้ดวงจรบวกและลบอย่างง่าย

ตัวอย่างที่ 2.21 ฝึกใช้คำสั่ง if และ case ใน การออกแบบวงจร decoder 2 to 4 (2 to 4 Decoder) โดยอินพุตเป็นบัส 2 บิต และเอาต์พุตเป็นบัส 4 บิต ในตัวอย่างที่ 2.10 โดยโค้ดต่างๆ ที่ได้แสดงดังรูปที่ 2.41a) ถึงรูปที่ 2.41b)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DECODER2TO4 is
6     port ( A : in  STD_LOGIC_VECTOR(1 downto 0);
7             Y : out  STD_LOGIC_VECTOR(3 downto 0));
8 end DECODER2TO4;
9

```

(ต่อ)

```

10 architecture BEHAVIORAL of DECODER2TO4 is
11 begin
12 process(A)
13 begin
14     if A="00" then Y <= "0001"; -- Y(3)='0',Y(2)='0',Y(1)='0',Y(0)='1'
15     elsif A="01" then Y <= "0010"; -- Y(3)='0',Y(2)='0',Y(1)='1',Y(0)='0'
16     elsif A="10" then Y <= "0100"; -- Y(3)='0',Y(2)='1',Y(1)='0',Y(0)='0'
17     else Y <= "1000"; -- Y(3)='1',Y(2)='0',Y(1)='0',Y(0)='0'
18     end if;
19 end process;
20 end BEHAVIORAL;

```

2.41a) โค้ดวงจร 2 to 4 Decoder ที่เขียนด้วยคำสั่ง if

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DECODER2TO4 is
6     port ( A : in STD_LOGIC_VECTOR(1 downto 0);
7             Y : out STD_LOGIC_VECTOR(3 downto 0));
8 end DECODER2TO4;
9
10 architecture BEHAVIORAL of DECODER2TO4 is
11 begin
12 process(A)
13 begin
14     case A is
15         when "00" => Y <= "0001"; -- Y(3)='0',Y(2)='0',Y(1)='0',Y(0)='1'
16         when "01" => Y <= "0010"; -- Y(3)='0',Y(2)='0',Y(1)='1',Y(0)='0'
17         when "10" => Y <= "0100"; -- Y(3)='0',Y(2)='1',Y(1)='0',Y(0)='0'
18         when others => Y <= "1000"; -- Y(3)='1',Y(2)='0',Y(1)='0',Y(0)='0'
19     end case;
20 end process;
21 end BEHAVIORAL;

```

2.41b) โค้ดวงจร 2 to 4 Decoder ที่เขียนด้วยคำสั่ง case

รูปที่ 2.41 โค้ด VHDL ของวงจร 2 to 4 Decoder

ตัวอย่างที่ 2.22 ฟังก์ชันที่ใช้คำสั่ง if และ case ในการออกแบบวงจร decoder ที่สัตว์แสดงผลเซกเมนต์แบบคอมมอนแอดจ์ โดยใช้รูปแบบเลขฐานสิบหกแสดงดังรูปที่ 2.15a) และมีตารางความจริงแสดงดังรูปที่ 2.15b) ในตัวอย่างที่ 2.3 โค้ดของวงจรที่ได้แสดงดังรูปที่ 2.42a) และรูปที่ 2.42b)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity HEX_TO_7SEGMENT is
6     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
7             Y : out STD_LOGIC_VECTOR (6 downto 0));
8 end HEX_TO_7SEGMENT;
9
10 architecture Behavioral of HEX_TO_7SEGMENT is
11 begin

```

(ต่อ)

```

12 process(A)
13 begin          --gfedcba
14   if A="1111" then Y <= "1110001"; --F
15   elsif A="1110" then Y <= "1111001"; --E      Y(0)=a
16   elsif A="1101" then Y <= "1011110"; --d      ---
17   elsif A="1100" then Y <= "0111001"; --C      Y(5)=f | Y(1)=b
18   elsif A="1011" then Y <= "1111100"; --b      --- Y(6)=g
19   elsif A="1010" then Y <= "1110111"; --A      Y(4)=e | Y(2)=c
20   elsif A="1001" then Y <= "1101111"; --9      ---
21   elsif A="1000" then Y <= "1111111"; --8      Y(3)=d
22   elsif A="0111" then Y <= "0000111"; --7
23   elsif A="0110" then Y <= "1111101"; --6
24   elsif A="0101" then Y <= "1101101"; --5
25   elsif A="0100" then Y <= "1100110"; --4
26   elsif A="0011" then Y <= "1001111"; --3
27   elsif A="0010" then Y <= "1011011"; --2
28   elsif A="0001" then Y <= "0000110"; --1
29   else               Y <= "0111111"; --0
30   end if;
31 end process;
32 end Behavioral;

```

2.42a) โค้ดว่างรอกอกรหัสตัวแสดงผลเซเว่นเซกเมนต์ที่เขียนด้วยคำสั่ง if

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity HEX_TO_7SEGMENT is
6   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
7         Y : out STD_LOGIC_VECTOR (6 downto 0));
8 end HEX_TO_7SEGMENT;
9
10 architecture Behavioral of HEX_TO_7SEGMENT is
11 begin
12 process(A)
13 begin
14   case A is          --gfedcba
15     when "1111" => Y <= "1110001"; --F
16     when "1110" => Y <= "1111001"; --E      Y(0)=a
17     when "1101" => Y <= "1011110"; --d      ---
18     when "1100" => Y <= "0111001"; --C      Y(5)=f | Y(1)=b
19     when "1011" => Y <= "1111100"; --b      --- Y(6)=g
20     when "1010" => Y <= "1110111"; --A      Y(4)=e | Y(2)=c
21     when "1001" => Y <= "1101111"; --9      ---
22     when "1000" => Y <= "1111111"; --8      Y(3)=d
23     when "0111" => Y <= "0000111"; --7
24     when "0110" => Y <= "1111101"; --6
25     when "0101" => Y <= "1101101"; --5
26     when "0100" => Y <= "1100110"; --4
27     when "0011" => Y <= "1001111"; --3
28     when "0010" => Y <= "1011011"; --2
29     when "0001" => Y <= "0000110"; --1
30     when others => Y <= "0111111"; --0
31   end case;
32 end process;
33 end Behavioral;

```

2.42b) โค้ดว่างรอกอกรหัสตัวแสดงผลเซเว่นเซกเมนต์ที่เขียนด้วยคำสั่ง case

รูปที่ 2.42 โค้ด VHDL ของวงจรอกรหัสตัวแสดงผลเซเว่นเซกเมนต์

ตัวอย่างที่ 2.23 ฝึกใช้คำสั่ง if และ case ในการออกแบบวงจรถอดรหัสตัวแสดงผลเซเว่นเซกเมนต์แบบคอมมอนแคทโดยพาราเลข 0-9 และเมื่ออินพุตเป็น 10-15 เอาต์พุตจะเป็น 0 ดังตัวอย่างที่ 2.12 โคล็อกของวงจรที่ได้แสดงดังรูปที่ 2.43a) และรูปที่ 2.43b)

```

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 process(A)
13 begin
14     if A="1001" then Y <= "1101111"; --9
15     elsif A="1000" then Y <= "1111111"; --8      Y(0)=a
16     elsif A="0111" then Y <= "0000111"; --7      ---
17     elsif A="0110" then Y <= "1111101"; --6 Y(5)=f | Y(1)=b
18     elsif A="0101" then Y <= "1101101"; --5      --- Y(6)=g
19     elsif A="0100" then Y <= "1100110"; --4 Y(4)=e | Y(2)=c
20     elsif A="0011" then Y <= "1001111"; --3      ---
21     elsif A="0010" then Y <= "1011011"; --2      Y(3)=d
22     elsif A="0001" then Y <= "0000110"; --1
23     else                 Y <= "0111111"; --0,A,b,C,d,E,F
24     end if;
25 end process;
26 end Behavioral;
```

2.43a) โคล็อกของวงจรถอดรหัสตัวแสดงผลเซเว่นเซกเมนต์ที่เขียนด้วยคำสั่ง if

```

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 process(A)
13 begin
14     case A is
15         when "1001" => Y <= "1101111"; --9
16         when "1000" => Y <= "1111111"; --8      Y(0)=a
17         when "0111" => Y <= "0000111"; --7      ---
18         when "0110" => Y <= "1111101"; --6 Y(5)=f | Y(1)=b
19         when "0101" => Y <= "1101101"; --5      --- Y(6)=g
20         when "0100" => Y <= "1100110"; --4 Y(4)=e | Y(2)=c
21         when "0011" => Y <= "1001111"; --3      ---
22         when "0010" => Y <= "1011011"; --2      Y(3)=d
23         when "0001" => Y <= "0000110"; --1
24         when others => Y <= "0111111"; --0,A,b,C,d,E,F
25     end case;
26 end process;
27 end Behavioral;
```

2.43b) โคล็อกของวงจรถอดรหัสตัวแสดงผลเซเว่นเซกเมนต์ที่เขียนด้วยคำสั่ง case

รูปที่ 2.43 โคล็อก VHDL ของวงจรถอดรหัสตัวแสดงผลเซเว่นเซกเมนต์ได้ผลพาราเลข 0-9

ตัวอย่างที่ 2.24 ฝึกใช้คำสั่ง if และ case ออกรูปแบบของเรเข้ารหัสในตัวอย่างที่ 2.13 โค๊ดที่ได้แสดงดังรูปที่ 2.44a) และรูปที่ 2.44b)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity PRI_ENCODER_BCD is
6     Port ( A : in STD_LOGIC_VECTOR (9 downto 0);
7             B : out STD_LOGIC_VECTOR (3 downto 0));
8 end PRI_ENCODER_BCD;
9 architecture Behavioral of PRI_ENCODER_BCD is
10 begin
11 process(A)
12 begin
13     if A(9)'=0' then B <= "1001"; --Key pad No.9
14     elsif A(8)'=0' then B <= "1000"; --Key pad No.8
15     elsif A(7)'=0' then B <= "0111"; --Key pad No.7
16     elsif A(6)'=0' then B <= "0110"; --Key pad No.6
17     elsif A(5)'=0' then B <= "0101"; --Key pad No.5
18     elsif A(4)'=0' then B <= "0100"; --Key pad No.4
19     elsif A(3)'=0' then B <= "0011"; --Key pad No.3
20     elsif A(2)'=0' then B <= "0010"; --Key pad No.2
21     elsif A(1)'=0' then B <= "0001"; --Key pad No.1
22     elsif A(0)'=0' then B <= "0000"; --Key pad No.0
23     else
24         B <= "1111";
25     end if;
26 end process;
27 end Behavioral;
```

2.44a) โค๊ดของวงจรเข้ารหัสที่มีลำดับความสำคัญที่เขียนโดยใช้คำสั่ง if

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity PRI_ENCODER_BCD is
7     Port ( A : in STD_LOGIC_VECTOR (9 downto 0);
8             B : out STD_LOGIC_VECTOR (3 downto 0));
9 end PRI_ENCODER_BCD;
10 architecture Behavioral of PRI_ENCODER_BCD is
11     signal X : integer range 0 to 1023;
12 begin
13     X <= conv_integer(A);
14 process(X)
15 begin
16     case X is
17         when 0 to 511      => B <= "1001"; --Key pad No.9
18         when 512 to 767    => B <= "1000"; --Key pad No.8
19         when 768 to 895    => B <= "0111"; --Key pad No.7
20         when 896 to 959    => B <= "0110"; --Key pad No.6
21         when 960 to 991    => B <= "0101"; --Key pad No.5
22         when 992 to 1007   => B <= "0100"; --Key pad No.4
23         when 1008 to 1015   => B <= "0011"; --Key pad No.3
24         when 1016 to 1019   => B <= "0010"; --Key pad No.2
25         when 1020 to 1021   => B <= "0001"; --Key pad No.1
26         when 1022           => B <= "0000"; --Key pad No.0
27         when 1023           => B <= "1111";
28     end case;
29 end process;
30 end Behavioral;
```

2.44b) โค๊ดของวงจรเข้ารหัสที่มีลำดับความสำคัญที่เขียนโดยใช้คำสั่ง case

รูปที่ 2.44 โค๊ดของวงจรเข้ารหัสที่มีลำดับความสำคัญที่ใช้แปลงสัญญาณปุ่มกดเป็นรหัส BCD

ตัวอย่างที่ 2.25 ทำความเข้าใจเกี่ยวกับ Sensitivity list โดยใช้โน้ตดาวงจร D Flip-flop แสดงดังรูปที่ 2.45 โดยที่ C คือ สัญญาณนาฬิกา (Clock) ที่ทริกด้วยขอบขาขึ้นหรือขอบลง รูปที่ 2.45a) เป็น D Flip-flop แบบ Asynchronous clear กล่าวคือ เมื่อขา CLR = '1' และเอาต์พุต Q = '0' (ทันที) โดยไม่ต้องรอการทริกด้วยสัญญาณนาฬิกา ส่วนรูปที่ 2.45b) เป็น D Flip-flop แบบ Synchronous reset กล่าวคือ เมื่อขา RESET = '1' แล้วจะรีเซ็ต Q = '0' แต่จะต้องรอสัญญาณนาฬิกา (Clock) ทริกก่อน แล้วจึงรีเซ็ตเอาต์พุต Q = '0' ผู้อ่านควรจดจำความหมายของคำว่า Asynchronous clear หรือ Clear และ Synchronous reset หรือ Reset ไว้ให้เพื่อป้องกันความสับสน

```

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 นั้น จะมีผลต่อเอาต์พุต Q โดยตรง (ทันที) จึงต้องเพียง C และ CLR ไว้ใน Sensitivity list

C'event and C='1' คือ C ทริกด้วยขอบขาขึ้น หรือขอบลง ซึ่งจะส่งผลให้ไม่ได้

2.45a) โน้ตดาวงจร 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 ไม่มีผลโดยตรงต่อเอาต์พุต Q (ทันที) เพราะต้องรอการทริกจาก C แต่การทริกด้วย C จะมีผลต่อเอาต์พุต Q โดยตรง (ทันที) ดังนั้นจึงต้องเพียง C เพียงตัวเดียวเท่านั้นไว้ใน Sensitivity list

C'event and C='1' คือ C ทริกด้วยขอบขาขึ้น หรือขอบลง ซึ่งจะส่งผลให้ไม่ได้

2.45b) โน้ตดาวงจร D Flip-flop แบบ Synchronous reset

รูปที่ 2.45 โน้ตดาวงจร D Flip-flop

ในกรณีของ D Flip-flop แบบ Asynchronous clear นั้นจะต้องเพียงอินพุต C และ CLR ไว้ใน Sensitivity list ในโน้ตรูปที่ 2.46a) เพราะว่าถ้าอินพุต C และ CLR เปลี่ยนสถานะก็จะมีผลต่อเอาต์พุต Q ทันที แต่ถ้าเป็น D Flip-flop แบบ Synchronous reset นั้นเราจะเพียงอินพุต C เพียงตัวเดียวไว้ใน Sensitivity list ในโน้ตรูปที่ 2.45b) เพราะว่าเอาต์พุต Q เปลี่ยนสถานะก็ต่อเมื่อมีการทริกด้วยอินพุต C (สัญญาณนาฬิกา) เท่านั้น

ตัวอย่างที่ 2.26 ฝึกการใช้ Mode “buffer” และ Mode “out” การใช้ Signal และ Variable รวมทั้งการเรียกใช้ std_logic_unsigned package หรือ numeric_std package โดยเขียนໂຄດວງຈຮນບັນ 16 (4 ບົຕ) ແບນັບຂຶ້ນແສດງດັ່ງຮູບທີ 2.46a) ດີງຮູບທີ 2.46h)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER4BIT is
6     port ( C,CLR : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end COUNTER4BIT;
9
10 architecture Behavioral of COUNTER4BIT is
11     signal X : integer range 0 to 15;
12 begin
13     -----counter : 0-15-----
14     process(C,CLR)
15     begin
16         if CLR='1' then X <= 0;
17         elsif C'event and C='1' then
18             if X >= 15 then X <= 0;
19             else
20                 X <= X + 1;
21             end if;
22         end if;
23     end process;
24     -----Integer to binary decoder-----
25     Q <= "0000" when X=0 else
26         "0001" when X=1 else
27         "0010" when X=2 else
28         "0011" when X=3 else
29         "0100" when X=4 else
30         "0101" when X=5 else
31         "0110" when X=6 else
32         "0111" when X=7 else
33         "1000" when X=8 else
34         "1001" when X=9 else
35         "1010" when X=10 else
36         "1011" when X=11 else
37         "1100" when X=12 else
38         "1101" when X=13 else
39         "1110" when X=14 else
40         "1111";
41 end Behavioral;

```

C'event and C='1' ຄືອ C ທີ່ກິດຕ້າຍຂອບໜ້າຂຶ້ນ
ຫຼືຂອບໜ້າວກ ຊັ່ງຈະໄສ່ວາເລີນຫຼືໄວ່ໄສກໍໄດ້

2.46a) ໂຄດວງຈຮນບັນ 16 ແບນັບຂຶ້ນທີ່ມີເອາະພຸດເປັນໜົດຂໍ້ມູນ Integer ກ່ອນຈະແປ່ງກຳລັບເປັນ std_logic_vector

```

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;

```

ເຮັດໃຊ້ Package ຊື່ std_logic_unsigned ເພື່ອໃຫ້ໜົດ
ຂໍ້ມູນ std_logic_vector ສາມາຮັດໃຊ້ກັນ “+” ໄດ້

Port Q ສາມາຮັດອ່ານຄ່າກຳລັບໄດ້
ດັ່ງນັ້ນຈຶ່ງຕ້ອງໃຊ້ Mode “buffer”

ເປັນການອ່ານຄ່າຂອງ Port Q ກຳລັບເຂົ້າມາ
ເພື່ອເພີ່ມຄ່າກົງລະ 1 ແລ້ວສັງຄ່າໄໝໄປ
ທີ່ Port Q ອີກກົງ ນັ້ນກີ່ແສດງວ່າ Port Q
ເປັນ Port ທີ່ອ່ານຄ່າກຳລັບໄດ້ຫຼື “buffer”

2.46b) ໂຄດວງຈຮນບັນ 16 ແບນັບຂຶ້ນທີ່ມີເອາະພຸດ Q ເປັນໜົດຂໍ້ມູນ std_logic_vector ແລະມີ Mode ເປັນ “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 เมื่อจากการ Update ค่าของ Signal จะกระทำเมื่อจบ Process

2.46c) 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 !ท่านั้น

Variable assignment

Variable ใช้ได้เฉพาะภายใน Process เท่านั้น ดังนั้นการส่งผ่านค่าออกนอก Process จึงต้อง Assign ค่า Q_var ให้กับ Q (Port หรือ Signal) ก่อน

2.46d) 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

2.46e) 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 กายใน Process เท่านั้น

แปลงชนิดข้อมูลของ Q_var จาก unsigned ไปเป็น std_logic_vector แล้วกำหนดค่าให้กับ Q

2.46f) โค้ดวงจรนับ 16 แบบนับขึ้นที่มีเอาต์พุต Q เป็นชนิดข้อมูล std_logic_vector และ Variable 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 unsigned (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 <= Q_temp;
21 end Behavioral;

```

เรียกใช้ Package ชื่อ numeric_std เพื่อให้ชนิดข้อมูล unsigned 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

2.46g) โค้ดวงจรนับ 16 แบบนับขึ้นที่มีเอาต์พุต Q 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 unsigned (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 <= 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 กายใน Process เท่านั้น

Variable ใช้ได้เฉพาะภายใน Process เท่านั้น ดังนั้นการส่งผ่านค่าออกนอก Process จึงต้อง Assign ค่า Q_var ให้กับ Q (Port หรือ Signal)

2.46h) โค้ดวงจรนับ 16 แบบนับขึ้นที่มีเอาต์พุต Q และ Variable เป็น unsigned

รูปที่ 2.46 โค้ดวงจรนับ 16 แบบนับขึ้น

จากรูปที่ 2.46a) เป็นโค้ดวงจรนับ 16 ให้มีเอาต์พุตเป็นชนิดข้อมูล Integer ก่อน เพราะว่าชนิดข้อมูล std_logic_vector ไม่สามารถใช้กับ Operator “+” ได้ หากนั้นจึงแปลงเอาต์พุต X ที่ได้กลับมาเป็น Q ซึ่งเป็นชนิดข้อมูล std_logic_vector อีกรึ

จากรูปที่ 2.46b) ในบรรทัดที่ 16 นั้น $Q \leq Q + 1$ คือ การอ่านค่า Port Q ที่เป็นเอาต์พุตกลับเข้ามาในวงจรเพื่อเพิ่มค่า ครั้งละ 1 แล้วส่งค่าเอาต์พุตออกไปอีกรึ แสดงว่า Port Q เป็นเอาต์พุตที่อ่านค่ากลับได้ จึงต้องใช้ Port mode เป็น “buffer” ในบรรทัดที่ 8 และจากโค้ดในรูปที่ 2.47b) ลิงรูปที่ 2.47g) เป็นการเพิ่มโค้ดเพื่อหลีกเลี่ยงการใช้ Port mode ที่เป็น “buffer” โดยการประกาศใช้ Signal หรือ Variable (ซึ่งไม่ใช่ Port) ดังนั้น Q จึงมี Port mode เป็น “out”

ชนิดข้อมูล std_logic_vector ที่อยู่ในโค้ดรูปที่ 2.46b) ลิงรูปที่ 2.46d) นั้นสามารถบอกกันได้โดยตรงระหว่างกับว่าเป็นชนิดข้อมูล integer เมื่อมีการเรียกใช้ std_logic_unsigned package ที่อยู่ใน Library ชื่อ IEEE ดังนั้น $Q \leq Q + 1$ จึงให้ผลเช่นเดียวกับ $Q \leq Q + "0001"$ หรือ $Q \leq Q + '1'$

ชนิดข้อมูล unsigned ในโค้ดรูปที่ 2.46e) ลิงรูปที่ 2.46h) สามารถบอกกันได้โดยตรงระหว่างกับว่าเป็นชนิดข้อมูล integer เมื่อมีการเรียกใช้ numeric_std package ที่อยู่ใน Library ชื่อ IEEE ดังนั้น $Q \leq Q + 1$ จึงให้ผลเช่นเดียวกับ $Q \leq Q + "0001"$ หรือ $Q \leq Q + "1"$ (“1” เป็นชนิดข้อมูลแบบอะเรย์ ในขณะที่ ‘1’ เป็นชนิดข้อมูลแบบบิต)

จากรูปที่ 2.46d) รูปที่ 2.46f) และรูปที่ 2.46h) นั้น Variable จะใช้ได้เฉพาะภายใน Process เท่านั้น การส่งผ่านค่าออกนอก Process จึงต้อง Assign ค่าให้กับ Port หรือ Signal ดังนั้น Variable จึงเป็นเพียงที่เก็บค่าชั่วคราวในระหว่างการทำงานภายใน Process Variable สามารถ Update ค่าได้ทันทีโดยไม่ต้องรอให้จบ Process ซึ่งจะต่างจาก Signal ที่ใช้ได้ทุกส่วนของโค้ดแต่การ Update ค่าจะทำได้เมื่อจบ Process ดังนั้นโค้ดในรูปที่ 2.46c) รูปที่ 2.46e) และรูปที่ 2.46g) จึงต้อง Assign ค่า Signal ภายนอก Process การ Assign ค่า Signal ภายใน Process สามารถทำได้ชั่วกันแต่ค่าที่ได้จะเป็นค่าที่ยังไม่ถูกทำการ Update

ถ้าเอาโค้ดวงจรนับ 16 (4 บิต) ในรูปที่ 2.46c) ลิงรูปที่ 2.46f) ไปจำลองการทำงาน (Simulation) ก็จะพบว่าค่าเอาต์พุตเป็น ‘U’ (Uninitialized) และเมื่อมีสัญญาณพิเศษทิริกวงจรนับ ค่าเอาต์พุตจะเป็น ‘X’ (Unknown) และเมื่อ Clear (CLR) เป็นลอจิก ‘1’ ค่าเอาต์พุตจะเป็น “0000” การแก้ไขปัญหานี้สามารถทำได้ด้วยการใส่ค่าเริ่มต้นดังตัวอย่างในรูปที่ 2.47 ผลจำลองการทำงานของโค้ดที่ไม่ใส่ค่าเริ่มต้นและถูกใจใส่ค่าเริ่มต้นแสดงลงรูปที่ 2.48a) และรูปที่ 2.48b) ภาษา VHDL ยอมให้ใส่ค่าเริ่มต้นได้เฉพาะกรณีของการจำลองการทำงาน แต่ Xilinx Synthesis Tool หรือ XST ยอมให้ใส่ค่าเริ่มต้นให้กับโค้ดที่สังเคราะห์ได้

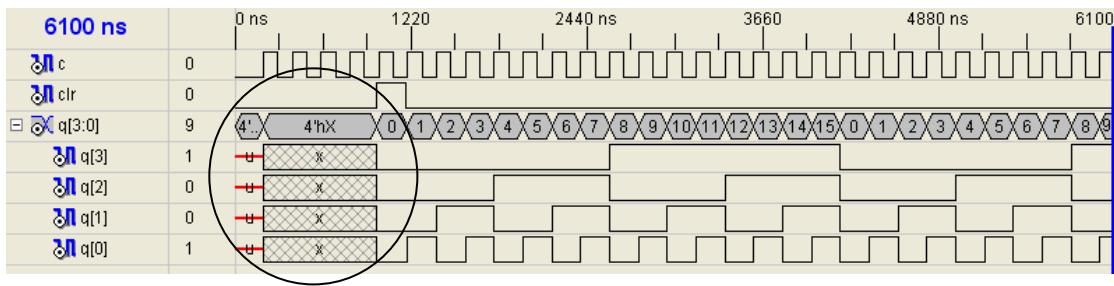
```

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;

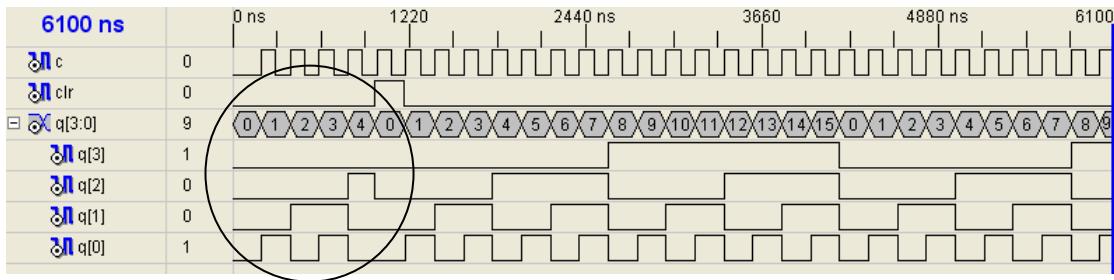
```

การใส่ค่าเริ่มต้น Q_temp
ให้มีค่าเป็น “0000”

รูปที่ 2.47 ตัวอย่างโค้ดวงจรนับ 16 ที่ใส่ค่าเริ่มต้นเป็น “0000”



2.48a) ผลจำลองการทำงานของโค้ดวงจรนับ 16 ที่ไม่ได้ค่าเริ่มต้น



2.48b) ผลจำลองการทำงานของโค้ดวงจรนับ 16 ที่ได้ค่าเริ่มต้นเป็น “0000”

รูปที่ 2.48 ผลจำลองการทำงานของโค้ดวงจรนับ 16

แม้ว่า Xilinx Synthesis Tool หรือ XST จะรองรับการใช้ Port ประเภท Buffer แต่ยังไรก็ตาม XST ได้แนะนำให้หลีกเลี่ยงการใช้ Buffer ดังนี้ในตัวอย่างต่อๆ ไปเราจะไม่พยามมาเขียนโค้ดที่ใช้ Buffer

ตัวอย่างที่ 2.27 ฝึกการใช้ Signal และ Variable รวมทั้งการเรียกใช้ std_logic_unsigned package หรือ numeric_std package โดยเขียนโค้ดวงจรนับ 10 (4 บิต) แบบนับขึ้นแสดงดังรูปที่ 2.49a) ถึงรูปที่ 2.49d)

```

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";
19             else QT <= QT + 1;
20             end if;
21         end if;
22     end process;
23     Q <= QT;
24 end Behavioral;

```

เรียกใช้ Package ชื่อ std_logic_unsigned เพื่อให้ชนิดข้อมูล std_logic_vector สามารถใช้กับ “+” ได้

เมื่อค่า QT >= 9 (หรือ Q >= “1001”) แล้วมีสัญญาณนาฬิกา C ทริกค้างของขาขึ้นจะทำให้ QT มีค่าเป็น “0000”

2.49a) โค้ดวงจรนับ 10 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 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";
19             else QT := QT + 1;
20             end if;
21         end if;
22         Q <= QT;
23     end process;
24 end Behavioral;

```

เรียกใช้ Package ชื่อ std_logic_unsigned เพื่อให้ชนิด
ข้อมูล std_logic_vector สามารถใช้กับ “+” ได้

เมื่อค่า QT ≥ 9 (หรือ Q $\geq "1001"$)
แล้วมีสัญญาณนาฬิกา C ทริกค์ข้อมูล
ขาขึ้นจะทำให้ QT มีค่าเป็น “0000”

2.49b) โค้ดวงจรรับ 10 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 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 : UNSIGNED (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";
19             else QT <= QT + 1;
20             end if;
21         end if;
22     end process;
23     Q <= STD_LOGIC_VECTOR ( QT );
24 end Behavioral;

```

เรียกใช้ Package ชื่อ numeric_std เพื่อให้ชนิดข้อมูล
unsigned สามารถใช้กับ “+” ได้

เมื่อค่า QT ≥ 9 (หรือ Q $\geq "1001"$)
แล้วมีสัญญาณนาฬิกา C ทริกค์ข้อมูล
ขาขึ้นจะทำให้ QT มีค่าเป็น “0000”

2.49c) โค้ดวงจรรับ 10 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 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

```

เรียกใช้ Package ชื่อ numeric_std เพื่อให้ชนิดข้อมูล
unsigned สามารถใช้กับ “+” ได้

```

13   process(C,CLR)
14     variable QT : UNSIGNED (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";
19       else QT := QT + 1;
20       end if;
21     end if;
22     Q <= STD_LOGIC_VECTOR ( QT );
23   end process;
24 end Behavioral;

```

เมื่อค่า QT ≥ 9 (หรือ Q $\geq "1001"$)
แล้วมีสัญญาณนาฬิกา C ทริกค้างของ
ขาขึ้นจะทำให้ QT มีค่าเป็น "0000"

2.49d) โค้ดวงจรนับ 10 แบบนับขึ้นที่มีआट्पुट Q เป็นชนิดข้อมูล std_logic_vector และ Variable เป็น unsigned

รูปที่ 2.49 โค้ดวงจรนับ 10 แบบนับขึ้น

จากรูปที่ 2.49 โค้ดวงจรนับ 10 มีขนาด 4 บิต ดังนี้จึงควรเขียน $Q \geq 9$ แทนการเขียนเป็น $Q = 9$ กีเพื่อป้องกันไม่ให้
วงจรนับมีโอกาสบันออก Range คือ 10-15 (คือ 1010, 1011, 1100, 1101, 1110, 1111) ซึ่งอาจเกิดขึ้นได้ขณะเริ่มต้นจ่ายไฟเลี้ยงหรือ
วงจรลูกรบกวนจากภายนอก ซึ่งในกรณีที่เขียนเป็น $Q = 9$ ถ้าเริ่มต้นนั้นค่าที่นับอยู่นอก Range วงจรจะนับต่อไปจนถึง 15
(1111) ก่อนแล้วจึงกลับมาบันใน Range ตามปกติ (0-9) อีกครั้ง ซึ่งก็หมายความว่างจรนับมีโอกาสบันผิดพลาดในรอบแรก

ตัวอย่างที่ 2.28 โค้ดวงจรนับ 10 แบบนับขึ้น-นับลงที่มีขา Clear และแสดงผลทางเซเว่นเซกเมนต์ กำหนดให้วงจรนับมีค่าเริ่มต้น
เป็น 0 เราจะเขียนโค้ดโดยประการใช้ Signal เป็นชนิดข้อมูล integer เพื่อจะได้ไม่ต้องเรียกใช้ std_logic_unsigned package หรือ
numeric_std package และดังรูปที่ 2.50 ซึ่งวงจรนี้ประกอบด้วยคำสั่งคอนแคร์เรนต์ 3 คำสั่งหรือ 3 วงจรย่อย คือ วงจรนับ 10
แบบนับขึ้น-นับลง วงจรดอครหัสตัวแสดงผลทางเซเว่นเซกเมนต์ และวงจรต่อคอมมอนแคปติดลงกราวด์

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_1DIGIT is
6   Port ( C,CLR,DIR : in STD_LOGIC;
7          COMMON : out STD_LOGIC; --Common cathode = GND
8          Y : out STD_LOGIC_VECTOR (6 downto 0));
9 end COUNTER_1DIGIT;
10
11 architecture Behavioral of COUNTER_1DIGIT is
12   signal QT : integer range 0 to 9 := 0;
13 begin
14   process(C,CLR)
15   begin
16     if CLR='1' then QT <= 0;
17     elsif C'event and C='1' then
18       if DIR='0' then -- Count up
19         if QT>9 then QT <= 0;
20         else QT <= QT + 1;
21         end if;
22       else -- Count down
23         if (QT=0 or QT>9) then QT <= 9;
24         else QT <= QT - 1;
25         end if;
26       end if;
27     end if;
28   end process;
29

```

เนื่องจากเราประการ Signal QT เป็นชนิดข้อมูล
integer จึงสามารถใช้กับ "+" ได้โดยตรงโดยไม่
ต้องเรียกใช้ std_logic_unsigned package หรือ
Numeric_std package แต่อย่างใด และขอให้
สังเกตว่าเรากำหนดค่าเริ่มต้นเป็น 0 อีกด้วย

ควรเขียน QT > 9 เป็นไปด้วยเพื่อ^{*}
ป้องกันการบันนอก Range คือ
10-15 ซึ่งอาจเกิดขึ้นขณะเริ่มต้น
จ่ายไฟเลี้ยงหรือวงจรลูกรบกวน

```

30   process(QT)
31     begin          --gfedcba
32       if      QT=9 then Y <= "1101111";  --9
33       elsif QT=8 then Y <= "1111111";  --8      Y(0)=a
34       elsif QT=7 then Y <= "0000111";  --7      ---
35       elsif QT=6 then Y <= "1111101";  --6  Y(5)=f | Y(1)=b
36       elsif QT=5 then Y <= "1101101";  --5      --- Y(6)=g
37       elsif QT=4 then Y <= "1100110";  --4  Y(4)=e | Y(2)=c
38       elsif QT=3 then Y <= "1001111";  --3      ---
39       elsif QT=2 then Y <= "1011011";  --2      Y(3)=d
40       elsif QT=1 then Y <= "0000110";  --1
41       else           Y <= "0111111";  --0
42     end if;
43   end process;
44
45   COMMON <= '0'; --Common cathode = '0' (GND)
46
47 end Behavioral;

```

รูปที่ 2.50 วงจรนับ 10 แบบนับขึ้น-นับลง 1 หลักที่แสดงผลทางเซเวนเชกเมนต์ (ที่มีค่าเริ่มต้นเป็น 0)

เนื่องจากเราเคยเขียนโค้ดของวงจรนับ 10 ที่แสดงผลทางเซเวนเชกเมนต์ไว้แล้วในรูปที่ 2.29a) รูปที่ 2.29b) รูปที่ 2.43a) และรูปที่ 2.43b) ถ้าเราเลือกโค้ดในรูปที่ 2.29b) มาทำเป็น Component เราสามารถเขียนโค้ดรูปที่ 2.50 ใหม่ได้ดังรูปที่ 2.51

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;          -- เรียกใช้ Package ชื่อ std_logic_unsigned เพื่อให้ชนิด
5                                         -- ข้อมูล std_logic_vector สามารถใช้กับ "+" ได้
6 entity COUNTER_1DIGIT is
7   Port ( C,CLR,DIR : in STD_LOGIC;
8         COMMON : out STD_LOGIC; --Common cathode = GND
9         Y : out STD_LOGIC_VECTOR (6 downto 0));
10 end COUNTER_1DIGIT;
11
12 architecture Behavioral of COUNTER_1DIGIT is
13   component DECODER_7SEGMENT
14     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
15            Y : out STD_LOGIC_VECTOR (6 downto 0));
16   end component;
17   signal QT : STD_LOGIC_VECTOR (3 downto 0) := "0000";
18 begin
19   process(C,CLR)
20     begin
21       if      CLR='1' then QT <= "0000";
22       elsif C'event and C='1' then
23         if DIR='0' then -- Count up
24           if QT>=9 then QT <= "0000";
25           else QT <= QT + 1;
26           end if;
27         else           -- Count down
28           if (QT=0 or QT>9) then QT <= "1001";
29           else QT <= QT - 1;
30           end if;
31         end if;
32       end if;
33     end process;
34
35 DISPLAY : DECODER_7SEGMENT port map(A=>QT, Y=>Y);
36
37   COMMON <= '0'; --Common cathode = '0' (GND)
38
39 end Behavioral;

```

รูปที่ 2.51 โค้ดวงจรนับ 10 แบบนับขึ้น-นับลง 1 หลักที่แสดงผลทางเซเวนเชกเมนต์ที่มีการใช้ Component

จากโลจิครูปที่ 2.51 นั้นมี INSTANCE_NAME ชื่อว่า DISPLAY ซึ่งการตั้งชื่อ INSTANCE_NAME นั้นควรตั้งชื่อให้สื่อความหมายที่ทำให้การอ่านໂຄดง่ายขึ้น และให้ผู้อ่านสังเกตว่าการเขียนໂຄดนี้เราจำเป็นต้องการประกาศใช้ Signal เป็นชนิดข้อมูล std_logic_vector เนื่องจากอินพุตของวงจรต้องหัสตัวแสดงผลเซเว่นเซกเมนต์ที่ใช้ทำ Component นั้นเป็นชนิดข้อมูล std_logic_vector ดังนั้นการเขียนໂຄดนี้จะต้องเรียกใช้ std_logic_unsigned package เพื่อให้สามารถใช้กับ “+” หรือ “-” ได้

ตัวอย่างที่ 2.29 ฝึกการใช้ Signal และ Variable โดยเขียนໂຄด Shift register (Shift left) 4 บิตแสดงดังรูปที่ 2.52a) ถึงรูปที่ 2.52c) โดยมีผลจำลองการทำงานดังรูปที่ 2.52e) ส่วนໂຄดในรูปที่ 2.52d) นั้นจะไม่เป็น Shift register 4 บิต โดยมีผลจำลองการทำงานดังรูปที่ 2.52f) ซึ่งการเขียนໂຄด Shift register (Shift left) 4 บิตที่สะคอกกว่านั้นควรเป็นดังรูปที่ 2.52g) หรือดังรูปที่ 2.52h)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER_4BIT is
6     Port ( C,D_IN : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end SHIFT_REGISTER_4BIT;
9
10 architecture Behavioral of SHIFT_REGISTER_4BIT is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0);
12 begin
13     process(C)
14     begin
15         if (C'event and C='1') then
16             QT(0) <= D_IN;
17             QT(1) <= QT(0);
18             QT(2) <= QT(1);
19             QT(3) <= QT(2);
20         end if;
21     end process;
22     Q <= QT;
23 end Behavioral;
```

2.52a) ໂຄดวงจร Shift register (Shift left) 4 บิตที่เขียนโดยการประกาศใช้ signal

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER_4BIT is
6     Port ( C,D_IN : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end SHIFT_REGISTER_4BIT;
9
10 architecture Behavioral of SHIFT_REGISTER_4BIT is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0);
12 begin
13     process(C)
14     begin
15         if (C'event and C='1') then
16             QT(3) <= QT(2);
17             QT(2) <= QT(1);
18             QT(1) <= QT(0);
19             QT(0) <= D_IN;
20         end if;
21     end process;
22     Q <= QT;
23 end Behavioral;
```

2.52b) ໂຄดวงจร Shift register (Shift left) 4 บิตที่เขียนโดยประกาศใช้ signal แต่เขียนໂຄดบรรทัดที่ 16-19 หลับกัน

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER_4BIT is
6     Port ( C,D_IN : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end SHIFT_REGISTER_4BIT;
9
10 architecture Behavioral of SHIFT_REGISTER_4BIT is
11 begin
12     process(C)
13         variable Q_temp : STD_LOGIC_VECTOR (3 downto 0);
14     begin
15         if (C'event and C='1') then
16             Q_temp(3) := Q_temp(2);
17             Q_temp(2) := Q_temp(1);
18             Q_temp(1) := Q_temp(0);
19             Q_temp(0) := D_IN;
20         end if;
21         Q <= Q_TEMP;
22     end process;
23 end Behavioral;

```

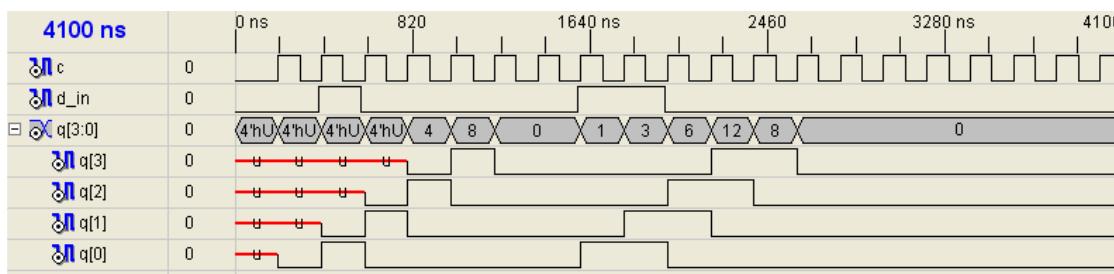
2.52c) โค้ดควบจր Shift register (Shift left) 4 บิตที่เขียนโดยปราการใช้ Variable

```

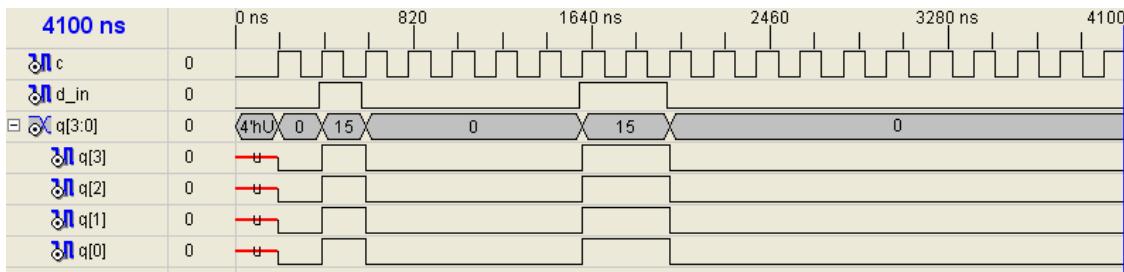
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER_4BIT is
6     Port ( C,D_IN : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end SHIFT_REGISTER_4BIT;
9
10 architecture Behavioral of SHIFT_REGISTER_4BIT is
11 begin
12     process(C)
13         variable QT : STD_LOGIC_VECTOR (3 downto 0);
14     begin
15         if (C'event and C='1') then
16             QT(0) := D_IN;
17             QT(1) := QT(0);
18             QT(2) := QT(1);
19             QT(3) := QT(2);
20         end if;
21         Q <= QT;
22     end process;
23 end Behavioral;

```

2.52d) โค้ดควบจร Register 1 บิต (ไม่ใช่ Shift register 4 บิต) ที่เกิดจากการเขียนโค้ดไม่ถูกต้อง



2.52e) ผลข้างของ การทำงานของ โค้ดควบจร Shift register (Shift left) ที่เขียนในรูปที่ 2.52a ถึงรูปที่ 2.52c



2.52f) ผลจำลองการทำงานของโค้ดในรูปที่ 2.52d) ที่เกิดจากการเขียนโค้ดไม่มีถูกต้อง

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER_4BIT is
6     Port ( C,D_IN : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end SHIFT_REGISTER_4BIT;
9
10 architecture Behavioral of SHIFT_REGISTER_4BIT is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0);
12 begin
13     process(C)
14     begin
15         if (C'event and C='1') then
16             QT <= QT(2 downto 0) & D_IN;
17         end if;
18     end process;
19     Q <= QT;
20 end Behavioral;

```

2.52g) โค้ดวงจร Shift register (Shift left) 4 บิตที่เขียนโดยประกาศใช้ signal และ Operator “&”

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER_4BIT is
6     Port ( C,D_IN : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end SHIFT_REGISTER_4BIT;
9
10 architecture Behavioral of SHIFT_REGISTER_4BIT is
11 begin
12     process(C)
13         variable QT : STD_LOGIC_VECTOR (3 downto 0);
14     begin
15         if (C'event and C='1') then
16             QT := QT(2 downto 0) & D_IN;
17         end if;
18         Q <= QT;
19     end process;
20 end Behavioral;

```

2.52h) โค้ดวงจร Shift register (Shift left) 4 บิตที่เขียนโดยประกาศใช้ Variable และ Operator “&”

รูปที่ 2.52 โค้ดวงจร Shift register (Shift left) 4 บิต

ซึ่งโค้ดรูปที่ 2.52a) และรูปที่ 2.52b) นี้จะให้ผลลัพธ์เหมือนกัน เพราะว่า QT(0) ถึง QT(3) เป็น Signal จึงต้องรออัพเดตค่าหลังจากจบ Process และเท่านั้น ส่วนโค้ดในรูปที่ 2.52c) ซึ่งเป็น Variable ก็ให้ผลลัพธ์เช่นเดียวกับโค้ดรูปที่ 2.52a) และรูปที่ 2.52b) เพราะคำสั่งที่อยู่หลัง “then” อยู่ในคำสั่งซีเคานเชียล ดังนั้นคำสั่งที่อยู่ในบรรทัดบนจึงถูกอัพเดตค่าก่อนโดยไม่

ต้องรอให้จบ Process ซึ่งทั้ง 3 กรณีนี้เมื่อสังเคราะห์ว่างจรแล้วได้รีจิสเตอร์ (Register) 4 ตัว โดยที่ผลจำลองการทำงานของโค้ดทั้ง 3 วงจรแรกแสดงดังรูปที่ 2.52e) ซึ่งจะต่างจากโค้ดในรูปที่ 2.52d) ที่มี QT เป็น Variable เช่นกัน แต่เนื่องจากคำสั่งที่อยู่หลัง “then” อยู่ในคำสั่งซึ่งความเชี่ยวลึกสามารถอพเดตค่าได้ทันทีโดยไม่ต้องรอให้จบ Process จึงทำให้ QT(3) := D_IN; , QT(2) := D_IN; , QT(1) := D_IN; และ QT(0) := D_IN; ซึ่งได้รีจิสเตอร์ (Register) เพียง 1 ตัวเท่านั้น ดังนั้นโค้ดในรูปที่ 2.52d) จึงไม่เป็นวงจร Shift register ขนาด 4 บิต โดยที่ผลจำลองการทำงานจะเป็นดังรูปที่ 2.52f) โค้ด Shift register (Shift left) 4 บิตนี้สามารถเขียนให้ถูกต้องโดยใช้ “&” ได้ดังรูปที่ 2.52g) และรูปที่ 2.52h)

ในทำนองเดียวกันเราสามารถเขียนโก้ดีช่วงขวา Shift register (Shift right) N ($N=4$) นิทได้ดังรูปที่ 2.53a และรูปที่ 2.53b โดยเราจะเห็นว่าโค้ดให้อยู่ในรูปฟอร์มทั่วไปด้วยการใช้คำสั่ง Generic และใช้ Attribute คือ “left”

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER is
6     generic (N : integer := 4);
7     Port ( C,D_IN : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (N-1 downto 0));
9 end SHIFT_REGISTER;
10
11 architecture Behavioral of SHIFT_REGISTER is
12     signal QT : STD_LOGIC_VECTOR (N-1 downto 0);
13 begin
14     process(C)
15     begin
16         if (C'event and C='1') then
17             QT <= D_IN & QT(QT'left downto 1); --QT'left = N-1
18         end if;
19     end process;
20     Q <= QT;
21 end Behavioral;

```

2.53a) โค๊ดคงจร Shift register (Shift right) N (N=4) บิตที่เขียนโดยประกาศใช้ signal และ Operator “&”

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER is
6     generic (N : integer := 4);
7     Port ( C,D_IN : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (N-1 downto 0));
9 end SHIFT_REGISTER;
10
11 architecture Behavioral of SHIFT_REGISTER is
12 begin
13     process(C)
14         variable QT : STD_LOGIC_VECTOR (N-1 downto 0);
15     begin
16         if (C'event and C='1') then
17             QT := D_IN & QT(QT'left downto 1); --QT'left = N-1
18         end if;
19         Q <= QT;
20     end process;
21 end Behavioral;

```

2.53b) ໂຄສະຈາກ Shift register (Shift right) N ($N=4$) ບີທີ່ເປັນໂຄຍປະກາດໃໝ່ Variable ແລະ Operator “&”

ตัวอย่างที่ 2.30 ฝึกใช้คำสั่ง Null ร่วมกับคำสั่ง case โดยการออกแบบวงจรดอครหัสตัวแสดงผลเซเว่นเซกเมนต์แบบคอมมอนแคโนดเฉพาะเลข 0-9 และเมื่ออินพุตเป็น 10-15 เอาต์พุตจะเป็น 0 ดังตัวอย่างที่ 2.23 โค้ดของวงจรที่ได้แสดงดังรูปที่ 2.54

จากโค้ดในรูปที่ 2.54 คำสั่ง null คือ คำสั่งไม่ทำอะไรมาก็จะเป็น 1010-1111 ดังนั้นาเอต์พุตจะแสดงค่าเริ่มต้นคือ $Y \leq "0111111"$ แต่ถ้าไม่กำหนดค่าเริ่มก็จะเกิด Latch โดยจะแสดงค่าเอาต์พุตสุดท้ายก่อนที่อินพุตจะเป็น 1010-1111 แทน

```

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 process(A)
13 begin
14   Y <= "0111111"; --Initial value : 0,a,b,C,d,E,F = 0
15   case A is
16     --gfedcba
17     when "1001" => Y <= "1101111"; --9
18     when "1000" => Y <= "1111111"; --8      Y(0)=a
19     when "0111" => Y <= "0000111"; --7      ---
20     when "0110" => Y <= "1111101"; --6 Y(5)=f| Y(1)=b
21     when "0101" => Y <= "1101101"; --5      --- Y(6)=g
22     when "0100" => Y <= "1100110"; --4 Y(4)=e| Y(2)=c
23     when "0011" => Y <= "1001111"; --3      ---
24     when "0010" => Y <= "1011011"; --2      Y(3)=d
25     when "0001" => Y <= "0000110"; --1
26     when others => null;
27   end case;
28 end process;
29 end Behavioral;
```

รูปที่ 2.54 โค้ดตัวดอครหัสเซเว่นเซกเมนต์ได้เฉพาะเลข 0-9 แบบ Common cathode ที่เขียนด้วยคำสั่ง case และ null

2.14.5 Wait statement

คำสั่ง wait เป็นคำสั่งซึ่งควบคุมที่เขียนไว้ในคำสั่ง Process ที่ไม่มี Sensitivity lists (Processes without sensitivity lists) เพื่อให้ Process หยุดทำงาน ณ ตำแหน่งที่ต้องการที่ซึ่งมีคำสั่ง wait อัญญาต์จะทำงานเมื่อคำสั่ง wait ได้ hely ตำแหน่ง ซึ่งจะแตกต่างจากการรันของ Process ที่มี Sensitivity list (Processes with sensitivity lists) กล่าวคือ Process จะทำงานทุกครั้งเมื่อสัญญาณอินพุตบางตัวใน Sensitivity lists เกิดการเปลี่ยนแปลงหรือเกิด Event และเมื่อที่คำสั่งใน Process ตามลำดับจนหมดทุกคำสั่งแล้วจึงจะหยุด รูปแบบการเขียนคำสั่ง wait เป็นดังนี้

```

wait [ on SENSITIVITY_LIST ];
[ until CONDITION ];
[ for TIME_EXPRESSION ];

```

โดยที่ 1) wait on SENSITIVITY_LIST; เป็นการสั่งให้ Process หยุดจนกว่า Signal ใน SENSITIVITY_LIST บางตัวเกิด Event ซึ่ง Signal ใน SENSITIVITY_LIST อาจจะมีได้หลายตัว ถ้าคำสั่ง wait on เป็นคำสั่งสุดท้ายใน Process ก็จะให้ผลเทียบเท่ากับคำสั่ง Process ที่มี Sensitivity list แต่ XST ไม่รองรับการเขียนโค้ดรูปแบบนี้

- 2) wait until CONDITION; เป็นการสั่งให้ Process หยุด الرحمنกว่า CONDITION (บุลิน) จะเป็นจริง คำสั่ง wait until ต้องเป็นคำสั่งแรกใน Process จึงให้ผลเทียบท่ากับคำสั่ง Process ที่มี Sensitivity list (Processes with sensitivity lists)
- 3) wait for TIME_EXPRESSION; เป็นการสั่งให้ Process หยุดรอเป็นเวลาที่กำหนด (เพื่อจำลองการทำงานของวงจร)
- 4) Xilinx synthesis tool หรือ XST รองรับการใช้คำสั่ง wait on หรือ wait until ได้เพียง Clock signal เดียวเท่านั้น

ตัวอย่างที่ 2.31 ฝึกเขียนโค้ดโดยใช้คำสั่ง wait on ของวงจรแอนด์เกต 3 อินพุต โดยที่โค้ดที่ใช้คำสั่ง Processes with sensitivity lists และคำสั่ง Processes without sensitivity lists (ใช้คำสั่ง wait on) แสดงดังรูปที่ 2.55a) และรูปที่ 2.55b) ตามลำดับ

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity WAIT_ON_EX is
6   Port ( A,B,C : in STD_LOGIC;
7         D : out STD_LOGIC);
8 end WAIT_ON_EX;
9
10 architecture Behavioral of WAIT_ON_EX is
11 begin
12   process (A,B,C)
13   begin
14     D <= A and B and C;
15   end process;
16 end Behavioral;

```

2.55a) โค้ดของวงจรแอนด์เกต 3 อินพุตที่เขียนโดยใช้คำสั่ง Processes with sensitivity lists

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity WAIT_ON_EX is
6   Port ( A,B,C : in STD_LOGIC;
7         D : out STD_LOGIC);
8 end WAIT_ON_EX;
9
10 architecture Behavioral of WAIT_ON_EX is
11 begin
12   process
13   begin
14     D <= A and B and C;
15     wait on A,B,C;
16   end process;
17 end Behavioral;

```

2.55b) โค้ดของวงจรแอนด์เกต 3 อินพุตที่เขียนโดยใช้คำสั่ง Processes without sensitivity lists

รูปที่ 2.55 โค้ดของวงจรแอนด์เกต 3 อินพุตที่เขียนด้วยคำสั่ง wait on (XST ไม่รองรับโค้ดที่มี Clock มากกว่า 1 ตัว)

คำสั่ง wait ต่อไปนี้ XST รองการใช้และมีความหมายเดียวกัน ก็อ CLK ทริกด้วยขอบขาขึ้น ก็ต่อเมื่อเขียนไว้ที่คำสั่งแรกใน Process และจะให้ผลเทียบท่ากับคำสั่ง Processes with sensitivity lists ซึ่งได้แก่

```

wait on CLK until CLK = '1';
wait until CLK = '1';
wait until CLK'event and CLK = '1';

```

ตัวอย่างที่ 2.32 ฝึกเขียนโค้ดของวงจร D Flip-flop แบบ Synchronous reset โดยโค้ดที่เขียนด้วยคำสั่ง Processes with sensitivity lists และคำสั่ง Processes without sensitivity lists แสดงดังรูปที่ 2.56a) และรูปที่ 2.56b) ตามลำดับ

```

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)--Processes with sensitivity lists
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;

```

2.56a) โค้ดงาน D Flip-flop ที่เขียนด้วยคำสั่ง Processes with sensitivity lists

```

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      --Processes without sensitivity lists
13 begin
14     wait until C'event and C='1';
15         if RESET='1' then Q <= '0';
16         else Q <= D;
17         end if;
18 end process;
19 end BEHAVIORAL;

```

2.56b) โค้ดงาน D Flip-flop ที่เขียนด้วยคำสั่ง Processes without sensitivity lists

รูปที่ 2.56 โค้ดงาน D Flip-flop ที่เขียนด้วยคำสั่ง wait

การใช้คำสั่ง wait หลายคำสั่ง (Multiple Wait statements) ภายใน Process เดียวกันในการออกแบบวงจรซีเควนเชียล (Sequential circuit) เมื่อใช้กับ Xilinx synthesis tool หรือ XST นั้นจะมีข้อจำกัดที่ควรปฏิบัติตั้งนี้

- ใน Process จะต้องมีคำสั่งลูป (Loop statement) เพียงลูปเดียวเท่านั้น (คำสั่งลูปจะอธิบายในหัวข้อถัดไป)
- คำสั่งแรกที่อยู่ในลูปจะต้องเป็นคำสั่ง wait (Wait statement)
- หลังคำสั่ง wait แต่ละคำสั่งจะต้องตามด้วยคำสั่ง Next หรือคำสั่ง Exit
- Condition ที่อยู่ในคำสั่ง wait แต่ละคำสั่งจะต้องเหมือนกันและเป็น Clock signal เพียงตัวเดียวเท่านั้น มีรูปแบบดังนี้

```

wait [on CLOCK_SIGNAL] until [(CLOCK_SIGNAL'event | not CLOCK_SIGNAL'stable) and ]
CLOCK_SIGNAL = {'0' | '1'};

```

2.14.6 Loop statements

คำสั่งลูป (Loop) เป็นคำสั่งซึ่งควบคุมที่คล้ายกับคำสั่ง Generate (คำสั่งคอนโทรลเรนต์) โดยเขียนไว้ในคำสั่ง Process, Function หรือ Procedure โดยที่คำสั่ง Loop จะเป็นคำสั่งในการทำคำสั่งซึ่งควบคุมเชิงลึกๆ มีรูปแบบการเขียนดังนี้

```
[ loop_label : ] [ ITERATION_SCHEME ] loop
    {SEQUENTIAL_STATEMENT}
    { next [ loop_label ] [ when condition ] ; }
    { exit [ loop_label ] [ when condition ] ; }
end loop [ loop_label ] ;
```

- โดยที่
- 1) ITERATION_SCHEME มี 3 แบบ คือ Loop (Basic loop), For/loop และ while/loop
 - 2) SEQUENTIAL_STATEMENT คือ คำสั่งซึ่งควบคุมเชิงลึก
 - 3) next คือ คำสั่งซึ่งควบคุมที่ใช้ออกจากลูป (Loop iteration) ที่ทำในขณะนั้น โดยไม่สนใจคำสั่งที่เหลือทั้งหมดเพื่อข้ามไปทำคำสั่งที่อยู่ในลูปทั้งหมดในรอบการทำงานถัดไป
 - 4) exit คือ คำสั่งซึ่งควบคุมที่ใช้ออกจากลูป (Loop iteration) ที่ทำในขณะนั้น โดยไม่สนใจคำสั่งที่เหลือทั้งหมดเพื่อจบการทำงานของลูป (end loop) และไปทำคำสั่งถัดไปที่อยู่ต่อจาก end loop;

1) Basic loop

Basic loop คือ คำสั่ง Loop ที่ไม่มี ITERATION_SCHEME คำสั่งนี้จะสั่งให้ทำงานคำสั่งซึ่งควบคุมเชิงลึกๆ การออกจากลูปจะใช้คำสั่ง next และ exit และในคำสั่ง loop จะมีคำสั่ง wait อย่างน้อย 1 คำสั่ง คำสั่ง Basic loop มีรูปแบบการเขียนดังนี้

```
[ loop_label : ] loop
    {SEQUENTIAL_STATEMENT}
end loop [ loop_label ] ;
```

2) While/loop

While/loop คือ คำสั่งที่ทำการคำสั่งซึ่งควบคุมเชิงลึกๆ ตราบใดที่ CONDITION (บูลีน) เป็นจริง คำสั่งนี้ไม่สามารถนำไปสั่งเคราะห์เป็นวงจรได้ ตัวอย่างการใช้คำสั่งนี้มีรายละเอียดในบทที่ 6 ข้อ 6.2 คำสั่ง While/loop มีรูปแบบการเขียนเป็นดังนี้

```
[ loop_label : ] while CONDITION loop
    {SEQUENTIAL_STATEMENT}
end loop [ loop_label ] ;
```

3) For/loop

For/loop คือ คำสั่งที่ทำการคำสั่งซึ่งควบคุมเชิงลึกๆ ตราบใดค่าของ INDEX_NAME อยู่ใน DISCRETE_RANGE ที่เป็น integer (แบบ to และ downto) ในคำสั่งนี้จะต้องไม่มีคำสั่ง wait อยู่ภายในลูป คำสั่ง For/loop มีรูปแบบการเขียนดังนี้

```
[ loop_label : ] INDEX_NAME in DISCRETE_RANGE loop
    {SEQUENTIAL_STATEMENT}
end loop [ loop_label ];
```

ตัวอย่างที่ 2.33 ฝึกใช้คำสั่ง Processes with sensitivity lists และคำสั่ง Processes without sensitivity lists ร่วมกับคำสั่ง loop ใน การเขียนโก้ด์ Shift register (Shift left) 4 บิต ในตัวอย่างที่ 2.29 แสดงดังรูปที่ 2.57a) ลึกรูปที่ 2.57d) โดยที่ i คือ INDEX_NAME

```
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER_4BIT is
6     Port ( C,D_IN : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end SHIFT_REGISTER_4BIT;
9
10 architecture Behavioral of SHIFT_REGISTER_4BIT is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0);
12 begin
13     process(C) --Processes with sensitivity lists
14     begin
15         if (C'event and C='1') then
16             QT(0) <= D_IN;
17             for i in 0 to 2 loop
18                 QT(i+1) <= QT(i);           DISCRETE_RANGE ไป to
19             end loop;
20         end if;
21     end process;
22     Q <= QT;
23 end Behavioral;
```

2.57a) โค้ดงาน Shift register (Shift left) 4 บิตที่เขียนด้วยคำสั่ง Processes with sensitivity lists

```
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER_4BIT is
6     Port ( C,D_IN : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end SHIFT_REGISTER_4BIT;
9
10 architecture Behavioral of SHIFT_REGISTER_4BIT is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0);
12 begin
13     process(C) --Processes with sensitivity lists
14     begin
15         if (C'event and C='1') then
16             QT(0) <= D_IN;
17             for i in 2 downto 0 loop           DISCRETE_RANGE ไป downto
18                 QT(i+1) <= QT(i);
19             end loop;
20         end if;
21     end process;
22     Q <= QT;
23 end Behavioral;
```

2.57b) โค้ดงาน Shift register (Shift left) 4 บิตที่เขียนด้วยคำสั่ง Processes with sensitivity lists

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER_4BIT is
6     Port ( C,D_IN : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end SHIFT_REGISTER_4BIT;
9
10 architecture Behavioral of SHIFT_REGISTER_4BIT is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0);
12 begin
13 process           --Processes without sensitivity lists
14 begin
15     wait until (C'event and C='1');
16     QT(0) <= D_IN;
17     for i in 0 to 2 loop
18         QT(i+1) <= QT(i);      DISCRETE_RANGE แบบ to
19     end loop;
20 end process;
21     Q <= QT;
22 end Behavioral;

```

2.57c) โค้ดควบ Shift register (Shift left) 4 บิตที่เขียนด้วยคำสั่ง Processes without sensitivity lists

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SHIFT_REGISTER_4BIT is
6     Port ( C,D_IN : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end SHIFT_REGISTER_4BIT;
9
10 architecture Behavioral of SHIFT_REGISTER_4BIT is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0);
12 begin
13 process           --Processes without sensitivity lists
14 begin
15     wait until (C'event and C='1');
16     QT(0) <= D_IN;
17     for i in 2 downto 0 loop
18         QT(i+1) <= QT(i);      DISCRETE_RANGE แบบ downto
19     end loop;
20 end process;
21     Q <= QT;
22 end Behavioral;

```

2.57d) โค้ดควบ Shift register (Shift left) 4 บิตที่เขียนด้วยคำสั่ง Processes without sensitivity lists

รูปที่ 2.57 โค้ดควบ Shift register (Shift left) 4 บิตที่เขียนด้วยคำสั่ง loop

2.15 ตัวอย่างการออกแบบวงจรที่ใช้งานจริง

เพื่อให้เห็นภาพรวมในการออกแบบวงจรที่มีขนาดใหญ่ เราจะแสดงการเขียนโค้ดโดยใช้คำสั่งหลายรูปแบบรวมกัน

ตัวอย่างที่ 2.34 วงจรนับ 4 หลักที่สามารถนับขึ้น-นับลงและเคลียร์ได้ โดยจะแสดงผลทางตัวแสดงผลเลขเวนเชกเม้นต์ เมื่อ DIR = '0' และ CLR = '0' และกดปุ่ม COUNT (แบบ Active low หรือ COUNT = '0') ทำให้วงจรจะนับขึ้น และเมื่อ DIR = '1' และ CLR = '0' และกดปุ่ม COUNT ทำให้วงจรนับลง แต่ถ้า CLR = '1' และเอาต์พุตเป็น 0000 (เลขฐานสิบ) ถ้ากำหนดให้ออสซิเลเตอร์ที่ใช้ = 32.768 kHz และให้นำโค้ดควบจรนับสิบดังรูปที่ 2.58 (ซึ่งจะอธิบายในบทที่ 4) มาทำเป็น Component เพื่อใช้กับวงจรนับทั้ง 4 หลัก โดยที่โค้ดของจรนับที่ออกแบบแล้วเสร็จสมบูรณ์แสดงดังรูปที่ 2.59

จากโค้ด VHDL ของวงจรนับในรูปที่ 2.59 วงจนับนี้จะประกอบด้วยวงจรย่อขึ้นมาดังนี้ คือ

- วงจร 10 Bits frequency generator จะใช้ในการสร้างความถี่ 32 Hz และ 64 Hz ให้กับวงจรสแกน (Scan) ตัวแสดงผล เซเวนเชกเม้นต์ทั้ง 4 หลัก โดยความถี่สแกนแต่ละหลัก = $32 \text{ Hz} > 30 \text{ Hz}$ ครั้งต่อวินาทีเพื่อไม่ให้เกิดการกระพริบที่ตัวแสดงผล และใช้สร้างความถี่ 128 Hz ($< 150 \text{ Hz}$) สำหรับวงจร Debouncer (ประสิทธิภาพสูง)
- วงจร Debouncer จะใช้สำหรับแก้ไขปัญหาในกรณีที่หน้าสัมผัสของปุ่มกด (ปุ่ม COUNT) แตกกันหรือจากกันไม่สนิท ขณะทำการกดปุ่ม ซึ่งการกดปุ่มในแต่ละครั้งอาจให้พัลส์ออกมากกว่า 1 พัลส์นั้นจะทำให้วงจนับนับผิดพลาดได้ ซึ่งเราเรียกว่าเกิดเบนซ์ (Bouncing) รายละเอียดของวงจร Debouncer จะอธิบายในบทที่ 4
- วงจร Bidirectional counter : 0-9999 เป็นวงจนับที่ประกอบด้วยวงจนับ 10 จำนวน 4 ชุด โดยการนำเอาโค้ดของ วงจนับ 10 แบบนับขึ้น-นับลงในรูปที่ 2.58 มาสร้างเป็น Component รายละเอียดของวงจนับนี้จะอธิบายในบทที่ 4
- วงจรสแกน ประกอบด้วย วงจร 4 Bits MUX4TO1 เป็นมัลติเพล็กเซอร์ขนาด 4 บิตแบบเข้า 4 ออก 1 เพื่อนำค่าเอาต์พุต จากวงจนับแต่ละหลักไปทั่ววงจรอุดรหัสตัวแสดงผลเซเวนเชกเม้นต์ และวงจร 2 to 4 Decoder ที่มีเอาต์พุตเป็นแบบ Active low (One cold decoder) จะใช้ในการเลือกขา Common cathode ของแต่ละหลักเพื่อให้ตัวแสดงผลหลักที่ต้องการติดสว่าง
- BCD to 7 Segment decoder จะเป็นวงจรอุดรหัสตัวแสดงผลเซเวนเชกเม้นต์ โดยการนำเอาโค้ดของวงจรอุดรหัสตัวแสดงผลเซเวนเชกเม้นต์ในรูปที่ 2.29b มาสร้างเป็น Component

```

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;

```

รูปที่ 2.58 โค้ดของวงจนับ 10 แบบนับขึ้น-นับลงที่มีขา CE และ CEO

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity COUNTER_4DIGIT_UPDN is
7     Port(CLR : in STD_LOGIC;           --CLEAR
8          COUNT : in STD_LOGIC;         --COUNT
9          DIR : in STD_LOGIC;          --DIR(UP/DOWN Direction)
10         OSC : in STD_LOGIC;          --OSC=32.768kHz
11         Y : out STD_LOGIC_VECTOR(6 downto 0);    --7 Segment
12         COMMON : out STD_LOGIC_VECTOR(3 downto 0));--COMMON Cathode
13 end COUNTER_4DIGIT_UPDN;
14
15 architecture Behavioral of COUNTER_4DIGIT_UPDN is
16     component C10UPDN_CE
17         Port ( C,CE,CLR,DIR : in STD_LOGIC;
18                 CEO : out STD_LOGIC;
19                 Q : out STD_LOGIC_VECTOR (3 downto 0));
20     end component;
21     component DECODER_7SEGMENT
22         Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
23                 Y : out STD_LOGIC_VECTOR (6 downto 0));
24     end component;
25     signal F : STD_LOGIC_VECTOR (9 downto 0);
26     signal QT : STD_LOGIC_VECTOR (3 downto 0);
27     signal C : STD_LOGIC;
28     signal CE_0,CE_1,CE_2,CE_3 : STD_LOGIC;
29     signal Q0,Q1,Q2,Q3 : STD_LOGIC_VECTOR (3 downto 0);
30     signal Q_DIGIT : STD_LOGIC_VECTOR (3 downto 0);
31 begin
32     -----10 Bits frequency generator-----
33     process(OSC)
34     begin --OSC=32.768kHz=(2**15)Hz,F(9)=OSC/(2** (9+1))=32Hz,F(8)=64Hz,F(7)=128Hz
35         if (OSC'event and OSC='1') then F <= F + 1;
36     end if;
37 end process;
38     -----Debouncer-----
39     process( F(7) )
40     begin--Debouncer frequency : F(7)=128Hz<150Hz
41         if (F(7)'event and F(7)='1') then
42             QT <= QT(2 downto 0)& (not COUNT);--COUNT active low
43         end if;
44     end process;
45         C <= QT(0) and QT(1) and QT(2) and not QT(3);
46     -----Bidirectional counter : 0-9999-----
47         CE_0 <= '1';           --CE='1'
48     DIGIT_0 : C10UPDN_CE port map( C  => C,
49                                     CE => CE_0,
50                                     CLR=> CLR,
51                                     DIR=> DIR,
52                                     CEO=> CE_1,
53                                     Q  => Q0);
54     DIGIT_1 : C10UPDN_CE port map( C  => C,
55                                     CE => CE_1,
56                                     CLR=> CLR,
57                                     DIR=> DIR,
58                                     CEO=> CE_2,
59                                     Q  => Q1);
60     DIGIT_2 : C10UPDN_CE port map( C  => C,
61                                     CE => CE_2,
62                                     CLR=> CLR,
63                                     DIR=> DIR,
64                                     CEO=> CE_3,
65                                     Q  => Q2);

```

```

66  DIGIT_3 : C10UPDN_CE port map( C => C,
67                               CE => CE_3,
68                               CLR=> CLR,
69                               DIR=> DIR,
70                               CEO=> open, --Port is left unconnected.
71                               Q => Q3);
72 -----
73  Q_DIGIT <= Q0 when F(9 downto 8)="00" else      --Select Data Digit No.0
74      Q1 when F(9 downto 8)="01" else      --Select Data Digit No.1
75      Q2 when F(9 downto 8)="10" else      --Select Data Digit No.2
76      Q3;   -- F(9 downto 8)="11"        --Select Data Digit No.3
77 -----
78  COMMON <= "1110" when F(9 downto 8)="00" else --Select Digit No.0
79      "1101" when F(9 downto 8)="01" else --Select Digit No.1
80      "1011" when F(9 downto 8)="10" else --Select Digit No.2
81      "0111"; -- F(9 downto 8)="11"      --Select Digit No.3
82 -----
83 Display_7SEGMENT : DECODER_7SEGMENT port map (A=>Q_DIGIT,Y=>Y);
84 end Behavioral;

```

รูปที่ 2.59 โล็คดาวน์บอร์ดที่สามารถนับขึ้น-ลงและเคลื่อนย้ายค่าเอาต์พุตได้

2.16 การใช้ซอฟต์แวร์ทูลออกแบบวงจรดิจิตอลโดยใช้ CPLD

ตัวอย่างนี้จะอธิบายการใช้ซอฟต์แวร์ทูล ISE WebPACK 8.1i ออกแบบวงจรดิจิตอลด้วย CPLD โดยใช้ภาษา VHDL แล้วให้นำไฟล์ที่ได้ไปดาวน์โหลดลงบนบอร์ด CPLD Explorer XC9572XL (หรือ CPLD Explorer XC9572) ส่วนคนที่ใช้ OS ที่เป็น Microsoft Vista Business (32-bit) ให้ใช้ซอฟต์แวร์ทูล ISE WebPACK 10.1i ซึ่งลักษณะการใช้งานแตกต่างกันเล็กน้อย

ตัวอย่างที่ 2.35 ให้ออกแบบวงจรบวกขนาด 3 บิตด้วย CPLD โดยมี Entity ชื่อ ex1vcx1 และโล็ค VHDL แสดงดังรูป E1.1

```

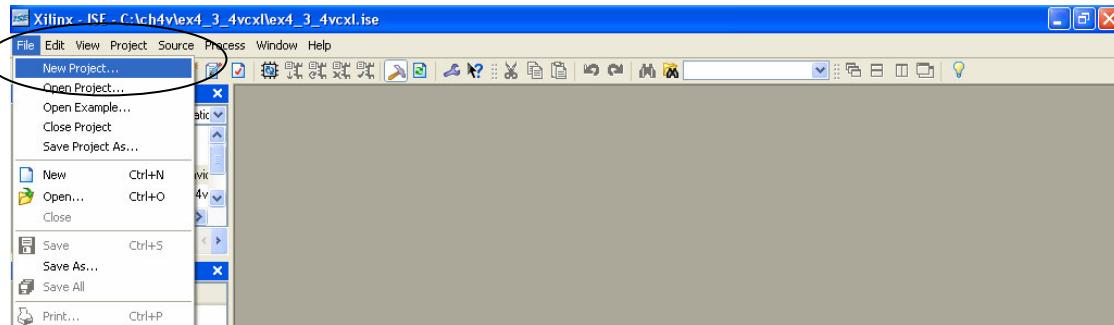
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex1vcx1 is
7     Port ( A,B : in STD_LOGIC_VECTOR (2 downto 0);
8             C : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex1vcx1;
10
11 architecture Behavioral of ex1vcx1 is
12 begin
13
14     C <= ('0' &A) + ('0' &B);
15
16 end Behavioral;

```

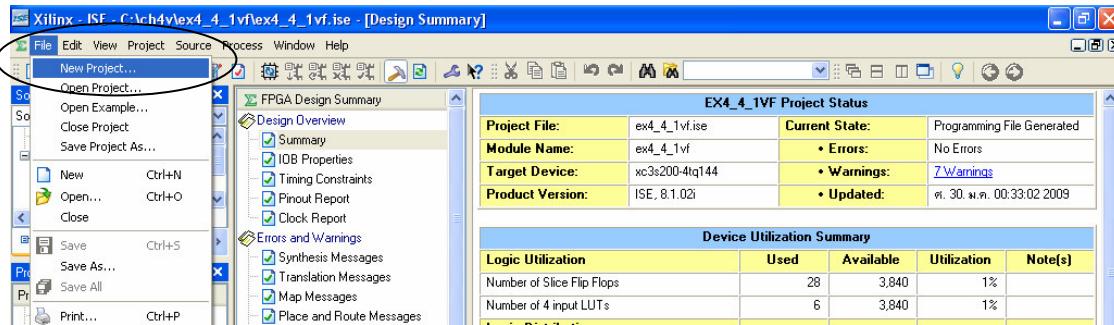
รูปที่ E1.1 โล็ควงจรบวก 3 บิต

2.16.1 ขั้นตอนการออกแบบวงจร (Design entry)

- ก่อนใช้ ISE WebPACK 8.1i ให้ตั้งความละเอียดภาพที่ 1024 x 768 pixels คอมพิวเตอร์ควรมี RAM ไม่น้อยกว่า 512 MB และการปิดโปรแกรมอื่นๆ ที่ไม่เกี่ยวข้องทั้งหมดเพื่อป้องกันความผิดพลาดเนื่องจาก RAM เพียงไม่พอ เพื่อความสะดวกผู้ใช้แนะนำให้มี RAM ไม่น้อยกว่า 1-2 GB และความพื้นที่ว่างในฮาร์ดดิสก์ประมาณไม่น้อยกว่า 10 GB ส่วนคนที่ใช้ซอฟต์แวร์ทูล ISE WebPACK 10.1i นั้นควรมีพื้นที่ว่างในฮาร์ดดิสก์ประมาณไม่น้อยกว่า 15-20 GB จากนั้นจึงสร้าง Folder ชื่อ ch2v (หรือชื่ออื่น) ไว้ในไดร์ฟ C เสร็จแล้วรีเซ็ตคอมพิวเตอร์โดยคลิกปุ่ม Start -> Programs -> Xilinx ISE 8.1i -> Project Navigator หรือดับเบิลคลิกที่ แล้วจะได้หน้าต่าง Xilinx-ISE (ถ้ามีหน้าต่าง Tip of the Day ข้อนี้มาให้คลิก OK) คลิกที่ File -> New Project ดังรูปที่ E1.2a หรือรูปที่ E1.2b) แล้วจะได้หน้าต่าง (หรือ Dialog box) New Project Wizard-Create New Project



E1.2a) หน้าต่าง Xilinx-ISE

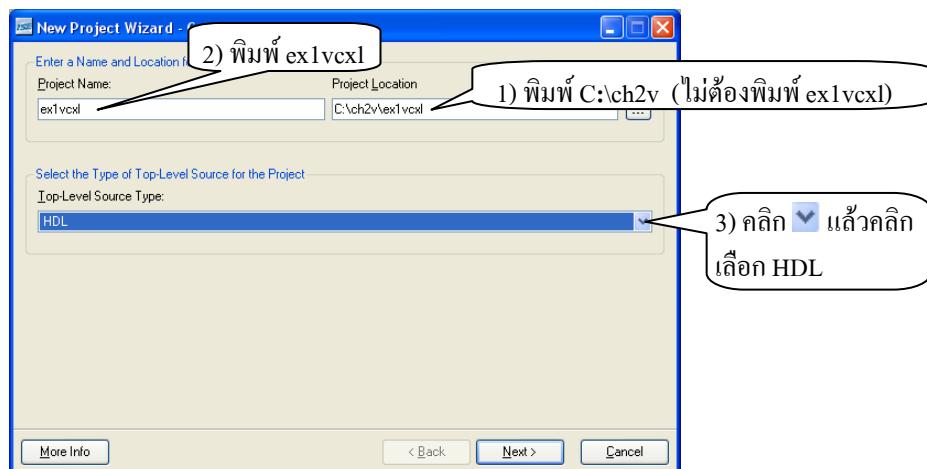


E1.2b) หน้าต่าง Xilinx-ISE

รูปที่ E1.2 หน้าต่าง Xilinx-ISE ที่ความละเอียดของภาพ 1024 x 768 pixels อาจจะเป็นดังรูป E1.2a) หรือ E1.2b)

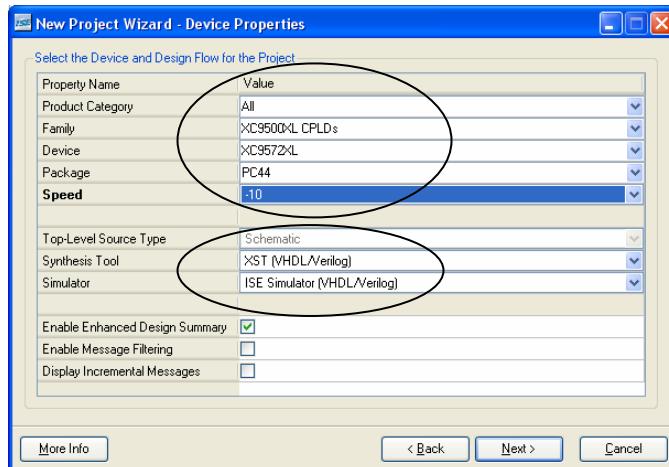
2) ที่หน้าต่าง New Project Wizard-Create New Project สร้างโปรเจกต์ไฟล์ (Project File) ใหม่โดยพิมพ์ชื่อ ch2v (ชื่อ Folder) ลงในช่องว่างของ Project Location ก่อน แล้วจึงพิมพ์ชื่อ ex1vcxl ลงในช่องว่างของ Project Name คลิกที่ Top-Level Source Type เป็น HDL แล้วจะได้ดังรูปที่ E1.3 คลิก Next แล้วจะได้หน้าต่าง New Project Wizard-Device Properties

หมายเหตุ การตั้งชื่อจะใช้กฎเกณฑ์ที่กำหนดในภาษา VHDL ในข้อ 2.4 คือ จะต้องเป็นตัวอักษร A-Z, a-z และตัวถูกไปอาจเป็น A-Z, a-z, 0-9 หรือ Underscores (_) แต่ห้ามจบด้วย (_) และห้ามเว้นช่องว่างระหว่างตัวอักษร

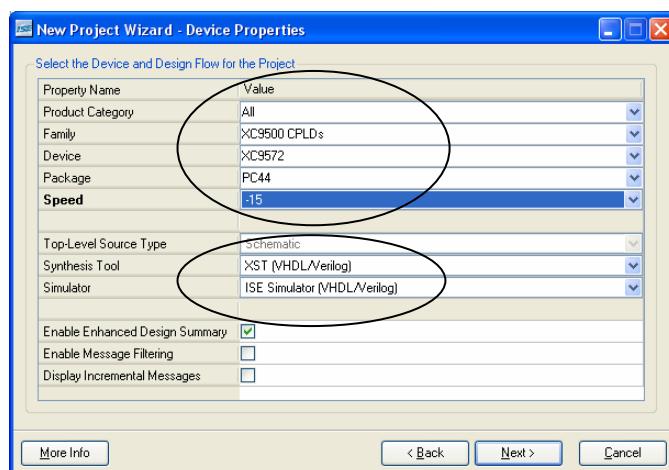


รูปที่ E1.3 หน้าต่าง New Project Wizard-Create New Project

3) ในการที่ใช้บอร์ด CPLD Explorer XC9572XL ให้คลิกดังรูปที่ E1.4 คือ CPLD ตระกูล (Family) XC9500XL, เบอร์ (Device) XC9572XL, Package แบบ PLCC 44 ชิ้น (Package:PC44) และ Speed Grade :-10 แต่ถ้าใช้บอร์ด CPLD Explorer XC9572 ให้คลิกดังรูปที่ E1.5 และเลือกที่ Synthesis tool เป็น XST (VHDL/Verilog) และ Simulator เป็น ISE Simulator (VHDL/Verilog)

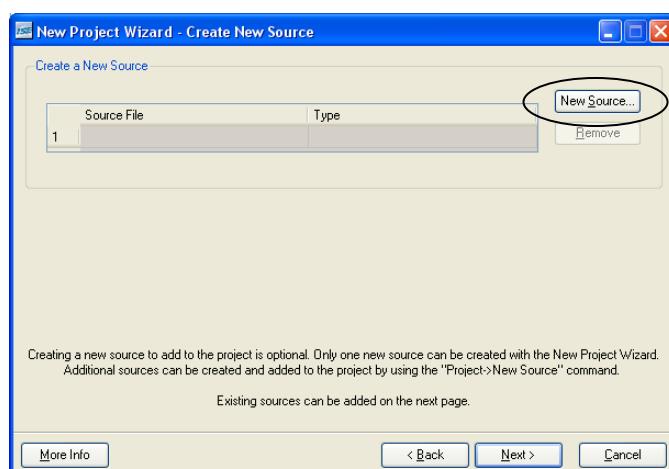


รูปที่ E1.4 หน้าต่าง New Project Wizard–Device Properties ในกรณีที่ใช้บอร์ด CPLD Explorer XC9572XL

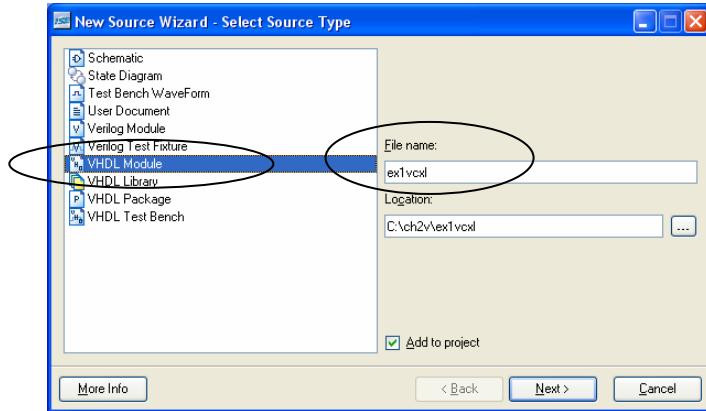


รูปที่ E1.5 หน้าต่าง New Project Wizard–Device Properties ในกรณีที่ใช้บอร์ด CPLD Explorer XC9572

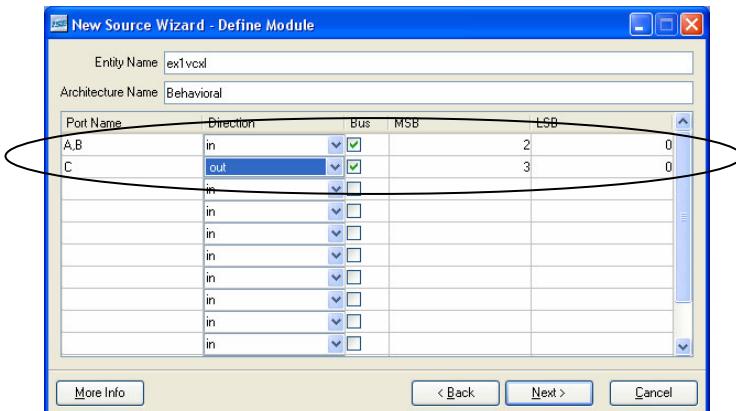
4) คลิก Next ในรูปที่ E1.4 (หรือในรูปที่ E1.5) และจะได้หน้าต่างดังรูปที่ E1.6 คลิกปุ่ม New Source และจะได้หน้าต่างกดไปจากนั้นพิมพ์ชื่อ Source File ชื่อ ex1vcx1 ลงในช่อง File Name และคลิกที่ VHDL Module ดังรูปที่ E1.7 คลิก Next จะได้หน้าต่างดังรูปที่ E1.8 เมื่อกำหนดอินพุต A,B และเอาต์พุต C เรียบร้อยแล้วคลิก Next 1 ครั้ง คลิก Finish 1 ครั้ง แล้วคลิก Next อีก 1 ครั้งก็จะได้หน้าต่าง New Project Wizard–Add Existing Source ดังรูปที่ E1.9 (ซึ่งจะอธิบายในภายหลัง)



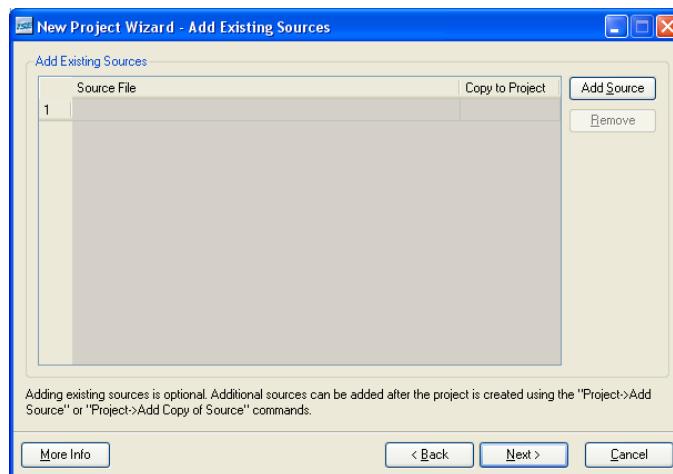
รูปที่ E1.6 หน้าต่าง New Project Wizard–Create New Source



รูปที่ E1.7 หน้าต่าง New Source Wizard–Select Source Type



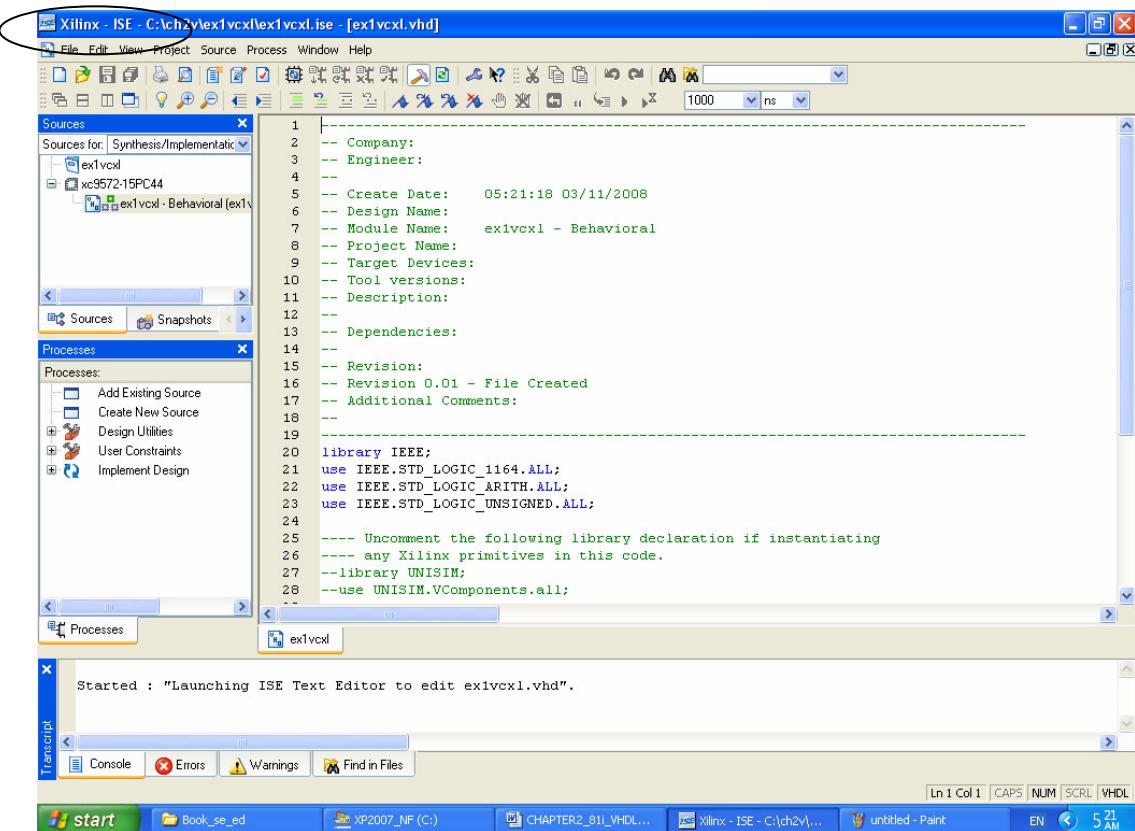
รูปที่ E1.8 หน้าต่าง New Source Wizard–Define Module ซึ่งกรณีเป็นบล๊อกคลิก “√” ที่คอลัมน์ Bus ด้วย



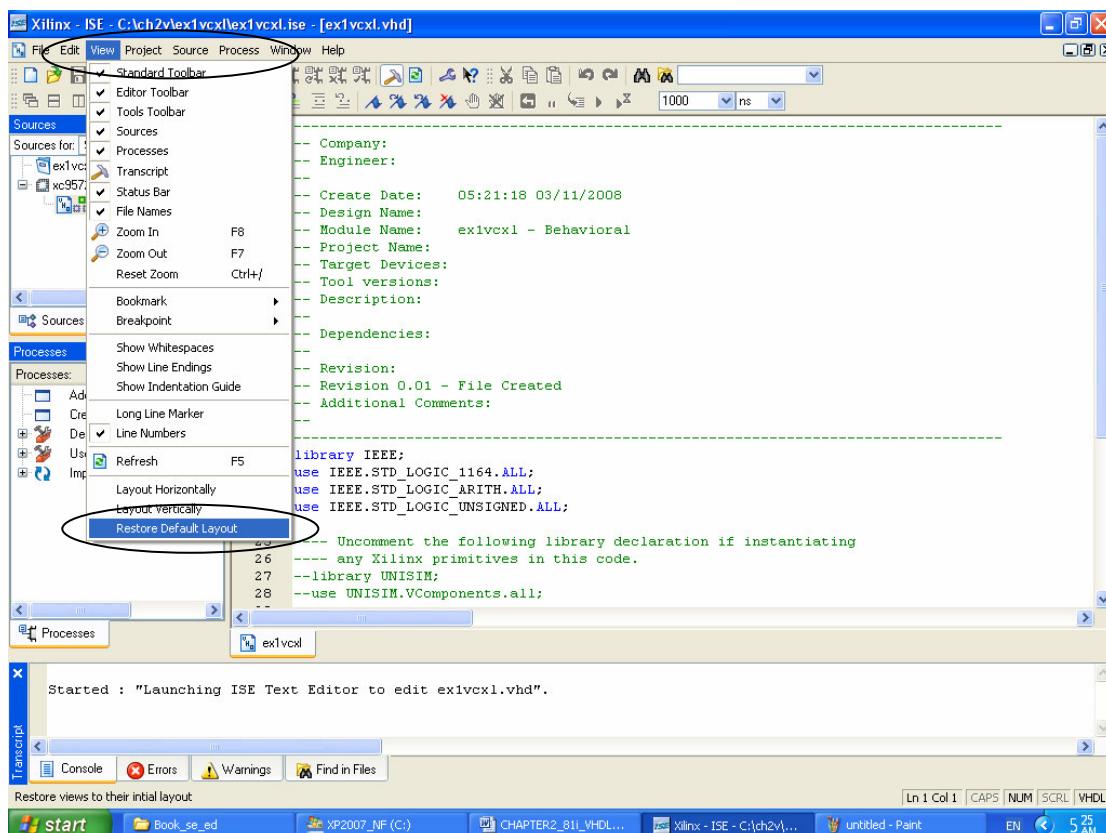
รูปที่ E1.9 หน้าต่าง New Project Wizard-Add Existing Source

5) จากรูปที่ E1.9 คลิก Next 1 ครั้ง คลิก Finish อีก 1 ครั้งแล้วจะได้หน้าต่าง Xilinx–ISE สำหรับเขียนโค้ด VHDL (โปรแกรม Text Editor) ดังรูปที่ E1.10

6) ที่หน้าต่าง Xilinx–ISE ดังรูปที่ E1.10 ให้ผู้อ่านทำความคุ้นเคยกับหน้าต่างนี้ โดยคลิก View ดังรูปที่ E1.11 แล้วคลิกที่แถบข้อความต่างๆ ดูพร้อมกับสังเกตการเปลี่ยนแปลงในส่วนของหน้าต่างย่อยต่างๆ ว่ามีหน้าต่างใดหายไปบ้าง และถ้าต้องการให้หน้าต่างต่างๆ กลับมา มีลักษณะเหมือนเดิมให้คลิกที่แถบล่างสุดคือ Restore Default Layout

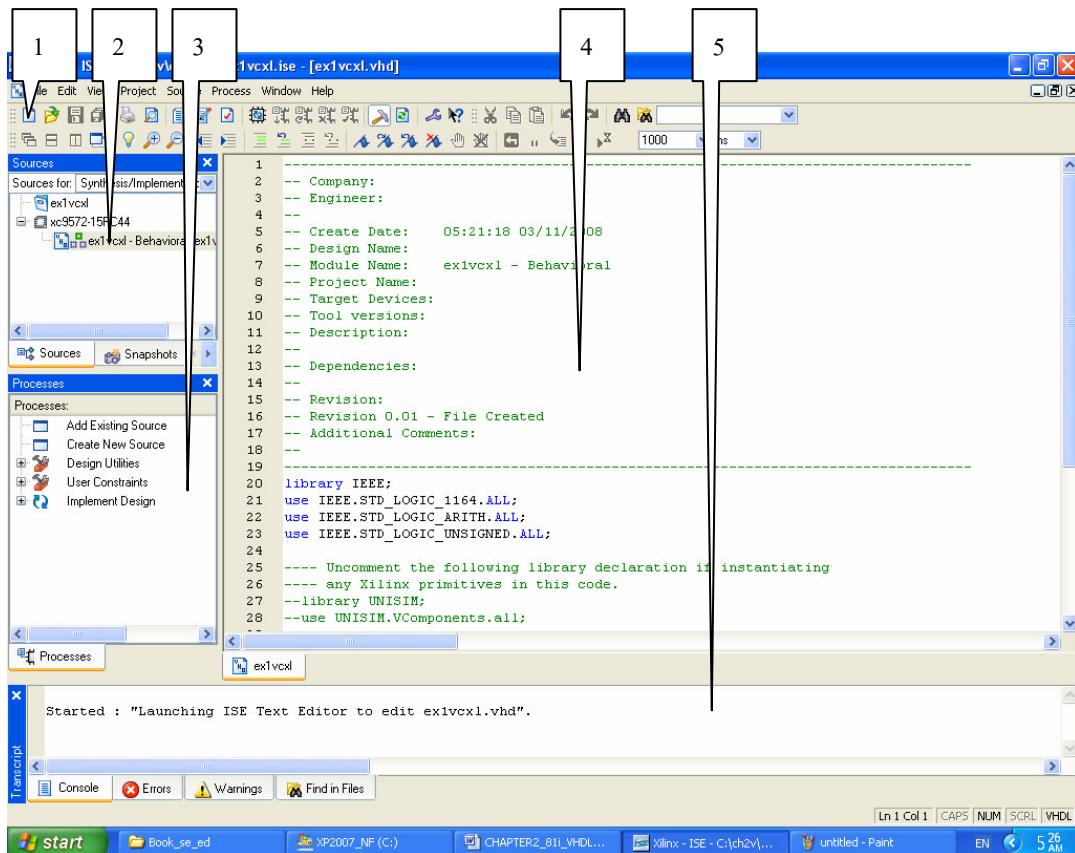


รูปที่ E1.10 หน้าต่าง Xilinx-ISE สำหรับเขียนโค้ด VHDL (โปรแกรม Text Editor)



รูปที่ E1.11 หน้าต่างการใช้ View

7) ก่อนทำขั้นตอนต่อไปควรทำความสะอาดหน้าต่างหลัก Xilinx-ISE (Project Navigator main window) ดังรูปที่ E1.12



รูปที่ E1.12 หน้าต่างหลัก Project Navigator (ความละเอียดของภาพ 1024 x 768 pixels)

หน้าต่างหลัก Project Navigator ดังรูปที่ E1.12 จะประกอบด้วยหมายเลขดังต่อไปนี้

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

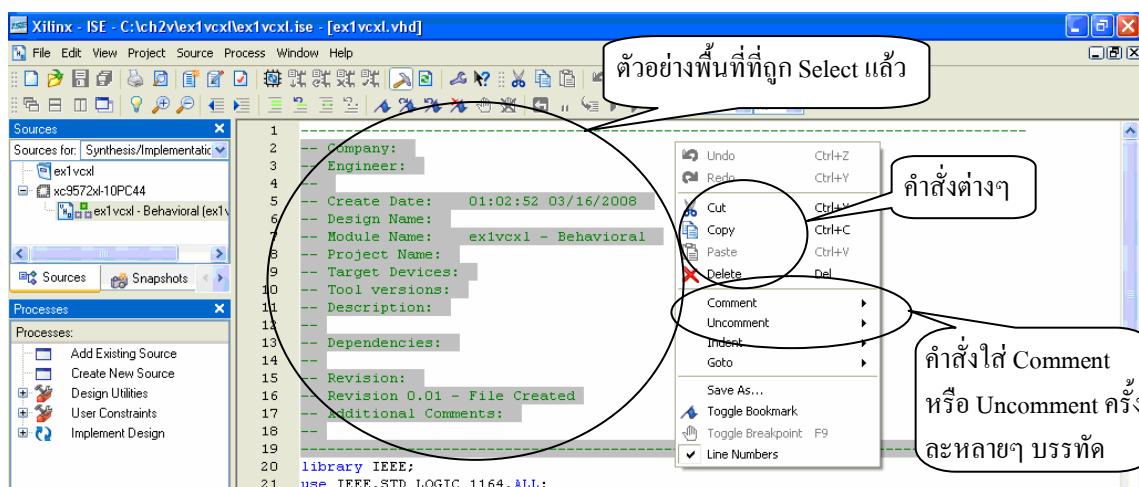
Standard toolbar จะแทนเป็นคำสั่งที่ถูกใช้บ่อย ซึ่งสัญลักษณ์บางส่วนเราคุ้นเคยกันดีอยู่แล้ว มีดังนี้คือ

	New		Implement Top Module
	Open		Run Process
	Save		Rerun All Process
	Save All		Stop Process
	Print		Edit Process Properties
	Print Preview		Toggle Transcript
	New Source		Refresh
	Open Source		Support and Services
	Edit Project Properties		Help

Editor toolbar จะเป็น Edit menu commands ที่ใช้บ่อย ซึ่งสัญลักษณ์ที่ใช้ส่วนใหญ่เราคุ้นเคยกันดีอยู่แล้ว มีดังนี้คือ

	Cut
	Copy
	Paste
	Undo
	Redo
	Find
	Find in Files

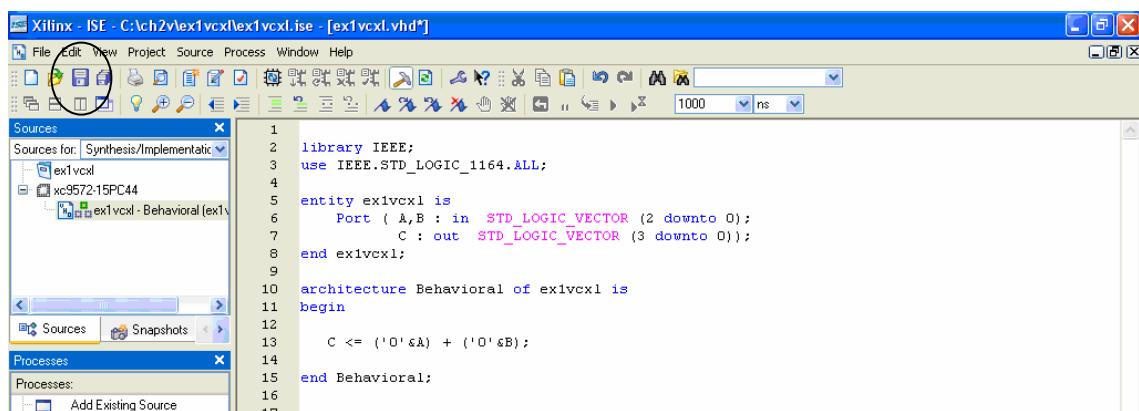
8) ให้ลองคลิกขวาที่หน้าต่างหลัก (Workspace) ก็จะพบว่ามีคำสั่งต่างๆ ที่เราคุ้นเคยกันดีอยู่แล้วดังในรูปที่ E1.13



รูปที่ E1.13 หน้าต่างหลัก (Workspace)

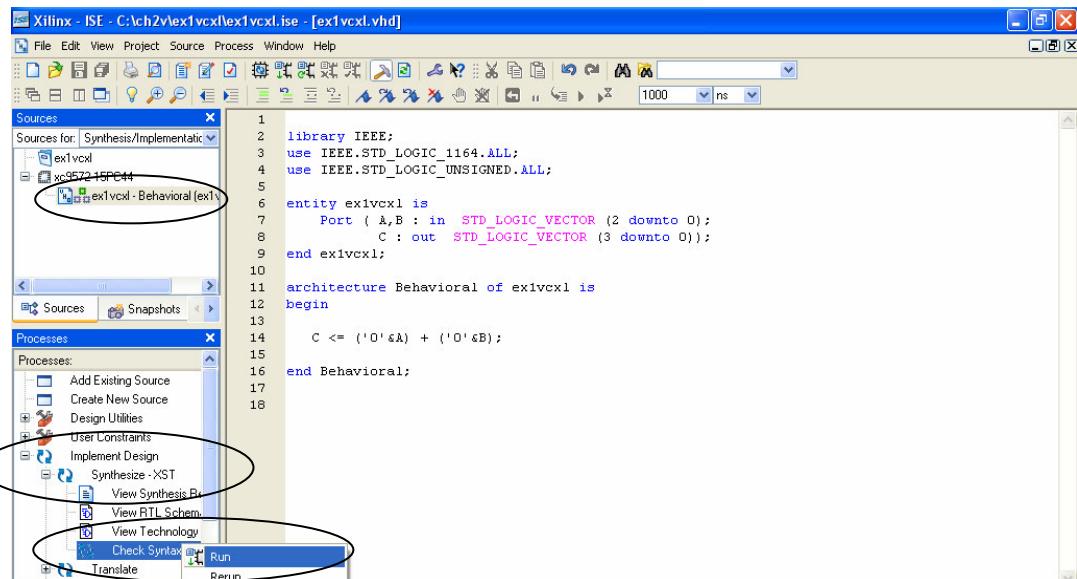
หมายเหตุ ในรูปที่ E1.13 แสดงการใส่ Comment (หรือ “-”) คลายๆ บรรทัดได้โดยเลือก (select) บริเวณที่จะการใส่ Comment ด้วยการคลิกเมาส์ก้างไว้แล้วเลื่อนเมาส์ลงมาคลายๆ บรรทัดตามที่ต้องการ เมื่อปล่อยเมาส์ก็จะได้พื้นที่ที่จะการใส่ Comment จากนั้นคลิกขวาบนพื้นที่ว่างแล้วคลิก Comment -> Line(s) หรือถ้าต้องการยกเลิก Comment แบบคลายๆ บรรทัดสามารถทำได้โดยเลือก (select) บริเวณที่จะการยกเลิก Comment แล้วคลิก Uncomment -> Line(s)

9) ทำการปรับปรุงแก้ไข โค๊ดให้เป็นตามรูปที่ E1.1 ซึ่งจะเห็นว่าซอฟต์แวร์ทูลเป็นคนจัดการให้เราเก็บทั้งหมด โดยที่เราจะเขียนเฉพาะบรรทัดที่ 14 เพียงบรรทัดเดียวเท่านั้น เมื่อเสร็จแล้วจะได้ตั้งรูปที่ E1.14 แล้วคลิก บันทึกไฟล์

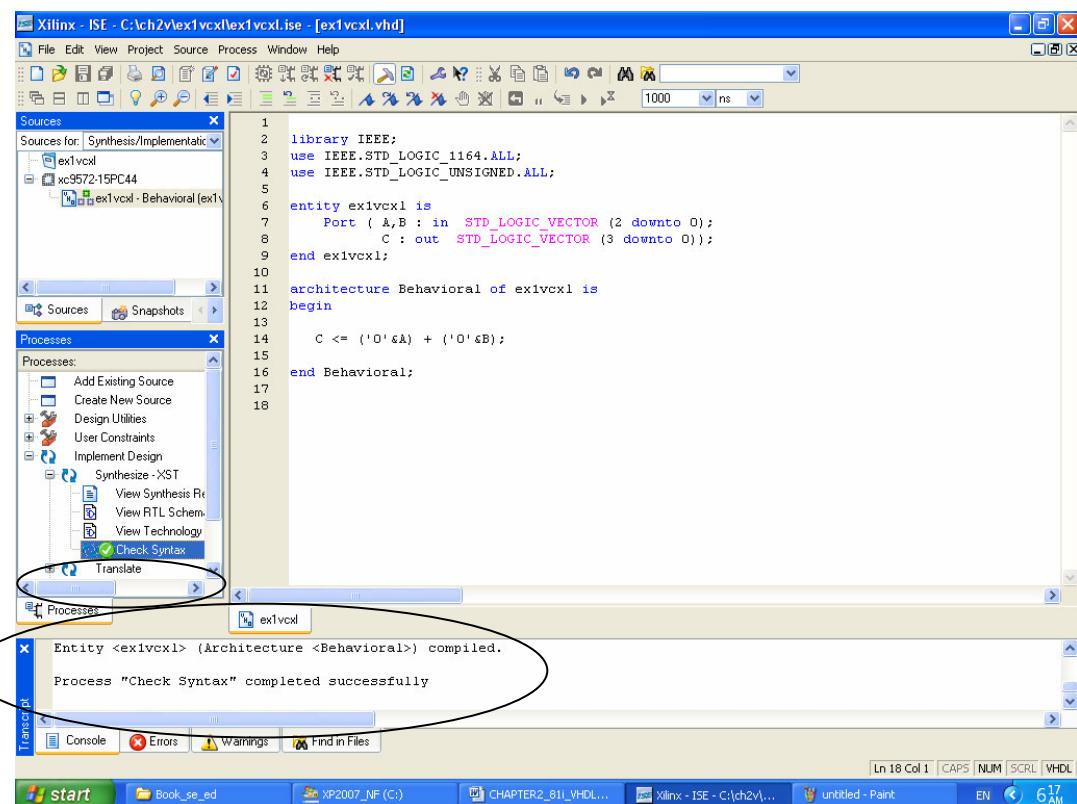


รูปที่ E1.14 โค๊ด VHDL ของวงจรนาฬิกา 3 บิตที่เขียนเสร็จเรียบร้อยแล้ว

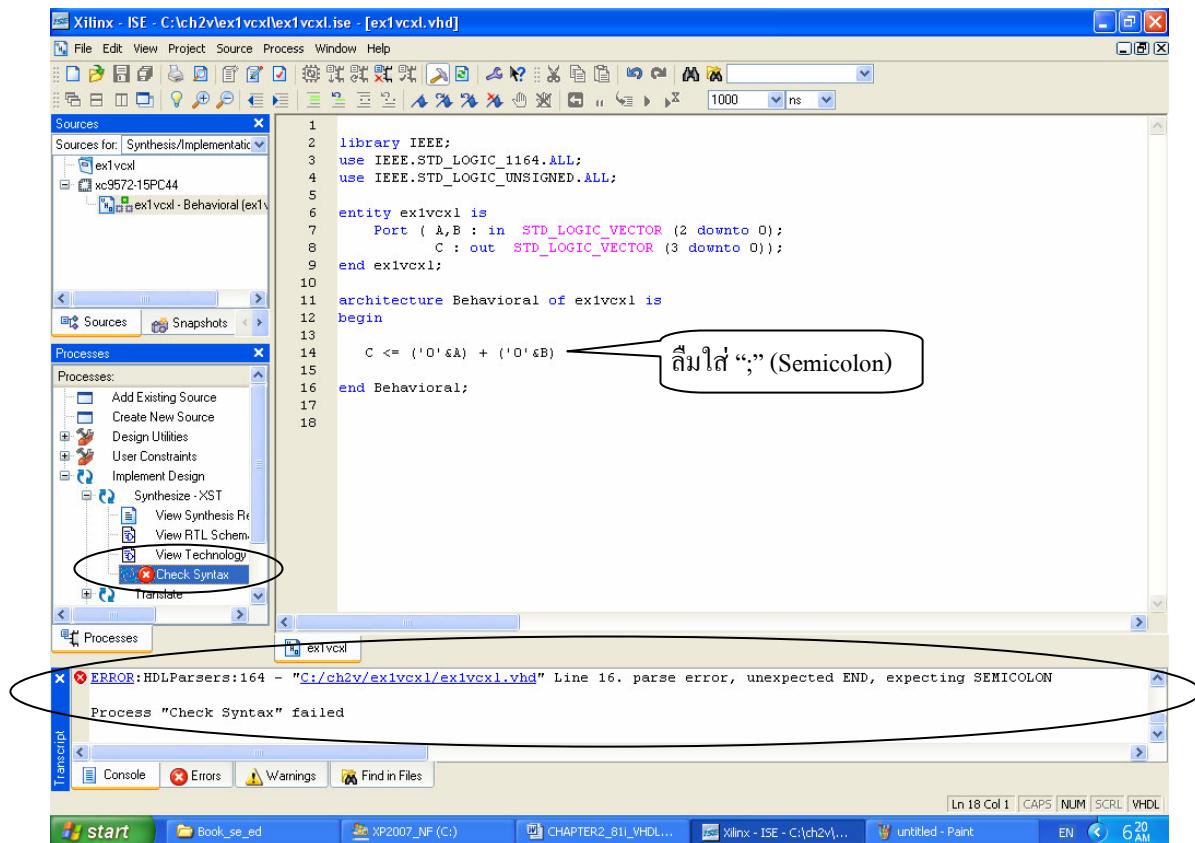
10) การตรวจความถูกต้อง (Syntax) ของโค้ด คลิกที่ชื่อไฟล์ ex1vcx1 ในหน้าต่าง Source แล้วคลิกที่ “+” หน้า Implement design และหน้า Synthesize-XST ในหน้าต่าง Process จะเป็น “-” คลิกขวาที่ Check Syntax แล้วคลิก Run ดังรูปที่ E1.15 ถ้าไม่มีข้อผิดพลาดก็จะปรากฏข้อความในหน้าต่าง Transcript ว่า Process “Check Syntax” completed successfully ดังรูปที่ E1.16 ขั้นตอนนี้ถือว่าไฟล์ได้ถูก Compiled เรียบร้อยแล้ว แต่ถ้ามีข้อผิดพลาดเข่น ในรูปที่ E1.17 บอกว่า Line 16. parse error, unexpected END, expecting SEMICOLON ซึ่งลีมใส่ “;” (Semicolon) ตอนจบบรรทัดที่ 14 เมื่อแก้ไขแล้วให้คลิก บันทึกไฟล์ และให้ตรวจความถูกต้องซ้ำอีกรัง ถ้าไม่มีข้อผิดพลาดก็ถือว่าขั้นตอน Design entry นี้เสร็จสมบูรณ์ คลิก X (สีดำ) เพื่อปิดโปรแกรม Text Editor และกลับไปที่หน้าต่าง Xilinx-ISE ดังรูปที่ E1.18 หรือคลิก X (สีแดง) เพื่ออกจากโปรแกรม ISE WebPACK 8.1i



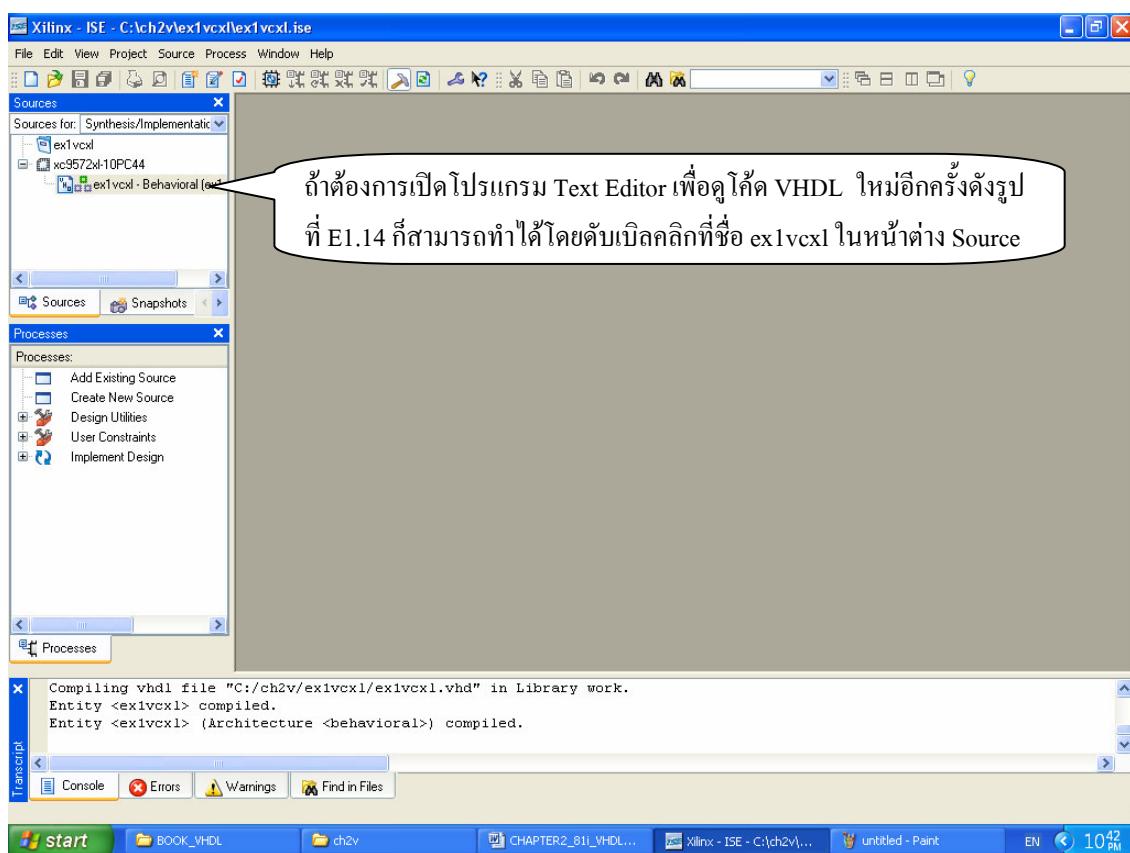
รูปที่ E1.15 การตรวจสอบความถูกต้อง



รูปที่ E1.16 หน้าต่าง Xilinx-ISE เมื่อ Check syntax ผ่าน



รูปที่ E1.17 การเขียนโค้ดที่มีข้อผิดพลาด



รูปที่ E1.18 หน้าต่าง Xilinx-ISE

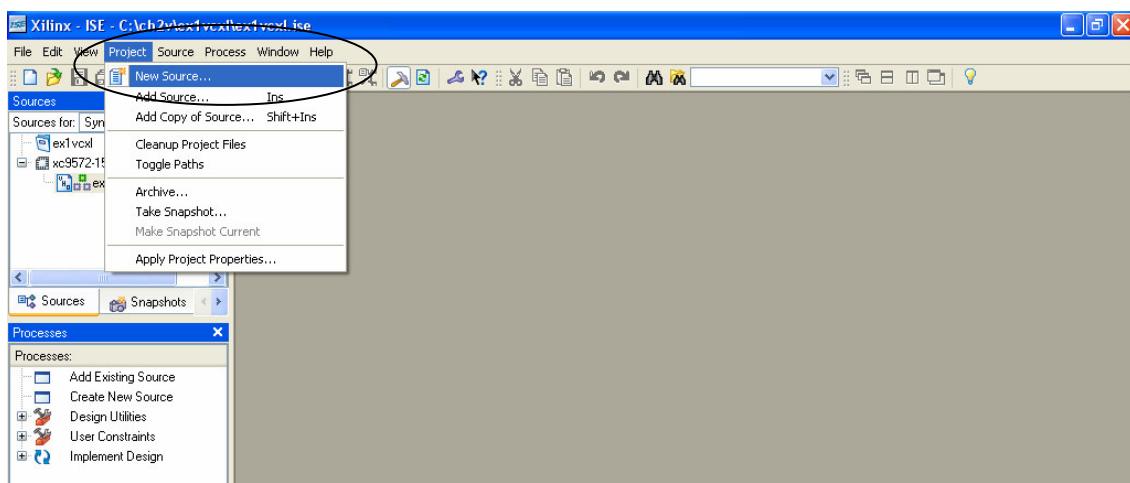
2.16.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification)

การตรวจสอบความถูกต้องของวงจรที่ออกแบบนั้นสามารถทำได้โดยการนำโค้ด VHDL ของวงจรที่ได้ไปจำลองการทำงาน (Simulation) ซึ่งการจำลองการทำงานในเบื้องต้นที่อธิบายในข้อนี้จะใช้ Waveform Editor ส่วนการจำลองการทำงานที่ซับซ้อนนั้นจะเป็นต้องเขียน Testbench โดยจะอธิบายในบทที่ 6 ซึ่งการจำลองสภาพพฤติกรรม (Behavioral simulation) ของวงจรนั้นจะไม่คำนึงผลของการล่าช้า (Delay time) ต่างๆ ที่เกิดขึ้น โค้ดที่เขียนนี้อาจนำไปสังเคราะห์งงาน (Synthesis) ได้หรือไม่ได้ก็ได้ แต่สามารถเขียนโค้ดได้รวดเร็ว

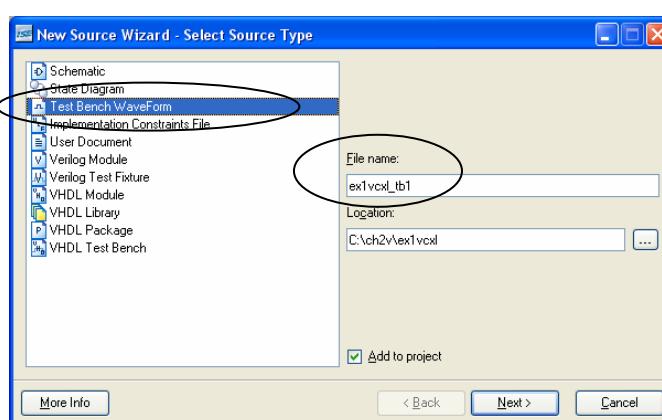
การออกแบบวงจรที่ไม่มีความซับซ้อนนัก ไม่จำเป็นต้องทำขั้นตอนการตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification) ก็ได้ ผู้อ่านสามารถข้ามไปท่าหัวข้อ 2.16.3 ซึ่งเป็นขั้นตอนการสังเคราะห์งงาน (Design synthesis) ได้เลย

หลังจากที่เราเขียนโค้ด VHDL และตรวจสอบความถูกต้องของโค้ด (Syntax) เรียบร้อยแล้ว ขั้นตอนจำลองเฉพาะพฤติกรรม (Behavioral simulation) ของวงจรโดยใช้ Waveform Editor สามารถทำได้ดังนี้

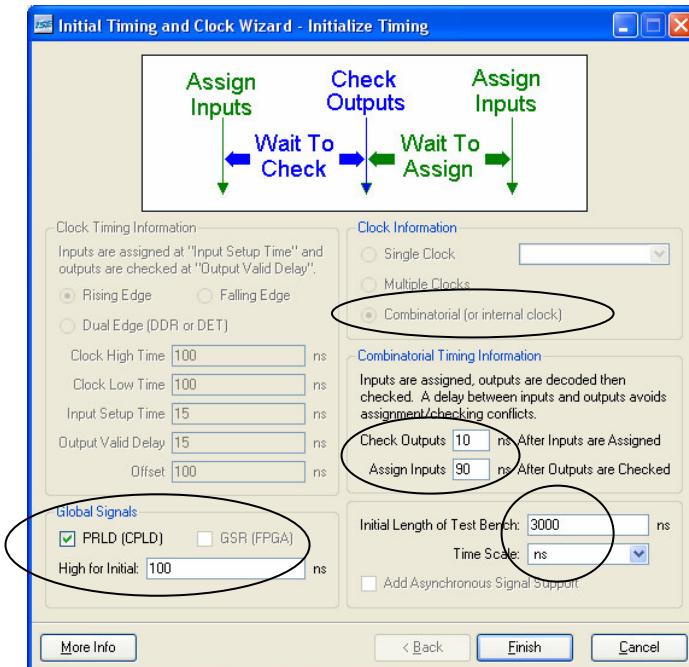
- คลิก Project -> New Source ดังรูปที่ E1.19 แล้วจะได้หน้าต่าง New Source Wizard-Select Source Type พิมพ์ชื่อ ex1vcxl_tb1 (หากตั้งชื่อไปช้ากับชื่อ ex1vcxl) และคลิกที่ Test Bench Waveform ดังรูปที่ E1.20 จากนั้นคลิก Next 2 ครั้งและคลิก Finish ไปอีก 1 ครั้ง จากนั้นเลือกและกำหนดค่าต่างๆ ดังรูปที่ E1.21



รูปที่ E1.19 หน้าต่าง Xilinx-ISE



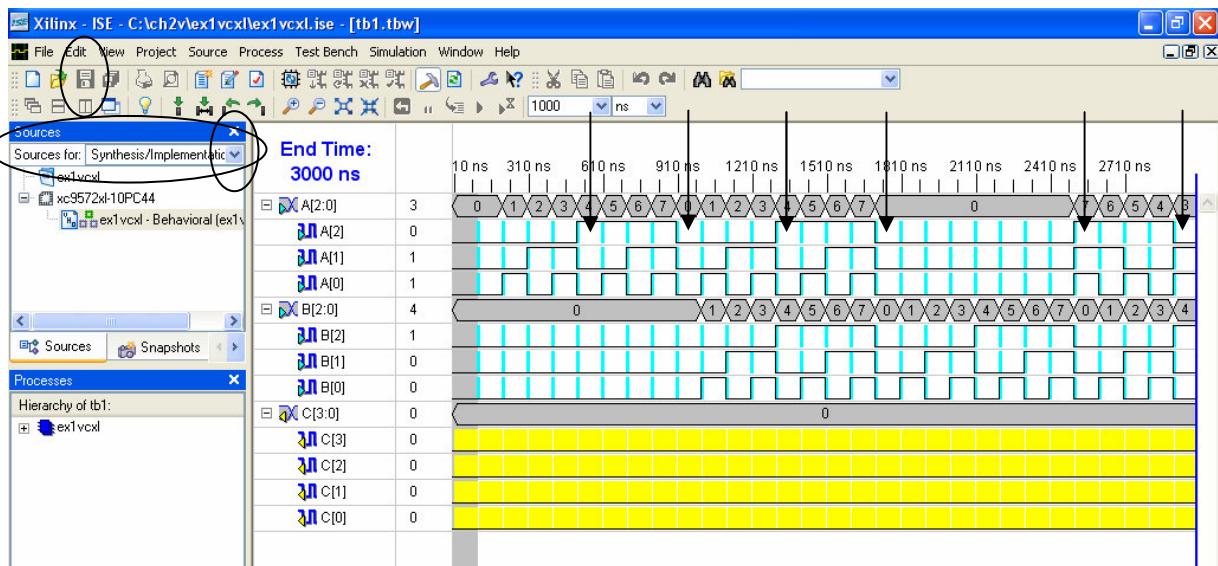
รูปที่ E1.20 หน้าต่าง New Source Wizard-Select Source Type



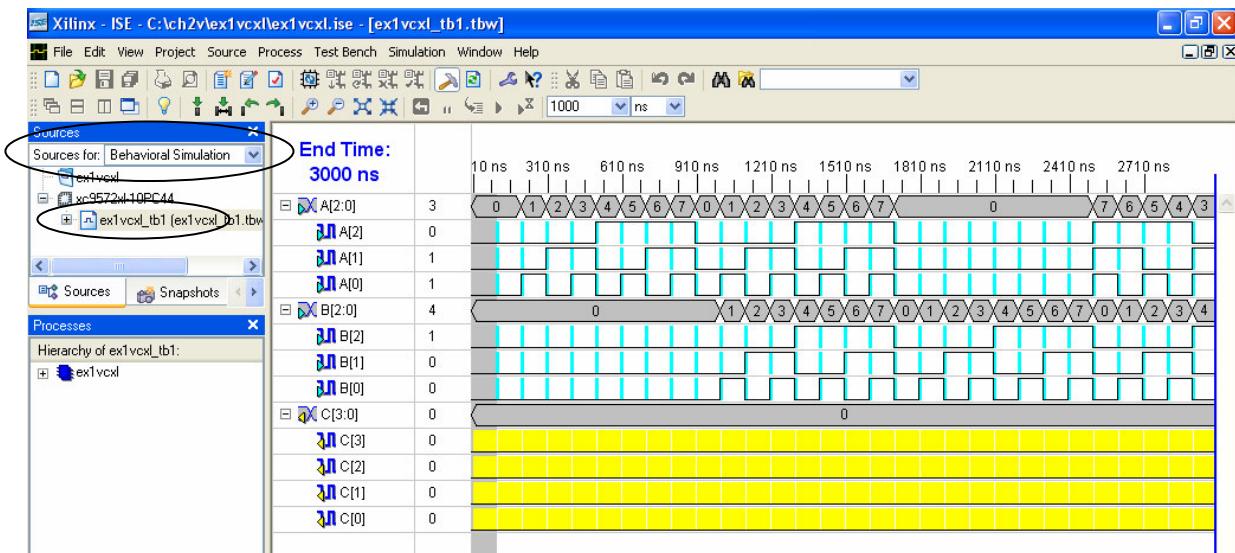
รูปที่ E1.21 หน้าต่าง Initial Timing and Clock Wizard

หมายเหตุ ในทางปฏิบัตินั้นเมื่อเราป้อนอินพุตให้กับวงจรต่างๆ แล้วจะไม่ได้อาดัพุทธันที เวลาที่รอไปจนกระทั่งได้อาดัพุต ออกมาก็เรียกว่า Propagation delay time ซึ่งเป็นเวลาล่าช้าที่เกิดในวงจร ในรูปที่ E1.21 นั้นเรารسمมุติให้เวลาล่าช้านี้เท่ากับ 10 ns

2) ทำการป้อนอินพุต A[2:0] และ B[2:0] เช่น การป้อนอินพุต A[2:0] ก็ให้เริ่มป้อนลักษณะรูปคลื่น (Waveform) โดยคลิกเครื่องหมาย “+” หน้า A[2:0] จนเป็น “-” แล้วจะปรากฏแผล A[2], A[1] และ A[0] จากนั้นจึงป้อนลักษณะรูปคลื่นในแต่ละแผล จาก A[0], A[1] และ A[2] ตามลำดับ เช่น ถ้าป้อน A[2] ก็สามารถทำได้โดยคลิกที่พื้นที่สีขาวที่อยู่บริเวณระหว่างແຄນสีฟ้าอ่อน (คลิกตรงช่องที่ปลายลูกศรชี้) จากนั้นจึงป้อน B[0], B[1] และ B[2] ตามลำดับ เมื่อป้อนลักษณะรูปคลื่นทั้งหมดเสร็จแล้วจะได้ดังรูปที่ E1.22 และวิธีคลิก บันทึกไฟล์ จากนั้นคลิก แล้วคลิกที่ Behavior Simulation เพื่อเพิ่มไฟล์ ex1vcxl_tb1 เข้าไปโดยอัตโนมัติดังรูปที่ E1.23 คลิก (สีดำ) เพื่อปิดโปรแกรมและกลับไปที่หน้าต่าง Xilinx-ISE (ครูปที่ E1.24)

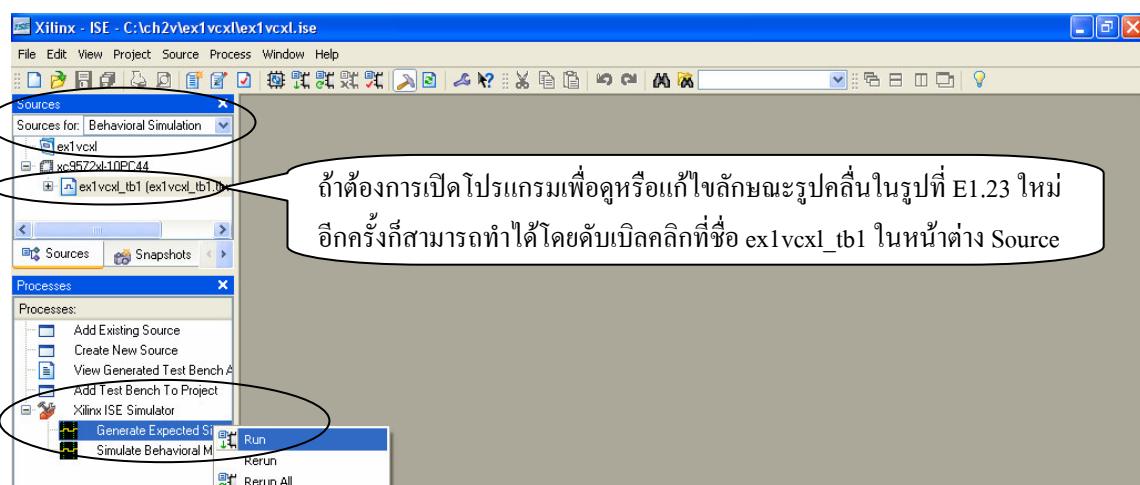


รูปที่ E1.22 หน้าต่างสำหรับกำหนดสัญญาณต่างๆ ที่ป้อนให้กับอินพุตของวงจรที่เราต้องการทดสอบ

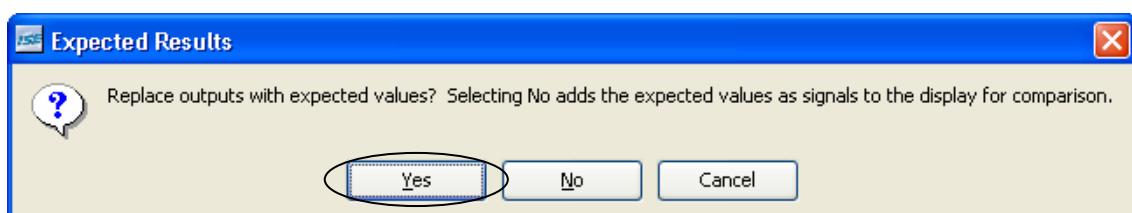


รูปที่ E1.23 หน้าต่างสำหรับกำหนดสัญญาณต่างๆ ที่ป้อนให้กับอินพุตของวงจรที่เราต้องการทดสอบ

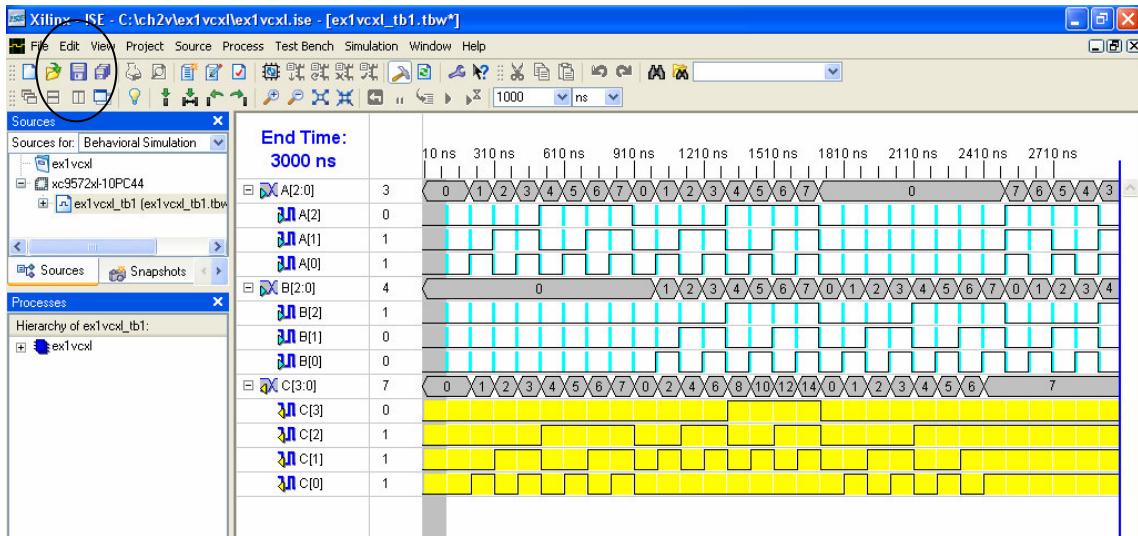
3) ขั้นตอน Create a self-checking test bench เพื่อสร้างสัญญาณเอาต์พุตโดยอัตโนมัติ เริ่มที่หน้าต่าง Xilinx-ISE คลิก แล้ว คลิกที่ Behavior Simulation และคลิกเลือกไฟล์ชื่อ ex1vcxl_tb1 ในหน้าต่าง Source คลิกเครื่องหมาย “+” หน้า Xilinx ISE Simulator จะเป็น “-” คลิกขวาที่ Generate Expected Simulation Results แล้วคลิก Run ในหน้าต่าง Processes ดังรูปที่ E1.24 แล้วจะได้ดังรูปที่ E1.25 เมื่อคลิก Yes แล้วจะได้ดังรูปที่ E1.26 คลิก บันทึกไฟล์ คลิก (สีดำ) เพื่อปิดโปรแกรมและกลับไปที่หน้าต่าง Xilinx-ISE อีกรึ



รูปที่ E1.24 หน้าต่าง Xilinx-ISE

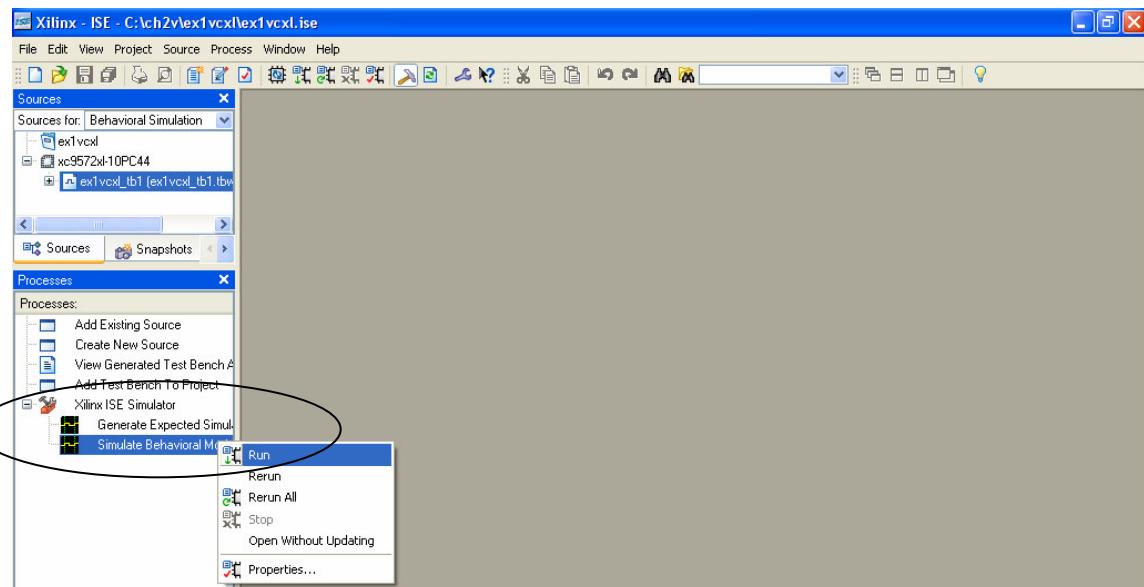


รูปที่ E1.25 หน้าต่าง Expected Results

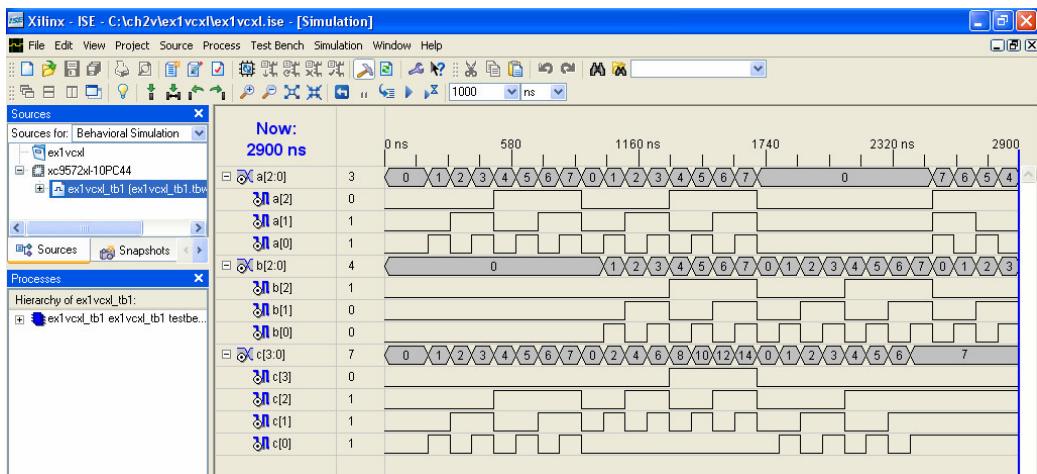


รูปที่ E1.26 ผลที่ได้ในขั้นตอน Create a self-checking test bench

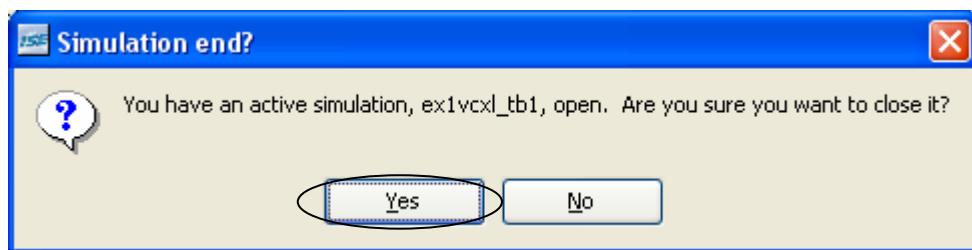
4) ในรูปที่ E1.26 ถ้าได้อาร์ฟตเป็นไปตามที่ออกแบบไว้ก็จะไปทำ Behavioral simulation โดยคลิกขวาที่ Simulate Behavioral Model แล้วคลิก Run ดังรูปที่ E1.27 เสร็จแล้วก็จะแสดงผล Behavioral simulation ดังรูปที่ E1.28 ซึ่งขั้นตอน Simulation นี้จะไม่นำผลของเวลาล่าช้าหรือ Delay time ต่างๆ มาคิด จึงเป็นการตรวจสอบความถูกต้องขั้นต้นเท่านั้น คลิก **X** (สีดำ) ปิดโปรแกรมแล้วคลิก Yes ในรูปที่ E1.29 เพื่อกลับไปที่หน้าต่าง Xilinx-ISE จากนั้นคลิก **✓** แล้วคลิกที่ Synthesis/Implementation ดังรูปที่ E1.30 เพื่อเตรียมพร้อมที่จะทำขั้นตอนต่อไป



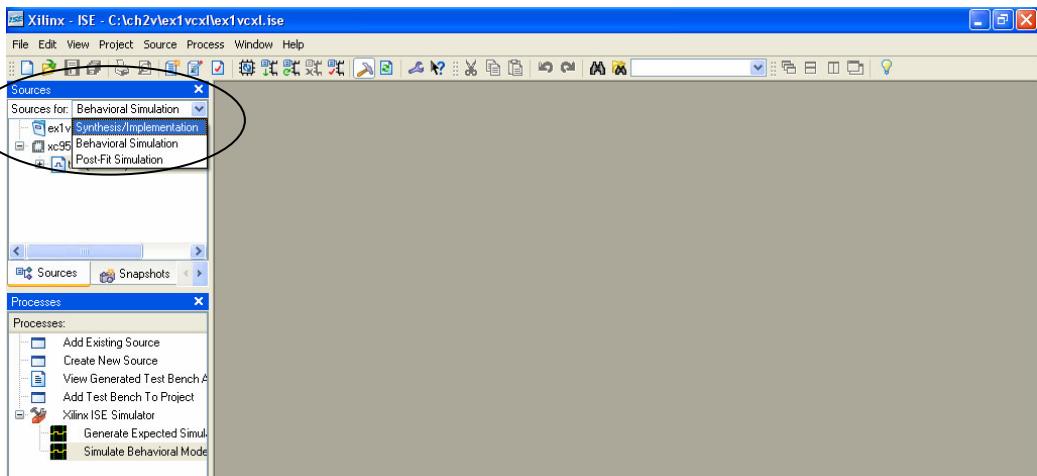
รูปที่ E1.27 ขั้นตอน Behavioral simulation



รูปที่ E1.28 แสดงผล Behavioral simulation ของวงจรบวก 3 บิต (ซึ่งในขั้นตอนนี้จะไม่นำผลของเวลาล่าช้ามาคิด)



รูปที่ E1.29 หน้าต่าง Simulation end

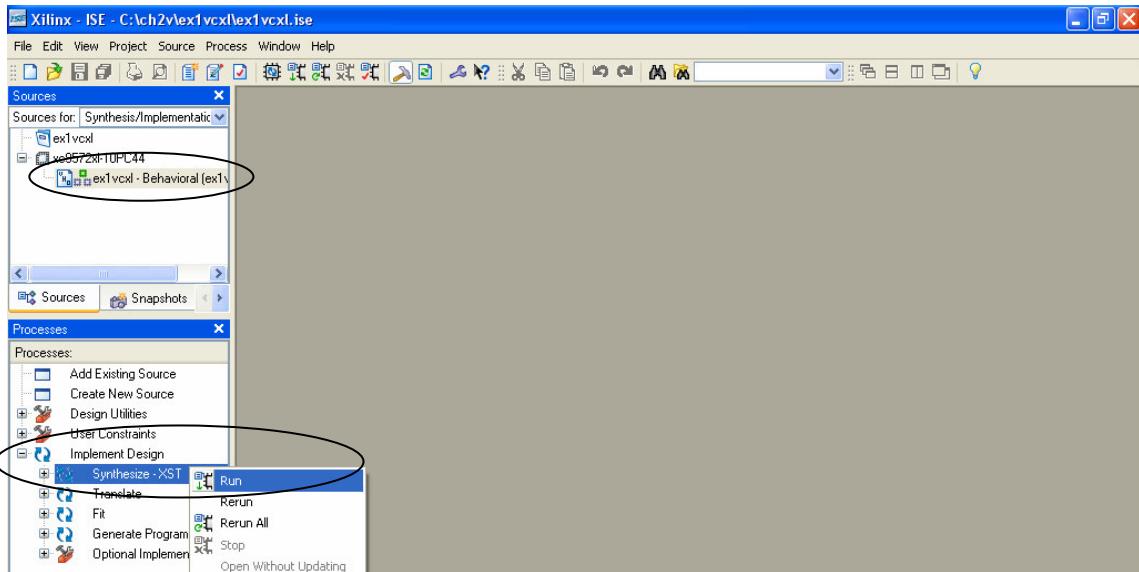


รูปที่ E1.30 หน้าต่าง Xilinx-ISE

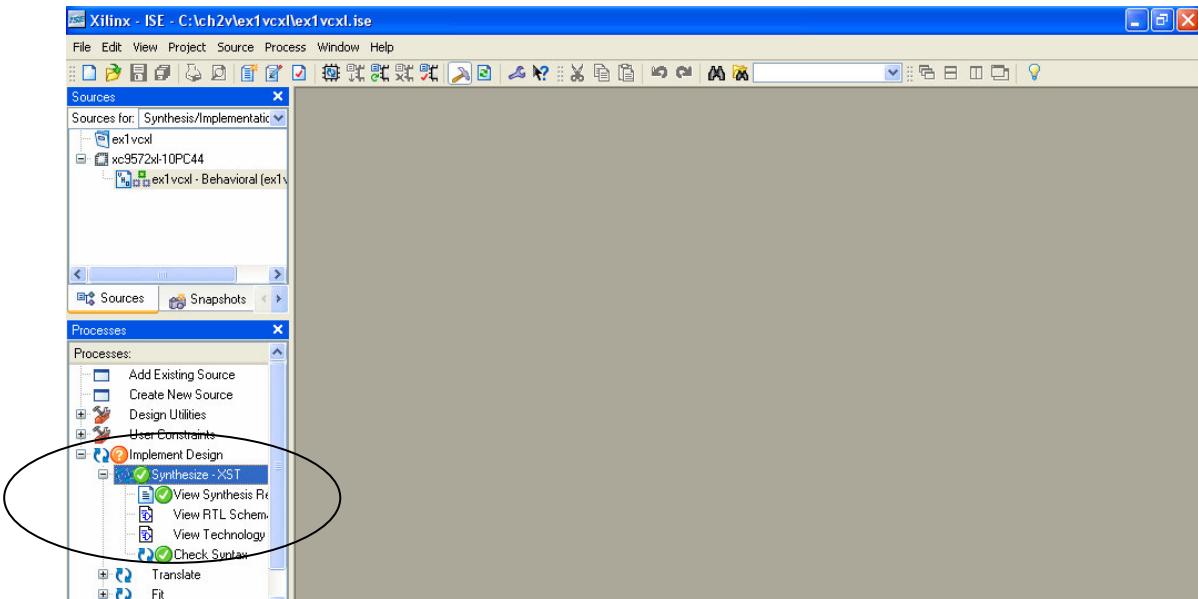
2.16.3 การสังเคราะห์วงจร (Design synthesis)

1) ขั้นตอนสังเคราะห์วงจร ให้คลิกที่ชื่อไฟล์ ex1vcxl ในหน้าต่าง Source และเลื่อนมาส่องไปที่หน้าต่าง Processes คลิก “+” ที่หน้า Implement Design จนเป็น “-” คลิกขวาแล้วคลิก Run ที่ Synthesize-XST ดังรูปที่ E1.31 (หรือดับเบิลคลิก) เสรีจแล้วถ้าได้ หรือ ดังรูปที่ E1.32 จะถือว่าสังเคราะห์วงจรผ่าน ซึ่งถ้าเราไม่ต้องการดูรายละเอียดผลการสังเคราะห์วงจรก็ไม่ต้องทำข้อ 2 และ 3) ที่จะอธิบายในข้อต่อไป ให้ข้ามไปทำหัวข้อ 2.16.4 ได้เลย ซึ่งเป็นการทำขั้นตอน Design Implementation

ความหมายของเครื่องหมายต่างๆ เป็นดังนี้ = Running, = Up-to-date(ถือว่าผ่าน), = Warnings Reported (ถือว่าผ่าน), = Errors Reported (ไม่ผ่าน), = Out-of-Date (ให้ทำขั้นตอนนี้ซ้ำอีกครั้ง)



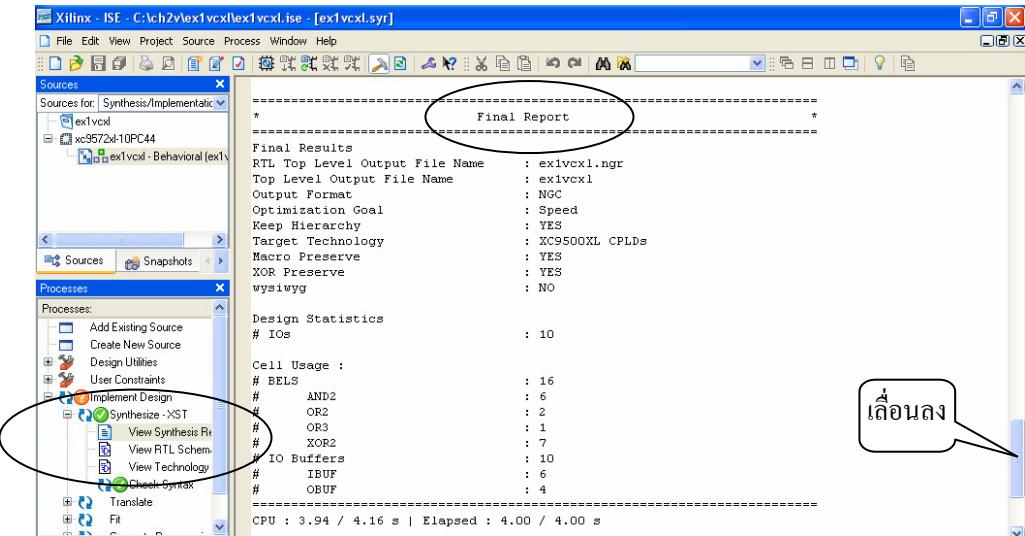
รูปที่ E1.31 ขั้นตอนสังเคราะห์วงจร



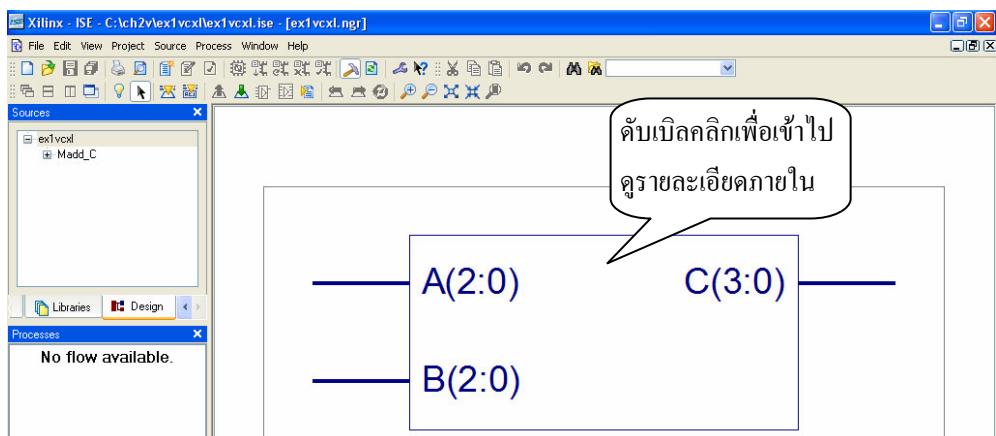
รูปที่ E1.32 หน้าต่าง Processes เมื่อสังเคราะห์วงจรเสร็จเรียบร้อยแล้ว

2) ถ้าต้องการดูรายงานผลการสังเคราะห์วงจร ให้คลิกขวาที่ View Synthesis Report แล้วคลิก Rerun ในรูปที่ E1.32 เสร็จแล้ว ให้เลื่อนมาตรงไปคุณลักษณะที่ Final Reports ดังรูปที่ E1.33 ซึ่งแสดงรายการอุปกรณ์พื้นฐานที่นำมาสร้างวงจรที่เราออกแบบจากนั้นคลิก **X** (สีดำ) ที่มุมบนขวาเพื่อกลับไปที่หน้าต่าง Xilinx-ISE อีกครั้ง

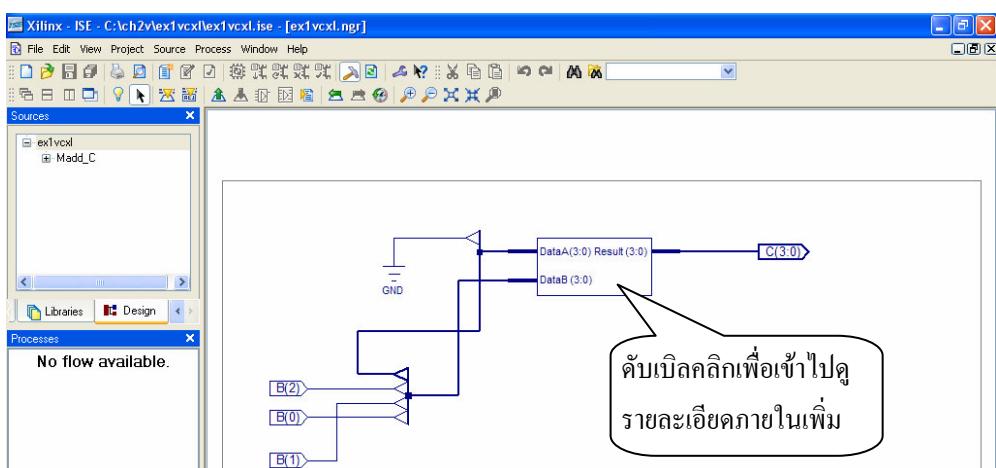
3) ถ้าต้องการดู View RTL Schematic ให้คลิกขวาที่ View RTL Schematic แล้วคลิก Run ในรูปที่ E1.32 เสร็จแล้วให้ดับเบิลคลิกที่รูป Schematic ในรูปที่ E1.34 แล้วจะได้ดังรูปที่ E1.35 ซึ่งแสดงรายการอุปกรณ์พื้นฐานที่นำมาสร้างวงจรที่เราออกแบบจากนั้นคลิก **X** (สีดำ) ที่มุมบนขวาเพื่อกลับไปที่หน้าต่าง Xilinx-ISE อีกครั้ง



รูปที่ E1.33 Synthesis Report

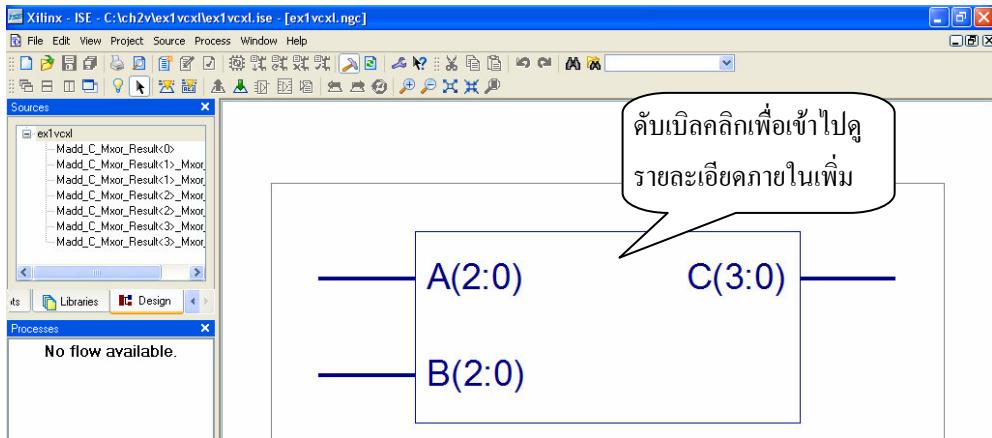


รูปที่ E1.34 View RTL Schematic

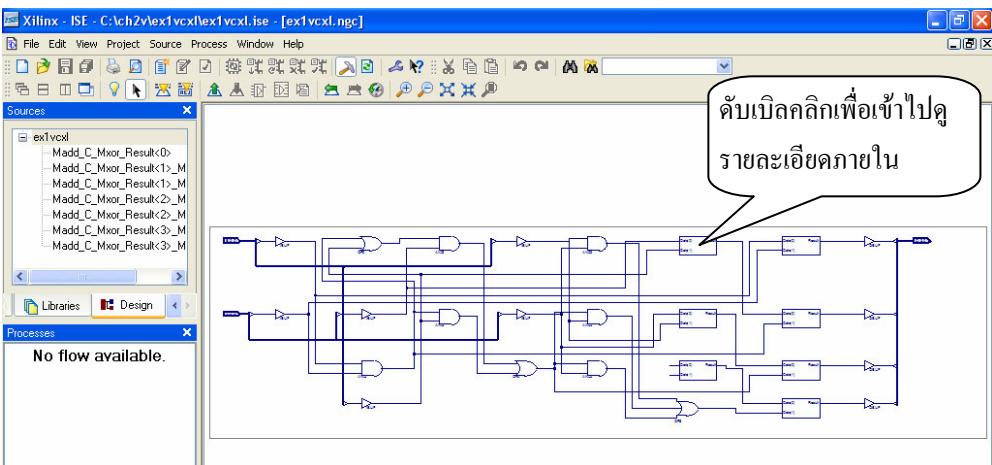


รูปที่ E1.35 View RTL Schematic ที่บอกรายละเอียดภายใน

4) การดู View Technology Schematic ให้คลิกขวาที่ View Technology Schematic แล้วคลิก Run ในรูปที่ E1.32 เสร็จแล้วให้ดับเบิลคลิกที่รูป Schematic ในรูปที่ E1.36 แล้วจะได้ดังรูปที่ E1.37 ซึ่งแสดงอุปกรณ์พื้นฐานที่นำมาสร้างวงจร ซึ่งทำให้เราทราบได้ว่าวงจรที่เราออกแบบนั้นถูกต้องหรือไม่ จากนั้นคลิก **X** (สีดำ) ที่มุมบนขวาเพื่อกลับไปที่หน้าต่าง Xilinx-ISE อีกครั้ง



รูปที่ E1.36 View Technology Schematic

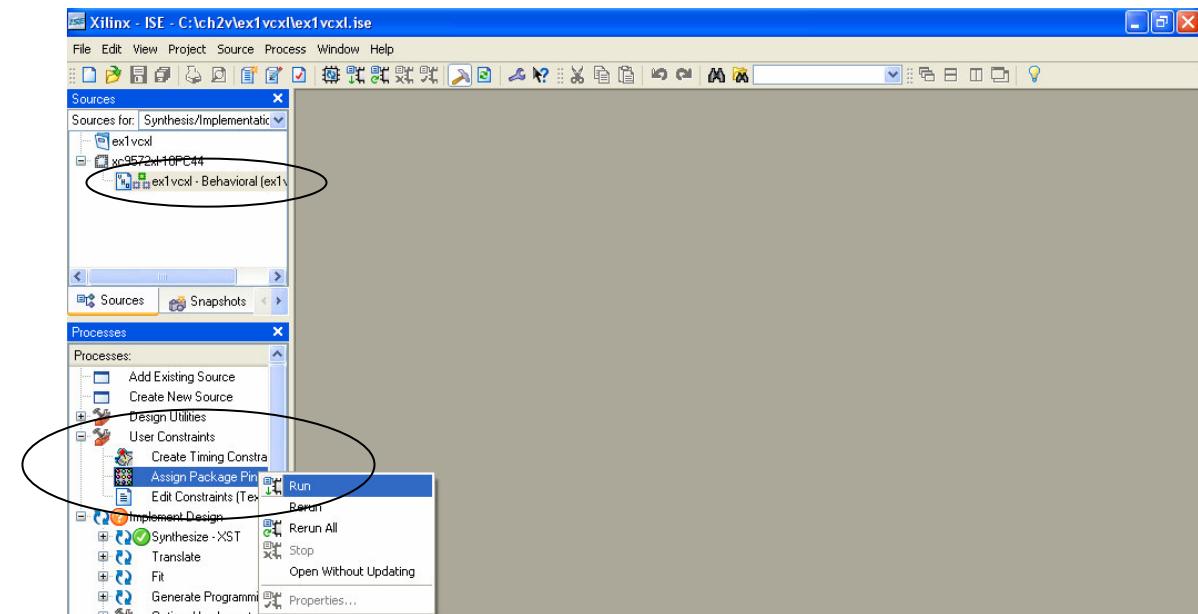


รูปที่ E1.37 View Technology Schematic ที่บอกรายละเอียดภายใน

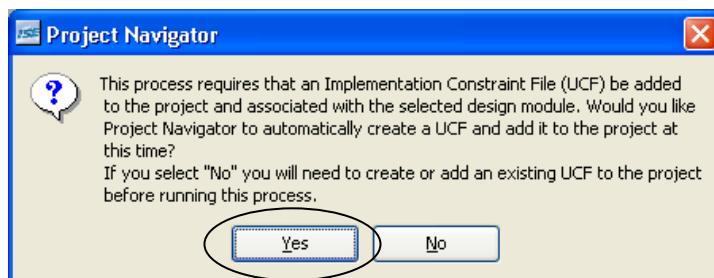
2.16.4 Design Implementation

- ขั้นตอน Implementation constraint file (UCF) เพื่อสร้างไฟล์ที่ระบุว่า อินพุต/เอาต์พุตของวงจรต่อ กับขา (Pin) ใดของ CPLD
 - คลิกที่ชื่อไฟล์ ex1vcxl ในหน้าต่าง Source และเลือกมาส์กไปที่หน้าต่าง Processes คลิก “+” หน้า User Constraints ในหน้าต่าง Processes จนเป็น “-” คลิกขวาที่ Assign Package Pins แล้วคลิก Run ดังรูปที่ E1.38 จะได้หน้าต่างดังรูปที่ E1.39 คลิก Yes เพื่อยืนยันที่จะสร้าง Implementation constraint file (UCF) โดยอัตโนมัติแล้วจะได้หน้าต่าง Xilinx-PACE ดังรูปที่ E1.40
 - การกำหนดขาสัญญาณต่างๆ กรณี เราใช้บอร์ด CPLD Explorer XC9572XL (หรือ CPLD Explorer XC9572) จึงจำเป็นต้องกำหนดให้อินพุต/เอาต์พุตของ CPLD accord ดังนี้ บอร์ด XC9572 ที่เราใช้ มีขา PB1-PB3 (PB3=Slide SW1) เป็นอินพุตของ A ปุ่มกด PB4-PB6 (Slide SW2-Slide SW 4) เป็นอินพุตของ B และใช้ LED1-LED4 เป็นเอาต์พุตของ C กล่าวคือ

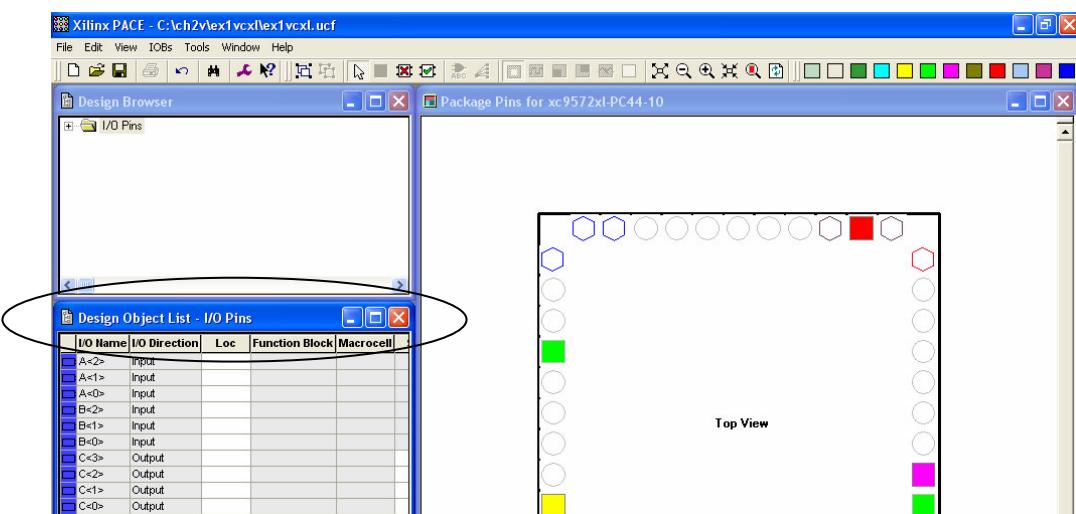
$A(2) = PB1 = INPUT = p39$	$B(2) = PB4 = INPUT = p43$	$C(3) = LED1 = OUTPUT = p38$
$A(1) = PB2 = INPUT = p40$	$B(1) = PB5 = INPUT = p44$	$C(2) = LED2 = OUTPUT = p37$
$A(0) = PB3 = INPUT = p42$	$B(0) = PB5 = INPUT = p1$	$C(1) = LED3 = OUTPUT = P36$
		$C(0) = LED4 = OUTPUT = p35$



รูปที่ E1.38 ขั้นตอน Assign Package Pins

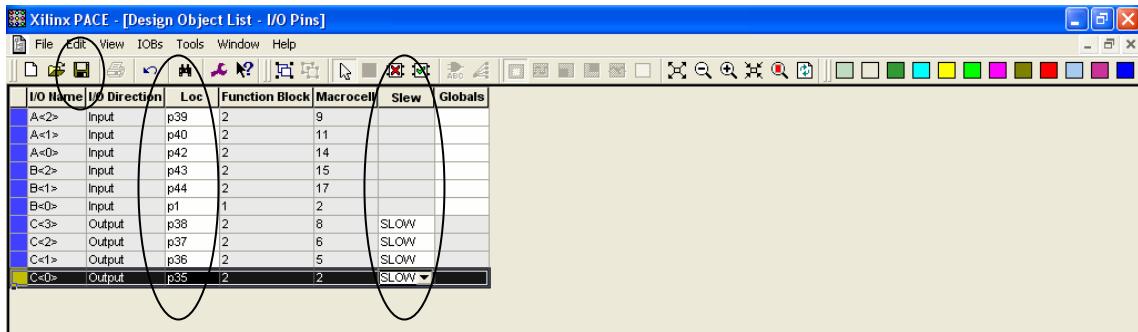


รูปที่ E1.39 หน้าต่าง Project Navigator

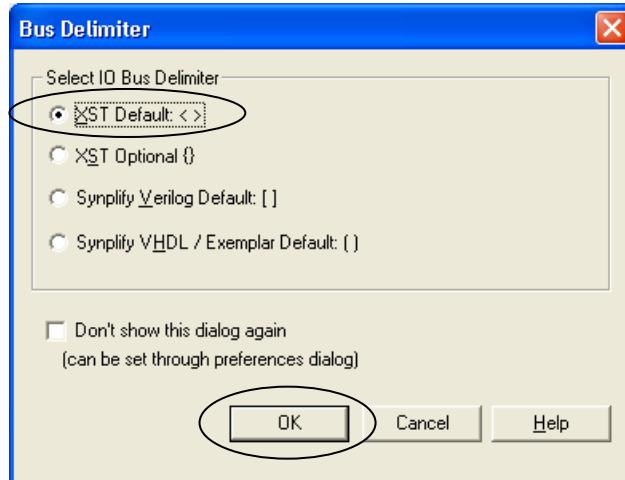


รูปที่ E1.40 หน้าต่าง Xilinx-PACE

คลิก ที่หน้าต่าง Design Object List-I/O Pins ในรูปที่ E1.40 เพื่อขยายขนาดหน้าต่าง กำหนดอินพุตและเอาต์พุตลงในคอลัมน์ Loc คลิกເອາຫຼືມພຸດທັງໝາຍດ້ວຍໃນຄອລັນນີ້ Slew เป็น Slow เพื่อลดการອອສະລິລາດແລະກາຮະກ້ອນໃນສາຍລັບງານເສົ່າງແກ້ຈະໄດ້ຈັງຮູບທີ E1.41 คลิก เพื่อบັນທຶກໄຟລ໌ແກ້ຈະມີໜ້າຕ່າງ Bus Delimiter ຂຶ້ນນີ້ມາ คลิกເລືອກທີ XST Default ດັ່ງຮູບທີ E1.42 ແລ້ວກົດ OK ຈາກນີ້ກົດ (ສືແຈງ) ເພື່ອປັບໜ້າຕ່າງ Xilinx-PACE ແລະກຳລັນໄປທີ່ໜ້າຕ່າງ Xilinx-ISE

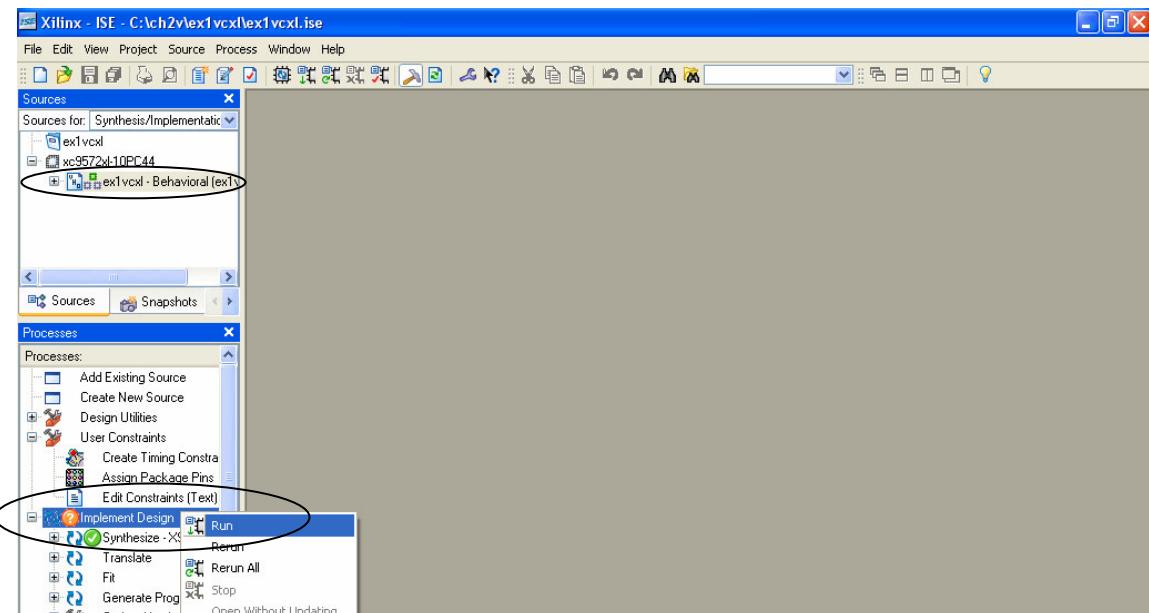


รูปที่ E1.41 การกำหนดอินพุต/เอาต์พุตที่หน้าต่างของ Design Object List-I/O Pins ของหน้าต่าง Xilinx-PACE

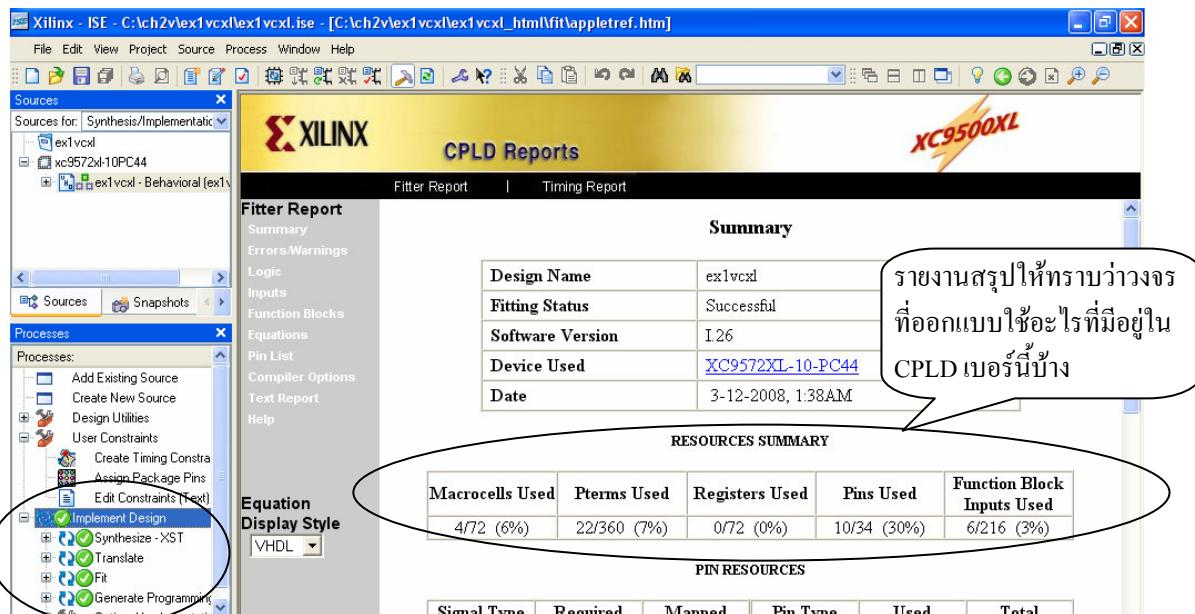


รูปที่ E1.42 หน้าต่าง Bus Delimiter

2) ขั้นตอน Implement Design คลิกที่ชื่อไฟล์ ex1vcx1 ที่หน้าต่าง Source จากนั้นคลิกขวาที่ Implement Design และคลิก Run ดังรูปที่ E1.43 แล้วรอสักครู่ เมื่อได้รับผลลัพธ์แล้วจะได้ ✓ หรือ ⚠ ดังรูปที่ E1.44 จะถือว่าขั้นตอน Implement Design ผ่าน



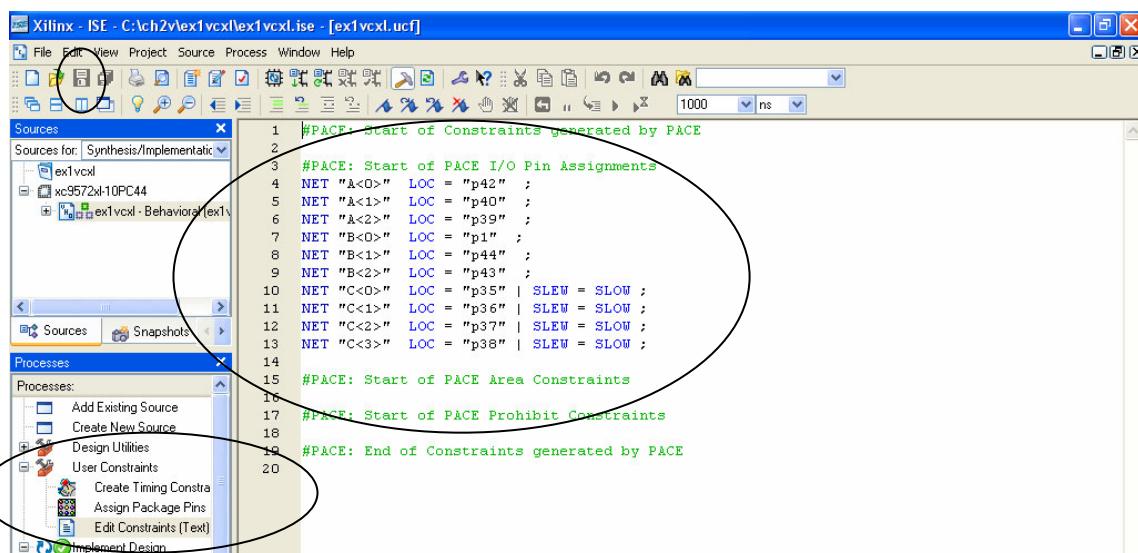
รูปที่ E1.43 หน้าต่างสำหรับขั้นตอน Implement Design



รูปที่ E1.44 หน้าต่าง Processes เมื่อทำขั้นตอน Implement Design เสร็จเรียบร้อยแล้ว

หมายเหตุ

- 1) ขั้นตอน Implement Design นี้โปรแกรมจะทำ Synthesize (ทำชำนาญ), Translate, Fit และ Generate Programming File
- 2) ถ้าผลที่ได้จากการทำ Implement แล้วได้ หน้า Implement Design (ไม่ผ่าน) และ หน้า Translate (ไม่ผ่าน) ให้คลิกขวาที่ Edit Constraints (Text) และคลิก Run แล้วจะได้ดังรูปที่ E1.45 จากนั้นให้ตรวจสอบคุ่าว่ามีการกำหนดขา CPLD เกินหรือผิดหรือไม่ เมื่อแก้ไขเสร็จแล้วให้บันทึกไฟล์ แต่ถ้าไม่แน่ใจให้ลบไฟล์ใน Edit Constraints (Text) ทิ้งทั้งหมดแล้วทำการบันทึกไฟล์อีกรั้ง จากนั้นให้ไปทำขั้นตอน Assign Package Pins อีกรั้ง เมื่อบันทึกไฟล์แล้วให้ทำขั้นตอน Implement ใหม่อีกรั้ง



รูปที่ E1.45 หน้าต่าง Edit Constraints (Text)

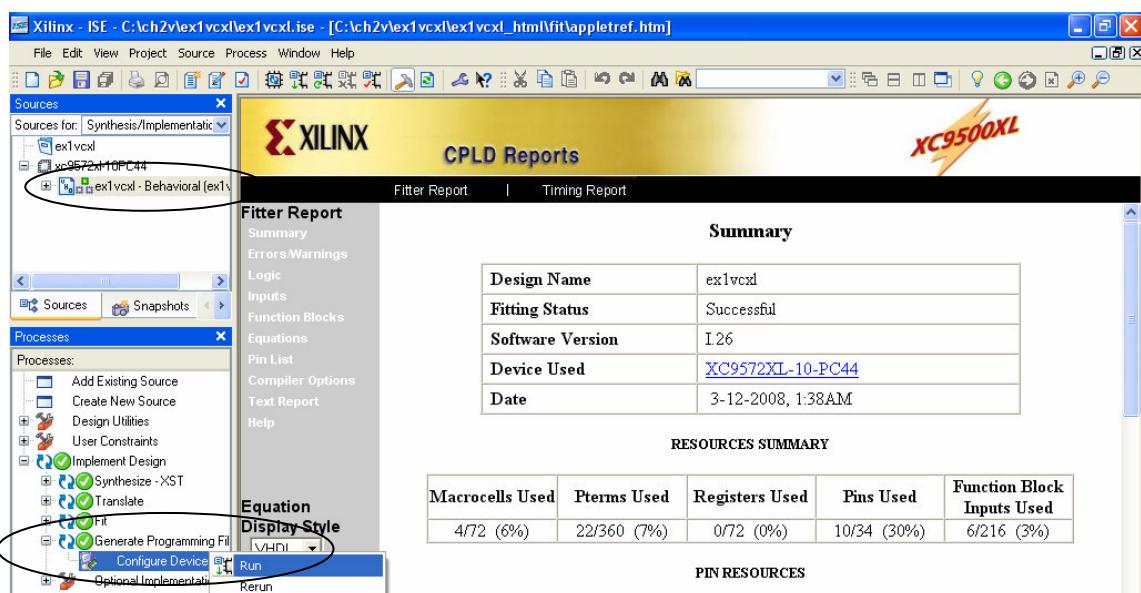
2.16.5 การโปรแกรมข้อมูลวงจรลงชิป CPLD

นำข้อมูลที่ได้จาก Generate Programming File ที่ได้ในขั้นตอน Implement Design มาดาวน์โหลดลงบอร์ดทดลอง CPLD Explorer XC9572XL (หรือ CPLD Explorer XC9572) โดยใช้ดาวน์โหลดเคเบิลดังที่ได้อธิบายในบทที่ 1 ตัวอย่างดาวน์โหลดเคเบิลในโหมด JTAG (สาย JTAG) แสดงดังรูปที่ E1.46 ขั้นตอนการดาวน์โหลดเป็นดังนี้

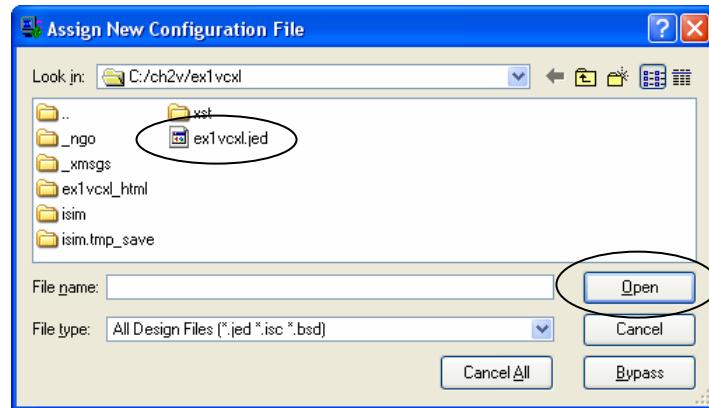


รูปที่ E1.46 ตัวอย่างดาวน์โหลดเคเบิลในโหมด JTAG

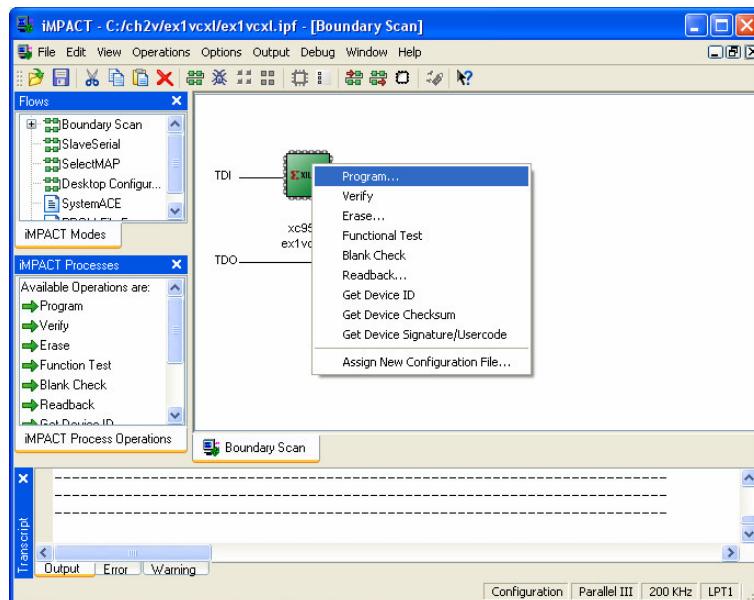
- 1) ต่อสาย JTAG เข้ากับพอร์ตบนคอมพิวเตอร์และบอร์ด CPLD Explorer XC9572XL จากนั้นจ่ายไฟเลี้ยงเข้าบอร์ดทดลอง
- 2) คลิกที่ชื่อไฟล์ ex1vcxl ที่หน้าต่าง Source คลิก “+” หน้า Generate Programming File จะเป็น “-” คลิกขวาที่ Configure Device (iMPACT) แล้วคลิก Run ดังรูปที่ E1.47 แล้วจะได้หน้าต่างถัดไป คลิก Finish แล้วจะได้หน้าต่างดังรูปที่ E1.48 คลิกไฟล์ชื่อ ex1vcxl.jed และคลิก Open แล้วจะได้หน้าต่างถัดไป
- 3) คลิกขวาที่ตัวชิป (สีเขียว) และคลิกที่ Program ดังรูปที่ E1.49 แล้วจะได้หน้าต่างถัดไป เมื่อคลิก OK แล้วโปรแกรมจะทำการดาวน์โหลดข้อมูลวงจรลง CPLD และเมื่อแล้วเสร็จจะได้ดังรูปที่ E1.50 จากนั้นจึงคลิก ✕ (สีแดง) เพื่อปิดหน้าต่าง iMPACT แล้วจะได้หน้าต่างดังรูปที่ E1.51 ให้คลิก No แล้วจึงคลิก ✕ (สีแดง) เพื่อปิดโปรแกรม ISE WebPACK 8.1i



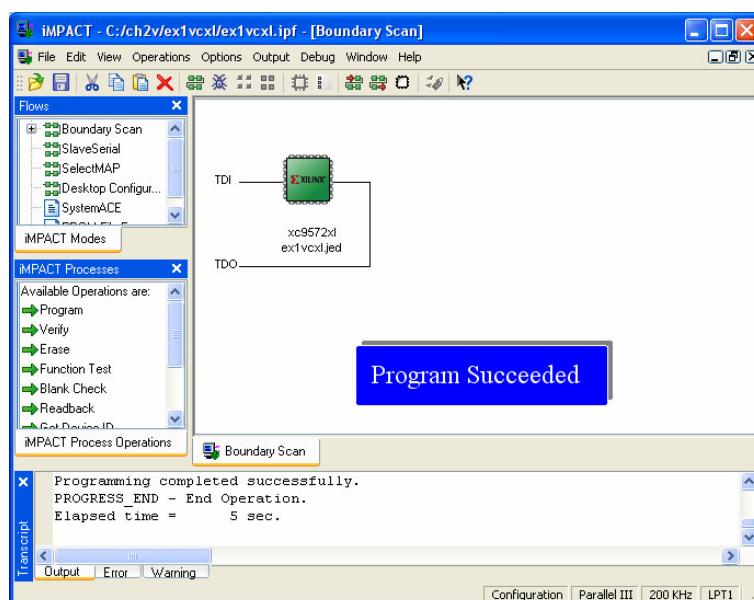
รูปที่ E1.47 ขั้นตอนเริ่มต้นการโปรแกรมข้อมูลวงจรลงชิป CPLD



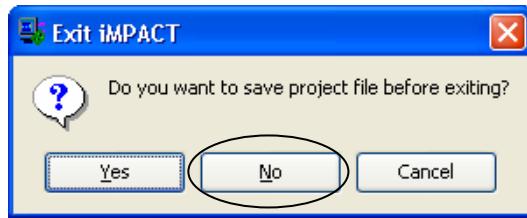
รูปที่ E1.48 หน้าต่าง Assign New Configuration File



รูปที่ E1.49 ขั้นตอน Program



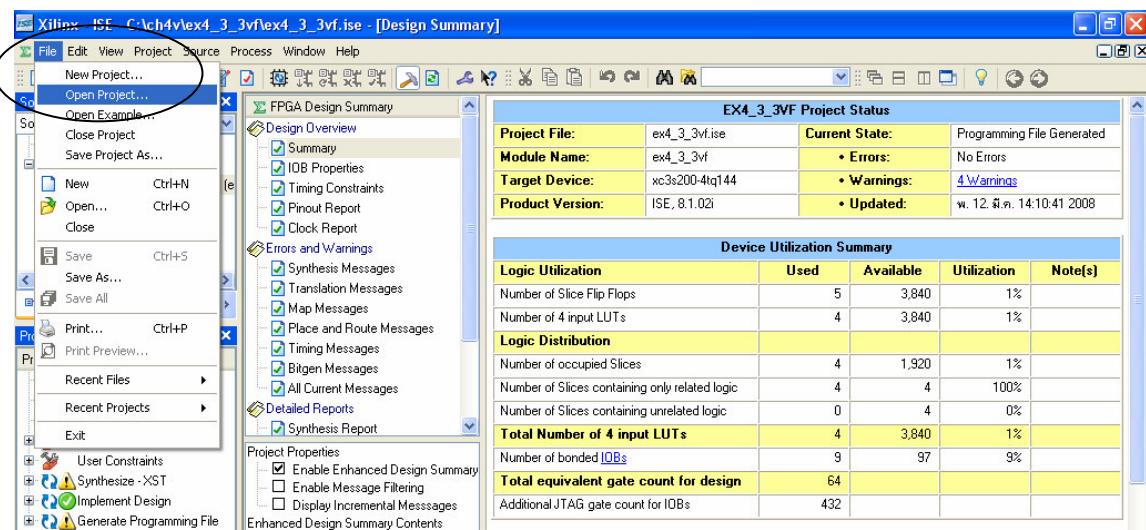
รูปที่ E1.50 ขั้นตอนเมื่อ Program เสร็จแล้ว



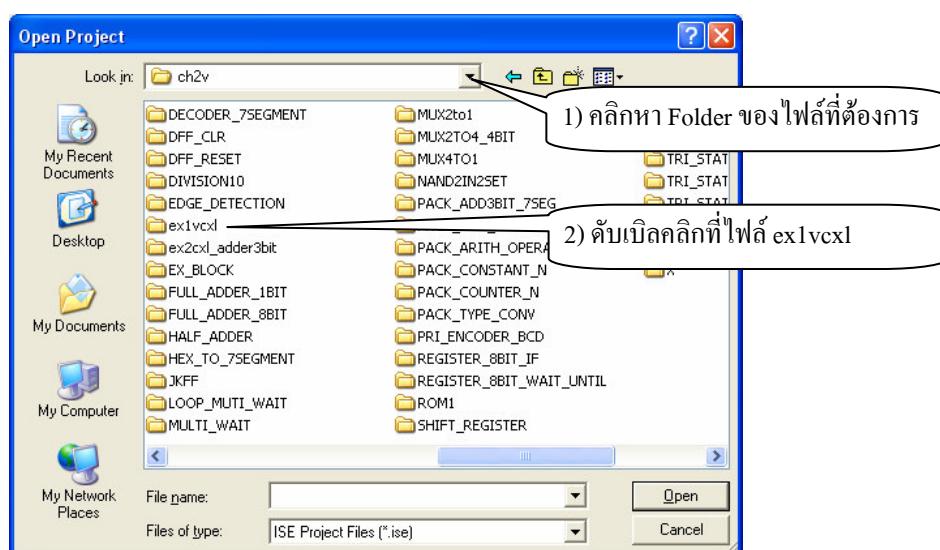
รูปที่ E1.51 หน้าต่าง Exit iMPACT

2.16.6 การเปิดไฟล์ของ CPLD ที่เคยออกแบบไว้แล้วมาใช้งาน

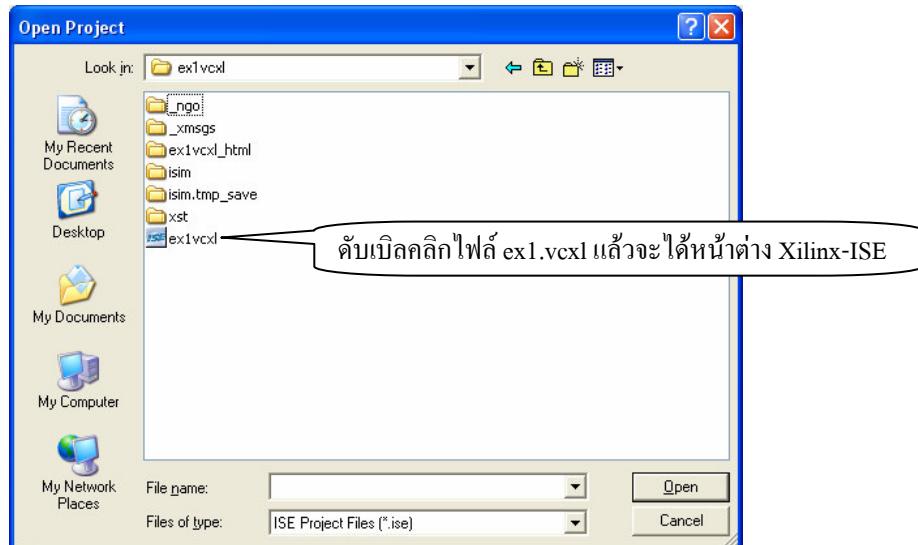
1) เช่นถ้าต้องการเปิดไฟล์ชื่อ ex1vcx1 ที่อยู่ใน Folder ชื่อ ch2v ให้เริ่มที่หน้าจอคอมพิวเตอร์แล้วให้ดับเบิลคลิกที่ Xilinx ISE 8.1i แล้วจะได้หน้าต่าง Xilinx-ISE จากนั้นคลิก File -> Open Project ดังรูปที่ E1.52 แล้วจะได้หน้าต่าง Open Project คลิกหาไฟล์ ex1vcx1 แสดงดังรูปที่ E1.53 แล้วให้ดับเบิลคลิกที่ไฟล์ ex1vcx1 แล้วจะได้ดังรูปที่ E1.54



รูปที่ E1.52 หน้าต่าง Xilinx-ISE

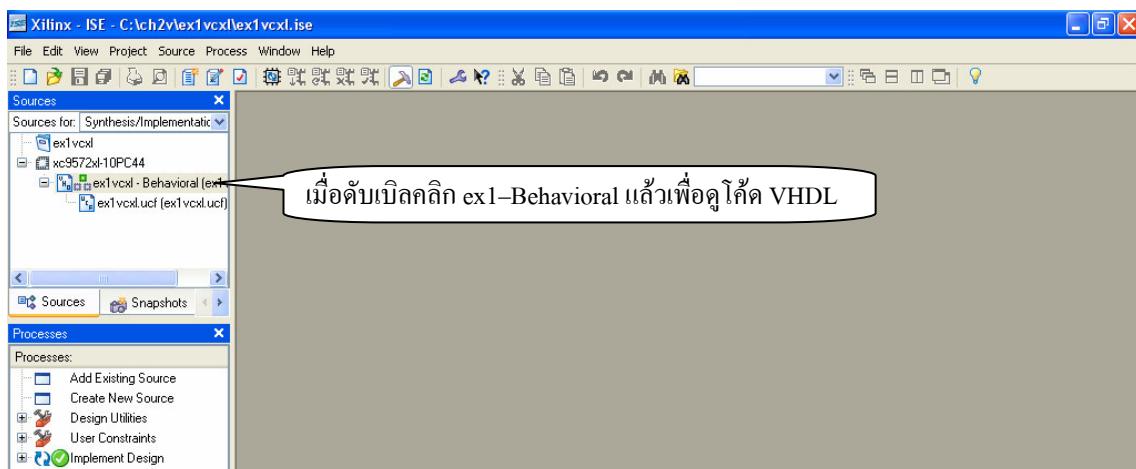


รูปที่ E1.53 หน้าต่าง Open Project (ตำแหน่งไฟล์จะไม่แน่นอน ขึ้นกับจำนวนไฟล์ใน Folder นั้น)



รูปที่ E1.54 หน้าต่าง Open Project

2) จากรูปที่ E1.54 เมื่อดับเบิลคลิกที่ไฟล์ ex1vcxl ก็จะได้หน้าต่าง Xilinx-ISE ของไฟล์ ex1vcxl ดังรูปที่ E1.55 จนนี้ดับเบิลคลิกที่ ex1vcxl-Behavioral ในหน้าต่าง Source ก็จะได้หน้าต่าง Xilinx-ISE ที่มีโค้ด VHDL ของไฟล์ ex1vcxl ดังรูปที่ E1.56



รูปที่ E1.55 หน้าต่าง Xilinx-ISE ของไฟล์ ex1vcxl

```

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

entity ex1vcxl is
    Port ( A, B : in STD_LOGIC_VECTOR (2 downto 0);
           C : out STD_LOGIC_VECTOR (3 downto 0));
end ex1vcxl;

architecture Behavioral of ex1vcxl is
begin
    C <= ('0' &A) + ('0' &B);
end Behavioral;

```

The screenshot shows the Xilinx-ISE interface with the title bar 'Xilinx - ISE - C:\ch2\ex1vcxl\ex1vcxl.ise - [ex1vcxl.vhd]'. The left sidebar shows the 'Sources' and 'Processes' panels. The main area displays the VHDL code for the 'Behavioral' architecture of the 'ex1vcxl' entity. The code uses IEEE libraries and defines a full adder circuit with three inputs (A, B) and four outputs (C).

รูปที่ E1.56 โค้ด VHDL ของวงจรบวก 3 บิตที่อยู่ในไฟล์ ex1vcxl

2.17 การใช้ซอฟต์แวร์ทูลออกแบบวงจรดิจิตอลโดยใช้ FPGA

ขั้นตอนการใช้ซอฟต์แวร์ทูล ISE WebPACK 8.1i ในการออกแบบวงจรดิจิตอลด้วย FPGA โดยใช้ภาษา VHDL นั้น เก็บจะเหมือนกับ CPLD ทุกประการ เสริชแล้วเจ้งความสนใจคลองบอร์ด FPGA Discovery-III XC3S200F (หรือบอร์ด FPGA Discovery-III XC3S200F4) ซึ่งขั้นตอนดาวน์โหลดนี้จะแตกต่างจาก CPLD ส่วนคนที่ใช้ OS ที่เป็น Microsoft Vista Business (32-bit) ให้ใช้ซอฟต์แวร์ทูล ISE WebPACK 10.1i ซึ่งลักษณะการใช้งานแตกต่างกันเล็กน้อย

ตัวอย่างที่ 2.36 ให้ออกแบบวงจรบวกขนาด 3 บิตด้วย FPGA โดยมี Entity ชื่อ ex2vf และโค้ด VHDL แสดงดังรูป E2.1

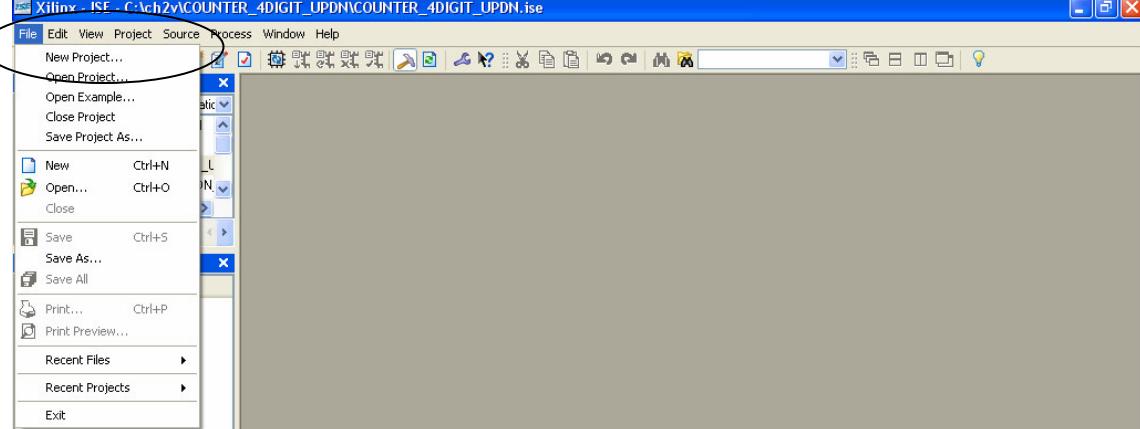
```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex2vf is
7     Port ( A,B : in STD_LOGIC_VECTOR (2 downto 0);
8             C : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex2vf;
10
11 architecture Behavioral of ex2vf is
12 begin
13
14     C <= ('0' &A) + ('0' &B);
15
16 end Behavioral;
```

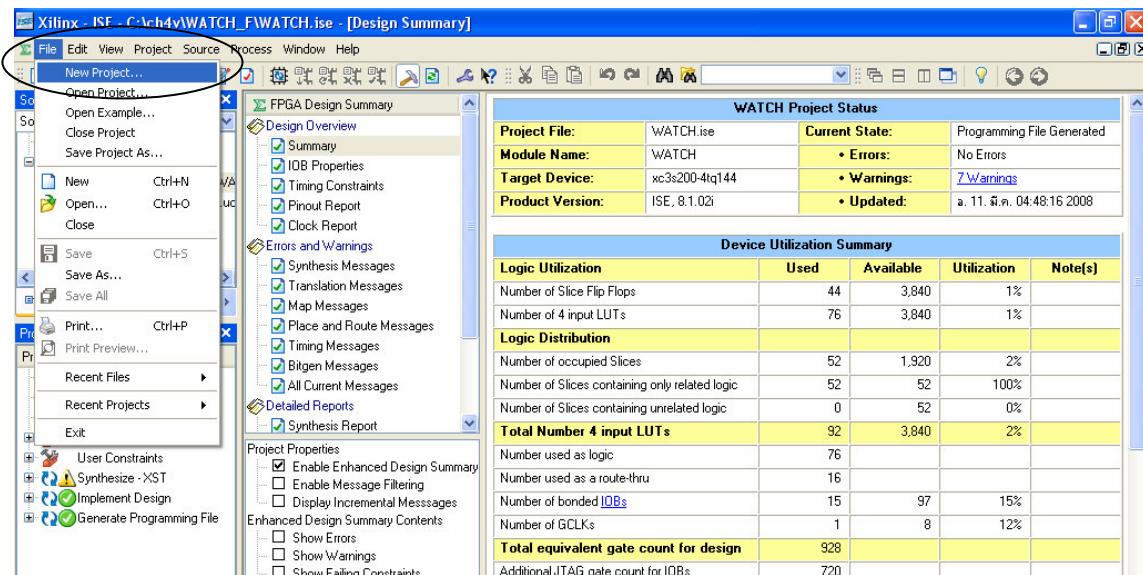
รูปที่ E2.1 โค้ดวงจรบวก 3 บิต

2.17.1 ขั้นตอนการออกแบบวงจร (Design entry)

1) ก่อนใช้ ISE WebPACK 8.1i ให้ตั้งความละเอียดภาพที่ 1024 x 768 pixels คอมพิวเตอร์ควรมี RAM ไม่น้อยกว่า 512 MB และควรปิดโปรแกรมอื่นๆ ที่ไม่เกี่ยวข้องทั้งหมดเพื่อป้องกันความติดพลาดเนื่องจาก RAM เพียงไม่พอ เพื่อความสะดวกผู้เขียนแนะนำให้มี RAM ไม่น้อยกว่า 1-2 GB และควรมีพื้นที่ว่างในฮาร์ดดิสก์ประมาณไม่น้อยกว่า 10 GB ส่วนคนที่ใช้ซอฟต์แวร์ทูล ISE WebPACK 10.1i นั้นควรมีพื้นที่ว่างในฮาร์ดดิสก์ประมาณไม่น้อยกว่า 15-20 GB จากนั้นจึงสร้าง Folder ชื่อ ch2v (หรือชื่อ อื่น) ไว้ในไดร์ฟ C จากนั้นเริ่มที่จัดการพิวเตอร์โดยคลิกปุ่ม Start -> Programs -> Xilinx ISE 8.1i -> Project Navigator หรือ ดับเบิลคลิกที่ Xilinx ISE 8.1i แล้วจะได้หน้าต่าง Xilinx-ISE (ถ้ามีหน้าต่าง Tip of the Day ข้อนี้มาให้คลิก OK) คลิกที่ File -> New Project ดังรูปที่ E2.2a) หรือรูปที่ E2.2b) และจะได้หน้าต่าง (หรือ Dialog box) New Project Wizard-Create New Project



E2.2a) หน้าต่าง Xilinx-ISE

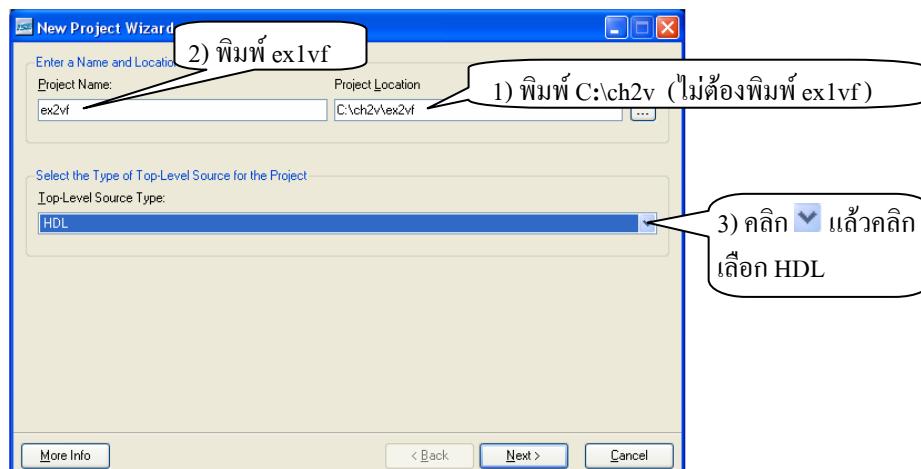


E2.2b) หน้าต่าง Xilinx-ISE

รูปที่ E1.2 หน้าต่าง Xilinx-ISE ที่ความละเอียดจอภาพ 1024 by 768 pixels อาจจะเป็นดังรูป E1.2a) หรือ E1.2b)

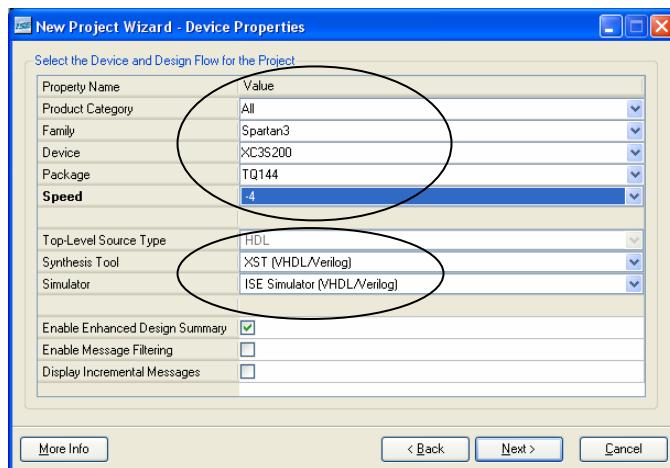
2) ที่หน้าต่าง New Project Wizard–Create New Project สร้างโปรเจกต์ไฟล์ (Project File) ใหม่โดยพิมพ์ชื่อ ch2v (ชื่อ Folder) ลงในช่องว่างของ Project Location ก่อน แล้วพิมพ์ชื่อ ex2vf ลงในช่องว่างของ Project Name คลิกที่ Top-Level Source Type เป็น HDL แล้วจะได้ดังรูปที่ E2.3 คลิก Next แล้วจะได้หน้าต่าง New Project Wizard–Device Properties

หมายเหตุ การตั้งชื่อจะใช้กฎเกณฑ์ที่กำหนดในภาษา VHDL ในข้อ 2.4 คือ จะต้องเขียนด้วย A-Z, a-z และตัวถัดไปอาจเป็น A-Z, a-z, 0-9 หรือ Underscores (_) แต่ห้ามจบด้วย (_) และห้ามเว้นช่องว่างระหว่างตัวอักษร

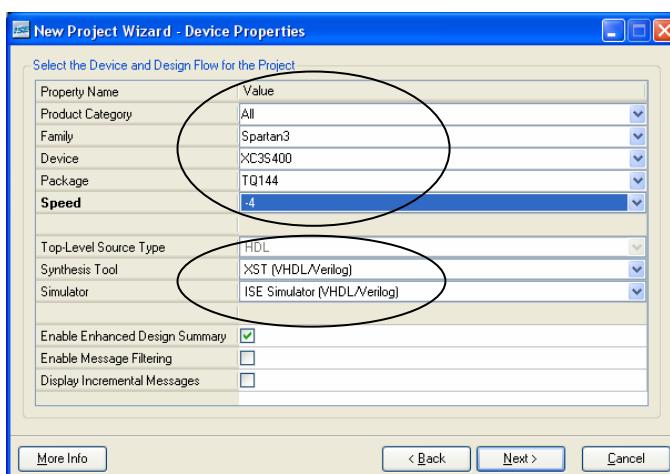


รูปที่ E2.3 หน้าต่าง New Project Wizard–Create New Project

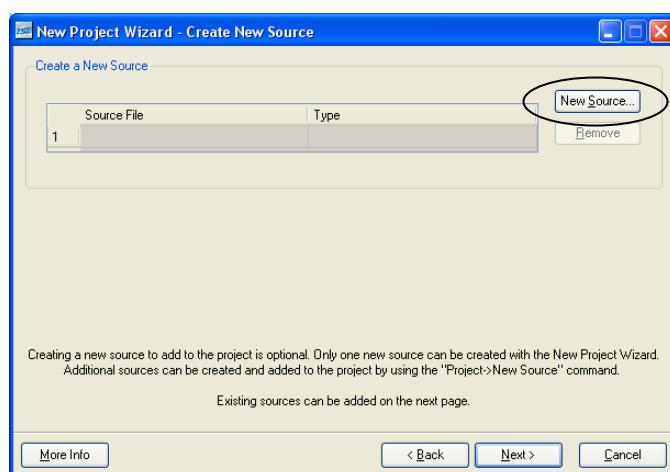
3) ในการนี้ที่ใช้บอร์ด FPGA Discovery-III XC3S200F ให้คลิกดังรูปที่ E2.4 คือ FPGA ตระกูล (Family) Spartan3, เบอร์ (Device) XC3S200, Package แบบ 144 ขา (Package : TQ144) และ Speed Grade : -4 แต่ที่ใช้บอร์ด FPGA Discovery-III XC3S200F4 ให้คลิกเป็นเบอร์ XC3S400 ดังรูปที่ E2.5 แล้วเลือกที่ Synthesis tool เป็น XST (VHDL/Verilog) และ Simulator เป็น ISE Simulator (VHDL/Verilog) จากนั้นคลิก Next ในรูปที่ E2.4 (หรือในรูปที่ E2.5) แล้วจะได้หน้าต่างดังรูปที่ E2.6



รูปที่ E2.4 หน้าต่าง New Project Wizard–Device Properties ในกรณีที่ใช้ FPGA เบอร์ XC3S200-4TQ144C

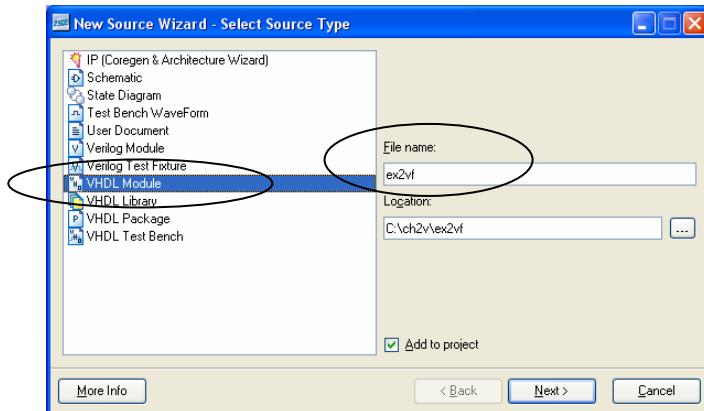


รูปที่ E2.5 หน้าต่าง New Project Wizard–Device Properties ในกรณีที่ใช้ FPGA เบอร์ XC3S400-4TQ144C

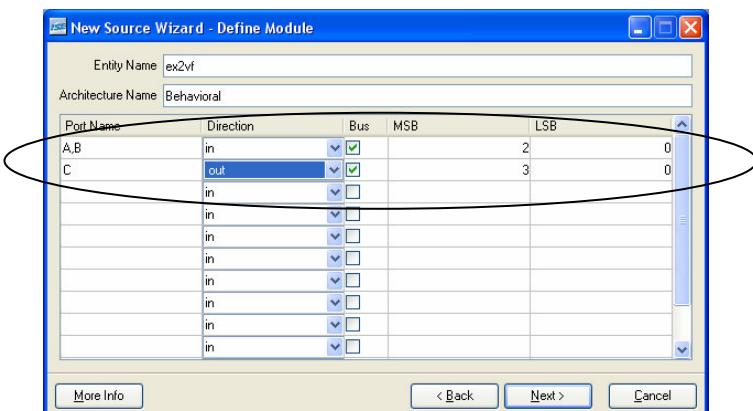


รูปที่ E2.6 หน้าต่าง New Project Wizard–Create New Source

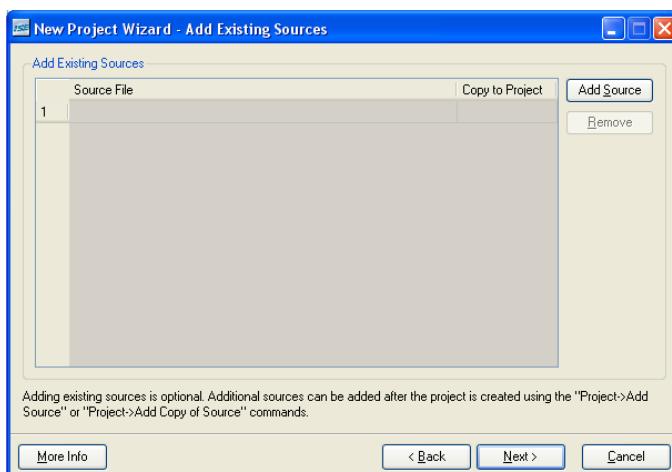
- 4) คลิกปุ่ม New Source ในรูปที่ E2.6 และจะได้หน้าต่างดังไป พิมพ์ชื่อ Source File ชื่อ ex2vf ลงในช่อง File Name และคลิกที่ VHDL Module ดังรูปที่ E2.7 คลิก Next และจะได้หน้าต่างดังรูปที่ E2.8 เมื่อกำหนดอินพุต A,B และเอาต์พุต C เรียบร้อยแล้ว คลิก Next ครั้ง คลิก Finish 1 ครั้ง และคลิก Next อีก 1 ครั้งก็จะได้หน้าต่าง New Project Wizard–Add Existing Source ดังรูปที่ E2.9 (ซึ่งจะอธิบายในภายหลัง)



รูปที่ E2.7 หน้าต่าง New Source Wizard–Select Source Type



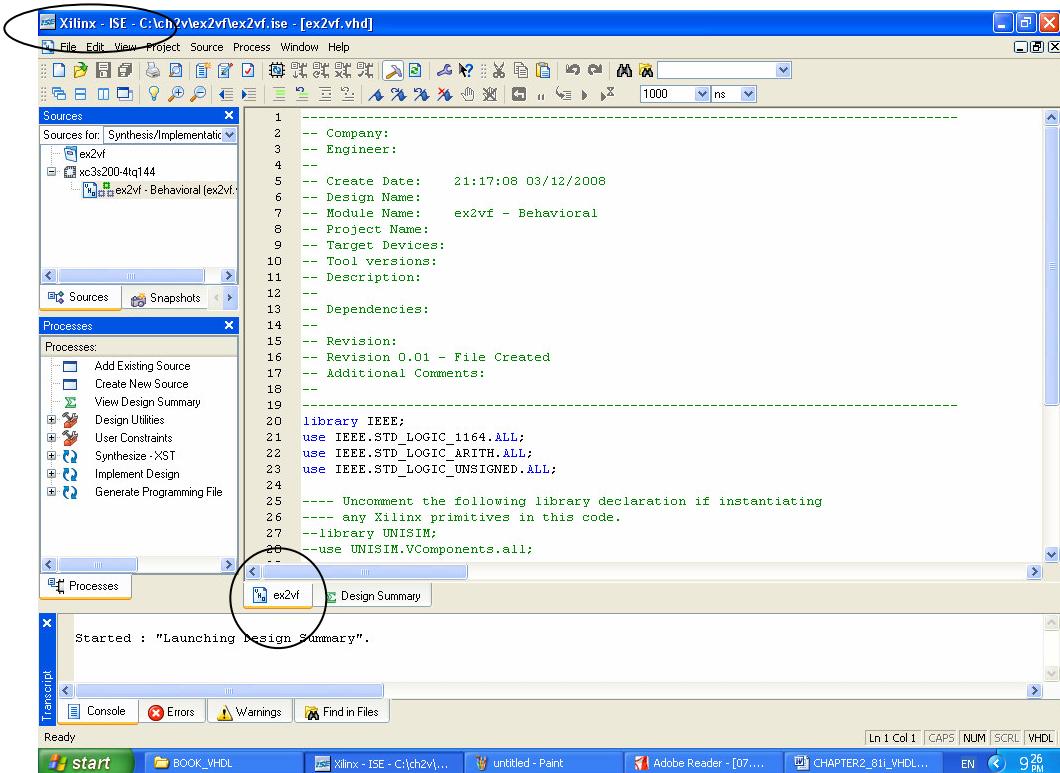
รูปที่ E2.8 หน้าต่าง New Source Wizard–Define Module ซึ่งกรณีเป็นบล๊อกคลิก “√” ที่คอลัมน์ Bus ด้วย



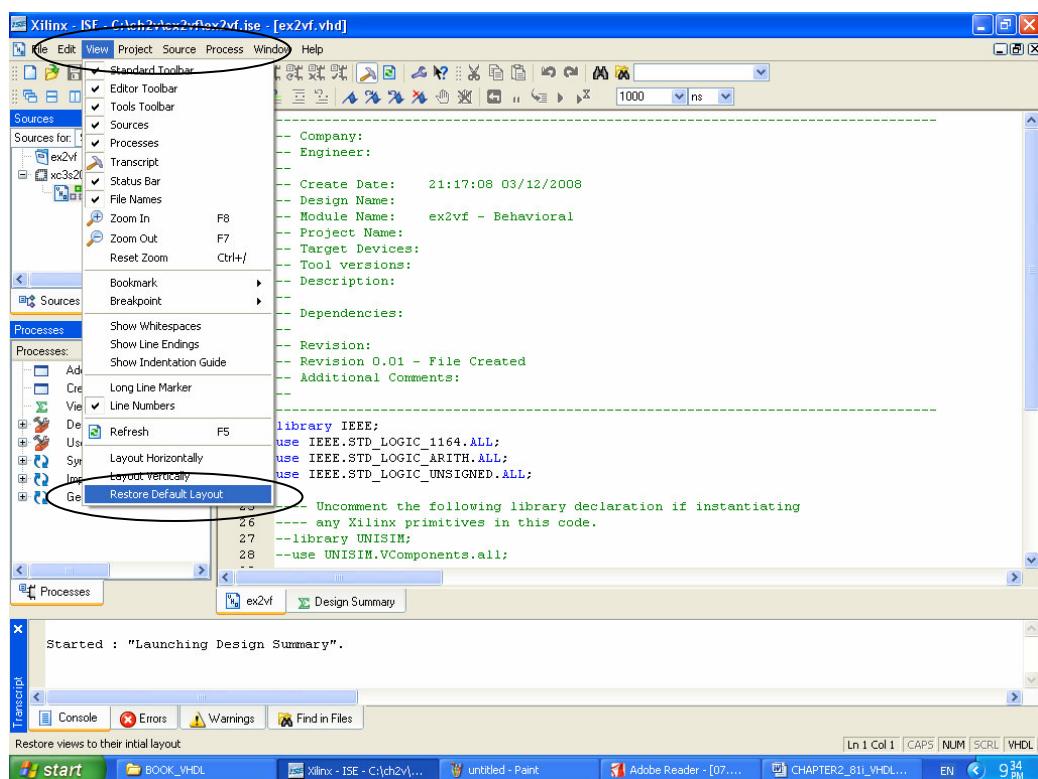
รูปที่ E2.9 หน้าต่าง New Project Wizard-Add Existing Source

5) จากรูปที่ E2.9 คลิก Next 1 ครั้ง คลิก Finish อีก 1 ครั้งแล้วจะได้หน้าต่าง Xilinx-ISE จากนั้นคลิก Tab ex2vf ที่อยู่ด้านล่างแล้วจะได้หน้าต่างสำหรับเขียนโค้ด VHDL (โปรแกรม Text Editor) ดังรูปที่ E2.10

6) ที่หน้าต่าง Xilinx-ISE ตั้งรูปที่ E2.10 ให้ผู้อ่านทำความคุ้นเคยกับหน้าต่างนี้ โดยคลิก View ตั้งรูปที่ E2.11 แล้วคลิกที่แถบของข้อความต่างๆ คุพร้อมกับสังเกตการเปลี่ยนแปลงในส่วนของหน้าต่างย่อยต่างๆ ว่ามีหน้าต่างใดหายไปบ้าง และถ้าต้องการให้หน้าต่างต่างๆ กลับมาไม่ลักษณะเหมือนเดิมให้คลิกที่แถบล่างสุดคือ Restore Default Layout

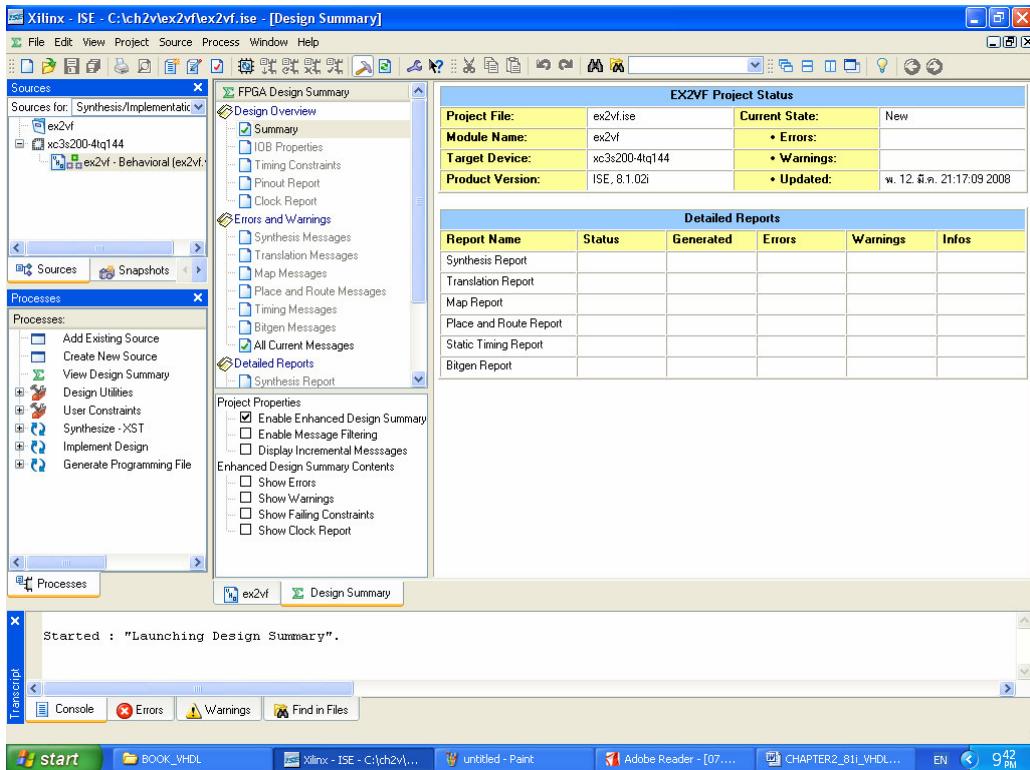


รูปที่ E2.10 หน้าต่าง Xilinx-ISE สำหรับเขียนโค้ด VHDL (โปรแกรม Text Editor)



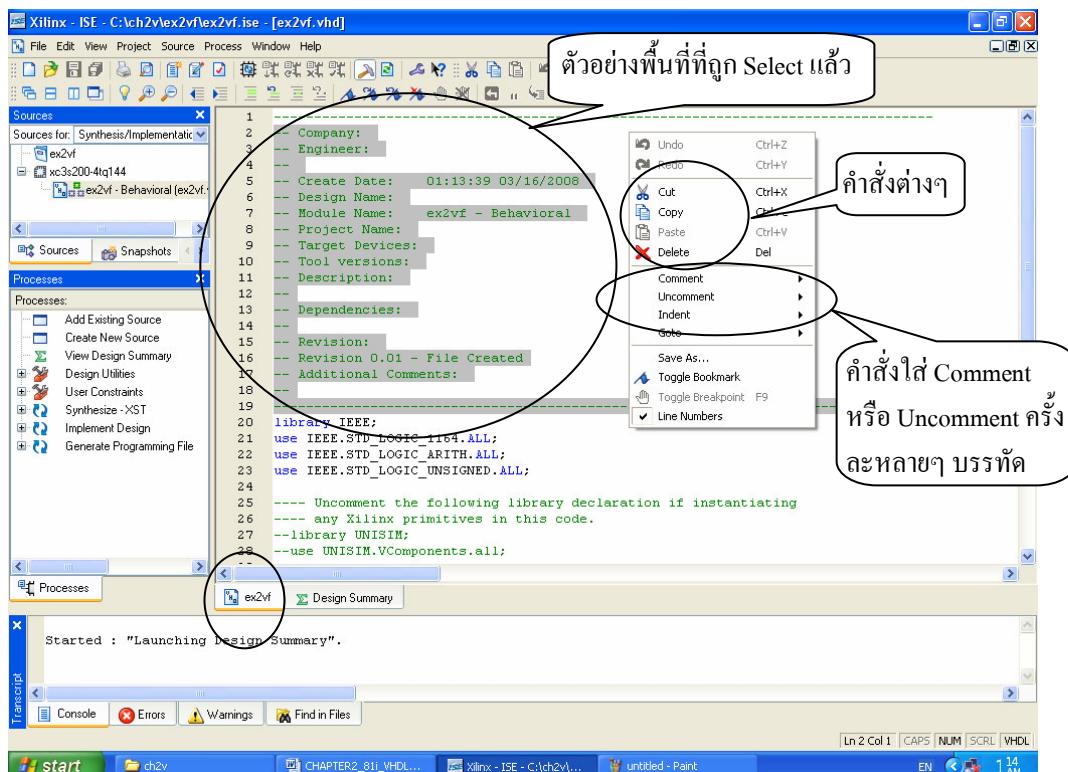
รูปที่ E2.11 การใช้ View

7) ก่อนทำการทำงานต่อไปควรทำความสะอาดกับหน้าต่างหลัก Xilinx-ISE (Project Navigator main window) ดังรูปที่ E2.12 ซึ่งจะเหมือนกับกรณี CPLD ทุกประการดังที่ได้อธิบายในรูปที่ E1.12



รูปที่ E2.12 หน้าต่างหลัก Project Navigator (ความละเอียดของภาพ 1024 x 768 pixels)

8) จากนั้นให้ลองคลิกขวาที่หน้าต่างหลัก (Workspace) ก็จะพบว่ามีคำสั่งต่างๆ ที่เราคุ้นเคยกันเดือดูแล้วดังในรูปที่ E2.13



รูปที่ E2.13 หน้าต่างหลัก (Workspace)

หมายเหตุ ในรูปที่ E2.13 การใส่ Comment แบบหลายๆ บรรทัดสามารถทำได้เช่นเดียวกับการออกแบบด้วย CPLD ทุกประการ

9) ทำการปรับปรุงแก้ไขโค้ดให้เป็นตามรูปที่ E2.1 ซึ่งจะเห็นว่าตัวซอฟต์แวร์ทุกเป็นคนจัดการให้เราเก็บทั้งหมด โดยเราเขียน เค้าโครงบรรทัดที่ 14 เพียงบรรทัดเดียวเท่านั้น เมื่อเสร็จแล้วจะได้ดังรูปที่ E2.14 แล้วคลิก บันทึกไฟล์

```

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

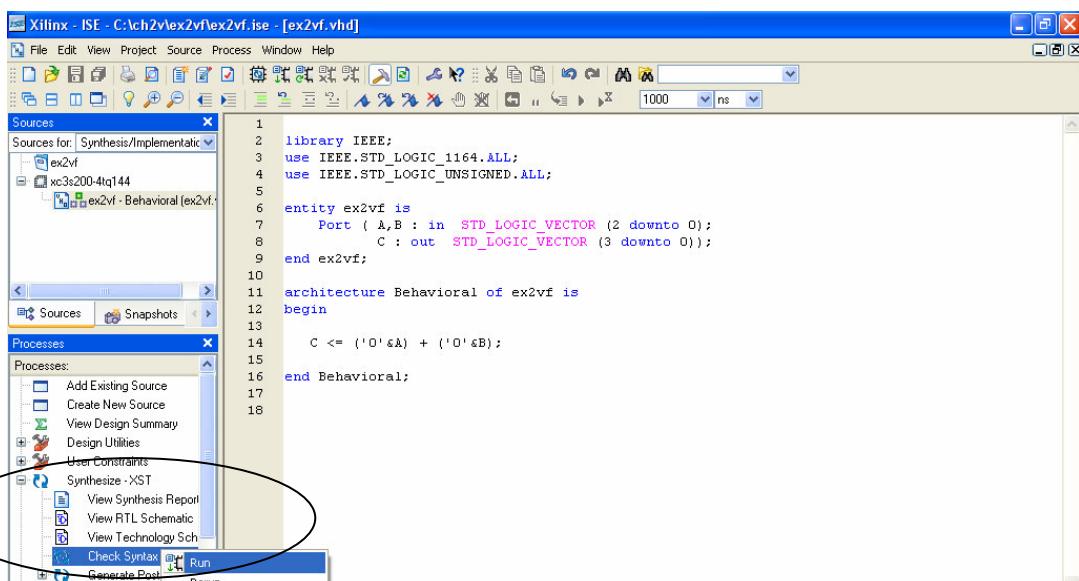
entity ex2vf is
    Port ( A,B : in STD_LOGIC_VECTOR (2 downto 0);
           C : out STD_LOGIC_VECTOR (3 downto 0));
end ex2vf;

architecture Behavioral of ex2vf is
begin
    C <= ('0' & A) + ('0' & B);
end Behavioral;

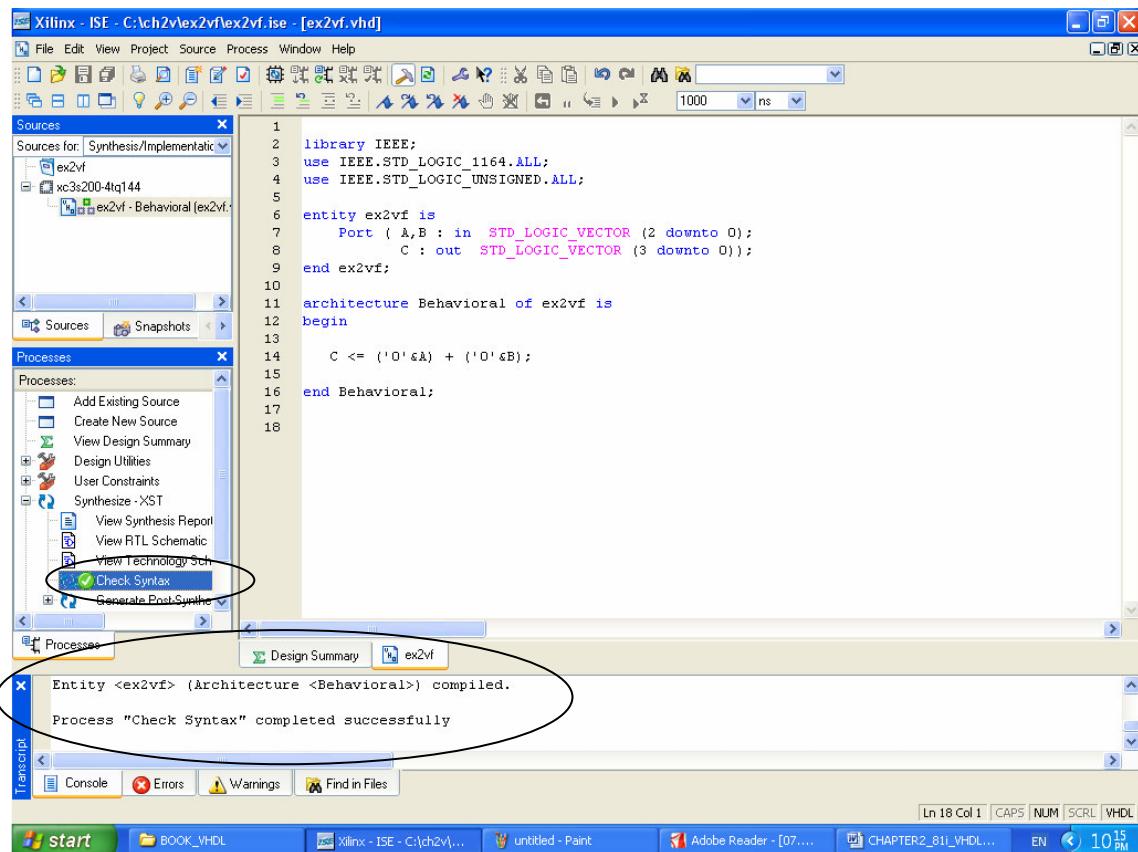
```

รูปที่ E2.14 โค้ด VHDL ของวงจรนาฬิกา 3 บิตที่เขียนเสร็จเรียบร้อยแล้ว

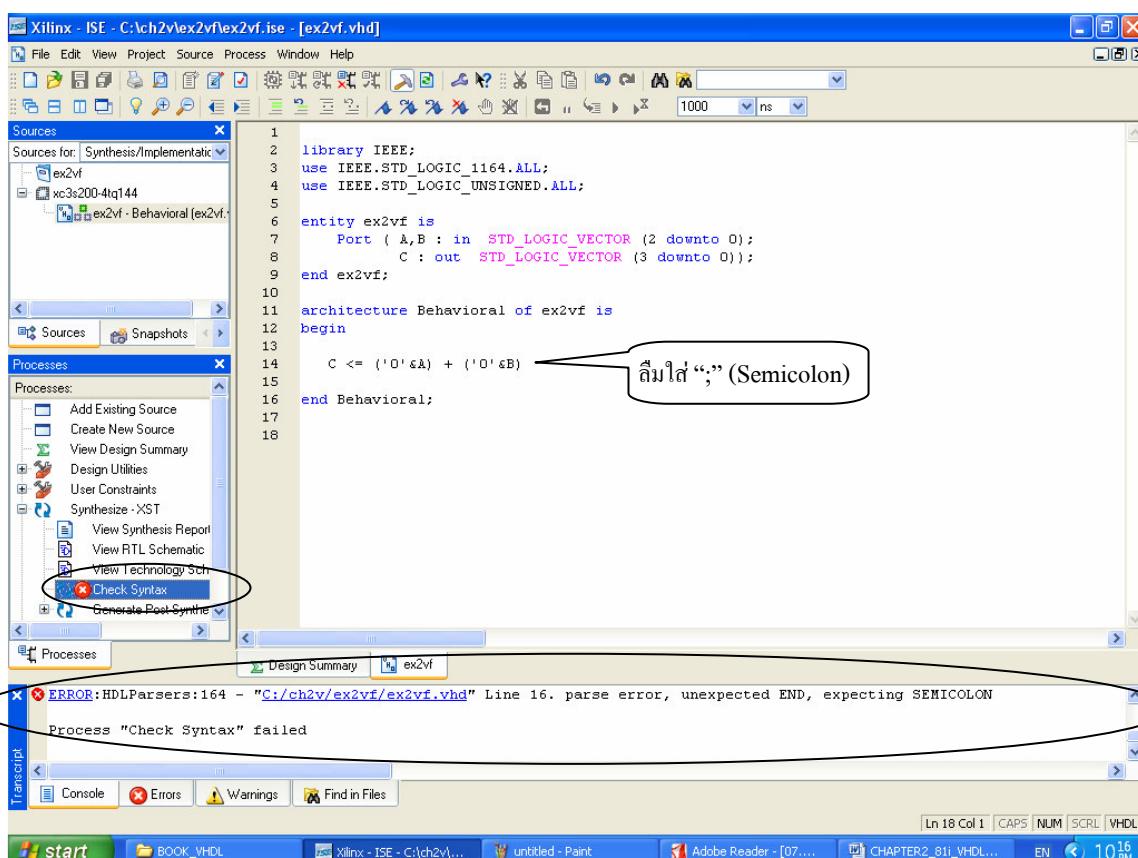
10) การตรวจสอบความถูกต้อง (Syntax) ของโค้ด คลิกที่ “+” หน้า Synthesize-XST ในหน้าต่าง Process จนเป็น “-” คลิกขวาที่ Check Syntax แล้วคลิก Run ดังรูปที่ E2.15 ถ้าไม่มีข้อผิดพลาดจะปรากฏข้อความในหน้าต่าง Transcript ว่า Process “Check Syntax” completed successfully ดังรูปที่ E2.16 ในขั้นตอนนี้ถือว่าโค้ดถูก Compiled เรียบร้อยแล้ว แต่ถ้ามีข้อผิดพลาด เช่น ดังรูปที่ E2.17 บอกว่า Line 16. parse error, unexpected END, expecting SEMICOLON ซึ่งล้มใส่ ";" (Semicolon) ตอนจบบรรทัดที่ 14 เมื่อแก้ไขแล้วคลิก บันทึกไฟล์ แล้วให้ตรวจสอบความถูกต้องซ้ำอีกครั้ง ถ้าไม่มีข้อผิดพลาดอีกต่อไปถือว่าขั้นตอน Design entry นี้เสร็จสมบูรณ์แล้ว คลิก (สีดำ) เพื่อปิดโปรแกรม Text Editor และกลับไปที่หน้าต่าง Xilinx-ISE แล้วจะได้ดังรูปที่ E2.18 หรือคลิก (สีแดง) หากมีความต้องการออกจากโปรแกรม ISE WebPACK 8.1i



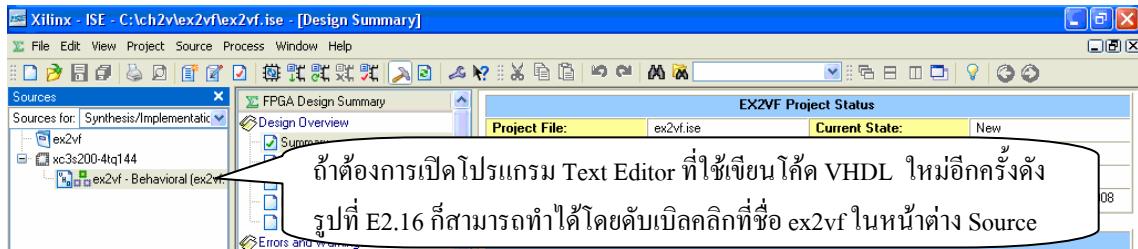
รูปที่ E2.15 การตรวจสอบความถูกต้อง



รูปที่ E2.16 หน้าต่าง Xilinx-ISE เมื่อ Check syntax ผ่าน



รูปที่ E2.17 การเขียนโค้ดที่มีข้อผิดพลาด



รูปที่ E2.18 หน้าต่าง Xilinx-ISE

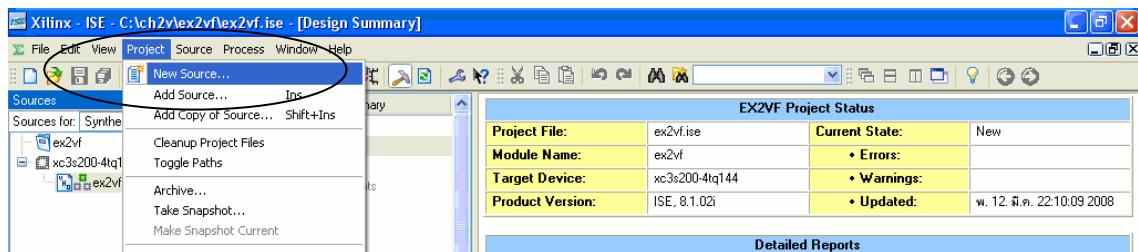
2.17.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification)

การตรวจสอบความถูกต้องของวงจรที่ออกแบบนั้นทำได้โดยนำโค้ด VHDL ที่ได้ไปจำลองการทำงาน (Simulation) ซึ่งการจำลองการทำงานในเมืองคืนที่อธิบายในข้อนี้จะใช้ Waveform Editor ส่วนการจำลองการทำงานที่ซับซ้อนนั้นจำเป็นต้องเขียน Testbench โดยจะอธิบายในบทที่ 6 การจำลองเฉพาะพฤติกรรม (Behavioral simulation) นั้นจะไม่คำนึงผลเวลาล่าช้า (Delay time) ต่างๆ โค้ดที่เขียนนี้อาจนำไปสังเคราะห์วงจร (Synthesis) ได้หรือไม่ได้ แต่สามารถเขียนโค้ดได้รวดเร็ว

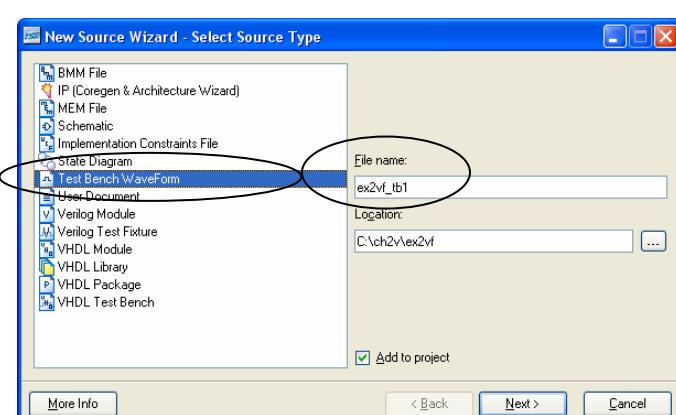
การออกแบบวงจรที่ไม่มีความซับซ้อนนั้นอาจไม่จำเป็นต้องทำขั้นตอนการตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification) ผู้อ่านสามารถข้ามไปท่าหัวข้อ 2.17.3 ซึ่งเป็นขั้นตอนการสังเคราะห์วงจร (Design synthesis) ได้เลย

หลังจากที่เราเขียนโค้ด VHDL และตรวจสอบความถูกต้อง (Syntax) ของโค้ดเรียบร้อยแล้ว ขั้นตอนจำลองเฉพาะพฤติกรรม (Behavioral simulation) ของวงจรโดยใช้ Waveform Editor สามารถทำได้ดังนี้

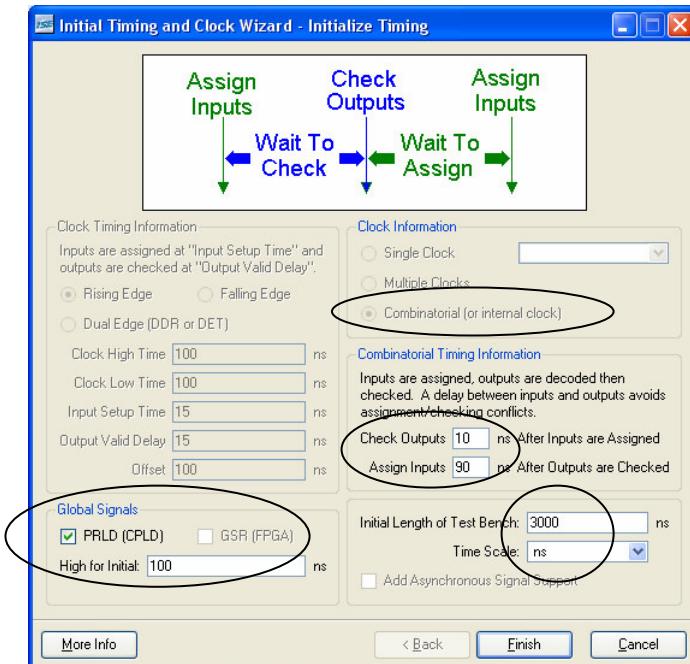
1. คลิก Project->New Source ดังรูปที่ E2.19 แล้วจะได้หน้าต่าง New Source Wizard-Select Source Type พิมพ์ชื่อ ex2vf_tb1 (ห้ามตั้งชื่อไปช้ากับชื่อ ex2vf) แล้วคลิกที่ Test Bench Waveform ดังรูปที่ E2.20 จากนั้นคลิก Next 2 ครั้งและคลิก Finish อีก 1 ครั้งแล้วจะได้หน้าต่างดังไป จากนั้นเลือกและกำหนดค่าต่างๆ ดังรูปที่ E2.21



รูปที่ E2.19 หน้าต่าง Xilinx-ISE



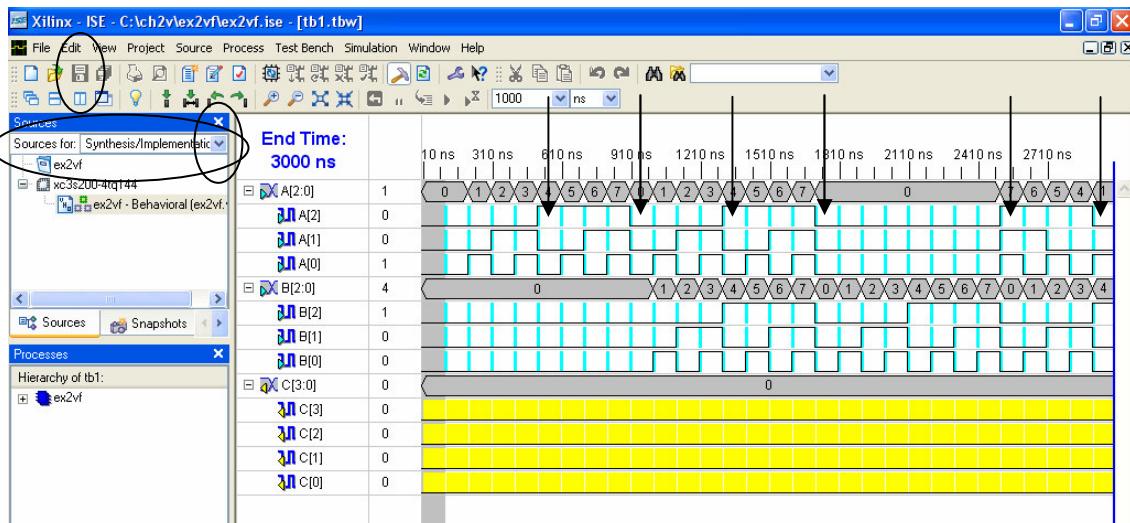
รูปที่ E2.20 หน้าต่าง New Source Wizard-Select Source Type



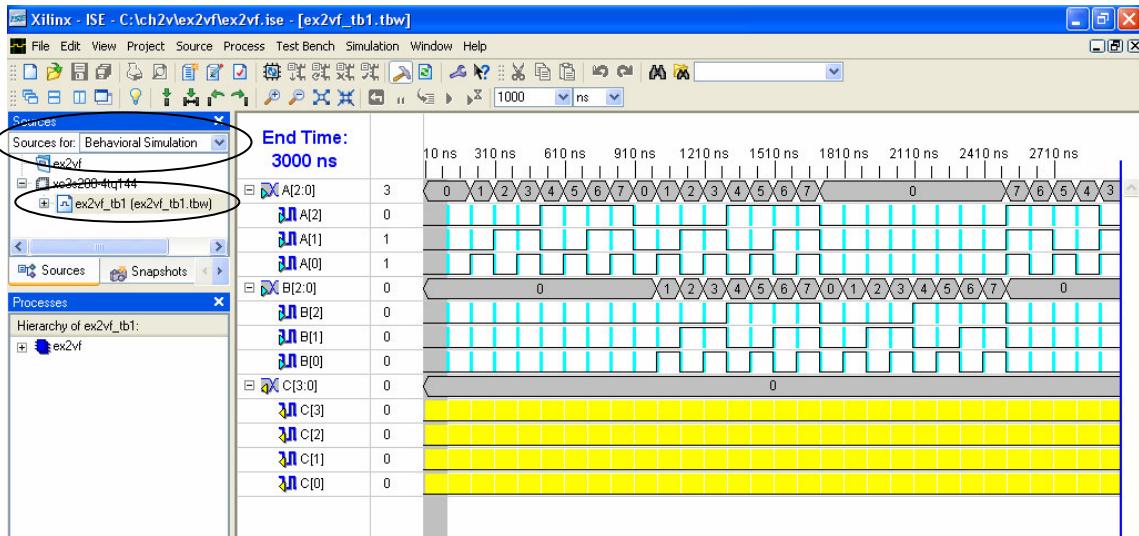
รูปที่ E2.21 หน้าต่าง Initial Timing and Clock Wizard

หมายเหตุ ในทางปฏิบัตินั้นเมื่อเราป้อนอินพุตให้กับวงจรต่างๆ แล้วจะไม่ได้อ่านพุททันที เวลาที่รอไปจนกระทั่งได้อ่านพุท อกมาเรียกว่า Propagation delay time ซึ่งเป็นเวลาล่าช้าที่เกิดในวงจร ในรูปที่ E2.21 นั้นเราจะมีเวลาล่าช้าที่เท่ากับ 10 ns

2) ทำการป้อนอินพุต A[2:0] และ B[2:0] เข้า การป้อนอินพุต A[2:0] ที่ให้เริ่มป้อนลักษณะรูปคลื่น (Waveform) โดยคลิกเครื่องหมาย “+” หน้า A[2:0] จะเป็น “-” แล้วจะปรากฏถ้า A[2], A[1] และ A[0] จากนั้นจึงป้อนลักษณะรูปคลื่นในแต่ละถ้าจาก A[0], A[1] และ A[2] ตามลำดับ เช่น ถ้าป้อน A[2] ที่สามารถทำได้โดยคลิกที่พื้นที่สีขาวที่อยู่บริเวณระหว่างแนบสีฟ้าอ่อน (คลิกตรงช่องที่ปลายลูกศรซึ่ง) จากนั้นจึงป้อน B[0], B[1] และ B[2] ตามลำดับ เมื่อป้อนลักษณะรูปคลื่นทั้งหมดเสร็จแล้วจะได้ดังรูปที่ E2.22 แล้วจึงคลิก บันทึกไฟล์ จากนั้นคลิก แล้วคลิกที่ Behavior Simulation เพื่อเพิ่มไฟล์ ex1vcxl_tb1 เข้าไปโดยอัตโนมัติังรูปที่ E2.23 คลิก (สีดำ) เพื่อปิดโปรแกรมและกลับไปที่หน้าต่าง Xilinx-ISE (ครูปที่ E2.24)

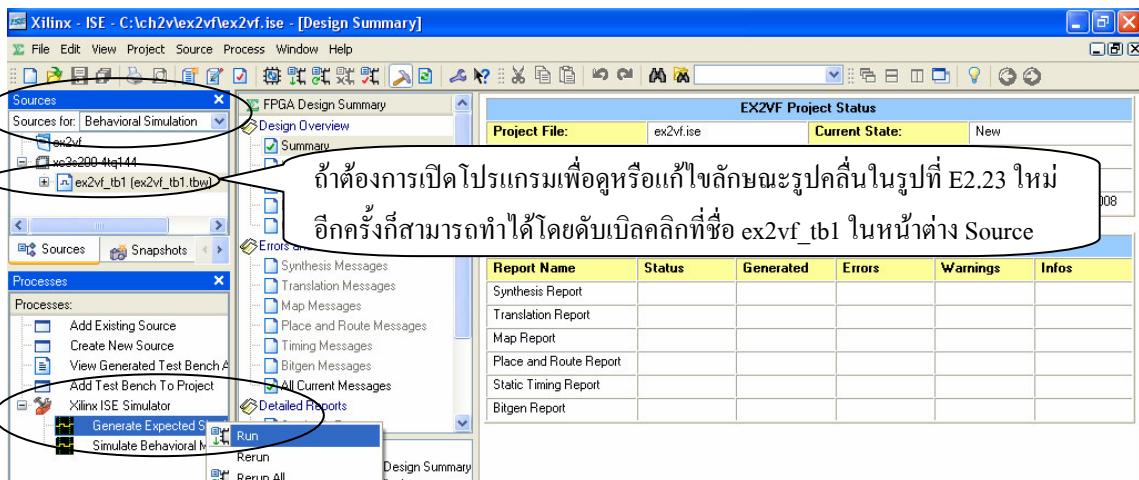


รูปที่ E2.22 หน้าต่างสำหรับกำหนดสัญญาณต่างๆ ที่ป้อนให้กับอินพุตของวงจรที่เราต้องการทดสอบ

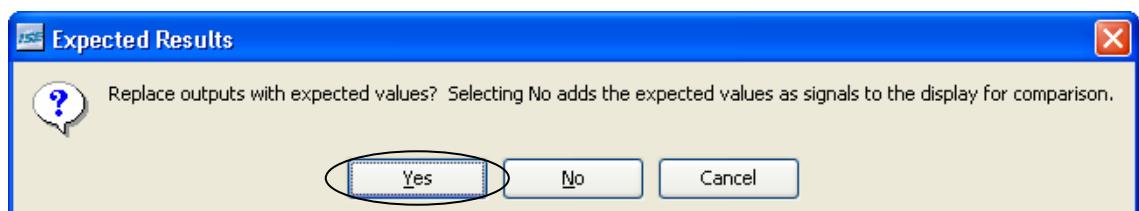


รูปที่ E2.23 หน้าต่างสำหรับกำหนดสัญญาณต่างๆที่ป้อนให้กับอินพุตของวงจรที่เราต้องการทดสอบ

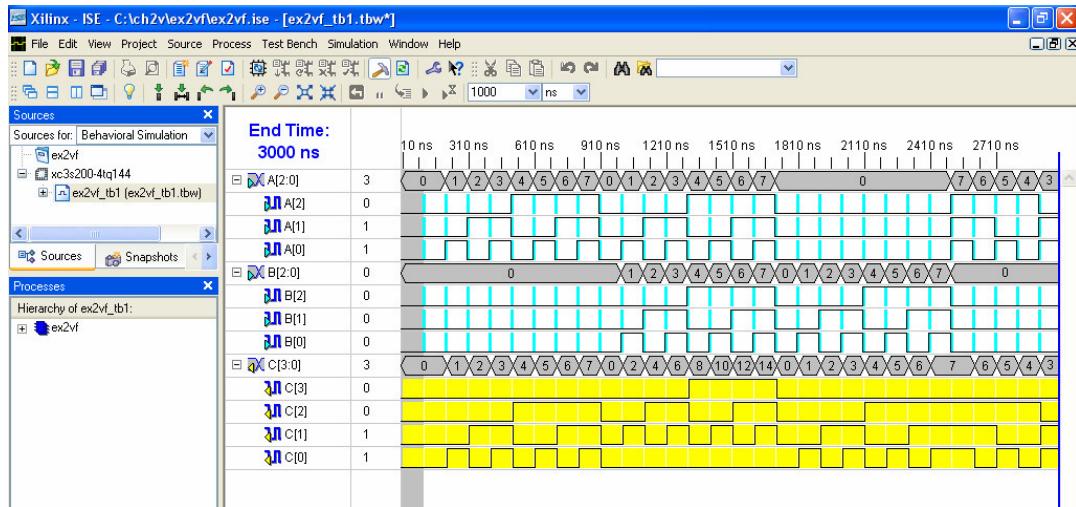
3) ขั้นตอน Create a self-checking test bench เพื่อตรวจสอบสัญญาณเอกสารพุดโดยอัตโนมัติ เริ่มที่หน้าต่าง Xilinx-ISE คลิก แล้ว คลิกที่ແນບ Behavior Simulation และคลิกเลือกไฟล์ชื่อ ex2vf_tb1 ในหน้าต่าง Source คลิกเครื่องหมาย “+” หน้า Xilinx ISE Simulator และคลิกขวาที่ແນບ Generate Expected Simulation Results !!แล้วคลิก Run ในหน้าต่าง Processes ดังรูปที่ E2.24 แล้วจะได้ดังรูปที่ E2.25 เมื่อคลิก Yes แล้วจะได้ดังรูปที่ E2.26 คลิก บันทึกไฟล์ คลิก (สีดำ) เพื่อปิดโปรแกรมและกลับไปที่หน้าต่าง Xilinx-ISE อีกรอบ



รูปที่ E2.24 หน้าต่าง Xilinx-ISE

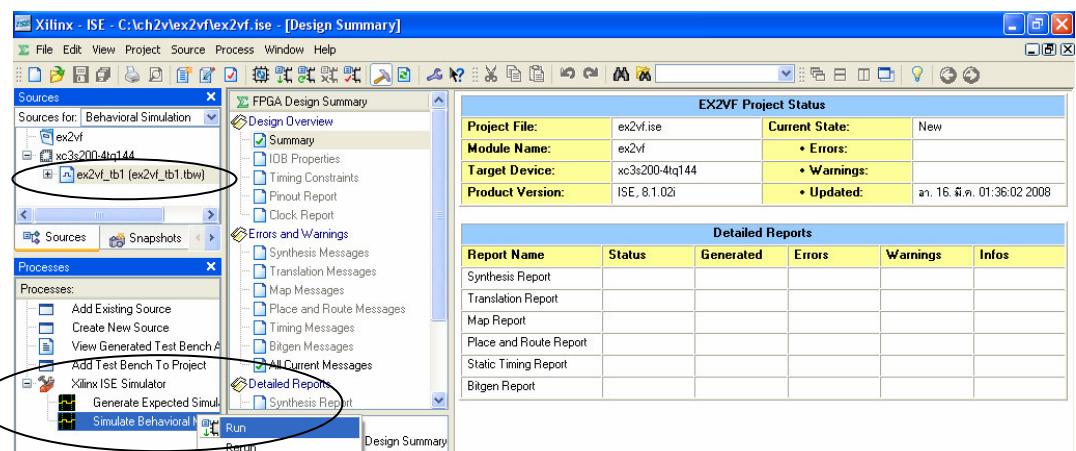


รูปที่ E2.25 หน้าต่าง Expected Results

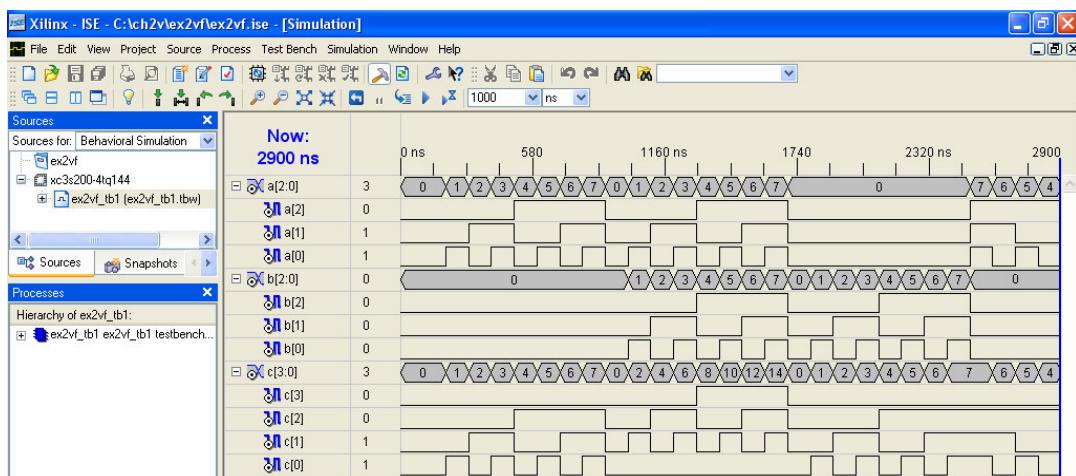


รูปที่ E2.26 ผลที่ได้ในขั้นตอน Create a self-checking test bench

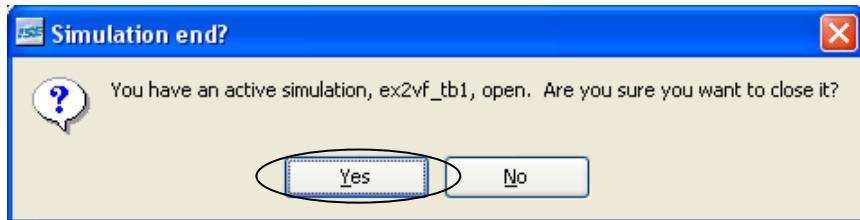
4) ในรูปที่ E2.26 ถ้าได้อ่าดพุตตรงตามที่ออกแบบก็จะทำ Behavioral simulation โดยคลิกขวาที่ Simulate Behavioral Model แล้วคลิก Run ดังรูปที่ E2.27 และจะได้ดังรูปที่ E2.28 ซึ่งการ Simulation นี้จะไม่นำผลของเวลาล่าช้าหรือ Delay ต่างๆ มาคิด จึงเป็นการตรวจสอบความถูกต้องขั้นต้นเท่านั้น คลิก  (สีดำ) ปิดโปรแกรมแล้วคลิก Yes ในรูปที่ E2.29 เพื่อกลับไปที่หน้าต่าง Xilinx-ISE จากนั้นคลิก  แล้วคลิกที่ Synthesis/Implementation ดังรูปที่ E2.30 เพื่อเตรียมพร้อมที่จะทำขั้นตอนต่อไป



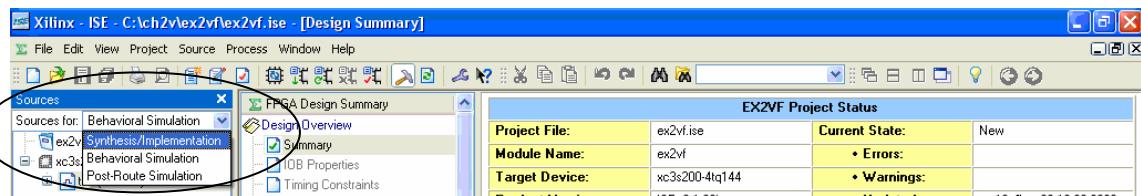
รูปที่ E2.27 ขั้นตอน Behavioral simulation



รูปที่ E2.28 แสดงผล Behavioral simulation ของวงจรบวก 3 บิต (ซึ่งในขั้นตอนนี้จะไม่นำผลของเวลาล่าช้ามาคิด)



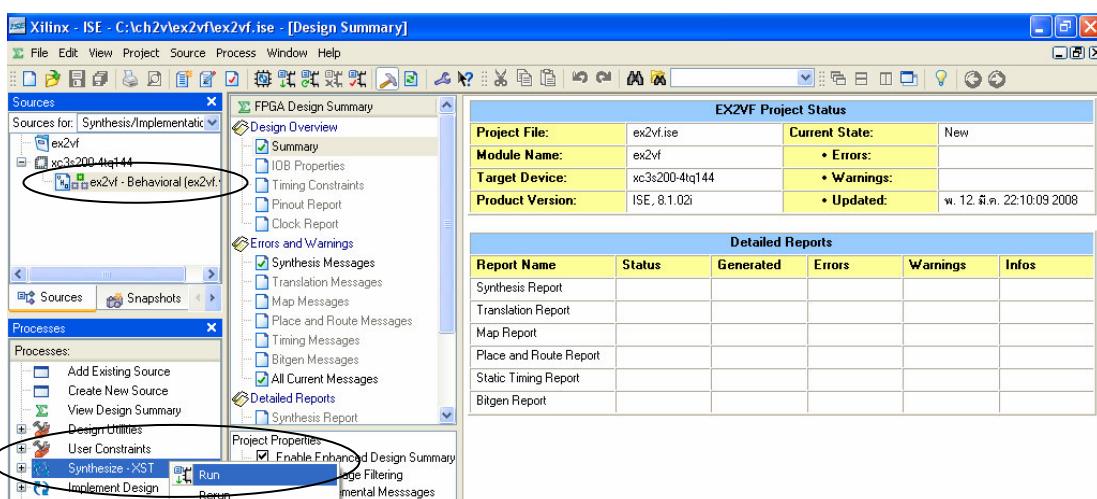
รูปที่ E2.29 หน้าต่าง Simulation end



รูปที่ E2.30 หน้าต่าง Xilinx-ISE

2.17.3 การสังเคราะห์วงจร (Design synthesis)

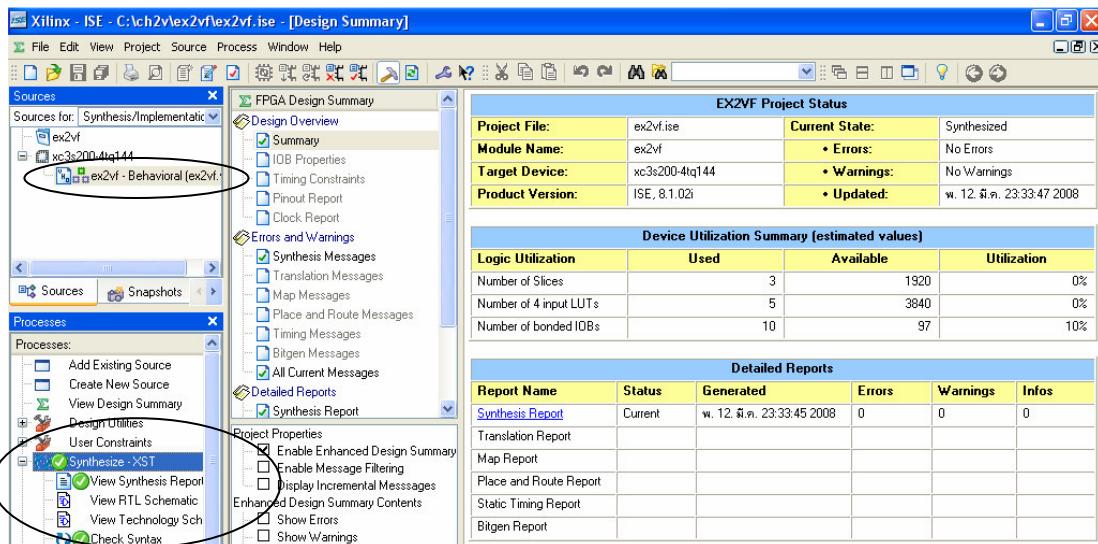
1) ขั้นตอนสังเคราะห์วงจร ให้คลิกที่ชื่อไฟล์ ex2vf ในหน้าต่าง Source แล้วเลือกมาสู่ไปที่หน้าต่าง Processes คลิกขวาแล้วคลิก Run ที่ Synthesize-XST ดังรูปที่ E2.31 (หรือดับเบิลคลิก) เสร็จแล้วถ้าได้ หรือ ดังรูปที่ E2.32 จะถือว่าสังเคราะห์วงจรผ่าน ซึ่งถ้าไม่ต้องการคุณรายละเอียดผลการสังเคราะห์วงจรก็ให้ข้ามไปทำหัวข้อ 2.17.4 ขั้นตอน Design Implementation



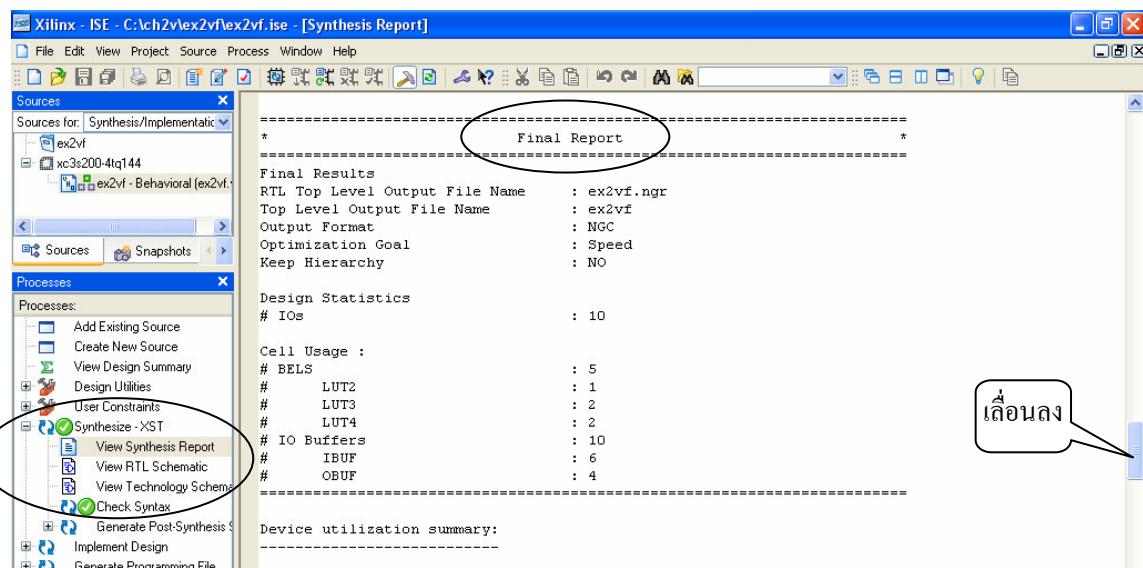
รูปที่ E2.31 ขั้นตอนสังเคราะห์วงจร

ความหมายของเครื่องหมายต่างๆ เป็นดังนี้ คือ = Running, = Up-to-date(ถือว่าผ่าน), = Warnings Reported (ถือว่าผ่าน), = Errors Reported ('ไม่ผ่าน'), = Out-of-Date (ให้ทำขั้นตอนนี้ซ้ำอีกครั้ง)

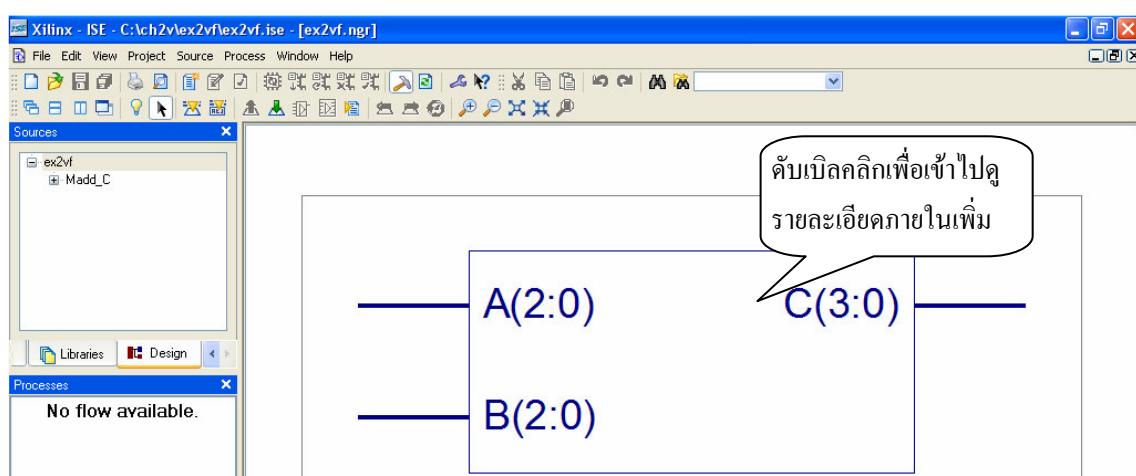
- 2) ถ้าต้องการดูรายงานผลการสังเคราะห์วงจร คลิก “+” ที่หน้า Synthesize-XST ให้เป็น “-” คลิกขวาที่ View Synthesis Report แล้วคลิก Rerun ในรูปที่ E2.32 เสร็จแล้วให้เลื่อนบาร์ลงไปดูด้านล่างที่ Final Reports ดังรูปที่ E2.33 ซึ่งแสดงรายการอุปกรณ์พื้นฐานที่นำมาสร้างวงจรที่เราออกแบบ จากนั้นคลิก (สีดำ)ที่มุมบนขวาเพื่อกลับไปที่หน้าต่าง Xilinx-ISE อีกครั้ง
- 3) ถ้าต้องการดู View RTL Schematic ให้คลิกขวาที่ View RTL Schematic แล้วคลิก Run ในรูปที่ E2.32 เสร็จแล้วให้ดับเบิลคลิกที่รูป Schematic ในรูปที่ E2.34 แล้วจะได้ดังรูปที่ E2.35 ซึ่งแสดงรายการอุปกรณ์พื้นฐานที่นำมาสร้างวงจรที่เราออกแบบ จากนั้นคลิก (สีดำ)ที่มุมบนขวาเพื่อกลับไปที่หน้าต่าง Xilinx-ISE อีกครั้ง



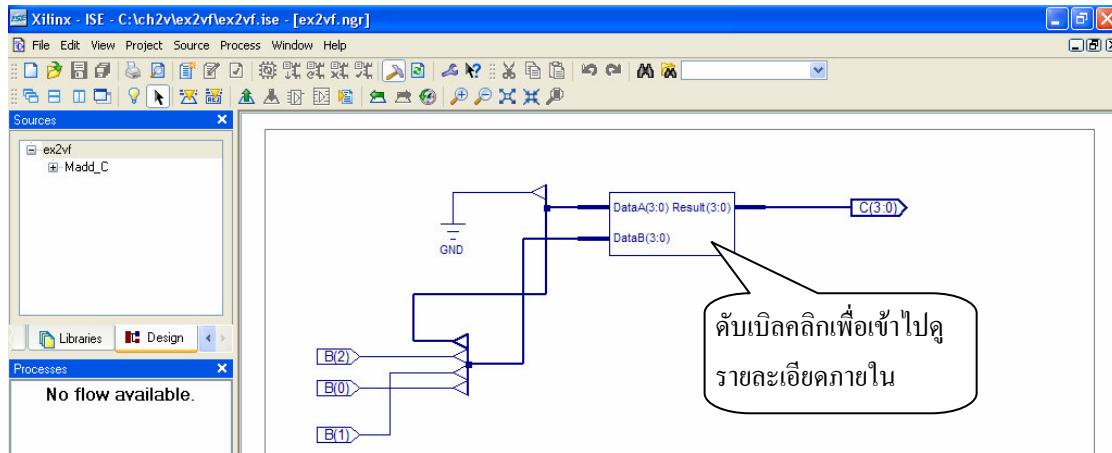
รูปที่ E2.32 หน้าต่าง Processes เป็นสังเคราะห์ของจริงเรียบร้อยแล้ว



รูปที่ E2.33 Synthesis Report

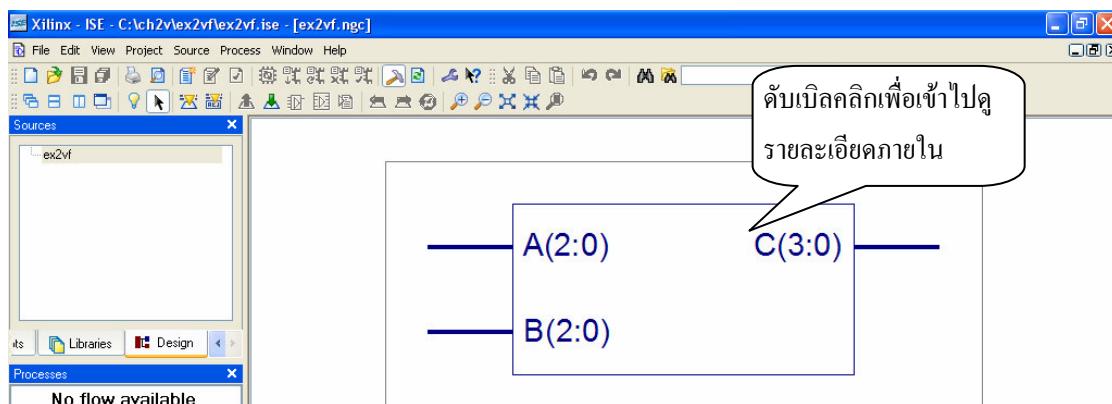


รูปที่ E2.34 View RTL Schematic

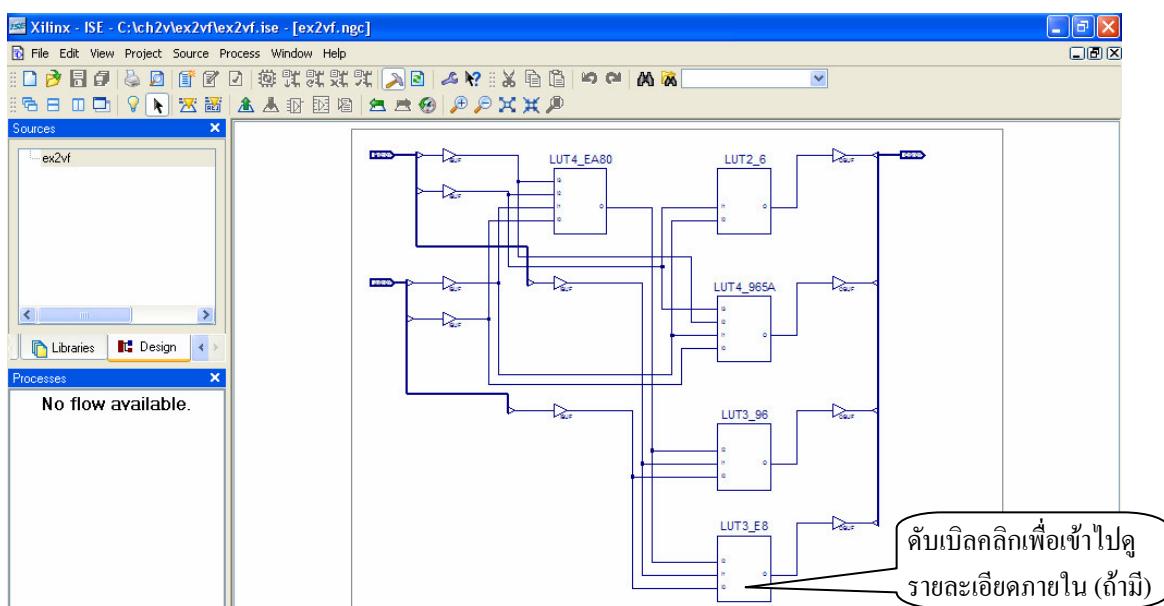


รูปที่ E2.35 View RTL Schematic ที่บอกรายละเอียดภายใน

4) ถ้าต้องการดู View Technology Schematic ให้คลิกขวาที่ View Technology Schematic แล้วคลิก Run ในรูปที่ E2.32 เสร็จแล้วให้ดับเบิลคลิกที่รูป Schematic ในรูปที่ E2.36 แล้วจะได้ดังรูปที่ E2.37 ซึ่งแสดงรายการอุปกรณ์พื้นฐานที่นำมาสร้างวงจรที่เรารอออกแบบ จากนั้นคลิก **X** (สีดำ) ที่มุมบนขวาเพื่อกลับไปที่หน้าต่าง Xilinx-ISE อีกครั้ง



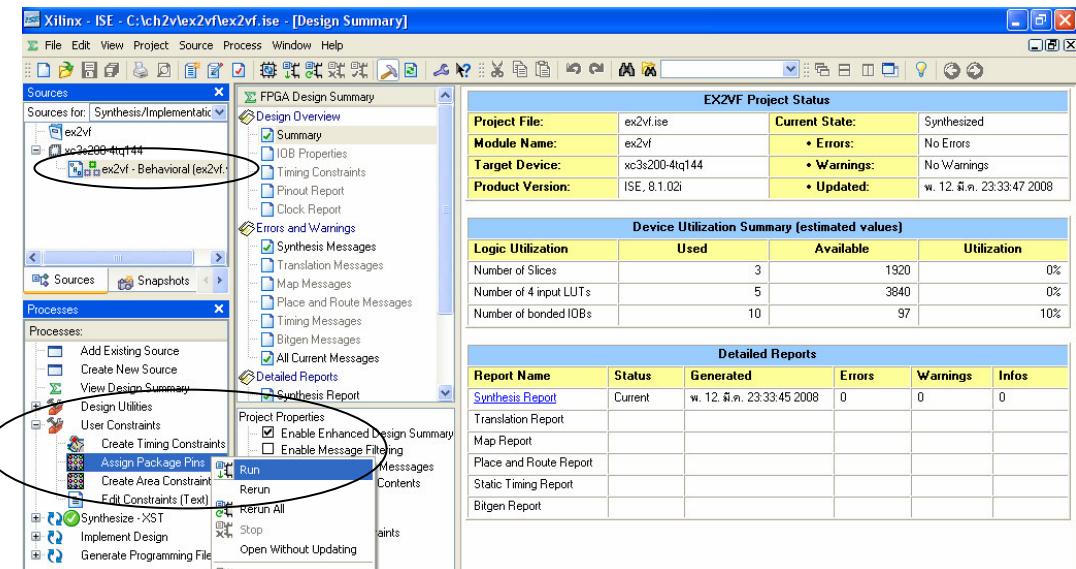
รูปที่ E2.36 View Technology Schematic



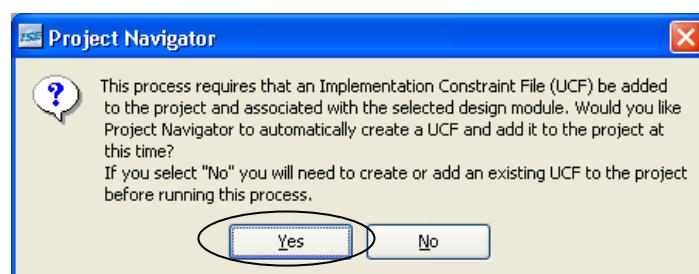
รูปที่ E2.37 View Technology Schematic ที่บอกรายละเอียดภายใน

2.17.4 Design Implementation

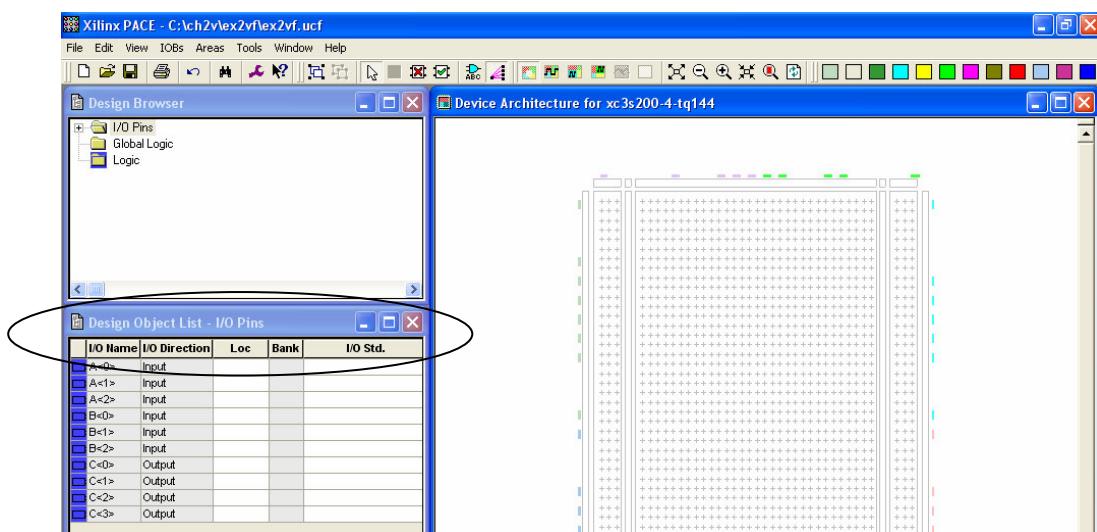
- 1) ขั้นตอน Implementation constraint file (UCF) เพื่อสร้างไฟล์ที่ระบุว่าจะให้อินพุต/เอาต์พุตต่อเข้ากับขา (Pin) ใดของ FPGA
- a) ให้คลิกที่ชื่อไฟล์ ex2vf ในหน้าต่าง Source แล้วเลือกมาลากไปที่หน้าต่าง Processes คลิก “+” หน้า User Constraints ในหน้าต่าง Processes จะเป็น “-” คลิกขวาที่ Assign Package Pins แล้วคลิก Run ดังรูปที่ E2.38 จะได้หน้าต่างดังรูปที่ E2.39 คลิก Yes เพื่อยืนยันที่จะสร้าง Implementation constraint file (UCF) โดยอัตโนมัติแล้วจะได้หน้าต่าง Xilinx-PACE ดังรูปที่ E2.40



รูปที่ E2.38 ขั้นตอน Assign Package Pins



รูปที่ E2.39 หน้าต่าง Project Navigator

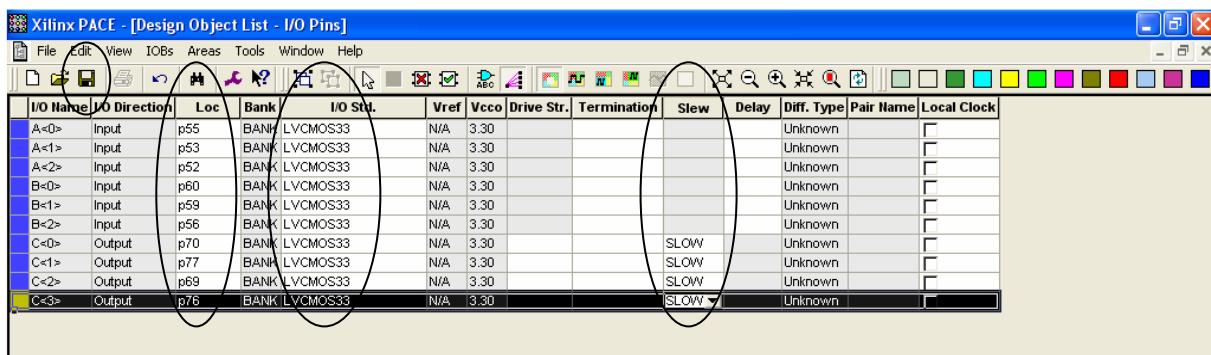


รูปที่ E2.40 หน้าต่าง Xilinx-PACE

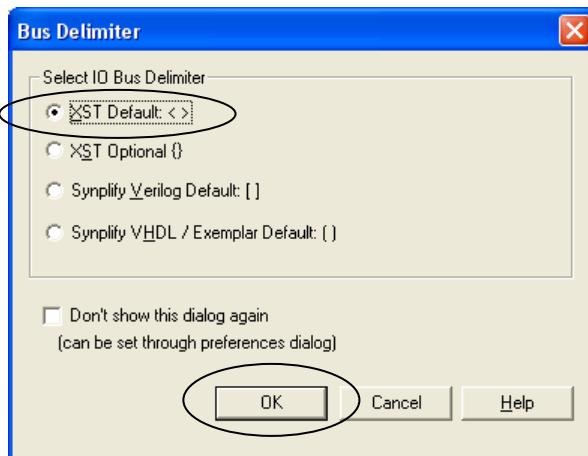
b) การกำหนดขาสัญญาณ ในกรณีนี้เราใช้บอร์ด FPGA Discovery-III XC3S200F (หรือ FPGA Discovery-III XC3S200F4) จะต้องกำหนดค่าอินพุต/เอาต์พุตของ FPGA ให้สอดคล้องกับอุปกรณ์ที่เตรียมไว้ที่บอร์ดทดลองตามตารางที่ 1.7 ในบทที่ 1 ซึ่งเราใช้ Dip SW1-Dip SW3 เป็นอินพุตของ A, Dip SW4-Dip SW6 เป็นอินพุตของ B และใช้ LED L0-L3 เป็นเอาต์พุตของ C กล่าวคือ

$$\begin{aligned}
 A(2) &= \text{Dip SW1} = \text{INPUT} = p52 & B(2) &= \text{Dip SW4} = \text{INPUT} = p56 & C(3) &= L3 = \text{OUTPUT} = p76 \\
 A(1) &= \text{Dip SW2} = \text{INPUT} = p53 & B(1) &= \text{Dip SW5} = \text{INPUT} = p59 & C(2) &= L2 = \text{OUTPUT} = p69 \\
 A(0) &= \text{Dip SW3} = \text{INPUT} = p55 & B(0) &= \text{Dip SW5} = \text{INPUT} = p60 & C(1) &= L1 = \text{OUTPUT} = p77 \\
 &&&& C(0) &= L0 = \text{OUTPUT} = p70
 \end{aligned}$$

ที่หน้าต่าง Design Object List–I/O Pins ในรูปที่ E2.40 คลิก เพื่อขยายหน้าต่างให้ใหญ่ขึ้น จากนั้นกำหนดอินพุต และเอาต์พุตลงในคอลัมน์ Loc กำหนดชนิดอินพุต/เอาต์พุตตามครรภานเป็น CMOS 3.3V (LVCMOS33) หรือ Low Voltage TTL 3.3V (LVTTL) ที่ได้ โดยระดับแรงดันอินพุต/เอาต์พุตของ LVCMOS33 จะสูงกว่า LVTTL เล็กน้อยแต่สามารถเข้ากันได้หรือต่อ กันได้โดยตรง ในตัวอย่างนี้จะใช้ CMOS 3.3V เนื่องจากแรงดันเอาต์พุตสูงกว่าจึงต้องคลิกคอลัมน์ I/O Std. เป็น LVCMOS33 ทั้งหมด จากนั้นคลิกในคอลัมน์ Slew เป็น Slow เพื่อลดอสัมฤทธิ์และลดการสะท้อนในสายสัญญาณเอาต์พุต เสร็จแล้วจะได้ดังรูปที่ E2.41 คลิก บันทึกไฟล์แล้วจะปรากฏหน้าต่าง Bus Delimiter ขึ้นมา ให้คลิกเลือกที่ XST Default ดังรูปที่ E2.42 และคลิก OK จากนั้นคลิก (สีแดง) เพื่อปิดหน้าต่าง Xilinx–PACE และกลับไปที่หน้าต่าง Xilinx–ISE

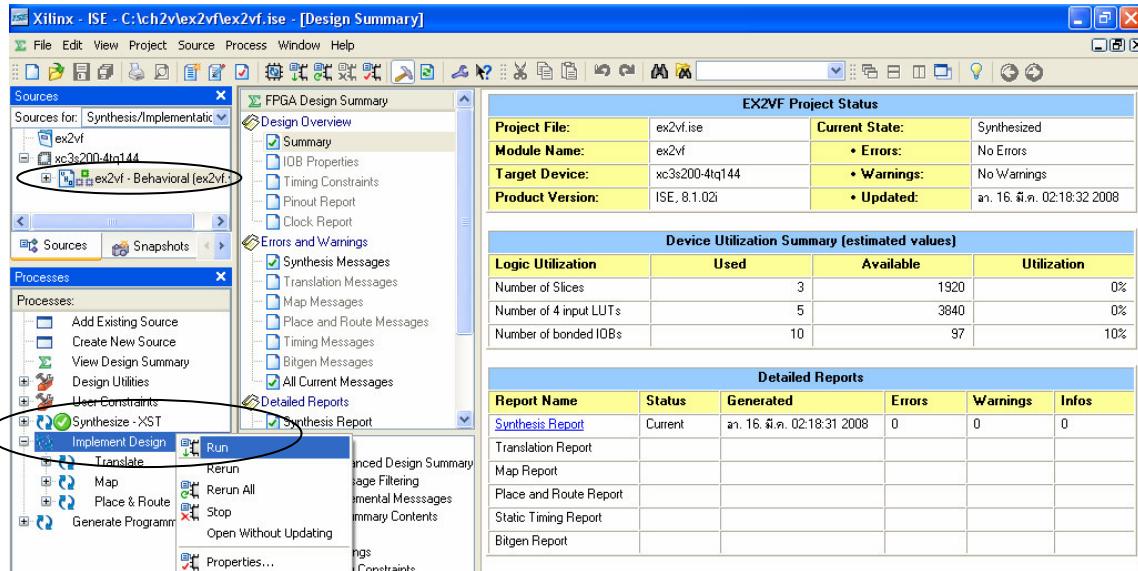


รูปที่ E2.41 การกำหนดอินพุต/เอาต์พุตที่หน้าต่างย่อย Design Object List–I/O Pins ของหน้าต่าง Xilinx–PACE

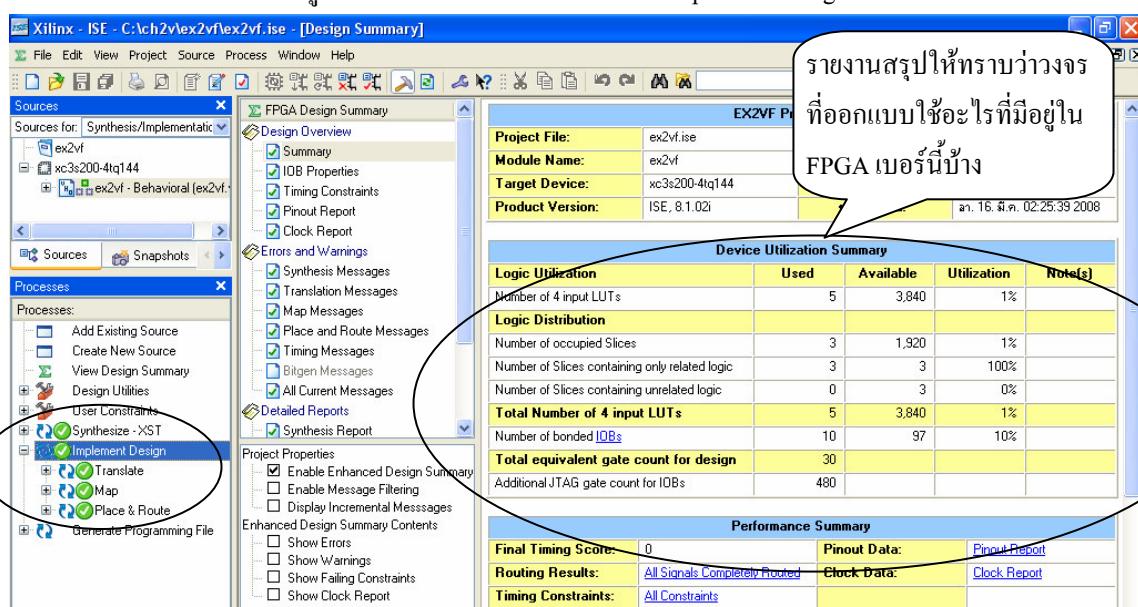


รูปที่ E2.42 หน้าต่าง Bus Delimiter

2) ขั้นตอน Implement Design ให้คลิกที่ชื่อไฟล์ ex2vf ในหน้าต่าง Source แล้วเลื่อนมาสู่ไปที่หน้าต่าง Processes คลิก “+” หน้า Implement Design ในหน้าต่าง Processes จะเป็น “-” คลิกขวาที่ Implement Design และคลิก Run ดังรูปที่ E2.43 แล้วรอสักครู่ เมื่อแล้วเสร็จจะได้ ✓ หรือ ⚠ ดังรูปที่ E2.44 จึงจะถือว่าขั้นตอน Implement Design ผ่าน



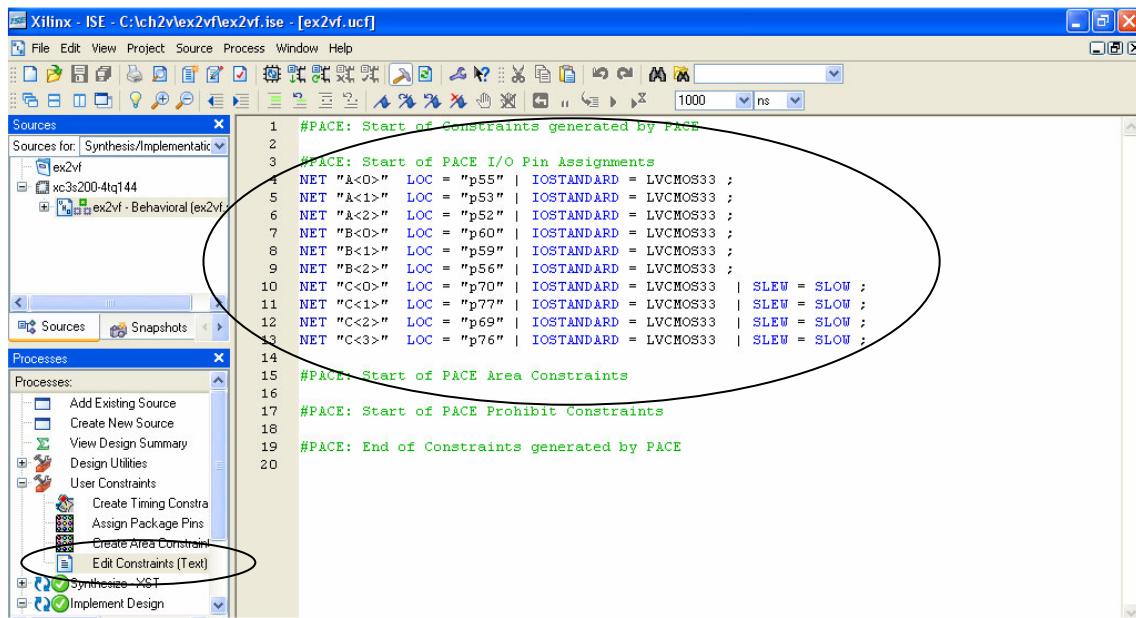
รูปที่ E2.43 หน้าต่างสำหรับขั้นตอน Implement Design



รูปที่ E2.44 หน้าต่าง Processes เมื่อทำขั้นตอน Implement !เสร็จเรียบร้อยแล้ว

หมายเหตุ

- 1) ขั้นตอน Implement design นี้โปรแกรมจะทำ Translate, Map และ Place & Route
- 2) ถ้าผลที่ได้จากการทำ Implement แล้วได้ ✗ หน้า Implement Design (ไม่ผ่าน) และ ✗ หน้า Translate (ไม่ผ่าน) ให้คลิกขวาที่ Edit Constraints (Text) และคลิก Run แล้วจะได้ดังรูปที่ E2.45 จากนั้นให้ตรวจสอบว่ามีการกำหนดขา FPGA เกินหรือผิดหรือไม่ เมื่อแก้ไขเสร็จแล้วให้บันทึกไฟล์ แต่ถ้าไม่แน่ใจให้ลบไฟล์ใน Edit Constraints (Text) ทิ้งทั้งหมดแล้วทำการบันทึกไฟล์ จากนั้นให้ไปทำขั้นตอน Assign Package Pins อีกครั้ง เมื่อบันทึกไฟล์แล้วให้ทำขั้นตอน Implement Design ใหม่อีกครั้ง

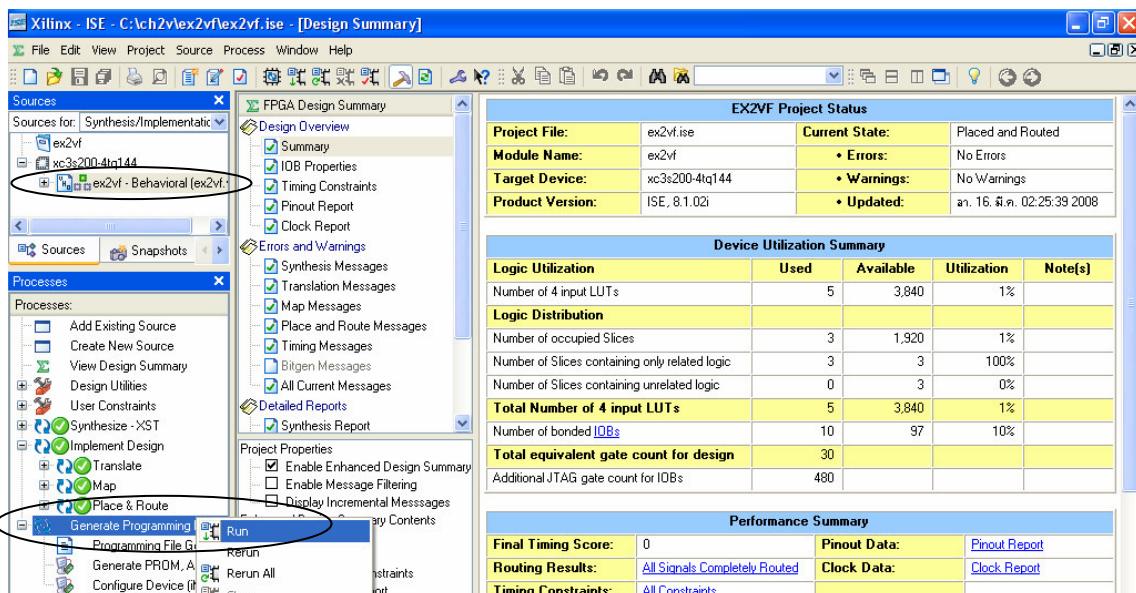


รูปที่ E2.45 หน้าต่าง Edit Constraints (Text)

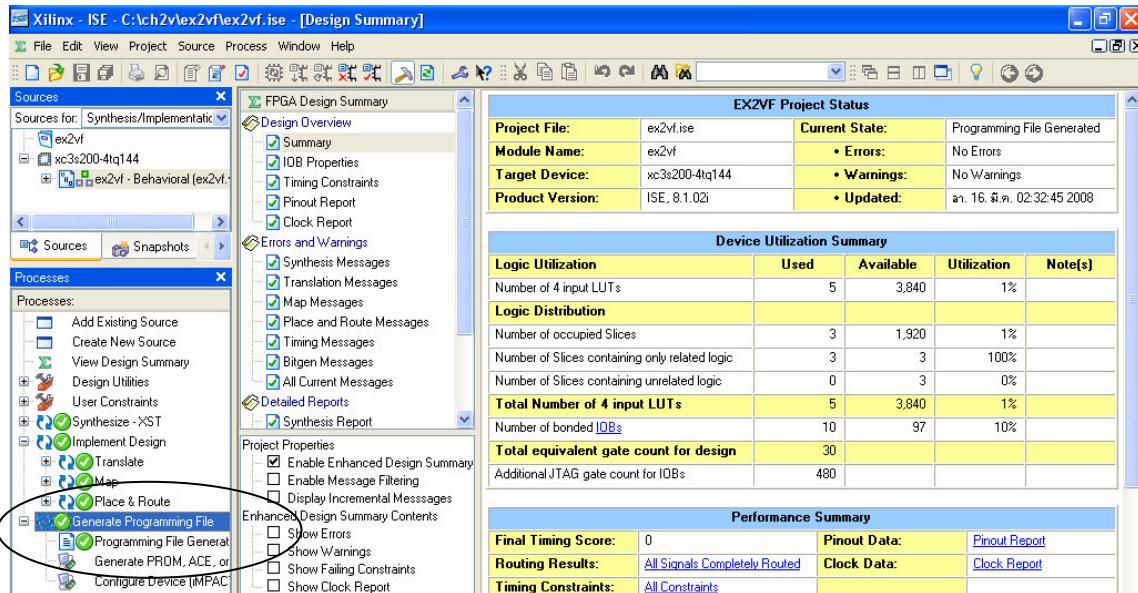
2.17.5 การโปรแกรมข้อมูลวงจรลงชิพ

1) ขั้นตอน Generate Programming File

คลิกที่ไฟล์ ex2vf-Behavioral ในหน้าต่าง Source และคลิกขวาที่ Generate Programming File ในหน้าต่าง Processes และคลิก Run ดังรูปที่ E2.46 (หรือดับเบิลคลิก) เมื่อแล้วเสร็จจะได้ดังรูปที่ E2.47



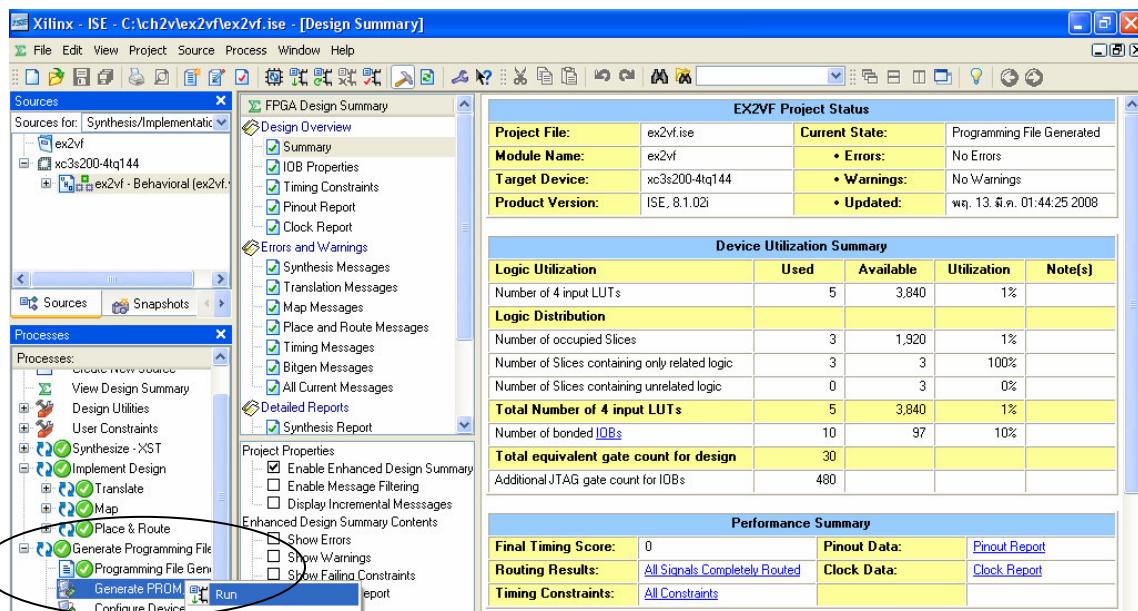
รูปที่ E2.46 ขั้นตอน Generate Programming File



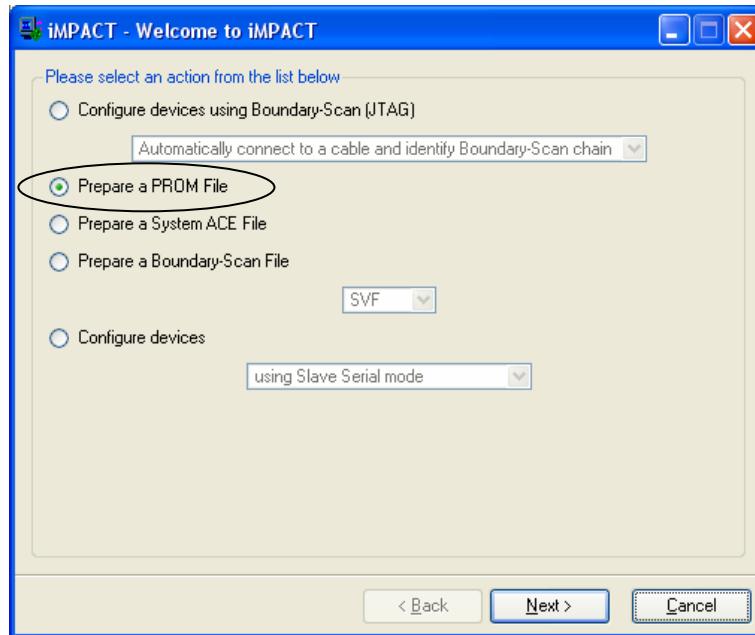
รูปที่ E2.47 เมื่อการทำ Generate Programming File เตรียมเรียบร้อยแล้ว

2) ขั้นตอนสร้างไฟล์ PROM

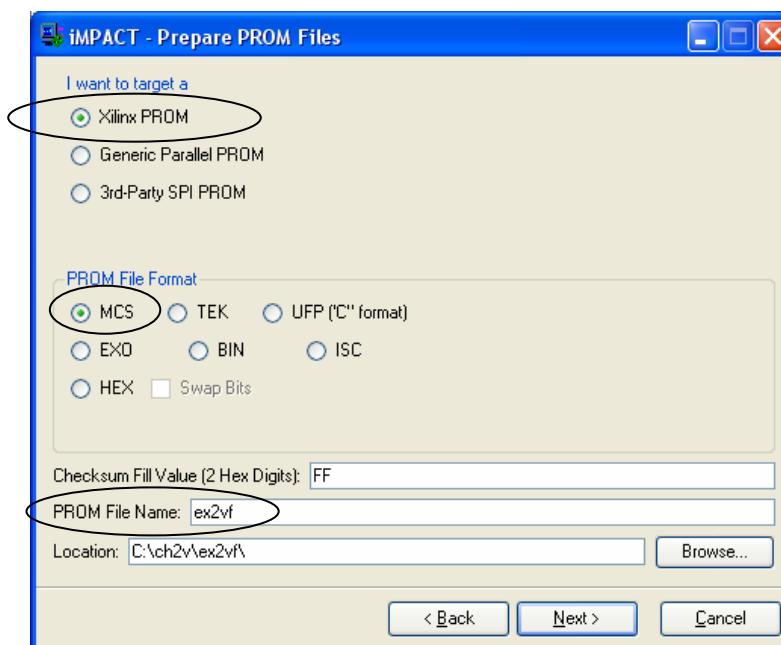
a) คลิกขวาที่ Generate PROM, ACE, or JTAG File ในหน้าต่าง Processes แล้วกด Run ดังรูปที่ E2.48 รอสักครู่ เมื่อได้หน้าต่างถัดไปแล้วคลิกช่อง Prepare a PROM File ดังรูปที่ E2.49 คลิก Next 1 ครั้ง จากนั้นพิมพ์ชื่อ ex2vf ดังในรูปที่ E2.50



รูปที่ E2.48 ขั้นตอน Generate PROM, ACE, or JTAG File

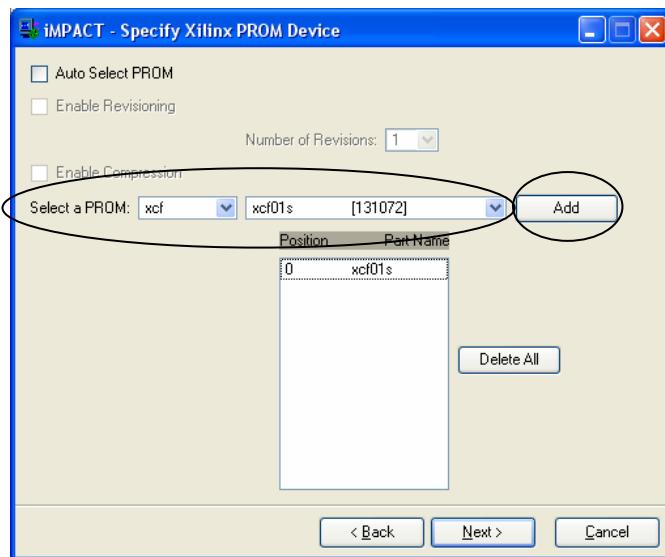


รูปที่ E2.49 หน้าต่าง iMPACT-Welcome to iMPACT

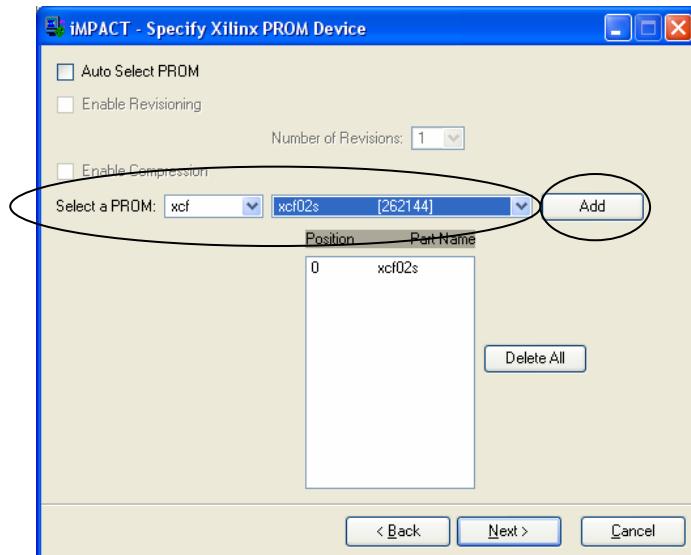


รูปที่ E2.50 พิมพ์ชื่อ ex2vf ที่ช่อง PROM File Name

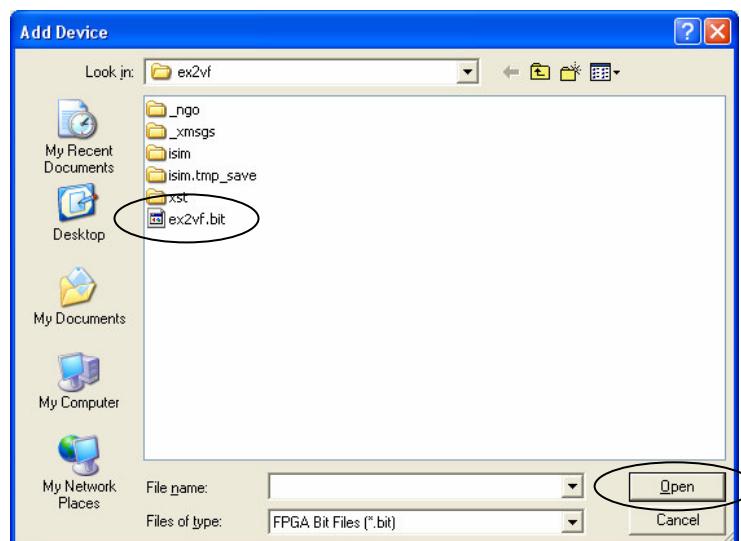
บ) จากรูปที่ E2.50 คลิก Next 1 ครั้ง ในกรณีที่ใช้บอร์ด FPGA Discovery-III XC3S200F จะใช้ PROM เบอร์ XCF01VO20C จึงต้องคลิกเลือก xcf เบอร์ xcf01s และคลิกปุ่ม Add แสดงดังรูปที่ E2.51 แต่ถ้าใช้บอร์ด FPGA Discovery-III XC3S200F4 จะใช้ PROM เบอร์ XCF02VO20C จึงต้องคลิกเลือก xcf เบอร์ xcf02s และคลิกปุ่ม Add แสดงดังรูปที่ E2.52 จากนั้นคลิก Next 1 ครั้ง คลิก Finish 1 ครั้ง คลิก OK และจะได้หน้าต่าง Add Device ดังรูปที่ E2.53 คลิกที่ไฟล์ชื่อ ex2vf.bit และคลิก Open แล้วจะได้ดังรูปที่ E2.54 คลิก No 1 ครั้งและคลิก OK 1 ครั้ง จากนั้นค้นเบิกคลิกที่ Generate File แล้วจะได้ดังรูปที่ E2.55 ในขั้นตอนนี้ถือว่าได้สร้างไฟล์ Flash PROM แล้วเสร็จ คลิก (สีแดง) เพื่อปิดโปรแกรม iMPACT และคลิก No ดังรูปที่ E2.56 เพื่อกลับไปที่หน้าต่าง Xilinx-ISE



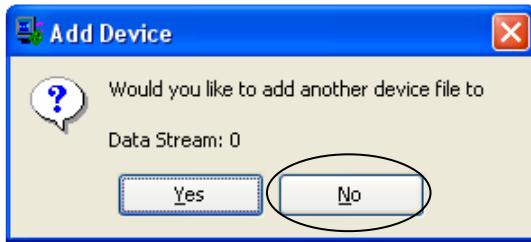
รูปที่ E2.51 หน้าต่าง iMPACT-Specify Xilinx PROM Drvce ในกรณีที่ใช้ PROM เปอร์ XCF01VO20C



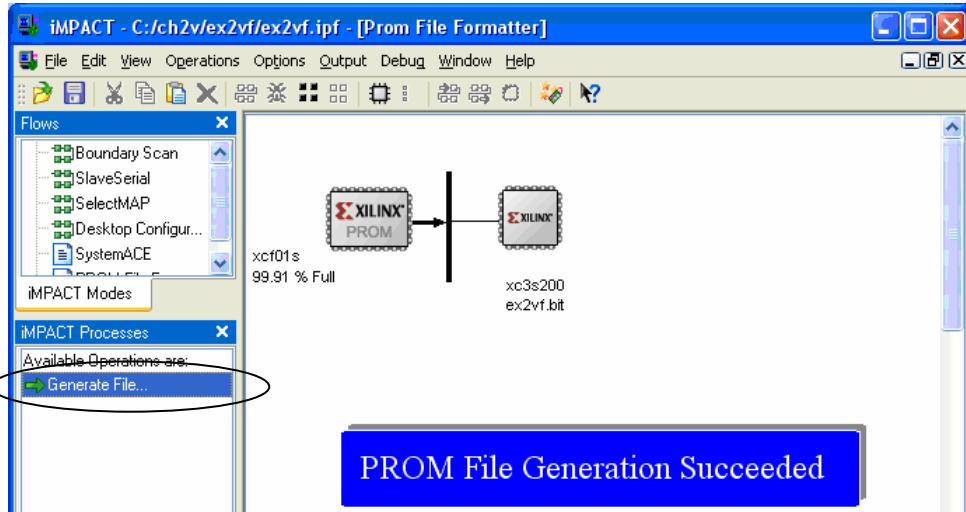
รูปที่ E2.52 หน้าต่าง iMPACT-Specify Xilinx PROM Drvce ในกรณีที่ใช้ PROM เปอร์ XCF02VO20C



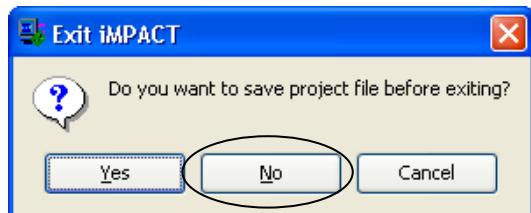
รูปที่ E2.53 หน้าต่าง Add Device



รูปที่ E2.54 หน้าต่าง Add Device



รูปที่ E2.55 หน้าต่าง iMPACT



รูปที่ E2.56 หน้าต่าง Exit iMPACT

ในการถักท่ออยู่ในขั้นตอนออกแบบหรืออยู่ระหว่างการวิจัยพัฒนา เราอาจจะไม่จำเป็นต้องทำขั้นตอนสร้างไฟล์ PROM และให้ขั้นตอนที่ 2) นี้ไปทำข้อถัดไป คือ ข้อ 3)

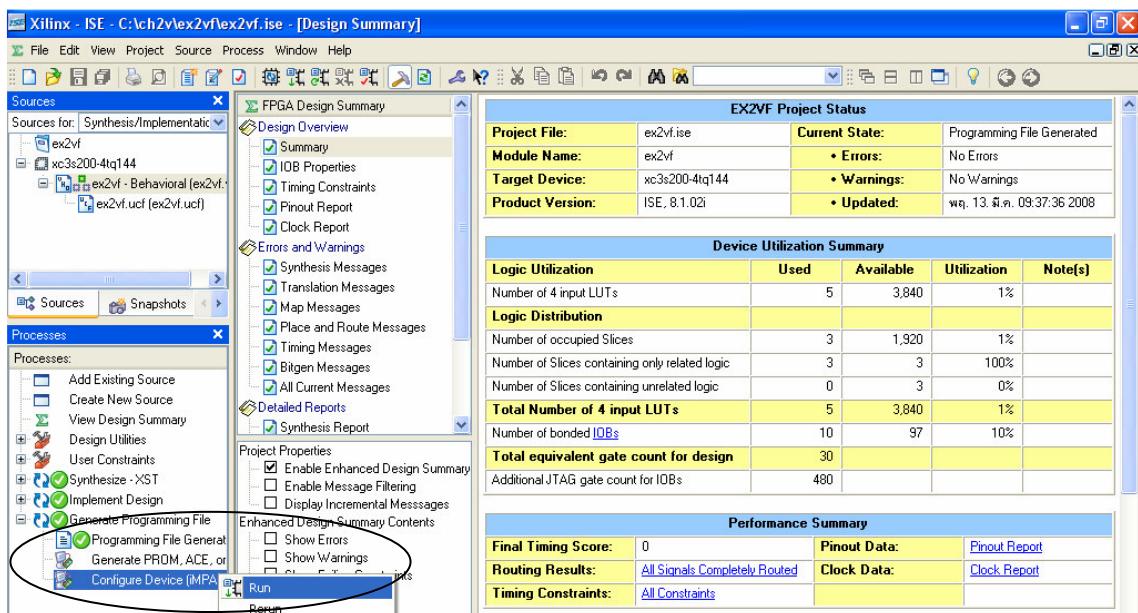
3) ขั้นตอนการโปรแกรมลง Flash PROM และ/หรือ FPGA

- a) ขั้นตอนการเซ็ตจัมเบอร์ J1 เพื่อเซ็ตค่าของ M0,M1,M2 โดยปกติจัมเบอร์ J1 บนบอร์ด FPGA Discovery-III XC3S200F จะถูกเซ็ต $M0,M1,M2 = "000"$ ไว้แล้ว ซึ่งเป็นโหมด Master serial (ดูตามตารางที่ 2.1) ข้อมูลวงจรใน Flash PROM จึงถูกดาวน์โหลดลง FPGA โดยอัตโนมัติทันทีที่เริ่มจ่ายไฟเลี้ยง การดาวน์โหลดไฟล์ข้อมูลวงจรใหม่ลง FPGA ผ่านทางสาย JTAG โดยไม่เซ็ตโหมดเป็น JTAG จึงอาจเกิดข้อผิดพลาดขึ้นได้ก็ตามคือ วงจรอาจจะทำงานไม่ตรงตามที่ออกแบบไว้เนื่องจากเป็นการดาวน์โหลดผิดโหมด การแก้ไขปัญหาวิธีที่ 1 ดาวน์โหลดไฟล์ข้อมูลวงจรใหม่ (นามสกุล.mcs) ลง Flash PROM ก่อน จากนั้นจึงใช้ไฟล์ วงจรเดียวกัน (นามสกุล.bit) ดาวน์โหลดลง FPGA วิธีที่ 2 ลงไฟล์ใน Flash PROM ทิ้งก่อนทำการดาวน์โหลดลง FPGA หรืออาจใช้วิธีที่ 3 โดยเซ็ตจัมเบอร์ J1 ในโหมด JTAG ซึ่งมีค่า $M0,M1,M2 = "101"$ ก่อนดาวน์โหลดไฟล์ข้อมูลวงจรใหม่ลง FPGA แต่ข้อเสียของวิธีนี้ คือ เมื่อเริ่มจ่ายไฟเลี้ยงจะเสื่อมไปเมื่อการดาวน์โหลดข้อมูลใน Flash PROM ลง FPGA โดยอัตโนมัติ

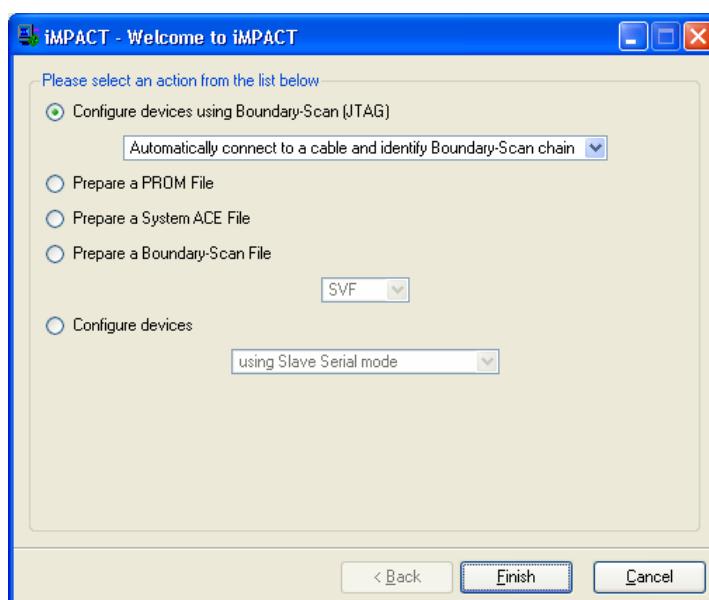
ตารางที่ 2.1 รายการเซตโภมดในการโปรแกรมลง FPGA

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

b) ขั้นตอนดาวน์โหลดไฟล์ลง Flash PROM และ FPGA ให้ต่อสาย JTAG เข้ากับพอร์ตบนนาของคอมพิวเตอร์และขั้ว JTAG ที่บอร์ดทดลอง FPGA Discovery-III XC3S200F และวิ่งไฟเลี้ยงเข้าบอร์ดโดยต่อเข้ากับอะแดปเตอร์ 9 VDC จากนั้นคลิกขวาที่ແນບ Configure Device (iMPACT) และคลิก Run และคงในรูปที่ E2.57 รอสักครู่แล้วจะได้หน้าต่างดังรูปที่ E2.58

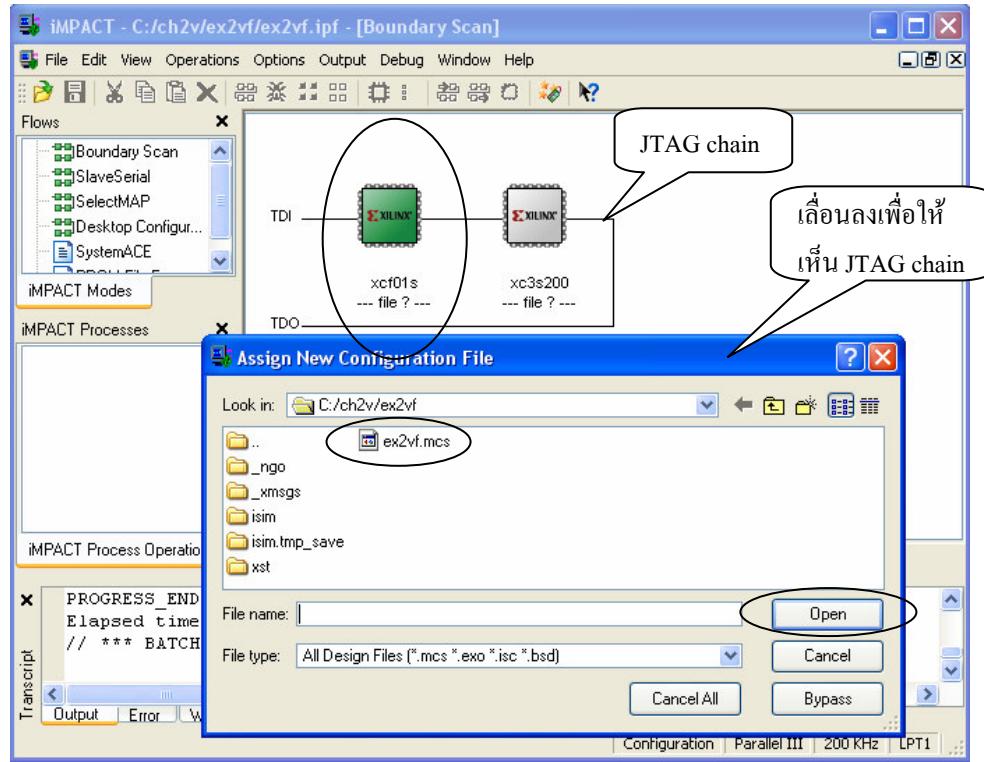


รูปที่ E2.57 แสดงขั้นตอน Configure Device (iMPACT)

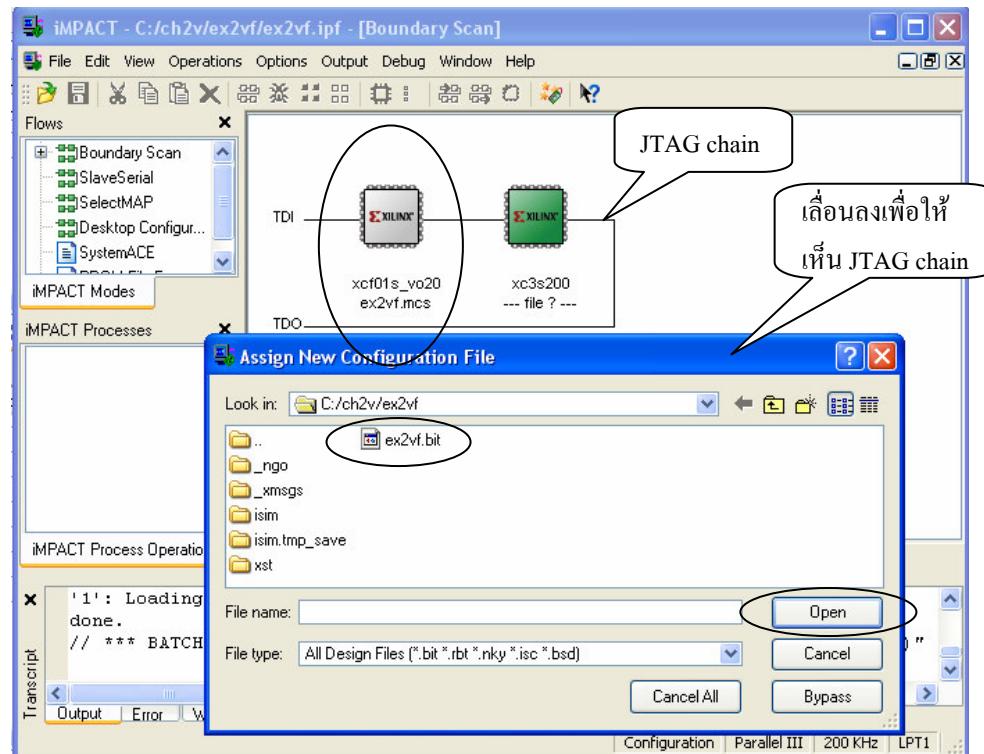


รูปที่ E2.58 หน้าต่าง iMPACT

c) จากรูปที่ E2.58 คลิก Finish แล้วจะได้ดังรูปที่ E2.59 คลิกที่ไฟล์ ex2vf.mcs (PROM configuration file) คลิก Open (หรือให้กดลิ๊ก Cancel หากไม่ได้สร้างไฟล์ PROM ในข้อ 2) เสร็จแล้วจะได้หน้าดังรูปที่ E2.60 คลิกที่ไฟล์ ex2vf.bit (FPGA configuration file) คลิก Open แล้วจะได้หน้าต่าง Warning ดังรูปที่ E2.61 จากนั้นให้คลิก OK แล้วจะได้หน้าต่าง iMPACT



รูปที่ E2.59 หน้าต่าง Assign New Configuration เพื่อเตรียมข้อมูลที่จะดาวน์โหลดลง PROM

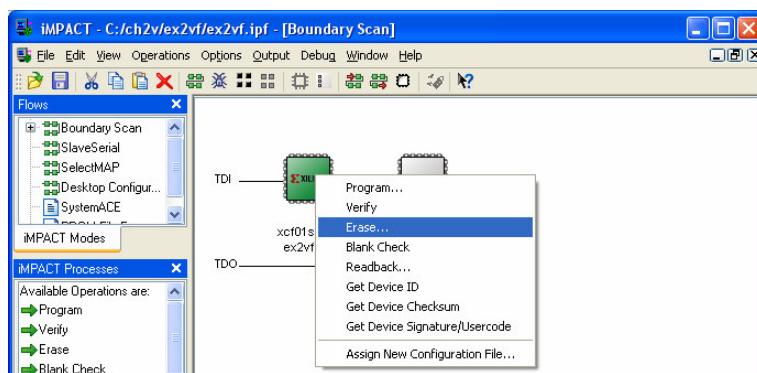


รูปที่ E2.60 หน้าต่าง Assign New Configuration เพื่อเตรียมข้อมูลที่จะดาวน์โหลดลง FPGA

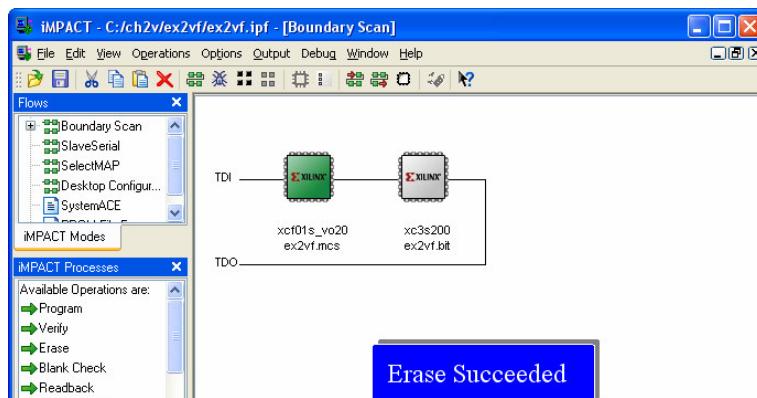


รูปที่ E2.61 หน้าต่าง Warning

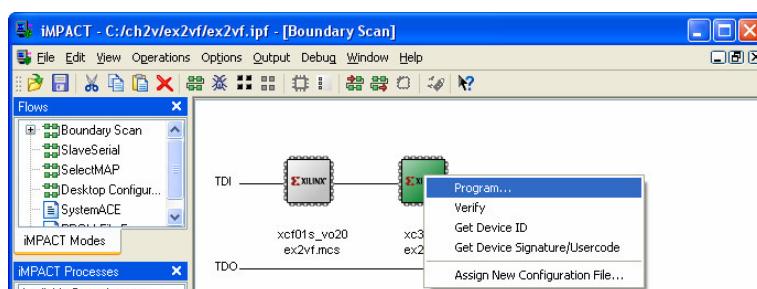
d) การดาวน์โหลดข้อมูลวงจรลง FPGA เนื่องจากจัมเปอร์ J1 ของบอร์ดทดลองถูกเซตอยู่ในโหมด Master serial ตั้งนั้นก่อนโปรแกรมจะต้องลบ (Erase) ข้อมูลวงจรที่เคยโปรแกรมลง Flash PROM ทิ้ง คลิกขวาที่รูป Flash PROM (สีเขียว) และคลิก Erase ดังรูปที่ E2.62 เสร็จแล้วจะได้ดังรูปที่ E2.63 จากนั้นคลิกขวาที่รูป FPGA จนเป็นสีเทาและคลิก Program ดังรูปที่ E2.64 แล้วจะได้หน้าต่างดังภาพ คลิก OK เสร็จแล้วจะได้ดังรูปที่ E2.65 ถ้าดาวน์โหลดไม่ผ่านเพื่อความแน่ใจการดาวน์โหลดซ้ำ ถ้าดาวน์โหลดหายครั้งไม่ผ่านให้ดึงแจ็คไฟเลี้ยงออกแล้วเสียบแจ็คกลับเข้าไปใหม่ แล้วทำการดาวน์โหลดซ้ำ



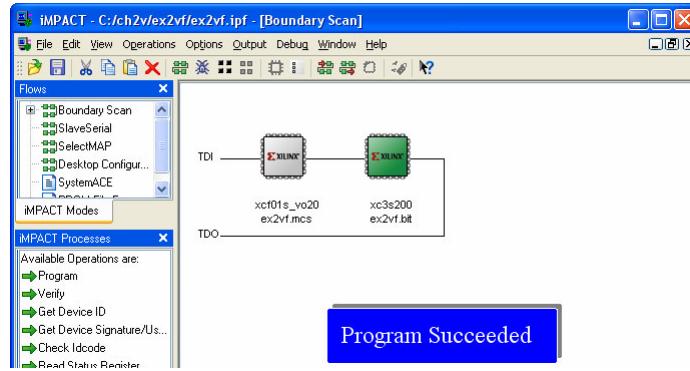
รูปที่ E2.62 ขั้นตอนลบ (Erase) ข้อมูลวงจรที่เคยโปรแกรมลง PROM ทิ้ง



รูปที่ E2.63 ลบ (Erase) เรียบร้อยแล้ว

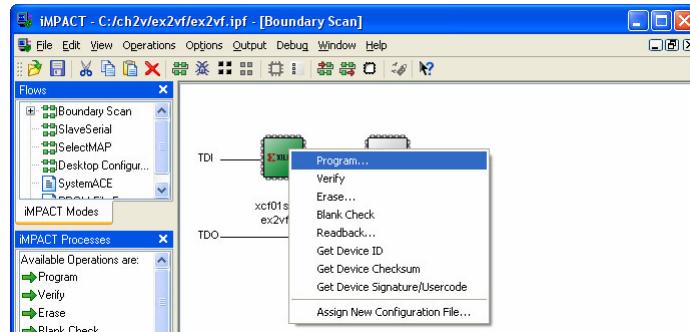


รูปที่ E2.64 หน้าต่าง iMPACT

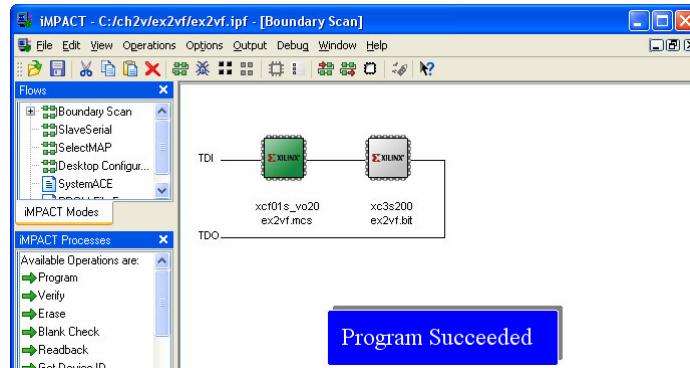


รูปที่ E2.65 หน้าต่าง iMPACT

e) การดาวน์โหลดข้อมูลลง Flash PROM ในกรณีที่ไม่มีการสร้างไฟล์ Flash PROM ในข้อ 2) หรือไม่ต้องการโปรแกรม Flash PROM ที่ไม่จำเป็นต้องทำขึ้น ทำการดาวน์โหลดนั้นให้คลิกขวาที่รูป Flash PROM และคลิก Program ดังรูปที่ E2.66 แล้วจะได้หน้าต่างดังรูปที่ E2.67 ถ้าดาวน์โหลดไม่ผ่านเพื่อความแน่ใจให้ดาวน์โหลดซ้ำ



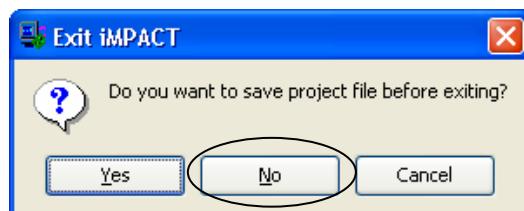
รูปที่ E2.66 หน้าต่าง iMPACT สำหรับดาวน์โหลด Flash PROM



รูปที่ E2.67 หน้าต่างแสดงเมื่อการดาวน์โหลด Flash PROM ผ่าน

ก) ปิดหน้าต่าง iMPACT โดยคลิก แล้วจะได้หน้าต่าง Exit iMPACT ขึ้นมาดังรูปที่ E2.35 แล้วคลิก No จากนั้นคลิก

(สีแดง) เพื่อปิดโปรแกรม ISE WebPACK 8.1i



รูปที่ E2.68 หน้าต่าง Exit iMPACT

2.17.6 การเปิดไฟล์ของ FPGA ที่เคยออกแบบไว้แล้วมาใช้งาน

การเปิดไฟล์ของ FPGA ที่เคยออกแบบไว้แล้วมาใช้งาน จะมีขั้นตอนชั้นเดียวกับกรณีของ CPLD ที่ได้อธิบายไปแล้วในข้อ 2.16.6 ทุกประการ

2.18 การใช้ซอฟต์แวร์ทุลออกแบบวงจรดิจิตอลที่มีคอมโพเนนต์รวมอยู่ด้วยโดยใช้ CPLD

ตัวอย่างที่ 2.37 ให้ออกแบบวงจรด้วย CPLD โดยสร้างวงจรนับในตัวอย่างที่ 2.34 ซึ่งเป็นวงจรนับ 4 หลักที่สามารถนับขึ้น-ลงและเคลื่อนไหวได้ และกำหนดให้ออตชิลด์ที่ใช้ = 32.768 kHz โดยวงจรนับนี้จะประกอบด้วยวงจรย่อขั้นที่คือ

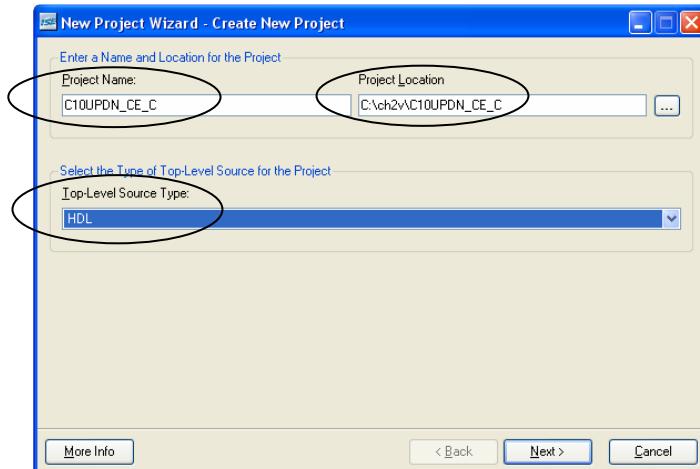
- วงจร 10 Bits frequency generator ใช้สร้างความถี่ 32 Hz และ 64 Hz จากอสซิเลเตอร์ความถี่ 32.768 kHz เพื่อจ่ายให้กับวงจรสแกน (Scan) ตัวแสดงผลเลขเวนเซกเมนต์ทั้ง 4 หลัก โดยความถี่สแกนแต่ละหลัก = 32 Hz > 30 ครั้งต่อวินาทีเพื่อไม่ให้ตัวแสดงผลการกระพริบ และใช้สร้างความถี่ 128 Hz (<150 Hz) สำหรับวงจร Debouncer
- วงจร Debouncer จะใช้สำหรับแก้ไขเบาช์ที่เกิดจากการกดปุ่มของวงจรนับ (ปุ่ม COUNT)
- วงจร Bidirectional counter : 0-9999 เป็นวงจรนับที่ประกอบด้วยวงจรนับ 10 จำนวน 4 ชุด โดยการนำเอาโกัดของวงจรนับ 10 แบบนับขึ้น-นับลง (ที่ใช้กันบ่อย) ในรูปที่ 2.58 มาสร้างเป็น Component
- วงจรสแกน ประกอบด้วย วงจร 4Bits MUX4TO1 เป็นมัลติเพล็กเซอร์ขนาด 4 บิตแบบเข้า 4 อินพุตออก 1 เอาต์พุตเพื่อนำค่าเอาต์พุตจากวงจรนับแต่ละหลักไปที่วงจรอุดรหัสตัวแสดงผลเลขเวนเซกเมนต์ และวงจร 2 to 4 Decoder ที่มีเอาต์พุตเป็นแบบ Active low (เอาต์พุตเป็นแบบ One cold) ซึ่งจะใช้ในการเลือกขา Common cathode ของแต่ละหลัก เพื่อให้ตัวแสดงผลหลักที่ต้องการนั้นติดสว่าง
- BCD to 7 Segment decoder จะเป็นวงจรอุดรหัสตัวแสดงผลเลขเวนเซกเมนต์ โดยการนำเอาโกัดของวงจรอุดรหัสตัวแสดงผลเลขเวนเซกเมนต์ (ที่ใช้กันบ่อย) ในรูปที่ 2.29b มาสร้างเป็น Component

2.18.1 ขั้นตอนการสร้างไฟล์คอมโพเนนต์

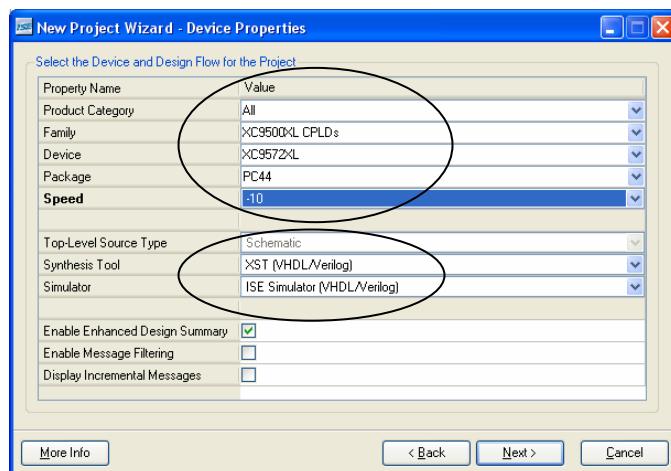
การสร้างไฟล์ Component ก็คือขั้นตอนการออกแบบวงจร (Design entry) ที่ได้อธิบายไปแล้วในข้อ 2.16.1 นั่นเอง โดยจะสร้างโกัดของวงจรนับ 10 แบบนับขึ้น-นับลงที่มีขา CE และ CEO ไว้ในไฟล์ชื่อ C10UPDN_CE_C และไฟล์โกัดของวงจรอุดรหัสตัวแสดงผลเลขเวนเซกเมนต์ไว้ในไฟล์ชื่อ DECODER_7SEGMENT_C

ขั้นตอนการสร้างไฟล์ Component ของโกัดของวงจรนับ 10 แบบนับขึ้น-นับลงเป็นดังนี้

- เริ่มที่หน้าจอคอมพิวเตอร์โดยคลิกปุ่ม Start -> Programs -> Xilinx ISE 8.1i -> Project Navigator หรือดับเบิลคลิกที่  แล้วจะได้หน้าต่าง Xilinx-ISE (ถ้ามีหน้าต่าง Tip of the Day ข้อนี้มาให้คลิก OK) คลิกที่ File -> New Project !แล้วจะได้หน้าต่าง (หรือ Dialog box) New Project Wizard>Create New Project
- ที่หน้าต่าง New Project Wizard>Create New Project สร้างโปรเจกต์ไฟล์ (Project File) ใหม่โดยพิมพ์ใน Project Location (หรือ Folder) ชื่อ ch2v และ Project Name ชื่อ C10UPDN_CE_C ตามลำดับ คลิกที่ Top-Level Source Type เป็น HDL และจะได้ตัวรูปที่ E3.1 คลิก Next แล้วจะได้หน้าต่าง New Project Wizard–Device Properties
- ใช้บอร์ด CPLD Explorer XC9572XL ให้คลิกตัวรูปที่ E3.2 CPLD ตระกูล (Family) XC9500XL, เบนซ์ (Device) XC9572XL, Package แบบ PLCC 44 ขา (Package:PC44) และ Speed Grade :-10 แล้วเลือกที่ Synthesis tool เป็น XST (VHDL/Verilog)

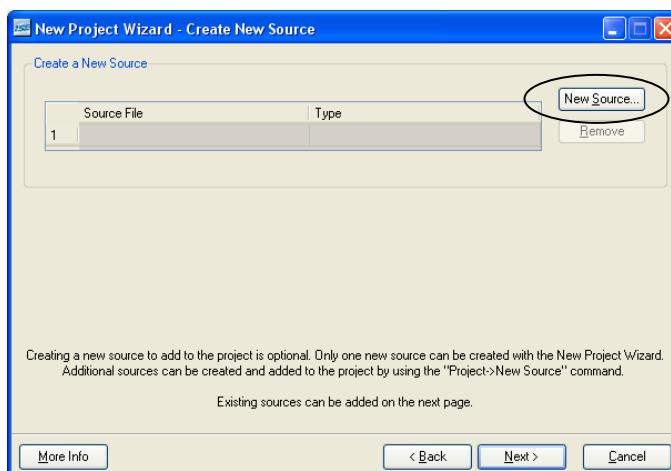


รูปที่ E3.1 หน้าต่าง New Project Wizard-Create New Project

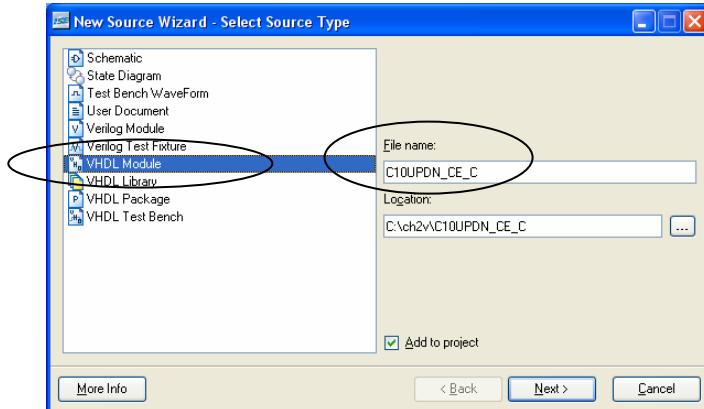


รูปที่ E3.2 หน้าต่าง New Project Wizard–Device Properties ในกรณีที่ใช้บอร์ด CPLD Explorer XC9572XL

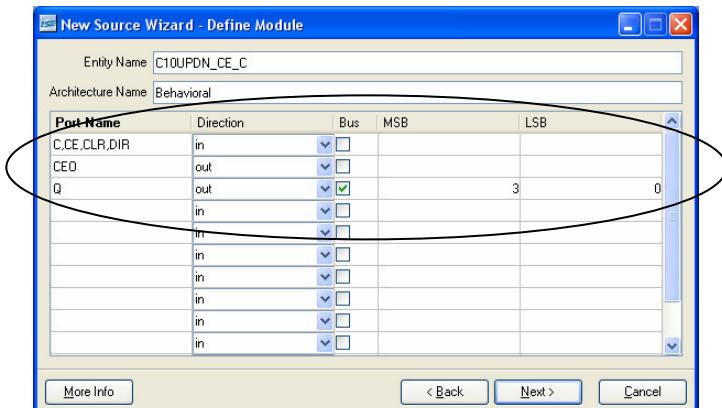
4) คลิก Next ในรูปที่ E3.2 เล้าจะได้หน้าต่างดังรูปที่ E3.3 คลิกปุ่ม New Source จะได้หน้าต่างลักษณะนี้ ให้พิมพ์ชื่อ Source File ชื่อ C10UPDN_CE_C ในช่อง File Name คลิกที่ VHDL Module ดังรูปที่ E3.4 คลิก Next จะได้หน้าต่างดังรูปที่ E3.5 เมื่อกำหนดอินพุต A,B และเอาต์พุต C เรียบร้อยแล้วคลิก Next 1 ครั้ง คลิก Finish 1 ครั้ง คลิก Next ไปอีก 2 ครั้ง คลิก Finish อีก 1 ครั้ง แล้วจะได้หน้าต่าง Xilinx–ISE จากนั้นทำการคัดลอกโค้ดในรูปที่ 2.58 มาเขียนใหม่ได้ดังรูปที่ E3.6 แล้วคลิก บันทึกไฟล์



รูปที่ E3.3 หน้าต่าง New Project Wizard–Create New Source



รูปที่ E3.4 หน้าต่าง New Source Wizard–Select Source Type



รูปที่ E3.5 New Source Wizard–Define Module

Xilinx - ISE - C:\ch2v\C10UPDN_CE_C\C10UPDN_CE_C.ise - [C10UPDN_CE_C.vhd]

```

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

entity C10UPDN_CE_C 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 C10UPDN_CE_C;

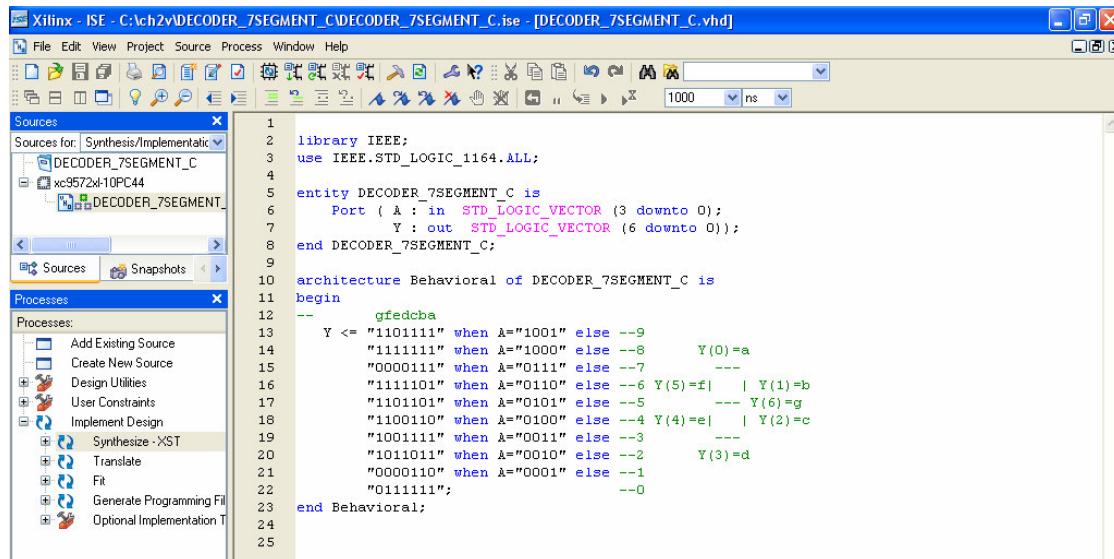
architecture Behavioral of C10UPDN_CE_C is
    signal QT : STD_LOGIC_VECTOR (3 downto 0);
begin
process(C,CLR)
begin
    if CLR='1' then QT <= "0000";
    elsif (C'event and C='1') then
        if CE='1' then
            if DIR='0' then -- Count up
                if QT>=9 then QT <= "0000";
                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 process;
Q <= QT; -- Counter output
CEO <= CE when (DIR ='0' and QT=9) else--CEO-count up
          CE when (DIR /='0' and QT=0) else--CEO-count down
          '0';
end Behavioral;

```

รูปที่ E3.6 ไฟล์ Component ของวงจรนับ 10 แบบนับขึ้น-นับลงและเคลียร์ค่าได้

5) การตรวจความถูกต้อง ถ้าไม่มีข้อผิดพลาดก็ถือว่าขั้นตอนการสร้างไฟล์ Component ของวงจรนับ 10 แล้วเสร็จ คลิก **X** (สีดำ) เพื่อปิดโปรแกรม Text Editor และกลับไปที่หน้าต่าง Xilinx-ISE จากนั้นคลิก **X** (สีแดง) เพื่อปิดโปรแกรม ISE WebPACK 8.1i

ในทำนองเดียวกันเราสามารถสร้างไฟล์ Component ของวงจรดอรหัสตัวแสดงผลเซกเมนต์ได้เช่นเดียวกัน โดยสร้างไว้ใน Project Location (หรือ Folder) ชื่อ ch2v และ Project Name ชื่อ DECODER_7SEGMENT_C จากนั้นทำการคัดลอกไฟล์ในรูปที่ 2.29b มาเขียนใหม่ คลิก  บันทึกไฟล์แล้วจะได้ดังรูปที่ E3.7

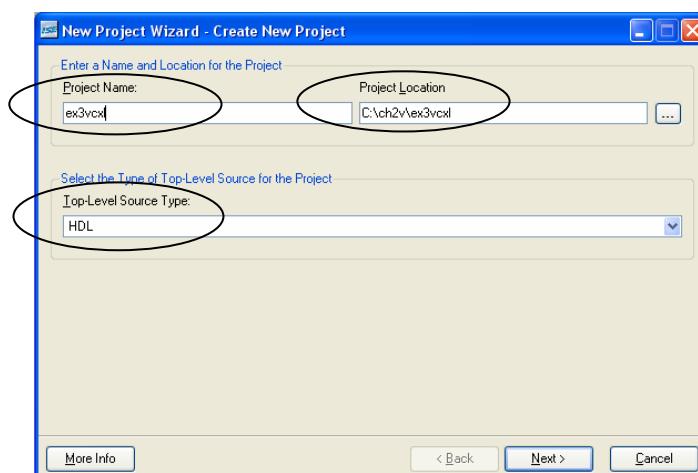


รูปที่ E3.7 ไฟล์ Component ของวงจรอุดรหัสตัวแสดงผลเซนเซอร์เซกเมนต์

2.18.2 ขั้นตอนการสร้างโค้ดหลัก (Main code) สำหรับการออกแบบวงจรดิจิตอลโดยใช้ CPLD

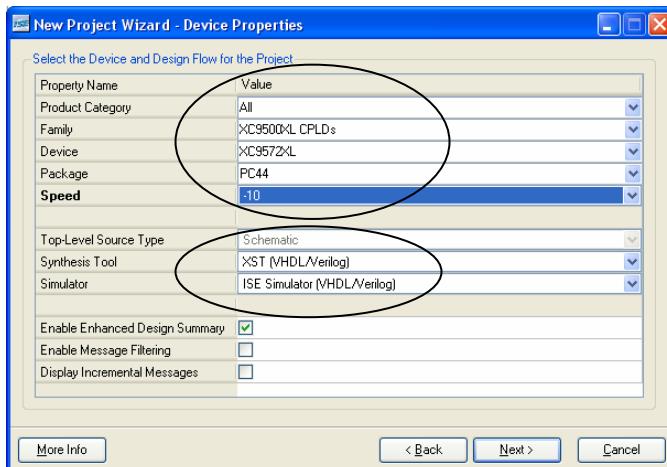
2.18.2.1 ขั้นตอนการออกแบบวงจร (Design entry)

- คลิกปุ่ม Start -> Programs -> Xilinx ISE 8.1i -> Project Navigator หรือดับเบิลคลิกที่  แล้วจะได้หน้าต่าง Xilinx-ISE (ถ้ามีหน้าต่าง Tip of the Day ข้อนี้บินมาให้คลิก OK) คลิกที่ File -> New Project แล้วจะได้หน้าต่างคลิกไป
 - พิมพ์ชื่อ ch2v ใน Project Location (หรือ Folder) และ Project Name ชื่อ ex3vcx1 ตามลำดับ คลิกที่ Top-Level Source Type เป็น HDL แล้วจะได้ตั้งรูปที่ E3.8 คลิก Next แล้วจะได้หน้าต่าง New Project Wizard–Device Properties

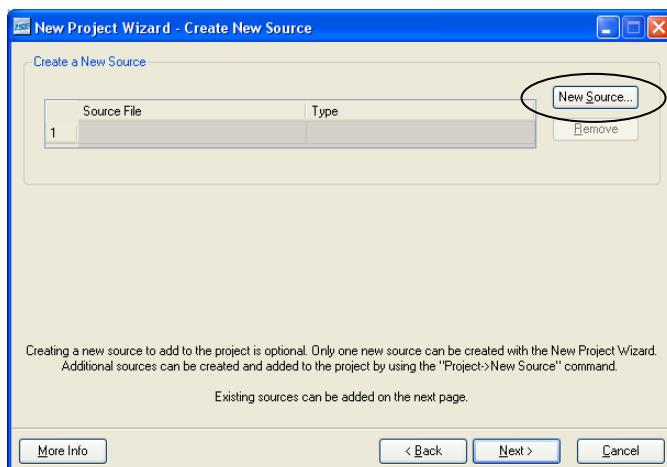


รูปที่ E3.8 หน้าต่าง New Project Wizard-Create New Project

3) กรณีที่ใช้บอร์ด CPLD Explorer XC9572XL ให้คลิกดังรูปที่ E3.9 คลิก Next แล้วจะได้หน้าต่างดังรูปที่ E3.10

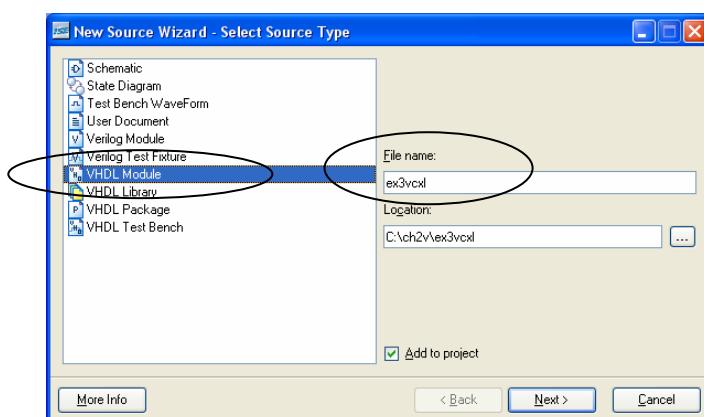


รูปที่ E3.9 หน้าต่าง New Project Wizard – Device Properties ในกรณีที่ใช้บอร์ด CPLD Explorer XC9572XL

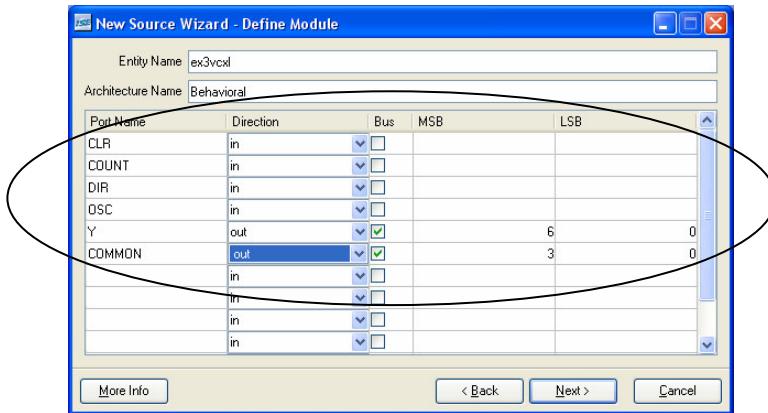


รูปที่ E3.10 หน้าต่าง New Project Wizard–Create New Source

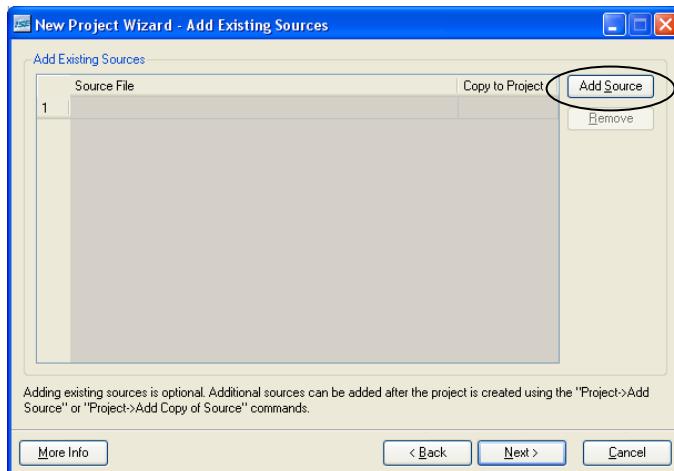
4) คลิกปุ่ม New Source จะได้หน้าต่างดังไป จากนั้นพิมพ์ชื่อ Source File ชื่อ ex3vcxl ลงในช่อง File Name แล้วคลิกที่ VHDL Module ดังรูปที่ E3.11 คลิก Next จะได้หน้าต่างดังรูปที่ E3.12 เมื่อกำหนดอินพุต A,B และเอาต์พุต C เรียบร้อยแล้วคลิก Next 1 ครั้ง คลิก Finish 1 ครั้ง แล้วคลิก Next อีก 1 ครั้งก็จะได้หน้าต่าง New Project Wizard-Add Existing Source ดังรูปที่ E3.13



รูปที่ E3.11 หน้าต่าง New Source Wizard–Select Source Type

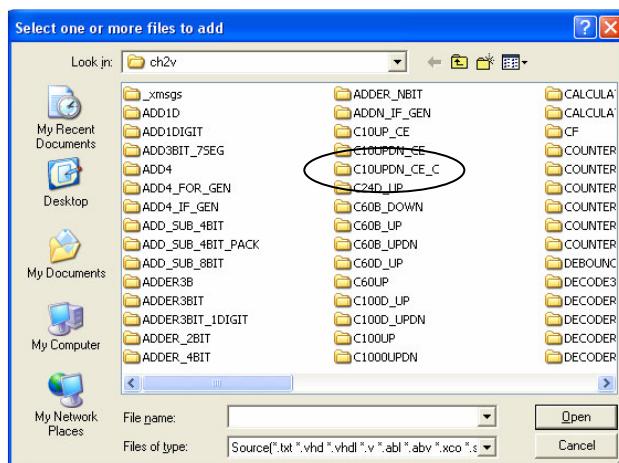


รูปที่ E3.12 หน้าต่าง New Source Wizard-Define Module

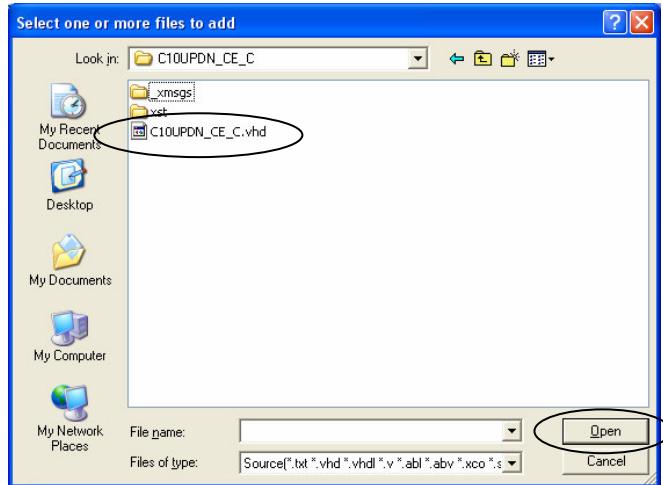


รูปที่ E3.13 หน้าต่าง New Project Wizard-Add Existing Source

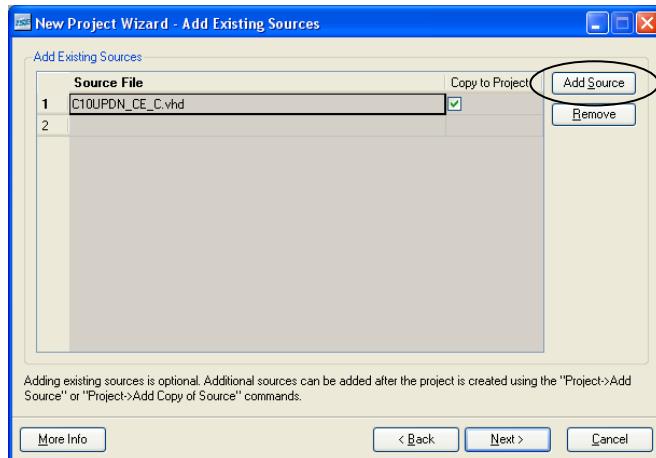
5) จากรูปที่ E3.13 คลิกปุ่ม Add Source แล้วจะได้หน้าต่าง Select one or more files to add ดังรูปที่ E3.14 จากนั้นคลิกหาไฟล์ Component (Project Name) ชื่อ C10UPDN_CE_C ที่อยู่ใน Folder ชื่อ ch2v เมื่อค้นพบคลิกไฟล์ชื่อ C10UPDN_CE_C แล้วจะได้ดังรูปที่ E3.15 คลิกที่ไฟล์ชื่อ C10UPDN_CE_C.vhd และคลิก Open เพื่อเพิ่มไฟล์จะได้ดังรูปที่ E3.16



รูปที่ E3.14 หน้าต่าง Select one or more files to add (ตำแหน่งไฟล์จะไม่แน่นอน ขึ้นกับจำนวนไฟล์ใน Folder นั้น)

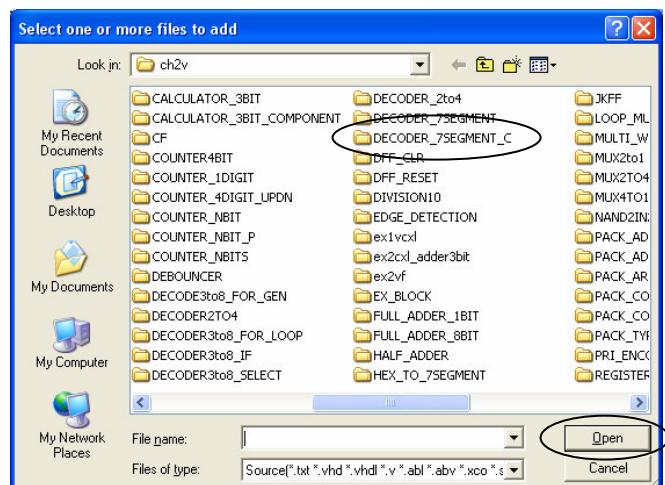


รูปที่ E3.15 หน้าต่าง Select one or more files to add

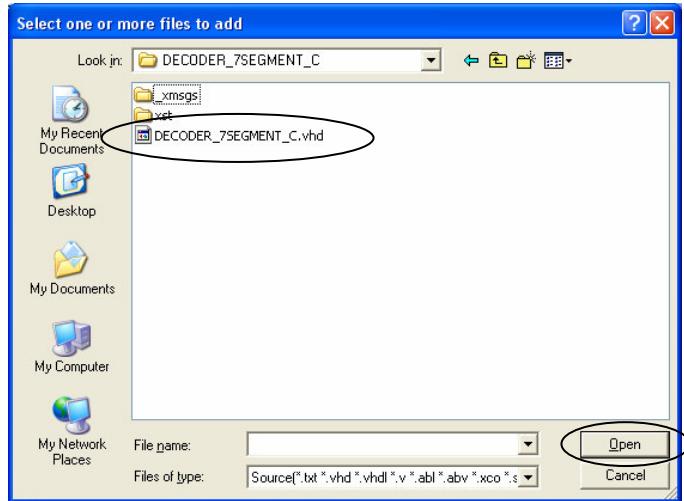


รูปที่ E3.16 หน้าต่าง New Project Wizard-Add Existing Source

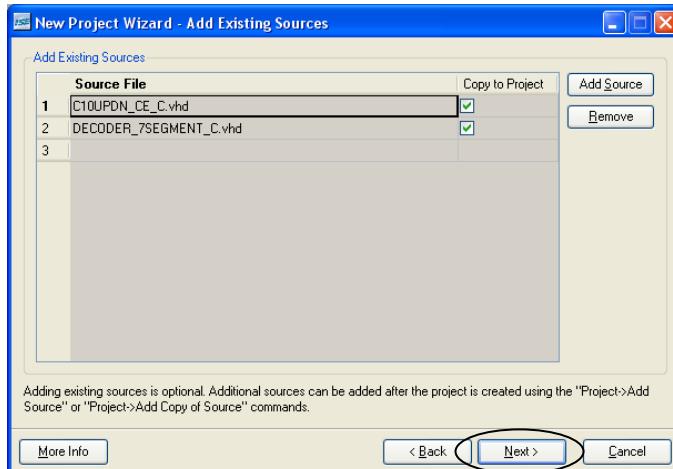
6) จากรูปที่ E3.16 คลิกปุ่ม Add Source อีกครั้งแล้วจะได้หน้าต่าง Select one or more files to add ดังรูปที่ E3.17 จากนั้นคลิกไฟล์ Component ชื่อ DECODER_7SEGMENT_C ใน Folder ชื่อ ch2v เมื่อคัมเบิลคลิกไฟล์ชื่อ DECODER_7SEGMENT_C แล้วจะได้ดังรูปที่ E3.18 คลิกที่ไฟล์ชื่อ DECODER_7SEGMENT_C.vhd และคลิก Open เพื่อเพิ่มไฟล์จะได้ดังรูปที่ E3.19



รูปที่ E3.17 หน้าต่าง Select one or more files to add

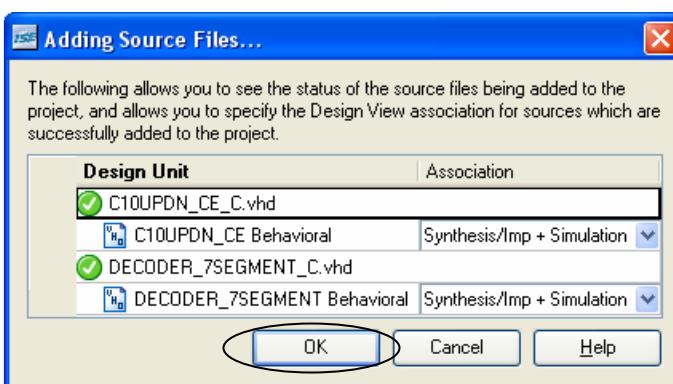


รูปที่ E3.18 หน้าต่าง Select one or more files to add

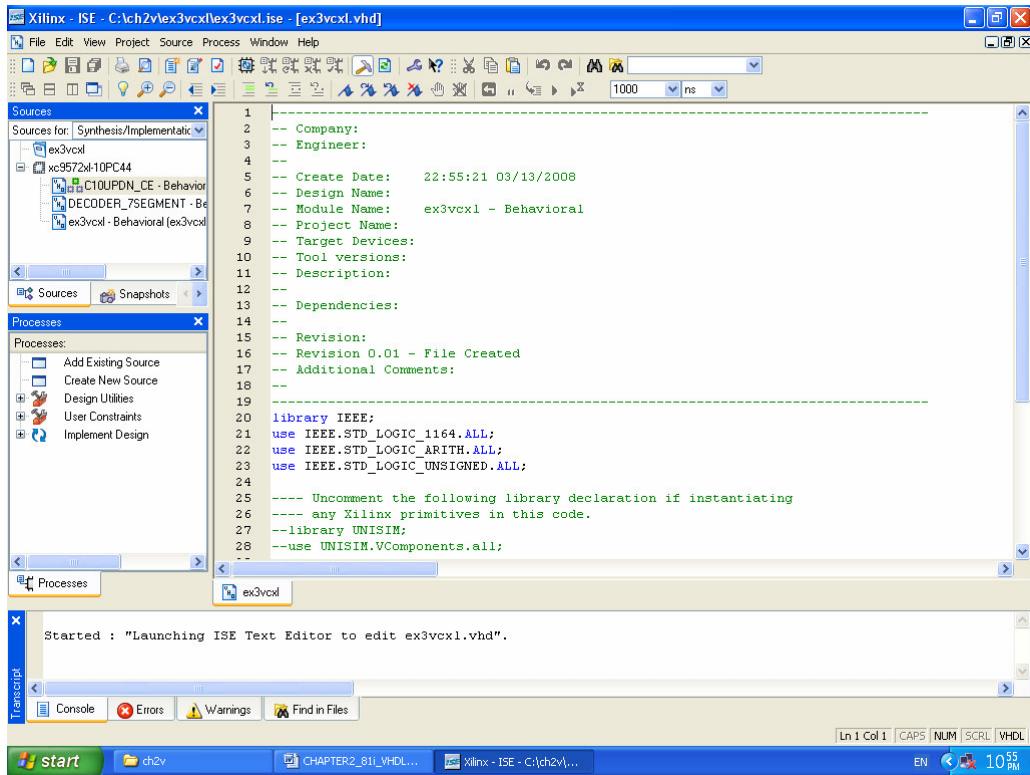


รูปที่ E3.19 หน้าต่าง New Project Wizard-Add Existing Source

7) จากรูปที่ E3.19 จากนั้นคลิก Next 1 ครั้ง คลิก Finish อีก 1 ครั้ง คลิก OK ในรูปที่ E3.20 แล้วจะได้หน้าต่าง Xilinx-ISE สำหรับเขียนโค้ด (โปรแกรม Text Editor) ดังรูปที่ E3.21

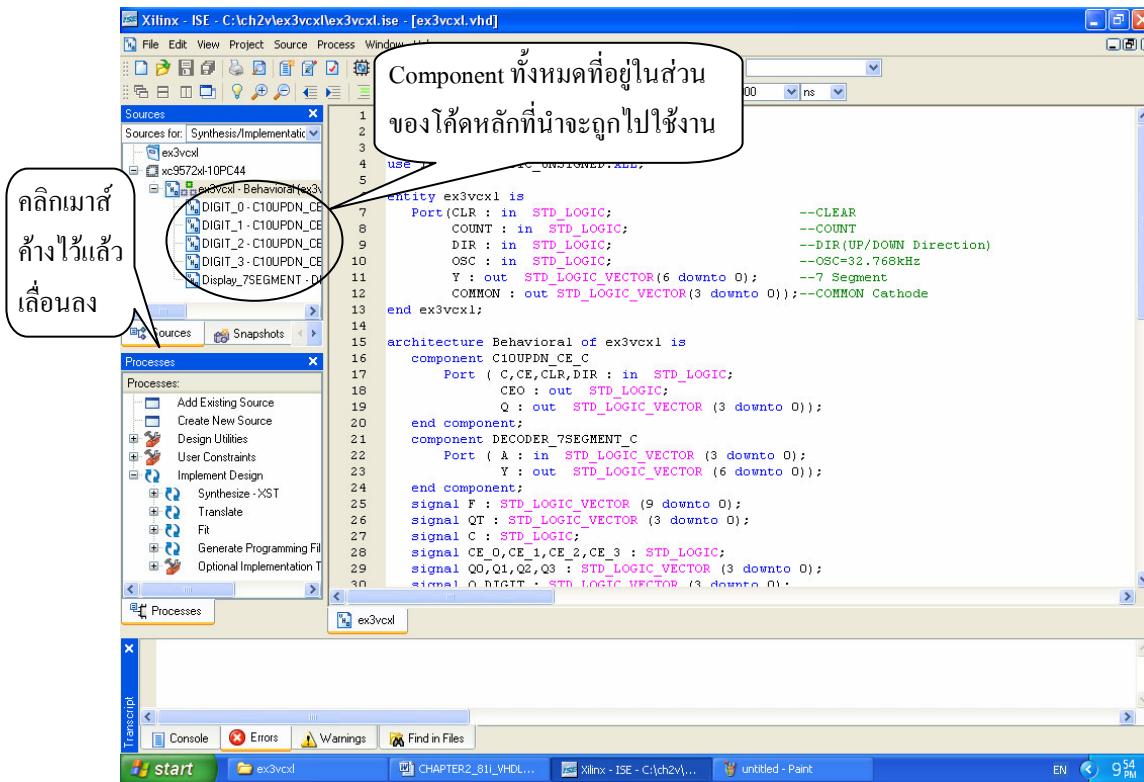


รูปที่ E3.20 หน้าต่าง Add Source file



รูปที่ E3.21 หน้าต่าง Xilinx-ISE สำหรับเขียนโค้ด VHDL (โปรแกรม Text Editor)

9) นำโค้ดของวงจรนับ 4 หลักในตัวอย่างที่ 2.34 มาปรับปรุงแก้ไข เสร็จแล้วจะได้ดังรูปที่ E3.22 แล้วคlik กับปุ่มที่ไฟล์ ซึ่งรายละเอียดของโค้ดหลักทั้งหมดในรูปที่ E3.22 แสดงดังรูปที่ E3.23 จากนั้นทำการตรวจสอบความถูกต้อง (Syntax) ถ้าไม่มีข้อผิดพลาดก็ถือว่าขั้นตอน Design entry เสร็จสมบูรณ์ คลิก X (สีดำ) ปิดโปรแกรม Text Editor เพื่อทำขั้นตอนต่อไป



รูปที่ E3.22 โค้ดหลักทั้งหมดของวงจรนับ 4 หลักในตัวอย่างที่ 2.34 ที่ปรับปรุงแก้ไขเสร็จแล้ว

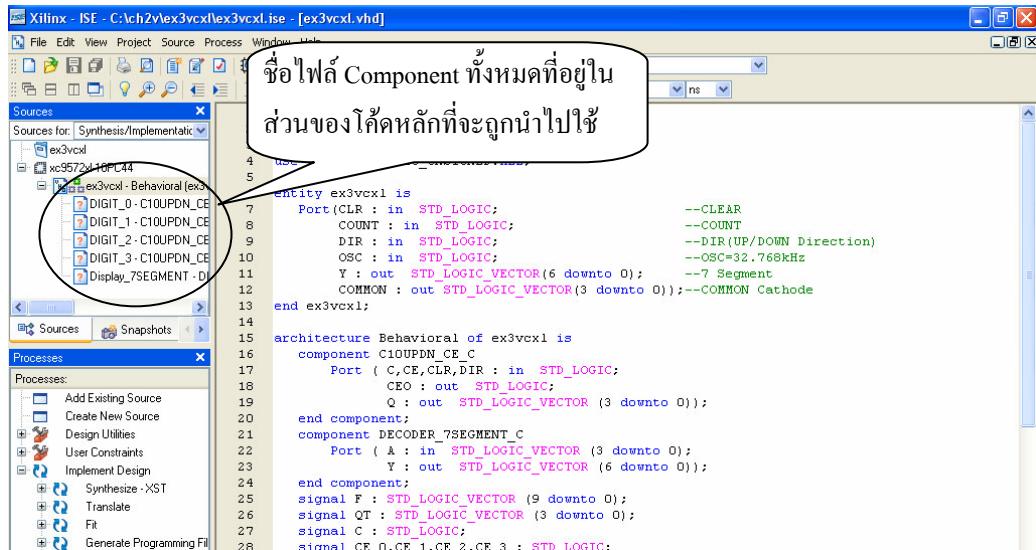
```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex3vcx1 is
7     Port(CLK : in STD_LOGIC;           --CLEAR
8             COUNT : in STD_LOGIC;        --COUNT
9             DIR : in STD_LOGIC;         --DIR(UP/DOWN Direction)
10            OSC : in STD_LOGIC;        --OSC=32.768kHz
11            Y : out STD_LOGIC_VECTOR(6 downto 0);   --7 Segment
12            COMMON : out STD_LOGIC_VECTOR(3 downto 0));--COMMON Cathode
13 end ex3vcx1;
14
15 architecture Behavioral of ex3vcx1 is
16     component C10UPDN_CE_C
17         Port ( C,CE,CLR,DIR : in STD_LOGIC;
18                 CEO : out STD_LOGIC;
19                 Q : out STD_LOGIC_VECTOR (3 downto 0));
20     end component;
21     component DECODER_7SEGMENT_C
22         Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
23                 Y : out STD_LOGIC_VECTOR (6 downto 0));
24     end component;
25     signal F : STD_LOGIC_VECTOR (9 downto 0);
26     signal QT : STD_LOGIC_VECTOR (3 downto 0);
27     signal C : STD_LOGIC;
28     signal CE_0,CE_1,CE_2,CE_3 : STD_LOGIC;
29     signal Q0,Q1,Q2,Q3 : STD_LOGIC_VECTOR (3 downto 0);
30     signal Q_DIGIT : STD_LOGIC_VECTOR (3 downto 0);
31 begin
32     -----10 Bits frequency generator-----
33     process(OSC)
34     begin --OSC=32.768kHz=(2**15)Hz,F(0)=OSC/(2**9+1)=32Hz,F(1)=64Hz,F(2)=128Hz
35         if (OSC'event and OSC='1') then F <= F + 1;
36     end if;
37 end process;
38     -----Debouncer-----
39     process( F(2) )
40     begin--Debouncer frequency : F(2)=128Hz<150Hz
41         if (F(2)'event and F(2)='1') then
42             QT <= QT(2 downto 0) &(not COUNT);--COUNT active low
43         end if;
44     end process;
45     C <= QT(0) and QT(1) and QT(2) and not QT(3);
46     -----Bidirectional counter : 0-9999-----
47     CE_0 <= '1';                         --CE='1'
48     DIGIT_0 : C10UPDN_CE_C port map( C=>C,CE=>CE_0,CLR=>DIR,DIR=>DIR,CEO=>CE_1,Q=>Q0);
49     DIGIT_1 : C10UPDN_CE_C port map( C=>C,CE=>CE_1,CLR=>CLR,DIR=>DIR,CEO=>CE_2,Q=>Q1);
50     DIGIT_2 : C10UPDN_CE_C port map( C=>C,CE=>CE_2,CLR=>CLR,DIR=>DIR,CEO=>CE_3,Q=>Q2);
51     DIGIT_3 : C10UPDN_CE_C port map( C=>C,CE=>CE_3,CLR=>CLR,DIR=>DIR,CEO=>open,Q=>Q3);
52     -----4Bits MUX4TO1-----
53     Q_DIGIT <= Q0 when F(1 downto 0)="00" else          --Select Data Digit No.0
54             Q1 when F(1 downto 0)="01" else          --Select Data Digit No.1
55             Q2 when F(1 downto 0)="10" else          --Select Data Digit No.2
56             Q3;  -- F(1 downto 0)="11"               --Select Data Digit No.3
57     -----2 to 4 Decoder (One cold)-----
58     COMMON <= "1110" when F(1 downto 0)="00" else      --Select Digit No.0
59             "1101" when F(1 downto 0)="01" else      --Select Digit No.1
60             "1011" when F(1 downto 0)="10" else      --Select Digit No.2
61             "0111"; -- F(1 downto 0)="11"           --Select Digit No.3
62     -----BCD to 7 Segment decoder-----
63     Display_7SEGMENT : DECODER_7SEGMENT_C port map (A=>Q_DIGIT,Y=>Y);
64 end Behavioral;

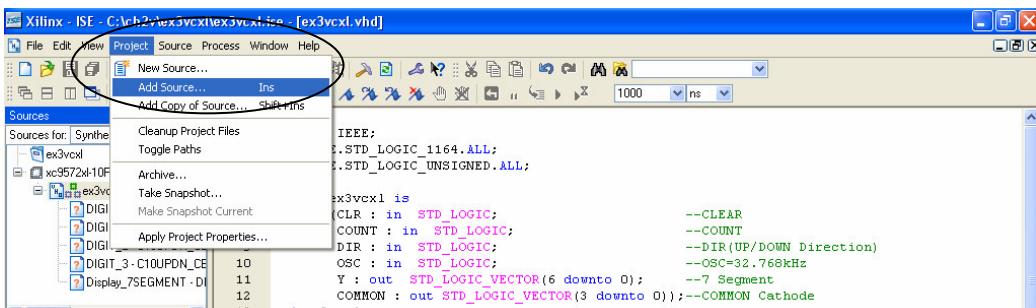
```

รูปที่ E3.23 รายละเอียดของโลคัลหลักทั้งหมดของวงจรนับ 10 ในตัวอย่างที่ 2.34 ที่ปรับปรุงแก้ไขเสร็จแล้ว

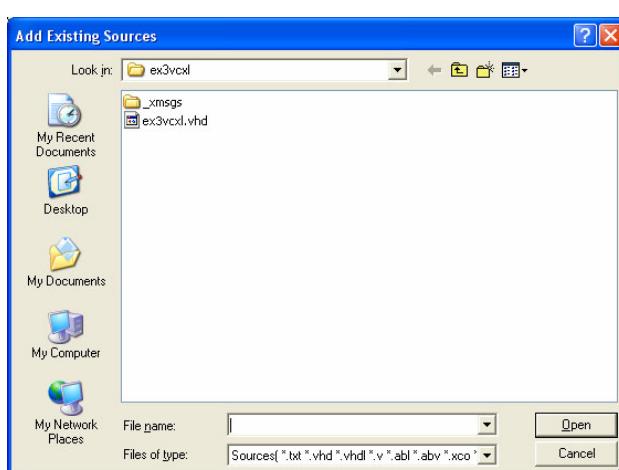
จากรูปที่ E3.13 ถ้าเราไม่คลิกปุ่ม Add Source แต่คลิก next เข้าเดียวกับตัวอย่างที่ 2.35 นั้น เมื่อเขียนโค้ดเสร็จแล้ว คลิก บันทึกไฟล์ จะทำให้ในหน้าต่าง Source ได้ดังรูปที่ E3.24 ซึ่งจะปรากฏเฉพาะชื่อไฟล์ Component แต่ไม่มีโค้ดของ Component ซึ่งการ Add Source ของ Component ในภายหลังสามารถทำได้โดยคลิก Add Source ดังรูปที่ E3.25 และจะได้หน้าต่าง Add Existing Source ดังรูปที่ E3.26



รูปที่ E3.24 ໂຄດຫລັກທີ່ໜົມດຂອງຈະຈົນນັບ 10 ໃນຕາວຍ່າງທີ່ 2.51 ທີ່ປ່ຽນປ່ອງແກ້ໄຂເສົ່າງແລ້ວຍື່ນຳໃໝ່ Add Source

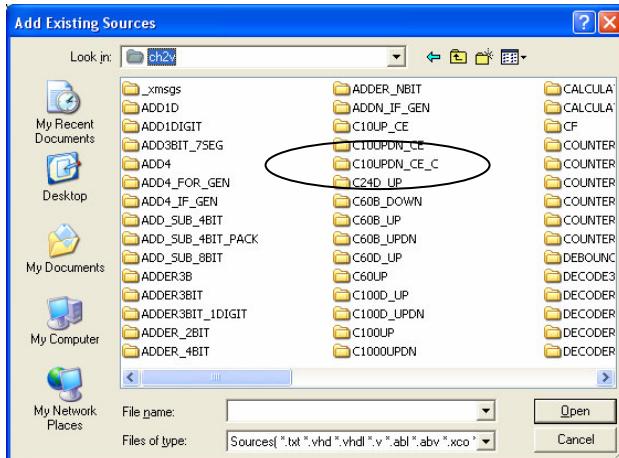


รูปที่ E3.25 เริ่มขั้นตอนการ Add Source

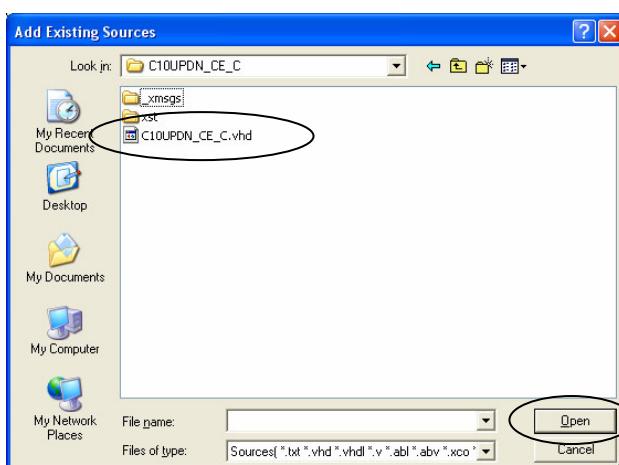


รูปที่ E3.26 หน้าต่าง Add Existing Source

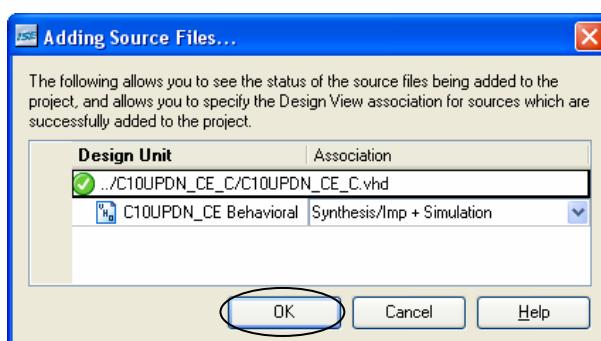
จากรูปที่ E3.26 ให้หาไฟล์ Component (Project Name) ชื่อ C10UPDN_CE_C ใน Folder ชื่อ ch2v จะได้ดังรูปที่ E3.27 ดับเบิลคลิกไฟล์ชื่อ C10UPDN_CE_C แล้วจะได้ดังรูปที่ E3.28 คลิกที่ไฟล์ชื่อ C10UPDN_CE_C.vhd แล้วคลิก Open เพื่อเพิ่มไฟล์จะได้ดังรูปที่ E3.29 แล้วคลิกOK จากนั้นทำขั้นตอนนี้ซ้ำแต่จะเปลี่ยนเป็นไฟล์ชื่อ DECODER_7SEGMENT_C ใน Folder ชื่อ ch2v



รูปที่ E3.27 หน้าต่าง Add Existing Source



รูปที่ E3.28 หน้าต่าง Add Existing Source



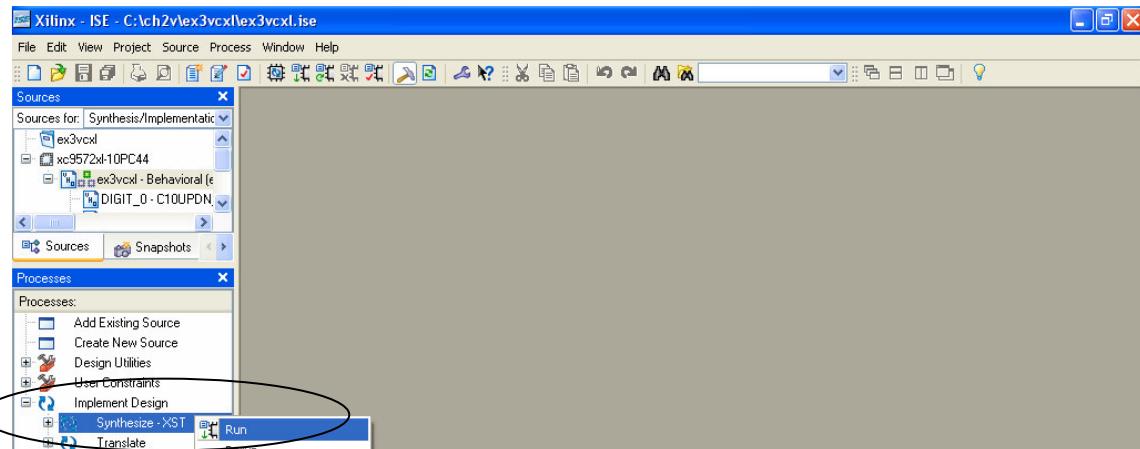
รูปที่ E3.29 หน้าต่าง Adding Source files

2.18.2.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification)

การออกแบบวงจรที่ไม่มีความซับซ้อนอาจไม่มีความจำเป็นต้องทำขั้นตอนนี้ก็ได้ ให้ข้ามไปทำข้อ 2.18.2.3 ได้เลย

2.18.2.3 การสังเคราะห์วงจร (Design Synthesis)

1) ขั้นตอนสังเคราะห์วงจร ที่หน้าต่าง Processes คลิก “+” หน้า Implement Design จะเป็น “-” คลิกขวาแล้วคลิก Run ที่ Synthesize-XST ดังรูปที่ E3.30 (หรือดับเบิลคลิก) เสร็จแล้วถ้าได้ หรือ ที่หน้า Synthesize-XST จะถือว่าสังเคราะห์วงจรผ่าน ซึ่งถ้าไม่ต้องการคุறำลดเวลาอีกด้วยการสังเคราะห์วงจรก็ให้ข้ามไปทำหัวข้อ 2.22.2.4 ขั้นตอน Design Implementation ได้เลย

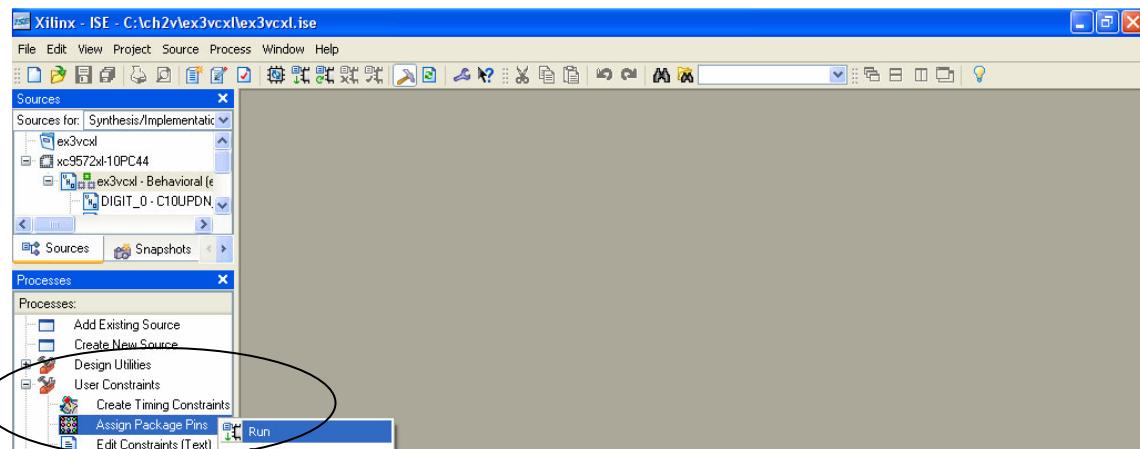


รูปที่ E3.30 ขั้นตอนสังเคราะห์วงจร

2.18.2.4 Design Implementation

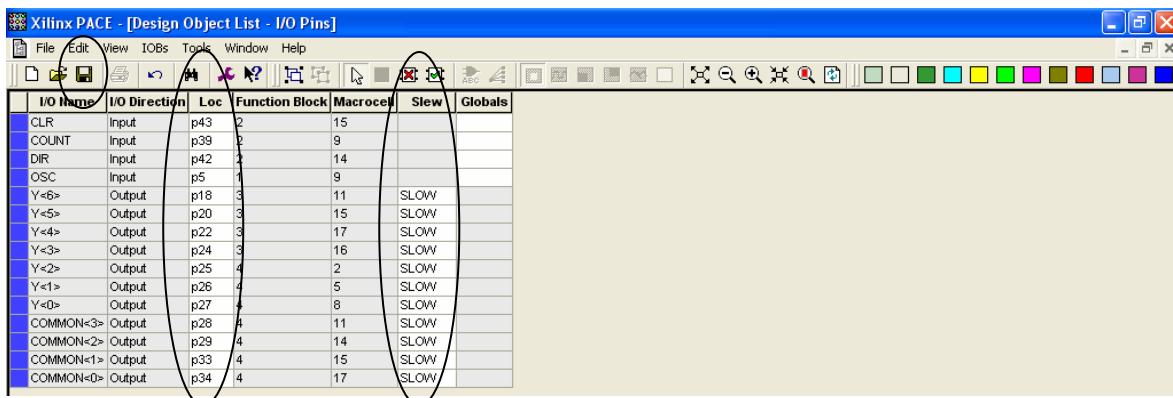
1) ขั้นตอน Implementation constraint file (UCF)

a) คลิก “+” หน้า User Constraints ในหน้าต่าง Processes จะเป็น “-” คลิกขวาที่ Assign Package Pins แล้วคลิก Run ดังรูปที่ E3.31 คลิก Yes เพื่อยืนยันที่จะสร้าง Implementation constraint file (UCF) โดยอัตโนมัติ แล้วจะได้หน้าต่าง Xilinx-PACE

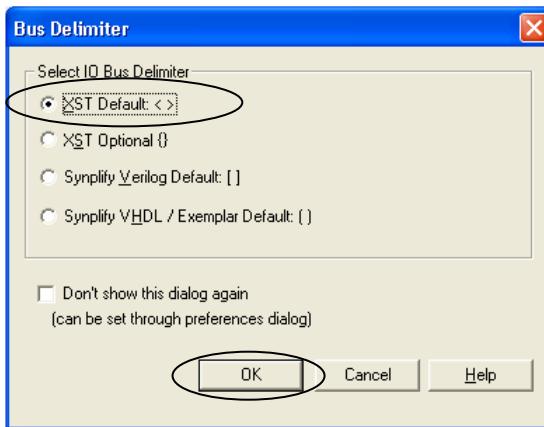


รูปที่ E3.31 ขั้นตอน Assign Package Pins

b) การกำหนดขาสัญญาณต่างๆ กรณีเราใช้บอร์ด CPLD Explorer XC9572XL เราจะกำหนดอินพุตและเอาต์พุตลงในคอลัมน์ Loc คลิกเอาต์พุททั้งหมดที่อยู่ในคอลัมน์ Slew เป็น Slow เพื่อลดการออฟชิลเดตและการสะท้อนในสายสัญญาณ เสร็จแล้วจะได้ดังรูปที่ E3.32 คลิก บันทึกไฟล์แล้วจะมีหน้าต่าง Bus Delimiter ขึ้นมา คลิกเลือกที่ XST Default ดังรูปที่ E3.33 แล้วคลิก OK จากนั้นคลิก (สีแดง) เพื่อปิดหน้าต่าง Xilinx-PACE และกลับไปที่หน้าต่าง Xilinx-ISE

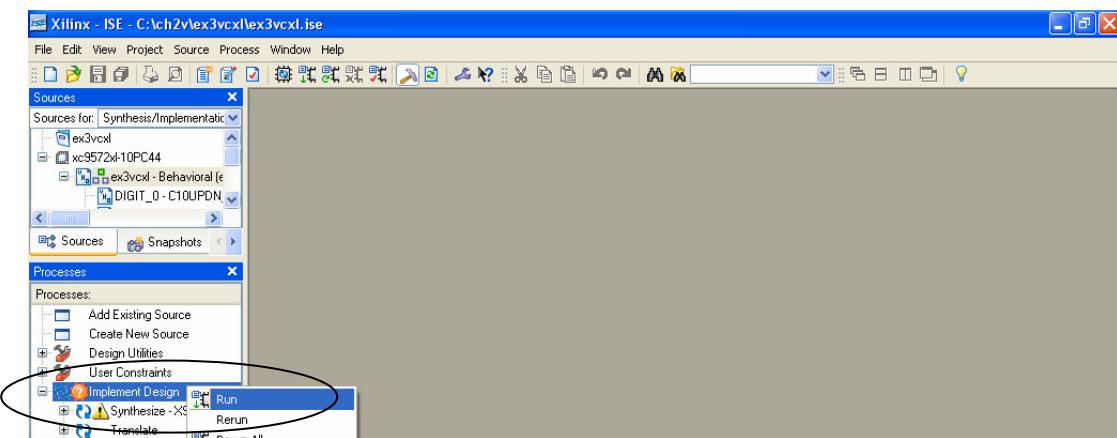


รูปที่ E3.32 การกำหนดค่าอินพุต/เอาต์พุตที่หน้าต่างย่อย Design Object List – I/O Pins ของหน้าต่าง Xilinx-PACE



รูปที่ E3.33 หน้าต่าง Bus Delimiter

2) ขั้นตอน Implement Design คลิก ex3vcxl-Behavioral ที่หน้าต่าง Source จากนั้นคลิกขวาที่ Implement Design แล้วคลิก Run ดังรูปที่ E3.34 แล้วรอซักครู่ เมื่อแล้วเสร็จถ้าได้ หรือ จะถือว่าขั้นตอน Implement Design ผ่าน



รูปที่ E3.34 หน้าต่างสำหรับขั้นตอน Implement Design

2.18.2.5 การโปรแกรมข้อมูลวงจรลงชิป CPLD

การโปรแกรมข้อมูลวงจรลงชิป CPLD จะมีขั้นตอนเหมือนข้อ 2.16.5 ทุกประการ

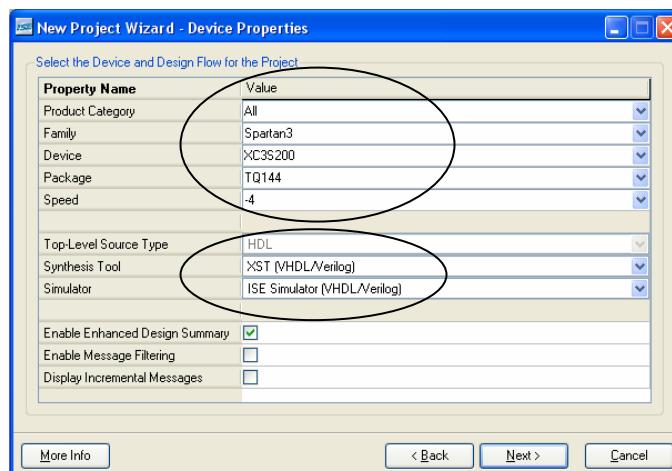
2.19 การใช้ซอฟต์แวร์ทูลอกแบบวงจรดิจิตอลที่มีคอมโพเนนต์รวมอยู่ด้วยโดยใช้ FPGA

การใช้ซอฟต์แวร์ทูลอกแบบวงจรดิจิตอลที่มีคอมโพเนนต์รวมอยู่ด้วยโดยใช้ FPGA จะมีขั้นตอนชั้นเดียวกับกรณีของ CPLD ที่ได้อธิบายไปแล้วในข้อ 2.18 ทุกประการ

ตัวอย่างที่ 2.38 ให้ออกแบบวงจรด้วย FPGA โดยสร้างวงจรนับในตัวอย่างที่ 2.34 ซึ่งเป็นวงจรนับ 4 หลักที่สามารถนับขึ้น-ลงและเคลื่อนค่าได้ และกำหนดให้ออตซิลิเตอร์ที่ใช้ = 25MHz โดยวงจรนับนี้จะประกอบด้วยวงจรย่ออย่างนี้ คือ

- วงจร 19 Bits frequency generator ใช้ในการสร้างความถี่ 47.68 Hz ($25M/2^{19}$ Hz) และ 95.36 Hz ($25M/2^{18}$ Hz) จาก ออสซิลเลเตอร์ความถี่ 25MHz เพื่อจ่ายให้กับวงจรสแกน (Scan) ตัวแสดงผลเซกเมนต์ทั้ง 4 หลัก โดยความถี่ สแกนแต่ละหลัก = $47.68 \text{ Hz} > 30 \text{ ครั้งต่อวินาที}$ เพื่อไม่ให้เห็นการกระพริบที่ตัวแสดงผล และใช้สร้างความถี่ 95.36 Hz (<150 Hz) สำหรับวงจร Debouncer
- วงจร Debouncer จะใช้สำหรับแก้ไขเบ้าซึ่งเกิดจากการกดปุ่มให้วางจนับ (ปุ่ม COUNT)
- วงจร Bidirectional counter : 0-9999 เป็นวงจรนับที่ประกอบด้วยวงจรนับ 10 จำนวน 4 ชุด โดยการนำเอาโคล์ดของ วงจนับ 10 แบบนับขึ้น-นับลง (ที่ใช้กันบ่อย) ในรูปที่ 2.58 มาสร้างเป็น Component
- วงจรสแกน ประกอบด้วย วงจร 4Bits MUX4TO1 เป็นมัลติเพล็กเซอร์ขนาด 4 บิตแบบเข้า 4 อินพุตออก 1 เอาต์พุตเพื่อ นำค่าเอาต์พุตจากวงจรนับแต่ละหลักไปที่วงจรอдрหัสตัวแสดงผลเซกเมนต์ และวงจร 2 to 4 Decoder ที่มี เอาต์พุตเป็นแบบ Active low (เอาต์พุตเป็นแบบ One cold) ซึ่งจะใช้ในการเลือกขา Common cathode ของแต่ละหลัก เพื่อให้ตัวแสดงผลหลักที่ต้องการนั้นติดสว่าง
- BCD to 7 Segment decoder จะเป็นวงจรอдрหัสตัวแสดงผลเซกเมนต์ โดยการนำเอาโคล์ดของวงจรอдрหัส ตัวแสดงผลเซกเมนต์ (ที่ใช้กันบ่อย) ในรูปที่ 2.29b มาสร้างเป็น Component

ขั้นตอนการสร้างไฟล์ Component ของ FPGA จะมีขั้นตอนต่างๆ เช่นเดียวกับกรณีของ CPLD ที่ได้อธิบายไปแล้วใน ข้อ 2.18.1 ทุกประการ ในกรณีที่ใช้บอร์ด FPGA Discovery-III XC3S200F นั้นให้คลิกดังรูปที่ E4.1 แต่ถ้าใช้บอร์ด FPGA Discovery-III XC3S200F4 ก็ให้คลิกแก้เฉพาะช่องที่เป็นบอร์ด (Device) โดยเปลี่ยนจากบอร์ด XC3S200 เป็น XC3S400



รูปที่ E4.1 หน้าต่าง New Project Wizard–Device Properties

เราจะสร้างไฟล์ Component ของวงจรนับ 10 แบบนับขึ้น-ลงที่มีขา CE และ CEO ไว้ใน Project Location (หรือ Folder) ชื่อ ch2v และ Project Name ชื่อ C10UPDN_CE_F และคัดลอกโค้ดในรูปที่ 2.58 มาเขียนใหม่ดังรูปที่ E4.2

```

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

entity C10UPDN_CE_F 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 C10UPDN_CE_F;

architecture Behavioral of C10UPDN_CE_F is
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
                        if QT>=9 then QT <= "0000";
                        else QT <= QT + 1;
                    end if;
                else
                    if (QT=0 or QT>9) then QT <= "1001";
                    else QT <= QT - 1;
                end if;
            end if;
        end if;
    end process;
    Q <= QT; -- Counter output
    CEO <= CE when (DIR ='0' and QT=9) else--CEO-count up
    CEO when (DIR /='0' and QT=0) else--CEO-count down
    '0';
end Behavioral;

```

รูปที่ E4.2 ไฟล์ Component ของวงจรนับ 10 แบบนับขึ้น-ลงและเคลียร์ค่าได้

ในการทำงานเดียวกันเราสามารถสร้างไฟล์ Component ของวงจรอุดรหัสตัวแสดงผลเซเว่นเซกเมนต์ได้เช่นเดียวกันโดยสร้างไว้ใน Project Location (หรือ Folder) ชื่อ ch2v และ Project Name ชื่อ DECODER_7SEGMENT_F จากนั้นทำการคัดลอกโค้ดในรูปที่ 2.29b มาเขียนใหม่ ได้ดังรูปที่ E4.3

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DECODER_7SEGMENT_F is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (6 downto 0));
end DECODER_7SEGMENT_F;

architecture Behavioral of DECODER_7SEGMENT_F 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
end Behavioral;

```

รูปที่ E4.3 ไฟล์ Component ของวงจรอุดรหัสตัวแสดงผลเซเว่นเซกเมนต์

ขั้นตอนการสร้างโค้ดหลัก (Main code) สำหรับการออกแบบวงจรดิจิตอลโดยใช้ FPGA นั้นจะมีขั้นตอนเช่นเดียวกับกรณีของ CPLD ที่ได้อธิบายไปแล้วในข้อ 2.18.2 ทุกประการ โดยเราจะสร้างไฟล์ไว้ใน ch2v ใน Project Location (หรือ Folder) และ Project Name ชื่อ ex4vf รายละเอียดโค้ดหลักทั้งหมดของวงจรนับ 4 หลักในตัวอย่างที่ 2.34 ที่ปรับปรุงแก้ไขเสร็จแล้วจะเป็นดังรูปที่ E4.4 ส่วนการกำหนดขาสัญญาณต่างๆ ให้กับ FPGA จะเป็นดังรูปที่ E4.5

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4vf is
7     Port(CLR : in STD_LOGIC;                      --CLEAR
8         COUNT : in STD_LOGIC;                     --COUNT
9         DIR : in STD_LOGIC;                      --DIR(UP/DOWN Direction)
10        OSC : in STD_LOGIC;                      --OSC=25MHz
11        Y : out STD_LOGIC_VECTOR(6 downto 0);    --7 Segment
12        COMMON : out STD_LOGIC_VECTOR(3 downto 0));--COMMON Cathode
13 end ex4vf;
14
15 architecture Behavioral of ex4vf is
16     component C10UPDN_CE_F
17         Port ( C,CE,CLR,DIR : in STD_LOGIC;
18                 CEO : out STD_LOGIC;
19                 Q : out STD_LOGIC_VECTOR (3 downto 0));
20     end component;
21     component DECODER_7SEGMENT_F
22         Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
23                 Y : out STD_LOGIC_VECTOR (6 downto 0));
24     end component;
25     signal F : STD_LOGIC_VECTOR (18 downto 0);
26     signal QT : STD_LOGIC_VECTOR (3 downto 0);
27     signal C : STD_LOGIC;
28     signal CE_0,CE_1,CE_2,CE_3 : STD_LOGIC;
29     signal Q0,Q1,Q2,Q3 : STD_LOGIC_VECTOR (3 downto 0);
30     signal Q_DIGIT : STD_LOGIC_VECTOR (3 downto 0);
31 begin
32     -----19 Bits frequency generator-----
33     process(OSC)
34     begin --OSC=25MHz ,F(18)=OSC/(2**((18+1))=47.68Hz,F(17)=95.36Hz
35         if (OSC'event and OSC='1') then F <= F + 1;
36     end if;
37 end process;
38     -----Debouncer-----
39     process( F(17) )
40     begin--Debouncer frequency : F(17)=95.36Hz<150Hz
41         if (F(17)'event and F(17)='1') then
42             QT <= QT(2 downto 0) & (not COUNT);--COUNT active low
43         end if;
44     end process;
45     C <= QT(0) and QT(1) and QT(2) and not QT(3);
46     -----Bidirectional counter : 0-9999-----
47     CE_0 <= '1';                                --CE='1'
48     DIGIT_0 : C10UPDN_CE_F port map( C=>C,CE=>CE_0,CLR=>CLR,DIR=>DIR,CEO=>CE_1,Q=>Q0);
49     DIGIT_1 : C10UPDN_CE_F port map( C=>C,CE=>CE_1,CLR=>CLR,DIR=>DIR,CEO=>CE_2,Q=>Q1);
50     DIGIT_2 : C10UPDN_CE_F port map( C=>C,CE=>CE_2,CLR=>CLR,DIR=>DIR,CEO=>CE_3,Q=>Q2);
51     DIGIT_3 : C10UPDN_CE_F port map( C=>C,CE=>CE_3,CLR=>CLR,DIR=>DIR,CEO=>open,Q=>Q3);
52     -----4Bits MUX4TO1-----
53     Q_DIGIT <= Q0 when F(18 downto 17)="00" else      --Select Data Digit No.0
54         Q1 when F(18 downto 17)="01" else      --Select Data Digit No.1
55         Q2 when F(18 downto 17)="10" else      --Select Data Digit No.2
56         Q3; -- F(18 downto 0)="11"           --Select Data Digit No.3
57     -----2 to 4 Decoder (One cold)-----
58     COMMON <= "1110" when F(18 downto 17)="00" else  --Select Digit No.0
59         "1101" when F(18 downto 17)="01" else  --Select Digit No.1
60         "1011" when F(18 downto 17)="10" else  --Select Digit No.2
61         "0111"; -- F(18 downto 17)="11"       --Select Digit No.3
62     -----BCD to 7 Segment decoder-----
63     Display_7SEGMENT : DECODER_7SEGMENT_F port map (A=>Q_DIGIT,Y=>Y);
64 end Behavioral;

```

รูปที่ E4.4 รายละเอียดของโค้ดหลักทั้งหมดของวงจรนับ 4 หลักในตัวอย่างที่ 2.34 ที่ปรับปรุงแก้ไขเสร็จแล้ว

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLR	Input	p53	BANK	LVCMS33	N/A	3.30					Unknown		
COMMON<0>	Output	p31	BANK	LVCMS33	N/A	3.30					Unknown		
COMMON<1>	Output	p33	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
COMMON<2>	Output	p36	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
COMMON<3>	Output	p41	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
COUNT	Input	p44	BANK	LVCMS33	N/A	3.30					Unknown		
DIR	Input	p52	BANK	LVCMS33	N/A	3.30					Unknown		
OSC	Input	p127	BANK	LVCMS33	N/A	3.30					Unknown		
Y<0>	Output	p40	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<1>	Output	p35	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<2>	Output	p32	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<3>	Output	p30	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<4>	Output	p27	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<5>	Output	p25	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<6>	Output	p23	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		

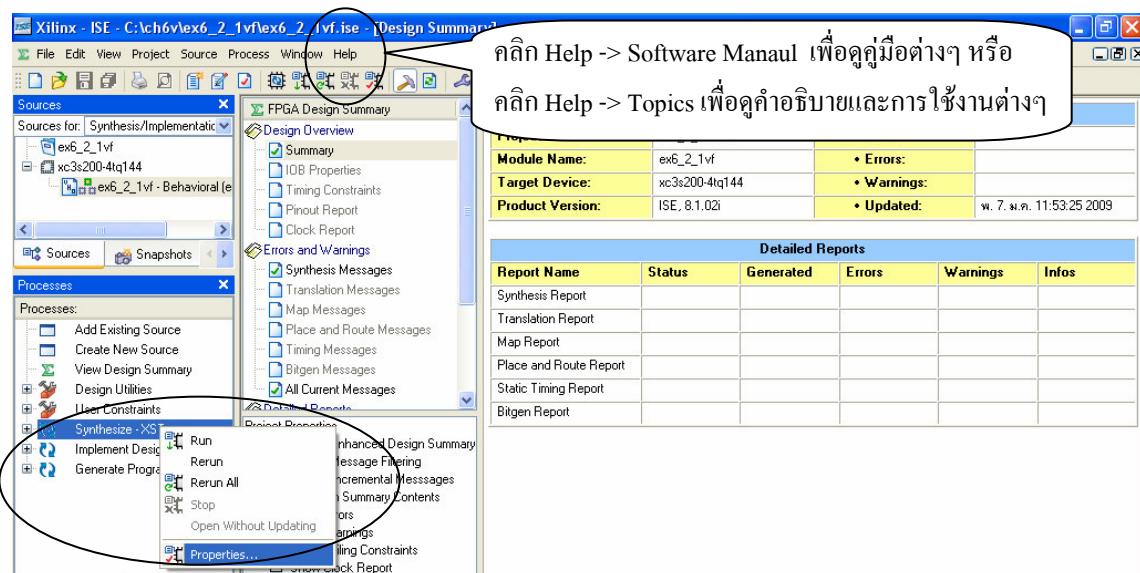
รูปที่ E4.5 การกำหนดอินพุต/เอาต์พุตที่หน้าต่างย่อย Design Object List-I/O Pins ของหน้าต่าง Xilinx-PACE

2.20 การกำหนด Property ให้กับซอฟต์แวร์ทุกในการออกแบบวงจรดิจิตอล

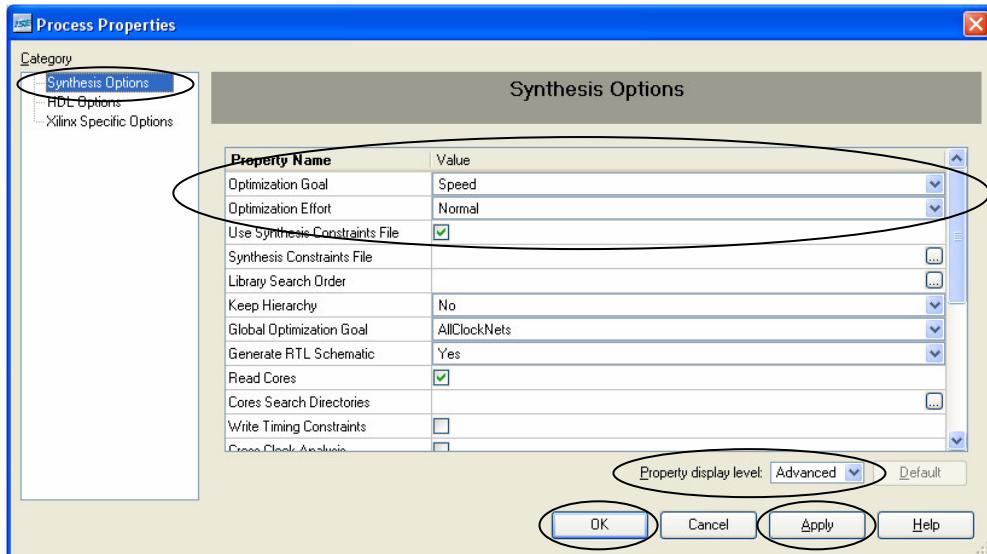
การกำหนด Property ให้กับซอฟต์แวร์ทุกในการออกแบบวงจรดิจิตอลนั้นจะเป็นการทำ Optimize ให้ได้จริงที่มีคุณสมบัติตามที่เราต้อง ซึ่งได้แก่การกำหนด Properties ต่างๆ ในขั้นตอน Synthesis ขั้นตอน Implementation และ Generate programming file เป็นต้น ซึ่งการกำหนด Properties ต่างๆ ต้องทำก่อนทำขั้นตอนนั้นๆ

ตัวอย่างการกำหนด Property ต่างๆ ในขั้นตอน Synthesis ทำได้หลังจากเปิดไฟล์ XCF3200-4TQ144 ให้คลิกขวาที่ Synthesize-XST แล้วคลิกที่ Properties แสดงดังรูปที่ 2.60 แล้วจะได้ดังรูปที่ 2.61 ซึ่งเมื่อเราเลือกที่ Category เป็น Synthesis Options เราจะพบว่าที่ซอง Optimization Goal ถูกกำหนดล่วงหน้าหรือ Default ไว้เป็น Speed ก็หมายความว่าในการทำ Synthesis จะเน้นความเร็วของวงจรเป็นหลัก แต่ถ้าเราต้องการประหดจำกันวนวนเกตหรืออุปกรณ์ที่อยู่ภายใน FPGA ก็ให้คลิกที่ แล้วคลิกเลือกเป็น Area

ในทำนองเดียวกันที่ซอง Optimization Effort ถูกกำหนดล่วงหน้าหรือ Default ไว้เป็น Normal ก็หมายความว่าในการทำ Synthesis นั้นจะคำนวนโดย Optimization เนพะสามารถเท่านั้นทำให้คำนวนได้รวดเร็ว แต่ถ้าต้องการ Optimization โดยต้องคำนึงถึงโครงสร้างภายใน FPGA ด้วยก็ให้คลิกที่ แล้วคลิกเลือกเป็น High จากนั้นให้คลิก Apply และคลิก OK ตามลำดับ ขั้นตอนในการสังเคราะห์วงจรอย่างละเอียดสามารถดูรายละเอียดจาก XST User Guide โดยคลิก Help -> Software Manual

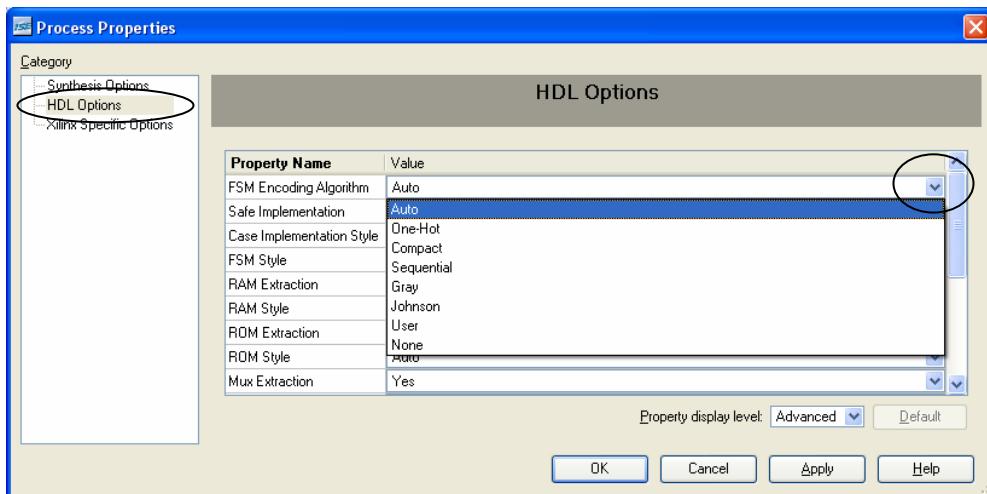


รูปที่ 2.60 ขั้นตอนการกำหนด Property ต่างๆ ก่อนทำขั้นตอน Synthesis



รูปที่ 2.61 กำหนด Properties ที่อยู่ใน Category เป็น Synthesis Options

การกำหนด Property ต่างๆ ในขั้นตอน Synthesis ถ้าเราคลิกเลือกที่ Category เพิ่มเติมที่ HDL Options ก็จะพบว่าที่ช่อง FSM Encoding Algorithm ถูกกำหนดค่าเป็น Default เป็น AUTO รูปที่ 2.62 ซึ่งก็หมายความว่าในการ Synthesis วงจรชีวนะจะเลือกที่ออกแบบโดยใช้วิธี Finite State Machine นั้นซอฟต์แวร์ทูลจะเลือก Algorithm ที่เหมาะสมเอง แต่ถ้าเราคลิกที่ แล้วคลิกเลือกเป็น User ก็จะเป็นปีกโอกาสให้ผู้ออกแบบเป็นคนกำหนด Algorithm เอง



รูปที่ 2.62 กำหนด Property ที่อยู่ใน Category เป็น HDL Options

การกำหนด Property ต่างๆ ในขั้นตอน Implementation ก็จะใช้หลักการเดียวกัน ซึ่งอาจจะต้องกำหนดเงื่อนไขบังคับทางเวลา (Timing constraints) ควบคู่กับการกำหนด Property ต่างๆ ขั้นตอนในการกำหนด Constraints สามารถดูรายละเอียดจาก Constraints Guide โดยคลิก Help -> Software

2.21 Level of abstraction

การเขียนโค้ด VHDL เพื่อ实现การทำงานของระบบหรือแบบจำลองหรือวงจรดิจิตอล เราอาจเลือกเลือกสไตล์การเขียนโค้ด (Coding style) หรือระดับรายละเอียด (Level of abstraction) ของ Architecture หรือ Architecture body ได้ 3 รูปแบบ คือ Behavioral, Dataflow และ Structural ซึ่งในทางปฏิบัติเราอาจพบว่ามีการเขียนโค้ดในรูปแบบต่างๆ ผสมกันก็ได้ เช่น กัน

- Behavioral level เป็นการเขียนโค้ดที่อธิบายพฤติกรรมการทำงานของวงจรที่คล้ายกับการเขียนโปรแกรมคอมพิวเตอร์ โดยใช้ภาษาระดับสูงทั่วๆไป ซึ่งมีลำดับในการทำงาน โดยปกติจะใช้คำสั่งชีวนิยมเช่น if-else, for, while ฯลฯ ที่ได้อธิบายในข้อ 2.14
- Dataflow level เป็นการเขียนโค้ดที่อธิบายว่าระหว่างเวลาต่อเนื่องกันมีการไหลของข้อมูลอย่างไร ซึ่งปกติจะอธิบายในรูปแบบสมการลอกิจหรือสมการบูลีน หรือใช้คำสั่งคอนโทรลเรนเดอร์ที่ได้อธิบายในข้อ 2.13
- Structural level เป็นการเขียนโค้ดที่อธิบายว่าบล็อกของวงจรย่อยที่ประกอบกันเป็นระบบหรือวงจรที่ใหญ่ขึ้นนั้นมีการเชื่อมต่อกันอย่างไร ถ้าวงจรย่อยอยู่ในระดับเกต ก็จะเรียกว่า Netlist ซึ่งเป็นการเขียนโค้ดในระดับต่ำ (Low level) และในการสังเคราะห์วงจรนั้น Synthesis tool จะแปลงไฟล์ที่เป็นภาษาระดับสูงให้เป็นไฟล์ Netlist

บางคราอาจแบ่ง Level of abstraction ออกเป็น Behavioral และ Structural โดย Behavioral จะแบ่งย่อยเป็น Dataflow และ Algorithm ซึ่ง Dataflow level นี้จะเป็นการเขียนโค้ดด้วยคำสั่งคอนโทรลเรนเดอร์ ส่วน Algorithm level นี้จะเป็นการเขียนโค้ดด้วยคำสั่งชีวนิยม เช่น คำสั่ง for, while, if-else ฯลฯ ที่เป็นภาษาระดับสูงให้เป็นแบบ Behavioral ว่าเป็นแบบ Algorithm ได้ เช่น กัน

ตัวอย่างที่ 2.39 ให้เขียนโค้ดวงจรมาติดเพล็กเซอร์แบบ 2 อินพุต 1 เอาต์พุตในด้าวอย่างที่ 2.2 ซึ่งมีผังวงแรดคองในรูปที่ 2.9 โดยให้เขียนโค้ดอธิบายละเอียด (Level of abstraction) ทั้ง 3 รูปแบบ คือ Behavioral, Dataflow และ Structural

โค้ด Behavioral style แสดงดังรูปที่ 2.63 โค้ด Dataflow และ Structural style ที่เขียนในระดับเกตแสดงดังรูปที่ 2.64 และรูปที่ 2.65 ตามลำดับ โค้ด Dataflow ที่เขียนในสูงกว่าระดับเกต เช่น คำสั่ง Selected signal assignment หรือ Conditional signal assignment เป็นต้น ดังที่ได้อธิบายละเอียดในข้อ 2.13

```

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 process(A,B,S)
14 begin
15   if S='0' then C <= A;
16   else C <= B;
17   end if;
18 end process;
19 end BEHAVIORAL;
```

รูปที่ 2.63 Behavioral (Algorithmic) coding style ของวงจร 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 begin
13   C <= (not S and A) or (S and B);
14 end BEHAVIORAL;
```

รูปที่ 2.64 Dataflow coding style ของวงจร 2 to 1 Multiplexer (ที่เขียนในระดับเกต)

```

2 -----Component code-----
3 -----Primitive Component : AND2-----
4 library IEEE;
5 use IEEE.STD_LOGIC_1164.ALL;
6
7 entity AND2 is
8     port ( A,B : in STD_LOGIC;
9             C : out STD_LOGIC);
10 end AND2;
11 architecture BEHAVIORAL of AND2 is
12 begin
13     C <= A and B;
14 end BEHAVIORAL;
15
16 -----Primitive Component : OR2-----
17 library IEEE;
18 use IEEE.STD_LOGIC_1164.ALL;
19
20 entity OR2 is
21     port ( A,B : in STD_LOGIC;
22             C : out STD_LOGIC);
23 end OR2;
24
25 architecture BEHAVIORAL of OR2 is
26 begin
27     C <= A or B;
28 end BEHAVIORAL;
29
30 -----Primitive Component : INV-----
31 library IEEE;
32 use IEEE.STD_LOGIC_1164.ALL;
33
34 entity INV is
35     port ( A : in STD_LOGIC;
36             B : out STD_LOGIC);
37 end INV;
38
39 architecture BEHAVIORAL of INV is
40 begin
41     B <= not A;
42 end BEHAVIORAL;
43
44 ----- Main code : MUX2TO1-----
45 library IEEE;
46 use IEEE.STD_LOGIC_1164.ALL;
47
48 entity MUX2TO1 is
49     port ( A,B : in STD_LOGIC;
50             S : in STD_LOGIC;
51             C : out STD_LOGIC);
52 end MUX2TO1;
53 -----Structral coding (Netlist) style-----
54 architecture BEHAVIORAL of MUX2TO1 is
55     component AND2
56         port ( A,B : in STD_LOGIC;
57                 C : out STD_LOGIC);
58     end component;
59     component OR2
60         port ( A,B : in STD_LOGIC;
61                 C : out STD_LOGIC);
62     end component;
63     component INV
64         port ( A : in STD_LOGIC;
65                 B : out STD_LOGIC);
66     end component;
67
68     signal SBAR,X,Y : STD_LOGIC;
69 begin
70     SBAR_SUB_CIRCUIT      : INV port map (A => S,B => SBAR);
71     SBAR_AND_A_SUB_CIRCUIT : AND2 port map (A => SBAR,B => A,C => X);
72     S_AND_B_SUB_CIRCUIT   : AND2 port map (A => S,B => B,C => Y);
73     X_OR_Y_SUB_CIRCUIT    : OR2 port map (A => X,B => Y,C => C);
74 end BEHAVIORAL;

```

รูปที่ 2.65 Structural coding style ของวงจร 2 to 1 Multiplexer (ที่เขียนในระดับเกต)

2.22 Transport delay และ Inertial delay

การเขียนแบบจำลองพฤติกรรม (Behavioral modeling) หรือแบบจำลองดิจิตอลเบื้องต้นเพื่อใช้ทำ Behavioral simulation โดยไม่สนใจที่จะน้าไปสังเคราะห์ว่างจรน์อาจมีได้ที่กำหนดเวลาล่าช้าหรือ Delay รวมอยู่ด้วย ซึ่ง Delay แบ่งเป็น 2 แบบคือ

- Transport delay หมายถึง Delay ธรรมชาติ โดยมีรูปแบบการเขียนดังนี้

```
TARGET <= transport EXPRESSION after DELAY_TIME;
```

- Inertial delay หมายถึง Delay ธรรมชาติ แต่จะมีข้อแม้ว่าระบบหรือแบบจำลองที่ออกแบบนั้นจะไม่ตอบสนองต่ออินพุต พลัส (บวกหรือลบ) ที่มีความกว้างน้อยกว่าค่า PULSE_WIDTH โดยที่ $PULSE_WIDTH < DELAY_TIME$ และเมื่อไม่มี [reject PULSE_WIDTH inertial] ระบบหรือแบบจำลองที่ออกแบบนั้นจะไม่ตอบสนองต่ออินพุตพลัส (บวกหรือลบ) ที่มีความกว้างน้อยกว่าค่า DELAY_TIME ซึ่ง Inertial delay มีรูปแบบการเขียนดังนี้

```
TARGET <= [ reject PULSE_WIDTH inertial ] EXPRESSION after DELAY_TIME;
```

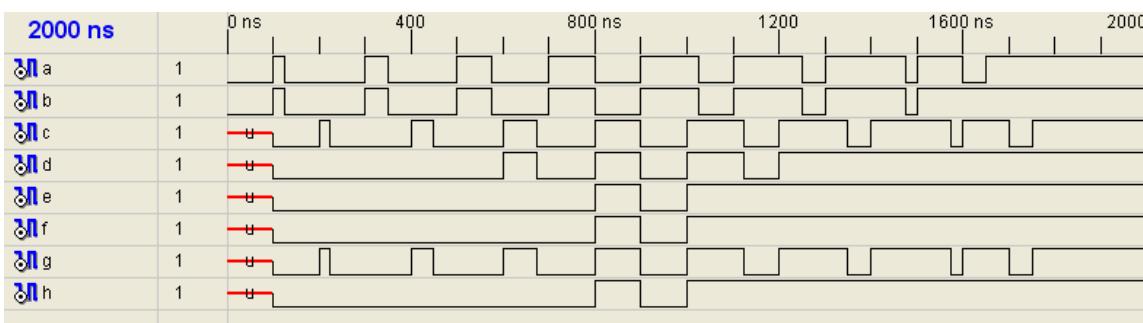
ตัวอย่างที่ 2.40 โค้ดแบบจำลองของระบบที่มี Transport delay และ Inertial delay โดยใช้คำสั่งในรูปแบบต่างๆ และผลจำลองการทำงานแสดงรูปที่ 2.66 และรูปที่ 2.67 ตามลำดับ ซึ่งมีพลัส (บวกหรือลบ) ที่ป้อนที่อินพุตกว้าง 25, 50, 75, 100, 125, 150, 175 ns

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity EX_MODELING_DELAY is
6   Port ( A,B : in STD_LOGIC;
7          C,D,E,F,G,H : out STD_LOGIC);
8 end EX_MODELING_DELAY;
9
10 architecture Behavioral of EX_MODELING_DELAY is
11 begin
12   C <= transport A after 100 ns; --Simple delay time
13   D <= reject 50 ns inertial A after 100 ns;--Reject any pulse whose width is less than 50ns.
14   E <= A after 100 ns; --Reject any pulse whose width is less than 100 ns.
15   F <= A and B after 100 ns;
16   process(A,B)
17   begin
18     G <= transport A after 100 ns; --Simple delay time
19     H <= A and B after 100 ns;
20   end process;
21 end Behavioral;

```

รูปที่ 2.66 โค้ดแบบจำลองที่มี Transport delay และ Inertial delay



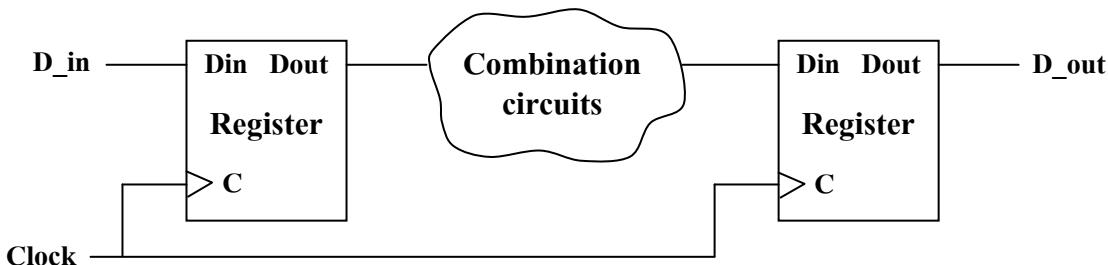
รูปที่ 2.67 ผล Behavioral simulation ของโค้ดแบบจำลองที่มี Transport delay และ Inertial delay

จากรูปที่ 2.67 เอ้าต์พุต D จะไม่ตอบสนองต่ออินพุตพัลส์ A (บวกหรือลบ) ที่มีความกว้างน้อยกว่า 50 ns เอ้าต์พุต E และเอ้าต์พุต D หรือ H จะไม่ตอบสนองต่ออินพุตพัลส์ที่มีความกว้างน้อยกว่า 100 ns

คำสั่ง Signal assignment ที่มี Transport delay หรือ Inertial delay นี้อาจนำไปเป็นไข้ในคำสั่งคอนแคร์เรนต์ได้ เช่น Selected signal assignment และ Conditional signal assignment เป็นต้น หรือคำสั่งซีเคแอนเซียล เช่น if และ case เป็นต้น

2.23 การเขียนโค้ดเพื่อการสังเคราะห์วงจร

โค้ดระดับ RTL (Register transfer level) หรือ RTL Code จะใช้ในการอธิบายพฤติกรรมของระบบที่ชัดช้อน ปกติจะใช้อธิบายพฤติกรรมระบบซึ่ง同步 (Synchronous system) โดยเขียนโค้ดลงลึกในระดับรีจิสเตอร์ (ถ้ามี) ซึ่งมองเห็นการไฟลของข้อมูลระหว่างรีจิสเตอร์โดยผ่านวงจรคอมบินेशันแบบ Clock-by-clock และคงดังรูปที่ 2.68 ซึ่งโดยทั่วไปแล้ว Synthesis tool จะสังเคราะห์วงจรโค้ดระดับ RTL มีประสิทธิภาพดีกว่าโค้ดระดับ Behavioral ดังนั้นโค้ดที่เขียนเพื่อนำไปสังเคราะห์วงจรส่วนใหญ่จะเป็นโค้ด RTL ถึงแม้ว่าในปัจจุบัน Synthesis tool สามารถสังเคราะห์วงจรโค้ดระดับ Behavioral มีประสิทธิภาพมากขึ้น แล้วก็ตาม ในบาง弋ารจะเรียกโค้ด RTL ว่าเป็นโค้ดระดับ Dataflow เพราะว่าในการเขียนโค้ดของ Architecture body นั้นโค้ดที่เกี่ยวนะว่าง begin จนไปถึง end หรือ end architecture จะอยู่ในบริเวณที่เรียกว่า Concurrent area โดยที่ทุกคำสั่งจะเป็นคำสั่งคอนแคร์เรนต์ คำสั่ง Process ก็เป็นคำสั่งคอนแคร์เรนต์เข่นกัน ถึงแม้ว่าภายในคำสั่ง Process นั้นจะเป็นคำสั่งซีเคแอนเซียลก็ตาม ตัวอย่างการออกแบบวงจรที่มีความซับซ้อนนั้นมีรายละเอียดในข้อ 2.15 ของบทที่ 2 และในบทที่ 4



รูปที่ 2.68 การไฟลของข้อมูลระหว่างรีจิสเตอร์ผ่านวงจรคอมบินेशันที่ใช้ในการเขียนโค้ดระดับ RTL

การเขียนโค้ดวงจรที่ซับซ้อนเพื่อนำไปสังเคราะห์วงจรในทางปฏิบัติจะเป็นโค้ดระดับ RTL โดยแบ่ง成หลักออกเป็นวงจรย่อยหลายวงจรหรือหลาย Component ซึ่งการเขียนวงจรย่อยในรูปของ Component มีข้อดีประการหนึ่ง คือ สามารถแยกทดสอบวงจรย่อยแต่ละวงจรในเบื้องต้นได้ก่อนที่จะนำไปประกอบรวมเป็นวงจรที่ใหญ่ขึ้น ในการเขียนโค้ด RTL นั้นรีจิสเตอร์อาจสร้างจาก Component ของรีจิสเตอร์หรือสร้าง (Inference) ด้วยการเขียนโค้ดย่อยโดยใช้คำสั่ง if ส่วนวงจรคอมบินेशันนั้นเราอาจเขียนโค้ดโดยใช้โปรแกรมย่อย (Subprogram) หรืออาจจะเลือกเขียนโค้ดในระดับใดก็ได้ กล่าวคือ Behavioral, Dataflow หรือ Structural แต่ไม่ควรเขียนโค้ดในระดับต่ำที่ต้องใช้ Component พื้นฐานที่มีอยู่ในเทคโนโลยีที่เราเลือกใช้ ซึ่งอาจจะเป็น CPLD หรือ FPGA หรือ ASIC เพราะหลักในการออกแบบด้วยภาษาระดับสูง คือ จะไม่ยืดติดกับเทคโนโลยีที่ใช้ ยกเว้นในกรณีที่เราต้องการอพชันซึ่งให้เหมาะสมกับเทคโนโลยีนั้นๆ และน้อยครั้ง Synthesis tool จะทำการอพชันซึ่งให้เหมาะสมกับเทคโนโลยีนั้นๆ เองโดยอัตโนมัติ ถึงแม้ว่าเราจะเขียนโค้ดในระดับที่สูงกว่าระดับต่ำที่ต้องใช้ Component พื้นฐานก็ตาม

บทที่ 3

ลอจิกเกตและวงจรคอมบินেชัน

3.1 ลอจิกเกต

1) ลอจิกเกตต่างๆ

ลอจิกเกตต่างๆ (Logic gates) และสมการรูบลิน (Boolean expression) มีรายละเอียดสรุปดังรูปที่ 3.1 สำหรับผู้ที่มีความรู้พื้นฐานด้านคณิตศาสตร์ก็จะเข้าใจตารางลอจิกเกตต่างๆ 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

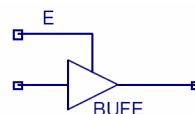
รูปที่ 3.1 ลอจิกเกตต่างๆ

2) บัฟเฟอร์และไตร-สเตตบัฟเฟอร์

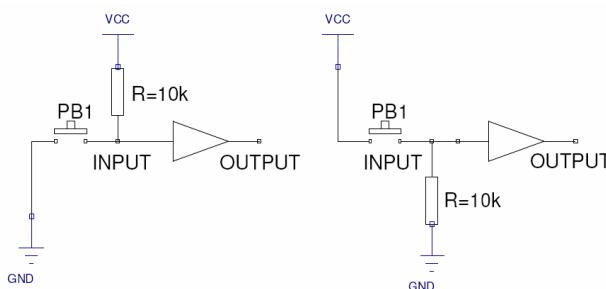
นอกจากลอจิกเกตต่างๆ แล้วยังมีบัฟเฟอร์ (Buffer) และไตร-สเตตบัฟเฟอร์ (Tri-state buffer) อีกด้วย มีรายละเอียดดังรูปที่ 3.2 และรูปที่ 3.3 ตามลำดับ อินพุตและเอาต์พุตของบัฟเฟอร์จะมีลักษณะเหมือนกัน ยกเว้นไตร-สเตตบัฟเฟอร์ดังตัวอย่างในรูปที่ 3.3 ถ้าขา E = '0' เอาต์พุตจะเป็น 'Z' หรือ High impedance แต่ถ้าขา E = '1' อินพุตและเอาต์พุตจะมีค่าอิจิกเหมือนกัน



รูปที่ 3.2 บัฟเฟอร์ (Buffer)

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

อินพุตของ ไอซีต่างๆ ที่รับจากสวิตช์คีย์บอร์ดหรือหน้าสัมผัสของเริลเบอร์ ในทางปฏิบัตินั้นจะใช้ตัวด้านทานพูลอัพ (Pull up) ต่อเข้ากับแหล่งจ่าย (V_{cc}) หรือใช้ตัวด้านทานพูลดาวน์ (Pull down) ต่อเข้ากับกราวด์ (GND) เพื่อบังคับให้อินพุตมีโลจิกเป็น ‘1’ หรือ ‘0’ ในขณะที่ยังไม่กดสวิตช์แสดงดังรูปที่ 3.4 ตัวด้านทานที่ใช้มีค่าความประมาน 2,200-10,000 โอห์ม ซึ่งผู้อ่านควรตรวจสอบรายละเอียดจากดาตاشีต (Data sheet) การปล่อยอินพุตโดยไว้ (Floating) อาจทำให้ได้ค่าโลจิกที่ไม่แน่นอนเนื่องจากอินพุตต้องการกระแสไฟแผลสหหรืออาจมีสัญญาณต่างๆ ربกวนจนทำให้อินพุตเปลี่ยนสถานะโลจิกได้ ดังนั้นอินพุตที่ไม่ได้ใช้งานจึงควรต่อเข้ากับ V_{cc} หรือ GND ยกเว้นอินพุต/เอาต์พุตของ FPGA หรือ CPLD ที่ไม่ได้กำหนดให้ใช้งานสามารถปล่อยขาโดยได้



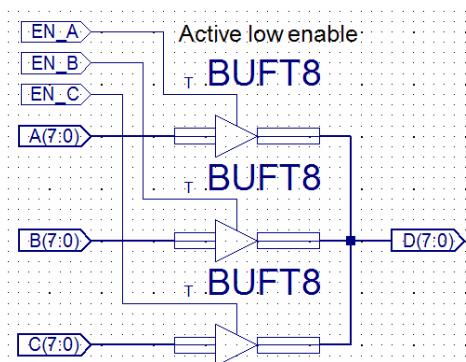
3.4 a) ขาอินพุตที่ต่อเข้ากับตัวด้านทานพูลอัพ 3.4 b) ขาอินพุตที่ต่อเข้ากับตัวด้านทานพูลดาวน์

รูปที่ 3.4 วงจรโลจิกเกตต่างๆ ที่ต่อขาอินพุตเข้ากับตัวด้านทานพูลอัพหรือพูลดาวน์

จากรูปที่ 3.4a) นั้นถ้าไม่กดสวิตช์แล้วอินพุตจะเป็นโลจิก ‘1’ แต่ถ้ากดจะเป็นโลจิก ‘0’ (เรียกว่า Active Low) ส่วนในรูปที่ 3.4b) นั้นถ้าไม่กดสวิตช์แล้วอินพุตจะเป็นโลจิก ‘0’ แต่ถ้ากดจะเป็นโลจิก ‘1’ (เรียกว่า Active high) บอร์ดทดลองทุกรุ่นที่ใช้อ้างอิงในหนังสือเล่มนี้จะใช้สวิตช์ที่เป็นปุ่มกดหรือดิพลิสวิตช์แบบ Active low ทั้งหมด

3) Multiple-drivers

ตัวอย่าง Multiple-drivers ของ Bus ขนาด 8 บิตดังรูปที่ 3.5 ซึ่งมีเอาต์พุตต่อเข้าด้วยกัน ดังนั้นถ้าต้องการเอาต์พุตบัสจากอินพุตบัสใดก็ให้ Enable ไตร-สเตตบัฟเฟอร์ของบัสชุดนั้นและ Disable ไตร-สเตตบัฟเฟอร์ของบัสอื่นๆ ที่เหลือทุกชุดเพื่อไม่ใช้อาต์พุตชนกัน



รูปที่ 3.5 ผังวงจรไตร-สเตตบัฟเฟอร์สำหรับความบันทึก 8 บิต

การเขียนโค้ด VHDL ที่มีเอาต์พุตต่อถึงกันหรือชนกันนั้นจำเป็นต้องใช้ชนิดข้อมูล std_logic หรือ std_logic_vector เท่านั้น เนื่องจากชนิดข้อมูลนี้ยอนให้ทำ Multiple-drivers ได้โดยใช้ Resolution function ดังรูปที่ 3.6 เพื่อหาค่าสถานะล็อกิกเอาต์พุต ซึ่งข้อมูลชนิดอื่นจะทำ Multiple-drivers ไม่ได้ ตัวอย่างโค้ด VHDL ของจริง (ไดร์เวอร์สำหรับควบคุมบัสขนาด 8 บิตในรูปที่ 3.5 สามารถเขียนได้ดังรูปที่ 3.7a) และรูปที่ 3.7b)

```
-- resolution function
CONSTANT resolution_table : stdlogic_table := (
-- | U X O 1 Z W L H - | |
--+-----+-----+-----+-----+-----+-----+-----+-----+-----+
( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
( 'U', 'X', 'O', 'X', 'O', 'O', 'O', 'X' ), -- | O |
( 'U', 'X', 'X', '1', '1', '1', '1', 'X' ), -- | 1 |
( 'U', 'X', 'O', '1', 'Z', 'W', 'L', 'H' ), -- | Z |
( 'U', 'X', 'O', '1', 'W', 'W', 'W', 'X' ), -- | W |
( 'U', 'X', 'O', '1', 'L', 'W', 'L', 'W' ), -- | L |
( 'U', 'X', 'O', '1', 'H', 'W', 'W', 'H' ), -- | H |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ) -- | - |
);

```

รูปที่ 3.6 Resolution function ที่นิยามไว้ใน std_logic_1164 package

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity BUS_ABC is
6     Port ( A,B,C : in STD_LOGIC_VECTOR (7 downto 0);
7             EN_A,EN_B,EN_C : in STD_LOGIC;--Active low enable
8             D : out STD_LOGIC_VECTOR (7 downto 0));
9 end BUS_ABC;
10
11 architecture Behavioral of BUS_ABC is
12 begin
13     BUS_A : process(A,EN_A)
14         begin
15             if EN_A='0' then D <= A;
16             else D <= (others => 'Z');    --"ZZZZZZZZ"
17             end if;
18     end process BUS_A;
19     BUS_B : process(B,EN_B)
20         begin
21             if EN_B='0' then D <= B;
22             else D <= (others => 'Z');    --"ZZZZZZZZ"
23             end if;
24     end process BUS_B;
25     BUS_C : process(C,EN_C)
26         begin
27             if EN_C='0' then D <= C;
28             else D <= (others => 'Z');    --"ZZZZZZZZ"
29             end if;
30     end process BUS_C;
31 end Behavioral;

```

3.7a) เขียนโดยใช้คำสั่ง if (โดยใช้ Label ชื่อ BUS_A, BUS_B และ BUS_C)

```

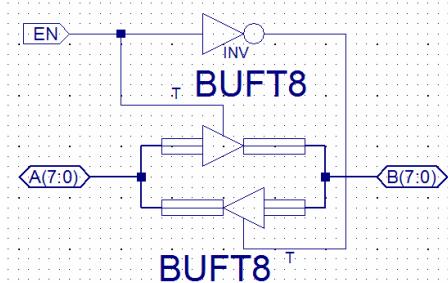
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity BUS_ABC is
6     Port ( A,B,C : in STD_LOGIC_VECTOR (7 downto 0);
7             EN_A,EN_B,EN_C : in STD_LOGIC;--Active low enable
8             D : out STD_LOGIC_VECTOR (7 downto 0));
9 end BUS_ABC;
10
11 architecture Behavioral of BUS_ABC is
12 begin
13     D <= A           when EN_A='0' else
14         (others => 'Z');          --"ZZZZZZZZ"
15     D <= B           when EN_B='0' else
16         (others => 'Z');          --"ZZZZZZZZ"
17     D <= C           when EN_C='0' else
18         (others => 'Z');          --"ZZZZZZZZ"
19 end Behavioral;

```

3.7b) โค้ดที่เขียนโดยใช้คำสั่ง Conditional signal assigment

รูปที่ 3.7 ไตร-สเตตบัสเฟอร์ว์สำหรับความคุณภาพขนาด 8 บิต

ตัวอย่าง Multiple-drivers ของ Bus transceiver ขนาด 8 บิตแสดงดังรูปที่ 3.8 เมื่อขา Enable EN = '0' แล้ว A เป็นอินพุตและ B เป็นเอาต์พุต แต่ถ้า EN = '1' แล้ว A และ B จะมีทิศทางกลับกัน ตัวอย่างโค้ด VHDL ของ Bus transceiver ขนาด 8 บิตแสดงดังรูปที่ 3.9a) ถึงรูปที่ 3.9d) ดังนั้นถ้าต้องการเอาต์พุตบัสจาก A ก็ให้ Enable ไตร-สเตตบัสเฟอร์ของบัสชุดบนและ Disable ไตร-สเตตบัสเฟอร์ของบัสชุดล่างเพื่อให้อเอาต์พุตนี้เป็น High impedance เพื่อป้องกันปัญหา Multiple-drivers ที่จะเกิดขึ้นจากเอาต์พุตของไตร-สเตตบัสเฟอร์ของชุดล่างชนกับสัญญาณที่ป้อนเข้าอินพุต A แต่ถ้าต้องการเอาต์พุตจาก B ก็ให้ทำกลับกัน



รูปที่ 3.8 ผังวงจรของ Bus transceiver ขนาด 8 บิต

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity BUS_TRANSCEIVER_V is
6     Port ( A,B : inout STD_LOGIC_VECTOR (7 downto 0);
7             EN : in STD_LOGIC);
8 end BUS_TRANSCEIVER_V;
9 architecture Behavioral of BUS_TRANSCEIVER_V is
10 begin
11     -----Top buffers-----
12     with EN select
13         B <=  A      when '0',
14             (others => 'Z') when others;
15     -----Bottom buffers-----
16     with EN select
17         A <=  B      when '1',
18             (others => 'Z') when others;
19 end Behavioral;

```

3.9a) ผังวงจรของ Bus transceiver ขนาด 8 บิตที่เขียนด้วยคำสั่ง Selected signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity BUS_TRANSCEIVER_V is
6     Port ( A,B : inout STD_LOGIC_VECTOR (7 downto 0);
7             EN : in STD_LOGIC);
8 end BUS_TRANSCEIVER_V;
9 architecture Behavioral of BUS_TRANSCEIVER_V is
10 begin
11     -----Top buffers-----
12     B <= A when EN='0' else (others => 'Z');
13     -----Bottom buffers-----
14     A <= B when EN='1' else (others => 'Z');
15 end Behavioral;

```

3.9b) ผังวงจรของ Bus transceiver ขนาด 8 บิตที่เขียนด้วยคำสั่ง Conditional signal assignment

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity BUS_TRANSCEIVER_V is
6     Port ( A,B : inout STD_LOGIC_VECTOR (7 downto 0);
7             EN : in STD_LOGIC);
8 end BUS_TRANSCEIVER_V;
9 architecture Behavioral of BUS_TRANSCEIVER_V is
10 begin
11 process(A,B,EN)
12 begin
13     if      EN='0' then B <= A;                      --Top buffers
14                     A <= (others => 'Z');--Bottom buffers
15     elsif   EN='1' then A <= B;                      --Bottom buffers
16                     B <= (others => 'Z');--Top buffers
17     else               B <= (others => 'Z');--Top buffers
18                     A <= (others => 'Z');--Bottom buffers
19     end if;
20 end process;
21 end Behavioral;

```

3.9c) โค้ดวงจรของ Bus transceiver ขนาด 8 บิตที่เขียนด้วยคำสั่ง if

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity BUS_TRANSCEIVER_V is
6     Port ( A,B : inout STD_LOGIC_VECTOR (7 downto 0);
7             EN : in STD_LOGIC);
8 end BUS_TRANSCEIVER_V;
9 architecture Behavioral of BUS_TRANSCEIVER_V is
10 begin
11 process(A,B,EN)
12 begin
13     case EN is
14         when '0'    => B <= A;                      --Top buffers
15                     A <= (others => 'Z');--Bottom buffers
16         when '1'    => A <= B;                      --Bottom buffers
17                     B <= (others => 'Z');--Top buffers
18         when others => B <= (others => 'Z');--Top buffers
19                     A <= (others => 'Z');--Bottom buffers
20     end case;
21 end process;
22 end Behavioral;

```

3.9d) โค้ดวงจรของ Bus transceiver ขนาด 8 บิตที่เขียนด้วยคำสั่ง case

การทดลองที่ 3.1.1 ลอจิกเกตต่างๆ

วัตถุประสงค์

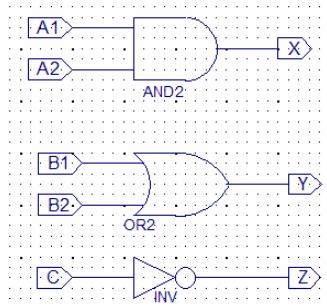
- 1) เพื่อทำความเข้าใจเกี่ยวกับแอนด์เกต ออร์เกต และ อินเวอร์เตอร์
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างลอจิกเกตและโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างแอนด์เกต ออร์เกต และอินเวอร์เตอร์ด้วย CPLD

ให้ออกแบบวงจรแอนด์เกต ออร์เกต และอินเวอร์เตอร์ด้วย CPLD ดังรูปที่ L1.1 ซึ่งมีโค้ดแสดงดังรูปที่ L1.2 ตามขั้นตอนในบทที่ 2 ข้อ 2.16 โดยที่ก่อนเข้าโปรแกรม ISE WebPACK นั้นให้สร้าง Folder ชื่อ ch3v ไว้ในไดรฟ์ C จากนั้นให้เก็บไฟล์ Project Location (หรือ Folder) ชื่อ ch3v และกำหนดให้ Project Name และ Source File ชื่อ ex3_1_1vcxl



รูปที่ L1.1 ผังวงจรลอจิกเกตต่างๆ

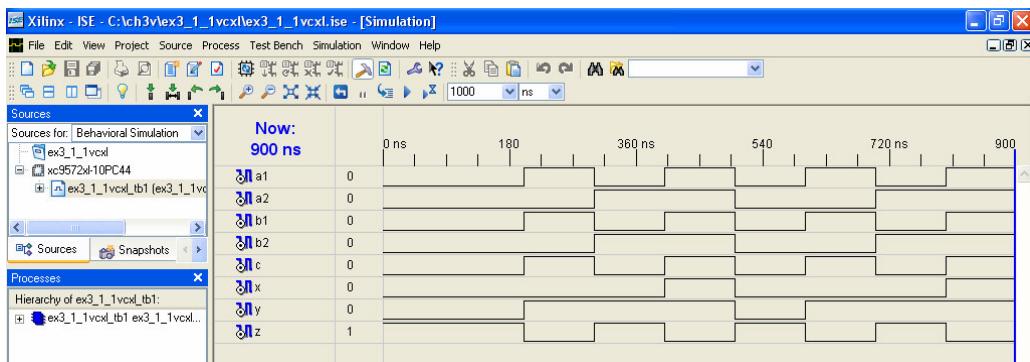
```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_1_1vcxl is
6     Port ( A1,A2 : in STD_LOGIC;
7             B1,B2 : in STD_LOGIC;
8             C : in STD_LOGIC;
9             X : out STD_LOGIC;
10            Y : out STD_LOGIC;
11            Z : out STD_LOGIC);
12 end ex3_1_1vcxl;
13
14 architecture Behavioral of ex3_1_1vcxl is
15 begin
16     X <= A1 and A2;
17     Y <= B1 or B2;
18     Z <= not C;
19 end Behavioral;

```

รูปที่ L1.2 โค้ด VHDL ของวงจรแอนด์เกต ออร์เกต และอินเวอร์เตอร์ (Dataflow coding style)

จากนั้นให้ทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 ข้อ 2.16.2 โดยใช้ Source File ชื่อ ex3_1_1vcxl_tb1 (ห้ามใช้ชื่อเดียวกับ ex3_1_1vcxl) เตรียมแล้วให้พิจารณาว่าผล Simulation ในรูปที่ L1.3 ว่าเป็นไปตามทฤษฎีหรือไม่



รูปที่ L1.3 ผล Behavioral simulation ของวงจรที่ออกแบบ

การกำหนดสายสัญญาณต่างๆ ของวงจรในรูปที่ L1.1 เข้ากับขา CPLD (มีรายละเอียดในบทที่ 1 ตารางที่ 1.6) ซึ่งเราจะใช้ปุ่มกด (Push button switch) PB1 ถึง PB5 เป็นอินพุตและ LED1 ถึง LED3 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{lll} A1 = PB1 = \text{INPUT} = p39 & B2 = PB4 = \text{INPUT} = p43 & X = \text{LED1} = \text{OUTPUT} = p38 \\ A2 = PB2 = \text{INPUT} = p40 & C = PB5 = \text{INPUT} = p44 & Y = \text{LED2} = \text{OUTPUT} = p37 \\ B1 = PB3 = \text{INPUT} = p42 & & Z = \text{LED3} = \text{OUTPUT} = P36 \end{array}$$

จากนั้นให้พิมพ์รายละเอียดต่างๆ ใน Assign Package Pins สรุปได้ดังนี้

Xilinx PACE - [Design Object List - I/O Pins]						
I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
A1	Input	p39	2	9		
A2	Input	p40	2	11		
B1	Input	p42	2	14		
B2	Input	p43	2	15		
C	Input	p44	2	17		
X	Output	p38	2	8	SLOW	
Y	Output	p37	2	6	SLOW	
Z	Output	p36	2	5	SLOW	

รูปที่ L1.4 การกำหนดขาในหน้าต่าง Xilinx-PACE

หลังจากโปรแกรมจะที่ออกแบบลงชิป CPLD แล้วให้ทดลองกดปุ่ม PB1-PB5 และให้สังเกตดูผลที่ LED1-LED3 ว่าให้ล็อกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นบันทึกผลการทดลอง และพิจารณาอยู่เสมอว่า Push button switch ทำงานเป็นแบบ Active low ซึ่งหมายความว่าถ้ากด Push button switch แล้วจะให้ล็อก ‘0’ เช่น ถ้าเราไม่กดปุ่ม PB1 และ PB2 ก็แสดงว่าอินพุตของแอนด์เกตคือ A1 = ‘1’ และ A2 = ‘1’ หากผลการทดลองเป็นตามทฤษฎี เอาต์พุตของแอนด์เกตต้องให้ล็อก ‘1’ คือ X = ‘1’ ดังนั้น LED1 จะต้องติดสว่าง ข้อควรระวังอีกประการหนึ่งคือ PB3-PB6 ต่อพ่วงอยู่กับ Slide SW1-Slide SW4 ตามลำดับ ตัวอย่างเช่น เมื่อ ON Slide SW1 แล้วจะให้ผลเข้าเดียวกับการกดปุ่ม PB3 ถ้าไกว ดังนั้นหากต้องการใช้ปุ่มกดตัวใดตัวหนึ่งเราจะต้อง OFF Slide SW ที่ต่อพ่วงดังนี้เสียก่อน

2 สร้างแอนด์เกต ออร์เกต และ อินเวอร์เตอร์ด้วย FPGA

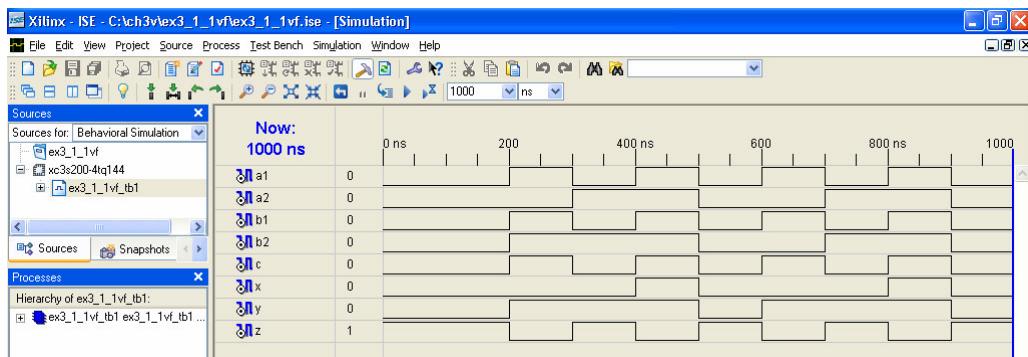
ให้ออกแบบวงจรแอนด์เกต ออร์เกต และอินเวอร์เตอร์ด้วย FPGA ดังรูปที่ L1.1 ที่มีโค้ดแสดงดังรูปที่ L2.1 ตามขั้นตอนในบทที่ 2 ข้อ 2.17 ก่อนเข้าโปรแกรม ISE WebPACK ถ่ายไม่ได้สร้าง Folder ชื่อ ch3v ไว้ในไดรฟ์ C ที่ให้สร้าง Folder ชื่อ ch3v ไว้ในไดรฟ์ C ก่อน แล้วเปลี่ยนจีบโค้ดไว้ใน Project Location (หรือ Folder) ชื่อ ch3v และกำหนดให้ Project Name และ Source File ชื่อ ex3_1_vf จากนั้นให้ทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 ข้อ 2.17.2 โค้ดใช้ Source File ชื่อ ex3_1_vf_tb1 (ห้ามใช้ชื่อซ้ำกับ ex3_1_vf) เสร็จแล้วให้พิจารณาว่าผล Simulation ในรูปที่ L2.2 ว่าเป็นไปตามทฤษฎีหรือไม่

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_1_lvf is
6     Port ( A1,A2 : in STD_LOGIC;
7             B1,B2 : in STD_LOGIC;
8             C : in STD_LOGIC;
9             X : out STD_LOGIC;
10            Y : out STD_LOGIC;
11            Z : out STD_LOGIC);
12 end ex3_1_lvf;
13
14 architecture Behavioral of ex3_1_lvf is
15 begin
16     X <= A1 and A2;
17     Y <= B1 or B2;
18     Z <= not C;
19 end Behavioral;

```

รูปที่ L2.1 โค้ด VHDL ของวงจรและอนค์เกต ออร์เกต และอินเวอร์เตอร์ (Dataflow coding style)



รูปที่ L2.2 ผล Behavioral simulation ของวงจรที่ออกแบบ

การกำหนดสายสัญญาณต่างๆ ของวงจรในรูปที่ L1.1 เช้ากับ FPGA (มีรายละเอียดในบทที่ 1 ตามตารางที่ 1.7) ซึ่งเราจะใช้ปุ่มกด (Push button switch) PB1 ถึง PB5 เป็นอินพุตและ LED L1 ถึง L3 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{lll}
 A1 = PB1 = \text{INPUT} = p44 & B2 = PB4 = \text{INPUT} = p50 & X = L3 = \text{OUTPUT} = p76 \\
 A2 = PB2 = \text{INPUT} = p46 & C = PB5 = \text{INPUT} = p51 & Y = L2 = \text{OUTPUT} = p69 \\
 B1 = PB3 = \text{INPUT} = p47 & & Z = L1 = \text{OUTPUT} = P77
 \end{array}$$

จากนั้นให้พิมพ์รายละเอียดต่างๆ ใน Assign Package Pins สรุปได้ดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
A1	Input	p44	BANK5	LVCMS33	N/A	3.30					Unknown		
A2	Input	p46	BANK5	LVCMS33	N/A	3.30					Unknown		
B1	Input	p47	BANK5	LVCMS33	N/A	3.30					Unknown		
B2	Input	p50	BANK5	LVCMS33	N/A	3.30					Unknown		
C	Input	p51	BANK5	LVCMS33	N/A	3.30					Unknown		
X	Output	p76	BANK3	LVCMS33	N/A	3.30		SLOW			Unknown		
Y	Output	p69	BANK4	LVCMS33	N/A	3.30		SLOW			Unknown		
Z	Output	p77	BANK3	LVCMS33	N/A	3.30		SLOW			Unknown		

รูปที่ L2.3 การกำหนดขาในหน้าต่าง Xilinx-PACE

หลังจากโปรแกรมวงจรที่ออกแบบลงชิป FPGA แล้วให้ทดลองกดปุ่ม PB1-PB5 และให้สังเกตคุณลักษณะ LED L1-L3 ว่าให้ล้อจิกเอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ จากนั้นบันทึกผลการทดลอง และพึงระลึกอยู่เสมอว่า Push button switch และ Dip switch ทุกตัวทำงานแบบ Active low ซึ่งก็หมายความว่าถ้ากด Push button switch แล้วจะให้ล้อจิก ‘0’ เช่น ถ้าเราไม่กดปุ่ม PB1 และ PB2 ก็แสดงว่าอินพุตของแอนค์เกตคือ A1 = ‘1’ และ A2 = ‘1’ หากผลการทดลองเป็นตามทฤษฎี เอาต์พุตของแอนค์เกตต้องให้ล้อจิก ‘1’ คือ X = ‘1’ ดังนั้น L3 จะต้องติดสว่าง

การทดลองที่ 3.1.2 Multiple-drivers

วัตถุประสงค์

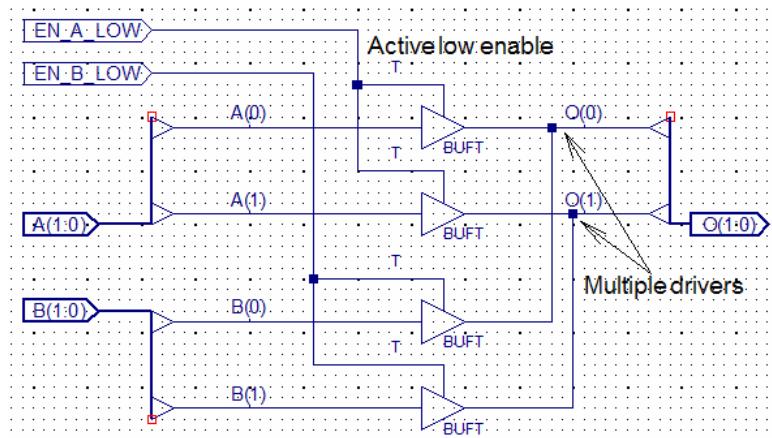
- 1) เพื่อทำความเข้าใจเกี่ยวกับ Multiple-drivers ของ Bus ขนาด 2 บิตซึ่งมีเอาต์พุตแต่ละเอาต์พุตต่อเข้าด้วยกัน
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างไตร-สเตตบัสเฟอร์ของ Bus และโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

บอร์ด CPLD Explorer XC9572XL และใช้ FPGA Surveyor-III XC3S200F หรือ FPGA Surveyor-III XC3S200F4 ที่มี LED 3 สถานะ (Logic monitor) แทนบอร์ด FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4 ตามลำดับ

1 สร้างไตร-สเตตบัสเฟอร์ของ Bus ขนาด 2 บิตด้วย CPLD

ให้ออกแบบวงจรไตร-สเตตบัสเฟอร์ของ Bus ขนาด 2 บิตด้วย CPLD ดังรูปที่ L1.1 และมีโค้ดแสดงดังรูปที่ L1.2 โดยใช้ Project Location (หรือ Folder) ชื่อ ch3v กำหนด Project Name และ Source File ชื่อ ex3_1_2vcx1



รูปที่ L1.1 ผังวงจรไตร-สเตตบัสเฟอร์ของ Bus ขนาด 2 บิต

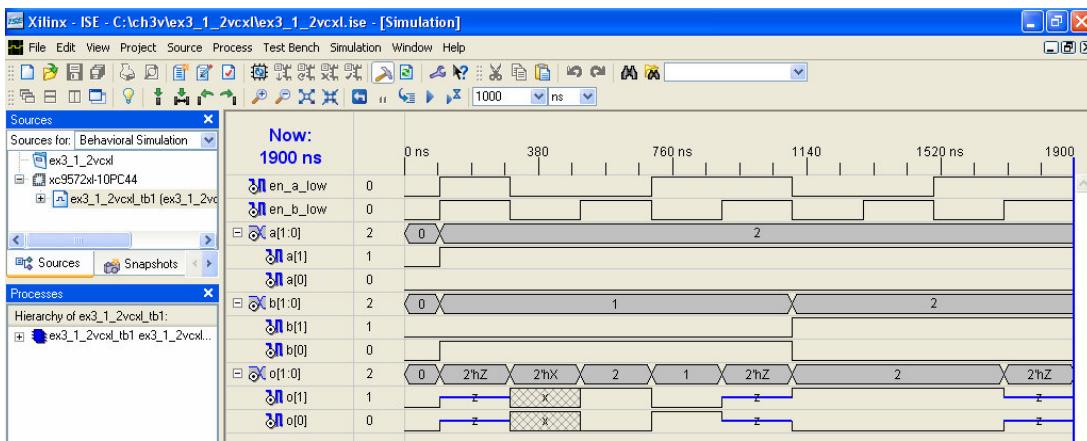
```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_1_2vcx1 is
6   Port ( EN_A_LOW,EN_B_LOW : in STD_LOGIC;
7          A,B : in STD_LOGIC_VECTOR (1 downto 0);
8          O : out STD_LOGIC_VECTOR (1 downto 0));
9 end ex3_1_2vcx1;
10
11 architecture Behavioral of ex3_1_2vcx1 is
12 begin
13   O <= A when EN_A_LOW = '0' else (others => 'Z');
14   O <= B when EN_B_LOW = '0' else (others => 'Z');
15 end Behavioral;

```

รูปที่ L1.2 โค้ด VHDL ของวงจรไตร-สเตตบัสเฟอร์ของ Bus ขนาด 2 บิต (Dataflow coding style)

จากนั้นให้ทำ Behavioral simulation ตามขั้นตอนในบทที่ 2 ข้อ 2.16.2 โดยใช้ Source File ชื่อ ex3_1_2vcx1_tb1 เสร็จแล้วให้พิจารณาว่าผล Simulation ในรูปที่ L1.3 ว่าเป็นไปตามทฤษฎีหรือไม่



รูปที่ L1.3 ผล Behavioral simulation ของวงจรที่ออกแบบ

การกำหนดขาสัญญาณต่างๆ เป็นดังนี้

A0 = Slide SW2 = INPUT = p43	B0 = Slide SW4 = INPUT = p1	O0 = MN1 = OUTPUT = p7
A1 = Slide SW1 = INPUT = p42	B1 = Slide SW3 = INPUT = p44	O1 = MN2 = OUTPUT = p6
EN_A_LOW = PB1 = INPUT = p39		
EN_B_LOW = PB2 = INPUT = p40		

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้คือ

Xilinx PACE - [Design Object List - I/O Pins]							
I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals	
EN_A_LOW	Input	p39	2	9			
EN_B_LOW	Input	p40	2	11			
A<1>	Input	p42	2	14			
A<0>	Input	p43	2	15			
B<1>	Input	p44	2	17			
B<0>	Input	p1	1	2			
O<1>	Output	p7	1	14	SLOW		
O<0>	Output	p5	1	11	SLOW		

รูปที่ L1.4 Assign Package Pins

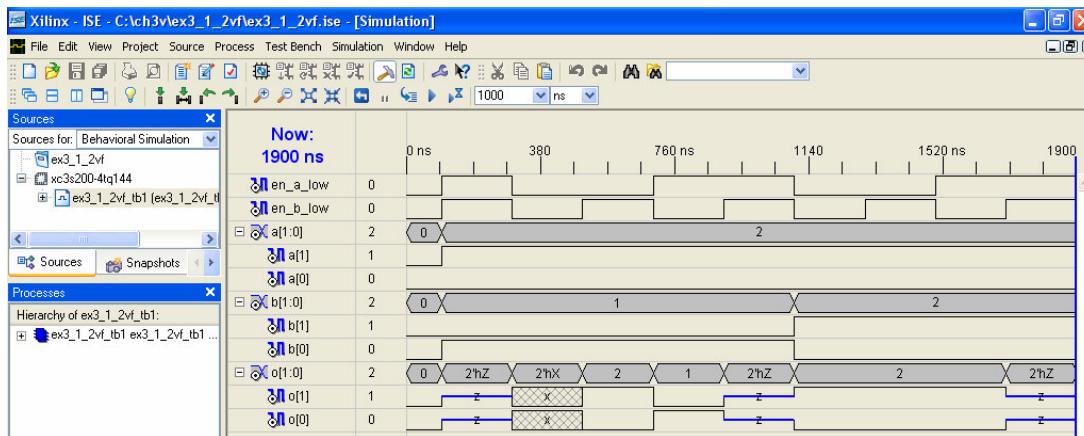
หลังจากโปรแกรมลงชิป CPLD แล้วให้ ON Slide SW1 และ OFF Slide SW2 แล้ว OFF Slide SW3 และ ON Slide SW4 ให้ปั๊บปุ่ม PB1-PB2 พร้อมกันแล้วให้สังเกตดูผลที่ MN1-MN2 (釅 = '0', เหลือง = '1', ดับ = 'Z') ว่าให้คูลอจิกເອົາດີພຸດເປັນໄປຕາມທຸກໆຫຼືໄວ່ມີ จากນີ້ໃຫ້ທົດລອງຄູ່ປຸ່ມ PB1-PB2 พร้อมກันแล้วให้ສังເກດຜົດການທົດລອງທີ່ MN1-MN2 ອີກຮັງຈຳວ່າໃຫ້ລອຈິກເອົາດີພຸດເປັນໄປຕາມທຸກໆຫຼືໄວ່ມີ ສະນະລອຈິກ 'Z' ແລະ 'X' ທີ່ໄດ້ຈາກການທົດລອງຈົງແລະຈາກ ISE Simulator ຕ່າງກັນ ຫຼືໄວ່ມີ (ທ່ານຄົດວ່າຜົດລັບໜີໃດຄູກຕ້ອງ) ເສົ່ງແລ້ວກູ່ປຸ່ມ PB1 ແລະ ປຸ່ມ PB2 ໃຫ້ຄູ່ຜົດອີກຮັງ ຈາກນີ້ທຳກັນໂດຍປ່ອຍປຸ່ມ PB1 ແລະ ຄູ່ປຸ່ມ PB2 ໃຫ້ຄູ່ຜົດອີກຮັງ ຈາກນີ້ເຮັມທົດລອງໃໝ່ທີ່ໜີແຕ່ເປັນ ON Slide SW3 ແລະ OFF Slide SW4 ດູ້ບ້າງ ນັ້ນທີ່ກົດການທົດລອງ

2 ສ້າງໄຕຣ-ສເຕຕບັຟເຝອຮ່ອງ Bus ຂາດ 2 ປິຕໍດ້ວຍ FPGA

ໃຫ້ອຳນວຍງານໃຕຣ-ສເຕຕບັຟເຝອຮ່ອງ Bus ຂາດ 2 ປິຕໍດ້ວຍ FPGA ດັ່ງຮູບທີ L1.1 ແລະ ມີໂຄ້ດແສດງດັ່ງຮູບທີ L2.1 ໂດຍໃຊ້ Project Location (ຫຼືອ Folder) ຊື່ ch3v ກໍານັດ Project Name ແລະ Source File ຊື່ ex3_1_2vf ຈາກນີ້ທຳ Behavioral simulation ໂດຍໃຊ້ Source File ຊື່ ex3_1_2vf_tb1 ເສົ່ງແລ້ວໃຫ້ພິຈານວ່າຜົດ Simulation ຮູບທີ L2.2 ວ່າເປັນໄປຕາມທຸກໆຫຼືໄວ່

```
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_1_2vf is
6     Port ( EN_A_LOW,EN_B_LOW : in STD_LOGIC;
7             A,B : in STD_LOGIC_VECTOR (1 downto 0);
8             O : out STD_LOGIC_VECTOR (1 downto 0));
9 end ex3_1_2vf;
10
11 architecture Behavioral of ex3_1_2vf is
12 begin
13     O <= A when EN_A_LOW = '0' else (others => 'Z');
14     O <= B when EN_B_LOW = '0' else (others => 'Z');
15 end Behavioral;
```

รูปที่ L2.1 โค้ด VHDL ของวงจรไตร-สเตตบัสเฟอร์ของ Bus ขนาด 2 บิต (Dataflow coding style)



รูปที่ L2.2 ผล Behavioral simulation ของวงจรที่ออกแบบ

การกำหนดขาสัญญาณต่างๆ เป็นดังนี้

A0 = Slide SW2 = INPUT = p23 B0 = Slide SW0 = INPUT = p25 O0 = MN0 = OUTPUT = p51

A1 = Slide SW3= INPUT = p21 B1 = Slide SW1 = INPUT = p24 O1 = MN1 = OUTPUT = p50

EN_A_LOW= PB5= INPUT= p13 EN_B_LOW= PB6= INPUT = p14

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้คือ

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcc0	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
B<1>	Input	p24	BANK8	LVCMSO33	N/A	3.30					Unknown		
EN_A_LOW	Input	p13	BANK7	LVCMSO33	N/A	3.30					Unknown		
A<1>	Input	p21	BANK6	LVCMSO33	N/A	3.30					Unknown		
EN_B_LOW	Input	p14	BANK7	LVCMSO33	N/A	3.30					Unknown		
A<0>	Input	p23	BANK6	LVCMSO33	N/A	3.30					Unknown		
O<0>	Output	p51	BANK8	LVCMSO33	N/A	3.30			SLOW		Unknown		
B<0>	Input	p25	BANK6	LVCMSO33	N/A	3.30					Unknown		
O<1>	Output	p50	BANK8	LVCMSO33	N/A	3.30			SLOW		Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมลงชิพ FPGA แล้วให้ ON Slide SW3 และ OFF Slide SW2 แล้ว OFF Slide SW1 และ ON Slide SW0 ให้ปล่อยปุ่ม PB5-PB6 พร้อมกันแล้วให้คูpler ที่ MN0-MN1 (แดง = '0', เหลือง = '1', ดับ = 'Z') ว่าให้อาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นให้กดปุ่ม PB5-PB6 พร้อมกันแล้วให้คูpler ที่ MN0-MN1 อิกครึ่งว่าให้ล็อกจิกআত์พุตเป็นไปตามทฤษฎีหรือไม่ สถานะโลจิก 'Z' และ 'X' ที่ได้จากการทดลองจริงและจาก ISE Simulator ต่างกันหรือไม่ (ท่านคิดว่าผลลัพธ์ใดถูกต้อง) เสร็จแล้วกดปุ่ม PB5 และปล่อยปุ่ม PB6 ให้คูpler อิกครึ่ง จากนั้นทำสลับกันโดยปล่อยปุ่ม PB5 และกดปุ่ม PB6 ให้คูpler อิกครึ่ง จากนั้นเริ่มทดลองใหม่ทั้งหมดแต่เปลี่ยนเป็น ON Slide SW1 และ OFF Slide SW0 บันทึกผลการทดลอง

การทดลองที่ 3.1.3 บัฟเฟอร์สองทิศทาง

ວັດຖຸປະສົງຄໍ

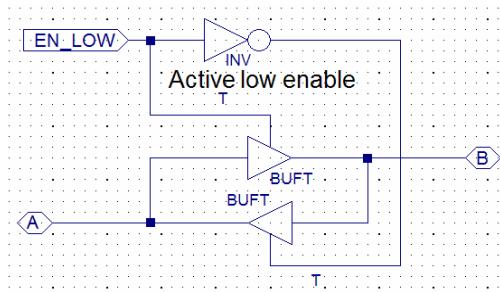
- 1) เพื่อทำความเข้าใจเกี่ยวกับบันฟเฟอร์สองทิศทาง
 - 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างบันฟเฟอร์สองทิศทางและโปรแกรมลงชิป CPLD

ອຸປະກອນໍທດລອງ

บอร์ด CPLD Explorer XC9572XL

1 สร้างน้ำฟเฟอร์ส่องทิศทางด้วย CPLD

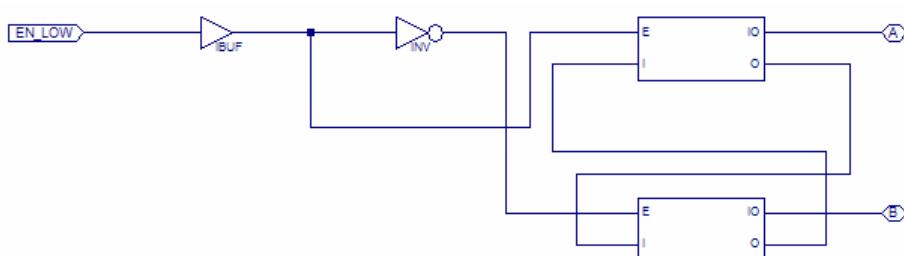
ให้ออกแบบบัฟเฟอร์สองทิศทางด้วย CPLD ดังรูปที่ L1.1 ที่มีโภคແສດງดังรูปที่ L1.2 โดยใช้ Project Location ชื่อ ch3v กำหนด Project Name และ Source File ชื่อ ex3_1_3vcx1 หลังจากสังเคราะห์วงจรแล้วให้พิจารณาว่า View Technology schematic ดังรูปที่ L1.3 ถูกต้องและเป็นตามทฤษฎีหรือไม่



รูปที่ L1.1 ผังวงจรบีฟเฟอร์สองทิศทาง

```
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_1_3vcxl is
6     Port ( EN_LOW : in STD_LOGIC;
7             A,B : inout STD_LOGIC);
8 end ex3_1_3vcxl;
9
10 architecture Behavioral of ex3_1_3vcxl is
11 begin
12     -----
13     --Top buffer--
14     B <= A when EN_LOW ='0' else 'Z';
15     -----
16     --Bottom buffer--
17     A <= B when EN_LOW ='1' else 'Z';
18 end Behavioral;
```

รูปที่ L1.2 โค้ด VHDL ของวงจรบัฟเฟอร์สองทิศทาง



รูปที่ L1.3 View Technology schematic ของวงจรบันฟเฟอร์สองทิศทาง โดยใช้ CPLD

การกำหนดขาสัญญาณต่างๆ เป็นดังนี้

A = IO6 = INPUT/ OUTPUT, MN1 = p7

B = IO7 = INPUT/ OUTPUT, MN2 = p6

EN_LOW = Slide SW1 = INPUT = p42

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้คือ

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
EN_LOW	Input	p42	2	14		
A	InOut	p7	1	14	SLOW	
B	InOut	p6	1	11	SLOW	

รูปที่ L1.4 Assign Package Pins

หลังจากโปรแกรมจะที่ออกแบบลงชิป CPLD แล้วให้ทดสอบวงจรบัสเฟอร์ส่องทิศทาง โดยให้ OFF Slide SW1 ('1') แล้วให้ต่อสายไฟที่ข้าว IO6 (Pin No.13 ของ K1) กับข้าว Vcc= 3.3V (Pin No.2 ของ K1) สลับกันไปมา กับข้าว GND (Pin No.4 ของ K1) แล้วสังเกตคุณผลที่ LED MN2 ('1' = เหลือง, '0' = แดง, ดับ = High impedance) หรือ IO7 จากนั้นให้ ON Slide SW1 ('0') แล้วให้ต่อสายไฟที่ข้าว IO7 (Pin No.15 ของ K1) กับข้าว Vcc= 3.3V (Pin No.2 ของ K1) สลับกันไปมา กับข้าว GND (Pin No.4 ของ K1) แล้วสังเกตคุณผลที่ LED MN1 (IO6) ทำการบันทึกผล และให้สรุปว่าผลจิกที่ได้เป็นไปตามทฤษฎีหรือไม่

จากนั้นให้แก้โค้ดเป็นดังรูป L1.5 แล้วทำการทดลองซ้ำอีกครั้ง

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_1_3vcx1 is
6     Port ( EN_LOW : in STD_LOGIC;
7             A,B : inout STD_LOGIC);
8 end ex3_1_3vcx1;
9
10 architecture Behavioral of ex3_1_3vcx1 is
11 begin
12 process(EN_LOW,A,B)
13 begin
14     if EN_LOW ='0' then      B <= A;    --Top buffer = enable
15                           A <= 'Z';--Bottom buffer = disable
16     elsif EN_LOW ='1' then   B <= 'Z';--Bottom buffer = disable
17                           A <= B;   --Top buffer = enable
18     else                   B <= 'Z';--Bottom buffer = disable
19                           A <= 'Z';--Top buffer = disable
20     end if;
21 end process;
22 end Behavioral;

```

รูปที่ L1.5 โค้ด VHDL ของวงจรบัสเฟอร์ส่องทิศทาง (Behavioral coding style)

หมายเหตุ บอร์ด FPGA Discovery-III XC3S200 ไม่มี LED Logic monitor จึงไม่มีการทดลองนี้

แบบผึ่งหัด

1) ให้อธิบายโค้ดของบัสเฟอร์ส่องทิศทางว่าทำงานอย่างไร

2) ให้เขียนโค้ด VHDL บัสเฟอร์ส่องทิศทางโดยใช้คำสั่ง Selected signal assignment และโดยใช้คำสั่ง Case

3.2 วงจรบวก-ลบ

วงจรบวกเป็นวงจรที่ประกอบด้วยลòจิกเกตต่างๆ หรือวงจรคอมบินेशัน (Combinational Logic circuits) ซึ่งโภคดของวงจรเหล่านี้อาจเขียนด้วยคำสั่งซีเควนเซียลและ/หรือคำสั่งคอนโทรลเรนต์ก็ได้ ในขั้นตอนนี้เราจะอธิบายวงจรบวกเพียง 2 แบบ คือ Half adder และ Full adder

วงจร Half adder เป็นวงจรบวกที่มี 2 อินพุตและ 2 เอาต์พุต มีตารางความจริง (Truth Table) แสดงดังรูปที่ 3.10

Input		Output	
A	B	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

รูปที่ 3.10 ตารางความจริงของวงจร Half adder

จากตารางความจริงของวงจร Half adder ขนาด 1 บิตในรูปที่ 3.10 เราสามารถเขียนสมการบูลีนจะได้ดังนี้

$$\begin{aligned} \text{Sum} &= \overline{A} \cdot B + A \cdot \overline{B} \quad (\text{สมการบูลีนในรูปฟอร์มของ Sum of product}) \\ &= A \oplus B \\ \text{Cout} &= A \cdot B \end{aligned}$$

จากสมการบูลีนเราสามารถเขียนโค้ด VHDL ของวงจร Half adder ขนาด 1 บิตในระดับเกตได้ดังรูปที่ 3.11

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity HALFADDER is
6   Port ( A,B : in STD_LOGIC;
7         Sum,Cout : out STD_LOGIC);
8 end HALFADDER;
9
10 architecture Behavioral of HALFADDER is
11 begin
12   Sum <= A xor B;
13   Cout <= A and B;
14 end Behavioral;
```

รูปที่ 3.11 โค้ด VHDL ของวงจร Half adder

วงจร Full adder เป็นวงจรบวกขนาด 1 บิตที่ประกอบด้วย 3 อินพุตและ 2 เอาต์พุต คือ A, B, ตัวทดเข้า (Carry-in) หรือ Cin, ผลรวม (Sum) หรือ Sum และตัวทดออก (Carry-out) หรือ Cout มีตารางความจริงแสดงดังรูปที่ 3.12

Input			Output	
A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

รูปที่ 3.12 ตารางความจริงของวงจร Full adder

จากตารางความจริงของวงจร Full adder 1 บิตในรูปที่ 3.12 นั้นสามารถเขียนโค้ดได้หลายรูปแบบ ซึ่งจะขึ้นอยู่กับสีต่อลักษณะของแต่ละคน เช่น ตัวอย่างดังในรูปที่ 3.13 (ซึ่งเป็นการเขียนโค้ดจากตารางความจริง) และดังรูป 3.14 เป็นต้น

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity FULLADDER is
6     Port ( A,B,Cin : in STD_LOGIC;
7             Sum,Cout : out STD_LOGIC);
8 end FULLADDER;
9
10 architecture Behavioral of FULLADDER is
11     signal X : STD_LOGIC_VECTOR(2 downto 0);
12 begin
13     X <= A&B&Cin;
14
15     Sum <= '1' when (X="001" or X="010" or X="100" or X="111") else
16         '0';
17
18     Cout <= '1' when (X="011" or X="101" or X="110" or X="111") else
19         '0';
20 end Behavioral;
```

รูปที่ 3.13 โค้ดวงจร Full adder 1 บิตที่เขียนโดยใช้จากตารางความจริงคำสั่ง Conditional signal assignment 2 คำสั่ง

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity FULLADDER is
6     Port ( A,B,Cin : in STD_LOGIC;
7             Sum,Cout : out STD_LOGIC);
8 end FULLADDER;
9
10 architecture Behavioral of FULLADDER is
11 begin
12     Sum <= A xor B xor Cin;
13     Cout <= (A and B) or (A and Cin) or (B and Cin);
14 end Behavioral;
```

รูปที่ 3.14 โค้ดวงจร Full adder 1 บิตที่เขียนในระดับเกต (โดยใช้ความรู้เกี่ยวกับสมการบูลีนมาเขียนโค้ด)

การสร้างวงจรบวก-ลบโดยใช้ Arithmetic operators นี้จะขึ้นอยู่กับความสามารถของซอฟต์แวร์ทุกตัวที่ใช้ ตัวอย่างโค้ดของวงจรบวก-ลบทั้งหมดสามารถคุ้ดจากตัวอย่างที่ 2.9 และตัวอย่างที่ 2.20 ในบทที่ 2

การทดลองที่ 3.2.1 วงจร Half adder

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับวงจรบวกแบบ Half adder
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างวงจรบวกและโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

บอร์ด 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 ซึ่งสามารถเขียนโค้ด VHDL ได้ดังรูปที่ L1.1 โดยในการทดลองนี้จะใช้ Project Location (หรือ Folder) ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_2_1vcx1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_2_1vcx1 is
6     Port ( A,B : in STD_LOGIC;
7             Sum,Cout : out STD_LOGIC);
8 end ex3_2_1vcx1;
9
10 architecture Behavioral of ex3_2_1vcx1 is
11 begin
12     Sum <= A xor B;
13     Cout <= A and B;
14 end Behavioral;

```

รูปที่ L1.1 โค้ด VHDL ของวงจร Half adder ที่เขียนในระดับแกต

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB2 เป็นอินพุตและ LED1-LED2 เป็นเอาต์พุต กล่าวคือ

A = PB1 = INPUT = p39

Sum = LED2 = OUTPUT = p37

B = PB2 = INPUT = p40

Cout = LED1 = OUTPUT = p38

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
A	Input	p39	2	9		
B	Input	p40	2	11		
Sum	Output	p37	2	6	SLOW	
Cout	Output	p38	2	8	SLOW	

รูปที่ L1.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิป CPLD แล้วให้ทดลองกดปุ่ม PB1-PB2 และให้สังเกตผลที่ LED1-LED2 ว่าให้ล้อจิกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

2 สร้างวงจร Half adder ด้วย FPGA

ให้ออกแบบวงจร Half adder ด้วย FPGA ซึ่งสามารถเขียนโค้ด VHDL ได้ดังรูปที่ L2.1 โดยในการทดลองนี้จะใช้ Project Location (หรือ Folder) ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_2_1vf

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_2_1vf is
6     Port ( A,B : in STD_LOGIC;
7             Sum,Cout : out STD_LOGIC);
8 end ex3_2_1vf;
9
10 architecture Behavioral of ex3_2_1vf is
11 begin
12     Sum <= A xor B;
13     Cout <= A and B;
14 end Behavioral;

```

รูปที่ L2.1 โค้ด VHDL ที่เขียนในระดับเกต

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB2 เป็นอินพุตและ LED L2-L3 เป็นเอาต์พุต กล่าวคือ

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

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

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

$$\text{Cout} = L3 = \text{OUTPUT} = p76$$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
A	Input	p44	BANK5	LVCMS33	N/A	3.30					Unknown		
B	Input	p46	BANK5	LVCMS33	N/A	3.30					Unknown		
Cout	Output	p76	BANK3	LVCMS33	N/A	3.30			SLOW		Unknown		
Sum	Output	p69	BANK4	LVCMS33	N/A	3.30			SLOW		Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลง FPGA และให้ทดลองกดปุ่ม PB1-PB2 และให้สังเกตผลที่ LED L2-L3 ว่าให้ล็อกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

การทดลองที่ 3.2.2 วงจร Full adder

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับวงจรบวกแบบ Full adder ขนาด 1 บิต
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างวงจรบวกและโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจร Full adder ขนาด 1 บิต ด้วย CPLD

ให้ออกแบบวงจร Full adder ขนาด 1 บิตด้วย CPLD โดยนำข้อมูลจากตารางความจริงในรูปที่ 3.12 มาเขียนเป็นโค้ด VHDL ได้ดังรูปที่ L1.1 ซึ่งเราจะกำหนดให้ $X(2) = A$, $X(1) = B$, $X(0) = Cin$ และ $Y(1) = Cout$, $Y(0) = Sum$ โดยการทดลองนี้จะใช้ Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_2_2vcx1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_2_2vcx1 is
6     Port ( X : in STD_LOGIC_VECTOR (2 downto 0);
7             Y : out STD_LOGIC_VECTOR (1 downto 0));
8 end ex3_2_2vcx1;
9 architecture Behavioral of ex3_2_2vcx1 is
10 begin
11     Y <= "00" when X="000" else
12         "01" when X="001" else
13         "01" when X="010" else
14         "10" when X="011" else
15         "01" when X="100" else
16         "10" when X="101" else
17         "10" when X="110" else
18         "11" when X="111" else
19         "00";
20 end Behavioral;

```

รูปที่ L1.1 โค้ดวงจร Full adder 1 บิตที่เขียนโดยใช้จากตารางความจริงด้วยคำสั่ง Conditional signal assignment 1 คำสั่ง

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1–PB3 เป็นอินพุตและ LED3–LED4 เป็นเอาท์พุต กล่าวคือ

$$X(2) = A = PB1 = \text{INPUT} = p39$$

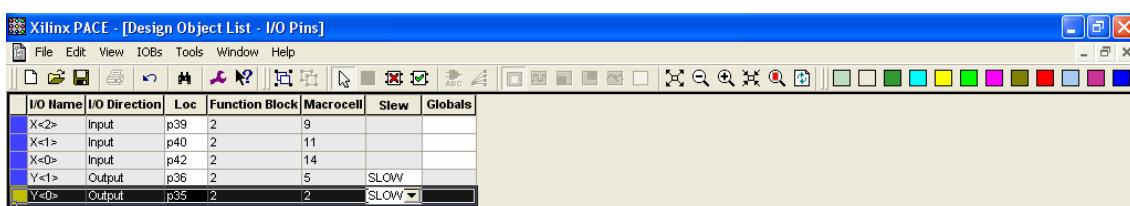
$$Y(0) = \text{Sum} = \text{LED4} = \text{OUTPUT} = p35$$

$$X(1) = B = PB2 = \text{INPUT} = p40$$

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

$$X(0) = \text{Cin} = PB3 = \text{INPUT} = p42$$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้คือ



รูปที่ L1.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบบนชิป CPLD แล้วให้ทดลองกดปุ่ม PB1–PB3 และให้สังเกตคุณลักษณะที่ LED3–LED4 ว่าให้ล้อจิกເອົາຕີພຸດເປັນໄປຕາມທາງໝູ້ຫຼືໄວ່ຈາກນັ້ນຈຶ່ງບັນທຶກຜລກາຮັດລອງ

2 สร้างวงจร Full adder ขนาด 1 บิตด้วย FPGA

ให้ออกแบบวงจร Full adder ขนาด 1 บิตด้วย FPGA โดยนำข้อมูลจากตารางความจริงในรูปที่ 3.12 มาเขียนเป็นโค้ด VHDL ได้ดังรูปที่ L2.1 ซึ่งเราจะกำหนดให้ $X(2) = A$, $X(1) = B$, $X(0) = Cin$ และ $Y(1) = Cout$, $Y(0) = Sum$ โดยการทดลองนี้จะใช้ Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_2_2vf

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_2_2vf is
6     Port ( X : in STD_LOGIC_VECTOR (2 downto 0);
7             Y : out STD_LOGIC_VECTOR (1 downto 0));
8 end ex3_2_2vf;
9 architecture Behavioral of ex3_2_2vf is
10 begin
11     Y <= "00" when X="000" else
12         "01" when X="001" else
13         "01" when X="010" else
14         "10" when X="011" else
15         "01" when X="100" else
16         "10" when X="101" else
17         "10" when X="110" else
18         "11" when X="111" else
19         "00";
20 end Behavioral;

```

รูปที่ L2.1 ໂຄ້ດວງຈະ Full adder 1 บิตທີ່ເຂົ້າໃຫ້ຈາກตารางความจริงດ້ວຍຄໍາສັ່ງ Conditional signal assignment 1 ຄໍາສັ່ງ

การกำหนดขาສัญญาณต่างๆจะใช้ปุ่มกด PB1 ถึง PB3 เป็นอินพຸດและ LED L0-L1 เป็นເອົາຕີພຸດ ກລ່ວກື່ອ

$$X(2) = A = PB1 = \text{INPUT} = p44 \quad Y(0) = \text{Sum} = L0 = \text{OUTPUT} = p70$$

$$X(1) = B = PB2 = \text{INPUT} = p46 \quad Y(1) = \text{Cout} = L1 = \text{OUTPUT} = p77$$

$$X(0) = \text{Cin} = PB3 = \text{INPUT} = p47$$

โดยพิมพ์ໃນ Assign Package Pins ສຽງປັດງນີ້

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
X<0>	Input	p44	BANK5	LVCMS33	N/A	3.30					Unknown		
X<1>	Input	p46	BANK5	LVCMS33	N/A	3.30					Unknown		
X<2>	Input	p47	BANK5	LVCMS33	N/A	3.30					Unknown		
Y<0>	Output	p70	BANK4	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<1>	Output	p77	BANK3	LVCMS33	N/A	3.30			SLOW		Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบบนชิป FPGA แล้วให้ทดลองกดปุ่ม PB1–PB3 และให้สังเกตคุณลักษณะที่ LED L0-L1 ว่าให้ล้อຈิกເອົາຕີພຸດເປັນໄປຕາມທາງໝູ້ຫຼືໄວ່ຈາກນັ້ນຈຶ່ງບັນທຶກຜລກາຮັດລອງ

แบบฝึกหัด

- ให้เขียนໂຄ້ດວງຈະ Full adder ขนาด 1 บิตทີ່ 3 วິທີໂຄ້ດວງຈະ 1 บิตด้วย VHDL โดยใช้คำสັ່ງ if คำสັ່ງ case และคำสັ່ງ Selected signal assignment

การทดลองที่ 3.2.3 วงจรบวก-ลบขนาดหลายบิตโดยใช้ Arithmetic operators

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับโค้ดวงจรบวก-ลบขนาดหลายบิตโดยใช้ Arithmetic operators
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างวงจรบวก-ลบขนาดหลายบิตและโปรแกรมลงชิป CPLD และ FPGA
- 3) ฝึกการเรียกใช้ Package ชื่อ std_logic_unsigned ของ Synopsys และ Package ชื่อ Numeric_std ของ IEEE

อุปกรณ์ทดลอง

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

1 สร้างวงจรบวก-ลบขนาด 2 บิตด้วย CPLD

ให้ออกแบบวงจรบวก-ลบขนาด 2 บิตด้วย CPLD โดยเขียนโค้ด VHDL และใช้ Arithmetic operators ดังรูปที่ L1.1 ตามขั้นตอนที่ได้เรียนมาแล้วในบทที่ 2 โดยใช้ Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_2_3vcx1 จากนั้นในขั้นตอนตรวจสอบความถูกต้องของโค้ดหรือ Check syntax หลังจากที่ Check syntax ผ่านเรียบร้อยแล้วให้ทำการทดลองใส่ค่าคอมเมนต์ “-” หรือเครื่องหมายลบติดกัน 2 ตัว แล้วทำการ Check syntax ใหม่อีกรอบเพื่อตรวจสอบว่า คอมไพล์รู้จักเครื่องหมาย “+” และ “-” หรือไม่ หากนั้นแก้โค้ดกลับคืนตามเดิมแล้ว Check syntax ใหม่อีกรอบ

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex3_2_3vcx1 is
7     generic(N : integer := 2);
8     Port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
9             OPER : in STD_LOGIC;
10            C : out STD_LOGIC_VECTOR (N downto 0));
11 end ex3_2_3vcx1;
12
13 architecture Behavioral of ex3_2_3vcx1 is
14 begin
15     C <= ('0' &A) + ('0' &B) when OPER = '0' else
16           ('0' &A) - ('0' &B);
17 end Behavioral;

```

รูปที่ L1.1 โค้ด VHDL ของวงจรบวก-ลบขนาด 2 บิตโดยใช้คำสั่ง Conditional signal assignment

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB2-PB6 (Slide SW1- Slide SW4) เป็นอินพุต และ LED2-LED4 เป็นเออเตอร์พุต กล่าวคือ

A(1) = PB3 (Slide SW1) = INPUT = p42	
A(0) = PB4 (Slide SW2) = INPUT = p43	C(2) = LED2 = OUTPUT = p37
B(1) = PB5 (Slide SW3) = INPUT = p44	C(1) = LED3 = OUTPUT = p36
B(0) = PB6 (Slide SW4) = INPUT = p1	C(0) = LED4 = OUTPUT = p35
OPER = PB2 = INPUT = p40	

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
A<1>	Input	p42	2	14		
A<0>	Input	p43	2	15		
B<1>	Input	p44	2	17		
B<0>	Input	p1	1	2		
OPER	Input	p40	2	11		
C<2>	Output	p37	2	6	SLOW	
C<1>	Output	p36	2	5	SLOW	
C<0>	Output	p35	2	2	SLOW	

รูปที่ L1.2 Assign Package Pins

หลังจากโปรแกรมจะรีท่อออกแบบลงชิป CPLD แล้วให้ทดลองกดปุ่ม PB3–PB6 (ON หรือ OFF Slide SW1- Slide SW4) และให้สังเกตดูผลที่ LED2–LED4 ว่าให้ล็อกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ แล้วจึงทำการบันทึกผลการทดลอง

จากนี้ทำการแก้ไขโค้ดรูปที่ L1.1 โดยเปลี่ยนไปเรียกใช้ Numeric_std package แล้วทำการทดลองซ้ำ เสร็จแล้วให้เปลี่ยนมาใช้คำสั่ง Selected signal assignment คำสั่ง if และคำสั่ง case แล้วทำการทดลองซ้ำ

2 สร้างวงจรบวก-ลบขนาด 4 บิตด้วย FPGA

ให้ออกแบบวงจรบวก-ลบขนาด 4 บิตด้วย FPGA โดยเขียนโค้ด VHDL และใช้ Arithmetic operators ดังรูปที่ L2.1 โดยใช้ Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_2_3vf

จากนี้ในขั้นตอนตรวจสอบความถูกต้องของโค้ดหรือ Check syntax หลังจาก Check syntax ผ่านเรียบร้อยแล้วให้ทำการทดลองใส่ค่าคอมแมนต์ “- -” หรือเครื่องหมายบวกติดกัน 2 ตัว แล้วจึงทำการ Check syntax ใหม่อีกรอบเพื่อตรวจสอบว่า คอมไพล์อร์รู้จักเครื่องหมาย “+” และ “-” หรือไม่ จากนั้นแก้โค้ดกลับคืนตามเดิมแล้ว Check syntax ใหม่อีกรอบ

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex3_2_3vf is
7     generic(N : integer := 4);
8     Port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
9             OPER : in STD_LOGIC;
10            C : out STD_LOGIC_VECTOR (N downto 0));
11 end ex3_2_3vf;
12
13 architecture Behavioral of ex3_2_3vf is
14 begin
15     C <= ('0'&A) + ('0'&B) when OPER = '0' else
16     ('0'&A) - ('0'&B);
17 end Behavioral;

```

รูปที่ L2.1 โค้ด VHDL ของวงจรบวก-ลบขนาด 4 บิตโดยใช้คำสั่ง Conditional signal assignment

การกำหนดขาสัญญาณต่างๆ จะใช้ PB1 และ Dip SW1-Dip SW8 เป็นอินพุตและ LED L0–L4 เป็นเอาต์พุต กล่าวคือ

A(3) = Dip SW1 = INPUT = p52	B(3) = Dip SW5 = INPUT = p59	C(4) = L4 = OUTPUT = p74
A(2) = Dip SW2 = INPUT = p53	B(2) = Dip SW6 = INPUT = p60	C(3) = L3 = OUTPUT = p76
A(1) = Dip SW3 = INPUT = p55	B(1) = Dip SW7 = INPUT = p63	C(2) = L2 = OUTPUT = p69
A(0) = Dip SW4 = INPUT = p56	B(0) = Dip SW8 = INPUT = p68	C(1) = L1 = OUTPUT = p77
OPER = PB1 = INPUT = p44		C(0) = L0 = OUTPUT = p70

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
A<0>	Input	p56	BANK LVCMOS33	N/A	3.30						Unknown		
A<1>	Input	p55	BANK LVCMOS33	N/A	3.30						Unknown		
A<2>	Input	p53	BANK LVCMOS33	N/A	3.30						Unknown		
A<3>	Input	p52	BANK LVCMOS33	N/A	3.30						Unknown		
B<0>	Input	p68	BANK LVCMOS33	N/A	3.30						Unknown		
B<1>	Input	p63	BANK LVCMOS33	N/A	3.30						Unknown		
B<2>	Input	p60	BANK LVCMOS33	N/A	3.30						Unknown		
B<3>	Input	p59	BANK LVCMOS33	N/A	3.30						Unknown		
C<0>	Output	p70	BANK LVCMOS33	N/A	3.30			SLOW			Unknown		
C<1>	Output	p77	BANK LVCMOS33	N/A	3.30			SLOW			Unknown		
C<2>	Output	p69	BANK LVCMOS33	N/A	3.30			SLOW			Unknown		
C<3>	Output	p76	BANK LVCMOS33	N/A	3.30			SLOW			Unknown		
C<4>	Output	p74	BANK LVCMOS33	N/A	3.30			SLOW			Unknown		
OPER	Input	p44	BANK LVCMOS33	N/A	3.30						Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมจะรู้ที่ออกแบบลงชิป FPGA แล้วให้ทดลองกดปุ่ม PB1 และ ON-OFF Dip SW1-Dip SW8 แล้วให้สังเกตคุณผลที่ LED L0-L4 ว่าให้ล้อจิกເອົາຕຸພູດເປັນໄປຕາມທາງໝູ້ຮູ້ໄນ້ แล้วຈຶ່ງทำการບັນທຶກผลการทดลอง

ทำการแก้ไขโค้ดรูปที่ L2.1 โดยเปลี่ยนໄປເຮີຍໃຫ້ Package ชื่อ Numeric_std แล้วทำการทดลองซໍາ เสວົງແລ້ວໃຫ້เปลี่ยนมาໃຊ້คำสั่ง Selected signal assignment คำสั่ง if และคำสั่ง case ແລ້ວทำการทดลองซໍາ

หมายเหตุ Xilinx Synthesis Tool หรือ XST จะรองรับการใช้งาน (Supported) Operator ดังในรูปที่ L2.4 เท่านั้น

Operators	Logical Operators: and, or, nand, nor, xor, xnor, not	Supported
	Relational Operators: =, /=, <, <=, >, >=	Supported
	& (concatenation)	Supported
	Adding Operators: +, -	Supported
	*	Supported
	/, rem	Supported if the right operand is a constant power of 2
	mod	Supported
	Shift Operators: sll, srl, sla, sra, rol, ror	Supported
	abs	Supported
	**	Only supported if the left operand is 2
	Sign: +, -	Supported

รูปที่ L2.4 Operators ที่ XST จะรองรับการใช้งาน

3.3 วงจรจอดรหัส

วงจรจอดรหัสเป็นวงจรคอมบินेशันชั้นกัน ตัวอย่างวงจรจอดรหัส เช่น วงจร 2 to 4 Decoder และวงจรจอดรหัส BCD เป็นเซเว่นเซกเมนต์ (BCD to 7 Segment decoder) เป็นต้น

วงจรจอดรหัสเข้า 2 ออก 4 (2 to 4 Decoder) นี้ได้อธิบายไปบ้างแล้วในตัวอย่างที่ 2.1 ตัวอย่างที่ 2.10 และตัวอย่างที่ 2.21 โดยในรูปที่ 3.15 และรูปที่ 3.16 เป็นตารางความจริงและโค้ดของวงจร 2 to 4 Decoder ที่ให้อาต์พุตเป็นลอจิก ‘1’ (One hot decoder) ส่วนในรูปที่ 3.17 และรูปที่ 3.18 จะให้อาต์พุตเป็นลอจิก ‘0’ (One cold decoder) โดยที่ “-” คือ Don’t care โค้ดในรูปที่ 3.16 และรูปที่ 3.18 จะเป็นการเขียนโดยใช้คำสั่ง Conditional signal assignment ซึ่งในทางปฏิบัติเราสามารถเขียนโดยใช้คำสั่ง selected signal assignment คำสั่ง if และคำสั่ง case ได้เช่นกัน

Input A			Output Y			
E	A(1)	A(0)	Y(3)	Y(2)	Y(1)	Y(0)
0	-	-	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 3.15 ตารางความจริงของวงจร 2 to 4 Decoder ที่ให้อาต์พุตเป็นลอจิก ‘1’ (One hot)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ONE_HOT2to4 is
6   Port ( E : in STD_LOGIC;
7          A : in STD_LOGIC_VECTOR (1 downto 0);
8          Y : out STD_LOGIC_VECTOR (3 downto 0));
9 end ONE_HOT2to4;
10
11 architecture Behavioral of ONE_HOT2to4 is
12   signal X : STD_LOGIC_VECTOR (2 downto 0);
13 begin
14   X <= E&A;
15   Y <= "0001" when X="100" else
16     "0010" when X="101" else
17     "0100" when X="110" else
18     "1000" when X="111" else
19     "0000";
20 end Behavioral;
```

รูปที่ 3.16 โค้ดของวงจร 2 to 4 Decoder ที่ให้อาต์พุตเป็นลอจิก ‘1’ (One hot)

Input A			Output Y			
E	A(1)	A(0)	Y(3)	Y(2)	Y(1)	Y(0)
0	-	-	1	1	1	1
1	0	0	1	1	1	0
1	0	1	1	1	0	1
1	1	0	1	0	1	1
1	1	1	0	1	1	1

รูปที่ 3.17 ตารางความจริงของวงจร 2 to 4 Decoder ที่ให้อาต์พุตเป็นลอจิก ‘0’ (One cold)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ONE_COLD2to4 is
6     Port ( E : in STD_LOGIC;
7             A : in STD_LOGIC_VECTOR (1 downto 0);
8             Y : out STD_LOGIC_VECTOR (3 downto 0));
9 end ONE_COLD2to4;
10
11 architecture Behavioral of ONE_COLD2to4 is
12     signal X : STD_LOGIC_VECTOR (2 downto 0);
13 begin
14     X <= E&A;
15     Y <= "1110" when X="100" else
16         "1101" when X="101" else
17         "1011" when X="110" else
18         "0111" when X="111" else
19         "1111";
20 end Behavioral;

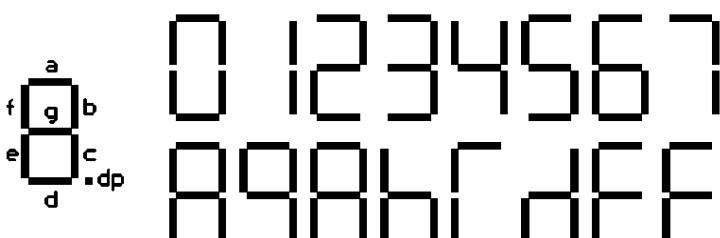
```

รูปที่ 3.18 โค้ดของวงจร 2 to 4 Decoder ที่ให้ออกพุ่มโลจิก ‘0’ (One cold)

วงจรดอร์หัสอีกประเภทหนึ่งที่นิยมใช้กันมาก คือ วงจรดอร์หัสของตัวแสดงผลเซเว่นเซกเมนต์ (7-Segment LED display) ได้แก่ วงจรดอร์หัสตัวแสดงผลเซเว่นเซกเมนต์ (7-Segment decoder) ในรูปแบบเลขฐานสิบ (Decimal digit display format) และเลขฐานสิบหก (Hexadecimal digit display format) แสดงดังรูปที่ 3.19a) และรูปที่ 3.19b) ตามลำดับ ซึ่งตัวแสดงผลเซเว่นเซกเมนต์ประกอบด้วย LED 7 ส่วน (Segment) คือ a, b, c, d, e, f, g และจุด (dp = Decimal point) หลักในการจดจำตำแหน่ง Segment บนแสดงผลนั้นให้เริ่มนับจาก Segment a อยู่ที่ตำแหน่ง 12 นาฬิกา จากนั้นนับวนตามเข็มนาฬิกาจะเป็น Segment b, c, d, e และ f ตามลำดับและมี Segment g อยู่ตรงกลาง ความแตกต่างระหว่างรูปแบบเลขฐานสิบและฐานสิบหกจะอยู่ที่เลข 6 และเลข 9 ที่มีเซกเมนต์ (Segment) a และ d เพิ่มเข้ามาเพื่อทำให้เลข 6 ไม่ไปซ้ำกับตัว b และรูปแบบเลขฐานสิบหกจะมี A, b, C, d, E, F และ g เพิ่มเข้ามา ตัวแสดงผลมี 2 แบบ คือ แອโนคร่าวม (Common Anode) และแคนโอดร่าวม (Common cathode) โดยที่แบบแคนโอดร่าวมนี้ LED แต่ละส่วนจะติดสว่างได้ก็ต่อเมื่อป้อนโลจิก ‘1’ (โดยต่อผ่านตัวต้านทานเพื่อจำกัดกระแส) และป้อนโลจิก ‘0’ หรือต่อลงกราวด์ที่ขาแອโนคร่าวม แต่ถ้าตัวแสดงผลแบบแອโนคร่าวมก็จะทำงานตรงกันข้าม



3.19a) รูปแบบเลขฐานสิบ



3.19b) รูปแบบเลขฐานสิบหก

รูปที่ 3.19 รูปแบบการแสดงผลบนตัวแสดงผลเซเว่นเซกเมนต์

ตารางความจริงของวงจรจดจำรหัสตัวแสดงผลเซเว่นเซกเมนต์แบบเลขฐานสิบและเลขฐานสิบหกแบบแคนโอดร่วมนั้นแสดงดังรูปที่ 3.20 และรูปที่ 3.21 ตามลำดับ

No.	Input A				Output Y							
	A(3)	A(2)	A(1)	A(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	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 3.20 ตารางความจริงของตัวจดจำรหัสเป็น Decimal digit display format แบบแคนโอดร่วม

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

รูปที่ 3.21 ตารางความจริงของตัวจดจำรหัสเป็น Hexadecimal digit display format แบบแคนโอดร่วม

เพื่อให้ตัวเลขทุกตัวที่แสดงผลมีรูปแบบเดียวกัน ดังนั้นวงจรจดจำรหัสตัวแสดงผลเซเว่นเซกเมนต์ที่ใช้ในหนังสือเล่มนี้จะใช้รูปแบบเลขฐานสิบหกดังมีรายละเอียดรูปที่ 3.19b ทั้งหมดแม้ว่าบางวงจรจะใช้จดจำรหัสเลข 0-9 ก็ตาม การออกแบบวงจรจดจำรหัส BCD เป็นเซกเมนต์ (BCD to 7-Segment Decoder) นั้นสามารถทำได้โดยนำตารางความจริงรูปที่ 3.21 เคฟาเล็ก 0-9 ไปเขียนໂคด์

ໂคด์ของวงจรจดจำรหัสตัวแสดงผลเซกเมนต์ (7-Segment decoder) แบบแคนโอดร่วมมีรายละเอียดในบทที่ 2 ในตัวอย่างที่ 2.3 ตัวอย่างที่ 2.11 ตัวอย่างที่ 2.12 ตัวอย่างที่ 2.22 และตัวอย่างที่ 2.23 ตามลำดับ

การทดลองที่ 3.3.1 วงจร 2 to 4 Decoder

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานของวงจร decoder 2 to 4 Decoder
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างวงจร decoder 2 to 4 และโปรแกรมลงชิป CPLD หรือ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจร 2 to 4 Decoder ด้วย CPLD

ให้ออกแบบวงจร 2 to 4 Decoder แบบ One hot ด้วย CPLD ดังตารางความจริงในรูปที่ 3.15 โดยเขียนโค้ดดังรูปที่ L1.1 โดยใช้คำสั่ง if การทดลองนี้จะใช้ Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_3_1vcx1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.all;
4
5 entity ex3_3_1vcx1 is
6     Port ( E : in STD_LOGIC;
7             A : in STD_LOGIC_VECTOR (1 downto 0);
8             D : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex3_3_1vcx1;
10
11 architecture Behavioral of ex3_3_1vcx1 is
12 begin
13 process(A,E)
14 begin
15     if E='0' then D <= "0000";
16     else if A="00" then D <= "0001";
17         elsif A="01" then D <= "0010";
18         elsif A="10" then D <= "0100";
19         elsif A="11" then D <= "1000";
20         else D <= "0000";
21     end if;
22     end if;
23 end process;
24 end Behavioral;

```

รูปที่ L1.1 โค้ดของวงจร 2 to 4 Decoder แบบ One hot โดยใช้คำสั่ง if

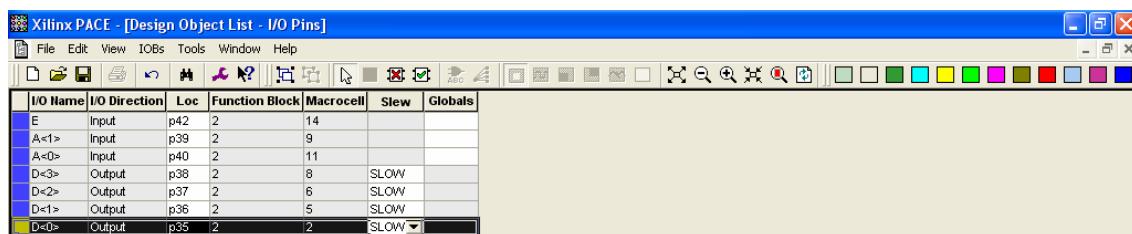
จากนั้นกำหนดขาตัวอย่างต่างๆ จะใช้ปุ่มกด PB1-PB3 เป็นอินพุตและ LED1-LED4 เป็นเอาต์พุต ก่อร่างกาย

$$A(1) = PB1 = \text{INPUT} = p39 \quad D(0) = \text{LED1} = \text{OUTPUT} = p35 \quad D(2) = \text{LED3} = \text{OUTPUT} = p37$$

$$A(0) = PB2 = \text{INPUT} = p40 \quad D(1) = \text{LED2} = \text{OUTPUT} = p36 \quad D(3) = \text{LED4} = \text{OUTPUT} = p38$$

$$E = PB3 (\text{Slide SW1}) = \text{INPUT} = p42$$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



รูปที่ L1.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิป CPLD แล้วให้ OFF Slide SW1 แล้วทดลองกดปุ่ม PB1–PB2 และให้สังเกตที่ LED1–LED4 ว่าให้กลิ่กเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ หากนั้น ON Slide SW1 แล้วทดลองซ้ำ แล้วบันทึกผลการทดลอง

เมื่อทดลองเสร็จแล้วให้แก้ไขโค้ดเป็นง่าย 2 to 4 Decoder แบบ One cold แล้วทำการทดลองซ้ำ

2 สร้างวงจร 2 to 4 Decoder ด้วย FPGA

ให้ออกแบบวงจร 2 to 4 Decoder แบบ One hot ด้วย FPGA ดังตารางความจริงในรูปที่ 3.15 โดยเขียนโค้ดดังรูปที่ L2.1 โดยใช้คำสั่ง if การทดลองนี้จะใช้ Project Location ชื่อ ch3v แล้วกำหนด Project Name และ Source File ชื่อ ex3_3_1vf

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.all;
4
5 entity ex3_3_1vf is
6     Port ( E : in STD_LOGIC;
7             A : in STD_LOGIC_VECTOR (1 downto 0);
8             D : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex3_3_1vf;
10
11 architecture Behavioral of ex3_3_1vf is
12 begin
13 process(A,E)
14 begin
15     if E='0' then D <= "0000";
16     else if A="00" then D <= "0001";
17         elsif A="01" then D <= "0010";
18         elsif A="10" then D <= "0100";
19         elsif A="11" then D <= "1000";
20         else D <= "0000";
21         end if;
22     end if;
23 end process;
24 end Behavioral;
```

รูปที่ L2.1 โค้ดของวงจร 2 to 4 Decoder แบบ One hot โดยใช้คำสั่ง if

หากนั้นกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1–PB3 เป็นอินพุตและ LED L0–L3 เป็นเอาต์พุต กล่าวคือ

$$A(1) = PB1 = \text{INPUT} = p44 \quad D(0) = L0 = \text{OUTPUT} = p70 \quad D(2) = L2 = \text{OUTPUT} = p69$$

$$A(0) = PB2 = \text{INPUT} = p46 \quad D(1) = L1 = \text{OUTPUT} = p77 \quad D(3) = L3 = \text{OUTPUT} = p76$$

$$E = PB3 = \text{INPUT} = p47$$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
A<0>	Input	p46	BANK LVC MOS33	N/A	3.30						Unknown		
A<1>	Input	p44	BANK LVC MOS33	N/A	3.30						Unknown		
D<0>	Output	p70	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
D<1>	Output	p77	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
D<2>	Output	p69	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
D<3>	Output	p76	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
E	Input	p47	BANK LVC MOS33	N/A	3.30						Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิป FPGA แล้วให้ทดลองกดปุ่ม PB1–PB2 และให้สังเกตที่ LED L0–L3 ว่าให้ลองกลิ่กเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ หากนั้นให้กดปุ่ม PB3 ถ้างานได้แล้วทำการทดลองซ้ำ แล้วบันทึกผลการทดลอง

เมื่อทดลองเสร็จแล้วให้แก้ไขโค้ดเป็นง่าย 2 to 4 Decoder แบบ One cold แล้วทำการทดลองซ้ำ

แบบฝึกหัด ให้เปลี่ยนโค้ด 2 to 4 Decoder โดยใช้คำสั่ง selected signal assignment, Conditional signal assignment และ case

กາຮັດລອງທີ່ 3.3.2 ວົງຈະຈອດຮັສຕັວແສດງພລເຊເວນເຊກເມນດໍ

ວັດຖຸປະສົງຄໍ

- 1) ເພື່ອທຳການເຂົ້າໃຈເກີ່ມກັບງານຈະຈອດຮັສຕັວແສດງພລເຊເວນເຊກເມນດໍ
- 2) ເພື່ອທຳລອງໃໝ່ ISE WebPACK 8.1i ສ້າງວົງຈະຈອດຮັສແລະ ໂປຣແກຣມລົງຈິພ CPLD ແລະ FPGA

ອຸປະກອນທຳລອງ

ບອຮົດ CPLD Explorer XC9572XL ມີ CPLD Explorer XC9572 ແລະ FPGA Discovery-III XC3S200F ມີ FPGA Discovery-III XC3S200F4

1 ສ້າງວົງຈະຈອດຮັສຕັວແສດງພລເຊເວນເຊກເມນດໍຕ້ວຍ CPLD

ໄດ້ດວງຈະຈອດຮັສຕັວແສດງພລເຊເວນເຊກເມນດໍແບບເລຂຽນສົບທັບແຄໂຄດ່ວມແລະ ວົງຈະຈອດຮັສຕັວແສດງພລເຊເວນເຊກເມນດໍຕ້ວຍ CPLD

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_3_2vcx1 is
6     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
7             COMMON : out STD_LOGIC;
8             Y : out STD_LOGIC_VECTOR (6 downto 0));
9 end ex3_3_2vcx1;
10
11 architecture Behavioral of ex3_3_2vcx1 is
12 begin
13     -----HEX to 7-segments decoder-----
14     --gfedcba
15     Y <= "1110001" when A="1111" else --F
16     "111001" when A="1110" else --E      Y(0)=a
17     "101110" when A="1101" else --d      ---
18     "0111001" when A="1100" else --C      Y(5)=f|  | Y(1)=b
19     "1111100" when A="1011" else --b      --- Y(6)=g
20     "1101011" when A="1010" else --A      Y(4)=e|  | Y(2)=c
21     "1101111" when A="1001" else --9      ---
22     "1111111" when A="1000" else --8      Y(3)=d
23     "0000111" when A="0111" else --7
24     "1111101" when A="0110" else --6
25     "1101101" when A="0101" else --5
26     "1100110" when A="0100" else --4
27     "1001111" when A="0011" else --3
28     "1011011" when A="0010" else --2
29     "0000110" when A="0001" else --1
30     "0111111";                      --0
31     -----Common cathode-----
32     COMMON <= '0';
33 end Behavioral;
```

ຮູບທີ່ L1.1 ວົງຈະຈອດສອບວົງຈະຈອດຮັສຕັວແສດງພລເຊເວນເຊກເມນດໍແບບເລຂຽນສົບທັບແຄໂຄດ່ວມ

ການກຳໜາຂາສັນຍາມຕ່າງໆ ຈະໃໝ່ປຸ່ມກົດ PB3-PB6 (Slide SW1- Slide SW4) ເປັນອິນພູດ ມີຕັວແສດງພລເຊເວນເຊກເມນດໍ ແບບແຄໂຄດ່ວມໜັກທີ່ 1 ຄື່ອ DIGIT1 ເປັນເອົາຕົກ ກລ່າວກື່ອ

A(3) = PB3 (Slide SW1) = INPUT = p42 Y(0) = a = OUTPUT = p27 Y(5) = f = OUTPUT = p20

A(2) = PB4(Slide SW2) = INPUT = p43 Y(1) = b = OUTPUT = p26 Y(6) = g = OUTPUT = p18

A(1) = PB5(Slide SW3) = INPUT = p44 Y(2) = c = OUTPUT = p25

$A(0) = PB6(\text{Slide SW4}) = \text{INPUT} = p1 \quad Y(3) = d = \text{OUTPUT} = p24$

$\text{COMMON} = \text{DIGIT1} = \text{OUTPUT} = p34 \quad Y(4) = e = \text{OUTPUT} = p22$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
A<3>	Input	p42	2	14		
A<2>	Input	p43	2	15		
A<1>	Input	p44	2	17		
A<0>	Input	p1	1	2		
COMMON	Output	p34	4	17	SLOW	
Y<6>	Output	p18	3	11	SLOW	
Y<5>	Output	p20	3	15	SLOW	
Y<4>	Output	p22	3	17	SLOW	
Y<3>	Output	p24	3	16	SLOW	
Y<2>	Output	p25	4	2	SLOW	
Y<1>	Output	p26	4	5	SLOW	
Y<0>	Output	p27	4	8	SLOW	

รูปที่ L1.2 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป CPLD แล้วให้ทดสอบ ON-OFF Slide SW1-Slide SW4 โดยเริ่มจาก ON Slide SW1-Slide SW4 ทุกตัว (คือ $A = "0000"$) ทีละค่าจนกระทั่ง OFF Slide SW1-Slide SW4 ทุกตัว (คือ $A = "1111"$) พร้อมกับให้สังเกตที่ตัวแสดงผลเซเว่นเซกเมนต์ว่ามีส่วนใดติดสว่าง ให้ออกจิตເອຕ์พุตเป็นไปตามกำหนดหรือไม่ แล้วบันทึกผลการทดสอบ

จากนั้นทำการแก้ไขโค้ดเป็นวงจรดอร์หัสแบบเลขฐานสิบแบบแคลคูลัสร่วม แล้วทำการทดสอบซ้ำ

2 สร้างวงจรดอร์หัสตัวแสดงผลเซเว่นเซกเมนต์ด้วย FPGA

ให้ดาวน์โหลดร์หัสตัวแสดงผลเซเว่นเซกเมนต์แบบเลขฐานสิบแบบแคลคูลัสร่วมและวงจร Common cathode และดังรูปที่ L2.1 โดยจะเขียนไว้ใน Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_3_2vf

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.all;
4
5 entity ex3_3_2vf is
6   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
7         COMMON : out STD_LOGIC;
8         Y : out STD_LOGIC_VECTOR (6 downto 0));
9 end ex3_3_2vf;
10
11 architecture Behavioral of ex3_3_2vf is
12 begin
13   -----HEX to 7-segments decoder-----
14   --gfedcba
15   Y <= "1110001" when A="1111" else --F
16   "1110011" when A="1110" else --E    Y(0)=a
17   "1011110" when A="1101" else --d    ---
18   "0111001" when A="1100" else --C    Y(5)=f | Y(1)=b
19   "1111100" when A="1011" else --b    --- Y(6)=g
20   "1110111" when A="1010" else --A    Y(4)=e | Y(2)=c
21   "1101111" when A="1001" else --9    ---
22   "1111111" when A="1000" else --8    Y(3)=d
23   "0000111" when A="0111" else --7
24   "1111101" when A="0110" else --6
25   "1101101" when A="0101" else --5
26   "1100110" when A="0100" else --4
27   "1001111" when A="0011" else --3
28   "1011011" when A="0010" else --2
29   "0000110" when A="0001" else --1
30   "0111111";                      --0
31   -----Common cathode-----
32   COMMON <= '0';
33 end Behavioral;

```

รูปที่ L2.1 วงจรทดสอบวงจรดอร์หัสตัวแสดงผลเซเว่นเซกเมนต์แบบเลขฐานสิบแบบแคลคูลัสร่วม

การกำหนดขาสัญญาณต่างๆ จะใช้ Dip SW1-Dip SW4 เป็นอินพุต มีตัวแสดงผลเซกเมนต์แบบแอลจาร์มหลักที่ 1 คือ DIGIT1 เป็นเอาต์พุต กล่าวคือ

A(3)= Dip SW1 = INPUT = p52	Y(0) = a = OUTPUT = p40	Y(5) = f = OUTPUT = p25
A(2)= Dip SW2 = INPUT = p53	Y(1) = b = OUTPUT = p35	Y(6) = g = OUTPUT = p23
A(1)= Dip SW3 = INPUT = p55	Y(2) = c = OUTPUT = p32	
A(0)= Dip SW4 = INPUT = p56	Y(3) = d = OUTPUT = p30	
COMMON = DIGIT1 = OUTPUT = p31	Y(4) = e = OUTPUT = p27	

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
A<0>	Input	p56	BANK4	LVCMS33	N/A	3.30					Unknown		
A<1>	Input	p55	BANK4	LVCMS33	N/A	3.30					Unknown		
A<2>	Input	p53	BANK5	LVCMS33	N/A	3.30					Unknown		
A<3>	Input	p52	BANK5	LVCMS33	N/A	3.30					Unknown		
COMMON	Output	p31	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<0>	Output	p40	BANK5	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<1>	Output	p35	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<2>	Output	p32	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<3>	Output	p30	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<4>	Output	p27	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<5>	Output	p25	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<6>	Output	p23	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมจะรีบอุปกรณ์ที่ออกแบบลงชิป FPGA แล้วให้ทดสอบ ON-OFF Dip SW1-Dip SW4 โดยเริ่มจาก ON Dip SW1-Dip SW4 ทุกตัว (คือ A = “0000”) ทีละตัวจนกระทั่งถึง OFF Dip SW1-Dip SW4 ทุกตัว (คือ A = “1111”) พร้อมกับสังเกตที่ตัวแสดงผลเซกเมนต์ว่ามีส่วนใดติดสว่าง ให้ลองเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ แล้วบันทึกผลการทดลอง

จากนั้นทำการแก้ไขโค้ดเป็นวงจรอดอร์หัสแบบเลขฐานสิบแบบแอลจาร์ม แล้วทำการทดสอบซ้ำ

แบบฝึกหัด จงสร้างวงจรบวกเลขที่สามารถบวกเลขจำนวนเต็มบวกขนาด 3 บิต โดยแสดงผลทางตัวแสดงผลเซกเมนต์

การทดลองที่ 3.3.3 วงจรบวกเลข 3 บิตที่แสดงผลทางตัวแสดงผลเซเว่นเซกเมนต์

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับการประยุกต์ใช้งานครอครหัสตัวแสดงผลเซเว่นเซกเมนต์
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างวงจรบวกและโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรบวกเลข 3 บิตที่แสดงผลทางตัวแสดงผลเซเว่นเซกเมนต์ด้วย CPLD

สร้างไฟล์โค้ดดังรูป L1.1 ใน Project Location ชื่อ ch3v แล้วกำหนด Project Name และ Source File ชื่อ ex3_3_3vcx1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex3_3_3vcx1 is
7     Port ( A,B : in STD_LOGIC_VECTOR (2 downto 0);
8             COM : out STD_LOGIC;
9             Y : out STD_LOGIC_VECTOR (6 downto 0));
10 end ex3_3_3vcx1;
11
12 architecture Behavioral of ex3_3_3vcx1 is
13     signal X : STD_LOGIC_VECTOR (3 downto 0);
14 begin
15     -----3 bits adder with 4 bits output-----
16     X <= ('0' &A) + ('0' &B);
17     -----HEX to 7-segments decoder-----
18     Y <= "1110001" when X="1111" else --F
19         "111001" when X="1110" else --E      Y(0)=a
20         "101110" when X="1101" else --d      ---
21         "0111001" when X="1100" else --C      Y(5)=f  | Y(1)=b
22         "1111100" when X="1011" else --b      --- Y(6)=g
23         "1110111" when X="1010" else --A      Y(4)=e  | Y(2)=c
24         "1101111" when X="1001" else --9      ---
25         "1111111" when X="1000" else --8      Y(3)=d
26         "0000111" when X="0111" else --7
27         "1111101" when X="0110" else --6
28         "1101101" when X="0101" else --5
29         "1100110" when X="0100" else --4
30         "1001111" when X="0011" else --3
31         "1011011" when X="0010" else --2
32         "0000110" when X="0001" else --1
33         "0111111";                         --0
34     -----Common cathode-----
35     COM <= '0';
36 end Behavioral;
```

รูปที่ L1.1 วงจรบวกเลข 3 บิตที่แสดงผลทางตัวแสดงผลเซเว่นเซกเมนต์

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1 ถึง PB6 เป็นอินพุต มีตัวแสดงผล DIGIT1 เป็นเอาต์พุต กล่าวคือ

$$A(2) = PB1 = \text{INPUT} = p39 \quad Y(0) = a = \text{OUTPUT} = p27 \quad Y(6) = g = \text{OUTPUT} = p18$$

$$A(1) = PB2 = \text{INPUT} = p40 \quad Y(1) = b = \text{OUTPUT} = p26 \quad \text{COM} = \text{DIGIT1} = \text{OUTPUT} = p34$$

$$A(0) = PB3 (\text{Slide SW1}) = \text{INPUT} = p42 \quad Y(2) = c = \text{OUTPUT} = p25$$

$$B(2) = PB4 (\text{Slide SW2}) = \text{INPUT} = p43 \quad Y(3) = d = \text{OUTPUT} = p24$$

B(1) = PB5 (Slide SW3)= INPUT= p44 Y(4) = e = OUTPUT= p22

B(0) = PB6 (Slide SW4)= INPUT= p1 Y(5) = f = OUTPUT = p20

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
A<2>	Input	p39	2	9		
A<1>	Input	p40	2	11		
A<0>	Input	p42	2	14		
B<2>	Input	p43	2	15		
B<1>	Input	p44	2	17		
B<0>	Input	p1	1	2		
COM	Output	p34	4	17	SLOW	
Y<6>	Output	p18	3	11	SLOW	
Y<5>	Output	p20	3	15	SLOW	
Y<4>	Output	p22	3	17	SLOW	
Y<3>	Output	p24	3	16	SLOW	
Y<2>	Output	p25	4	2	SLOW	
Y<1>	Output	p26	4	5	SLOW	
Y<0>	Output	p27	4	8	SLOW	

รูปที่ L1.2 Assign Package Pins

หลังจากโปรแกรม CPLD แล้วให้กดปุ่ม PB1-PB6 ร่วมกับ Slide SW1-Slide SW4 เพื่อสร้างเป็นอินพุตค่าต่างๆ แล้วให้สังเกตเอาต์พุตว่าเป็นไปตามทฤษฎีหรือไม่ แล้วจึงบันทึกผลการทดลอง

2 สร้างวงจรบวกเลข 3 บิตที่แสดงผลทางตัวแสดงผลเซเว่นเซกเมนต์ด้วย FPGA

นำໄດ້ໃນรูป L1.1 มาแก้ชื่อ Entity เป็น ex3_3_vf และนำไปสร้างไว้ใน Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_3_vf

การกำหนดขาสัญญาณต่างๆ จะใช้ Dip SW1-Dip SW6 เป็นอินพุต มีตัวแสดงผล DIGIT1 เป็นเอาต์พุต ก่อร่างกาย

A(2) = Dip SW1 = INPUT= p52 Y(0) = a = OUTPUT = p40 Y(6) = g = OUTPUT = p23

A(1) = Dip SW2 = INPUT= p53 Y(1) = b = OUTPUT = p35 COM= DIGIT1= OUTPUT= p31

A(0) = Dip SW3 = INPUT= p55 Y(2) = c = OUTPUT = p32

B(2) = Dip SW4 = INPUT= p56 Y(3) = d = OUTPUT= p30

B(1) = Dip SW5 = INPUT= p59 Y(4) = e = OUTPUT= p27

B(0) = Dip SW6 = INPUT= p60 Y(5) = f = OUTPUT = p25

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Bank	IO Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
A<0>	Input	p55	BANK4	LVCMS33	N/A	3.30					Unknown		
A<1>	Input	p53	BANK5	LVCMS33	N/A	3.30					Unknown		
A<2>	Input	p52	BANK5	LVCMS33	N/A	3.30					Unknown		
B<0>	Input	p60	BANK4	LVCMS33	N/A	3.30					Unknown		
B<1>	Input	p59	BANK4	LVCMS33	N/A	3.30					Unknown		
B<2>	Input	p56	BANK4	LVCMS33	N/A	3.30					Unknown		
COM	Output	p31	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<6>	Output	p40	BANK5	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<5>	Output	p35	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<4>	Output	p32	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<3>	Output	p30	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<2>	Output	p27	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<1>	Output	p25	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<0>	Output	p23	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมจะที่ออกแบบลงชิป FPGA แล้วให้ทดลอง ON-OFF Dip SW1-Dip SW6 เพื่อสร้างเป็นอินพุตค่าต่างๆ แล้วให้สังเกตเอาต์พุตว่าเป็นไปตามทฤษฎีหรือไม่ แล้วจึงบันทึกผลการทดลอง

3.4 การสร้างวงจรเปรียบเทียบ

วงจรเปรียบเทียบ (Comparator) เป็นวงจรคอมปิเนชันที่ใช้ในการเปรียบเทียบเลข 2 จำนวน ตารางความจริงสำหรับวงจรเปรียบเทียบขนาด 1 บิตแสดงดังรูปที่ 3.22 โดยคดของวงจรเปรียบเทียบสามารถเขียนได้หลายวิธี ขึ้นอยู่กับสีตัวการเปรียบของแต่ละคน เช่น ตัวอย่างแสดงดังรูปที่ 3.23 และรูปที่ 3.24 เป็นด้าน

A	B	$E = (A = B)$	$G = (A > B)$	$L = (A < B)$
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

รูปที่ 3.22 ตารางความจริงสำหรับวงจรเปรียบ 1 บิต

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COMPARATOR_1BIT is
6   Port ( A,B : in STD_LOGIC;
7         E,G,L : out STD_LOGIC);
8 end COMPARATOR_1BIT;
9
10 architecture Behavioral of COMPARATOR_1BIT is
11 begin
12   E <= '1' when (A=B) else
13     '0';
14   G <= '1' when (A>B) else
15     '0';
16   L <= '1' when (A<B) else
17     '0';
18 end Behavioral;
```

รูปที่ 3.23 โดยคดของวงจรเปรียบเทียบ 1 บิตที่เขียนโดยใช้คำสั่ง Conditional signal assignment 3 คำสั่ง

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COMPARATOR_1BIT is
6   Port ( A,B : in STD_LOGIC;
7         E,G,L : out STD_LOGIC);
8 end COMPARATOR_1BIT;
9
10 architecture Behavioral of COMPARATOR_1BIT is
11 begin
12 process(A,B)
13 begin
14   if (A=B) then E <= '1'; G <= '0'; L <= '0';
15   elsif (A>B) then E <= '0'; G <= '1'; L <= '0';
16   elsif (A<B) then E <= '0'; G <= '0'; L <= '1';
17   else
18     E <= '0'; G <= '0'; L <= '0';
19   end if;
20 end process;
21 end Behavioral;
```

รูปที่ 3.24 โดยคดของวงจรเปรียบเทียบ 1 บิตที่เขียนโดยใช้คำสั่ง if

ในกรณีที่เป็นอินพุตแบบบัสซิ่งมีหลายบิตก็จะใช้หลักการเดียวกัน แต่ชนิดข้อมูลของอินพุต A และ B ในบรรทัดที่ 6 ในรูปที่ 3.23 และรูปที่ 3.24 จะเป็นชนิดข้อมูล std_logic_vector เช่น ถ้า 4 บิตก็จะเป็น A,B : in std_logic_vector(3 downto 0);

การทดลองที่ 3.4.1 การสร้างวงจรเปรียบเทียบ

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานของวงจรเปรียบเทียบ (Comparator)
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างวงจรเปรียบเทียบและโปรแกรมลงชิป CPLD หรือ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรเปรียบเทียบ 3 บิตด้วย CPLD

การทดลองนี้ใช้ Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_4_1vcx1 จากนั้นทำการเขียนโค้ดของวงจรเปรียบเทียบ 3 บิตโดยใช้คำสั่ง Conditional signal assignment 1 คำสั่งแสดงดังรูปที่ L1.1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_4_1vcx1 is
6     generic ( N : integer := 3 );
7     port ( A,B : in STD_LOGIC_VECTOR(N-1 downto 0);
8            E,G,L : out STD_LOGIC);
9 end ex3_4_1vcx1;
10
11 architecture Behavioral of ex3_4_1vcx1 is
12 signal X : STD_LOGIC_VECTOR(2 downto 0);
13 begin
14     X <= "100" when (A=B) else
15         "010" when (A>B) else
16         "001" when (A<B) else
17         "000";
18     E <= X(2); G <= X(1); L <= X(0);
19 end Behavioral;

```

รูปที่ L1.1 โค้ดของวงจรเปรียบเทียบ 3 บิตที่เขียนโดยใช้คำสั่ง Conditional signal assignment 1 คำสั่ง

การกำหนดขาสัญญาณต่างๆจะใช้ปุ่มกด PB1-PB6 เป็นอินพุต มี LED1-LED3 เป็นเอาต์พุต กล่าวคือ

A(2) = PB1 = INPUT= p39 B(2) = PB4 (Slide SW2) = INPUT= p43 G = LED1 = OUTPUT = p38

A(1) = PB2 = INPUT= p40 B(1) = PB5 (Slide SW3) = INPUT= p44 E = LED2 = OUTPUT = p37

A(0) = PB3 (Slide SW1)= INPUT= p42 B(0) = PB6 (Slide SW4) = INPUT= p1 L = LED3 = OUTPUT = p36

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
A<2>	Input	p39	2	9		
A<1>	Input	p40	2	11		
A<0>	Input	p42	2	14		
B<2>	Input	p43	2	15		
B<1>	Input	p44	2	17		
B<0>	Input	p1	1	2		
E	Output	p38	2	8		
G	Output	p37	2	6		
L	Output	p36	2	5		

รูปที่ L1.2 Assign Package Pins

หลังจากโปรแกรม CPLD แล้วให้กดปุ่ม PB1-PB6 และให้สังเกตคุณผลที่ LED1-LED3 ว่าเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ บันทึกผลการทดลอง จากนั้นให้แก้ไขโค้ดวงจรเปรียบเทียบ 3 บิต ในรูปที่ L1.1 โดยใช้คำสั่ง if แล้วทำการทดลองซ้ำ

2 สร้างวงจรเปรียบเทียบ 4 บิตด้วย FPGA

การทดลองนี้ใช้ Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_4_1vf จากนั้นทำการเขียนโค้ดของวงจรเปรียบเทียบ 4 บิตโดยใช้คำสั่ง Conditional signal assignment 1 คำสั่งแสดงดังรูปที่ L2.1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_4_1vf is
6     generic ( N : integer := 4 );
7     port ( A,B : in STD_LOGIC_VECTOR(N-1 downto 0);
8            E,G,L : out STD_LOGIC);
9 end ex3_4_1vf;
10
11 architecture Behavioral of ex3_4_1vf is
12     signal X : STD_LOGIC_VECTOR(2 downto 0);
13 begin
14     X <= "100" when (A=B) else
15         "010" when (A>B) else
16         "001" when (A<B) else
17         "000";
18     E <= X(2); G <= X(1); L <= X(0);
19 end Behavioral;

```

รูปที่ L2.1 โค้ดของวงจรเปรียบเทียบ 4 บิตที่เขียนโดยใช้คำสั่ง Conditional signal assignment 1 คำสั่ง

การกำหนดขาสัญญาณต่างๆ จะใช้ Dip SW1-Dip SW8 เป็นอินพุต และ LED L0-L2 เป็นเอาต์พุต กล่าวคือ

A(3) = Dip SW1 = INPUT = p52 B(3) = Dip SW1 = INPUT = p59 E = L2 = OUTPUT = p69

A(2) = Dip SW1 = INPUT = p53 B(2) = Dip SW1 = INPUT = p60 G = L1 = OUTPUT = p77

A(1) = Dip SW1 = INPUT = p55 B(1) = Dip SW1 = INPUT = p63 L = L0 = OUTPUT = p70

A(0) = Dip SW1 = INPUT = p56 B(0) = Dip SW1 = INPUT = p68

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
A<0>	Input	p52	BANK LVC MOS33	N/A	3.30						Unknown		
A<1>	Input	p53	BANK LVC MOS33	N/A	3.30						Unknown		
A<2>	Input	p55	BANK LVC MOS33	N/A	3.30						Unknown		
A<3>	Input	p56	BANK LVC MOS33	N/A	3.30						Unknown		
B<0>	Input	p59	BANK LVC MOS33	N/A	3.30						Unknown		
B<1>	Input	p60	BANK LVC MOS33	N/A	3.30						Unknown		
B<2>	Input	p63	BANK LVC MOS33	N/A	3.30						Unknown		
B<3>	Input	p68	BANK LVC MOS33	N/A	3.30						Unknown		
E	Output	p69	BANK LVC MOS33	N/A	3.30				SLOW		Unknown		
G	Output	p77	BANK LVC MOS33	N/A	3.30				SLOW		Unknown		
L	Output	p70	BANK LVC MOS33	N/A	3.30				SLOW		Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรม FPGA ให้ ON-OFF Dip SW1-Dip SW8 และสังเกตคุณผลที่ LED L0-L2 ว่าเอาต์พุตเป็นตามทฤษฎีหรือไม่ บันทึกผลการทดลอง จากนั้นแก้ไขโค้ดวงจรเปรียบเทียบ 4 บิตในรูปที่ L2.1 โดยใช้คำสั่ง if แล้วทำการทดลองซ้ำ

แบบฝึกหัด

- ให้เขียนโค้ดของวงจรเปรียบเทียบทหาร 3 บิตโดยใช้คำสั่ง Conditional signal assignment และใช้คำสั่ง if โดยที่ GE = '1' ก็ต่อเมื่อ A >= B

3.5 วงจรเข้ารหัส

วงจรเข้ารหัสเป็นวงจรคอมบินेशันที่ใช้ทำหน้าที่ตรงข้ามกับวงจรจดจำรหัส เช่น วงจรเข้ารหัสสวิตช์ (Switch encoder) ที่ใช้ในการเปลี่ยนสัญญาณalog ที่ได้จากการกดปุ่มของสวิตช์เพื่อแปลงเป็นรหัสเลขในรูปแบบ binary ปัญหาของ Encoder แบบธรรมดาก็คือ เมื่อกดปุ่มพร้อมๆ กันหลายปุ่มจะงงใจเข้ารหัสผิดพลาด การแก้ไขปัญหานี้ทำได้โดยใช้วงจรเข้ารหัสที่มีลำดับความสำคัญ (Priority encoder)

ตารางความจริงของวงจรเข้ารหัสที่มีลำดับความสำคัญดังรูปที่ 3.25 ใช้แปลงสัญญาณปุ่มกดแบบ Active low (กล่าวคือ เมื่อกดปุ่มจะให้รหัส ‘0’) เป็นรหัส BCD (หรือเลขฐานสิบเศษเลข 0-9) และเมื่อกดปุ่มพร้อมกันหลายปุ่มจะรู้ว่ากดปุ่มใดก่อน สำหรับโดยให้อาต์พุตเป็นของปุ่มกดที่มีเลขเป็นค่าสูงสุดเท่านั้น ถ้าไม่มีการกดปุ่มเลขจะให้อาต์พุตเป็น “1111” โดยที่ “-” คือ Don’t care โดยวงจร Priority encoder มีรายละเอียดดังที่ได้อธิบายไปแล้วในบทที่ 2 ตัวอย่างที่ 2.13 และตัวอย่างที่ 2.24

No.	Input A										Output B			
	A(9)	A(8)	A(7)	A(6)	A(5)	A(4)	A(3)	A(2)	A(1)	A(0)	B(3)	B(2)	B(1)	B(0)
	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1	0	-	0	0	0	1
2	1	1	1	1	1	1	1	0	-	-	0	0	1	0
3	1	1	1	1	1	1	0	-	-	-	0	0	1	1
4	1	1	1	1	1	0	-	-	-	-	0	1	0	0
5	1	1	1	1	0	-	-	-	-	-	0	1	0	1
6	1	1	1	0	-	-	-	-	-	-	0	1	1	0
7	1	1	0	-	-	-	-	-	-	-	0	1	1	1
8	1	0	-	-	-	-	-	-	-	-	1	0	0	0
9	0	-	-	-	-	-	-	-	-	-	1	0	0	1

รูปที่ 2.25 ตารางความจริงของวงจรเข้ารหัสที่มีลำดับความสำคัญที่ใช้แปลงสัญญาณปุ่มกดเป็นรหัส BCD

การทดลองที่ 3.5.1 วงจรเข้ารหัส

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานของวงจรเข้ารหัสที่มีลำดับความสำคัญ (Priority encoder)
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างวงจรเข้ารหัสที่มีลำดับความสำคัญและโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจร Priority encoder ที่ใช้แปลงสัญญาณปุ่มกดเป็นรหัส BCD ด้วย CPLD

การทดลองนี้ใช้ Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_5_1vcx1 จากนั้นทำการเขียนโค้ดของวงจรตั้ง Priority encoder รูปที่ L1.1 โดยการทำงานจะเป็นตามตารางความจริงในรูปที่ 2.30

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.all;
4
5 entity ex3_5_1vcx1 is
6     Port ( A : in STD_LOGIC_VECTOR (9 downto 0);
7             B : out STD_LOGIC_VECTOR (3 downto 0));
8 end ex3_5_1vcx1;
9
10 architecture Behavioral of ex3_5_1vcx1 is
11 begin
12     B <= "1001" when A(9)'=0' else --Key pad No.9
13         "1000" when A(8)'=0' else --Key pad No.8
14         "0111" when A(7)'=0' else --Key pad No.7
15         "0110" when A(6)'=0' else --Key pad No.6
16         "0101" when A(5)'=0' else --Key pad No.5
17         "0100" when A(4)'=0' else --Key pad No.4
18         "0011" when A(3)'=0' else --Key pad No.3
19         "0010" when A(2)'=0' else --Key pad No.2
20         "0001" when A(1)'=0' else --Key pad No.1
21         "0000" when A(0)'=0' else --Key pad No.0
22         "1111";
23 end Behavioral;

```

รูปที่ L1.1 โค้ดของวงจร Priority encoder ที่ใช้แปลงสัญญาณปุ่มกดแบบ Active low เป็นรหัส BCD

การกำหนดสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB6 และ IO6-IO9 เป็นอินพุต มี LED1-LED4 เป็นเอาต์พุต กล่าวคือ

A(0) = PB1 = INPUT= p39	A(5) = PB6 = INPUT= p1	B(3) = LED1 = OUTPUT= p38
A(1) = PB2 = INPUT= p40	A(6) = IO6 = INPUT= p7	B(2) = LED2 = OUTPUT= p37
A(2) = PB3 = INPUT= p42	A(7) = IO7 = INPUT= p6	B(1) = LED3 = OUTPUT= p36
A(3) = PB4 = INPUT= p43	A(8) = IO8 = INPUT= p4	B(0) = LED4 = OUTPUT= p35
A(4) = PB5 = INPUT= p44	A(9) = IO9 = INPUT= p3	

โดยพิมพ์ใน Assign Package Pins สรุปดังรูปที่ 2.2 หลังจากโปรแกรม CPLD ให้ทดลองกดปุ่ม PB1-PB6 หรือต่อ IO6-IO9 ลงกราว์ด (โดยใช้ Jumper เสี่ยงเข้ากับขาที่อยู่ด้านตรงข้ามกับ IO6-IO9) ให้สังเกตคุณลักษณะ LED1-LED4 ให้อาต์พุตเป็นไปตามทฤษฎีหรือไม่ หากนั้นให้ทดลองกดพร้อมกันหลายปุ่มแล้วเทียบผลการทดลองกับการกดเพียงปุ่มเดียวว่าให้ผลเป็นอย่างไร แล้วจึงบันทึกผลการทดลอง

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
A<9>	Input	p3	1	6		
A<8>	Input	p4	1	8		
A<7>	Input	p5	1	11		
A<6>	Input	p7	1	14		
A<5>	Input	p1	1	2		
A<4>	Input	p44	2	17		
A<3>	Input	p43	2	15		
A<2>	Input	p42	2	14		
A<1>	Input	p40	2	11		
A<0>	Input	p39	2	9		
B<3>	Output	p38	2	8	SLOW	
B<2>	Output	p37	2	6	SLOW	
B<1>	Output	p36	2	5	SLOW	
B<0>	Output	p35	2	2	SLOW	

รูปที่ L1.2 Assign Package Pins

2 สร้างวงจร Priority encoder ที่ใช้แปลงสัญญาณปุ่มกดเป็นรหัส BCD ด้วย FPGA

นำโค้ดในรูป L1.1 มาเก็บชื่อ Entity เป็น ex3_5_1vf และนำไปสร้างไว้ใน Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_5_1vf

การกำหนดสัญญาณต่างๆ จะใช้ PB1-PB5 และ Dip SW1-Dip SW5 เป็นอินพุต มี LED L0-L3 เป็นเอาต์พุต กล่าวคือ

$$\begin{aligned}
 A(0) &= PB1 = \text{INPUT} = p44 & A(5) &= \text{Dip SW1} = \text{INPUT} = p52 & B(3) &= L3 = \text{OUTPUT} = p76 \\
 A(1) &= PB2 = \text{INPUT} = p46 & A(6) &= \text{Dip SW2} = \text{INPUT} = p53 & B(2) &= L2 = \text{OUTPUT} = p69 \\
 A(2) &= PB3 = \text{INPUT} = p47 & A(7) &= \text{Dip SW3} = \text{INPUT} = p55 & B(1) &= L1 = \text{OUTPUT} = p77 \\
 A(3) &= PB4 = \text{INPUT} = p50 & A(8) &= \text{Dip SW4} = \text{INPUT} = p56 & B(0) &= L0 = \text{OUTPUT} = p70 \\
 A(4) &= PB5 = \text{INPUT} = p51 & A(9) &= \text{Dip SW5} = \text{INPUT} = p59
 \end{aligned}$$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
A<9>	Input	p44	BANK LVCMOS33	N/A	3.30						Unknown		
A<1>	Input	p46	BANK LVCMOS33	N/A	3.30						Unknown		
A<2>	Input	p47	BANK LVCMOS33	N/A	3.30						Unknown		
A<3>	Input	p50	BANK LVCMOS33	N/A	3.30						Unknown		
A<4>	Input	p51	BANK LVCMOS33	N/A	3.30						Unknown		
A<5>	Input	p52	BANK LVCMOS33	N/A	3.30						Unknown		
A<6>	Input	p53	BANK LVCMOS33	N/A	3.30						Unknown		
A<7>	Input	p55	BANK LVCMOS33	N/A	3.30						Unknown		
A<8>	Input	p56	BANK LVCMOS33	N/A	3.30						Unknown		
A<9>	Input	p59	BANK LVCMOS33	N/A	3.30						Unknown		
B<0>	Output	p70	BANK LVCMOS33	N/A	3.30				SLOW		Unknown		
B<1>	Output	p77	BANK LVCMOS33	N/A	3.30				SLOW		Unknown		
B<2>	Output	p69	BANK LVCMOS33	N/A	3.30				SLOW		Unknown		
B<3>	Output	p76	BANK LVCMOS33	N/A	3.30				SLOW		Unknown		

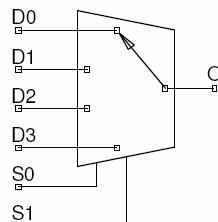
รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมจะที่ออกแบบลงชิป FPGA แล้วให้ทดลองกด ปุ่ม PB1-PB5 และ ON_OFF Dip SW1-Dip SW5 และให้สังเกตผลว่า LED L0-L3 ให้ออกเจ้าต์พุตเป็นไปตามทฤษฎีหรือไม่ แล้วให้ทดลองกดพร้อมกันหลายปุ่มและเก็บผล การทดลองกับการกดเพียงปุ่มเดียวว่าให้ผลเป็นอย่างไร จากนั้นจึงบันทึกผลการทดลอง

แบบฝึกหัด ให้เขียนโค้ด Priority encoder ที่ใช้แปลงสัญญาณปุ่มกดแบบ Active low เป็นรหัส BCD โดยใช้คำสั่ง selected signal assignment และคำสั่ง case

3.6 วงจรมัลติเพล็กเซอร์

วงจรมัลติเพล็กเซอร์ (Multiplexer) หรือ MUX เป็นวงจรคอมบินेशันที่ใช้ในการเลือกอินพุตเพื่อส่งออกไปที่เอาต์พุต ครั้งละ 1 อินพุต รูปที่ 3.26 แสดงหลักการทำงานของมัลติเพล็กเซอร์แบบ 4 อินพุต 1 เอาต์พุต ซึ่งสามารถเลือกอินพุต D0–D3 ไปออกที่เอาต์พุต O ครั้งละ 1 อินพุต โดยใช้ Control S0 และ S1 เป็นตัวกำหนดว่าจะเลือกอินพุตใด



รูปที่ 3.26 หลักการทำงานของมัลติเพล็กเซอร์แบบ 4 อินพุต 1 เอาต์พุต

ตัวอย่างการเขียนโค้ดของวงจรมัลติเพล็กเซอร์โดยใช้คำสั่งต่อๆ กันๆ มีรายละเอียดในบทที่ 2 ตัวอย่างที่ 2.7 ตัวอย่างที่ 2.8 ตัวอย่างที่ 2.18 และตัวอย่างที่ 2.19

ตัวอย่างโค้ดที่ใช้คำสั่ง Conditional signal assignment ของวงจรมัลติเพล็กเซอร์แบบ 2 อินพุต 1 เอาต์พุต (2 to 1 Multiplexer) และแบบ 4 อินพุต 1 เอาต์พุต (4 to 1 Multiplexer) แบบบัญชายาบิจจะแสดงดังรูปที่ 3.27 และรูปที่ 3.28 ตามลำดับ

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity MUX2to1_4BITS is
6     generic ( N : integer :=4 );
7     Port ( DO,D1 : in STD_LOGIC_VECTOR (N-1 downto 0);
8             S : in STD_LOGIC;
9             O : out STD_LOGIC_VECTOR (N-1 downto 0));
10 end MUX2to1_4BITS;
11
12 architecture Behavioral of MUX2to1_4BITS is
13 begin
14     O <= DO when S = '0' else
15         D1;
16 end Behavioral;

```

รูปที่ 3.27 โค้ดของวงจรมัลติเพล็กเซอร์ขนาด 4 บิตแบบ 2 อินพุต 1 เอาต์พุต

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity MUX4to1_4BITS is
6     generic ( N : integer :=4 );
7     Port ( DO,D1,D2,D3 : in STD_LOGIC_VECTOR (N-1 downto 0);
8             S : in STD_LOGIC_VECTOR (1 downto 0);
9             O : out STD_LOGIC_VECTOR (N-1 downto 0));
10 end MUX4to1_4BITS;
11
12 architecture Behavioral of MUX4to1_4BITS is
13 begin
14     O <= DO when S = "00" else
15         D1 when S = "01" else
16             D2 when S = "10" else
17                 D3;
18 end Behavioral;

```

รูปที่ 3.28 โค้ดของวงจรมัลติเพล็กเซอร์ขนาด 4 บิตแบบ 4 อินพุต 1 เอาต์พุต

การทดลองที่ 3.6.1 วงจร 4 to 1 Multiplexer

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานและออกแบบวงจร 4 to 1 Multiplexer
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างวงจร 4 to 1 Multiplexer และโปรแกรมลงชิป CPLD หรือ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจร 4 to 1 Multiplexer ด้วย CPLD

สร้างไฟล์ใน Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_6_1vcx1 จากนั้นทำการเขียนโค้ดดังรูปที่ L1.1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_6_1vcx1 is
6     Port ( D0,D1,D2,D3 : in STD_LOGIC;
7             S : in STD_LOGIC_VECTOR (1 downto 0);
8             O : out STD_LOGIC);
9 end ex3_6_1vcx1;
10
11 architecture Behavioral of ex3_6_1vcx1 is
12 begin
13     O <= D0 when S = "00" else
14         D1 when S = "01" else
15         D2 when S = "10" else
16         D3;
17 end Behavioral;

```

รูปที่ L1.1 โค้ดของวงจร 4 to 1 Multiplexer

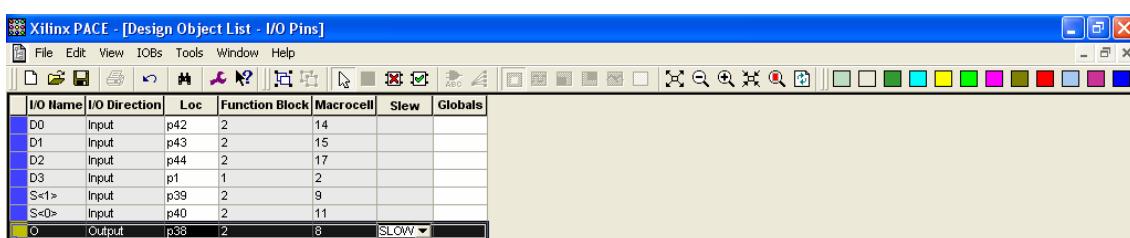
การกำหนดขาสัญญาณต่างๆจะใช้ปุ่มกด PB1 ถึง PB6 เป็นอินพุต มี LED1 เป็นเอาต์พุต กล่าวคือ

$$D0 = PB3 (\text{SLIDE SW1}) = p42 \quad D2 = PB5 (\text{SLIDE SW3}) = p44 \quad S(1) = PB1 = p39$$

$$D1 = PB4 (\text{SLIDE SW2}) = p43 \quad D3 = PB6 (\text{SLIDE SW4}) = p41 \quad S(0) = PB2 = p40$$

$$O = LED1 = p38$$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



รูปที่ L1.2 Assign Package Pins

หลังຈາກໄປຮັດການຈະໃຫ້ທົດລອງກົດປຸ່ມ PB1-PB2 ແລະ ON-OFF PB3 (Slide SW1) ປຶ້ງ PB6 (Slide SW4) ແລ້ວໃຫ້ສັງເກດຄູພຸດທີ່ LED1 ວ່າຕິດສ່ວ່າໂດຍໃຫ້ລອຈິກເອາຫຼຸດເປັນໄປຕາມທຸນຸ້ງຂຶ້ນ ຂະໜັນທຶນກຳພັດການທົດລອງ

2 ສ່ຽງວ່າງຈົບ 4 to 1 Multiplexer ດ້ວຍ FPGA

ສ່ຽງໄຟຟ້າໃນ Project Location ຊ່ອ ch3v ແລ້ວກໍາທັນ Project Name ແລະ Source File ຊ່ອ ex3_6_2vf ຈາກນັ້ນກໍາທັນເກີຍໂຄດວັງຈຽດຮູບທີ່ L2.1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_6_1vf is
6     Port ( D0,D1,D2,D3 : in STD_LOGIC;
7             S : in STD_LOGIC_VECTOR (1 downto 0);
8             O : out STD_LOGIC);
9 end ex3_6_1vf;
10
11 architecture Behavioral of ex3_6_1vf is
12 begin
13     O <= D0 when S = "00" else
14         D1 when S = "01" else
15         D2 when S = "10" else
16         D3;
17 end Behavioral;
```

ຮູບທີ່ L2.1 ໂຄິດຂອງຈົບ 4 to1 Multiplexer

ກໍາທັນຂາສ້າງໝານຈະໃຊ້ປຸ່ມກົດ PB1-PB4 ແລະ Dip SW1-Dip SW2 ເປັນອິນພຸດ ມີ LED L0 ເປັນເອາຫຼຸດ ກລ່າວຄື່ອງ

$$D0 = \text{Dip SW1} = p44 \quad D2 = \text{Dip SW3} = p47 \quad S(1) = \text{PB1} = p52$$

$$D1 = \text{Dip SW2} = p46 \quad D3 = \text{Dip SW4} = p50 \quad S(0) = \text{PB2} = p53$$

$$O = L0 = p70$$

ໂດຍພິມຟີ່ໃນ Assign Package Pins ສຽງດັ່ງນີ້

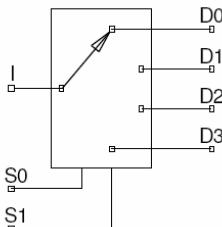
Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
D0	Input	p44	BANK LVCMOS33	N/A	3.30						Unknown		
D1	Input	p46	BANK LVCMOS33	N/A	3.30						Unknown		
D2	Input	p47	BANK LVCMOS33	N/A	3.30						Unknown		
D3	Input	p50	BANK LVCMOS33	N/A	3.30						Unknown		
O	Output	p70	BANK LVCMOS33	N/A	3.30			SLOW			Unknown		
S<0>	Input	p53	BANK LVCMOS33	N/A	3.30						Unknown		
S<1>	Input	p52	BANK LVCMOS33	N/A	3.30						Unknown		

ຮູບທີ່ L2.2 Assign Package Pins

หลังຈາກໄປຮັດການຈະໃຫ້ທົດລອງກົດປຸ່ມ PB1-PB4 ແລ້ວໃຫ້ທົດລອງກົດປຸ່ມ PB3 ແລະ ON-OFF Dip SW1 ແລະ Dip SW2 ແລ້ວໃຫ້ສັງເກດຄູພຸດທີ່ L0 ວ່າຕິດສ່ວ່າໂດຍໃຫ້ລອຈິກເອາຫຼຸດເປັນໄປຕາມທຸນຸ້ງຂຶ້ນ ຂະໜັນທຶນກຳພັດການທົດລອງ

3.7 วงจรดิมัลติเพล็กเซอร์

วงจรดิมัลติเพล็กเซอร์ (Demultiplexer) หรือ DEMUX เป็นวงจรคอมบินेशันที่ทำงานตรงข้ามกับวงจรมัลติเพล็กเซอร์ ตัวอย่างคือมัลติเพล็กเซอร์แสดงดังรูปที่ 3.29 มีตารางความจริงแสดงดังรูปที่ 3.30 มีโค้ด VHDL แสดงดังรูปที่ 3.31 และรูปที่ 3.32



รูปที่ 3.29 แสดงวงจรดิมัลติเพล็กเซอร์แบบเข้า 1 อินพุตออก 4 เอาต์พุต

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 3.30 ตารางความจริงของวงจรดิมัลติเพล็กเซอร์แบบเข้า 1 อินพุตออก 4 เอาต์พุต

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DMUX1to4 is
6   Port ( I,S1,S2 : in STD_LOGIC;
7         D3,D2,D1,DO : out STD_LOGIC);
8 end DMUX1to4;
9
10 architecture Behavioral of DMUX1to4 is
11   signal S : STD_LOGIC_VECTOR(1 downto 0);
12 begin
13   S <= S1&S0;
14   DO <= I when S = "00" else '0';
15   D1 <= I when S = "01" else '0';
16   D2 <= I when S = "10" else '0';
17   D3 <= I when S = "11" else '0';
18 end Behavioral;

```

รูปที่ 4.31 โค้ด VHDL ของวงจรดิมัลติเพล็กเซอร์แบบเข้า 1 อินพุตออก 4 เอาต์พุต

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DMUX1to4 is
6   Port ( I,S1,S0 : in STD_LOGIC;
7         DO,D1,D2,D3 : out STD_LOGIC);
8 end DMUX1to4;
9
10 architecture Behavioral of DMUX1to4 is
11   signal S : STD_LOGIC_VECTOR (1 downto 0);
12 begin
13   S <= S1&S0;
14   process(I,S)
15   begin
16     if S="00" then DO <= I; D1 <= '0'; D2 <= '0'; D3 <= '0';
17     elsif S="01" then DO <= '0'; D1 <= I; D2 <= '0'; D3 <= '0';
18     elsif S="10" then DO <= '0'; D1 <= '0'; D2 <= I; D3 <= '0';
19     elsif S="11" then DO <= '0'; D1 <= '0'; D2 <= '0'; D3 <= I;
20     else
21       DO <= '0'; D1 <= '0'; D2 <= '0'; D3 <= '0';
22     end if;
23   end process;
24 end Behavioral;

```

รูปที่ 4.32 โค้ด VHDL ของวงจรดิมัลติเพล็กเซอร์แบบเข้า 1 อินพุตออก 4 เอาต์พุต

การทดลองที่ 3.7.1 สร้างวงจร Demultiplexer 1 บิต ด้วย CPLD

วัสดุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานและออกแบบวงจร Demultiplexer (Demultiplexer)
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างวงจร Demultiplexer และโปรแกรมลงชิป CPLD หรือ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจร 1 to 4 Demultiplexer 1 บิต ด้วย CPLD

ให้สร้างวงจร 1 to 4 Demultiplexer ตามตารางความจริงดังรูปที่ 3.30 ด้วย CPLD ซึ่งมีโค้ดแสดงดังรูปที่ L1.1 โดยใช้ Project Location ชื่อ ch3v แล้วกำหนด Project Name และ Source File ชื่อ ex3_7_1vcx1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_7_1vcx1 is
6     Port ( S1,S0 : in STD_LOGIC;
7             I : in STD_LOGIC;
8             D0,D1,D2,D3 : out STD_LOGIC);
9 end ex3_7_1vcx1;
10
11 architecture Behavioral of ex3_7_1vcx1 is
12     signal SEL : STD_LOGIC_VECTOR (1 downto 0);
13 begin
14     SEL <= S1&S0;
15     D0 <= I when SEL="00" else '0';
16     D1 <= I when SEL="01" else '0';
17     D2 <= I when SEL="10" else '0';
18     D3 <= I when SEL="11" else '0';
19 end Behavioral;

```

รูปที่ L1.1 โค้ดของวงจร 1 to 4 Demultiplexer ที่เขียนด้วยคำสั่ง Condition signal assignment

การกำหนดขาสัญญาณต่างๆจะใช้ปุ่มกด PB1 ถึง PB3 เป็นอินพุต มี LED1-LED4 เป็นเอาต์พุต กล่าวคือ

$$\begin{array}{lll}
 S1 = PB1 = \text{INPUT} = p39 & D0 = \text{LED1} = \text{OUTPUT} = p38 & D2 = \text{LED3} = \text{OUTPUT} = p36 \\
 S0 = PB2 = \text{INPUT} = p40 & D1 = \text{LED2} = \text{OUTPUT} = p37 & D3 = \text{LED4} = \text{OUTPUT} = p35 \\
 I = PB3 = \text{INPUT} = p42 & &
 \end{array}$$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
S1	Input	p39	2	9		
S0	Input	p40	2	11		
I	Input	p42	2	14		
D0	Output	p38	2	8	SLOW	
D1	Output	p37	2	6	SLOW	
D2	Output	p36	2	5	SLOW	
D3	Output	p35	2	2	SLOW	

รูปที่ L1.1 Assign Package Pins

หลังจากโปรแกรมวงจรลง CPLD แล้วให้ทดสอบกับปุ่ม PB1–PB3 และให้สังเกตคุณที่ LED1–LED4 ว่าให้ล้อจิก เอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

เมื่อทดลองเสร็จแล้วให้แก้ไขโค้ดในรูปที่ L1.1 โดยใช้คำสั่ง if แล้วทำการทดลองซ้ำ

2 สร้างวงจร 1 to 4 Demultiplexer ขนาด 1 มิต ด้วย FPGA

ให้สร้างวงจร 1 to 4 Demultiplexer ตามตารางความจริงดังรูปที่ 3.30 ด้วย FPGA ซึ่งมีโค้ดแสดงดังรูปที่ L2.1 โดยใช้ Project Location ชื่อ ch3v และกำหนด Project Name และ Source File ชื่อ ex3_7_1vf

```

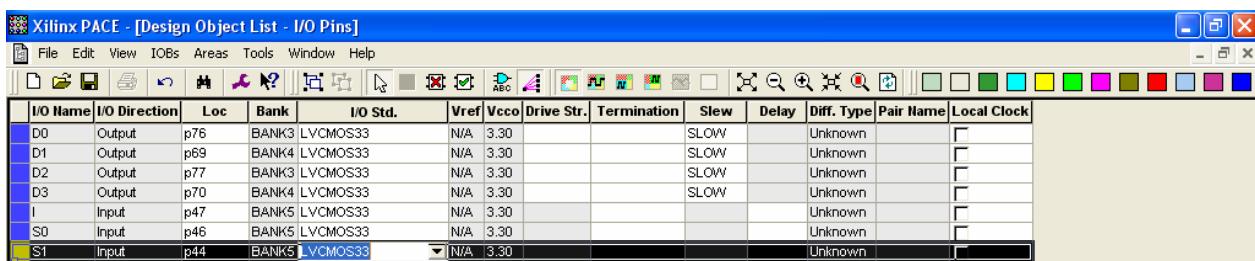
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex3_7_1vf is
6     Port ( S1,S0 : in STD_LOGIC;
7             I : in STD_LOGIC;
8             D0,D1,D2,D3 : out STD_LOGIC);
9 end ex3_7_1vf;
10
11 architecture Behavioral of ex3_7_1vf is
12     signal SEL : STD_LOGIC_VECTOR (1 downto 0);
13 begin
14     SEL <= S1&S0;
15     D0 <= I when SEL="00" else '0';
16     D1 <= I when SEL="01" else '0';
17     D2 <= I when SEL="10" else '0';
18     D3 <= I when SEL="11" else '0';
19 end Behavioral;
```

รูปที่ L2.1 โค้ดของวงจร 1 to 4 Demultiplexer ที่เขียนด้วยคำสั่ง Condition signal assignment

การกำหนดขาสัญญาณต่างๆจะใช้ปุ่มกด PB1 ถึง PB3 เป็นอินพุต มี LED L0-L3 เป็นเอาต์พุต กล่าวคือ

S1 = PB1 = INPUT = p44 D0 = L3 = OUTPUT = p73 D2 = L1 = OUTPUT = p77
 S0 = PB2 = INPUT = p46 D1 = L2 = OUTPUT = p69 D3 = L0 = OUTPUT = p70
 I = PB3 = INPUT = p47

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรลง FPGA แล้วให้ทดสอบกับปุ่ม PB1–PB3 และให้สังเกตคุณที่ LED L0–L3 ว่าให้ล้อจิก เอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

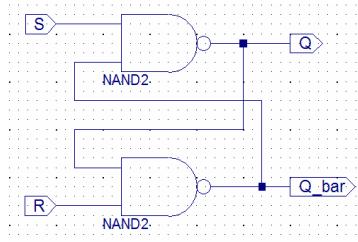
เมื่อทดลองเสร็จแล้วให้แก้ไขโค้ดในรูปที่ L2.1 โดยใช้คำสั่ง if แล้วทำการทดลองซ้ำ

บทที่ 4

แลตช์ พลีปฟลอป และวงจรซีเควนเชียล

4.1 แลตช์

แลตช์พื้นฐานได้แก่ แลตช์แบบ NAND Gate และ NOR Gate และแสดงดังรูปที่ 4.1 และรูปที่ 4.2 ตามลำดับ โดยที่ Invalid จะเป็นสถานะที่ Q และ Q_bar (หรือ \overline{Q}) มีผลจิกเหมือนกันซึ่งไม่เป็นความจริง ส่วนสถานะ Hold นั้นวงจรจะไม่มีการเปลี่ยนสถานะ (Latch) แลตช์เป็นหน่วยความจำพื้นฐานและไม่จัดว่าเป็นวงจรซีเควนเชียล เนื่องจากເອົາດີພຸດຂອງແລຕ້ນັ້ນໄຟ້ມີບັນກັບຄ່າອິນພຸດແລະ/ຫຼືຄ່າທີ່ຈຳເອາໄວ່ (State)



4.1a) ผังวงจร NAND Gate Latch

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

4.1b) ตารางความจริงของวงจร NAND Gate Latch

```

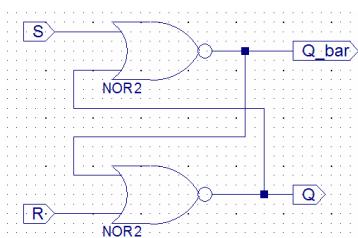
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity NAND_GATE_LATCH is
6     Port (R,S : in STD_LOGIC;
7            Q,Q_bar : buffer STD_LOGIC);
8 end NAND_GATE_LATCH;
9
10 architecture Behavioral of NAND_GATE_LATCH is
11 begin
12     Q <= S nand Q_bar;
13     Q_bar <= R nand Q;
14 end Behavioral;

```

Q และ Q_bar มี Mode เป็น buffer เมื่อจากในรูปที่ 4.1a) นั้นເອົາດີພຸດຂອງ Q และ Q_bar ຄູກຕ່ອກລັບເຂົ້າມາທີ່ອິນພຸດຂອງແນນຄໍເກຕອອກຕັ້ງທີ່ເຫດລືອ

รูปที่ 4.1c) ໂຄດຂອງวงจร NAND Gate Latch

รูปที่ 4.1 NAND Gate Latch



4.2a) ผังวงจร 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

4.2b) ตารางความจริงของวงจร NOR Gate Latch

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity NOR_GATE_LATCH is
6     Port(R,S : in STD_LOGIC;
7           Q,Q_bar : buffer STD_LOGIC);
8 end NOR_GATE_LATCH;
9
10 architecture Behavioral of NOR_GATE_LATCH is
11 begin
12     Q <= R nor Q_bar;
13     Q_bar <= S nor Q;
14 end Behavioral;

```

Q และ Q_bar มี Mode เป็น buffer เนื่องจากในรูปที่ 4.1a) นั้นเราตั้งค่า Q และ Q_bar ถูกต่อคลัมเบี้ยมที่อินพุตของnorゲートอีกตัวที่เหลือ

4.2c) โค้ดของวงจร NOR Gate Latch

รูปที่ 4.2 NOR Gate Latch

แต่ชื่อখันนิดที่ผู้อ่านควรทราบ คือ D Latch ซึ่ง D Latch จะมี D เป็นอินพุตและ Q เป็นเอาต์พุต โดยมีตารางความจริงแสดงดังรูปที่ 4.3a) และมีโค้ด VHDL แสดงดังรูปที่ 4.3b) โดยที่การแล็คช์ค่าของอินพุตจะทำได้ก็ต่อเมื่อขา Gate G = '1'

Input		Output	Remark
G	D	Q	
0	0	Q (0/1)	Hold (No change)
0	1	Q (0/1)	Hold (No change)
1	0	0	
1	1	1	

4.3a) ตารางความจริงของ D Latch

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity D_LATCH is
6     Port ( G,D : in STD_LOGIC;
7             Q : out STD_LOGIC);
8 end D_LATCH;
9
10 architecture Behavioral of D_LATCH is
11 begin
12     process (G,D)
13     begin
14         if G='1' then Q <= D;
15     end if;
16     end process;
17 end Behavioral;

```

4.3b) โค้ดของวงจร D Latch

รูปที่ 4.3 ตารางความจริงและโค้ดของวงจร D Latch

การทดลองที่ 4.1.1 แล็ตช์

วัตถุประสงค์

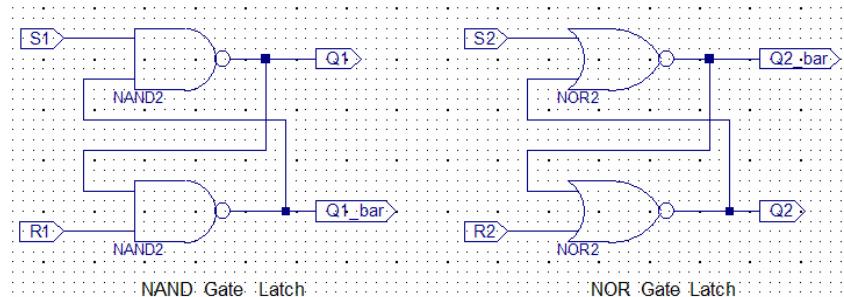
- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานของวงจรแล็ตช์ต่างๆ
- 2) เพื่อสร้างวงจรแล็ตช์โดยวิธีการเขียนด้วยโค้ด VHDL แล้วโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

บอร์ด 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 ch4v ไว้ในไดร์ฟ C และเขียนโค้ด VHDL ของวงจรผังวงจรดังรูปที่ L1.1 ได้ดังรูปที่ L1.2 จากนั้นสร้างไฟล์โดยใช้ Project Location (หรือ Folder) ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_1_1vcx1



รูปที่ L1.1 ผังวงจร NAND Gate latch และ NOR Gate latch

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_1_1vcx1 is
6     Port ( S1,R1,S2,R2 : in STD_LOGIC;
7             Q1,Q1_bar,Q2,Q2_bar : buffer STD_LOGIC);
8 end ex4_1_1vcx1;
9
10 architecture Behavioral of ex4_1_1vcx1 is
11 begin
12     -----NAND GATE LATCH-----
13     Q1 <= S1 nand Q1_bar;
14     Q1_bar <= R1 nand Q1;
15     -----NOR GATE LATCH-----
16     Q2_bar <= S2 nor Q2;
17     Q2 <= R2 nor Q2_bar;
18 end Behavioral;

```

รูปที่ L1.2 โค้ด VHDL ของวงจร NAND Gate latch และ NOR Gate latch

การกำหนดขาสัญญาณต่างๆ ให้กับวงจร โดยจะใช้ปุ่มกด PB1-PB4 เป็นอินพุตและ LED1-LED4 เป็นเอาต์พุต กล่าวคือ

S1 = PB1 = INPUT = p39

Q1 = LED1 = OUTPUT = p38

R1 = PB2 = INPUT = p40

Q1_bar = LED2 = OUTPUT = p37

S2 = PB3 = INPUT = p42

Q2 = LED3 = OUTPUT = p36

R2 = PB4 = INPUT = p43

Q2_bar = LED4 = OUTPUT = p35

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
S1	Input	p39	2	9		
R1	Input	p40	2	11		
S2	Input	p42	2	14		
R2	Input	p43	2	15		
Q1	Output	p38	2	8	SLOW	
Q1_bar	Output	p37	2	6	SLOW	
Q2	Output	p36	2	5	SLOW	
Q2_bar	Output	p35	2	2	SLOW	

รูปที่ L1.3 Assign Package Pins

หลังจากโปรแกรมว่างจรลงชิป CPLD แล้วทดลองกดปุ่ม PB1-PB4 และให้สังเกตคุณผลที่ LED1-LED4 ว่าให้ล้อจิก เอ้าต์พุตเป็นไปตามทฤษฎีหรือไม่ หากนั้นให้สังเกตคุณสถานะที่เป็น “Invalid” ว่าเป็นอย่างไร แล้วทำการบันทึกผลการทดลอง

แม้ว่า Xilinx Synthesis Tool หรือ XST จะรองรับโภมด Buffer แต่ซอฟต์แวร์ทุก XST แนะนำให้หลีกเลี่ยงการใช้ โภมด Buffer ดังนั้นเราวิจัยเขียนโค้ดใหม่ได้ดังรูปที่ L1.4 จากนั้นให้ทำการทดลองซ้ำ

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_1_lvcx1 is
6     Port ( S1,R1,S2,R2 : in STD_LOGIC;
7             Q1,Q1_bar,Q2,Q2_bar : out STD_LOGIC);
8 end ex4_1_lvcx1;
9
10 architecture Behavioral of ex4_1_lvcx1 is
11     signal Q1T,Q1T_bar,Q2T,Q2T_bar : STD_LOGIC;
12 begin
13     -----NAND GATE LATCH-----
14     Q1T <= S1 nand Q1T_bar;
15     Q1T_bar <= R1 nand Q1T;
16     Q1 <= Q1T;
17     Q1_bar <= Q1T_bar;
18     -----NOR GATE LATCH-----
19     Q2T_bar <= S2 nor Q2T;
20     Q2T <= R2 nor Q2T_bar;
21     Q2 <= Q2T;
22     Q2_bar <= Q2T_bar;
23 end Behavioral;

```

รูปที่ L1.4 โค้ด VHDL ที่เขียนโดยประการใช้ Signal

2 สร้างวงจร NAND Gate latch และ NOR Gate latch ด้วย FPGA

สร้าง Folder ชื่อ ch4v ไว้ในโฟลเดอร์ C (ในกรณีที่ยังไม่สร้าง Folder ชื่อ ch4v) จากนั้นออกแบบ NAND Gate latch และ NOR Gate latch ซึ่งมีผังวงจรแสดงดังรูปที่ L1.1 และโค้ด VHDL และแสดงดังรูปที่ L2.1 โดยใช้ Project Location ชื่อ ch4v แล้ว กำหนด Project Name และ Source File ชื่อ ex4_1_lvf

การกำหนดขาสัญญาณต่างๆ ให้กับวงจรโดยจะใช้ปุ่มกด PB1-PB4 เป็นอินพุตและ LED L0-L3 เป็นเอ้าต์พุต กล่าวคือ

$$S1 = PB1 = \text{INPUT} = p44$$

$$Q1 = L3 = \text{OUTPUT} = p76$$

$$R1 = PB2 = \text{INPUT} = p46$$

$$Q1_{\text{bar}} = L2 = \text{OUTPUT} = p69$$

$$S2 = PB3 = \text{INPUT} = p47$$

$$Q2 = L1 = \text{OUTPUT} = p77$$

$$R2 = PB4 = \text{INPUT} = p50$$

$$Q2_{\text{bar}} = L0 = \text{OUTPUT} = p70$$

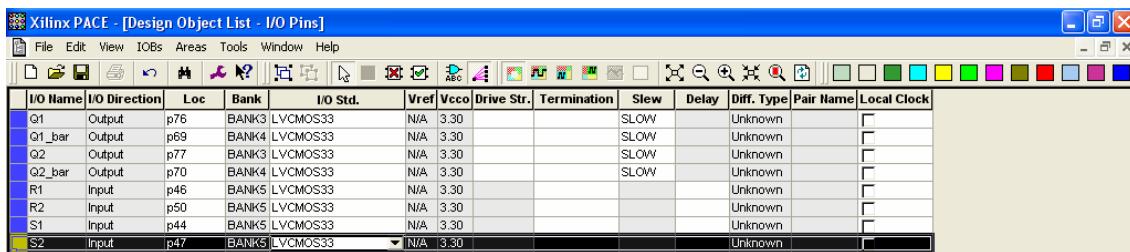
```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_1_lvf is
6     Port ( S1,R1,S2,R2 : in STD_LOGIC;
7             Q1,Q1_bar,Q2,Q2_bar : buffer STD_LOGIC);
8 end ex4_1_lvf;
9
10 architecture Behavioral of ex4_1_lvf is
11 begin
12     -----NAND GATE LATCH-----
13     Q1 <= S1 nand Q1_bar;
14     Q1_bar <= R1 nand Q1;
15     -----NOR GATE LATCH-----
16     Q2_bar <= S2 nor Q2;
17     Q2 <= R2 nor Q2_bar;
18 end Behavioral;

```

รูปที่ L2.1 โค้ดของวงจร NAND Gate latch และ NOR Gate latch

พิมพ์ใน Assign Package Pins สรุปดังรูปที่ L2.2



รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป FPGA แล้วให้ทดสอบกับปุ่ม PB1–PB4 และไฟสังเกตคุณลักษณะ LED L0-L3 ว่าให้ผลลัพธ์ตามที่ต้องการหรือไม่ หากนั้นให้สังเกตคุณลักษณะที่เป็น “Invalid” ว่าเป็นอย่างไร แล้วทำการบันทึกผลการทดลอง

แม้ว่า Xilinx Synthesis Tool หรือ XST จะรองรับโหมด Buffer แต่ซอฟต์แวร์ทุก XST แนะนำให้หลีกเลี่ยงการใช้โหมด Buffer ดังนั้นเราจะเขียนโค้ดใหม่ได้ดังรูปที่ L2.3 จากนั้นให้ทำการทดลองซ้ำ

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_1_lvf is
6     Port ( S1,R1,S2,R2 : in STD_LOGIC;
7             Q1,Q1_bar,Q2,Q2_bar : out STD_LOGIC);
8 end ex4_1_lvf;
9
10 architecture Behavioral of ex4_1_lvf is
11     signal Q1T,Q1T_bar,Q2T,Q2T_bar : STD_LOGIC;
12 begin
13     -----NAND GATE LATCH-----
14     Q1T <= S1 nand Q1T_bar;
15     Q1T_bar <= R1 nand Q1T;
16     Q1 <= Q1T;
17     Q1_bar <= Q1T_bar;
18     -----NOR GATE LATCH-----
19     Q2T_bar <= S2 nor Q2T;
20     Q2T <= R2 nor Q2T_bar;
21     Q2 <= Q2T;
22     Q2_bar <= Q2T_bar;
23 end Behavioral;

```

L2.3 โค้ด VHDL ที่เขียนโดยปราศจากการใช้ Signal

4.2 ฟลิปฟลอป

ฟลิปฟลอป (Flip-Flop) เป็นหน่วยความจำ (Memory element) ที่อาจประกอบด้วยແລຕ້ໜ່າຍตัว โดยที่ເອົ້າພຸດຂອງ ฟลิปฟลอปจะเปลี่ยนสถานะກີ່ຕ່ອມເມື່ອມີການທຽບດ້ວຍຂອນ (Edge triggered) ຂອງສັນຍາມນາພິກາ (Clock) ເທົ່ານີ້

1) D Flip-Flop

D Flip-Flop ແຕ່ລະຫົນດັບດັດຕັ້ງຕາຮາງຄວາມຈິງແລະ ໂກັດ VHDL ໃນຮູບທີ 4.4 ລຶ່ງຮູບທີ 4.6 ໂດຍ C ເປັນສັນຍາມນາພິກາ (Clock) ທີ່ທຽບດ້ວຍຂອນນວກ (ຂອນບໍ່ນ້ອງ Positive edge-triggered) ໂກັດທີ່ເປີຍນັດຮູບທີ 4.4b) ແລະຮູບທີ 4.4c) ຈະໄຟພລັພີ້ ດຽວຕາຮາງຄວາມຈິງໃນຮູບທີ 4.4a) ແຕ່ຜົງວຈຣທີ່ໄດ້ຈາກການສັງເຄຣະໜ້າແຕກຕ່າງກັນດັ່ງໃນຮູບທີ 4.4d) ແລະຮູບທີ 4.4e) ແລະພລກາສັງເຄຣະໜ້າຈະຈຳກັດຕັ້ງຕັ້ງມັນແນ້ຈະໃຫ້ໂກັດ VHDL ເດີວກັນພຣະໂຄຮງສ້າງກາຍໃນ CPLD ແລະ FPGA ແຕກຕ່າງກັນ ບໍ່ມີຕົວຢ່າງ D Flip-Flop ທີ່ທຽບດ້ວຍຂອບລົງກີ່ໃຫ້ແກ້ໄກ້ໂກັດນຽກທີ່ 14 ໃນຮູບທີ 4.4b) ເປັນ C'event and C = '0'

Input		Output		Remark
D	C	Q	Q_bar	
0	↑	0	1	Reset
1	↑	1	0	Set

4.4a) ຕາຮາງຄວາມຈິງຂອງ D Flip-Flop

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity D_FF is
6   Port ( C,D : in STD_LOGIC;
7         Q,Q_bar : out STD_LOGIC);
8 end D_FF;
9
10 architecture Behavioral of D_FF is
11 begin
12 process(c)
13 begin
14   if (C'event and C='1') then Q <= D; Q_bar <= not D;
15   end if;
16 end process;
17 end Behavioral;

```

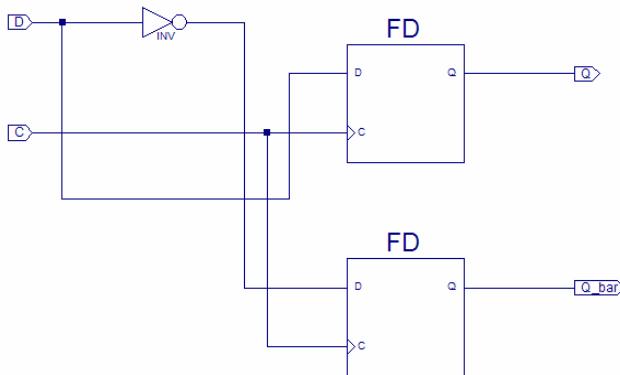
4.4b) ໂກັດ VHDL ຂອງ D Flip-Flop ທີ່ເປີຍນັດວິທີທີ່ 1

```

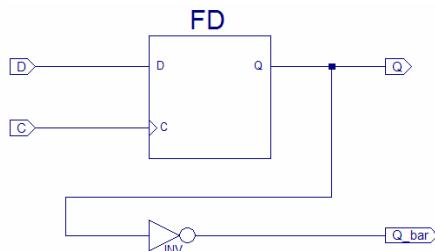
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity D_FF is
6   Port ( C,D : in STD_LOGIC;
7         Q : buffer STD_LOGIC;
8         Q_bar : out STD_LOGIC);
9 end D_FF;
10
11 architecture Behavioral of D_FF is
12 begin
13 process(c)
14 begin
15   if (C'event and C='1') then Q <= D;
16   end if;
17 end process;
18   Q_bar <= not Q;
19 end Behavioral;

```

4.4c) ໂກັດ VHDL ຂອງ D Flip-Flop ທີ່ເປີຍນັດວິທີທີ່ 2



4.4d) ผลการสังเคราะห์จากโค้ดในรูปที่ 4.4b) เมื่อสังเคราะห์โดยใช้ CPLD (View RTL schematic)



4.4e) ผลการสังเคราะห์จากโค้ดในรูปที่ 4.4c) เมื่อสังเคราะห์โดยใช้ CPLD (View RTL schematic)

รูปที่ 4.4 โค้ด VHDL และผังวงจรของ D Flip-Flop แบบทริกค้ายของขึ้น

หลักการทำงาน D Flip-Flop แบบอะซิงไครอนัสเคลียร์ (D Flip-Flop with asynchronous clear) แสดงดังตารางความจริง ในรูปที่ 4.5a) โดยที่ “-” คือ Don't care และโค้ด VHDL ของ D Flip-Flop แสดงดังรูปที่ 4.5b)

Input			Output	Remark
CLR	D	C	Q	
1	-	-	0	Clear
0	0	↑	0	Reset
0	1	↑	1	Set

4.5a) ตารางความจริงของ D Flip-Flop แบบอะซิงไครอนัสเคลียร์

```

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;
```

4.5b) โค้ด VHDL ของ D Flip-Flop แบบอะซิงไครอนัสเคลียร์

รูปที่ 4.5 โค้ด VHDL และผังวงจร D Flip-Flop แบบอะซิงไครอนัสเคลียร์

หลักการทำงาน D Flip-Flop แบบซิงโครนัสรีเซต (D Flip-Flop with synchronous reset) แสดงดังตารางความจริงในรูปที่ 4.6a) และ โค้ด VHDL แสดงดังรูปที่ 4.6b) เมื่อ R (Reset) = '1' แล้วเอาต์พุต Q จะยังไม่รีเซตจนกว่าจะมีการทริกคั่งของขาขึ้นของสัญญาณนาฬิกา ซึ่งจะแตกต่างจาก D Flip-Flop แบบซิงโครนัสเคลียร์ที่เคลียร์ค่าเอาต์พุตทันทีเมื่อ CLR = '1'

Input			Output	Remark
R	D	C	Q	
1	-		0	Reset
0	0		0	Reset
0	1		1	Set

4.6a) ตารางความจริงของ D Flip-Flop แบบซิงโครนัสรีเซต

```

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;

```

4.6b) โค้ด VHDL ของ D Flip-Flop แบบซิงโครนัสรีเซต

รูปที่ 4.6 โค้ด VHDL และผังวงจร D Flip-Flop แบบซิงโครนัสรีเซตที่ทริกคั่งของขาขึ้น

2) JK Flip-Flop

JK Flip-Flop แบบอะซิงโครนัสเคลียร์ (JK Flip-Flop with asynchronous clear) มีตารางความจริงแสดงดังรูปที่ 4.7a) ซึ่งมีโค้ดแสดงดังรูปที่ 4.7b) และรูปที่ 4.7c) โดยในการเขียนโค้ดนี้เราจะประกาศใช้ Signal เพื่อหลีกเลี่ยงการใช้ห้อมด Buffer ที่เกิดเนื่องจากเอาต์พุตอ่านค่ากลับ คือ $Q \leq Q$; และ $Q \leq \text{not } Q$;

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

4.7a) ตารางความจริงของ JK Flip-Flop แบบอะซิงโครนัสเคลียร์

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity JKFF is
6     Port ( C,CLR,J,K: in STD_LOGIC;
7             Q : out STD_LOGIC);
8 end JKFF;
9
10 architecture Behavioral of JKFF is
11     signal JK : STD_LOGIC_VECTOR(1 downto 0);
12     signal QT : STD_LOGIC;
13 begin
14     JK <= (J&K); -- Concatenation
15     process(C,CLR)
16     begin
17         if CLR='1' then QT <= '0';
18         elsif (C'event and C='1') then
19             if JK="00" then QT <= QT;
20             elsif JK="01" then QT <= '0';
21             elsif JK="10" then QT <= '1';
22             elsif JK="11" then QT <= not QT;
23             end if;
24         end if;
25     end process;
26     Q <= QT;
27 end Behavioral;

```

4.7b) โค้ด VHDL ของ JK Flip-Flop ที่เขียนโดยใช้คำสั่ง if

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity JKFF is
6     Port ( C,CLR,J,K: in STD_LOGIC;
7             Q : out STD_LOGIC);
8 end JKFF;
9
10 architecture Behavioral of JKFF is
11     signal JK : STD_LOGIC_VECTOR(1 downto 0);
12     signal QT : STD_LOGIC;
13 begin
14     JK <= (J&K); -- Concatenation
15     process(C,CLR)
16     begin
17         if CLR='1' then QT <= '0';
18         elsif (C'event and C='1') then
19             case JK is
20                 when "00" => QT <= QT;
21                 when "01" => QT <= '0';
22                 when "10" => QT <= '1';
23                 when "11" => QT <= not QT;
24                 when others => null; --No action
25             end case;
26         end if;
27     end process;
28     Q <= QT;
29 end Behavioral;

```

4.7c) โค้ด VHDL ของ JK Flip-Flop ที่เขียนโดยใช้คำสั่ง case และใช้คำสั่ง null

รูปที่ 4.7 โค้ดของ JK Flip-Flop แบบอะซิงไครนัสเคเลียร์ที่เขียนโดยการประกาศใช้ Signal

ในทำนองเดียวกัน จากรูปที่ 4.7b) และรูปที่ 4.7c) เราสามารถเขียนโค้ดของ JK Flip-Flop แบบอะซิงไครนัสเคเลียร์ (JK Flip-Flop with asynchronous clear) โดยการประกาศใช้ Variable เพื่อหลีกเลี่ยงการใช้ใหม่ Buffer ที่เกิดเนื่องจากເອົາຕີ່ພຸດ ອໍານຄ່າກັບ ຄື້ອ Q <= Q; ແລະ Q <= not Q; ໄດ້ເຫັນກັນ ໂຄ້ດທີ່ໄດ້ແສດງດັ່ງຮູບທີ່ 4.8a) ແລະຮູບທີ່ 4.8b)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity JKFF is
6   Port ( C,CLR,J,K: in STD_LOGIC;
7         Q : out STD_LOGIC);
8 end JKFF;
9
10 architecture Behavioral of JKFF is
11 begin
12   process(C,CLR)
13     variable JK : STD_LOGIC_VECTOR(1 downto 0);
14     variable QT : STD_LOGIC;
15   begin
16     JK := (J&K); -- Concatenation
17     if CLR='1' then QT := '0';
18     elsif (C'event and C='1') then
19       if JK="00" then QT := QT;
20       elsif JK="01" then QT := '0';
21       elsif JK="10" then QT := '1';
22       elsif JK="11" then QT := not QT;
23       end if;
24     end if;
25     Q <= QT;
26   end process;
27 end Behavioral;

```

4.8a) โค้ด VHDL ของ JK Flip-Flop ที่เขียนโดยใช้คำสั่ง if แต่ประกาศใช้ Variable แทน Signal

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity JKFF is
6   Port ( C,CLR,J,K: in STD_LOGIC;
7         Q : out STD_LOGIC);
8 end JKFF;
9
10 architecture Behavioral of JKFF is
11 begin
12   process(C,CLR)
13     variable JK : STD_LOGIC_VECTOR(1 downto 0);
14     variable QT : STD_LOGIC;
15   begin
16     JK := (J&K); -- Concatenation
17     if CLR='1' then QT := '0';
18     elsif (C'event and C='1') then
19       case JK is
20         when "00" => QT := QT;
21         when "01" => QT := '0';
22         when "10" => QT := '1';
23         when "11" => QT := not QT;
24         when others => null;
25       end case;
26     end if;
27     Q <= QT;
28   end process;
29 end Behavioral;

```

4.8b) โค้ด VHDL ของ JK Flip-Flop ที่เขียนโดยใช้คำสั่ง case แต่ประกาศใช้ Variable แทน Signal

รูปที่ 4.8 โค้ดของ JK Flip-Flop แบบอะล็อกิคัลน็อตเกลียร์ที่เขียนโดยการประกาศใช้ Variable แทน Signal

ในทำนองเดียวกันที่สามารถออกแบบ JK Flip-Flop ที่ทริกตัวของขาลงได้ เช่นกัน โดยทำการแท็กไปโค้ดในรูปที่ 4.7 และรูปที่ 4.8 ทั้งหมดโดยแก้ไขโค้ดจาก C'event and C = '1' เป็น C'event and C = '0' แทน

3) T Flip-Flop

T Flip-Flop แบบอะซิงโครนัสเคลือบ มีตารางความจริงและตัวอย่างโค้ดดังรูปที่ 4.9a) ถึงรูปที่ 4.9c)

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

4.9a) ตารางความจริงของ T Flip-Flop

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity TFF is
6   Port ( C,CLR,T : in STD_LOGIC;
7         Q : out STD_LOGIC);
8 end TFF;
9
10 architecture Behavioral of TFF is
11   signal QT : STD_LOGIC;
12 begin
13   process(C,CLR)
14   begin
15     if CLR='1' then QT <= '0';
16     elsif (C'event and C='1') then
17       if T='0' then QT <= QT;
18       elsif T='1' then QT <= not QT;
19       end if;
20     end if;
21   end process;
22   Q <= QT;
23 end Behavioral;
```

4.9b) โค้ด VHDL ของ T Flip-Flop แบบอะซิงโครนัสเคลือบที่เขียนโดยใช้คำสั่ง if

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity TFF is
6   Port ( C,CLR,T : in STD_LOGIC;
7         Q : out STD_LOGIC);
8 end TFF;
9
10 architecture Behavioral of TFF is
11   signal QT : STD_LOGIC;
12 begin
13   process(C,CLR)
14   begin
15     if CLR='1' then QT <= '0';
16     elsif (C'event and C='1') then
17       case T is
18         when '0' => QT <= QT;
19         when '1' => QT <= not QT;
20         when others => null;
21       end case;
22     end if;
23   end process;
24   Q <= QT;
25 end Behavioral;
```

4.9c) โค้ด VHDL ของ T Flip-Flop แบบอะซิงโครนัสเคลือบที่เขียนโดยใช้คำสั่ง case

รูปที่ 4.9 โค้ด VHDL และผังวงจรของ T Flip-Flop แบบอะซิงโครนัสเคลือบที่ทริกด้วยขอบขาขึ้น

การทดลองที่ 4.2.1 D Flip-Flop แบบอะซิงโครนัสเกลียร์

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานของวงจร D Flip-Flop แบบอะซิงโครนัสเกลียร์
- 2) เพื่อสร้างวงจร D Flip-Flop แบบอะซิงโครนัสเกลียร์แล้วโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจร D Flip-Flop แบบอะซิงโครนัสเกลียร์ด้วย CPLD

สร้างไฟล์โดยใช้ Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_2_1vcx1 จากนั้น เขียนโค้ดของวงจร D Flip-Flop แบบอะซิงโครนัสเกลียร์ดังรูปที่ L1.1 โดยมี C เป็นขาสัญญาณนาฬิกา (Clock) ที่ทริกด้วยขอบขาลง และมี CLR เป็นขาเคลียร์ที่ทำงานแบบ Active low (จะเคลียร์เมื่อ CLR='0')

```

2 -----D-flip-flop with asynchronous clear-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity ex4_2_1vcx1 is
7     Port ( C,CLR,D : in STD_LOGIC;
8             Q,Q_bar : out STD_LOGIC);
9 end ex4_2_1vcx1;
10
11 architecture Behavioral of ex4_2_1vcx1 is
12 begin
13 process(C,CLR)
14 begin
15     if CLR='0' then
16         Q      <= '0';
17         Q_bar <= '1';
18     elsif (C'event and C='0') then
19         Q      <= D;
20         Q_bar <= not D;
21     end if;
22 end process;
23 end Behavioral;

```

รูปที่ L1.1 D Flip-Flop แบบอะซิงโครนัสเกลียร์

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB3 เป็นอินพุตและ LED1-LED2 เป็นเอาต์พุต กล่าวคือ

D = PB1 = INPUT = p39 CLR = PB3 = INPUT = p42 Q = LED1 = OUTPUT = p38

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

โดยพิมพ์ใน Assign Package Pins สรุปดังรูปที่ L1.2

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Stew	Globals
C	Input	p40	2	11		
CLR	Input	p42	2	14		
D	Input	p39	2	9		
Q	Output	p38	2	8	SLOW	
Q_bar	Output	p37	2	6	SLOW	

รูปที่ L1.2 Assign Package Pins

หลังจากโปรแกรมจะที่ต้องออกแบบลงชิป CPLD แล้วให้กดปุ่ม PB2 และให้สั่งเกตคูเพลที่ LED1-LED2 จากนั้นให้กดปุ่ม PB1 ค้างไว้แล้วกดปุ่ม PB2 และให้สั่งเกตคูเพลที่ LED1-LED2 อีกครั้ง และให้ปล่อยปุ่ม PB1 แล้วกดปุ่ม PB2 แล้วให้สั่งเกตคูเพลที่ LED1-LED2 ว่า LED1 ติดสว่างหรือไม่ จากนั้นให้กดปุ่ม PB3 แล้วให้สั่งเกตคูเพลที่ LED1-LED2 แล้วให้สรุปผลการทดลองห้องหมัดว่าเป็นไปตามทฤษฎีหรือไม่

2 สร้างวงจร D Flip-Flop แบบอะซิงโครนัสเคลียร์ด้วย FPGA

สร้างไฟล์โดยใช้ Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_2_1vf จากนั้นเขียนโค้ดของวงจร D Flip-Flop แบบอะซิงไกรอนัสเคลียร์ ดังรูปที่ L2.1 โดยมี C เป็นขาสัญญาณนาฬิกา (Clock) ที่ทริกคัวของบานาลง และมี CLR เป็นขาเคลียร์ที่ทำงานแบบ Active low (จะเคลียร์เมื่อ CLR= '0')

```

2 -----D-flip-flop with asynchronous clear-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity ex4_2_1vf is
7     Port ( C,CLR,D : in STD_LOGIC;
8             Q,Q_bar : out STD_LOGIC);
9 end ex4_2_1vf;
10
11 architecture Behavioral of ex4_2_1vf is
12 begin
13 process(C,CLR)
14 begin
15     if CLR='0' then
16         Q <= '0';
17         Q_bar <= '1';
18     elsif (C'event and C='0') then
19         Q <= D;
20         Q_bar <= not D;
21     end if;
22 end process;
23 end Behavioral;

```

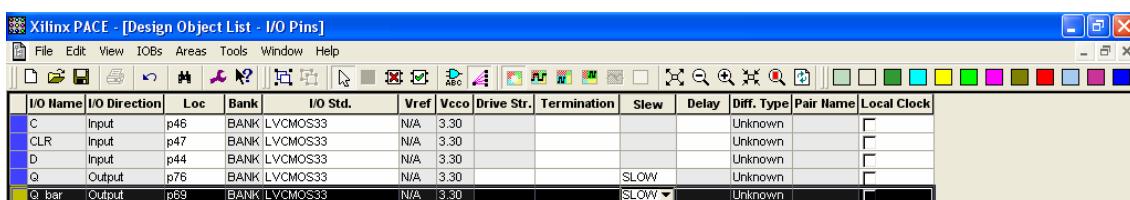
รูปที่ L2.1 โค๊ด D Flip-Flop แบบอะซิง โครนัสเคลียร์

การกำหนดขาสัมภាពต่างๆจะใช้ปุ่มกด PB1-PB3 เป็นอินพตและ LED L2-L3 เป็นเอาต์พต กล่าวคือ

D = PB1 = INPUT = p44 CLR = PB3 = INPUT = p47 Q = L3 = OUTPUT = p76

C = PB2 = INPUT = p46 Q bar = L2 = OUTPUT = p69

โดยพิมพ์ใน Assign Package Pins สรุปดังรูปที่ L2.2



ຈັດຕີ L2.2 Assign Package Pins

หลังจากโปรแกรมจะที่ออกแบบลงชิป FPGA แล้วให้กดปุ่ม PB2 และให้สังเกตคุณลักษณะที่ LED L2-L3 จากนั้นให้กดปุ่ม PB1 ค้างไว้แล้วกดปุ่ม PB2 และให้สังเกตคุณลักษณะที่ LED L2-L3 อีกครั้ง และให้ปล่อยปุ่ม PB1 แล้วกดปุ่ม PB2 แล้วให้สังเกตคุณลักษณะที่ LED L2-L3 ว่า LED L3 ติดสว่างหรือไม่ หากนั้นให้กดปุ่ม PB3 และให้สังเกตคุณลักษณะที่ LED L2-L3 และให้สรุปผลการทดลองทั้งหมดว่าเป็นไปตามทฤษฎีหรือไม่

การทดลองที่ 4.2.2 D Flip-Flop แบบซิงโครนัสรีเซต

วัตถุประสงค์

- เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานของวงจร D Flip-Flop แบบซิงก์ไครน์สตรีเชต
 - เพื่อสร้างวงจร D Flip-Flop แบบซิงก์ไครน์สตรีเชตแล้วโปรแกรมลงชิป CPLD และ FPGA

ឧបករណ៍ទិន្នន័យ

ນອრົດ CPLD Explorer XC9572XL ທີ່ຈະ CPLD Explorer XC9572 ແລະ FPGA Discovery-III XC3S200F ທີ່ຈະ FPGA Discovery-III XC3S200F4

1 สร้างวงจร D Flip-Flop แบบซิงโครนัสรีเซตด้วย CPLD

สร้างไฟล์โดยใช้ Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_2_vcxl จากนั้น เขียนโค้ดของวงจร D Flip-Flop แบบซิงโครนัสเรซต์ดังรูปที่ L1.1 โดยมี C เป็นขาสัญญาณนาฬิกา (Clock) ที่ทริกค้ายของขาลง และมี RESET เป็นขาที่ทำงานแบบ Active low (จะรีเซ็ตเมื่อ RESET = '0')

```

2 -----D-flip-flop with synchronous reset-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity ex4_2_2vcx1 is
7     Port ( C,RESET,D : in STD_LOGIC;
8             Q,Q_bar : out STD_LOGIC);
9 end ex4_2_2vcx1;
10
11 architecture Behavioral of ex4_2_2vcx1 is
12 begin
13 process(C)
14 begin
15     if (C'event and C='0') then
16         if      RESET='0' then Q <= '0'; Q_bar <= '1';
17         else    Q <= D; Q_bar <= not D;
18         end if;
19     end if;
20 end process;
21 end Behavioral;

```

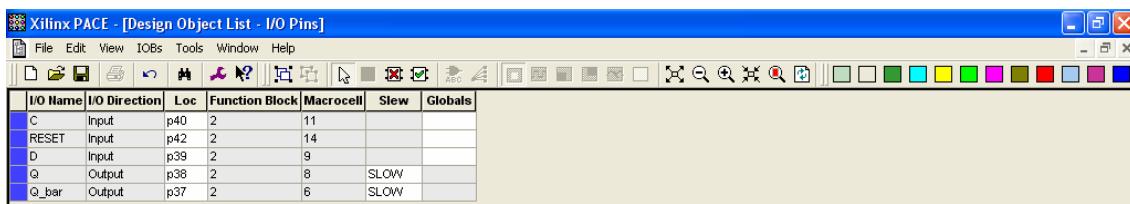
รูปที่ L1.1 D Flip-Flop แบบซิงโครนัสรีเซต

การกำหนดขาสัมภาระต่างๆ จะใช้ปุ่มกด PB1-PB3 เป็นอินพุตและ LED1-LED2 เป็นเอาต์พุต กล่าวคือ

D = PB1 = INPUT = p39 RESET = PB3 = INPUT = p42 Q = LED1 = OUTPUT = p38

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

โดยพิมพ์ใน Assign Package Pins สรุปดังรูปที่ L1.2



រូបថត L1.2 Assign Package Pins

หลังจากโปรแกรมจะรื่นที่ออกแบบลงชิป CPLD แล้วให้กดปุ่ม PB2 และให้สังเกตคุณลักษณะ LED1-LED2 จากนั้นให้กดปุ่ม PB1 ค้างไว้แล้วกดปุ่ม PB2 และให้สังเกตคุณลักษณะ LED1-LED2 อีกครั้ง และให้ปล่อยปุ่ม PB1 แล้วกดปุ่ม PB2 แล้วให้สังเกตคุณลักษณะ LED1-LED2 ว่า LED1 ติดสว่างหรือไม่ ให้กดปุ่ม PB3 แล้วให้สังเกตคุณลักษณะ LED1-LED2 จากนั้นให้กดปุ่ม PB3 ค้างไว้แล้วกดปุ่ม PB2 และให้สังเกตคุณลักษณะ LED1-LED2 ให้สรุปผลการทดลองทั้งหมดว่าเป็นไปตามทฤษฎีหรือไม่

2 สร้างวงจร D Flip-Flop แบบซิงโครนัสรีเซ็ตด้วย FPGA

สร้างไฟล์โดยใช้ Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_2_2vf จากนั้นเขียนโค้ดของวงจร D Flip-Flop แบบซิงโครนัสรีเซ็ตดังรูปที่ L2.1 โดยมี C เป็นขาสัญญาณนาฬิกา (Clock) ที่ทริกด้วยขอบข้างและมี RESET เป็นขาที่ทำงานแบบ Active low (จะรีเซ็ตเมื่อ RESET= '0')

```

2 -----D-flip-flop with synchronous reset-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity ex4_2_2vf is
7     Port ( C,RESET,D : in STD_LOGIC;
8             Q,Q_bar : out STD_LOGIC);
9 end ex4_2_2vf;
10
11 architecture Behavioral of ex4_2_2vf is
12 begin
13 process(C)
14 begin
15     if (C'event and C='0') then
16         if      RESET='0' then Q <= '0'; Q_bar <= '1';
17         else   Q <= D; Q_bar <= not D;
18         end if;
19     end if;
20 end process;
21 end Behavioral;

```

รูปที่ L2.1 โค้ด D Flip-Flop แบบซิงโครนัสรีเซ็ต

การกำหนดขาสัญญาณต่างๆจะใช้ปุ่มกด PB1-PB3 เป็นอินพุตและ LED L2-L3 เป็นเอาต์พุต กล่าวคือ

$$D = PB1 = \text{INPUT} = p44 \quad \text{RESET} = PB3 = \text{INPUT} = p47 \quad Q = L3 = \text{OUTPUT} = p76$$

$$C = PB2 = \text{INPUT} = p46 \quad Q_{\text{bar}} = L2 = \text{OUTPUT} = p69$$

โดยพิมพ์ใน Assign Package Pins สรุปดังรูปที่ L2.2

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
C	Input	p46	BANK5	LVCMS33	N/A	3.30					Unknown		
D	Input	p44	BANK5	LVCMS33	N/A	3.30					Unknown		
Q	Output	p76	BANK3	LVCMS33	N/A	3.30		SLOW			Unknown		
Q_bar	Output	p69	BANK4	LVCMS33	N/A	3.30		SLOW			Unknown		
RESET	Input	p47	BANK5	LVCMS33	N/A	3.30					Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมจะรื่นที่ออกแบบลงชิป FPGA แล้วให้กดปุ่ม PB2 และให้สังเกตคุณลักษณะ LED L2-L3 จากนั้นให้กดปุ่ม PB1 ค้างไว้แล้วกดปุ่ม PB2 และให้สังเกตคุณลักษณะ LED L2-L3 อีกครั้ง และให้ปล่อยปุ่ม PB1 แล้วกดปุ่ม PB2 แล้วให้สังเกตคุณลักษณะ LED L2-L3 ว่า LED L3 ติดสว่างหรือไม่ ให้กดปุ่ม PB3 แล้วให้สังเกตคุณลักษณะ LED L2-L3 จากนั้นให้กดปุ่ม PB3 ค้างไว้แล้วกดปุ่ม PB2 และให้สังเกตคุณลักษณะ LED L2-L3 ให้สรุปผลการทดลองทั้งหมดว่าเป็นไปตามทฤษฎีหรือไม่

การทดลองที่ 4.2.3 วงจรดีเบาเซอร์อย่างง่าย

วัตถุประสงค์

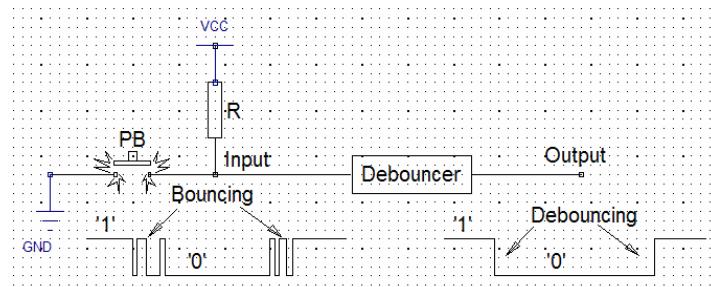
- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานของวงจรดีเบาเซอร์ (Debouncer) อย่างง่าย
- 2) เพื่อสร้างวงจรดีเบาเซอร์อย่างง่ายแล้วโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

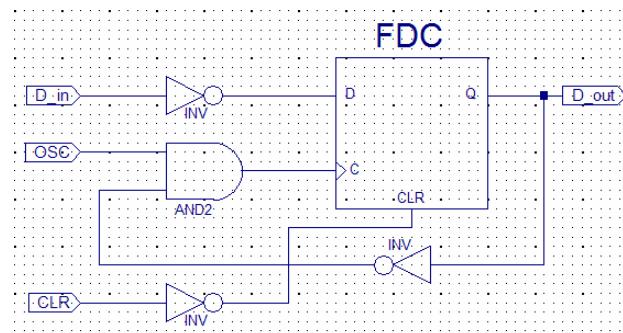
1 สร้างวงจรดีเบาเซอร์อย่างง่ายด้วย CPLD

การกดคีย์บอร์ดหรือปุ่มกดเพื่อสร้างสัญญาณนาฬิกาครั้งละ 1 พลั๊ส (Pulse) อาจจะทำไม่ได้เนื่องจากการกดคีย์บอร์ดในช่วงเริ่มต้นหน้าสัมผัสอาจจะขังแตะกันไม่สนิท (Bouncing) ทำให้สัญญาณเอาต์พุตที่ได้ไม่แน่นอนจนกว่าหน้าสัมผัสจะแตกกันสนิท และในขณะที่ปล่อยคีย์บอร์ดก็จะเกิดลักษณะเดียวกัน คือ หน้าสัมผัสจะขังไม่แยกจากกันสนิท ซึ่งสัญญาณที่ได้แสดงดังตัวอย่างในรูปที่ L1.1 โดยจะให้สัญญาณนาฬิกาหลายพลั๊สทั้งๆ ที่มีการกดคีย์บอร์ดเพียงครั้งเดียว โดยปกติแล้วเวลาเกิดเบาซ์ (Bouncing) ขนาดก้อนหรือปล่อยคีย์บอร์ดจะไม่เกิน 20 มิลลิวินาที ปัญหาการเกิดเบาซ์นี้สามารถแก้ไขได้โดยใช้วงจรดีเบาเซอร์ (Debouncer) หรือวงจรโมโนสเตเบิล (Monostable) เพื่อทำให้ได้พลั๊สเอาต์พุตออกมากครั้งละ 1 พลั๊สแสดงดังในรูปที่ L1.1



รูปที่ L1.1 ตัวอย่างสัญญาณอินพุตและเอาต์พุตของวงจรดีเบาเซอร์ขั้นตอนคีย์บอร์ดหรือปุ่มกด

วงจรดีเบาเซอร์อย่างง่ายที่ใช้ D Flip-Flop แบบอะซิงไโกรานส์เคลลีบร์ (FDC) มีผังวงจรแสดงดังรูปที่ L1.2



รูปที่ L1.2 วงจรดีเบาเซอร์อย่างง่ายที่อินพุตเป็นแบบ Active low และเอาต์พุตเป็นแบบ Active High

จากการจดจำรูปที่ L1.2 เมื่อไม่มีการกดปุ่ม $D_{in} = '1'$ ($D = '0'$) นั้น D Flip-Flop จะถูกทริกเกอร์ด้วย Clock ความถี่สูงจาก OSC ผ่านทางแอนด์เกตได้ตลอดเวลา เพราะว่า $D_{out} = '0'$ จึงทำให้แอนด์เกตอึกษาที่ต่อจากอินเวอร์เตอร์

เป็นลอกอิก ‘1’ แต่เมื่อมีการกดปุ่ม D_in = ‘0’ (D = ‘1’) จะทำให้ D Flip-Flop ถูกทริกและเอาต์พุต D_out = ‘1’ ดังนั้นแอนด์เกต อิกขาที่ต่อจากอินเวอร์เตอร์จึงเป็นลอกอิก ‘0’ ทำให้ Clock จาก OSC ไม่สามารถผ่านแอนด์เกตไปทริก D Flip-Flop ได้อีก ดังนั้น เอาต์พุตของ D Flip-Flop จึงถูก Latch ค่าค้างไว้คือ D_out = ‘1’ โดยที่พัลส์ลูกต่อๆ ไปจาก D_in ไม่สามารถผ่านไปที่เอาต์พุต ของ D Flip-Flop ได้จนกว่าจะกดปุ่ม CLR ดังนั้นการสร้างพัลส์เอาต์พุต D_out 1 พัลส์จะต้องกดปุ่ม D_in 1 ครั้งและกดปุ่ม เคลียร์ CLR อีก 1 ครั้ง วงจรดีเบาเซอร์นี้จะทำงานแบบ Manual ได้ดีมาก แต่ยังไงก็ตามวงจรดีเบาเซอร์นี้จะมีประสิทธิภาพ ต่ำมากเมื่อทำงานโดยอัตโนมัติด้วยการป้อนความถี่ประมาณ 2-4 Hz ที่ขา CLR ซึ่งทำให้งานนี้เคลียร์เอาต์พุตทุกๆ 0.25-0.5 วินาที ดังนั้นถ้ากดเร็วไปก็อาจจะไม่ให้พัลส์ออกมานะ แต่ถ้ากดช้าไปก็อาจให้พัลส์เกิน 1 พัลส์ หรือถ้ากดปุ่มค้างไว้จะให้พัลส์ ต่อเนื่องหรือ Clock ความถี่ประมาณ 2-4 Hz เช่นเดียวกับที่ขา CLR ซึ่งการออกแบบวงจรดีเบาเซอร์ประสิทธิภาพสูงจะอธิบาย ในข้อ 4.5 และข้อ 4.6 (วิธี Finite state machine)

จากหลักการที่กล่าวมาแล้ว เราสามารถเขียน โค้ดของวงจรดีเบาเซอร์ได้ดังรูปที่ L1.3 โดยเขียนไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_2_3vcx1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_2_3vcx1 is
6     Port ( OSC,D_in,CLR : in STD_LOGIC;
7             D_out : out STD_LOGIC);
8 end ex4_2_3vcx1;
9
10 architecture Behavioral of ex4_2_3vcx1 is
11     signal QT,C_DB : STD_LOGIC;
12 begin
13     C_DB <= not QT and OSC;
14 process(OSC,CLR)
15     begin
16         if CLR='0' then QT <= '0';
17         elsif C_DB'event and C_DB='1' then
18             if D_in ='0' then QT <= '1';
19             end if;
20         end if;
21     end process;
22     D_out <= QT;
23 end Behavioral;
```

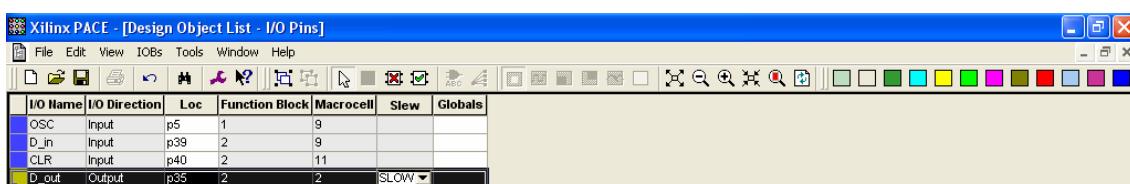
รูปที่ L1.3 โค้ดของวงจรดีเบาเซอร์ย่างง่ายสำหรับนักศึกษาและเคลียร์แบบ Active low

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

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

CLR = PB2 = INPUT = p40 OSC = OSC = INPUT = p5

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



รูปที่ L1.4 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิพ CPLD และให้กดปุ่ม PB1 และ PB2 ลับกันไปเรื่อยๆแล้วให้สังเกตคูลท์ที่ LED4 งานนี้ให้สรุปผลการทดลองว่าเป็นไปตามทฤษฎีหรือไม่

2 สร้างวงจรดีเบนเชอร์อย่างง่ายด้วย FPGA

ให้ทำการเข้าใจหลักการทำงานของวงจรดีเบนเชอร์ (Debouncer) หรือโมโนสเตเบิล (Monostable) ในข้อ 1 จากนั้นจึงเขียนโค้ดของวงจรดีเบนเชอร์ได้ดังรูปที่ L2.1 โดยเขียนไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_2_3vf

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_2_3vf is
6     Port ( OSC,D_in,CLR : in STD_LOGIC;
7             D_out : out STD_LOGIC);
8 end ex4_2_3vf;
9
10 architecture Behavioral of ex4_2_3vf is
11     signal QT,C_DB : STD_LOGIC;
12 begin
13     C_DB <= not QT and OSC;
14 process(OSC,CLR)
15 begin
16     if CLR='0' then QT <= '0';
17     elsif C_DB'event and C_DB='1' then
18         if D_in ='0' then QT <= '1';
19         end if;
20     end if;
21 end process;
22 D_out <= QT;
23 end Behavioral;

```

รูปที่ L2.1 โค้ดของวงจรดีเบนเชอร์อย่างง่ายสำหรับปุ่มกดและเคลื่ร์แบบ Active low

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25MHz ปุ่มกด PB1-PB2 เป็นอินพุต และ LED L0 เป็นเออต์พุต กล่าวคือ

D_in = PB1 = INPUT = p44 Q = L0 = OUTPUT = p70
 CLR = PB2 = INPUT = p46 OSC = OSC = INPUT = p127

โดยพิมพ์ใน Assign Package Pins สรุปดังรูปที่ L2.2

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLR	Input	p46	BANK	LVCMS33	N/A	3.30					Unknown		
D_in	Input	p44	BANK	LVCMS33	N/A	3.30					Unknown		
D_out	Output	p70	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
OSC	Input	p127	BANK	LVCMS33	N/A	3.30					Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิป FPGA และให้กดปุ่ม PB1 และ PB2 ลับกันไปเรื่อยๆแล้วให้สังเกตคุณผลที่ LED L0 จากนั้นให้สรุปผลการทดลองว่าเป็นไปตามทฤษฎีหรือไม่

การทดลองที่ 4.2.4 JK Flip-Flop และ T Flip-Flop แบบอะซิงโกรนัสเคลียร์

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับวงจร JK Flip-Flop และ T Flip-Flop แบบอะซิงโกรนัสเคลียร์และวงจรดีเบาเซอร์
- 2) เพื่อสร้าง JK Flip-Flop และวงจรดีเบาเซอร์ด้วยโค้ด VHDL และโปรแกรม CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจร JK Flip-Flop แบบอะซิงโกรนัสเคลียร์ด้วย CPLD

จากการทดลองที่ 4.2.3 ให้สร้างไฟล์โค้ดวงจรดีเบาเซอร์ดังรูปที่ L1.1 เพื่อทำเป็น Component ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vcx1_DEBOUNCER (ชื่อโค้ดนี้ต้องยกตรวจสอบความถูกต้องແລ້ວ)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ch4vcx1_DEBOUNCER is
6     Port ( OSC,D_in,CLR : in STD_LOGIC;
7             D_out : out STD_LOGIC);
8 end ch4vcx1_DEBOUNCER;
9
10 architecture Behavioral of ch4vcx1_DEBOUNCER is
11     signal QT,C_DB : STD_LOGIC;
12 begin
13     C_DB <= not QT and OSC;
14 process(OSC,CLR)
15 begin
16     if CLR='0' then QT <= '0';
17     elsif C_DB'event and C_DB='1' then
18         if D_in ='0' then QT <= '1';
19         end if;
20     end if;
21 end process;
22     D_out <= QT;
23 end Behavioral;

```

รูปที่ L1.1 โค้ดของวงจรดีเบาเซอร์ย่างง่ายสำหรับนิ่มกดและเคลียร์แบบ Active low

จากนั้นเขียนไฟล์โค้ด JK Flip-Flop แบบอะซิงโกรนัสเคลียร์ที่รวม Component ของวงจรดีเบาเซอร์ไว้แล้วแสดงดังในรูปที่ L1.2 ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_2_4vcx1 ซึ่งขั้นตอน Add Source ของไฟล์ Component เข้าไปในไฟล์ Project สามารถอ่านได้จากข้อ 2.18.2.1 ของบทที่ 2 (ดูรูปที่ E3.13 ถึงรูปที่ E3.20)

```

2 -----Main code:JK Flip-Flop with DEBOUNCER-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity ex4_2_4vcx1 is
7     Port ( OSC,Din,CLR_DB,J,K,CLR : in STD_LOGIC;
8             Q,Q_bar : out STD_LOGIC);
9 end ex4_2_4vcx1;
10
11 architecture Behavioral of ex4_2_4vcx1 is

```

(ต่อ)

```

12 component ch4vcx1_DEBOUNCER
13   Port ( OSC,D_in,CLR : in STD_LOGIC;
14           D_out : out STD_LOGIC);
15 end component;
16 signal C_DB,C,QT : STD_LOGIC;
17 signal JK : STD_LOGIC_VECTOR(1 downto 0);
18 begin
19   -----
20   DEBOUNCE_JKFF : ch4vcx1_DEBOUNCER port map ( OSC  => OSC,
21                                                 D_in => Din,
22                                                 CLR  => CLR_DB,
23                                                 D_out=> C      );
24   -----
25   JK <= (J&K); -- Concatenation
26 process(C,CLR)
27 begin
28   if CLR='1' then QT <= '0';
29   elsif (C'event and C='1') then
30     if JK="00" then QT <= QT;
31     elsif JK="01" then QT <= '0';
32     elsif JK="10" then QT <= '1';
33     elsif JK="11" then QT <= not QT;
34     end if;
35   end if;
36 end process;
37   Q <= QT;  Q_bar <= not QT;
38 end Behavioral;

```

รูปที่ L1.2 โค้ด JK Flip-Flop ที่รวมวงจรคีเบาเซอร์ในรูปของ Component ไว้แล้ว

การกำหนดขาสัญญาณต่างๆ ให้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB4 เป็นอินพุต มี LED3-LED4 เป็นเอาต์พุตกล่าวคือ

Din = PB1 = INPUT = p39

Q = LED3 = OUTPUT = p36

CLR_DB = PB2 = INPUT = p40

Q_bar = LED4 = OUTPUT = p35

J = PB3 (Slide SW1) = INPUT = p42

OSC = OSC = INPUT = p5

K = PB4 (Slide SW2) = INPUT = p43

CLR = PB5 (Slide SW3) = INPUT = p44

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]						
I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
OSC	Input	p5	1	9		
Din	Input	p39	2	9		
CLR_DB	Input	p40	2	11		
J	Input	p42	2	14		
K	Input	p43	2	15		
CLR	Input	p44	2	17		
Q	Output	p36	2	5	SLOW	
Q_bar	Output				SLOW	

รูปที่ L1.3 Assign Package Pins

หลังจากโปรแกรมวงจรลง CPLD แล้วให้เลื่อน Slide SW3 ไปที่ ON (CLR = '0') จากนั้นเซตค่า J และ K เป็นค่าต่างๆ เริ่มจาก J = '0' และ K = '0' ไปจนกระทั่ง J = '1' และ K = '1' โดยใช้ Slide SW1 และ Slide SW2 (Slide SW1 ON แล้ว J = '0', OFF แล้ว J = '1' และ Slide SW2 ON แล้ว K = '0', OFF แล้ว K = '1') แล้วกดปุ่ม PB1 และปุ่ม PB2 ลับกันเพื่อสร้าง Clock พร้อมกับให้สังเกตคุณ LED3 และ LED4 ในแต่ละเงื่อนไขว่าเป็นตามทฤษฎีหรือไม่ กดปุ่ม PB1 และปุ่ม PB2 ลับกันจนกระทั่ง LED3 ติดสว่างแล้วเลื่อน Slide SW3 ไปที่ตำแหน่ง OFF (CLR = '1') พร้อมกับให้สังเกตคุณ LED3 ว่าดับเป็นตามทฤษฎีหรือไม่ (กรณีที่เป็น T Flip-Flop ให้เซต J = K = '1') เสร็จแล้วจึงบันทึกผลการทดลอง

2 สร้างวงจร JK Flip-Flop แบบอะซิงโกรนสเกลียร์ด้วย FPGA

จากการทดลองที่ 4.2.3 ให้สร้างไฟล์โค้ดวงจรดีเบาเซอร์ดังรูปที่ L2.1 เพื่อทำเป็น Component ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vf_DEBOUNCER (ซึ่งโค้ดนี้ต้องถูกตรวจสอบความถูกต้องแล้ว)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ch4vf_DEBOUNCER is
6     Port ( OSC,D_in,CLR : in STD_LOGIC;
7             D_out : out STD_LOGIC);
8 end ch4vf_DEBOUNCER;
9
10 architecture Behavioral of ch4vf_DEBOUNCER is
11     signal QT,C_DB : STD_LOGIC;
12 begin
13     C_DB <= not QT and OSC;
14 process(OSC,CLR)
15 begin
16     if CLR='0' then QT <= '0';
17     elsif C_DB'event and C_DB='1' then
18         if D_in ='0' then QT <= '1';
19         end if;
20     end if;
21 end process;
22 D_out <= QT;
23 end Behavioral;
```

รูปที่ L2.1 โค้ดของวงจรดีเบาเซอร์อย่างง่ายสำหรับปุ่มกดและเคลียร์แบบ Active low

จากนั้นเขียนไฟล์โค้ด JK Flip-Flop แบบอะซิงโกรนสเกลียร์ที่รวม Component ของวงจรดีเบาเซอร์ไว้แล้วแสดงดังในรูปที่ L2.2 ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_2_4vf ซึ่งขั้นตอน Add Source ของไฟล์ Component เข้าไปในไฟล์ Project ของ FPGA นั้นจะเหมือนกับ CPLD ซึ่งอธิบายในข้อ 2.18.2.1 ของบทที่ 2 (ครุภัติ E3.13 ถึงรูปที่ E3.20) และในข้อ 2.19 ของบทที่ 2

```

2 -----Main code:JK Flip-Flop with DEBOUNCER-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity ex4_2_4vf is
7     Port ( OSC,Din,CLR_DB,J,K,CLR : in STD_LOGIC;
8             Q,Q_bar : out STD_LOGIC);
9 end ex4_2_4vf;
10
11 architecture Behavioral of ex4_2_4vf is
12     component ch4vf_DEBOUNCER
13         Port ( OSC,D_in,CLR : in STD_LOGIC;
14                 D_out : out STD_LOGIC);
15     end component;
16     signal C_DB,C,QT : STD_LOGIC;
17     signal JK : STD_LOGIC_VECTOR(1 downto 0);
18 begin
19     -----Debouncer-----
20     DEBOUNCE_JKFF : ch4vf_DEBOUNCER port map ( OSC => OSC,
21                                                 D_in => Din,
22                                                 CLR => CLR_DB,
23                                                 D_out=> C );
24     -----JK FLIP-FLOP-----
25     JK <= (J&K); -- Concatenation
26     process(C,CLR)
27     begin
28         if CLR='1' then QT <= '0';
29         elsif (C'event and C='1') then
30             if JK="00" then QT <= QT;
31             elsif JK="01" then QT <= '0';
32             elsif JK="10" then QT <= '1';
33             elsif JK="11" then QT <= not QT;
```

(ต่อ)

```

34      end if;
35  end if;
36 end process;
37   Q <= QT;  Q_bar <= not QT;
38 end Behavioral;

```

รูปที่ L2.2 โค้ด JK Flip-Flop ที่รวมวงจรดีเบาเซอร์ในรูปของ Component ไว้แล้ว

การกำหนดขาสัญญาณต่างๆ โดยใช้อาร์ชิเตกเจอร์ OSC = 25MHz และปุ่มกด PB1-PB2 และ Dip SW1-Dip SW3 เป็นอินพุต มี LED L0-L1 เป็นเอาต์พุตกล่าวคือ

Din = PB1 = INPUT = p44	Q = L1 = OUTPUT = p77
CLR_DB = PB2 = INPUT = p46	Q_bar = L0 = OUTPUT = p70
J = Dip SW1 = INPUT = p52	OSC = OSC = INPUT = p127
K = Dip SW2 = INPUT = p53	CLR = Dip SW3 = INPUT = p55

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLR	Input	p55	BANK4	LVCMS33	N/A	3.30					Unknown		
CLR_DB	Input	p46	BANK5	LVCMS33	N/A	3.30					Unknown		
Din	Input	p44	BANK5	LVCMS33	N/A	3.30					Unknown		
J	Input	p52	BANK5	LVCMS33	N/A	3.30					Unknown		
K	Input	p53	BANK5	LVCMS33	N/A	3.30					Unknown		
OSC	Input	p127	BANK0	LVCMS33	N/A	3.30					Unknown		
Q	Output	p77	BANK3	LVCMS33	N/A	3.30			SLOW		Unknown		
Q_bar	Output	p70	BANK4	LVCMS33	N/A	3.30			SLOW		Unknown		

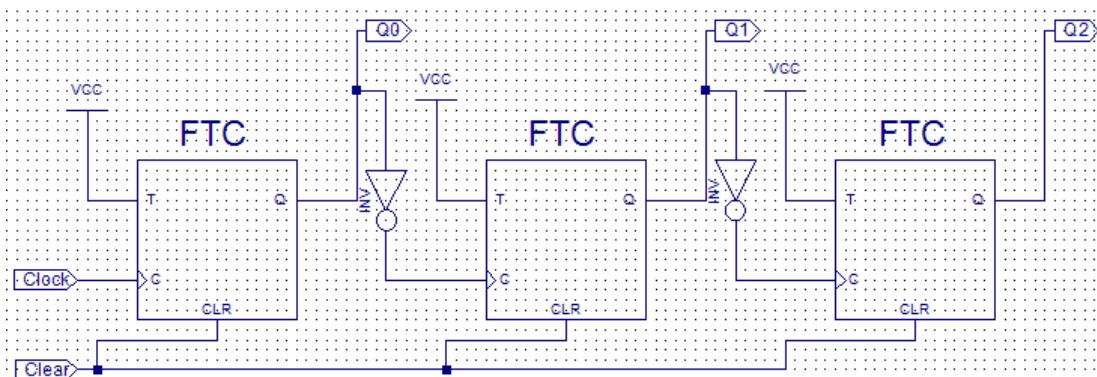
รูปที่ L2.3 Assign Package Pins

หลังจากโปรแกรมวงจรลง FPGA แล้วให้เซต Dip SW3 ไปที่ ON (CLR = '0') จากนั้นเซตค่า J และ K เป็นค่าต่างๆ เริ่มจาก J = '0' และ K = '0' ไปจนกระทั่ง J = '1' และ K = '1' โดยใช้ Dip SW1 และ Dip SW2 (Dip SW1 ON แล้ว J = '0', OFF แล้ว J = '1' และ Dip SW2 ON แล้ว K = '0', OFF แล้ว K = '1') แล้วกดปุ่ม PB1 และปุ่ม PB2 ลับกันเพื่อสร้าง Clock พร้อมกับให้สังเกตดูที่ LED L0 และ L1 ในแต่ละเงื่อนไขว่าเป็นตามทฤษฎีหรือไม่ กดปุ่ม PB1 และปุ่ม PB2 ลับกันจนกระทั่ง LED L1 ติดสว่างเสร็จแล้วให้เซต Dip SW3 ไปที่ OFF (CLR = '1') พร้อมกับให้สังเกตดูที่ LED L1 ว่าดับเป็นตามทฤษฎีหรือไม่ (กรณีที่เป็น T Flip-Flop ให้เซต J = K = '1') เสร็จแล้วจึงบันทึกผลการทดลอง

4.3 วงจรนับเลขไบนาเรีย (Binary counter)

4.3.1 วงจรนับแบบอะซิงโกรนัส (Asynchronous counter)

วงจรนับแบบอะซิงโกรนัสหรือแบบริปเปล (Ripple counter) จะสร้างจาก T Flip-Flop โดยเขต $T = '1'$ หรือถ้าใช้ JK Flip-Flop จะเขต $J = K = '1'$ เพื่อทำงานในโหมด Toggle ทุกๆ ครั้งที่มีการทริกด้วยขอบขาขึ้นของ Clock (หรือออกแบบให้ทริกขอบขาลงก็ได้) วงจรนับขึ้นแบบเลขไบนาเรีย (Binary up-counter) ขนาด 3 บิตแสดงดังรูปที่ 4.10 ซึ่งจะใช้ T Flip-Flop แบบอะซิงโกรนัสเคลียร์ (FTC) ไปทำเป็นวงจรนับ 8 กล่าวคือนับได้สูงสุด $= 2^N - 1$ โดยที่ N คือ จำนวนบิตหรือจำนวน Flip-Flop วงจรนับจะเกลี้ยงเมื่อ Clear = '1' ในกรณีของวงจรนับขึ้นเราจะใส่อินเวอร์เตอร์ INV เพื่อให้อาตพุตจาก Q0 และ Q1 กลายเป็นขอบขาขึ้นเพื่อใช้ทริก T Flip-Flop ตัวถัดไป และขอให้สังเกตว่าการ Toggle ของ T Flip-Flop แต่ละตัวจะทำให้ความถี่สัญญาณนาฬิกา (Clock) ที่อาตพุตลดลงครึ่งหนึ่งจากความถี่เดิมหรือทำหน้าที่เป็นวงจรหารส่วนของความถี่เดิมนั่นเอง



รูปที่ 4.10 ผังวงจรนับขึ้นแบบอะซิงโกรนัสหรือแบบริปเปล 3 บิต

จากผังวงจรนับขึ้นแบบอะซิงโกรนัสหรือแบบริปเปล 3 บิตในรูปที่ 4.10 นี้เราสามารถนำโค้ดของ T Flip-Flop แบบอะซิงโกรนัสเคลียร์แสดงดังรูปที่ 4.11 มาทำเป็น Component เพื่อนำไปเก็บโค้ดของวงจรนับ 8 ได้ดังรูปที่ 4.12 ซึ่งผลการสังเคราะห์วงจร (View technology schematic) จะได้ดังรูปที่ 4.13 และผลจำลองการทำงานของวงจรนับจะได้ดังรูปที่ 4.14

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity TFF is
6   Port ( C,CLR,T : in STD_LOGIC;
7         Q : out STD_LOGIC);
8 end TFF;
9
10 architecture Behavioral of TFF is
11   signal QT : STD_LOGIC;
12 begin
13   process(C,CLR)
14   begin
15     if CLR='1' then QT <= '0';
16     elsif (C'event and C='1') then
17       if T='0' then QT <= QT;
18       elsif T='1' then QT <= not QT;
19       end if;
20     end if;
21   end process;
22   Q <= QT;
23 end Behavioral;

```

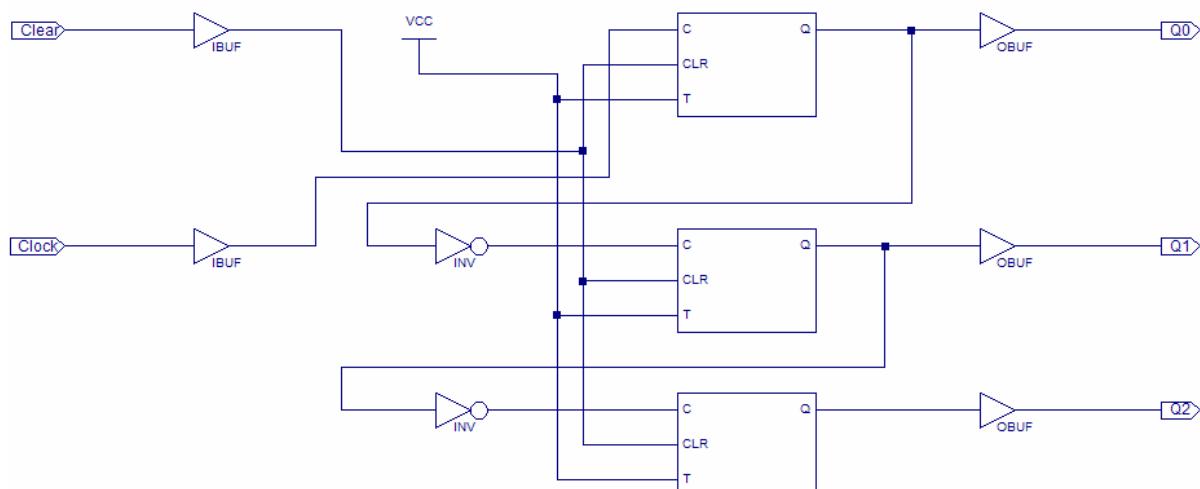
รูปที่ 4.11 โค้ดของวงจร T Flip-Flop แบบอะซิงโกรนัสเคลียร์ที่ทริกด้วยขอบขาขึ้นที่จะนำไปสร้างเป็น Component

```

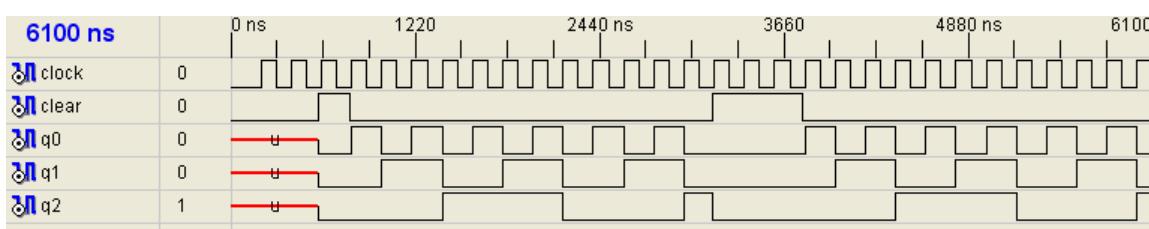
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_3BIT_ASYNC_UP is
6     Port ( Clock, Clear : in STD_LOGIC;
7             Q0,Q1,Q2 : out STD_LOGIC);
8 end COUNTER_3BIT_ASYNC_UP;
9
10 architecture Behavioral of COUNTER_3BIT_ASYNC_UP is
11     signal Tin0,Tin1,Tin2 : STD_LOGIC;
12     signal C1,C2 : STD_LOGIC;
13     signal Q0T,Q1T,Q2T : STD_LOGIC;
14     component TFF
15         Port ( C,CLR,T : in STD_LOGIC;
16                 Q : out STD_LOGIC);
17     end component;
18 begin
19     Tin0 <= '1'; -- T=Vcc
20     Tin1 <= '1'; -- T=Vcc
21     Tin2 <= '1'; -- T=Vcc
22     BIT_0 : TFF port map( C=>Clock,CLR=>Clear,T=>Tin0,Q=>Q0T );
23     C1 <= not Q0T;
24     BIT_1 : TFF port map( C=>C1,CLR=>Clear,T=>Tin1,Q=>Q1T );
25     C2 <= not Q1T;
26     BIT_2 : TFF port map( C=>C2,CLR=>Clear,T=>Tin2,Q=>Q2T );
27     Q0 <= Q0T;
28     Q1 <= Q1T;
29     Q2 <= Q2T;
30 end Behavioral;

```

รูปที่ 4.12 โค้ดของวงจรนับขึ้นแบบอะซิงไครนัส 3 บิตที่สร้างจาก Component ในรูปที่ 4.11

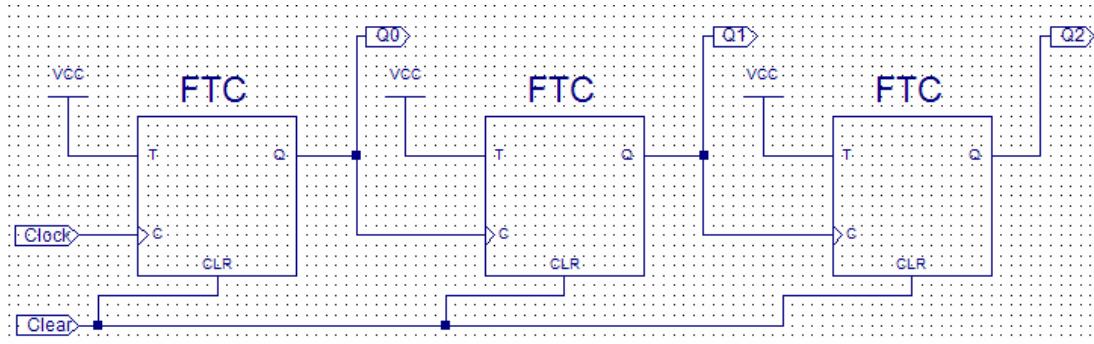


รูปที่ 4.13 พังวงจรนับขึ้นแบบอะซิงไครนัส 3 บิตที่ได้จากการสังเคราะห์วงจร (View technology schematic)



รูปที่ 4.14 ผลจำลองการทำงานของวงจรนับขึ้นแบบอะซิงไครนัส 3 บิต ชี้งค่า U (Uninitial) เกิดจากการไม่ใส่ค่าเริ่มต้น

ในทำนองเดียวกันการออกแบบวงจรนับลงแบบอะซิงไครนัสหรือแบบริปเปิล 3 บิตนี้จะได้ดังรูปที่ 4.15 ซึ่งวงจรนี้จะใช้ T Flip-Flop แบบอะซิงไครนัสเคลียร์ (FTC) มาทำเป็นวงจรนับ 8 และโค๊ดของวงจรนับลงแบบอะซิงไครนัส 3 บิตที่สร้างจาก Component ในรูปที่ 4.11 นั้นจะได้ดังรูปที่ 4.16 และผลจำลองการทำงานของวงจรนับจะได้ดังรูปที่ 4.17



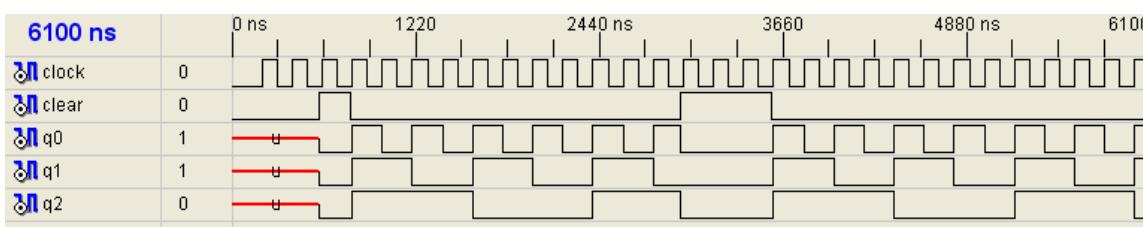
รูปที่ 4.15 ผังวงจรนับลงแบบอะซิงไครนัสหรือแบบริปเปิล 3 บิต

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_3BIT_ASYNC_DN is
6     Port ( Clock, Clear : in STD_LOGIC;
7             Q0,Q1,Q2 : out STD_LOGIC);
8 end COUNTER_3BIT_ASYNC_DN;
9
10 architecture Behavioral of COUNTER_3BIT_ASYNC_DN is
11     signal Tin0,Tin1,Tin2 : STD_LOGIC;
12     signal Q0T,Q1T,Q2T : STD_LOGIC;
13     component TFF
14         Port ( C,CLR,T : in STD_LOGIC;
15                 Q : out STD_LOGIC);
16     end component;
17 begin
18     Tin0 <= '1'; -- T=Vcc
19     Tin1 <= '1'; -- T=Vcc
20     Tin2 <= '1'; -- T=Vcc
21     BIT_0 : TFF port map( C=>Clock,CLR=>Clear,T=>Tin0,Q=>Q0T );
22     BIT_1 : TFF port map( C=>Q0T,CLR=>Clear,T=>Tin1,Q=>Q1T );
23     BIT_2 : TFF port map( C=>Q1T,CLR=>Clear,T=>Tin2,Q=>Q2T );
24     Q0 <= Q0T;
25     Q1 <= Q1T;
26     Q2 <= Q2T;
27 end Behavioral;

```

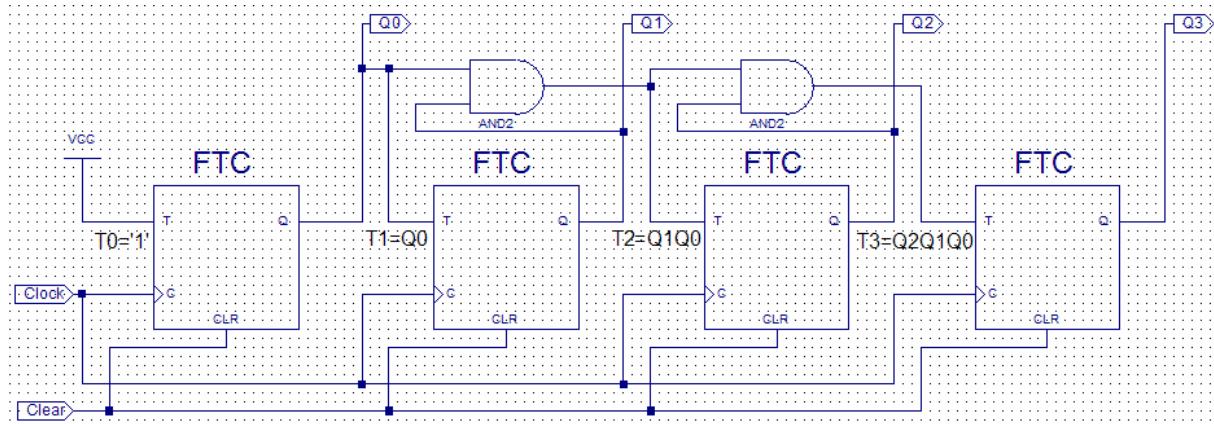
รูปที่ 4.16 วงจรนับลงแบบอะซิงไครนัส 3 บิตที่สร้างจาก Component ในรูปที่ 4.11



รูปที่ 4.17 ผลจำลองการทำงานของวงจรนับลงแบบอะซิงไครนัส 3 บิต (U = Uninitial)

4.3.2 วงจรนับแบบซิงโครนัส (Synchronous counter)

วงจรนับแบบซิงโครนัสเป็นวงจรที่ถูกออกแบบให้ Clock ทริก Flip-Flop ทุกตัวพร้อมกัน ทำให้วงจรนับมีเวลาหน่วงภายในน้อยจึงสามารถทำงานที่ความเร็วสูงได้ วงจรนับขึ้นแบบซิงโครนัส 4 บิตแสดงดังรูปที่ 4.18 ซึ่งวงจรนี้จะใช้ T Flip-Flop แบบอะซิงโครนัสเคลียร์ (FTC) มาทำเป็นวงจรนับ ซึ่งปกติจะใช้ T Flip-Flop หรือใช้ JK Flip-Flop (โดยชุดให้ $J = K = 1$) มาทำเป็นวงจรนับได้ ซึ่งค่าสูงสุดที่นับได้ $= 2^N - 1$ โดยที่ N คือ จำนวนบิต



รูปที่ 4.18 วงจรนับขึ้นแบบซิงโครนัส 4 บิต

จากรูปที่ 4.18 เราสามารถเขียนโค้ดวงจรนับนี้ได้ดังรูปที่ 4.19 โดยสรุปเป็นเงื่อนไขทั่วๆ ไปในการออกแบบได้ดังนี้

$$T_0 = 1$$

$$T_1 = Q_0$$

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

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

หรือ

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

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

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_4BIT_SYNC_UP is
6   Port ( Clock, Clear : in STD_LOGIC;
7         Q0,Q1,Q2,Q3 : out STD_LOGIC);
8 end COUNTER_4BIT_SYNC_UP;
9
10 architecture Behavioral of COUNTER_4BIT_SYNC_UP is
11   signal T0,T1,T2,T3 : STD_LOGIC;
12   signal Q0T,Q1T,Q2T,Q3T : STD_LOGIC;
13   component TFF
14     Port ( C,CLR,T : in STD_LOGIC;
15             Q : out STD_LOGIC);
16   end component;
```

(ต่อ)

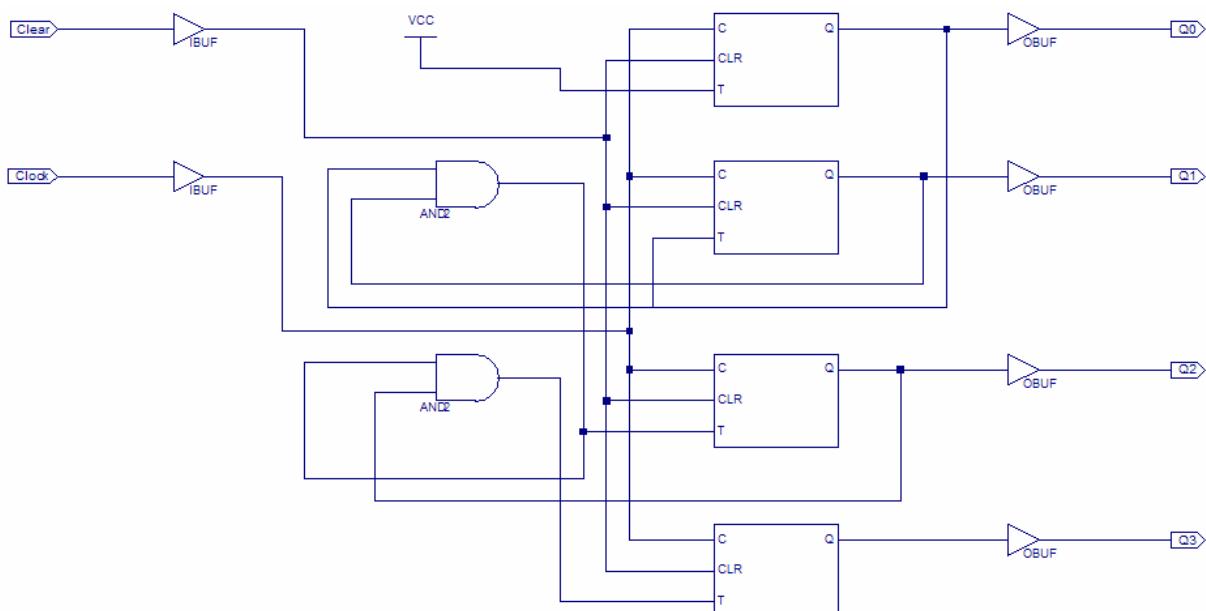
```

17 begin
18   T0 <= '1'; -- T=Vcc
19   BIT_0 : TFF port map( C=>Clock,CLR=>Clear,T=>T0,Q=>Q0T );
20   T1 <= Q0T;
21   BIT_1 : TFF port map( C=>Clock,CLR=>Clear,T=>T1,Q=>Q1T );
22   T2 <= Q1T and T1;
23   BIT_2 : TFF port map( C=>Clock,CLR=>Clear,T=>T2,Q=>Q2T );
24   T3 <= Q2T and T2;
25   BIT_3 : TFF port map( C=>Clock,CLR=>Clear,T=>T3,Q=>Q3T );
26   Q0 <= Q0T;
27   Q1 <= Q1T;
28   Q2 <= Q2T;
29   Q3 <= Q3T;
30 end Behavioral;

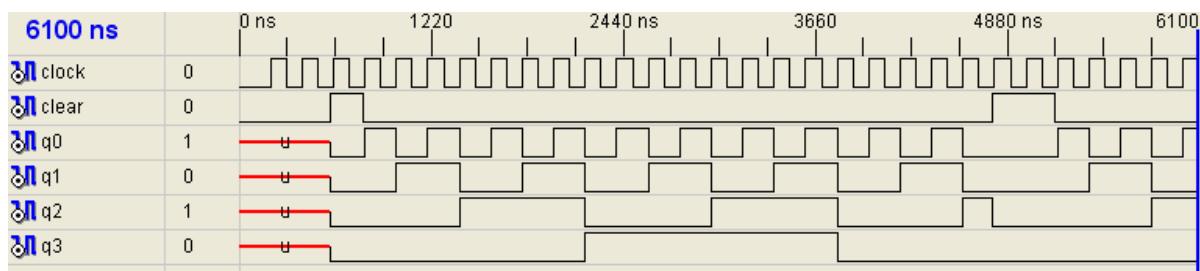
```

รูปที่ 4.19 โค้ดของวงจรนับขีนแบบซิงโครนัส 4 บิตที่สร้างจาก Component ในรูปที่ 4.11

จากโค้ดของวงจรนับขีนแบบซิงโครนัส 4 บิตในรูปที่ 4.19 เมื่อทำการสังเคราะห์วงจร (View technology schematic) จะได้ดังรูปที่ 4.20 และผลจำลองการทำงานของวงจรจะได้ดังรูปที่ 4.21

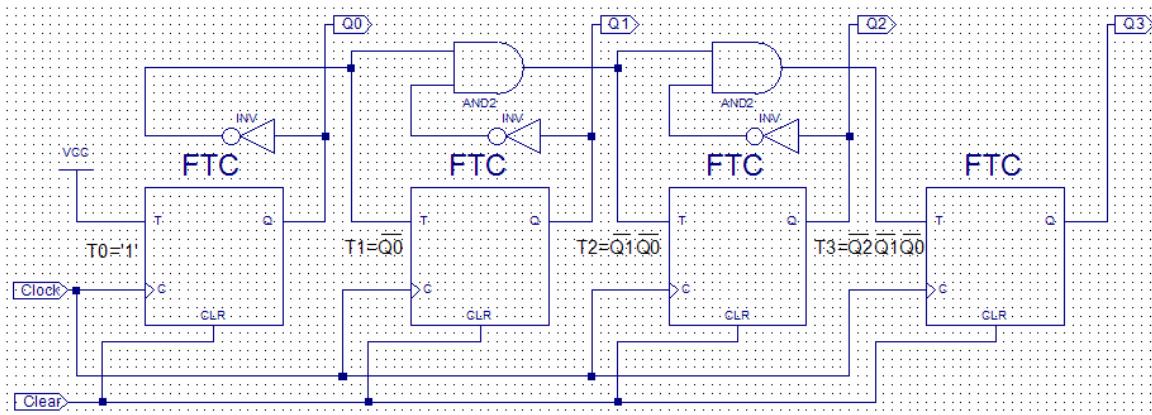


รูปที่ 4.20 ผังวงจรนับขีนแบบซิงโครนัส 4 บิตที่ได้จากการสังเคราะห์ (View technology schematic)



รูปที่ 4.21 ผลจำลองการทำงานของวงจรนับขีนแบบซิงโครนัส 4 บิตที่ใช้โค้ดในรูปที่ 4.19 (U = Uninitial)

ในทำงานองค์ประกอบของเราสามารถเขียนโค้ดของวงจรนับลงแบบซิงโครนัส 4 บิตจากผังวงจรในรูปที่ 4.22 ได้ดังรูปที่ 4.23 และเมื่อทำการจำลองการทำงานของวงจรจะได้ดังรูปที่ 4.24 ซึ่งค่าสูงสุดที่นับได้ $= 2^N - 1$ โดยที่ N คือจำนวนบิต



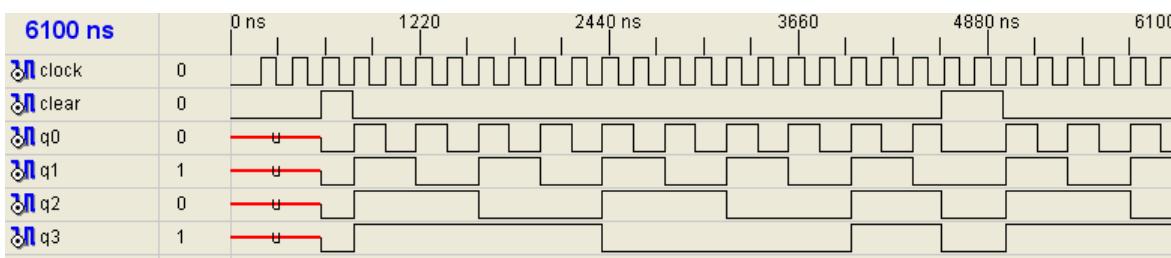
รูปที่ 4.22 ผังวงจรนับลงแบบชิงโครนัส 4 บิต

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_4BIT_SYNC_DN is
6     Port ( Clock,Clear : in STD_LOGIC;
7             Q0,Q1,Q2,Q3 : out STD_LOGIC);
8 end COUNTER_4BIT_SYNC_DN;
9
10 architecture Behavioral of COUNTER_4BIT_SYNC_DN is
11     signal T0,T1,T2,T3 : STD_LOGIC;
12     signal Q0T,Q1T,Q2T,Q3T : STD_LOGIC;
13     component TFF
14         Port ( C,CLR,T : in STD_LOGIC;
15                 Q : out STD_LOGIC);
16     end component;
17 begin
18     T0 <= '1'; -- T=Vcc
19     BIT_0 : TFF port map( C=>Clock,CLR=>Clear,T=>T0,Q=>Q0T );
20     T1 <= not Q0T;
21     BIT_1 : TFF port map( C=>Clock,CLR=>Clear,T=>T1,Q=>Q1T );
22     T2 <= not Q1T and T1;
23     BIT_2 : TFF port map( C=>Clock,CLR=>Clear,T=>T2,Q=>Q2T );
24     T3 <= not Q2T and T2;
25     BIT_3 : TFF port map( C=>Clock,CLR=>Clear,T=>T3,Q=>Q3T );
26     Q0 <= Q0T;
27     Q1 <= Q1T;
28     Q2 <= Q2T;
29     Q3 <= Q3T;
30 end Behavioral;

```

รูปที่ 4.23 โค้ดวงจรนับลงแบบชิงโครนัส 4 บิตที่สร้างจาก Component ในรูปที่ 4.11



รูปที่ 4.24 ผลจำลองการทำงานของวงจรนับลงแบบชิงโครนัส 4 บิตที่ใช้โค้ดในรูปที่ 4.22 (U = Uninitial)

การเขียนโค้ด VHDL จากผังวงจรนับขึ้นมา ก่อน ทำให้เสียเวลาและต้องใช้ทักษะในการออกแบบวงจรอย่างมาก ซึ่งแตกต่างจากการเขียนโค้ด วงจรนับแบบชิงโครนัส 4 บิตในตัวอย่างที่ 2.26 ของบทที่ 2 ที่เป็นการเขียนโค้ดหลากหลายสไตล์ และเป็นการออกแบบโดยใช้ความสามารถของภาษา VHDL และซอฟต์แวร์ทุกช่วงในการออกแบบ ซึ่งทางเลือกที่ดีกว่า เพราะสามารถออกแบบได้รวดเร็ว

และมีประสิทธิภาพ ตัวอย่างการเขียนโค้ดของวงจรนับขึ้นแบบซิงโกรนัส 4 บิตแสดงดังรูปที่ 4.25 ซึ่งเมื่อทำการสังเคราะห์ วงจร โค้ดที่ใช้ CPLD จะได้ผังวงจรดังรูปที่ 4.26 และผลจำลองการทำงานแสดงดังรูปที่ 4.27

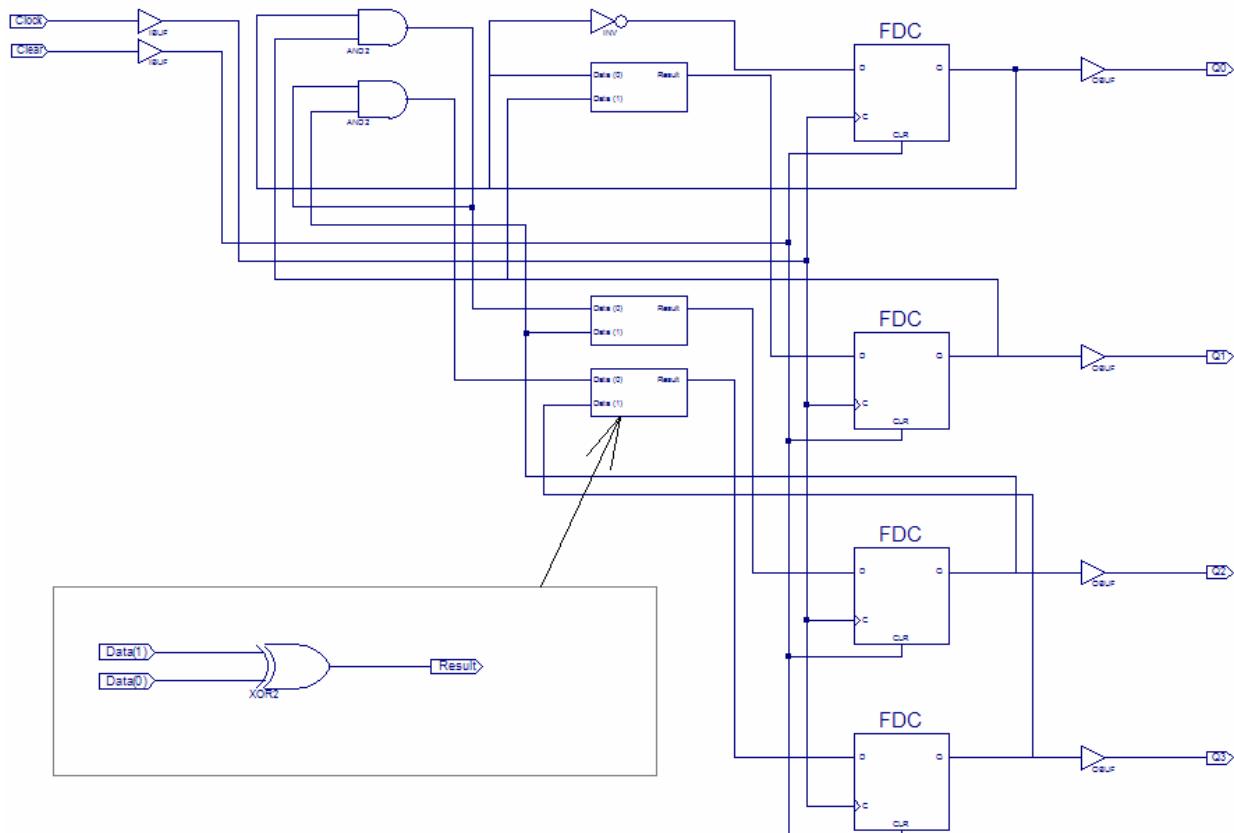
จากรูปที่ 4.26 คณที่มีความรู้เรื่องดิจิตอลมาเป็นอย่างดีก็จะมองออกว่างจร D Flip-Flop ที่มีอินเวอร์เตอร์ต่อหลังเข้ามา ที่อินพุต D ก็คือ T Flip-Flop ที่ถูกกำหนดให้ $T = '1'$ และ D Flip-Flop ที่มี XOR ต่อหลังเข้ามาที่อินพุต D ก็คือ T Flip-Flop นั่นเอง นั่นก็หมายความว่า Xilinx Synthesis Tool หรือ XST สังเคราะห์วงจรนับขึ้น 4 บิตนี้เป็นวงจรนับแบบซิงโกรนัส

```

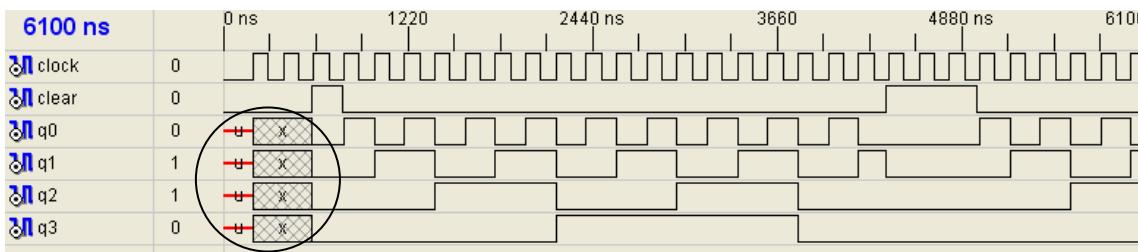
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity COUNTER_4BIT_SYNC_UP is
7     Port ( Clock, Clear : in STD_LOGIC;
8             Q0,Q1,Q2,Q3 : out STD_LOGIC);
9 end COUNTER_4BIT_SYNC_UP;
10
11 architecture Behavioral of COUNTER_4BIT_SYNC_UP is
12     signal QT : STD_LOGIC_VECTOR (3 downto 0);
13 begin
14     process(Clock, Clear)
15     begin
16         if Clear='1' then QT <= "0000";
17         elsif Clock'event and Clock='1' then QT <= QT + 1;
18         end if;
19     end process;
20     Q0 <= QT(0); Q1 <= QT(1); Q2 <= QT(2); Q3 <= QT(3);
21 end Behavioral;

```

รูปที่ 4.25 โค้ดของวงจนับขึ้นแบบซิงโกรนัส 4 บิต



รูปที่ 4.26 โค้ดของวงจนับขึ้นแบบซิงโกรนัส 4 บิตที่ XST สังเคราะห์ห่วงจร (View technology schematic) โค้ดที่ใช้ CPLD



รูปที่ 4.27 ผลจำลองการทำงานของโก้ดังวงจรนับขึ้นแบบซิงไครนัส 4 บิต (U = Uninitial และ 'X' = Unknown)

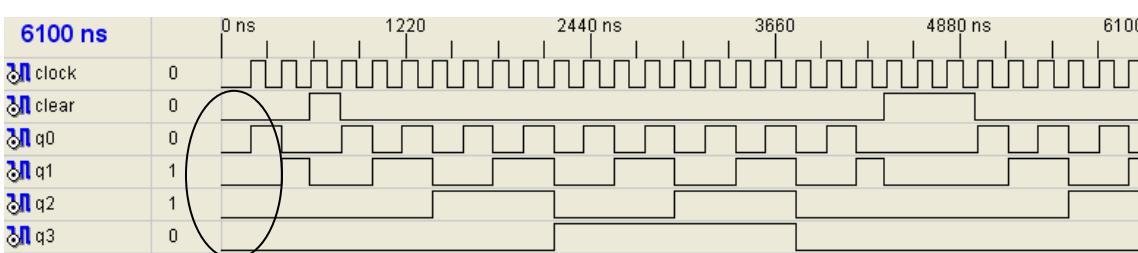
การกำหนดค่าเริ่มต้น (Initial value) ให้รีจิสเตอร์หรือ Flip-Flop จ้าไม่มีการกำหนดค่าเริ่มต้นนี้ XST จะกำหนด (Default) ค่าเริ่มต้นเป็น ‘0’ แต่ ISE Simulator จะกำหนดค่าเป็น ‘U’ (U = Uninitial) ดังรูปที่ 4.27 ซึ่งโก้ดังวงจรนับในรูปที่ 4.25 นั้นเมื่อกำหนดค่าเริ่มต้นให้กับ Signal QT เป็น “0000” ดังในรูปที่ 4.28 แล้วจะให้ผลจำลองการทำงานแสดงดังรูปที่ 4.29

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity COUNTER_4BIT_SYNC_UP is
7     Port ( Clock,Clear : in STD_LOGIC;
8             Q0,Q1,Q2,Q3 : out STD_LOGIC);
9 end COUNTER_4BIT_SYNC_UP;
10
11 architecture Behavioral of COUNTER_4BIT_SYNC_UP is
12     signal QT : STD_LOGIC_VECTOR (3 downto 0):= "0000";--Initial value
13 begin
14 process(Clock,Clear)
15 begin
16     if      Clear='1' then QT <= "0000";
17     elsif Clock'event and Clock='1' then QT <= QT + 1;
18     end if;
19 end process;
20     Q0 <= QT(0); Q1 <= QT(1); Q2 <= QT(2); Q3 <= QT(3);
21 end Behavioral;

```

รูปที่ 4.28 โก้ดังวงจรนับขึ้นแบบซิงไครนัส 4 บิตที่ได้ค่าเริ่มต้นให้กับ Signal QT เป็น “0000”



รูปที่ 4.29 ผลจำลองการทำงานของวงจรนับที่กำหนดค่าเริ่มต้นให้กับ Signal QT เป็น “0000”

ตัวอย่างโก้ดังวงจรนับขึ้น-นับลง 4 บิตแบบคาสเคดแสดงดังรูปที่ 4.30 เมื่อขา Direction หรือ DIR = ‘0’ และเมื่อวงจรนับขึ้นไปถึงค่าสูงสุดแล้วขา Clock enable output หรือ CEO = ‘1’ ในทำงานองเดียวกันเมื่อขา DIR = ‘1’ และวงจรนับลงจนถึงค่าต่ำสุดแล้วขา Clock enable output หรือ CEO = ‘1’ โดยผลจำลองการทำงานแสดงดังรูปที่ 4.31

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity COUNTER_4BIT_UPDN is
7     Port ( C,CE,CLR,DIR : in STD_LOGIC;
8             CEO : out STD_LOGIC;
9             Q : out STD_LOGIC_VECTOR (3 downto 0));
10 end COUNTER_4BIT_UPDN;

```

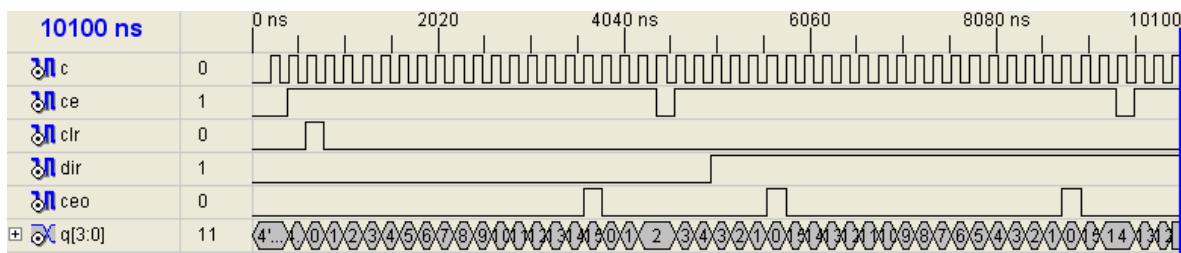
(ต่อ)

```

11
12 architecture Behavioral of COUNTER_4BIT_UPDN is
13   signal QT : STD_LOGIC_VECTOR (3 downto 0);
14 begin
15   -----4Bits Counter-----
16   process(C,CLR)
17   begin
18     if CLR='1' then QT <= "0000";
19   elsif C'event and C='1' then
20     if CE='1' then
21       if DIR='0' then QT <= QT + 1; --Count up
22       else QT <= QT - 1;           --Count down
23     end if;
24   end if;
25 end process;
26
27 Q <= QT;
28 -----CEO-----
29 CEO <= CE when (DIR='0' and QT=15) else
30   CE when (DIR='1' and QT=0) else
31   '0';
32 end Behavioral;

```

รูปที่ 4.30 โค้ดของวงจรนับขึ้น-นับลง 4 บิตแบบภาษาสเปคเมื่อเรียกใช้ Std_logic_unsigned package



รูปที่ 4.31 ผลจำลองการทำงานของวงจรนับที่ใช้โค้ดในรูปที่ 4.29 หรือรูปที่ 4.30

วงจรนับขึ้น-นับลง 16 บิตที่สร้างจากวงจรนับแบบภาษาสเปค 4 บิตในรูปที่ 4.30 ที่เป็น Component และคงดังรูปที่ 4.32 ซึ่งขอให้ผู้อ่านควรสังเกตว่าข้าของ Component ที่ไม่ใช้งานให้ใช้คำสั่ง Open (ดูบรรทัดที่ 21 ในรูปที่ 4.32)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_16BIT_UPDN is
6   Port ( C,CLR,DIR : in STD_LOGIC;
7         Q0,Q1,Q2,Q3 : out STD_LOGIC_VECTOR (3 downto 0));
8 end COUNTER_16BIT_UPDN;
9
10 architecture Behavioral of COUNTER_16BIT_UPDN is
11   component COUNTER_4BIT_UPDN
12     Port ( C,CE,CLR,DIR : in STD_LOGIC;
13            CEO : out STD_LOGIC;
14            Q : out STD_LOGIC_VECTOR (3 downto 0));
15   end component;
16   signal CEO : STD_LOGIC_VECTOR (2 downto 0);
17 begin
18   D_0 : COUNTER_4BIT_UPDN port map(C=>C,CE=>'1',CLR=>CLR,DIR=>DIR,CEO=>CEO(0),Q=>Q0);
19   D_1 : COUNTER_4BIT_UPDN port map(C=>C,CE=>CEO(0),CLR=>CLR,DIR=>DIR,CEO=>CEO(1),Q=>Q1);
20   D_2 : COUNTER_4BIT_UPDN port map(C=>C,CE=>CEO(1),CLR=>CLR,DIR=>DIR,CEO=>CEO(2),Q=>Q2);
21   D_3 : COUNTER_4BIT_UPDN port map(C=>C,CE=>CEO(2),CLR=>CLR,DIR=>DIR,CEO=>open,Q=>Q3);
22 end Behavioral;

```

รูปที่ 4.32 โค้ดและวงจรนับขึ้น-นับลงขนาด 16 บิตที่สร้างจากวงจรนับ 4 บิตในรูปที่ 4.30 มาทำเป็น Component

การทดลองที่ 4.3.1 การสร้างวงจรนับขีนแบบริปเปล

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานและออกแบบวงจรนับขีนแบบริปเปล
- 2) เพื่อสร้างวงจรนับขีนแบบริปเปลโดยวิธีเขียนด้วย โค้ด VHDL และโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรนับขีนแบบริปเปลด้วย CPLD

1.1) เขียนโค้ดวงจรนับขีนแบบซิงโกรนัชนาด 3 บิต โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ch4vcxl_ripple_c8_up จากนั้นจึงเขียนโค้ดใหม่โดยคัดลอกโค้ดในรูปที่ 4.12 และแก้ไขชื่อ entity เป็น ch4vcxl_ripple_c8_up แล้วให้คัดลอกโค้ด T Flip-Flop แบบซิงโกรนัสเคเลียร์ในรูปที่ 4.11 ทั้งหมดไปแทรกไว้ด้านบนสุดจะได้ดังรูปที่ L1.1 ซึ่งเป็นการเขียนโค้ดของ Component รวมไว้ในไฟล์โค้ดหลัก (ถือเป็นการ Add Source ของ Component โดยอัตโนมัติ) เสร็จแล้วตรวจสอบความถูกต้องและบันทึกไฟล์ จากนั้นจึงทำ Behavioral simulation โดยใช้ Source File ชื่อ ch4vcxl_ripple_c8_up_tb1 แล้วกำหนดค่าต่างๆ ดังรูปที่ L1.2 และ Waveform ดังรูปที่ L1.3 แล้วคุณจะสามารถทำงานในรูปที่ L1.4 ว่าเป็นไปตามทฤษฎีหรือไม่

```

2 -----Component:T Flip-Flop-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity TFF is
7     Port ( C,CLR,T : in STD_LOGIC;
8             Q : out STD_LOGIC);
9 end TFF;
10
11 architecture Behavioral of TFF is
12     signal QT : STD_LOGIC;
13 begin
14 process(C,CLR)
15     begin
16         if CLR='1' then QT <= '0';
17         elsif (C'event and C='1') then
18             if      T='0' then QT <= QT;
19             elsif T='1' then QT <= not QT;
20             end if;
21         end if;
22     end process;
23     Q <= QT;
24 end Behavioral;
25
26 -----Main code : Ripple counter-----
27 library IEEE;
28 use IEEE.STD_LOGIC_1164.ALL;
29
30 entity ch4vcxl_ripple_c8_up is
31     Port ( Clock,Clear : in STD_LOGIC;
32             Q: out STD_LOGIC_VECTOR (2 downto 0));
33 end ch4vcxl_ripple_c8_up;

```

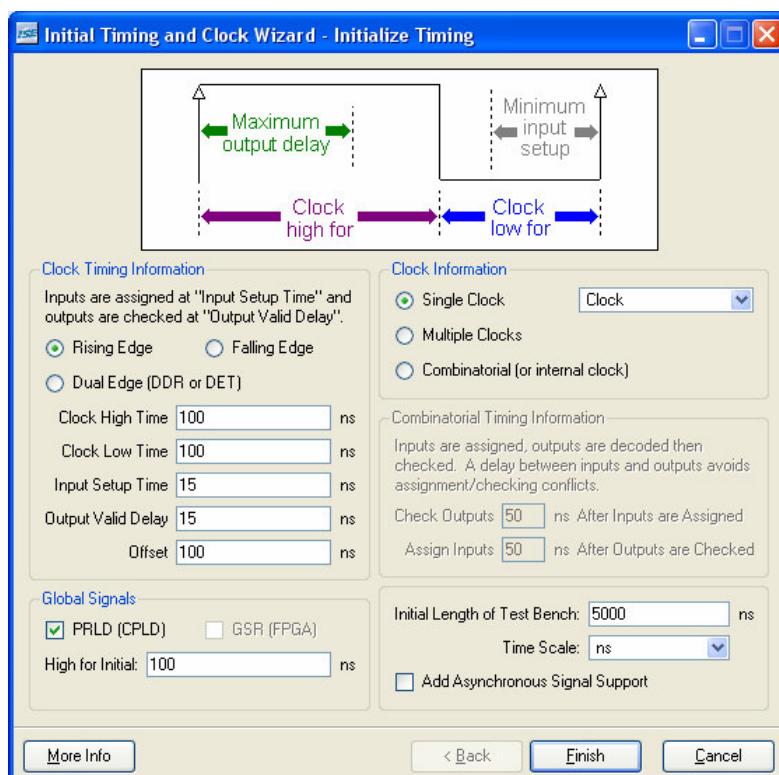
(ต่อ)

```

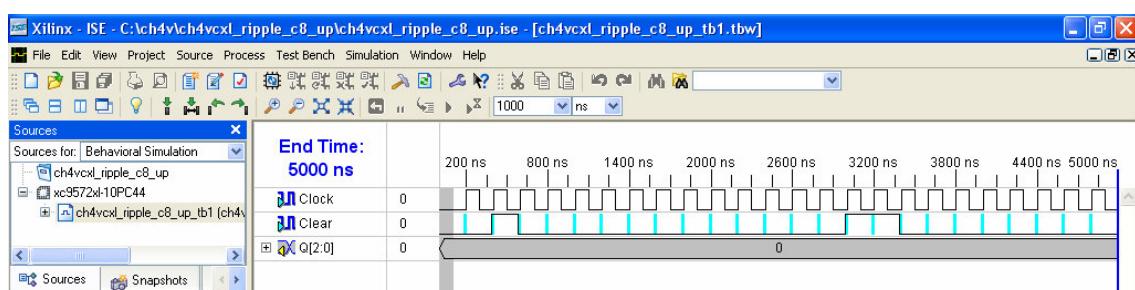
34
35 architecture Behavioral of ch4vcxl_ripple_c8_up is
36   signal C1,C2 : STD_LOGIC;
37   signal QT : STD_LOGIC_VECTOR (2 downto 0);
38   component TFF
39     Port ( C,CLR,T : in STD_LOGIC;
40            Q : out STD_LOGIC);
41   end component;
42 begin
43   BIT_0 : TFF port map( C=>Clock,CLR=>Clear,T=>'1',Q=>QT(0));
44   C1 <= not QT(0);
45   BIT_1 : TFF port map( C=>C1,CLR=>Clear,T=>'1',Q=>QT(1));
46   C2 <= not QT(1);
47   BIT_2 : TFF port map( C=>C2,CLR=>Clear,T=>'1',Q=>QT(2));
48   Q <= QT;
49 end Behavioral;

```

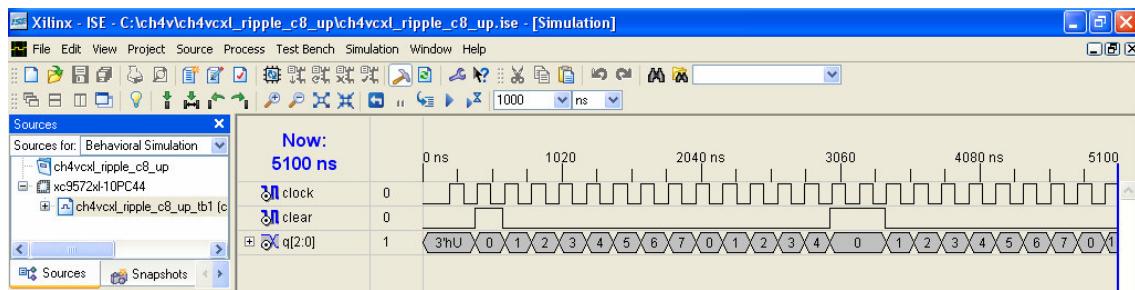
รูปที่ L1.1 โค้ดวงจรนับขึ้นแบบซิงโครนัสขนาด 3 บิตที่เขียนรวม logic ของ T Flip-Flop ที่เป็น Component ไว้แล้ว



รูปที่ L1.2 หน้าต่าง Initial Timing and Clock Wizard



รูปที่ L1.3 หน้าต่างสำหรับกำหนดสัญญาณต่างๆที่ป้อนให้กับอินพุตของวงจรที่เราต้องการทดสอบ



รูปที่ L1.4 ผล Behavioral simulation ของวงจรนับ 3 บิตแบบนับขึ้น

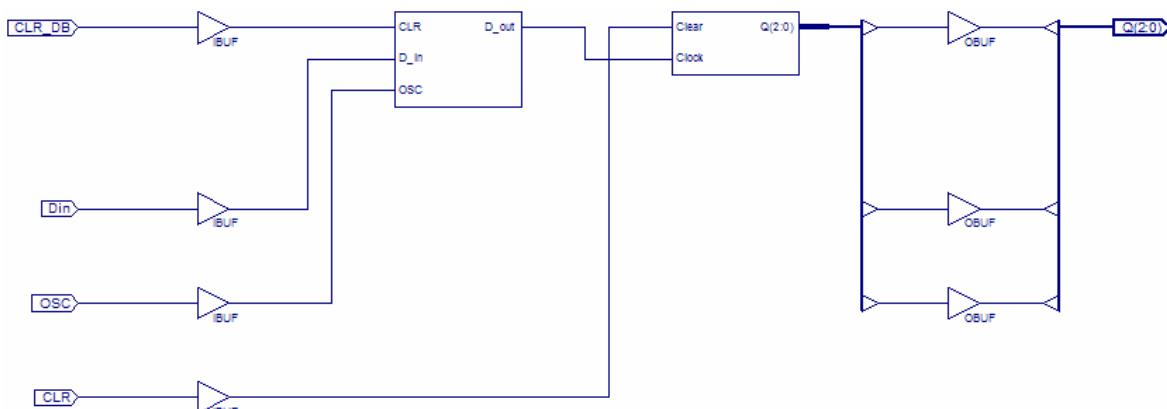
1.2) ในทางปฏิบัติ ผลการทดลองของวงจรนับจะไม่ตรงกับผลจำลองการทำงานเนื่องจากการเกิดเบ้าช์ในขณะกดปุ่ม การแก้ไขปัญหานี้ทำได้โดยเพิ่นโภคความจดจำเบาช์หรือรวมไว้ในโภคของวงจรนับ โดยสร้างไฟล์ใน Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_3_1vcxl จากนั้นนำไฟล์ความจดจำเบาช์ชื่อ ch4vf_DEBOUNCER และไฟล์ของวงจรนับขึ้นชื่อ ch4vcxl_ripple_c8_up ที่มีอยู่แล้วมาทำเป็น Component และ Add Source ของ Component ทั้ง 2 ไฟล์เข้าไปในไฟล์ Project โดยโภคที่ได้แสดงดังรูปที่ L1.5 และผลการสังเคราะห์วงจร (View technology schematic) และดังรูปที่ L1.6

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_3_1vcxl is
6     Port ( CLR,CLR_DB,Din,OSC : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (2 downto 0));
8 end ex4_3_1vcxl;
9
10 architecture Behavioral of ex4_3_1vcxl is
11     component ch4vcxl_DEBOUNCER
12         Port ( OSC,D_in,CLR : in STD_LOGIC;
13                 D_out : out STD_LOGIC);
14     end component;
15     component ch4vcxl_ripple_c8_up
16         Port ( Clock,Clear : in STD_LOGIC;
17                 Q: out STD_LOGIC_VECTOR (2 downto 0));
18     end component;
19     signal C : STD_LOGIC;
20 begin
21     DEBOUNCER : ch4vcxl_DEBOUNCER
22         port map( OSC=>OSC,D_in=>Din,CLR=>CLR_DB,D_out=>C);
23     COUNTER3BIT : ch4vcxl_ripple_c8_up
24         port map( Clock=>C,Clear=>CLR,Q=>Q);
25 end Behavioral;

```

รูปที่ L1.5 โภค VHDL ของวงจรนับ 3 บิตแบบนับขึ้นที่รวมวงจรสีเบาช์หรือบ่ายเบ้าช์ไว้แล้ว



รูปที่ L1.6 ผลการสังเคราะห์วงจร (View technology schematic)

การกำหนดขาสัญญาณต่างๆ จะใช้อสัชโนเดตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB3(Slide SW1) เป็นอินพุต และ LED2-LED4 เป็นเอาต์พุต กล่าวคือ

Din = PB1 = INPUT = p39	Q(2) = LED2 = OUTPUT = p37
CLR_DB = PB2 = INPUT = p40	Q(1) = LED3 = OUTPUT = p36
OSC = OSC = OUTPUT = p5	Q(0) = LED4 = OUTPUT = p35
CLR = PB3(Slide SW1) = INPUT = p42	

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
CLR	Input	p42		2		14
CLR_DB	Input	p40		2		11
Din	Input	p39		2		9
OSC	Input	p5		1		9
Q<2>	Output	p37		2	SLOW	
Q<1>	Output	p36		2	SLOW	
Q<0>	Output	p35		2	SLOW	

รูปที่ L1.7 Assign Package Pins

หลังจากโปรแกรมจะที่ออกแบบลงชิป CPLD แล้วเลื่อน Slide SW1 ไปที่ตำแหน่ง ON (CLR= '0') และให้กดลง กดปุ่ม PB1 และ PB2 สลับกันไปเรื่อยๆ แล้วให้สังเกตดูผลที่ LED2-LED4 ว่าติดสว่างโดยให้ออกเอาต์พุตเป็นไปตามทฤษฎี หรือไม่ หากนั้นเลื่อน Slide SW1 ไปที่ตำแหน่ง OFF (CLR= '1') แล้ว ON แล้วกดลงซ้ำ หากนั้นจึงบันทึกผลการทำงาน

2 สร้างวงจรนับขีนแบบริบเบลด้วย FPGA

2.1) เขียนโค้ดวงจรนับขีนแบบซิงโครนัสนาฬิกา 3 บิต โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ch4vf_ripple_c8_up จากนั้นจึงเขียนโค้ดใหม่โดยคัดลอกโค้ดในรูปที่ 4.12 และแก้ไขชื่อ entity เป็น ch4vf_ripple_c8_up และให้คัดลอกโค้ด T Flip-Flop แบบซิงโครนัสเกลียร์ในรูปที่ 4.11 ทั้งหมดไปแทรกไว้ด้านบนสุดจะได้ ดังรูปที่ L2.1 ซึ่งเป็นการเขียนโค้ดของ Component รวมไว้ในไฟล์โค้ดหลัก (ถือเป็นการ Add Source ของ Component โดยอัตโนมัติ) เตรียมแล้วตรวจสอบความถูกต้องและบันทึกไฟล์ จากนั้นจึงทำ Behavioral simulation โดยใช้ Source File ชื่อ ch4vf_ripple_c8_up_tb1 และกำหนดค่าต่างๆ ดังรูปที่ L2.2 และ Waveform ดังรูปที่ L2.3 และคุณผลจำลองการทำงานในรูปที่ L2.4 ว่าเป็นไปตามทฤษฎีหรือไม่

```

2 -----Component:T_Flip-Flop-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity TFF is
7     Port ( C,CLR,T : in STD_LOGIC;
8             Q : out STD_LOGIC);
9 end TFF;
10
11 architecture Behavioral of TFF is
12     signal QT : STD_LOGIC;
13 begin
14     process(C,CLR)
15         begin

```

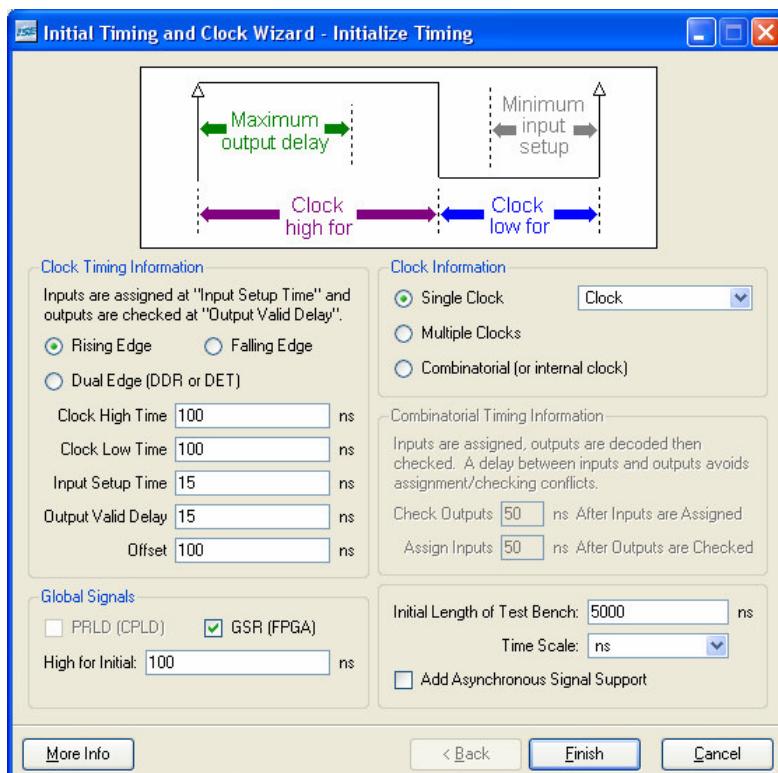
(ต่อ)

```

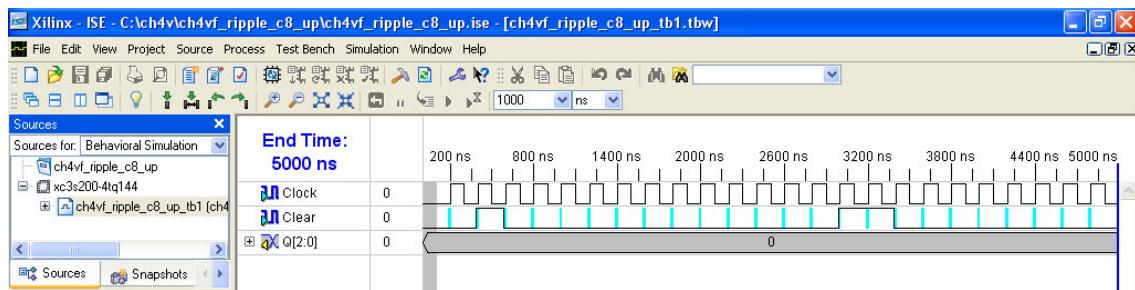
16      if CLR='1' then QT <= '0';
17      elsif (C'event and C='1') then
18          if T='0' then QT <= QT;
19          elsif T='1' then QT <= not QT;
20          end if;
21      end if;
22  end process;
23  Q <= QT;
24 end Behavioral;
25
26 -----Main code : Ripple counter-----
27 library IEEE;
28 use IEEE.STD_LOGIC_1164.ALL;
29
30 entity ch4vf_ripple_c8_up is
31     Port ( Clock, Clear : in STD_LOGIC;
32             Q: out STD_LOGIC_VECTOR (2 downto 0));
33 end ch4vf_ripple_c8_up;
34
35 architecture Behavioral of ch4vf_ripple_c8_up is
36     signal C1,C2 : STD_LOGIC;
37     signal QT : STD_LOGIC_VECTOR (2 downto 0);
38     component TFF
39         Port ( C,CLR,T : in STD_LOGIC;
40                 Q : out STD_LOGIC);
41     end component;
42 begin
43 BIT_0 : TFF port map( C=>Clock,CLR=>Clear,T=>'1',Q=>QT(0));
44     C1 <= not QT(0);
45 BIT_1 : TFF port map( C=>C1,CLR=>Clear,T=>'1',Q=>QT(1));
46     C2 <= not QT(1);
47 BIT_2 : TFF port map( C=>C2,CLR=>Clear,T=>'1',Q=>QT(2));
48     Q <= QT;
49 end Behavioral;

```

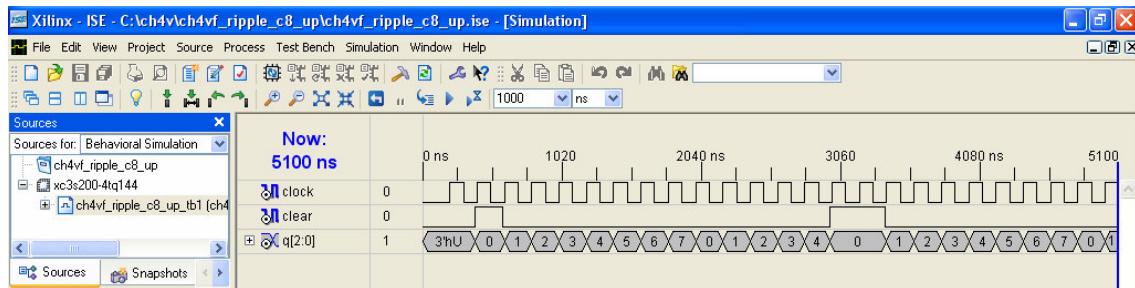
รูปที่ L2.1 โค้ดวงจรนับขีนแบบอะซิงไครน์ส่วนดาด 3 บิตที่เขียนรวมโค้ดของ T Flip-Flop ที่เป็น Component ไว้แล้ว



รูปที่ L2.2 หน้าต่าง Initial Timing and Clock Wizard



รูปที่ L2.3 หน้าต่างสำหรับกำหนดสัญญาณต่างๆที่ป้อนให้กับอินพุตของวงจรที่เราต้องการทดสอบ



รูปที่ L2.4 ผล Behavioral simulation ของวงจรนับ 3 บิตแบบนับขึ้น

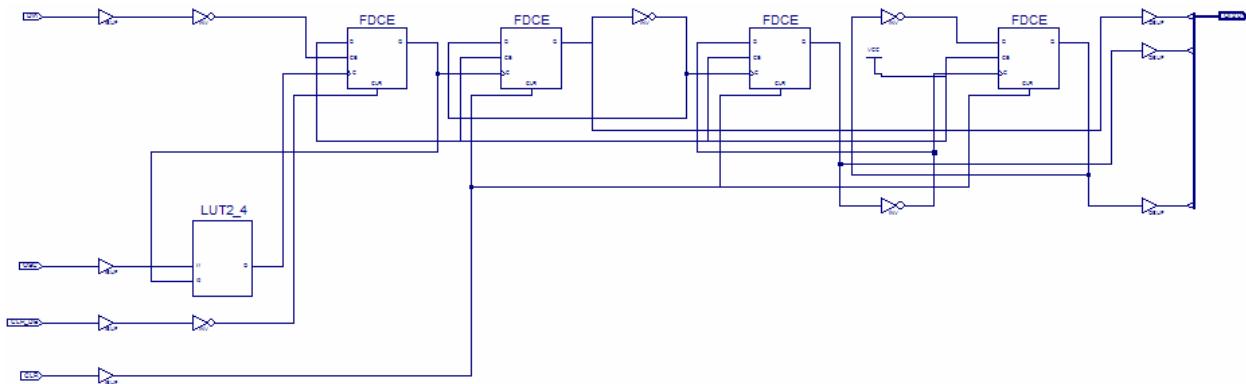
2.2) ในทางปฏิบัติ ผลการทดลองของวงจรนับจะไม่ตรงกับผลจำลองการทำงานเนื่องจากการเกิดเบซ์ในขณะกดปุ่ม การแก้ไขปัญหานี้ทำได้โดยเพิ่นโค้ดวงจรดีเบาเซอร์ไว้ในโค้ดของวงจรนับ โดยสร้างไฟล์ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_3_1vf จากนั้นนำไฟล์วงจรดีเบาเซอร์ชื่อ ch4vf_DEBOUNCER และไฟล์วงจรนับขึ้นชื่อ ch4vf_ripple_c8_up ที่มีอยู่แล้วมาทำเป็น Component แล้ว Add Source ของ Component ทั้ง 2 ไฟล์เข้าไปในไฟล์ Project โดยโค้ดที่ได้แสดงดังรูปที่ L2.5 และผลการสังเคราะห์วงจร (View technology schematic) แสดงดังรูปที่ L2.6

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_3_1vf is
6     Port ( CLR,CLR_DB,Din,OSC : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (2 downto 0));
8 end ex4_3_1vf;
9
10 architecture Behavioral of ex4_3_1vf is
11     component ch4vf_DEBOUNCER
12         Port ( OSC,D_in,CLR : in STD_LOGIC;
13                 D_out : out STD_LOGIC);
14     end component;
15     component ch4vf_ripple_c8_up
16         Port ( Clock,Clear : in STD_LOGIC;
17                 Q: out STD_LOGIC_VECTOR (2 downto 0));
18     end component;
19     signal C : STD_LOGIC;
20 begin
21     DEBOUNCER : ch4vf_DEBOUNCER
22         port map( OSC=>OSC,D_in=>Din,CLR=>CLR_DB,D_out=>C);
23     COUNTER3BIT : ch4vf_ripple_c8_up
24         port map( Clock=>C,Clear=>CLR,Q=>Q);
25 end Behavioral;

```

รูปที่ L2.5 โค้ด VHDL ของวงจรนับ 3 บิตแบบนับขึ้นที่รวมวงจรดีเบาเซอร์อย่างง่ายไว้แล้ว



รูปที่ L2.6 ผลการสังเคราะห์วงจร (View technology schematic)

การกำหนดขาสัญญาณต่างๆ จะใช้อสัตโนห์ OSC = 25 MHz และปุ่มกด PB1-PB2 เป็นอินพุต และ LED L0-L2 เป็นเอาต์พุต กล่าวคือ

$$\text{Din} = \text{PB1} = \text{INPUT} = \text{p44} \quad Q(2) = \text{L2} = \text{OUTPUT} = \text{p69}$$

$$\text{CLR_DB} = \text{PB2} = \text{INPUT} = \text{p46} \quad Q(1) = \text{L1} = \text{OUTPUT} = \text{p77}$$

$$\text{OSC} = \text{OSC} = \text{INPUT} = \text{p127} \quad Q(0) = \text{L0} = \text{OUTPUT} = \text{p70}$$

$$\text{CLR} = \text{Dip SW1} = \text{INPUT} = \text{p52}$$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLR	Input	p52	BANK LVC MOS33	N/A	3.30						Unknown		
CLR_DB	Input	p46	BANK LVC MOS33	N/A	3.30						Unknown		
Din	Input	p44	BANK LVC MOS33	N/A	3.30						Unknown		
OSC	Input	p127	BANK LVC MOS33	N/A	3.30						Unknown		
Q<0>	Output	p70	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
Q<1>	Output	p77	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
Q<2>	Output	p69	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		

รูปที่ L2.7 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิป FPGA แล้วเลื่อน Dip SW1 ไปที่ตำแหน่ง ON (CLR= '0') และให้ทดลองกดปุ่ม PB1 และ PB2 ลับกันไปเรื่อยๆ แล้วให้สังเกตดูผลที่ LED L0-L2 ว่าติดสว่างโดยให้ลองเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นเลื่อน Dip SW1 ไปที่ตำแหน่ง OFF (CLR= '1') แล้ว ON แล้วทดลองซ้ำ จากนั้นจึงบันทึกผลการทดลอง

การทดลองที่ 4.3.2 การสร้างวงจรนับลงแบบริปเปิล

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานและออกแบบวงจรนับลงแบบริปเปิล
- 2) เพื่อสร้างวงจรนับลงแบบริปเปิลโดยใช้ VHDL และโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรนับลงแบบริปเปิลด้วย CPLD

การทดลองนี้จะเหมือนการทดลองที่ 4.3.1 ทุกประการ แต่เป็นการทดลองของวงจรนับลง โดยนำโค้ดของวงจรนับลงในรูปที่ 4.16 มาเขียนใหม่ (แก้ไขชื่อ entity เป็น ch4vcxl_ripple_c8_dn) โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ch4vcxl_ripple_c8_dn เมื่อเขียนโค้ดของวงจรนับแล้วให้เขียนโค้ดของ T Flip-Flop แบบอะซิงโกรานส์เกลียร์แทรกไว้ด้านสุดของไฟล์โค้ดของวงจรนับเพื่อให้รวมไฟล์เดียวกัน ตรวจสอบความถูกต้องแล้วบันทึกไฟล์ แล้วทำการ Behavioral simulation เช่นเดียวกับการทดลองที่ 4.3.1 โดยใช้ Source File ชื่อ ch4vcxl_ripple_c8_dn_tb1 แล้วให้คุณทดลองการทำงานว่าเป็นไปตามทฤษฎีหรือไม่

ในการทดลองนี้ทำได้โดยเขียนโค้ดของวงจรด้วยภาษา Verilog ในไฟล์ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_3_2vcxl นำไฟล์ลงจรดเบาเซอร์ชื่อ ch4vcxl_DEBOUNCER และไฟล์ของวงจรนับลงชื่อ ch4vcxl_ripple_c8_dn ที่มีอยู่แล้วมาทำเป็น Component และ Add Source ของ Component ทั้ง 2 ไฟล์เข้าไปในไฟล์ Project จากนั้นทำการกำหนดขาสัญญาณต่างๆ ให้ CPLD และทดลองเช่นเดียวกับการทดลองที่ 4.3.1 ทุกประการ

2 สร้างวงจรนับลงแบบริปเปิลด้วย FPGA

การทดลองนี้จะเหมือนการทดลองที่ 4.3.1 ทุกประการ แต่เป็นการทดลองของวงจรนับลง โดยนำโค้ดของวงจรนับลงในรูปที่ 4.16 มาเขียนใหม่ (แก้ไขชื่อ entity เป็น ch4vf_ripple_c8_dn) โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ch4vf_ripple_c8_dn เมื่อเขียนโค้ดของวงจรนับแล้วให้เขียนโค้ดของ T Flip-Flop แบบอะซิงโกรานส์เกลียร์แทรกไว้ด้านสุดของไฟล์โค้ดของวงจรนับเพื่อให้รวมไฟล์เดียวกัน ตรวจสอบความถูกต้องแล้วบันทึกไฟล์ จากนั้นทำการ Behavioral simulation เช่นเดียวกับการทดลองที่ 4.3.1 โดยใช้ Source File ชื่อ ch4vf_ripple_c8_dn_tb1 แล้วให้คุณทดลองการทำงานว่าเป็นไปตามทฤษฎีหรือไม่

ในการทดลองนี้ทำได้โดยเขียนโค้ดของวงจรด้วยภาษา Verilog ในไฟล์ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_3_2vf จากนั้นนำไฟล์ลงจรดเบาเซอร์ชื่อ ch4vf_DEBOUNCER และไฟล์ของวงจรนับลงชื่อ ch4vf_ripple_c8_dn ที่มีอยู่แล้วมาทำเป็น Component และ Add Source ของ Component ทั้ง 2 ไฟล์เข้าไปในไฟล์ Project จากนั้นทำการกำหนดขาสัญญาณต่างๆ ให้ FPGA และทดลองเช่นเดียวกับการทดลองที่ 4.3.1 ทุกประการ

การทดลองที่ 4.3.3 การสร้างวงจรนับขีนแบบซิงโครนัส

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานและออกแบบวงจรนับขีนแบบซิงโครนัส
- 2) เพื่อสร้างวงจรนับขีนแบบซิงโครนัสโดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรนับขีนซิงโครนัสด้วย CPLD

1.1) เขียนโค้ดวงจรนับขีนแบบซิงโครนัสขนาด 3 บิตดังรูปที่ L1.1 โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ch4vcxl_sync_c8_up จากนั้นจึงตรวจสอบความถูกต้องและบันทึกไฟล์ แล้วทำการ Simulation เช่นเดียวกับการทดลองที่ 4.3.1 โดยใช้ Source File ชื่อ ch4vcxl_sync_c8_up_tb1 และให้คุณว่าเป็นไปตามทฤษฎีหรือไม่

```

2 -----Component:T Flip-Flop-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity TFF is
7     Port ( C,CLR,T : in STD_LOGIC;
8             Q : out STD_LOGIC);
9 end TFF;
10
11 architecture Behavioral of TFF is
12     signal QT : STD_LOGIC;
13 begin
14 process(C,CLR)
15     begin
16         if CLR='1' then QT <= '0';
17         elsif (C'event and C='1') then
18             if      T='0' then QT <= QT;
19             elsif T='1' then QT <= not QT;
20             end if;
21         end if;
22     end process;
23     Q <= QT;
24 end Behavioral;
25
26 -----Main code : Synchronous counter-----
27 library IEEE;
28 use IEEE.STD_LOGIC_1164.ALL;
29
30 entity ch4vcxl_sync_c8_up is
31     Port ( Clock,Clear : in STD_LOGIC;
32             Q: out STD_LOGIC_VECTOR (2 downto 0));
33 end ch4vcxl_sync_c8_up;
34
35 architecture Behavioral of ch4vcxl_sync_c8_up is
36     signal QT,Tin : STD_LOGIC_VECTOR (2 downto 0);
37     component TFF
38         Port ( C,CLR,T : in STD_LOGIC;
39                 Q : out STD_LOGIC);
40     end component;

```

```

41 begin
42     Tin(0) <= '1';
43     BIT_0 : TFF port map( C=>Clock,CLR=>Clear,T=>Tin(0),Q=>QT(0));
44     Tin(1) <= QT(0);
45     BIT_1 : TFF port map( C=>Clock,CLR=>Clear,T=>Tin(1),Q=>QT(1));
46     Tin(2) <= QT(1) and Tin(1);
47     BIT_2 : TFF port map( C=>Clock,CLR=>Clear,T=>Tin(2),Q=>QT(2));
48     Q <= QT;
49 end Behavioral;

```

รูปที่ L1.1 โค้ดวงจรนับขึ้นแบบซิงโครนัสนาด 3 บิตที่เขียนรวมโค้ดของ T Flip-Flop ที่เป็น Component ไว้แล้ว

1.2) ในทางปฏิบัตินี้ผลการทดลองของวงจรนับจะไม่ตรงกับผลจำลองการทำงานเนื่องจากเกิดเบ้าช์ในขณะกดปุ่ม การแก้ไขปัญหานี้ทำได้โดยเขียนโค้ดวงจรคีเบาเซอร์ไว้ในโค้ดของวงจรนับ โดยสร้างไฟล์ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_3_3vcx1 จากนั้นนำไฟล์วงจรคีเบาเซอร์ชื่อ ch4vcx1_DEBOUNCER และไฟล์วงจรนับขึ้นชื่อ ch4vcx1_sync_c8_up ที่มีอยู่แล้วมาทำเป็น Component และ Add Source ของ Component ทั้ง 2 ไฟล์เข้าไปในไฟล์ Project โดยโค้ดที่ได้แสดงดังรูปที่ L1.2

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_3_3vcx1 is
6     Port ( CLR,CLR_DB,Din,OSC : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (2 downto 0));
8 end ex4_3_3vcx1;
9
10 architecture Behavioral of ex4_3_3vcx1 is
11     component ch4vcx1_DEBOUNCER
12         Port ( OSC,D_in,CLR : in STD_LOGIC;
13                 D_out : out STD_LOGIC);
14     end component;
15     component ch4vcx1_sync_c8_up
16         Port ( Clock,Clear : in STD_LOGIC;
17                 Q: out STD_LOGIC_VECTOR (2 downto 0));
18     end component;
19     signal C : STD_LOGIC;
20 begin
21     DEBOUNCER : ch4vcx1_DEBOUNCER
22         port map( OSC=>OSC,D_in=>Din,CLR=>CLR_DB,D_out=>C);
23     COUNTER3BIT : ch4vcx1_sync_c8_up
24         port map( Clock=>C,Clear=>Clear,Q=>Q);
25 end Behavioral;

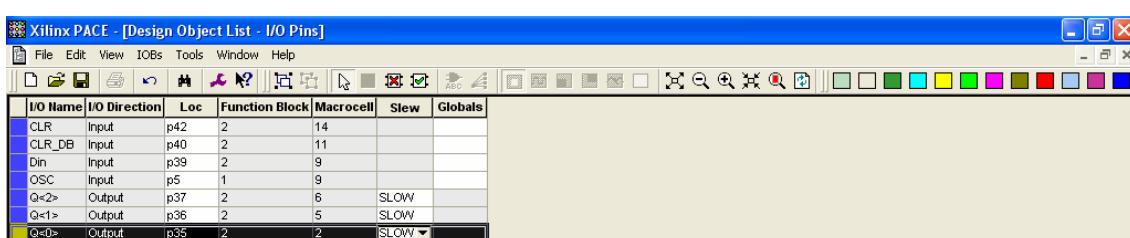
```

รูปที่ L1.2 โค้ด VHDL ของวงจรนับ 3 บิตแบบนับขึ้นที่รวมวงจรคีเบาเซอร์อย่างง่ายไว้แล้ว

การทำหน้าตาสัญญาณต่างๆ ให้ OSC = 32.768 kHz และ PB1-PB3 เป็นอินพุตและ LED2-LED4 เป็นเอาต์พุต กล่าวคือ

Din = PB1 = INPUT = p39 OSC = OSC = OUTPUT = p5 Q(2) = LED2 = OUTPUT = p37
 CLR_DB = PB2 = INPUT = p40 CLR = PB3(Slide SW1) = INPUT = p42 Q(1) = LED3 = OUTPUT = p36
Q(0) = LED4 = OUTPUT = p35

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



รูปที่ L1.3 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิป CPLD แล้วเลื่อน Slide SW1 ไปที่ตำแหน่ง ON (CLR= '0') และให้ทดลอง กดปุ่ม PB1 และ PB2 ลับกันไปเรื่อยๆ แล้วให้สังเกตคุณลักษณะ LED2-LED4 ว่าติดสว่างโดยให้ออกผลเป็นไปตามทฤษฎี หรือไม่ จากนั้นเลื่อน Slide SW1 ไปที่ตำแหน่ง OFF (CLR= '1') และ ON แล้วทดลองซ้ำ จากนั้นจึงบันทึกผลการทดลอง

2 สร้างวงจรนับขึ้นแบบซิงโครนัสด้วย FPGA

2.1) เขียนโค้ดควบคุมวงจรนับขึ้นแบบซิงโครนัสขนาด 3 บิตดังรูปที่ L2.1 โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ch4vf_sync_c8_up เตรียมแล้วตรวจสอบความถูกต้องและบันทึกไฟล์ จากนั้นทำ Simulation เช่นเดียวกับการทดลองที่ 4.3.1 โดยใช้ Source File ชื่อ ch4vf_sync_c8_up_tb1 และให้คุณลักษณะเป็นไปตามทฤษฎีหรือไม่

```

2 -----Component: T Flip-Flop-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity TFF is
7     Port ( C,CLR,T : in STD_LOGIC;
8             Q : out STD_LOGIC);
9 end TFF;
10
11 architecture Behavioral of TFF is
12     signal QT : STD_LOGIC;
13 begin
14 process(C,CLR)
15     begin
16         if CLR='1' then QT <= '0';
17         elsif (C'event and C='1') then
18             if      T='0' then QT <= QT;
19             elsif T='1' then QT <= not QT;
20             end if;
21         end if;
22     end process;
23     Q <= QT;
24 end Behavioral;
25
26 -----Main code : Synchronous counter-----
27 library IEEE;
28 use IEEE.STD_LOGIC_1164.ALL;
29
30 entity ch4vf_sync_c8_up is
31     Port ( Clock,Clear : in STD_LOGIC;
32             Q: out STD_LOGIC_VECTOR (2 downto 0));
33 end ch4vf_sync_c8_up;
34
35 architecture Behavioral of ch4vf_sync_c8_up is
36     signal QT,Tin : STD_LOGIC_VECTOR (2 downto 0);
37     component TFF
38         Port ( C,CLR,T : in STD_LOGIC;
39                 Q : out STD_LOGIC);
40     end component;
41 begin
42     Tin(0) <= '1';
43     BIT_0 : TFF port map( C=>Clock,CLR=>Clear,T=>Tin(0),Q=>QT(0));
44     Tin(1) <= QT(0);
45     BIT_1 : TFF port map( C=>Clock,CLR=>Clear,T=>Tin(1),Q=>QT(1));
46     Tin(2) <= QT(1) and Tin(1);
47     BIT_2 : TFF port map( C=>Clock,CLR=>Clear,T=>Tin(2),Q=>QT(2));
48     Q <= QT;
49 end Behavioral;

```

รูปที่ L2.1 โค้ดควบคุมวงจรนับขึ้นแบบซิงโครนัสขนาด 3 บิตที่เขียนรวมโค้ดของ T Flip-Flop ที่เป็น Component ไว้แล้ว

2.2) ในทางปฏิบัตินั้นผลการทดลองของวงจรนับจะไม่ตรงกับผลจำลองการทำงานเนื่องจากเกิดเบ้าซึ่นบนคุณภาพ นี้ทำได้โดยเขียนโค้ดควบคุมคีบนาฬิกาไว้ในโค้ดของวงจรนับ โดยสร้างไฟล์ใน Project Location ชื่อ ch4v และกำหนด Project

Name และ Source File ชื่อ ex4_3_3vf จากนั้นนำไฟล์วงจรดีเบาเซอร์ ชื่อ ch4vf_DEBOUNCER และไฟล์วงจรนับขึ้นชื่อ ch4vf_sync_c8_up ที่มีอยู่แล้วมาทำเป็น Component แล้ว Add Source ของ Component ทั้ง 2 ไฟล์เข้าไปในไฟล์ Project โดย ก็ตดที่ได้แสดงดังรูปที่ L2.2

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_3_3vf is
6     Port ( CLR,CLR_DB,Din,OSC : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (2 downto 0));
8 end ex4_3_3vf;
9
10 architecture Behavioral of ex4_3_3vf is
11     component ch4vf_DEBOUNCER
12         Port ( OSC,D_in,CLR : in STD_LOGIC;
13                 D_out : out STD_LOGIC);
14     end component;
15     component ch4vf_sync_c8_up
16         Port ( Clock,Clear : in STD_LOGIC;
17                 Q: out STD_LOGIC_VECTOR (2 downto 0));
18     end component;
19     signal C : STD_LOGIC;
20 begin
21 DEBOUNCER : ch4vf_DEBOUNCER
22     port map( OSC=>OSC,D_in=>Din,CLR=>CLR_DB,D_out=>C);
23 COUNTER3BIT : ch4vf_sync_c8_up
24     port map( Clock=>C,Clear=>CLR,Q=>Q);
25 end Behavioral;
```

รูปที่ L2.2 ก็ตด VHDL ของวงจรนับ 3 บิตแบบนับขึ้นที่รวมวงจรดีเบาเซอร์อย่างง่ายไว้แล้ว

การกำหนดขาสัญญาณต่างๆ ให้ OSC = 25 MHz และ PB1-PB2 เป็นอินพุต และมี LED L0-L2 เป็นเอาต์พุต กล่าวคือ

Din = PB1 = INPUT = p44 OSC = OSC = INPUT= p127 Q(2) = L2 = OUTPUT = p69

CLR_DB = PB2 = INPUT = p46 CLR = Dip SW1 = INPUT= p52 Q(1) = L1 = OUTPUT= p77

Q(0) = L0 = OUTPUT= p70

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLR	Input	p52	BANK LVC MOS33	N/A	3.30						Unknown		
CLR_DB	Input	p46	BANK LVC MOS33	N/A	3.30						Unknown		
Din	Input	p44	BANK LVC MOS33	N/A	3.30						Unknown		
OSC	Input	p127	BANK LVC MOS33	N/A	3.30						Unknown		
Q<0>	Output	p70	BANK LVC MOS33	N/A	3.30				SLOW		Unknown		
Q<1>	Output	p77	BANK LVC MOS33	N/A	3.30				SLOW		Unknown		
Q<2>	Output	p69	BANK LVC MOS33	N/A	3.30				SLOW		Unknown		

รูปที่ L2.3 Assign Package Pins

หลังจากโปรแกรมจะที่ออกแบบบล็อก FPGA และเลือก Dip SW1 ไปที่ตำแหน่ง ON (CLR= '0') และให้กดลงกดปุ่ม PB1 และ PB2 ลับกันไปเรื่อยๆ แล้วให้สังเกตคุณลักษณะ LED L2-L4 ว่าคิดสว่างโดยให้กดอิจิกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นเลือก Dip SW1 ไปที่ตำแหน่ง OFF (CLR= '1') และ ON แล้วทดลองซ้ำ จากนั้นจึงบันทึกผลการทดลอง

การทดลองที่ 4.3.4 การสร้างวงจรนับลงแบบซิงโครนัส

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานและออกแบบวงจรนับลงแบบซิงโครนัส
- 2) เพื่อสร้างวงจรนับลงแบบซิงโครนัสโดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรนับลงแบบซิงโครนัสด้วย CPLD

การทดลองนี้จะเนื่องจากการทดลองที่ 4.3.3 ทุกประการ แต่เป็นการทดลองวงจรนับลง 3 บิต โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ch4vcxl_sync_c8_dn จากนั้นจึงเขียนโค้ดใหม่ ทำงานองเดียวกับโค้ดในรูปที่ 4.23 แต่เป็นวงจรนับ 3 บิตและแก้ไขชื่อ entity เป็น ch4vcxl_sync_c8_dn แล้วให้กัดลอกโค้ด T Flip-Flop แบบซิงโครนัสเคลียร์ในรูปที่ 4.11 ทั้งหมดไปแทรกไว้ด้านบนสุด ตรวจสอบความถูกต้องแล้วบันทึกไฟล์ จากนั้นทำ Behavioral simulation เช่นเดียวกับการทดลองที่ 4.3.3 โดยใช้ Source File ชื่อ ch4vcxl_sync_c8_dn_tb1 แล้วให้คุณทดลองการทำงานว่าเป็นไปตามทฤษฎีหรือไม่

ในทางปฏิบัตินั้นผลการทดลองวงจรนับจะไม่ตรงกับผลจำลองการทำงานเนื่องจากเกิดเบ้าช์ในขณะกดปุ่ม การแก้ไขปัญหานี้ทำได้โดยเขียนโค้ดวงจรดีเบาเซอร์ไว้ในโค้ดของวงจรนับ โดยสร้างไฟล์ใน Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_3_4vcxl จากนั้นนำไฟล์วงจรดีเบาเซอร์ ชื่อ ch4vcxl_DEBOUNCER และไฟล์วงจรนับลงชื่อ ch4vcxl_sync_c8_dn ที่มีอยู่แล้วมาทำเป็น Component แล้ว Add Source ของ Component ทั้ง 2 ไฟล์เข้าไปในไฟล์ Project จากนั้นทำการกำหนดขาสัญญาณต่างๆ ให้ CPLD และทดลองเช่นเดียวกับการทดลองที่ 4.3.1 ทุกประการ

2 สร้างวงจรนับลงแบบซิงโครนัสด้วย FPGA

การทดลองนี้จะเนื่องจากการทดลองที่ 4.3.3 ทุกประการ แต่เป็นการทดลองวงจรนับลง 3 บิต โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ch4vf_sync_c8_dn จากนั้นจึงเขียนโค้ดใหม่ ทำงานองเดียวกับโค้ดในรูปที่ 4.23 แต่เป็นวงจรนับ 3 บิตและแก้ไขชื่อ entity เป็น ch4vf_sync_c8_dn แล้วให้กัดลอกโค้ด T Flip-Flop แบบซิงโครนัสเคลียร์ในรูปที่ 4.11 ทั้งหมดไปแทรกไว้ด้านบนสุด ตรวจสอบความถูกต้องแล้วบันทึกไฟล์ จากนั้นทำ Behavioral simulation เช่นเดียวกับการทดลองที่ 4.3.3 โดยใช้ Source File ชื่อ ch4vf_sync_c8_dn_tb1 แล้วให้คุณทดลองการทำงานว่าเป็นไปตามทฤษฎีหรือไม่

ในทางปฏิบัตินั้นผลการทดลองวงจรนับจะไม่ตรงกับผลจำลองการทำงานเนื่องจากเกิดเบ้าช์ในขณะกดปุ่ม การแก้ไขปัญหานี้ทำได้โดยเขียนโค้ดวงจรดีเบาเซอร์ไว้ในโค้ดของวงจรนับ โดยสร้างไฟล์ใน Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_3_4vf จากนั้นนำไฟล์วงจรดีเบาเซอร์ ชื่อ ch4vf_DEBOUNCER และไฟล์วงจรนับลงชื่อ ch4vf_sync_c8_dn ที่มีอยู่แล้วมาทำเป็น Component แล้ว Add Source ของ Component ทั้ง 2 ไฟล์เข้าไปในไฟล์ Project จากนั้นทำการกำหนดขาสัญญาณต่างๆ ให้ FPGA และทดลองเช่นเดียวกับการทดลองที่ 4.3.1 ทุกประการ

การทดลองที่ 4.3.5 ใช้ความสามารถของซอฟต์แวร์ทุลช่วยในการออกแบบวงจรนับ

วัตถุประสงค์

- 1) เพื่อฝึกการใช้ Signal และ Variable ใน การออกแบบวงจรนับ
- 2) เพื่อฝึกการเรียกใช้ Std_logic_unsigned package หรือการเรียกใช้ Numeric_std package
- 3) เพื่อสร้างวงจรนับแบบซิงโครนัสที่เขียนโค้ดด้วย VHDL และโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรนับแบบซิงโครนัสด้วย CPLD

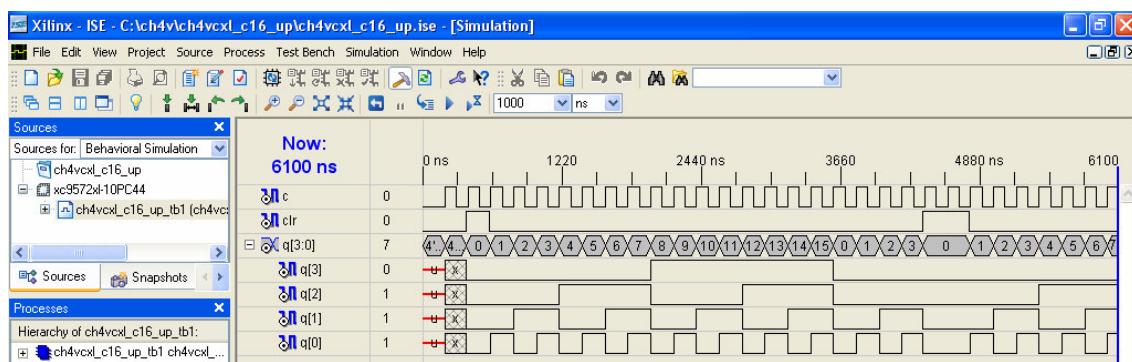
1.1) เขียนโค้ดวงจรนับขึ้นแบบซิงโครนัส 4 บิตที่เขียนโดยการประกาศใช้ Signal โดยนำในตัวอย่างที่ 2.26 โค้ดในรูปที่ 2.46c มาแก้ไขและสร้างไฟล์ Component โดยไว้ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ch4vcxl_c16_up เสร็จแล้วจะได้ดังรูปที่ L1.1 ทำการ Check syntax และบันทึกไฟล์ จากนั้นทำ Behavioral simulation โดยใช้ Source File ชื่อ ch4vcxl_c16_up_tb1 และวัดผลจำลองการทำงานดังรูปที่ L1.2 ว่าเป็นไปตามทฤษฎีหรือไม่

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ch4vcxl_c16_up is
7     port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ch4vcxl_c16_up;
10
11 architecture Behavioral of ch4vcxl_c16_up 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;

```

รูปที่ L1.1 โค้ดวงจรนับขึ้นแบบซิงโครนัส 4 บิตที่เขียนโดยการประกาศใช้ Signal



รูปที่ L1.2 ผลจำลองการทำงานของโค้ดในรูปที่ L1.1

1.2) ทำการทดสอบวงจรนับขึ้นแบบซิง ไครนัส 4 บิตโดยสร้างไว้ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_3_5vcx1 โดยโค้ดที่ได้แสดงดังรูปที่ L1.3 แล้วนำไฟล์วงจรคีเบาเซอร์ชื่อ ch4vcx1_DEBOUNCER และไฟล์วงจรนับชื่อ ch4vcx1_c16_up มาทำเป็น Component แล้ว Add Source ของ Component ทั้ง 2 ไฟล์เข้าไปในไฟล์ Project

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_3_5vcx1 is
6     Port ( CLR,CLR_DB,Din,OSC : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end ex4_3_5vcx1;
9
10 architecture Behavioral of ex4_3_5vcx1 is
11 component ch4vcx1_DEBOUNCER
12     Port ( OSC,D_in,CLR : in STD_LOGIC;
13             D_out : out STD_LOGIC);
14 end component;
15 component ch4vcx1_c16_up
16     Port ( C,CLR : in STD_LOGIC;
17             Q : out STD_LOGIC_VECTOR (3 downto 0));
18 end component;
19     signal C : STD_LOGIC;
20     signal QT : STD_LOGIC_VECTOR (3 downto 0);
21 begin
22     -----
23     DEBOUNCER : ch4vcx1_DEBOUNCER
24         port map( OSC=>OSC,D_in=>Din,CLR=>CLR_DB,D_out=>C);
25     -----
26     COUNTER4BIT : ch4vcx1_c16_up
27         port map( C=>C,CLR=>CLR,Q=>Q);
28 end Behavioral;

```

รูปที่ L1.3 โค้ด VHDL ของวงจรนับขึ้น (แบบซิงไครนัส) 4 บิตที่รวมวงจรคีเบาเซอร์ย่างง่ายไว้แล้ว

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB3 (Slide SW1) เป็นอินพุต และ LED1-LED4 เป็นเอาต์พุต กล่าวว่าคือ

Din = PB1 = INPUT = p39	Q(3) = LED1 = OUTPUT = p38
CLR_DB= PB2 = INPUT = p40	Q(2) = LED2 = OUTPUT = p37
OSC = OSC = OUTPUT = p5	Q(1) = LED3 = OUTPUT = p36
CLR = PB3(Slide SW1)= INPUT = p42	Q(0) = LED4 = OUTPUT = p35

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]						
I/O Name	I/O Direction	Loc	Function Block	Macrocell	Stew	Globals
CLR	Input	p42	2	14		
CLR_DB	Input	p40	2	11		
Din	Input	p39	2	9		
OSC	Input	p5	1	9		
Q<3>	Output	p38	2	8		
Q<2>	Output	p37	2	6	SLOW	
Q<1>	Output	p36	2	5	SLOW	
Q<0>	Output	p35	2	2	SLOW	

รูปที่ L1.4 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิป CPLD และเลื่อน Slide SW1 ไปที่ตำแหน่ง ON (CLR= '0') และให้กดลง กดปุ่ม PB1 และ PB2 ลับกันไปเรื่อยๆ แล้วให้สังเกตผลที่ LED1-LED4 ว่าติดสว่างโดยให้ออกเอาต์พุตเป็นไปตามทฤษฎี หรือไม่ หากนั้นเลื่อน Slide SW1 ไปที่ตำแหน่ง OFF (CLR= '1') และ ON แล้วทดลองซ้ำ แล้วจึงบันทึกผลการทดลอง

1.3) ทำการแก้ไขโค้ดที่ใช้ทำ Component ในรูปที่ L1.1 โดยเปิดไฟล์ที่อยู่ใน Project Location ชื่อ ch4v ที่มี Project Name และ Source File ชื่อ ch4vcxl_c16_up จากนั้นแก้ไขโค้ดของรับข้อมูลแบบซิงโกรนัส 4 บิตที่เขียนโดยการประกาศใช้ Variable ดังรูปที่ L1.5 แล้ว Check syntax และบันทึกไฟล์ชื่อเดิม ทำการ Behavioral simulation ชี้อีกครั้งว่าให้ผลเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงทำการทดลองซ้ำ เสร็จแล้วให้แก้ไข Component โค้ดของรับข้อมูลแบบซิงโกรนัส 4 บิตอีกครั้ง โดยใช้โค้ดในตัวอย่างที่ 2.26 ในรูปที่ 2.46e) และรูปที่ 2.46f) ตามลำดับ จากนั้นจึงทำการทดลองซ้ำจนครบถ้วน

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ch4vcxl_c16_up is
7     port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ch4vcxl_c16_up;
10
11 architecture Behavioral of ch4vcxl_c16_up 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;

```

รูปที่ L1.5 โค้ดของรับข้อมูลแบบซิงโกรนัส 4 บิต

2 สร้างวงจรรับข้อมูลแบบซิงโกรนัสด้วย FPGA

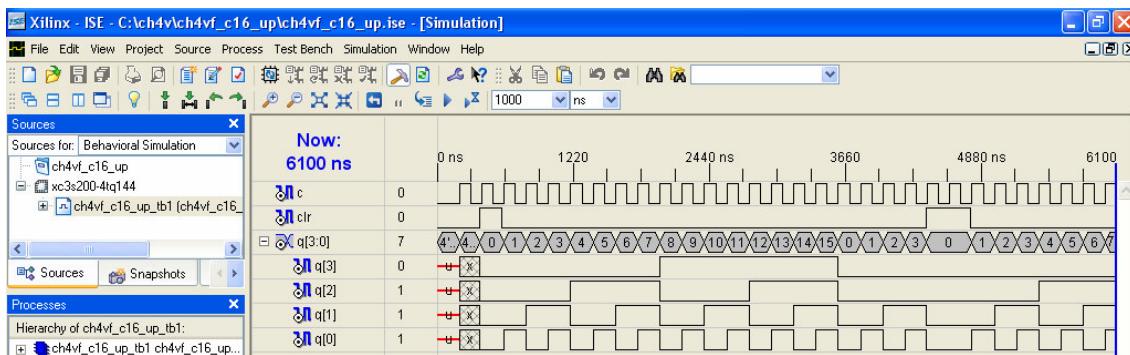
- 2.1) เขียนโค้ดของรับข้อมูลแบบซิงโกรนัส 4 บิตที่เขียนโดยการประกาศใช้ Signal โดยนำในตัวอย่างที่ 2.26 โค้ดในรูปที่ 2.46c) มาแก้ไขและสร้างไฟล์ Component โดยไว้ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ch4vf_c16_up เสร็จแล้วจะได้ดังรูปที่ L2.1 ทำการ Check syntax และบันทึกไฟล์ จากนั้นทำการ Behavioral simulation โดยใช้ Source File ชื่อ ch4vf_c16_up_tb1 และดูผลการทำงานดังรูปที่ L2.2 ว่าเป็นไปตามทฤษฎีหรือไม่
- 2.2) ทำการทดสอบของรับข้อมูลแบบซิงโกรนัส 4 บิตโดยสร้างไว้ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_3_5vf โดยโค้ดที่ได้แสดงดังรูปที่ L2.3 และนำไฟล์ของรีบีเบาเซอร์ชื่อ ch4vf_DEBOUNCER และไฟล์ของรับข้อมูลชื่อ ch4vf_c16_up มาทำเป็น Component และ Add Source ของ Component ทั้ง 2 ไฟล์เข้าไปในไฟล์ Project

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ch4vf_c16_up is
7     port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ch4vf_c16_up;
10
11 architecture Behavioral of ch4vf_c16_up 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;

```

รูปที่ L2.1 โค้ดของรับข้อมูลแบบซิงโกรนัส 4 บิต



รูปที่ L2.2 ผลจำลองการทำงานของโค้ดในรูปที่ L2.1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex4_3_5vf is
6     Port ( CLR,CLR_DB,Din,OSC : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end ex4_3_5vf;
9
10 architecture Behavioral of ex4_3_5vf is
11 component ch4vf_DEBOUNCER
12     Port ( OSC,D_in,CLR : in STD_LOGIC;
13             D_out : out STD_LOGIC);
14 end component;
15 component ch4vf_c16_up
16     Port ( C,CLR : in STD_LOGIC;
17             Q : out STD_LOGIC_VECTOR (3 downto 0));
18 end component;
19     signal C : STD_LOGIC;
20     signal QT : STD_LOGIC_VECTOR (3 downto 0);
21 begin
22     -----Debouncer-----
23 DEBOUNCER : ch4vf_DEBOUNCER
24     port map( OSC=>OSC,D_in=>Din,CLR=>CLR_DB,D_out=>C);
25     -----4Bits Counter-----
26 COUNTER4BIT : ch4vf_c16_up
27     port map( C=>C,CLR=>CLR,Q=>Q);
28 end Behavioral;

```

รูปที่ L2.3 โค้ด VHDL ของวงจรนับบีน (แบบซิงโกรนัส) 4 บิตที่รวมวงจรดีเบาเซอร์อย่างง่ายไว้แล้ว

การกำหนดขาสัญญาณต่างๆ จะใช้อสชิลเดเตอร์ OSC = 25 MHz และปุ่มกด PB1-PB2 และ Dip SW1 เป็นอินพุต และเมจ LED L0-L3 เป็นเอาต์พุต ก่อว่าคือ

Din = PB1 = INPUT = p44	Q(2) = L3 = OUTPUT = p76
CLR_DB = PB2 = INPUT = p46	Q(2) = L2 = OUTPUT = p69
OSC = OSC = INPUT = p127	Q(1) = L1 = OUTPUT = p77
CLR = Dip SW1 = INPUT = p52	Q(0) = L0 = OUTPUT = p70

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLR	Input	p52	BANK5	LVCMS33	N/A	3.30					Unknown		
CLR_DB	Input	p46	BANK5	LVCMS33	N/A	3.30					Unknown		
Din	Input	p44	BANK5	LVCMS33	N/A	3.30					Unknown		
OSC	Input	p127	BANK0	LVCMS33	N/A	3.30					Unknown		
Q<0>	Output	p70	BANK4	LVCMS33	N/A	3.30			SLOW		Unknown		
Q<1>	Output	p77	BANK3	LVCMS33	N/A	3.30			SLOW		Unknown		
Q<2>	Output	p69	BANK4	LVCMS33	N/A	3.30			SLOW		Unknown		
Q<3>	Output	p76	BANK3	LVCMS33	N/A	3.30			SLOW		Unknown		

รูปที่ L2.4 Assign Package Pins

หลังจากโปรแกรมจะที่ออกแบบลงชิป FPGA แล้วเลื่อน Dip SW1 ไปที่ตำแหน่ง ON (CLR= '0') และให้ทดลองกดปุ่ม PB1 และ PB2 ลับกันไปเรื่อยๆ แล้วให้สังเกตคุณลักษณะที่ LED L0-L3 ว่าติดสว่างโดยให้อิจิເຕີພຸດເປັນໄປຕາມທຸນຸ້ງຮູ່ອື່ນຈາກນັ້ນເລືອນ Dip SW1 ไปที่ตำแหน่ง OFF (CLR= '1') แล้ว ON แล้วทดลองซ้ำ จากนັ້ນຈຶ່ງບັນທຶກผลการทดลอง

2.3) ทำการแก้ไขที่ใช้ทำ Component ในรูปที่ L2.1 โดยเปิดไฟล์ที่อยู่ใน Project Location ชื่อ ch4v ที่มี Project Name และ Source File ชื่อ ch4vf_c16_up จากนັ້ນແກ້ໄຂໂຄດວາງຈານນີ້ແນບຊີງໂຄຣນັ້ນ 4 ບົດທີ່ເພີຍໂຄຍາປະກາດໃຫ້ Variable ດັ່ງນີ້

L2.5 ทำการ Check syntax และບັນທຶກໄຟລ໌ຊ້າອີກຮັ້ງ จากนັ້ນທຳ Behavioral simulation ຊ້າອີກຮັ້ງແລ້ວດູ່ຜລຈຳດອງການທຳງານວ່າເໜີ້ອນກັບດັ່ງນີ້

L2.1.2 ອູ່ອື່ນໄປຕາມທຸນຸ້ງຮູ່ອື່ນຈາກນັ້ນຈຶ່ງທຳການທຳດອງຊ້າ ເສັ່ງແລ້ວໃຫ້ແກ້ໄຂ Component ຂອງໂຄດວາງຈານນີ້ແນບຊີງໂຄຣນັ້ນ 4 ບົດອີກຮັ້ງໂດຍໃຫ້ໂຄດໃນຕົວຢ່າງທີ່ 2.26 ໃນຮູບທີ່ 2.46e) ແລະຮູບທີ່ 2.46f) ຕາມດຳນັ້ນ จากນັ້ນຈຶ່ງທຳການທຳດອງຊ້າຈຳນຽມທຸກວິທີ

```

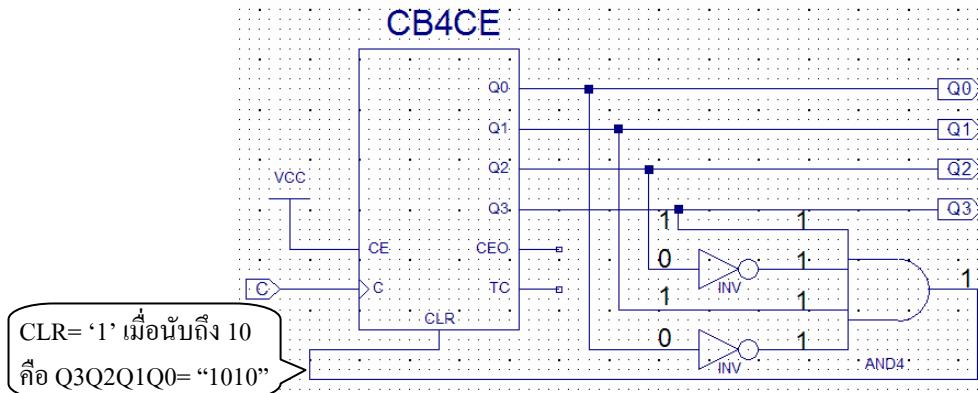
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ch4vf_c16_up is
7     port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ch4vf_c16_up;
10
11 architecture Behavioral of ch4vf_c16_up is
12 begin
13     process(C,CLR)
14         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;

```

ຮູບທີ່ L2.5 ໂຄດວາງຈານນີ້ແນບຊີງໂຄຣນັ້ນ 4 ບົດ

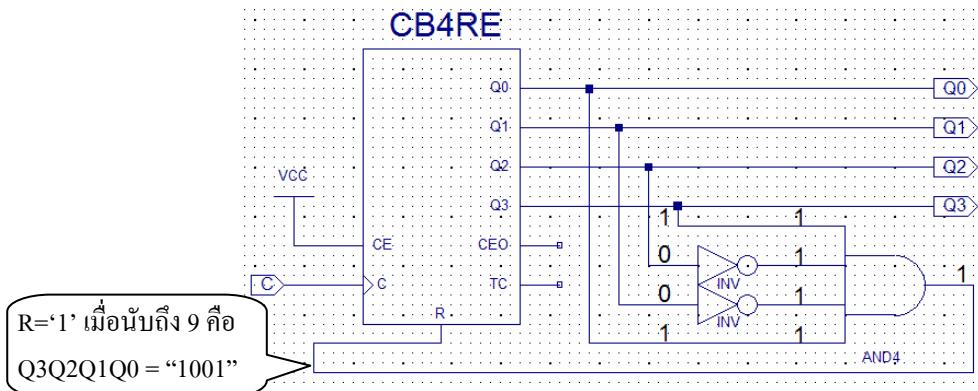
4.4 การออกแบบวงจรนับ N

วงจรนับ N แบบชิงโครนัสเคลียร์นั้นจะใช้ค่า N เป็นเงื่อนไขในการเคลียร์วงจรนับ เช่น วงจรนับ 10 แสดงดังในรูปที่ 4.33 ซึ่งจะนำเอาวงจรนับ 16 แบบ Asynchronous clear (CB4CE) ไปใช้ เมื่อเอาต์พุต = “1010” ($N = 10$) แล้ว CLR = ‘1’ จะทำให้มีการเคลียร์เอาต์พุต = “0000” ทันทีโดยไม่รอการทริกจากสัญญาณนาฬิกา ข้อเสียของวงจรนับนี้ คือ ใช้งานได้ที่ความถี่ต่ำ และเอาต์พุตของวงจรนับเกิดสถานะ “1010” ช้าๆ คู่ๆ การแก้ไขปัญหานี้ทำได้โดยใช้งานวงจรนับ N แบบชิงโครนัสเรซต์



รูปที่ 4.33 วงจรนับ 10 ที่ดัดแปลงมาจากวงจรนับ 16 แบบ Asynchronous clear (CB4CE)

วงจรนับ N แบบชิงโครนัสเรซต์นั้นจะใช้ค่า $N-1$ เป็นเงื่อนไขในการรีเซ็ตวงจรนับ เช่น วงจรนับ 10 แสดงดังในรูปที่ 4.34 ซึ่งจะนำเอาวงจรนับ 16 แบบ Synchronous reset (CB4RE) มาใช้ เมื่อเอาต์พุต = “1001” ($N-1 = 9$) แล้ว R = ‘1’ วงจรนับนี้จะไม่รีเซ็ตทันที แต่จะต้องรอสัญญาณนาฬิกาลูกคัดไปทริกจึงจะรีเซ็ตค่าเอาต์พุตของวงจรนับ = “0000” อีกครั้ง ข้อดีของการออกแบบวงจรนับ N หรือหาร N แบบนี้ คือ วงจรใหม่ที่ได้จะบังคับเป็นวงจรนับแบบชิงโครนัส ซึ่งการออกแบบวงจรด้วย CPLD หรือ FPGA นั้นควรออกแบบวงจรนับแบบชิงโครนัสเป็นหลัก เนื่องจากวงจรนับนี้จะทำงานที่ความถี่สูงได้ดี



รูปที่ 4.34 วงจรนับ 10 ที่ดัดแปลงมาจากวงจรนับ 16 แบบ Synchronous Reset (CB4RE)

ตัวอย่างที่โภคดวงวงจรนับ 60 แบบชิงโครนัสที่แสดงผลเป็นเลขฐานสองแสดงดังรูปที่ 4.35 และ 4.36 โดยกรณีที่เป็นการนับขึ้นนับลงจะอาศัยหลักการ Synchronous reset โดยเอา 59 เป็นเงื่อนไขในการรีเซ็ตวงจรนับ 64 (6 บิต) เพื่อให้กลับเป็นวงจรนับ 60 ส่วนกรณีที่เป็นการนับลงจะอาศัยหลักการ Synchronous reset โดยเอา 0 หรือ 59 เป็นเงื่อนไขในการรีเซ็ต

ขอให้ผู้อ่านสังเกตว่าโภคดวงวงจรที่ 18 ในรูปที่ 4.35 นี้เราจะเขียน $QT \geq 59$ แทน $QT = 59$ เพื่อป้องกันการนับเริ่มต้นผิดพลาด ซึ่งอาจมีการนับนอกช่วงคือ 60-63 เช่น ถ้าในรอบแรกเริ่มต้นที่ 61 ก็จะนับต่อเป็น 62 และ 63 แล้วจึงจะวน

กลับมานับถูกต้องอยู่ในช่วง 0-59 ในรอบคั่วไป ในทำนองเดียวกัน โค้ดดังจะรันบล็อกในบรรทัดที่ 18 ในรูปที่ 4.36 นี้เราจะเขียน ($QT = 0$ or $QT > 59$) แทน $QT = 0$ เพื่อป้องกันการนับเริ่มต้นพิดพลาด ซึ่งอาจนับนอกช่วงคือ 60-63 ได้เช่นกัน

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity C60B_UP is
7     Port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (5 downto 0));
9 end C60B_UP;
10
11 architecture Behavioral of C60B_UP is
12     signal QT : STD_LOGIC_VECTOR (5 downto 0);
13 begin
14 process(C,CLR)
15 begin
16     if CLR='1' then QT <= "000000";
17     elsif (C'event and C='1') then
18         if QT>=59 then QT <= "000000";--Synchronous reset:N>=59
19         else QT <= QT + 1;
20     end if;
21     end if;
22 end process;
23     Q <= QT;
24 end Behavioral;
```

รูปที่ 4.35 โค้ดดังจะรันบล็อก 60 (เลขฐานสอง) แบบนับขึ้นที่เรียกใช้ Std_logic_unsigned package

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity C60B_DOWN is
7     Port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (5 downto 0));
9 end C60B_DOWN;
10
11 architecture Behavioral of C60B_DOWN is
12     signal QT : STD_LOGIC_VECTOR (5 downto 0);
13 begin
14 process(C,CLR)
15 begin
16     if CLR='1' then QT <= "000000";
17     elsif (C'event and C='1') then
18         if (QT=0 or QT>59) then QT <= "111011";--Synchronous reset:N=0 or N>59
19         else QT <= QT - 1;
20     end if;
21     end if;
22 end process;
23     Q <= QT;
24 end Behavioral;
```

รูปที่ 4.36 โค้ดดังจะรันบล็อก 60 (เลขฐานสอง) แบบนับลงที่เรียกใช้ Std_logic_unsigned package

การออกแบบวงจรนับ (ขึ้น) 10 แบบชิงโครนัสก์ได้เช่นกัน เราจะอธิบายหลักการ Synchronous reset โดยใช้ 9 ไบรีเซต วงจรนับดังมีรายละเอียดในรูปที่ 2.49a) ถึงรูปที่ 2.49d) ตัวอย่างที่ 2.27 ในบทที่ 2 ในการออกแบบหารความถี่ก็จะใช้หลักการเดียวกัน ซึ่งวงจรนับ 10 และหารความถี่ 10 แสดงดังรูปที่ 4.37 โดยที่วงจรหารความถี่นี้จะมีเอาต์พุตแบบธรรมด้าและเอาต์พุตที่มี Duty 50% (ใน 1 คาบจะมีเวลาที่เป็นครึ่งบวกและศูนย์เท่ากัน) ผลจำลองการทำงานที่ได้แสดงดังรูปที่ 4.38

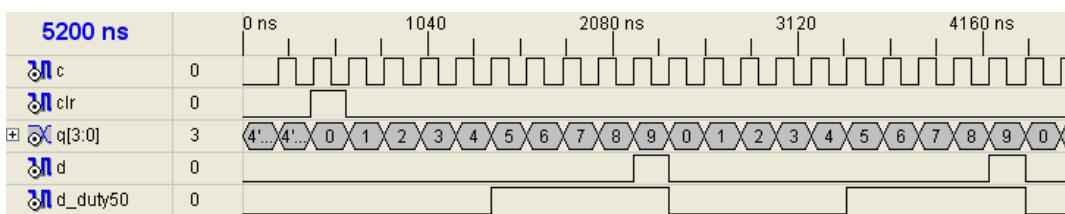
ขอให้ผู้อ่านสังเกตว่า ในรูปที่ 4.37 สำหรับการเอาต์พุต Q ที่ให้ไว้ Comment ที่บรรทัดที่ 8 และ 25 และหากต้องการ เคลื่อนไหวต่อไปที่มี Duty cycle 50% เทียงอย่างเดียวกันให้ใส่ Comment เพิ่มในบรรทัดที่ 9, 26 และ 27

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity DEVIDER10 is
7     Port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0);
9             D : out STD_LOGIC;
10            D_DUTY50 : out STD_LOGIC);
11 end DEVIDER10;
12
13 architecture Behavioral of DEVIDER10 is
14     signal QT : STD_LOGIC_VECTOR (3 downto 0);
15 begin
16     process(C,CLR)
17     begin
18         if CLR='1' then QT <= "0000";
19         elsif (C'event and C='1') then
20             if QT>=9 then QT <= "0000";
21             else QT <= QT + 1;
22             end if;
23         end if;
24     end process;
25     Q <= QT;                                --Counter output
26     D <= '1' when (QT>=9 and QT<10) else --Duty cycle=10%
27         '0';
28     D_DUTY50 <= '1' when (QT>=5 and QT<10) else --Duty cycle=50%
29         '0';
30 end Behavioral;

```

รูปที่ 4.37 โค้ดวงจรนับ 10 และหารความถี่ 10



รูปที่ 4.38 ผลจำลองการทำงานวงจรนับ 10 และหารความถี่ 10 ที่มีเอต้าพุดาและเอต้าพุตที่มี Duty 50%

ตัวอย่าง โค้ดของวงจรนับ 10 (BCD) แบบนับขึ้น-นับลงแบบคาสเคด (Cascade) ที่ใช้สำหรับทำบีนวงจรนับหลายหลักแสดงดังรูปที่ 4.39 ซึ่งวงจรนับพร้อมจะนับเมื่อ Clock Enable input (CE) = ‘1’ และหยุดนับเมื่อ CE = ‘0’ ถ้า Direction (DIR) = ‘0’ จะเป็นการนับขึ้นและเมื่อนับถึง 9 แล้ว Clock Enable output (CEO) = ‘1’ แต่ถ้า DIR = ‘1’ จะเป็นการนับลงและเมื่อนับลงถึง 0 แล้ว CEO = ‘1’ ซึ่งผลจำลองการทำงานของวงจรนับแสดงดังรูปที่ 4.40

```

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

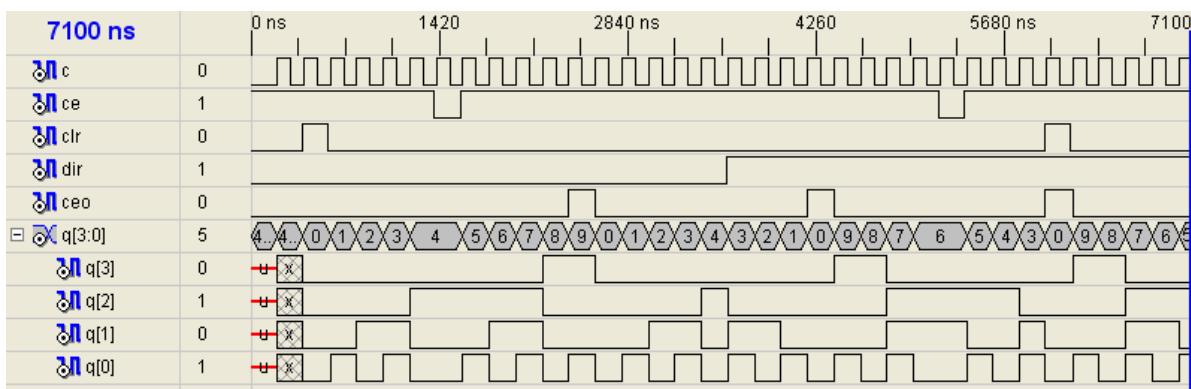
```

```

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 process;
31  Q <= QT;                                -- Counter output
32  CEO <= CE when (DIR ='0' and QT=9) else --CEO-count up
33      CE when (DIR ='1' and QT=0) else --CEO-count down
34      '0';
35  --
36 end Behavioral;

```

รูปที่ 4.39 โค้ดของวงจรนับ 10 แบบนับขึ้น-นับลงที่มีขา CE และ CEO



รูปที่ 4.40 ผลจำลองการทำงานของวงจรนับ 10 แบบนับขึ้น-นับลงที่มีขา CE และ CEO

ในทำนองเดียวกันเราสามารถเขียนโค้ดของวงจรนับ 100 (0-99) แบบนับขึ้น-นับลงที่แสดงผลเป็นรหัส BCD หรือเลขฐานสิบ 2 หลักได้ เช่น กันแสดงดังรูปที่ 4.41 โดยให้หลักหน่วยเป็น Q0 (4 บิต) และหลักสิบเป็น Q1 (4 บิต)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity C100D_UPDN is
7     Port ( C,CLR,DIR : in STD_LOGIC;
8             Q0,Q1 : out STD_LOGIC_VECTOR (3 downto 0));
9 end C100D_UPDN;
10
11 architecture Behavioral of C100D_UPDN is
12     signal Q0T,Q1T : STD_LOGIC_VECTOR (3 downto 0);
13 begin
14 process(C,CLR)
15 begin
16     if CLR='1' then Q0T <= "0000"; Q1T <= "0000";
17     elsif (C'event and C='1') then
18         if DIR='0' then -----Count up
19             if (Q0T>=9) then
20                 Q0T <= "0000";
21                 if (Q1T>=9) then Q1T <= "0000";
22                     else Q1T <= Q1T + 1;
23                     end if;
24             else Q0T <= Q0T + 1;
25             end if;

```

(ต่อ)

```

26      else      -----Count down
27          if (QOT=0 or QOT>9) then
28              QOT <= "1001";
29              if (Q1T=0 or Q1T>9) then Q1T <= "1001";
30              else Q1T <= Q1T - 1;
31              end if;
32          else QOT <= QOT - 1;
33          end if;
34      end if;
35  end if;
36 end process;
37 Q0 <= QOT;  Q1 <= Q1T;
38 end Behavioral;

```

รูปที่ 4.41 โค้ดของวงจรนับ 100 (0-99) แบบนับขึ้น-นับลงที่แสดงผลเป็นรหัส BCD หรือเลขฐานสิบ 2 หลัก

ตัวอย่างโค้ดของวงจรนับ 24 (0-23) ที่แสดงผลเป็นรหัส BCD 2 หลักแสดงดังรูปที่ 4.42 นั้นเป็นการเอาวงจรนับ 100 ไปดัดแปลงโดยใช้หลักการ Synchronous reset โดยใช้ 23 ไปรีเซตวงจรนับ วงจนีจะนับเมื่อ CE = '1' และจากหลักการนี้เราสามารถสร้างวงจรนับค่าอื่นๆ ที่แสดงผลเป็นรหัส BCD หรือเลขฐานสิบ 2 หลักได้ง่าย ซึ่งเราสามารถทำได้โดยแก้ไขเฉพาะที่บรรทัดที่ 19 เท่านั้น เช่น ถ้าต้องการออกแบบวงจรนับ 0-60 ก็ให้แก้บรรทัดที่ 19 ของโค้ดเป็น Q1T>=5 และ Q0T>=9

```

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 QOT,Q1T : STD_LOGIC_VECTOR (3 downto 0);
13 begin
14 process(C,CLR)
15 begin
16     if CLR='1' then Q1T <= "0000"; QOT <= "0000";--Clear
17     elsif (C'event and C='1') then
18         if CE='1' then           --Clock enable
19             if (Q1T>=2 and QOT>=3) then      --Synchronous reset:N>=23
20                 Q1T <= "0000";
21                 QOT <= "0000";
22             else
23                 if (QOT>=9) then           --Counter:N=0-99
24                     QOT <= "0000";
25                     if (Q1T>=9) then Q1T <= "0000";
26                     else Q1T <= Q1T + 1;
27                     end if;
28                 else QOT <= QOT + 1;
29                 end if;
30             end if;
31         end if;
32     end if;
33 end process;
34     Q0 <= QOT;  Q1 <= Q1T;
35 end Behavioral;

```

รูปที่ 4.42 โค้ดของวงจรนับ 24 (0-23) แบบนับขึ้นที่แสดงผลเป็นรหัส BCD หรือเลขฐานสิบ 2 หลัก

การทดลองที่ 4.4.1 การสร้างวงจรนับ 10 แบบซิงโครนัสที่เป็นวงจรนับขึ้น

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานและออกแบบวงจรนับ 10 แบบซิงโครนัสที่เป็นวงจรนับขึ้น
- 2) เพื่อสร้างวงจรนับ 10 ที่เป็นวงจรนับขึ้นโดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรนับ 10 แบบซิงโครนัสที่เป็นวงจรนับขึ้นด้วย CPLD

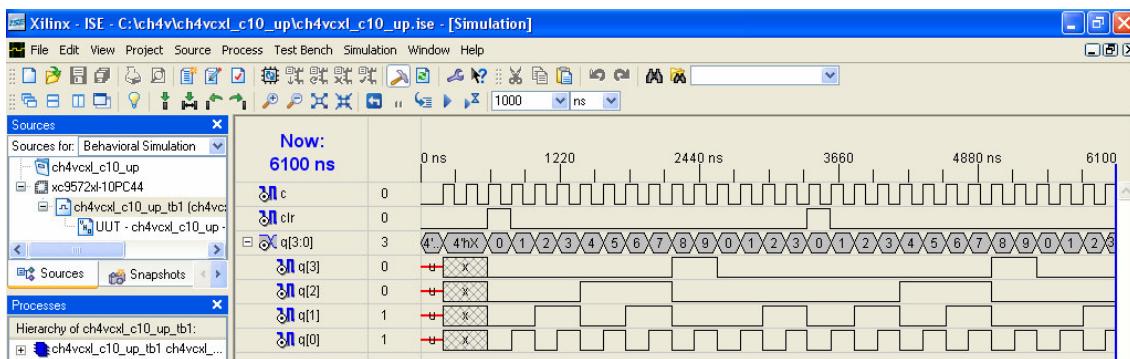
- 1.1) เขียนวงจรนับ 10 แบบซิงโครนัสโดยนำโค้ดในตัวอย่างที่ 2.27 รูปที่ 2.49a ในบทที่ 2 มาแก้ไขและสร้างไฟล์ Component โดยไว้ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ch4vcxl_c10_up เสร็จแล้วจะได้ดังรูปที่ L1.1 ทำการ Check syntax และบันทึกไฟล์ จากนั้นทำ Behavioral simulation โดยใช้ Source File ชื่อ ch4vcxl_c10_up_tb1 และดูผลจำลองการทำงานดังรูปที่ L1.2 ว่าเป็นไปตามทฤษฎีหรือไม่

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ch4vcxl_c10_up is
7     Port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ch4vcxl_c10_up;
10
11 architecture Behavioral of ch4vcxl_c10_up 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";
19             else QT <= QT + 1;
20             end if;
21         end if;
22     end process;
23     Q <= QT;
24 end Behavioral;

```

รูปที่ L1.1 โค้ดวงจรนับ 10 แบบซิงโครนัสที่เป็นวงจรนับขึ้นที่ประมวลใช้ Signal



รูปที่ L1.2 ผล Behavioral simulation ของวงจรนับที่ใช้โค้ดในรูปที่ L1.1 (U = Uninitial และ 'X' = Unknown)

1.2) เขียนโค้ดของวงจรนับ 10 แบบซิงโครนัสที่เป็นวงจรนับขึ้นที่รวมวงจรดีเบาเซอร์ไว้แล้วไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_4_1vcxl แล้วเขียนโค้ดของวงจรนับขึ้น 15 บิตโดยเลือกใช้ความถี่ 2 Hz เพื่อเคลียร์ วงจรดีเบาเซอร์โดยอัตโนมัติทุกๆ $1/2\text{Hz} = 0.5$ วินาที จากนั้นนำไฟล์วงจรดีเบาเซอร์ชื่อ ch4vcxl_debounce และไฟล์วงจรนับ ชื่อ ch4vcxl_c10_up มาทำเป็น Component เตรียมแล้วจะได้ดังรูปที่ L1.3

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_4_1vcxl is
7     Port ( CLR,Din,OSC : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex4_4_1vcxl;
10
11 architecture Behavioral of ex4_4_1vcxl is
12     component ch4vcxl_DEBOUNCER
13         Port ( OSC,D_in,CLR : in STD_LOGIC;
14                 D_out : out STD_LOGIC);
15     end component;
16     component ch4vcxl_c10_up
17         Port ( C,CLR : in STD_LOGIC;
18                 Q : out STD_LOGIC_VECTOR (3 downto 0));
19     end component;
20     signal C : STD_LOGIC;
21     signal F : STD_LOGIC_VECTOR (14 downto 0);
22 begin
23     -----15Bits Counter-----
24 process(OSC) --Output F(14)=1Hz,F(13)=2Hz,F(12)=4Hz,F(11)=8Hz
25 begin
26     if OSC'event and OSC='1' then F <= F + 1;
27     end if;
28 end process;
29     -----DEBOUNCER-----
30 DEBOUNCER : ch4vcxl_DEBOUNCER
31     port map(OSC=>OSC,D_in=>Din,CLR=>F(13),D_out=>C);
32     -----COUNTER : C10_UP-----
33 COUNTER_10UP : ch4vcxl_c10_up
34     port map(C=>C,CLR=>CLR,Q=>Q);
35 end Behavioral;

```

รูปที่ L1.3 โค้ดของวงจรนับ 10 แบบซิงโครนัสที่เป็นวงจรนับขึ้นที่รวมวงจรดีเบาเซอร์ไว้แล้ว

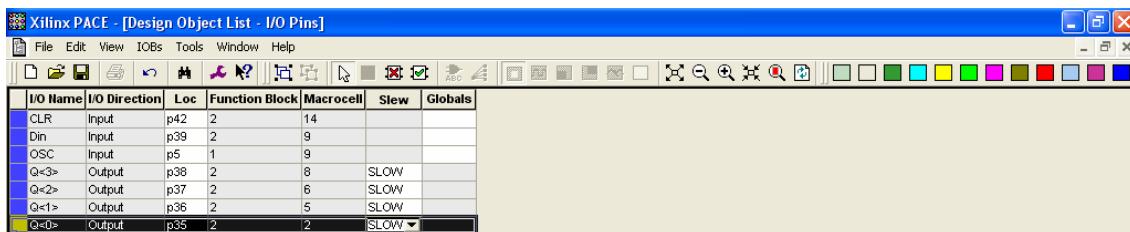
การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz ปุ่มกด PB1 และ PB3(Slide SW1) เป็นอินพุต และ LED1-LED4 เป็นเอาต์พุต ก่อร่องคือ

Din = PB1 = INPUT = p39 Q(3) = LED1 = OUTPUT = p38 Q(1) = LED3 = OUTPUT = p36

CLR= PB3(Slide SW1) = INPUT= p42 Q(2) = LED2 = OUTPUT = p37 Q(0) = LED4 = OUTPUT = p35

OSC= OSC = OUTPUT = p5

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



รูปที่ L1.4 Assign Package Pins

หลังจากโปรแกรมจะรลงซิฟ CPLD แล้วคุณต้องต่อ LED1-LED4 ในตอนเริ่มต้นว่า LED ทุกดวงดับหรือให้ล็อกอิกເອຕີພຸດ เป็น '0' จริงหรือไม่ (ถ้าไม่มีการใส่ค่าเริ่มต้น (Initial value) ให้กับบริจิตเตอร์นี้ XST จะกำหนดค่า (Default) เป็นລອຈິກ '0' ຄື່ອ QT เป็น "0000") จากนั้นเลื่อน Slide SW1 ไปที่ตำแหน่ง ON (CLR = '0') ให้กดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกด PB1 ค้างไว้ แล้วให้คุณต้อง LED1-LED4 ว่าติดสว่างโดยให้ล็อกເອຕີພຸດเป็นໄປຕາມທຸກໆໃຫຍ້ ທີ່ກ່ຽວຂ້ອງກົດ PB1 ค้างໄວ້ຈະໄຫ້ຄວາມຄື່ 2 Hz (ເທົ່າກັບຄວາມຄື່ທີ່ຂາ CLR ທີ່ປົ້ນຈາກ F(13)) จากนั้นเลื่อน Slide SW1 ไปที่ตำแหน่ง OFF (CLR = '1') เพื่อເຄີຍເອຕີພຸດແລ້ວ ON ແລ້ວທົດລອງຫໍ້າ

2 สร้างวงจรนับ 10 แบบชิงໂຄຣນັສທີ່ເປັນວັງຈານບື້ນດ້ວຍ FPGA

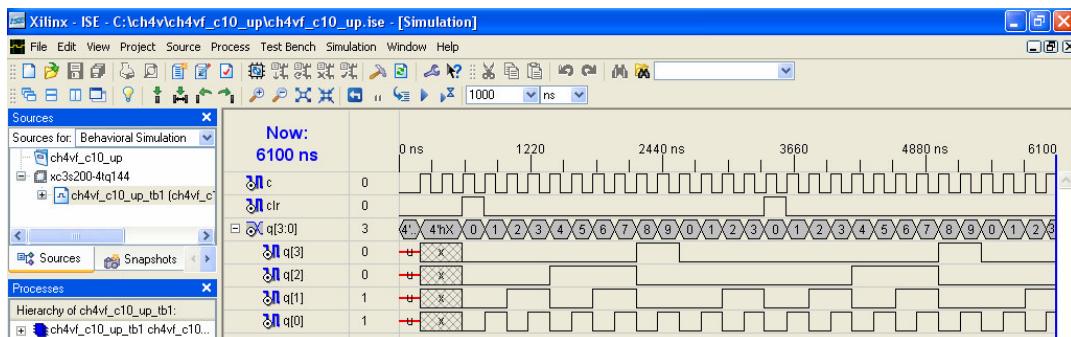
2.1) สร้างไฟล์ของໂຄດວັງຈານ 10 แบบชิงໂຄຣນັສທີ່ເປັນວັງຈານບື້ນດ້ງຮູບທີ່ L2.1 ໃວ່າໃນ Project Location ຊື່ອ ch4v ກໍານັດ Project Name ແລະ Source File ຊື່ອ ch4vf_c10_up ບັນທຶກໄຟລ໌ແລ້ວ Check syntax ທຳ Behavioral simulation ໂດຍໃຊ້ໄຟລ໌ທີ່ອ ch4vf_c10_up_tb1 ແລ້ວພິຈາລະນາຜົດ Behavioral simulation ໃນຮູບທີ່ L2.2 ວ່າເປັນໄປຕາມທຸກໆໃຫຍ້ ໃຫຍ້

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ch4vf_c10_up is
7     Port ( C,CLR : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ch4vf_c10_up;
10
11 architecture Behavioral of ch4vf_c10_up 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";
19         else QT <= QT + 1;
20         end if;
21     end if;
22 end process;
23 Q <= QT;
24 end Behavioral;

```

ຮູບທີ່ L2.1 ໂຄດວັງຈານ 10 แบบชิงໂຄຣນັສທີ່ເປັນວັງຈານບື້ນທີ່ປະກາດໃຊ້ Signal



ຮູບທີ່ L2.2 ຜົດ Behavioral simulation ຂອງວັງຈານທີ່ໃຊ້ໄຟດີໃນຮູບທີ່ L2.1 (U = Uninitial และ 'X' = Unknown)

2.2) ເປັນໄຟດີຂອງວັງຈານ 10 แบบชິງໂຄຣນັສທີ່ເປັນວັງຈານບື້ນທີ່ຮັມວັງຈານດີເບາເຊອງໄວ້ແລ້ວໄວ້ໃນ Project Location ຊື່ອ ch4v ກໍານັດ Project Name ແລະ Source File ຊື່ອ ex4_4_1vf ເປັນໄຟດີຂອງວັງຈານບື້ນ 24 ບົດ ໂດຍເລືອກໃຊ້ຄວາມຄື່ 2.98 Hz ເພື່ອເຄີຍເວັງຈານດີເບາເຊອງໄດ້ອັດໂນມັດຖຸກໆ 0.34 ວິນາທີ່ ນຳໄຟລົງຈານດີເບາເຊອງທີ່ ch4vf_debounce ແລະໄຟລົງຈານທີ່ ch4vf_c10_up ມາທຳເປັນ Component ເສົ່ງແລ້ວຈະໄດ້ດັງຮູບທີ່ L2.3

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_4_ivf is
7     Port ( CLR,Din,OSC : in STD_LOGIC;
8            Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex4_4_ivf;
10
11 architecture Behavioral of ex4_4_ivf is
12     component ch4vf_DEBOUNCER
13         Port ( OSC,D_in,CLR : in STD_LOGIC;
14                 D_out : out STD_LOGIC);
15     end component;
16     component ch4vf_c10_up
17         Port ( C,CLR : in STD_LOGIC;
18                 Q : out STD_LOGIC_VECTOR (3 downto 0));
19     end component;
20     signal C : STD_LOGIC;
21     signal F : STD_LOGIC_VECTOR (23 downto 0);
22 begin
23     -----24Bits Counter-----
24     process(OSC) --Output F(23)=1.49Hz,F(22)=2.98Hz,F(21)=5.96Hz
25     begin
26         if OSC'event and OSC='1' then F <= F + 1;
27     end if;
28 end process;
29     -----DEBOUNCER-----
30     DEBOUNCKER : ch4vf_DEBOUNCER
31     port map(OSC=>OSC,D_in=>Din,CLR=>F(22),D_out=>C);
32     -----COUNTER : C10_UP-----
33     COUNTER_10UP : ch4vf_c10_up
34     port map(C=>C,CLR=>CLR,Q=>Q);
35 end Behavioral;

```

รูปที่ L2.3 โค้ดของวงจรนับ 10 แบบชิงโครนัสที่เป็นวงจรนับขึ้นที่รวมวงจรดีเบาเซอร์ไว้แล้ว

การกำหนดขาสัญญาณต่างๆ จะใช้ OSC = 25 MHz, PB1 และ Dip SW1 เป็นอินพุต มี LED L0-L3 เป็นเอาต์พุต

Din = PB1 = INPUT = p44 Q(3)=L3 = OUTPUT = p76 Q(1)=L1 = OUTPUT = p77

CLR= Dip SW1 = INPUT = p52 Q(2)=L2 = OUTPUT = p69 Q(0)=L0 = OUTPUT = p70

OSC = OSC= INPUT= p127

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLR	Input	p52	BANK LVC MOS33	N/A	3.30						Unknown		
Din	Input	p44	BANK LVC MOS33	N/A	3.30						Unknown		
OSC	Input	p127	BANK LVC MOS33	N/A	3.30						Unknown		
Q<0>	Output	p70	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
Q<1>	Output	p77	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
Q<2>	Output	p69	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
Q<3>	Output	p76	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		

รูปที่ L2.4 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป FPGA แล้วคุณต้องตั้งค่า LED L0-L3 ในตอนเริ่มต้นว่า LED ทุกดวงตั้งหรือให้ถูกจิกเอาต์พุตเป็น ‘0’ จริงหรือไม่ (ถ้าไม่มีการใส่ค่าเริ่มต้น (Initial value) ให้กับบริจิตเตอร์นั้น XST จะกำหนดค่า (Default) เป็นโลจิก ‘0’ คือ QT เป็น “0000”) จากนั้นเลื่อน Dip SW1 ไปที่ตำแหน่ง ON (CLR = ‘0’) ให้กดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกด PB1 ค้างไว้แล้วคุณต้อง LED0-LED3 ว่าคิดสว่างโดยให้ถูกจิกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ ซึ่งการกด PB1 ค้างไว้จะให้ความถี่ 2.98 Hz (เท่ากับความถี่ที่ CLR ซึ่งป้อนจาก F(22)) จากนั้นเลื่อน Dip SW1 ไปที่ตำแหน่ง OFF (CLR= ‘1’) เพื่อเคลียร์เอาต์พุตแล้ว ON แล้วทดสอบซ้ำ

การทดลองที่ 4.4.2 การสร้างวงจรนับ 10 ที่เป็นวงจรนับขึ้น-นับลงแบบค่าสเกด 1 หลัก

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจหลักการทำงานและออกแบบวงจรนับ 10 ที่เป็นวงจรนับขึ้น-นับลงแบบค่าสเกด (Cascade)
- 2) เพื่อประยุกต์ใช้งานรีบูเตอร์ (Debouncer) และวงจรดอครหัสตัวแสดงผลเซกเมนต์ (7-Segment decoder)
- 3) เพื่อสร้างวงจรนับขึ้น-นับลงโดยใช้ภาษา VHDL และโปรแกรมลังชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

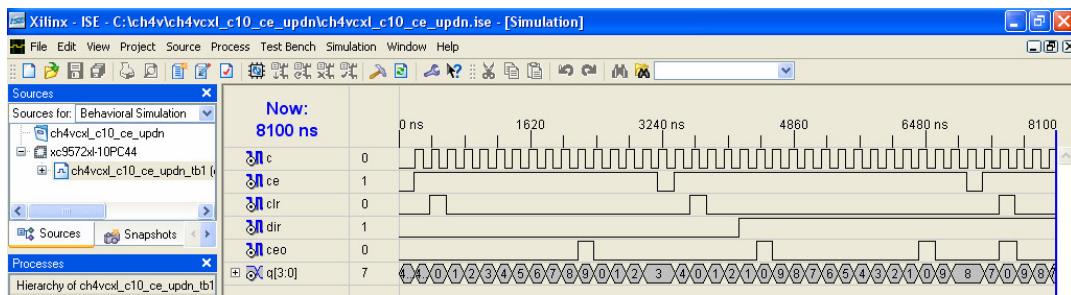
1 สร้างวงจรนับ 10 ที่เป็นวงจรนับขึ้น-นับลงแบบค่าสเกด 1 หลักด้วย CPLD

1.1) นำโค้ดควบวงจรนับ 10 ที่เป็นวงจรนับขึ้น-นับลงแบบค่าสเกดในตัวอย่างที่ 4.38a) มาแก้ไขและสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vcx1_c10_ce_updn เสร็จแล้วจะได้ดังรูปที่ L1.1 ทำการบันทึกไฟล์ และ Check syntax จากนั้นทำ Behavioral simulation โดยใช้ Source File ชื่อ ch4vcx1_c10_ce_updn_tb1 แล้วให้คุณลองการทำงานดังรูปที่ L1.2 ว่าเป็นไปตามทฤษฎีหรือไม่

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ch4vcx1_c10_ce_updn is
7     Port ( C,CE,CLR,DIR : in STD_LOGIC;
8             CEO : out STD_LOGIC;
9             Q : out STD_LOGIC_VECTOR (3 downto 0));
10 end ch4vcx1_c10_ce_updn;
11
12 architecture Behavioral of ch4vcx1_c10_ce_updn is
13     signal QT : STD_LOGIC_VECTOR (3 downto 0);
14 begin
15     -----Counter with CE and CEO-----
16 process(C,CLR)
17     begin
18         if CLR='1' then QT <= "0000";
19         elsif (C'event and C='1') then
20             if CE='1' then
21                 if DIR='0' then          -- Count up
22                     if QT>=9 then QT <= "0000";
23                     else QT <= QT + 1;
24                     end if;
25                 else                      -- Count down
26                     if (QT=0 or QT>9) then QT <= "1001";
27                     else QT <= QT - 1;
28                     end if;
29                 end if;
30             end if;
31         end if;
32     end process;
33     Q <= QT;                                -- Counter output
34     -----CEO-----
35     CEO <= CE when (DIR ='0' and QT=9) else --CEO-count up
36         CE when (DIR ='1' and QT=0) else --CEO-count down
37         '0';
38 end Behavioral;
```

รูปที่ L1.1 โค้ดของวงจรนับ 10 ที่เป็นวงจรนับขึ้น-นับลงแบบค่าสเกดที่ใช้ทำเป็น Component



รูปที่ L1.2 ผล Behavioral simulation ของวงจรนับที่ใช้โค้ดในรูปที่ L1.1

1.2) นำไฟล์วงจรอдрหัสตัวแสดงผลเลขเวนเซกเมนต์ (0-9) จากรูปที่ 2.29b) ในบทที่ 2 มาแก้ไขแล้วสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vcxl_HEX_to_7SEGMENT เสร็จแล้วจะได้ดังรูปที่ L1.3

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ch4vcxl_BCD_to_7SEGMENT is
6     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
7             Y : out STD_LOGIC_VECTOR (6 downto 0));
8 end ch4vcxl_BCD_to_7SEGMENT;
9
10 architecture Behavioral of ch4vcxl_BCD_to_7SEGMENT is
11 begin
12     -----BCD to 7-segments decoder-----
13     gfedcba
14     Y <= "1101111" when A="1001" else --9
15     "1111111" when A="1000" else --8      Y(0)=a
16     "0000111" when A="0111" else --7      ---
17     "1111101" when A="0110" else --6 Y(5)=f | Y(1)=b
18     "1101101" when A="0101" else --5      --- Y(6)=g
19     "1100110" when A="0100" else --4 Y(4)=e | Y(2)=c
20     "1001111" when A="0011" else --3      ---
21     "1011011" when A="0010" else --2      Y(3)=d
22     "0000110" when A="0001" else --1      ---
23     "0111111";                         --0,a,b,C,d,E,F
24 end Behavioral;

```

รูปที่ L1.3 โค้ดวงจรอдрหัสตัวแสดงผลเลขเวนเซกเมนต์แบบเลขฐานสิบแบบแคปติวัตที่ใช้ทำเป็น Component

1.3) เขียนโค้ดวงจรนับ 1 หลัก โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_4_1vcxl เขียนโค้ดวงจรนับขึ้น 15 บิตโดยเลือกใช้ความถี่ 2 Hz เพื่อเคลียร์วงจรดีเบาเซอร์โดยอัตโนมัติทุกๆ $1/2\text{Hz} = 0.5$ วินาที จากนั้นนำไฟล์วงจรดีเบาเซอร์ชื่อ ch4vcxl_debounce ไฟล์วงจรนับขึ้น-นับลงชื่อ ch4vcxl_c10_ce_updn และไฟล์วงจรอдрหัสตัวแสดงผลเลขเวนเซกเมนต์ชื่อ ch4vcxl_BCD_to_7SEGMENT มาทำเป็น Component เสร็จแล้วจะได้ดังรูปที่ L1.4

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_4_2vcxl is
7     Port ( CE,CLR,Din,DIR,OSC : in STD_LOGIC;
8             CEO,COM : out STD_LOGIC;
9             Y : out STD_LOGIC_VECTOR (6 downto 0));
10 end ex4_4_2vcxl;
11
12 architecture Behavioral of ex4_4_2vcxl is
13     component ch4vcxl_DEBOUNCER
14         Port ( OSC,D_in,CLR : in STD_LOGIC;
15                 D_out : out STD_LOGIC);
16     end component;

```

(ต่อ)

```

17 component ch4vcx1_c10_ce_updn
18     Port ( C,CE,CLR,DIR : in STD_LOGIC;
19             CEO : out STD_LOGIC;
20             Q : out STD_LOGIC_VECTOR (3 downto 0));
21 end component;
22 component ch4vcx1_BCD_to_7SEGMENT
23     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
24             Y : out STD_LOGIC_VECTOR (6 downto 0));
25 end component;
26 signal C : STD_LOGIC;
27 signal F : STD_LOGIC_VECTOR (14 downto 0);
28 signal Q : STD_LOGIC_VECTOR (3 downto 0);
29 begin
30 -----15Bits Counter-----
31 process(OSC) --Output F(14)=1Hz,F(13)=2Hz,F(12)=4Hz,F(11)=8Hz
32 begin
33     if OSC'event and OSC='1' then F <= F + 1;
34     end if;
35 end process;
36 -----DEBOUNCER-----
37 DEBOUNCER : ch4vcx1_DEBOUNCER
38     port map(OSC=>OSC,D_in=>Din,CLR=>F(13),D_out=>C);
39 -----COUNTER : C10_UPDN-----
40 COUNTER_10UPDN : ch4vcx1_c10_ce_updn
41     port map(C=>C,CE=>CE,CLR=>CLR,DIR=>DIR,CEO=>CEO,Q=>Q);
42 -----7 Segment decoder-----
43 COM <= '0'; -- Common cathode = GND
44 DECODER_7SEG : ch4vcx1_BCD_to_7SEGMENT
45     port map(A=>Q,Y=>Y);
46 end Behavioral;

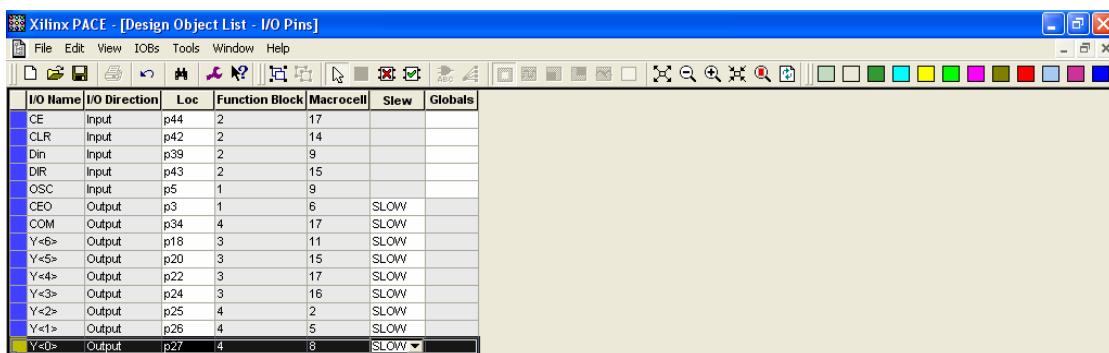
```

รูปที่ L1.4 โค้ดของวงจรนับ 10 ที่เป็นวงจรนับบีน-นับลงแบบค่าสเกลที่รวมวงจรดีเบาเซอร์ไว้แล้ว

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1, PB3(Slide SW1) ถึง PB5(Slide SW3) เป็นอินพุต มีตัวแสดงผลเซเว่นเซกเมนต์และ LED MN4 เป็นเอาต์พุต กล่าวคือ

Din = PB1 = INPUT = p39	Y(0) = a = OUTPUT = p27	Y(5) = f = OUTPUT = p20
CLR = PB3(Slide SW1) = INPUT = p42	Y(1) = b = OUTPUT = p26	Y(6) = g = OUTPUT = p18
DIR = PB4(Slide SW2) = INPUT = p43	Y(2) = c = OUTPUT = p25	CEO = MN4 = OUTPUT = p3
CE = PB5(Slide SW3) = INPUT = p44	Y(3) = d = OUTPUT = p24	COM = DIGIT1= OUTPUT= p34
OSC = OSC = INPUT = p5	Y(4) = e = OUTPUT = p22	

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



รูปที่ L1.5 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป CPLD แล้วเดือน Slide SW1 และ Slide SW2 ไปที่ ON (CLR = '0' DIR = '0') และ Slide SW3 ไปที่ OFF CE = '1') ให้กดลงกดปุ่ม PB1 ไปเรื่อยๆ สลับกับการกด PB1 ค้างไว้ แล้วให้สังเกตคุณที่ตัวแสดงผลเซเว่นเซกเมนต์และ LED MN4 (สีแดง = '0', เหลือง = '1') ว่าติดสว่าง โดยให้ออกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ ซึ่งการกด

PB1 ค้างไว้จะให้ความถี่ 2 Hz (เท่ากับความถี่ที่ขา CLR) จากนั้นเลื่อน Slide SW1 ไปที่ OFF (CLR= '1') เพื่อเคลียร์เอาต์พุตแล้ว ON แล้วทดลองซ้ำ ในระหว่างทดลองให้การกด PB1 ค้างไว้แล้วเลื่อน Slide SW3 ไปที่ ON (CE= '0') เพื่อให้วงจรนับหยุดเดิน แล้ว OFF แล้วทดลองซ้ำ จากนั้นเลื่อน Slide SW2 ไปที่ OFF (DIR= '1') เพื่อนับลงแล้วทดลองซ้ำเดิมและบันทึกผลการทดลอง

2 สร้างโค้ดของวงจรนับ 10 ที่เป็นวงจรนับขึ้น-นับลงแบบคาสเคด 1 หลักด้วย FPGA

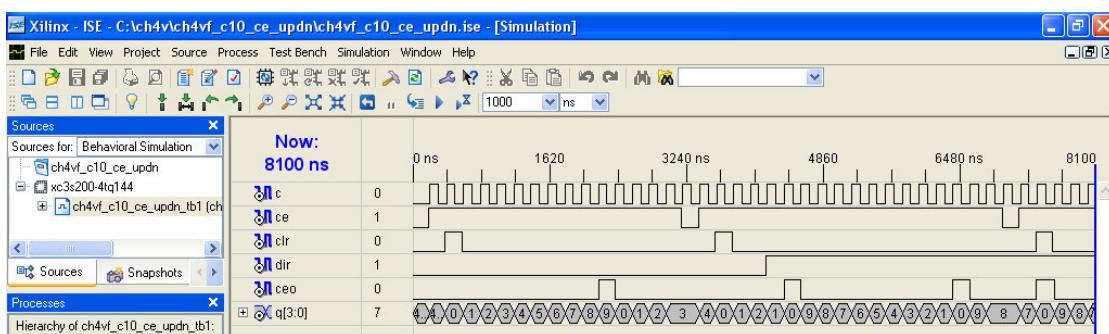
2.1) การทดลองสร้างโค้ดของวงจรนับ 10 ที่เป็นวงจรนับขึ้น-นับลงแบบคาสเคด 1 หลักด้วย FPGA จะเนื่องกับ CPLD ทุกประการ โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v ชื่อเดิม แต่กำหนด Project Name และ Source File เป็นชื่อ ch4vf_c10_ce_updn เสร็จแล้วจะได้ดังรูปที่ L2.1 ทำการบันทึกไฟล์และ Check syntax จากนั้นทำ Behavioral simulation โดยใช้ Source File ชื่อ ch4vf_c10_ce_updn_tb1 แล้วให้ดูผลจำลองการทำงานดังรูปที่ L2.2 ว่าเป็นไปตามทฤษฎีหรือไม่

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity ch4vf_c10_ce_updn is
7      Port ( C,CE,CLR,DIR : in STD_LOGIC;
8             CEO : out STD_LOGIC;
9             Q : out STD_LOGIC_VECTOR (3 downto 0));
10 end ch4vf_c10_ce_updn;
11
12 architecture Behavioral of ch4vf_c10_ce_updn is
13     signal QT : STD_LOGIC_VECTOR (3 downto 0);
14 begin
15     -----Counter with CE and CEO-----
16 process(C,CLR)
17 begin
18     if CLR='1' then QT <= "0000";
19     elsif (C'event and C='1') then
20         if CE='1' then
21             if DIR='0' then          -- Count up
22                 if QT>=9 then QT <= "0000";
23                 else QT <= QT + 1;
24                 end if;
25             else                      -- Count down
26                 if (QT=0 or QT>9) then QT <= "1001";
27                 else QT <= QT - 1;
28                 end if;
29             end if;
30         end if;
31     end if;
32 end process;
33 Q <= QT;                                -- Counter output
34 -----CEO-----
35 CEO <= CE when (DIR = '0' and QT=9) else --CEO-count up
36     CE when (DIR = '1' and QT=0) else --CEO-count down
37     '0';
38 end Behavioral;

```

รูปที่ L2.1 โค้ดของวงจรนับ 10 ที่เป็นวงจรนับขึ้น-นับลงแบบคาสเคดที่ใช้ทำเป็น Component



รูปที่ L2.2 ผล Behavioral simulation ของวงจรนับที่ใช้โค้ดในรูปที่ L2.1

2.2) นำไฟล์วงจรอุดรหัสตัวแสดงผลเลขฐานเซกเมนต์ (0-9) จากรูปที่ 2.29b) ในบทที่ 2 มาแก้ไขแล้วสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vf_HEX_to_7SEGMENT เสร็จแล้วจะได้ดังรูปที่ L2.3

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ch4vf_BCD_to_7SEGMENT is
6     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
7             Y : out STD_LOGIC_VECTOR (6 downto 0));
8 end ch4vf_BCD_to_7SEGMENT;
9
10 architecture Behavioral of ch4vf_BCD_to_7SEGMENT is
11 begin
12     -----BCD to 7-segments decoder-----
13     -- gfedcba
14     Y <= "1101111" when A="1001" else --9
15         "1111111" when A="1000" else --8      Y(0)=a
16         "0000111" when A="0111" else --7      ---
17         "1111101" when A="0110" else --6 Y(5)=f|  | Y(1)=b
18         "1101101" when A="0101" else --5      --- Y(6)=g
19         "1100110" when A="0100" else --4 Y(4)=e|  | Y(2)=c
20         "1001111" when A="0011" else --3      ---
21         "1011011" when A="0010" else --2      Y(3)=d
22         "0000110" when A="0001" else --1
23         "0111111";                         --0,A,b,C,d,E,F
24 end Behavioral;

```

รูปที่ L2.3 โภคดาวงจรอุดรหัสตัวแสดงผลเลขฐานเซกเมนต์แบบเลขฐานสิบแบบแคปติว์ร่วมที่ใช้ทำเป็น Component

2.3) เขียนโภคดาวงจรนับโดยสารไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_4_2vf เขียนโภคดาวงจรนับขีน 24 บิตโดยเลือกใช้ความถี่ 2.98 Hz เพื่อเคลียร์วงจรดีเบาเซอร์โดยอัตโนมัติทุกๆ = 0.34 วินาที งานนี้นำไฟล์วงจรดีเบาเซอร์ชื่อ ch4vf_debounce ไฟล์วงจรนับขีน-นับลงชื่อ ch4vf_c10_ce_updn และไฟล์วงจรอุดรหัสตัวแสดงผลเลขฐานเซกเมนต์ชื่อ ch4vf_BCD_to_7SEGMENT มาทำเป็น Component เสร็จแล้วจะได้ดังรูปที่ L2.3

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_4_2vf is
7     Port ( CE,CLR,Din,DIR,OSC : in STD_LOGIC;
8             CEO,COM : out STD_LOGIC;
9             Y : out STD_LOGIC_VECTOR (6 downto 0));
10 end ex4_4_2vf;
11
12 architecture Behavioral of ex4_4_2vf is
13     component ch4vf_DEBOUNCER
14         Port ( OSC,D_in,CLR : in STD_LOGIC;
15                 D_out : out STD_LOGIC);
16     end component;
17     component ch4vf_c10_ce_updn
18         Port ( C,CE,CLR,DIR : in STD_LOGIC;
19                 CEO : out STD_LOGIC;
20                 Q : out STD_LOGIC_VECTOR (3 downto 0));
21     end component;
22     component ch4vf_BCD_to_7SEGMENT
23         Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
24                 Y : out STD_LOGIC_VECTOR (6 downto 0));
25     end component;
26     signal C : STD_LOGIC;
27     signal F : STD_LOGIC_VECTOR (23 downto 0);
28     signal Q : STD_LOGIC_VECTOR (3 downto 0);
29 begin
30     -----23Bits Counter-----
31     process(OSC) --Output F(23)=1.49Hz,F(22)=2.98Hz,F(21)=5.96Hz
32         begin

```

```

33      if OSC'event and OSC='1' then F <= F + 1;
34      end if;
35  end process;
36  -----
37 DEBOUNCER : ch4vf_DEBOUNCER
38  port map(OSC=>OSC,D_in=>Din,CLR=>F(13),D_out=>C);
39  -----
40 COUNTER_10UPDN : ch4vf_c10_ce_updn
41  port map(C=>C,CE=>CE,CLR=>CLR,DIR=>DIR,CEO=>CEO,Q=>Q);
42  -----
43  COM <= '0'; -- Common cathode = GND
44 DECODER_7SEG : ch4vf_BCD_to_7SEGMENT
45  port map(A=>Q,Y=>Y);
46 end Behavioral;

```

รูปที่ L2.3 โค้ดของวงจรนับ 10 ที่เป็นวงจรนับขึ้น-ลงแบบค่าสเกลคือรวมวงจรดีเบาเซอร์ไว้แล้ว

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25 MHz และปุ่มกด PB1, Dip SW1-Dip SW3 เป็นอินพุต และมีตัวแสดงผลเซเว่นเซกเมนต์และ LED L4 เป็นเอาต์พุต กล่าวว่าคือ

Din = PB1 = INPUT = p44	Y(0) = a = OUTPUT = p40	Y(5) = f = OUTPUT = p25
CLR = Dip SW1 = INPUT = p52	Y(1) = b = OUTPUT = p35	Y(6) = g = OUTPUT = p23
DIR = Dip SW2 = INPUT = p53	Y(2) = c = OUTPUT = p32	CEO = L4 = OUTPUT = p74
CE = Dip SW3 = INPUT = p55	Y(3) = d = OUTPUT = p30	COM = DIGIT1= OUTPUT= p31
OSC = OSC = INPUT = p127	Y(4) = e = OUTPUT = p27	

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CE	Input	p55	BANK4	LVCMS33	N/A	3.30					Unknown		
CEO	Output	p74	BANK3	LVCMS33	N/A	3.30					Unknown		
CLR	Input	p52	BANK5	LVCMS33	N/A	3.30					Unknown		
COM	Output	p31	BANK6	LVCMS33	N/A	3.30					Unknown		
Din	Input	p44	BANK5	LVCMS33	N/A	3.30					Unknown		
DIR	Input	p53	BANK5	LVCMS33	N/A	3.30					Unknown		
OSC	Input	p127	BANK0	LVCMS33	N/A	3.30					Unknown		
Y<0>	Output	p40	BANK5	LVCMS33	N/A	3.30					Unknown		
Y<1>	Output	p35	BANK6	LVCMS33	N/A	3.30					Unknown		
Y<2>	Output	p32	BANK6	LVCMS33	N/A	3.30					Unknown		
Y<3>	Output	p30	BANK6	LVCMS33	N/A	3.30					Unknown		
Y<4>	Output	p27	BANK6	LVCMS33	N/A	3.30					Unknown		
Y<5>	Output	p25	BANK6	LVCMS33	N/A	3.30					Unknown		
Y<6>	Output	p23	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		

รูปที่ L2.4 Assign Package Pins

หลังจากโปรแกรม FPGA แล้วเดือน Dip SW1 และ Dip SW2 ไปที่ ON (CLR = '0' DIR = '0') และ Dip SW3 ไปที่ OFF CE = '1' ให้กดลงกดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกด PB1 ค้างไว้ แล้วให้สังเกตคุณที่ตัวแสดงผลเซเว่นเซกเมนต์ และ LED L4 ว่าติดสว่างโดยให้กดปุ่ม PB1 ไปตามทฤษฎีหรือไม่ ซึ่งการกด PB1 ค้างไว้จะให้ความถี่ 2.98 Hz (เท่ากับความถี่ที่ CLA) จากนั้นเลื่อน Dip SW1 ไปที่ OFF (CLR = '1') เพื่อเคลียร์เอาต์พุตแล้ว ON และกดลงช้า ในระหว่างกดลง ให้การกด PB1 ค้างไว้แล้วเลื่อน Dip SW3 ไปที่ ON (CE = '0') เพื่อให้วงจรนับหยุดเดินแล้ว OFF และกดลงช้า จากนั้นเลื่อน Dip SW2 ไปที่ OFF (DIR = '1') เพื่อนับลง แล้วกดลงช้าเดิมและบันทึกผลการทดสอบ

การทดลองที่ 4.4.3 การสร้างวงจรนับเลขฐานสิบที่เป็นวงจรนับขีน-นับลง 2 หลัก

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจหลักการทำงานของวงจรนับเลขฐานสิบที่เป็นวงจรนับขีน-นับลง 2 หลัก
- 2) เพื่อประยุกต์ใช้งานรีบูตเตอร์ (Debouncer) และวงจรดอครหัสตัวแสดงผลเซเว่นเซกเมนต์ (7-Segment decoder)
- 3) เพื่อประยุกต์ใช้งานรีบอตต์ 1 to 2 Decoder และ 2 to 1 Multiplexer สำหรับทำงานรีสแกน (Scan)
- 4) เพื่อสร้างวงจรนับขีน-นับลง โดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิป CPLD และ FPGA

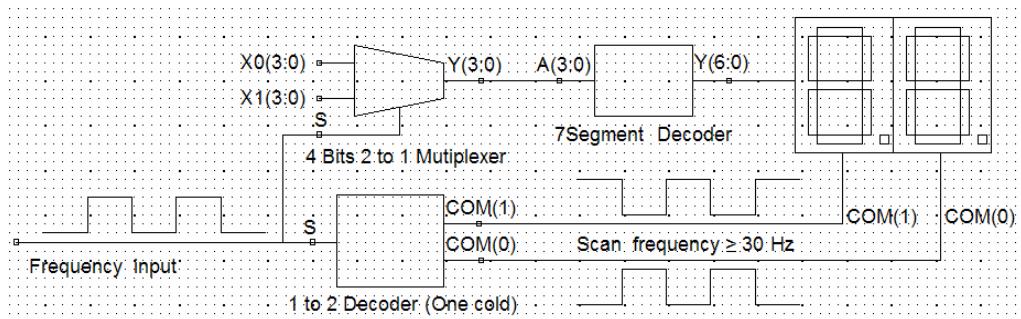
อุปกรณ์ทดลอง

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

1 สร้างวงจรนับเลขฐานสิบที่เป็นวงจรนับขีน-นับลง 2 หลักด้วย CPLD

1.1) ก่อนทำการทดลองให้ผู้อ่านทำความเข้าใจเกี่ยวกับวงจรรีสแกนและวงจรดอครหัสตัวแสดงผลเซเว่นเซกเมนต์ (แบบแอดโอดร์) 2 หลักแสดงดังรูปที่ L1.1 ซึ่งจะประกอบด้วย

- วงจรดอครหัส (1 to 2 Decoder) แบบ Active low คือ จะให้อาร์พุตเป็นลอจิก ‘0’ (หรือ One cold Decoder) โดยมีหลักการทำงานตามตารางความจริงในรูปที่ L1.2
- วงจรแมตติเพล็กเซอร์ (2 to 1 Multiplexer) 4 บิต โดยมีหลักการทำงานตามตารางความจริงในดังรูปที่ L1.3
- วงจรดอครหัสตัวแสดงผลเซเว่นเซกเมนต์



รูปที่ L1.1 วงจรรีสแกนและดอครหัสตัวแสดงผลทางเซเว่นเซกเมนต์ 2 หลัก

Input S	Output	
	COM(0)	COM(1)
0	0	1
1	1	0

รูปที่ L1.2 ตารางความจริงของวงจรดอครหัส 1 to 2 Decoder (One cold Decoder)

Control S	Output
0	$Y(3:0) = X0(3:0)$
1	$Y(3:0) = X1(3:0)$

รูปที่ L1.3 ตารางความจริงของวงจรแมตติเพล็กเซอร์ (2 to 1 Multiplexer) 4 บิต

หลักการทำงานของสแกน ซึ่งประกอบด้วยของร์มัลติเพล็กเซอร์และวงจร One cold Decoder จะเป็นดังนี้คือ

- เมื่อ $S = '0'$ วงจรแมติเพล็กเซอร์จะเลือก $X0(3:0)$ 送ไปที่วงจรดอร์หัสตัวแสดงผลเซเว่นเซกเมนต์เพื่อไปขับตัวแสดงผลเซเว่นเซกเมนต์ทั้ง 2 หลัก ในขณะเดียวกันวงจร One cold Decoder จะดอร์หัสให้ $COM(0) = '0'$ และ $COM(1) = '1'$ ทำให้ตัวแสดงผลเซเว่นเซกเมนต์หลักหน่วยติดสว่างเพียงหลักเดียว
- เมื่อ $S = '1'$ วงจรแมติเพล็กเซอร์จะเลือก $X1(3:0)$ 送ไปที่วงจรดอร์หัสตัวแสดงผลเซเว่นเซกเมนต์เพื่อไปขับตัวแสดงผลเซเว่นเซกเมนต์ทั้ง 2 หลัก ในขณะเดียวกันวงจร One cold Decoder จะดอร์หัสให้ $COM(0) = '1'$ และ $COM(1) = '0'$ ทำให้ตัวแสดงผลเซเว่นเซกเมนต์หลักสิบติดสว่างเพียงหลักเดียว
- ความถี่ที่ขา $COM(0)$ และ $COM(1)$ เพื่อใช้ในการสแกนที่ขาแคปติกอร์รัมของเซเว่นเซกเมนต์แต่ละหลัก จะต้อง ≥ 30 ครั้งต่อวินาที จึงจะทำให้ตัวแสดงผลเซเว่นเซกเมนต์ติดสว่างทั้ง 2 หลักโดยไม่เห็นการกระพริบ ดังนั้นในการนี้ความถี่อยู่ที่ $S \geq 30 \text{ Hz}$

1.2 สร้างไฟล์ของวงจรนับเลขฐานสิบที่เป็นวงจรนับขึ้น-นับลง 2 หลักไว้ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_4_3vcx1 จากนั้นนำไฟล์วงจรดีเบาเซอร์ชื่อ ch4vcx1_debounce และไฟล์วงจรนับขึ้น-นับลงชื่อ ch4vcx1_c10_ce_updn มาทำเป็น Component และป้อนโค้ดดังรูปที่ L1.4 โดยรวมโค้ดของวงจรนับขึ้น 15 บิต โค้ดของวงจรดอร์หัสตัวแสดงผลเซเว่นเซกเมนต์เป็นเลขฐานสิบ โค้ดของร์มัลติเพล็กเซอร์ (2 to 1 Multiplexer) 4 บิตและโค้ดของวงจรดอร์หัส 1 to 2 Decoder (One cold Decoder) ไว้แล้ว วงจนี้จะเคลียร์ค่าเอ้าต์พุตเป็น 00 เมื่อ $CLR = '0'$ และจะนับขึ้นเมื่อ $DIR = '0'$ แต่ถ้า $DIR = '1'$ จะเป็นการนับลง ในการทดลองนี้จะใช้ความถี่ 2 Hz จากเอ้าต์พุต F(13) ของวงจรนับขึ้น 15 บิตเพื่อเคลียร์เอ้าต์พุตของวงจรดีเบาเซอร์โดยอัตโนมัติทุกๆ 0.5 วินาที และความถี่ 64 Hz จากเอ้าต์พุต F(8) จ่ายให้วงจรสแกน

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_4_3vcx1 is
7     Port ( CE,CLR,Din,DIR,OSC : in STD_LOGIC;
8             COM : out STD_LOGIC_VECTOR (1 downto 0);
9             Y : out STD_LOGIC_VECTOR (6 downto 0));
10 end ex4_4_3vcx1;
11
12 architecture Behavioral of ex4_4_3vcx1 is
13     component ch4vcx1_DEBOUNCER
14         Port ( OSC,D_in,CLR : in STD_LOGIC;
15                 D_out : out STD_LOGIC);
16     end component;
17     component ch4vcx1_c10_ce_updn
18         Port ( C,CE,CLR,DIR : in STD_LOGIC;
19                 CEO : out STD_LOGIC;
20                 Q : out STD_LOGIC_VECTOR (3 downto 0));
21     end component;
22     signal C : STD_LOGIC;
23     signal F : STD_LOGIC_VECTOR (14 downto 0);
24     signal CE1 : STD_LOGIC;
25     signal XO,X1,A : STD_LOGIC_VECTOR (3 downto 0);
26 begin
27     -----15Bits Counter-----
28     process(OSC) --Output F(14)=1Hz,F(13)=2Hz,F(12)=4Hz,F(11)=8Hz
29     begin
30         if OSC'event and OSC='1' then F <= F + 1;
31         end if;
32     end process;
33     -----DEBOUNCER-----
34     DEBOUNCER : ch4vcx1_DEBOUNCER
35         port map(OSC=>OSC,D_in=>Din,CLR=>F(13),D_out=>C);

```

(ต่อ)

```

36 -----COUNTER : 2 Digits-----
37 COUNTER4BIT_DIGIT_0 : ch4vcx1_c10_ce_updn
38   port map(C=>C,CE=>CE,CLR=>CLR,DIR=>DIR,CEO=>CE1,Q=>X0);
39 COUNTER4BIT_DIGIT_1 : ch4vcx1_c10_ce_updn
40   port map(C=>C,CE=>CE1,CLR=>CLR,DIR=>DIR,CEO=>open,Q=>X1);
41 -----Scan circuit-----
42 -----4Bits MUX2to1-----
43   A <= X0 when F(8)='0' else
44     X1;
45 -----1to2 Decoder(One cold Decoder)-----
46   COM <= "10" when F(8)='0' else
47     "01";
48 -----BCD to 7-segments decoder-----
49   gfedcba
50   Y <= "1101111" when A="1001" else --9
51     "1111111" when A="1000" else --8      Y(0)=a
52     "0000111" when A="0111" else --7      ---
53     "1111101" when A="0110" else --6      Y(5)=f | Y(1)=b
54     "1101101" when A="0101" else --5      --- Y(6)=g
55     "1100110" when A="0100" else --4      Y(4)=e | Y(2)=c
56     "1001111" when A="0011" else --3      ---
57     "1011011" when A="0010" else --2      Y(3)=d
58     "0000110" when A="0001" else --1      ---
59     "0111111";                           --0,a,b,C,d,E,F
60 end Behavioral;

```

รูปที่ L1.4 โค้ดวงจรนับเลขฐานสิบที่เป็นวงจรนับขีน-นับลง 2 หลักและเรียกใช้ Std_logic_unsigned package

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1, PB3(Slide SW1) ถึง PB5(Slide SW3) เป็นอินพุต มีตัวแสดงผลเซเว่นเซกเมนต์และแคปติว่ามูลของ DIGIT1 และ DIGIT2 เป็นอ่าต์พุต กล่าวคือ

Din = PB1 = INPUT = p39	Y(0) = a = OUTPUT = p27	Y(5) = f = OUTPUT = p20
CLR = PB3(Slide SW1) = INPUT = p42	Y(1) = b = OUTPUT = p26	Y(6) = g = OUTPUT = p18
DIR = PB4(Slide SW2) = INPUT = p43	Y(2) = c = OUTPUT = p25	COM(1) = DIGIT2 = OUTPUT = p33
CE = PB5(Slide SW3) = INPUT = p44	Y(3) = d = OUTPUT = p24	COM(0) = DIGIT1 = OUTPUT = p34
OSC = OSC = INPUT = p5	Y(4) = e = OUTPUT = p22	

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]						
I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
CE	Input	p44	2	17		
CLR	Input	p42	2	14		
Din	Input	p39	2	9		
DIR	Input	p43	2	15		
OSC	Input	p5	1	9		
COM<1>	Output	p33	4	15	SLOW	
COM<0>	Output	p34	4	17	SLOW	
Y<6>	Output	p18	3	11	SLOW	
Y<5>	Output	p20	3	15	SLOW	
Y<4>	Output	p22	3	17	SLOW	
Y<3>	Output	p24	3	16	SLOW	
Y<2>	Output	p25	4	2	SLOW	
Y<1>	Output	p26	4	5	SLOW	
Y<0>	Output	p27	4	8	SLOW	

รูปที่ L1.5 Assign Package Pins

หลังจากโปรแกรม CPLD แล้วเลื่อน Slide SW1 และ Slide SW2 ไปที่ ON (CLR = '0' DIR = '0') และ Slide SW3 ไปที่ OFF CE = '1' ให้กดปุ่ม PB1 ไปเรื่อยๆ แล้วกับการกด PB1 ค้างไว้ แล้วให้สังเกตคุณตัวที่ตัวแสดงผลเซกเมนต์ว่าติดสว่างโดยไห้ล้อจิกເອາต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นเลื่อน Slide SW1 ไปที่ OFF (CLR = '1') เพื่อเคลียร์ເອາต์พุตแล้ว ON แล้วกดลงซ้ำ ในระหว่างกดลงให้การกด PB1 ค้างไว้แล้วเลื่อน Slide SW3 ไปที่ ON (CE = '0') เพื่อให้งานนับลงหยุดเดิน แล้ว OFF แล้วกดลงซ้ำ เลื่อน Slide SW2 ไปที่ OFF (DIR = '1') เพื่อนับลงแล้วทำการทดสอบซ้ำเดิมและบันทึกผลการทดสอบ

2 สร้างวงจรนับเลขฐานสิบที่เป็นวงจรนับขีน-นับลง 2 หลักด้วย FPGA

- 2.1) ให้ผู้อ่านทำการเขียนเกี่ยวกับวงจรสแกนและวงจรอคูเตอร์สตัวแสดงผลเซเว่นเซกเมนต์ 2 หลักในข้อ 1.1)
- 2.2) สร้างไฟล์ของวงจรนับเลขฐานสิบที่เป็นวงจรนับขีน-นับลง 2 หลักไว้ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_4_3vf นำไฟล์วงจรดีเบาเชอร์ชื่อ ch4vf_debounce และไฟล์วงจรนับขีน-นับลงชื่อ ch4vf_c10_ce_updn มาทำเป็น Component จากนั้นทำการเขียนโค้ดดังรูปที่ L2.1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_4_3vf is
7     Port ( CE,CLR,Din,DIR,OSC : in STD_LOGIC;
8             COM : out STD_LOGIC_VECTOR (1 downto 0);
9             Y : out STD_LOGIC_VECTOR (6 downto 0));
10 end ex4_4_3vf;
11
12 architecture Behavioral of ex4_4_3vf is
13     component ch4vf_DEBOUNCER
14         Port ( OSC,D_in,CLR : in STD_LOGIC;
15                 D_out : out STD_LOGIC);
16     end component;
17     component ch4vf_c10_ce_updn
18         Port ( C,CE,CLR,DIR : in STD_LOGIC;
19                 CEO : out STD_LOGIC;
20                 Q : out STD_LOGIC_VECTOR (3 downto 0));
21     end component;
22     signal C : STD_LOGIC;
23     signal F : STD_LOGIC_VECTOR (23 downto 0);
24     signal CE1 : STD_LOGIC;
25     signal XO,X1,A : STD_LOGIC_VECTOR (3 downto 0);
26 begin
27     -----24Bits Counter-----
28 process(OSC) --Output F(23)=1.49Hz,F(22)=2.98Hz,F(21)=5.96Hz
29 begin
30     if OSC'event and OSC='1' then F <= F + 1;
31     end if;
32 end process;
33 -----DEBOUNCER-----
34 DEBOUNCER : ch4vf_DEBOUNCER
35     port map(OSC=>OSC,D_in=>Din,CLR=>F(22),D_out=>C);
36 -----COUNTER : 2 Digits-----
37 COUNTER4BIT_DIGIT_0 : ch4vf_c10_ce_updn
38     port map(C=>C,CE=>CE,CLR=>CLR,DIR=>DIR,CEO=>CE1,Q=>XO);
39 COUNTER4BIT_DIGIT_1 : ch4vf_c10_ce_updn
40     port map(C=>C,CE=>CE1,CLR=>CLR,DIR=>DIR,CEO=>open,Q=>X1);
41 -----Scan circuit-----
42 -----4Bits MUX2to1-----
43     A <= XO when F(18)='0' else
44             X1;
45 -----1to2 Decoder(One cold Decoder)-----
46 COM <= "10" when F(18)='0' else
47             "01";
48 -----BCD to 7-segments decoder-----
49     -- gfedcba
50     Y <= "1101111" when A="1001" else --9
51             "1111111" when A="1000" else --8      Y(0)=a
52             "0000111" when A="0111" else --7      ---
53             "1111101" when A="0110" else --6 Y(5)=f|  Y(1)=b
54             "1101101" when A="0101" else --5      --- Y(6)=g
55             "1100110" when A="0100" else --4 Y(4)=e|  Y(2)=c
56             "1001111" when A="0011" else --3      ---
57             "1011011" when A="0010" else --2      Y(3)=d
58             "0000110" when A="0001" else --1      ---
59             "0111111";                      --0,a,b,C,d,E,F
60 end Behavioral;

```

รูปที่ L2.1 โค้ดวงจรนับเลขฐานสิบที่เป็นวงจรนับขีน-นับลง 2 หลักและเรียกใช้ Std_logic_unsigned package

ໂຄຳດໃນຮູບທີ L2.1 ຈະຮວມໂຄຳດຂອງວົງຈຽນບື້ນ 24 ບິຕ ໂຄຳດວງຈຣອດຮ້າສັກແສດງພລເຊວນເຊກເມນຕີເປັນເລຂຮ້ານສົນໄຄຳດວງຈຣມລັດເພີລີກເຊ່ອຮ້ (2 to 1 Multiplexer) 4 ບິຕ ແລະ ໂຄຳດຂອງວົງຈຣອດຮ້າສ 1 to 2 Decoder (One cold Decoder) ໄວ້ແລ້ວວົງຈຽນນີ້ຈະເຄລີຍີ່ຄ່າເອົາດີພຸດເປັນ 00 ເມື່ອ CLR = ‘0’ ແລະ ຈະນັບບື້ນເມື່ອ DIR = ‘0’ ແຕ່ຖ້າ DIR = ‘1’ ຈະເປັນການນັບລົງ ກາຮທດລອງນີ້ຈະໃຫ້ຄວາມຄື 2.98 Hz ຈາກເອົາດີພຸດ F(22) ຂອງວົງຈຽນບື້ນ 24 ບິຕເພື່ອເຄລີຍີ່ເອົາດີພຸດຂອງວົງຈຣີບໍາເຊ່ອຮ້ໄດ້ຢັດໄວມັດຖຸກາ 0.34 ວິນາທີແລະໃຫ້ຄວາມຄື 47.68 Hz ຈາກເອົາດີພຸດ F(18) ຈ້າຍໃຫ້ວົງຈຣສແກນ

ກາຮການນັດຫາສັ່ນງານຕ່າງໆ ຈະໃຫ້ອສໜີລາເດເຕອຮ້ OSC = 25 MHz ແລະ ປຸ່ມກົດ PB1, Dip SW1-Dip SW3 ເປັນອິນພຸດ ແລະ ມີຕົວແສດງພລເຊວນເຊກເມນຕີແລະ ແກ້ໂຄດຮ່ວມຂອງ DIGIT1 ແລະ DIGIT2 ເປັນເອົາດີພຸດ ກລ່າວກືອ

Din = PB1 = INPUT = p44	Y(0) = a = OUTPUT = p40	Y(5) = f = OUTPUT = p25
CLR= Dip SW1 = INPUT = p52	Y(1) = b = OUTPUT = p35	Y(6) = g = OUTPUT = p23
DIR = Dip SW2 = INPUT = p53	Y(2) = c = OUTPUT = p32	COM(1)= DIGIT2= OUTPUT=p33
CE = Dip SW3 = INPUT = p55	Y(3) = d = OUTPUT = p30	COM(0)= DIGIT1= OUTPUT=p31
OSC= OSC = INPUT = p127	Y(4) = e = OUTPUT = p27	

ໂດຍພິມພຶກໃນ Assign Package Pins ສາງປັດນີ້

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CE	Input	p55	BANK4	LVCMS33	N/A	3.30					Unknown		
CLR	Input	p52	BANK5	LVCMS33	N/A	3.30					Unknown		
COM<0>	Output			LVCMS33	N/A	3.30		SLOW			Unknown		
COM<1>	Output	p33	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Din	Input	p44	BANK5	LVCMS33	N/A	3.30					Unknown		
DIR	Input	p53	BANK5	LVCMS33	N/A	3.30					Unknown		
OSC	OSC	p127	BANK0	LVCMS33	N/A	3.30					Unknown		
Y<0>	Output	p40	BANK5	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<1>	Output	p35	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<2>	Output	p32	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<3>	Output	p30	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<4>	Output	p27	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<5>	Output	p25	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<6>	Output	p23	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		

ຮູບທີ L2.2 Assign Package Pins

ໜັງຈາກໂປຣແກຣມວົງຈຣລົງໝຶກ FPG ແລ້ວເລືອນ Dip SW1 ແລະ Dip SW2 ໄປທີ່ ON (CLR = ‘0’ DIR = ‘0’) ແລະ Dip SW3 ໄປທີ່ OFF CE = ‘1’) ໃຫ້ກົດປຸ່ມ PB1 ໄປເຮືອຍໆ ສລັບກັບກາຮກົດ PB1 ຄ້າງໄວ້ແລ້ວໃຫ້ຄູພີທີ່ຕົວແສດງພລເຊວນເຊກເມນຕີວ່າຕິດສ່ວ່າງໂດຍໃຫ້ເອົາດີພຸດເປັນໄປຄາມທຸກ໌ທີ່ໄວ້ມີ ຈາກນັ້ນເລືອນ Dip SW1 ໄປທີ່ OFF (CLR= ‘1’) ເພື່ອເຄລີຍີ່ເອົາດີພຸດແລ້ວ ON ແລ້ວກົດລອງໜ້າ ໃນຮະຫວ່າງກົດລອງໃຫ້ກາຮກົດ PB1 ຄ້າງໄວ້ແລ້ວເລືອນ Dip SW3 ໄປທີ່ ON (CE= ‘0’) ເພື່ອໃຫ້ວົງຈຽນຫຼຸດເດີນແລ້ວ OFF ແລ້ວກົດລອງໜ້າ ເລືອນ Dip SW2 ໄປທີ່ OFF (DIR= ‘1’) ເພື່ອນັບລົງເພື່ອນັບລົງແລ້ວທຳກາຮກົດລອງໜ້າເດີນແລະບັນທຶກກາຮກົດລອງ

ແບບຜຶກຫັດ

- 1) ໃຫ້ຫາຄວາມຄືສແກນທີ່ເໝາະສົມວ່າກວາມເປັນເທົ່າໄດ ໂດຍແກ້ໄຂໂຄຳດບຣທັດທີ 43 ແລະ ບຣທັດທີ 46 ໃນຮູບທີ L1.4 ຈາກ F(8) ຜົ່ງເປັນຄວາມຄື 64 Hz ເປັນ F(11) ຜົ່ງເປັນຄວາມຄື 8 Hz ແລະ ເມື່ອດາວຸ່ນໄຫວດີໂລດ ໂຄຳດໃໝ່ລົງ CPLD ເຮັດວຽກແລ້ວໃຫ້ສັງເກດກາຮກົດກະພວບທີ່ຕົວແສດງພລ ຈາກນັ້ນທຳກາຮກົດລອງໜ້າໂດຍແກ້ໄຂເປັນ F(10) (ຫົວ້າ 16 Hz), F(9) (ຫົວ້າ 32 Hz) ແລະ F(8) (ຫົວ້າ 64 Hz) ຕາມລຳດັບ
- 2) ຫົວ້າໃຫ້ຫາຄວາມຄືສແກນທີ່ເໝາະສົມວ່າກວາມເປັນເທົ່າໄດ ໂດຍແກ້ໄຂໂຄຳດບຣທັດທີ 43 ແລະ ບຣທັດທີ 46 ໃນຮູບທີ L2.1 ຈາກ F(18) ຜົ່ງເປັນຄວາມຄື 47.68 Hz ເປັນ F(20) ຜົ່ງເປັນຄວາມຄື 11.92 Hz ແລະ ເມື່ອດາວຸ່ນໄຫວດີໂລດ ໂຄຳດໃໝ່ລົງ FPGA ແລ້ວໃຫ້ສັງເກດກາຮກົດກະພວບທີ່ຕົວແສດງພລ ຈາກນັ້ນທຳກາຮກົດລອງໜ້າໂດຍແກ້ໄຂເປັນ F(19) (ຫົວ້າ 23.84 Hz) ແລະ F(18) (ຫົວ້າ 47.68 Hz) ຕາມລຳດັບ

กາຮທດລອງທີ່ 4.4.4 ກາຮສ້າງວົງຈຽນນັບເລຂ້າງສົບທີ່ເປັນວົງຈຽນບິ່ນ-ນັບລົງ 4 ພັກ

ວັດຖຸປະສົງຄໍ

- 1) ເພື່ອທຳຄວາມເຂົ້າໃຈຫຼັກກາຮທຳງານວົງຈຽນນັບເລຂ້າງສົບທີ່ເປັນວົງຈຽນບິ່ນ-ນັບລົງ 4 ພັກ
- 2) ເພື່ອປະຢູກຕີໃໝ່ຈຽນເບົາເຊ່ອຣ (Debouncer) ແລະ ວົງຈຽນຄອດຮ້າສຕັວແສດງພລເຊວນເຊກເມນຕໍ່ (7-Segment decoder)
- 3) ເພື່ອປະຢູກຕີໃໝ່ຈຽນຄອດຮ້າສ 2 to 4 Decoder ແບບ One cold ແລະ 4 to 1 Multiplexer ສໍາຫຼັບທຳງານສະແກນ (Scan)
- 4) ເພື່ອສ້າງວົງຈຽນນັບບິ່ນ-ນັບລົງ ໂດຍວິທີເປີຍນັດ້ວຍໂຄ້ດ VHDL ແລະ ໂປຣແກຣມລົງໝຶກ CPLD ແລະ FPGA

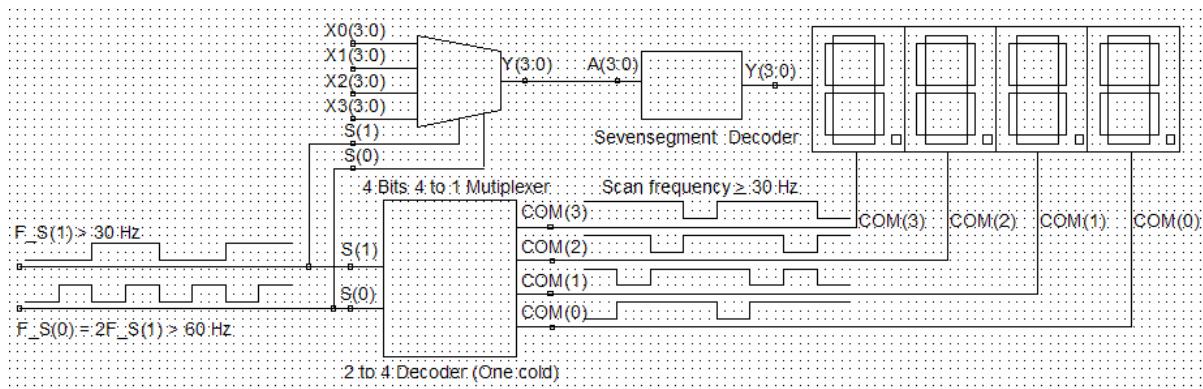
ອຸປະກອດທິດລອງ

ບ່ອຮົດ CPLD Explorer XC9572XL ແລະ FPGA Discovery-III XC3S200F ມີໂຄງການ

1 ສ້າງວົງຈຽນນັບເລຂ້າງສົບທີ່ເປັນວົງຈຽນບິ່ນ-ນັບລົງ 4 ພັກດ້ວຍ CPLD

1.1) ກ່ອນທຳກາຮທດລອງໃຫ້ຜູ້ອ່ານທຳຄວາມເຂົ້າໃຈເກີຍກັບວົງຈຽນສະແກນແລະ ວົງຈຽນຄອດຮ້າສຕັວແສດງພລເຊວນເຊກເມນຕໍ່ (ແບບແຄໂຄດ ຮ່ວມ) 4 ພັກແສດງດັ່ງຮູບປີ່ L1.1 ຜື້ຈະປະກອບດ້ວຍ

- ວົງຈຽນຄອດຮ້າສ (2 to 4 Decoder) ແບບ Active low ກີ່ອ ຂະໃໜ້ເອົາຕີພຸດເປັນລອອິກ ‘0’ (ຫຸ້ອ One cold Decoder) ໂດຍມີຫຼັກກາຮທຳງານຕາມຕາງຄວາມຈິງໃນຮູບປີ່ L1.2
- ວົງຈຽນມັດຕີເພີ້ເກຊ່ອຣ (4 to 1 Multiplexer) 4 ປີຕ ໂດຍມີຫຼັກກາຮທຳງານຕາມຕາງຄວາມຈິງໃນດັ່ງຮູບປີ່ L1.3
- ວົງຈຽນຄອດຮ້າສຕັວແສດງພລເຊວນເຊກເມນຕໍ່



ຮູບປີ່ L1.1 ວົງຈຽນສະແກນແລະ ອົງຄອດຮ້າສຕັວແສດງພລທາງເຊກເມນຕໍ່ 4 ພັກ

Input S		Output COM			
S(1)	S(0)	COM(3)	COM(2)	COM(1)	COM(0)
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

ຮູບປີ່ L1.2 ຕາງຄວາມຈິງຂອງ 2 to 4 Decoder (One cold Decoder)

Input S		Output
S(1)	S(0)	
0	0	Y(3:0) = X0(3:0)
0	1	Y(3:0) = X1(3:0)
1	0	Y(3:0) = X2(3:0)
1	1	Y(3:0) = X3(3:0)

รูปที่ L1.3 ตารางความจริงของวงจรแมตติเพล็กเซอร์ (4 to 1 Multiplexer) 4 บิต

หลักการทำงานของสแกน ซึ่งประกอบด้วยวงจรแมตติเพล็กเซอร์และวงจร One cold Decoder จะเป็นดังนี้คือ

- เมื่อ S = "00" วงจรแมตติเพล็กเซอร์จะเลือกสัญญาณ X0(3:0) ไปที่เอาต์พุตและขา COM(0) = '0'
- เมื่อ S = "01" วงจรแมตติเพล็กเซอร์จะเลือกสัญญาณ X1(3:0) ไปที่เอาต์พุตและขา COM(1) = '0'
- เมื่อ S = "10" วงจรแมตติเพล็กเซอร์จะเลือกสัญญาณ X2(3:0) ไปที่เอาต์พุตและขา COM(2) = '0'
- เมื่อ S = "11" วงจรแมตติเพล็กเซอร์จะเลือกสัญญาณ X3(3:0) ไปที่เอาต์พุตและขา COM(3) = '0'

ความถี่สแกนที่ขาคอมมอนแคปติกองแต่ละหลักเพื่อทำให้เวลาเชกเม้นต์ติดสว่างโดยไม่เห็นการกระพริบนั้นจะต้องไม่น้อยกว่า 30 ครั้งต่อวินาที ดังนั้นความถี่ที่ป้อนให้ S(1) และ S(0) จึงต้องไม่น้อยกว่า 30 Hz และ 60 Hz ตามลำดับ

1.2) สร้างไฟล์วงจรนับขึ้น-นับลง 4 หลักซึ่งนับค่า 0000-9999 ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_4_4vcx1 นำไฟล์วงจรดีเบนาเซอร์ชื่อ ch4vcx1_debounce ไฟล์วงจรนับขึ้น-นับลงชื่อ ch4vcx1_c10_ce_updn และไฟล์วงจรจดจำรหัสตัวแสดงผลเฉพาะเวนเชกเม้นต์ชื่อ ch4vcx1_BCD_to_7SEGMENT มาทำเป็น Component ทำการเขียนโค้ดดังรูปที่ L1.4 ซึ่งรวมโค้ดของวงจรนับขึ้น 15 บิต โค้ดของวงจรแมตติเพล็กเซอร์ (4 to 1 Multiplexer) 4 บิตและโค้ดของวงจรจดจำรหัส 2 to 4 Decoder (One cold Decoder) ไว้แล้ว ซึ่งวงจรนี้จะเคลียร์เอาต์พุตเป็น 0000 เมื่อ CLR = '0' และจะนับขึ้นเมื่อ DIR = '0' แต่ถ้า DIR = '1' จะเป็นการนับลง การทดสอบนี้จะใช้ความถี่ 2 Hz จากเอาต์พุต F(13) ของวงจรนับขึ้น 15 บิตเพื่อเคลียร์เอาต์พุตของวงจรดีเบนาเซอร์โดยอัตโนมัติทุกๆ $1/2 \text{ Hz} = 0.5 \text{ วินาที}$ และจะใช้ความถี่ 64 Hz และ 128 Hz จากเอาต์พุต F(8) และ F(7) ของวงจรนับขึ้น 15 บิตจ่ายให้วงจรสแกนเพื่อสแกนแต่ละหลักที่ความถี่ $64 > 30 \text{ Hz}$)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_4_4vcx1 is
7     Port ( CE,CLR,Din,DIR,OSC : in STD_LOGIC;
8             COM : out STD_LOGIC_VECTOR (3 downto 0);
9             Y : out STD_LOGIC_VECTOR (6 downto 0));
10 end ex4_4_4vcx1;
11
12 architecture Behavioral of ex4_4_4vcx1 is
13     component ch4vcx1_DEBOUNCER
14         Port ( OSC,D_in,CLR : in STD_LOGIC;
15                 D_out : out STD_LOGIC);
16     end component;
17     component ch4vcx1_c10_ce_updn
18         Port ( C,CE,CLR,DIR : in STD_LOGIC;
19                 CEO : out STD_LOGIC;
20                 Q : out STD_LOGIC_VECTOR (3 downto 0));
21     end component;
22     component ch4vcx1_BCD_to_7SEGMENT
23         Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
24                 Y : out STD_LOGIC_VECTOR (6 downto 0));
25     end component;
26     signal C : STD_LOGIC;
27     signal F : STD_LOGIC_VECTOR (14 downto 0);
28     signal CE1,CE2,CE3 : STD_LOGIC;
29     signal X0,X1,X2,X3,A : STD_LOGIC_VECTOR (3 downto 0);

```

(ต่อ)

```

30 begin
31 -----15Bits Counter-----
32 process(OSC) --Output F(14)=1Hz,F(13)=2Hz,F(12)=4Hz,F(11)=8Hz
33 begin
34     if OSC'event and OSC='1' then F <= F + 1;
35     end if;
36 end process;
37 -----DEBOUNCER-----
38 DEBOUNCER : ch4vcx1_DEBOUNCER
39     port map(OSC=>OSC,D_in=>Din,CLR=>F(13),D_out=>C);
40 -----COUNTER : 4 Digits-----
41 COUNTER_DIGIT_0 : ch4vcx1_c10_ce_updn
42     port map(C=>C,CE=>CE,CLR=>CLR,DIR=>DIR,CEO=>CE1,Q=>X0);
43 COUNTER_DIGIT_1 : ch4vcx1_c10_ce_updn
44     port map(C=>C,CE=>CE1,CLR=>CLR,DIR=>DIR,CEO=>CE2,Q=>X1);
45 COUNTER_DIGIT_2 : ch4vcx1_c10_ce_updn
46     port map(C=>C,CE=>CE2,CLR=>CLR,DIR=>DIR,CEO=>CE3,Q=>X2);
47 COUNTER_DIGIT_3 : ch4vcx1_c10_ce_updn
48     port map(C=>C,CE=>CE3,CLR=>CLR,DIR=>DIR,CEO=>open,Q=>X3);
49 -----Scan circuit-----
50 -----4Bits MUX4to1-----
51 A <= X0      when F(8 downto 7)="00" else
52     X1      when F(8 downto 7)="01" else
53     X2      when F(8 downto 7)="10" else
54     X3;
55 -----2to4 Decoder(One cold Decoder)-----
56 COM <= "1110" when F(8 downto 7)="00" else
57     "1101" when F(8 downto 7)="01" else
58     "1011" when F(8 downto 7)="10" else
59     "0111";
60 -----7 Segment decoder-----
61 DECODER_7SEG : ch4vcx1_BCD_to_7SEGMENT
62     port map(A=>A,Y=>Y);
63 end Behavioral;

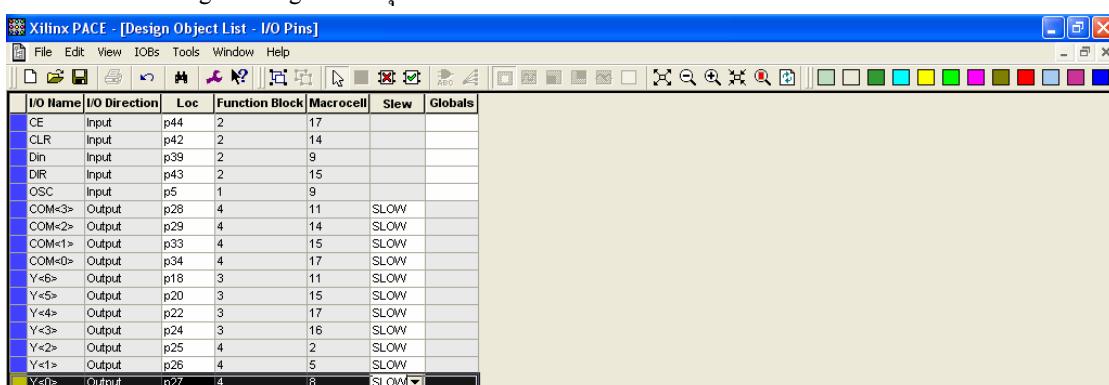
```

รูปที่ L1.4 โค้ดวงจรนับเลขฐานสิบที่เป็นวงจรนับขึ้น-ลง 4 หลัก

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1, PB3(Slide SW1) ถึง PB5(Slide SW3) เป็นอินพุต มีตัวแสดงผลเซเว่นเซกเมนต์และแคปโอดร่วมของ DIGIT1-DIGIT4 เป็นเอาต์พุต กล่าวคือ

Din = PB1 = INPUT = p39	Y(0) = a = OUTPUT = p27	Y(5) = f = OUTPUT = p20
CLR = PB3(Slide SW1) = INPUT = p42	Y(1) = b = OUTPUT = p26	Y(6) = g = OUTPUT = p18
DIR = PB4(Slide SW2) = INPUT = p43	Y(2) = c = OUTPUT = p25	COM(3) = DIGIT2 = OUTPUT = p28
CE = PB5(Slide SW3) = INPUT = p44	Y(3) = d = OUTPUT = p24	COM(2) = DIGIT1 = OUTPUT = p29
OSC = OSC = INPUT = p5	Y(4) = e = OUTPUT = p22	COM(1) = DIGIT1 = OUTPUT = p33
		COM(0) = DIGIT1 = OUTPUT = p34

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



รูปที่ L1.5 Assign Package Pins

หลังจากโปรแกรม CPLD แล้วเลื่อน Slide SW1 และ Slide SW2 ไปที่ ON (CLR = '0' DIR = '0') และ Slide SW3 ไปที่ OFF CE = '1') ให้กดปุ่ม PB1 ไปเรื่อยๆ ลับกันการกด PB1 ค้างไว้แล้วให้สังเกตคุณผลที่ตัวแสดงผลบนเซกเมนต์ว่าติดสว่างโดยใช้หลอดจีเอต์พุดเป็นไปตามทฤษฎีหรือไม่ จากนั้นเลื่อน Slide SW1 ไปที่ OFF (CLR= '1') เพื่อเคลียร์เอาต์พุดแล้ว ON แล้วทดลองซ้ำ ในระหว่างทดลองให้การกด PB1 ค้างไว้แล้วเลื่อน Slide SW3 ไปที่ตำแหน่ง ON (CE= '0') เพื่อให้วงจรนับหยุดเดินแล้ว OFF แล้วทดลองซ้ำ เลื่อน Slide SW2 ไปที่ OFF (DIR= '1') เพื่อนับลงแล้วทดลองซ้ำและบันทึกผลการทดลอง

2 สร้างวงจรนับเลขฐานสิบที่เป็นวงจรนับขึ้น-นับลง 4 หลักด้วย FPGA

- 2.1) ให้ผู้อ่านทำความเข้าใจเกี่ยวกับวงจรสแกนและวงจรลดรหัสตัวแสดงผลบนเซกเมนต์ 4 หลักในข้อ 1.1)
- 2.2) สร้างไฟล์วงจรนับขึ้น-นับลง 4 หลักซึ่งนับค่า 0000-9999 ไว้ใน Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_4_vf นำไฟล์วงจรดีเบาเซอร์ชื่อ ch4vf_debounce ไฟล์วงจรนับขึ้น-นับลงชื่อ ch4vf_c10_ce_updn และไฟล์วงจรลดรหัสตัวแสดงผลบนเซกเมนต์ชื่อ ch4vf_BCD_to_7SEGMENT มาทำเป็น Component ทำการเขียนโค้ดดังรูปที่ L2.2 ซึ่งรวมโค้ดวงจรนับขึ้น 24 บิต โค้ดวงจรมาสเตอร์เพล็กเซอร์ (4 to 1 Multiplexer) 4 บิต และโค้ดวงจรลดรหัส 2 to 4 Decoder (One cold Decoder) ไว้แล้ว วงจรนี้จะเคลียร์ค่าเอาต์พุดเป็น 0000 (Hex) เมื่อ CLR = '0' และจะนับขึ้นเมื่อ DIR = '0' แต่ถ้า DIR = '1' จะเป็นการนับลง ในการทดลองนี้จะใช้ความถี่ 2.98 Hz จากเอาต์พุด F(22) ของวงจรนับขึ้น 24 บิตเพื่อเคลียร์เอาต์พุดของวงจรดีเบาเซอร์โดยอัตโนมัติทุกๆ 1/2.98 Hz = 0.34 วินาที และจะใช้ความถี่ 47.68 Hz และ 95.36 Hz จากเอาต์พุด F(18) และ F(17) ของวงจรนับขึ้น 24 บิตจ่ายให้วงจรสแกนเพื่อสแกนแต่ละหลักที่ความถี่ 47.68 > 30 Hz)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_4_vf is
7     Port ( CE,CLR,Din,DIR,OSC : in STD_LOGIC;
8             COM : out STD_LOGIC_VECTOR (3 downto 0);
9             Y : out STD_LOGIC_VECTOR (6 downto 0));
10 end ex4_4_vf;
11
12 architecture Behavioral of ex4_4_vf is
13     component ch4vf_DEBOUNCER
14         Port ( OSC,D_in,CLR : in STD_LOGIC;
15                 D_out : out STD_LOGIC);
16     end component;
17     component ch4vf_c10_ce_updn
18         Port ( C,CE,CLR,DIR : in STD_LOGIC;
19                 CEO : out STD_LOGIC;
20                 Q : out STD_LOGIC_VECTOR (3 downto 0));
21     end component;
22     component ch4vf_BCD_to_7SEGMENT
23         Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
24                 Y : out STD_LOGIC_VECTOR (6 downto 0));
25     end component;
26     signal C : STD_LOGIC;
27     signal F : STD_LOGIC_VECTOR (23 downto 0);
28     signal CE1,CE2,CE3 : STD_LOGIC;
29     signal X0,X1,X2,X3,A : STD_LOGIC_VECTOR (3 downto 0);
30 begin
31     -----15Bits Counter-----
32 process(OSC) --Output F(23)=1.49Hz,F(22)=2.98Hz,F(21)=5.69Hz
33     begin
34         if OSC'event and OSC='1' then F <= F + 1;
35         end if;
36     end process;
37     -----DEBOUNCER-----
38 DEBOUNCER : ch4vf_DEBOUNCER
39     port map(OSC=>OSC,D_in=>Din,CLR=>F(22),D_out=>C);

```

(ต่อ)

```

40      -----COUNTER : 4 Digits-----
41  COUNTER_DIGIT_0 : ch4vf_c10_ce_updn
42    port map(C=>C,CE=>CE,CLR=>CLR,DIR=>DIR,CEO=>CE1,Q=>X0);
43  COUNTER_DIGIT_1 : ch4vf_c10_ce_updn
44    port map(C=>C,CE=>CE1,CLR=>CLR,DIR=>DIR,CEO=>CE2,Q=>X1);
45  COUNTER_DIGIT_2 : ch4vf_c10_ce_updn
46    port map(C=>C,CE=>CE2,CLR=>CLR,DIR=>DIR,CEO=>CE3,Q=>X2);
47  COUNTER_DIGIT_3 : ch4vf_c10_ce_updn
48    port map(C=>C,CE=>CE3,CLR=>CLR,DIR=>DIR,CEO=>open,Q=>X3);
49      -----Scan circuit-----
50      -----4Bits MUX4to1-----
51      A <= X0      when F(18 downto 17)="00" else
52          X1      when F(18 downto 17)="01" else
53          X2      when F(18 downto 17)="10" else
54          X3;
55      -----1to2 Decoder(One cold Decoder)-----
56  COM <= "1110" when F(18 downto 17)="00" else
57      "1101" when F(18 downto 17)="01" else
58      "1011" when F(18 downto 17)="10" else
59      "0111";
60      -----7 Segment decoder-----
61  DECODER_7SEG : ch4vf_BCD_to_7SEGMENT
62    port map(A=>A,Y=>Y);
63 end Behavioral;

```

รูปที่ L2.2 โล็คดาวน์นับเลขฐานสิบที่เป็นวงจรนับขึ้น-ลง 4 หลัก

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25 MHz และปุ่มกด PB1, Dip SW1-Dip SW3 เป็นอินพุต และมีตัวแสดงผลเซเว่นเซกเมนต์และแคปติว์ร่วมของ DIGIT1-DIGIT4 เป็นเอาต์พุต กล่าวคือ

Din = PB1 = INPUT = p44	Y(0) = a = OUTPUT = p40	Y(5) = f = OUTPUT = p25
CLR = Dip SW1 = INPUT = p52	Y(1) = b = OUTPUT = p35	Y(6) = g = OUTPUT = p23
DIR = Dip SW2 = INPUT = p53	Y(2) = c = OUTPUT = p32	COM(3) = DIGIT4= OUTPUT= p41
CE = Dip SW3 = INPUT = p55	Y(3) = d = OUTPUT = p30	COM(2) = DIGIT3= OUTPUT= p36
OSC= OSC = INPUT = p127	Y(4) = e = OUTPUT = p27	COM(1) = DIGIT2= OUTPUT= p33
		COM(0) = DIGIT1= OUTPUT= p31

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CE	Input	p55	BANK4	LVCMS33	N/A	3.30					Unknown		
CLR	Input	p52	BANK5	LVCMS33	N/A	3.30					Unknown		
COM<0>	Output	p31	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
COM<1>	Output	p33	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
COM<2>	Output	p36	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
COM<3>	Output	p41	BANK5	LVCMS33	N/A	3.30		SLOW			Unknown		
Din	Input	p44	BANK5	LVCMS33	N/A	3.30					Unknown		
DIR	Input	p53	BANK5	LVCMS33	N/A	3.30					Unknown		
OSC	Input	p127	BANK0	LVCMS33	N/A	3.30					Unknown		
Y<0>	Output	p40	BANK5	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<1>	Output	p35	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<2>	Output	p32	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<3>	Output	p30	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<4>	Output	p27	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<5>	Output	p25	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<6>	Output	p23	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรม FPGA แล้วเลื่อน Dip SW1 และ Dip SW2 ไปที่ ON (CLR = '0' DIR = '0') และ Dip SW3 ไปที่ OFF CE = '1') ให้กดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกด PB1 ถ้าไม่ได้ให้สังเกตคุณที่ตัวแสดงผลเซเว่นเซกเมนต์ว่าติดสว่าง โดยให้ลองเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จนกว่าจะได้ Dip SW1 ไปที่ OFF (CLR= '1') เพื่อเคลียร์เอาต์พุตแล้ว OFF แล้ว ทดลองซ้ำ ในระหว่างทดลองให้การกด PB1 ถ้าไม่ได้เลื่อน Dip SW3 ไปที่ตำแหน่ง ON (CE= '0') เพื่อให้วงจรนับหยุดเดิน แล้ว OFF แล้วทดลองซ้ำ เลื่อน Dip SW2 ไปที่ตำแหน่ง OFF (DIR= '1') เพื่อนับลงแล้วทดลองซ้ำและบันทึกผลการทดลอง

การทดลองที่ 4.4.5 นาฬิกาดิจิตอลแบบตั้งเวลาหลักนาทีและชั่วโมงได้

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจหลักการทำางานและออกแบบวงจรนับรูปแบบต่างๆ
- 2) เพื่อสร้างวงจรนาฬิกาดิจิตอลโดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรนาฬิกาดิจิตอลด้วย CPLD

1.1) เขียนโค้ดวงจรนับ 60 และวงจนับ 24 เพื่อทำเป็นวงรассดคงผลเวลาวินาที นาที และชั่วโมงดังรูปที่ L1.1 โดยสร้างไฟล์ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vcx1_SEC_MIN_HR โดยที่ SET_MIN คือปุ่มตั้งเวลาเป็นนาทีและมี SET_HR คือปุ่มตั้งเวลาเป็นชั่วโมง ทำการบันทึกไฟล์และ Check syntax แล้วทำ Behavioral simulation โดยใช้ Source File ชื่อ ch4vcx1_SEC_MIN_HR_tb1 แล้วให้คุณจำลองการทำงานว่าเป็นไปตามทฤษฎีหรือไม่

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ch4vcx1_SEC_MIN_HR is
7     Port ( F1HZ,SET_MIN,SET_HR : in STD_LOGIC;
8             Q0,Q1,Q2,Q3 : out STD_LOGIC_VECTOR (3 downto 0));
9 end ch4vcx1_SEC_MIN_HR;
10
11 architecture Behavioral of ch4vcx1_SEC_MIN_HR is
12     signal Q0T,Q1T,Q2T,Q3T : STD_LOGIC_VECTOR (3 downto 0):="0000";
13     signal SEC : STD_LOGIC_VECTOR (5 downto 0):="000000";
14     signal MIN,HR : STD_LOGIC;
15 begin
16     -----SEC-----
17     SEC_DIGIT : process(F1HZ)
18     begin
19         if F1Hz'event and F1Hz='1' then
20             if SEC>=59 then SEC <= (others =>'0');-- SEC <= "000000";
21             else SEC <= SEC + 1;
22             end if;
23         end if;
24     end process SEC_DIGIT;
25     MIN <= '1' when SEC=59 else
26         '0';
27     -----MIN-----
28     MIN_DIGIT : process(F1Hz)
29     begin
30         if F1Hz'event and F1Hz='1' then
31             if (MIN='1' or SET_MIN='0') then
32                 if (Q1T>=5 and Q0T>=9) then Q1T <= "0000"; Q0T <= "0000";
33                 elsif (Q0T>=9) then
34                     Q0T <= "0000";
35                     if (Q1T>=9) then Q1T <= "0000";
36                     else Q1T <= Q1T + 1;
37                     end if;
38                 else Q0T <= Q0T + 1;

```

(ต่อ)

```

39         end if;
40     end if;
41   end if;
42 end process MIN_DIGIT;
43   HR <= '1' when (Q1T>=5 and Q0T>=9) else
44     '0';
45   Q1 <= Q1T; Q0 <= Q0T;
46   -----HR-----
47 HR_DIGIT : process(F1Hz)
48 begin
49   if F1Hz'event and F1Hz='1' then
50     if (HR='1' or SET_HR='0') then
51       if (Q3T>=2 and Q2T>=3) then Q3T <= "0000"; Q2T <= "0000";
52     elsif (Q2T>=9) then
53       Q2T <= "0000";
54       if (Q3T>=9) then Q3T <= "0000";
55       else Q3T <= Q3T + 1;
56     end if;
57     else Q2T <= Q2T + 1;
58   end if;
59   end if;
60 end if;
61 end process HR_DIGIT;
62   Q3 <= Q3T; Q2 <= Q2T;
63 end Behavioral;

```

รูปที่ L1.1 โค้ดของวงจรนับ 60 และวงจรนับ 24 เพื่อทำเป็นวงจรแสดงผลเวลาเป็นนาทีและชั่วโมง

1.2) สร้างไฟล์วงจรนาฬิกาดิจิตอลใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_4_5vcxl แล้วนำไฟล์ชื่อ ch4vcxl_SEC_MIN_HR และไฟล์ชื่อ ch4vcxl_BCD_to_7SEGMENT มาทำเป็น Component ทำการเรียกโดยคัดค้างรูปที่ L1.2 ซึ่งรวมโค้ดของวงจรนับเพื่อสร้างความถี่สำหรับวงจรสแกนและความถี่เป็นสัญญาณนาฬิกา (Clock) โค้ดของวงจรมัลติเพล็กซอร์ (4 to 1 Multiplexer) 4 บิต โค้ดของวงจรดอค รหัส 2 to 4 Decoder (One cold Decoder) และวงจรทำเครื่องหมาย ":" (Colon) ไว้แล้ว การทดลองนี้จะใช้ความถี่ 1 Hz จากเอกสารพุต F(14) ของวงจรนับขึ้น 15 บิตเป็นสัญญาณนาฬิกา (Clock) และใช้ความถี่ 64 Hz และ 128 Hz จากเอกสารพุต F(8) และ F(7) จ่ายให้วงจรสแกนเพื่อสแกนแต่ละหลักที่ความถี่ 64 Hz

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_4_5vcxl is
7   Port ( OSC,SET_HR,SET_MIN : in STD_LOGIC;
8         DP : out STD_LOGIC;
9         COM : out STD_LOGIC_VECTOR (3 downto 0);
10        Y : out STD_LOGIC_VECTOR (6 downto 0));
11 end ex4_4_5vcxl;
12
13 architecture Behavioral of ex4_4_5vcxl is
14   component ch4vcxl_SEC_MIN_HR
15     Port ( F1HZ,SET_MIN,SET_HR : in STD_LOGIC;
16            Q0,Q1,Q2,Q3 : out STD_LOGIC_VECTOR (3 downto 0));
17   end component;
18   component ch4vcxl_BCD_to_7SEGMENT
19     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
20            Y : out STD_LOGIC_VECTOR (6 downto 0));
21   end component;
22
23   signal F : STD_LOGIC_VECTOR (14 downto 0);
24   signal X0,X1,X2,X3,A : STD_LOGIC_VECTOR (3 downto 0);
25 begin
26   -----15Bits Counter-----
27 process(OSC) --Output F(14)=1Hz,F(8)=64Hz,F(7)=128Hz
28 begin
29   if OSC'event and OSC='1' then F <= F + 1;
30   end if;
31 end process;

```

(ต่อ)

```

32 -----SEC_MIN_HR-----
33 SEC_MIN_HR : ch4vcx1_SEC_MIN_HR
34     port map ( F1HZ=>F(14),
35                 SET_MIN=>SET_MIN,
36                 SET_HR=>SET_HR,
37                 Q0=>X0,
38                 Q1=>X1,
39                 Q2=>X2,
40                 Q3=>X3 );
41 -----Scan circuit-----
42 -----4Bits MUX4to1-----
43     A <= X0      when F(8 downto 7)="00" else
44             X1      when F(8 downto 7)="01" else
45             X2      when F(8 downto 7)="10" else
46             X3;
47 -----2to4 Decoder(One cold Decoder)-----
48     COM <= "1110" when F(8 downto 7)="00" else
49             "1101" when F(8 downto 7)="01" else
50             "1011" when F(8 downto 7)="10" else
51             "0111";
52 -----7 Segment decoder-----
53     DP <= F(14)  when F(8 downto 7)="01" else --Colon
54             F(14)  when F(8 downto 7)="10" else --Colon
55             '0';
56 DECODER_7SEG : ch4vcx1_BCD_to_7SEGMENT
57     port map(A=>A, Y=>Y);
58 end Behavioral;

```

รูปที่ L1.2 โค้ดงานนาฬิกาดิจิตอล

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB2 เป็นอินพุต และมีตัวแสดงผลเซเว่นเซกเมนต์ DIGIT1-DIGIT4 เป็นเอาต์พุต กล่าวคือ

SET_HR = PB1 = INPUT = p39	Y(0) = a = OUTPUT = p27	Y(6) = g = OUTPUT = p18
SET_MIN = PB2 = INPUT = p40	Y(1) = b = OUTPUT = p26	DP = dp = OUTPUT = p19
OSC = OSC = INPUT = p5	Y(2) = c = OUTPUT = p25	COM(3) = DIGIT2 = OUTPUT = p28
	Y(3) = d = OUTPUT = p24	COM(2) = DIGIT1 = OUTPUT = p29
	Y(4) = e = OUTPUT = p22	COM(1) = DIGIT1 = OUTPUT = p33
	Y(5) = f = OUTPUT = p20	COM(0) = DIGIT1 = OUTPUT = p34

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]						
I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
OSC	Input	p5	1	9		
SET_HR	Input	p39	2	9		
SET_MIN	Input	p40	2	11		
DP	Output	p19	3	14	SLOW	
COM<3>	Output	p28	4	11	SLOW	
COM<2>	Output	p29	4	14	SLOW	
COM<1>	Output	p33	4	15	SLOW	
COM<0>	Output	p34	4	17	SLOW	
Y<6>	Output	p18	3	11	SLOW	
Y<5>	Output	p20	3	15	SLOW	
Y<4>	Output	p22	3	17	SLOW	
Y<3>	Output	p24	3	16	SLOW	
Y<2>	Output	p25	4	2	SLOW	
Y<1>	Output	p26	4	5	SLOW	
Y<0>	Output	p27	4	8	SLOW	

รูปที่ L1.3 Assign Package Pins

หลังจากโปรแกรม CPLD แล้วให้กดปุ่ม PB2 ไปเรื่อยๆ ลับกับการกด PB2 ค้างไว้เพื่อตั้งเวลาลักษณะที่ แล้วให้สังเกตคุณผลที่ตัวแสดงผลเซเว่นเซกเมนต์ว่าหลักนาทีทำงานถูกหรือไม่ จากนั้นให้กดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกด PB1 ค้างไว้เพื่อตั้งเวลาลักษณะที่ ไม่ทำงาน แล้วให้สังเกตคุณผลที่ตัวแสดงผลว่าหลักนาทีทำงานถูกหรือไม่ แล้วทำการบันทึกผลการทดลอง

2 สร้างวงจรนาฬิกาดิจิตอลด้วย FPGA

2.1) เขียนโค้ดวงจรนับ 60 และวงจรนับ 24 เพื่อทำเป็นวงจรแสดงผลเวลาวินาที นาที และชั่วโมงดังรูปที่ L2.1 โดยสร้างไฟล์ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vf_SEC_MIN_HR โดยที่ SET_MIN คือปุ่มตั้งเวลา เป็นนาทีและมี SET_HR คือปุ่มตั้งเวลาเป็นชั่วโมง ทำการบันทึกไฟล์และ Check syntax

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ch4vf_SEC_MIN_HR is
7     Port ( F1HZ,SET_MIN,SET_HR : in STD_LOGIC;
8             Q0,Q1,Q2,Q3 : out STD_LOGIC_VECTOR (3 downto 0));
9 end ch4vf_SEC_MIN_HR;
10
11 architecture Behavioral of ch4vf_SEC_MIN_HR is
12     signal Q0T,Q1T,Q2T,Q3T : STD_LOGIC_VECTOR (3 downto 0):="0000";
13     signal SEC : STD_LOGIC_VECTOR (5 downto 0):="000000";
14     signal MIN,HR : STD_LOGIC;
15 begin
16     -----
17 SEC_DIGIT : process(F1HZ)
18     begin
19         if F1Hz'event and F1Hz='1' then
20             if SEC>=59 then SEC <= (others =>'0');-- SEC <= "000000";
21             else SEC <= SEC + 1;
22             end if;
23         end if;
24     end process SEC_DIGIT;
25     MIN <= '1' when SEC=59 else
26         '0';
27     -----
28 MIN_DIGIT : process(F1Hz)
29     begin
30         if F1Hz'event and F1Hz='1' then
31             if (MIN='1' or SET_MIN='0') then
32                 if (Q1T>=5 and Q0T>=9) then Q1T <= "0000"; Q0T <= "0000";
33                 elsif (Q0T>=9) then
34                     Q0T <= "0000";
35                     if (Q1T>=9) then Q1T <= "0000";
36                     else Q1T <= Q1T + 1;
37                     end if;
38                 else Q0T <= Q0T + 1;
39                 end if;
40             end if;
41         end if;
42     end process MIN_DIGIT;
43     HR <= '1' when (Q1T>=5 and Q0T>=9) else
44         '0';
45     Q1 <= Q1T; Q0 <= Q0T;
46     -----
47 HR_DIGIT : process(F1Hz)
48     begin
49         if F1Hz'event and F1Hz='1' then
50             if (HR='1' or SET_HR='0') then
51                 if (Q3T>=2 and Q2T>=3) then Q3T <= "0000"; Q2T <= "0000";
52                 elsif (Q2T>=9) then
53                     Q2T <= "0000";
54                     if (Q3T>=9) then Q3T <= "0000";
55                     else Q3T <= Q3T + 1;
56                     end if;
57                 else Q2T <= Q2T + 1;
58                 end if;
59             end if;
60         end if;
61     end process HR_DIGIT;
62     Q3 <= Q3T; Q2 <= Q2T;
63 end Behavioral;
```

รูปที่ L2.1 โค้ดของวงจรนับ 60 และวงจรนับ 24 เพื่อทำเป็นวงจรแสดงผลเวลาวินาที นาที และชั่วโมง

2.2) สร้างไฟล์วงจรนาฬิกาดิจิตอลใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_4_5vf และนำไฟล์ชื่อ ch4vf_SEC_MIN_HR และไฟล์ชื่อ ch4vf_BCD_to_7SEGMENT มาทำเป็น Component ทำการเขียนโค้ดดังรูปที่ L2.2 ซึ่งรวมโค้ดของวงจรนับ (วงจรหารความถี่) เพื่อสร้างความถี่ 50 Hz และ 100 Hz สำหรับวงจรสแกนและความถี่ 1 Hz เป็นสัญญาณนาฬิกา (Clock) โค้ดของวงจรมัคกิเพล็กเซอร์ (4 to 1 Multiplexer) 4 บิต โค้ดของวงจรต่อครั้ง 2 to 4 Decoder (One cold Decoder) และวงจรทำเครื่องหมาย ":" (Colon) ไว้แล้ว การทดลองนี้จะสร้างความถี่ 50 Hz และ 100 Hz โดยสร้างวงจรหารความถี่ 125,000 เพื่อหารความถี่จาก 25 MHz เป็นความถี่ 200 Hz จากนั้นนำความถี่นี้ไปป้อนให้กับวงจรนับ 4 ก็จะได้เอาต์พุตเป็นความถี่ 50 Hz และ 100 Hz ตามที่ต้องการ จากรันน้ำความถี่ 50 Hz ไปป้อนให้กับวงจรหารความถี่ 50 ก็จะได้ความถี่ 1 Hz เป็นสัญญาณนาฬิกา (Clock)

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity ex4_4_5vf is
7      Port ( OSC,SET_HR,SET_MIN : in STD_LOGIC;
8             DP : out STD_LOGIC;
9             COM : out STD_LOGIC_VECTOR (3 downto 0);
10            Y : out STD_LOGIC_VECTOR (6 downto 0));
11 end ex4_4_5vf;
12
13 architecture Behavioral of ex4_4_5vf is
14     component ch4vf_SEC_MIN_HR
15         Port ( F1HZ,SET_MIN,SET_HR : in STD_LOGIC;
16                 Q0,Q1,Q2,Q3 : out STD_LOGIC_VECTOR (3 downto 0));
17     end component;
18     component ch4vf_BCD_to_7SEGMENT
19         Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
20                 Y : out STD_LOGIC_VECTOR (6 downto 0));
21     end component;
22
23     signal F1 : STD_LOGIC_VECTOR (16 downto 0);
24     signal F2 : STD_LOGIC_VECTOR (1 downto 0);
25     signal F3 : STD_LOGIC_VECTOR (5 downto 0);
26     signal F1Hz : STD_LOGIC;
27     signal X0,X1,X2,X3,A : STD_LOGIC_VECTOR (3 downto 0);
28 begin
29     -----Divider : N=25,000,000-----
30     -----Divider : N=125,000-----
31 process(OSC)
32     begin
33         if (OSC'event and OSC='1') then
34             if (F1>=124999) then F1 <= (others=>'0');
35             else F1 <= F1 + 1;
36             end if;
37         end if;
38     end process;
39     -----Divider : N=4-----
40 process(F1(16))--F2(1)=50Hz and F2(0)=100Hz for scan circuit
41     begin
42         if (F1(16)'event and F1(16)='1') then F2 <= F2 + 1;
43     end if;
44     end process;
45     -----Divider : N=50-----
46 process(F2(1))    --F1Hz = 25MHz/(125,000x4x50)--
47     begin
48         if (F2(1)'event and F2(1)='1') then
49             if (F3>=49) then F3 <= (others=>'0');
50             else F3 <= F3 + 1;
51             end if;
52         end if;
53     end process;
54     F1Hz <= '0' when (F3<=24) else
55             '1';                                --1Hz with 50% duty
                                                --for Colon

```

(ต่อ)

```

56 -----SEC_MIN_HR-----
57 SEC_MIN_HR : ch4vf_SEC_MIN_HR
58     port map ( F1HZ=>F1Hz,
59                 SET_MIN=>SET_MIN,
60                 SET_HR=>SET_HR,
61                 Q0=>X0,
62                 Q1=>X1,
63                 Q2=>X2,
64                 Q3=>X3 );
65 -----Scan circuit-----
66 -----4Bits MUX4to1-----
67 A <= X0      when F2="00" else
68     X1      when F2="01" else
69     X2      when F2="10" else
70     X3;
71 -----2to4 Decoder(One cold Decoder)-----
72 COM <= "1110" when F2="00" else
73     "1101" when F2="01" else
74     "1011" when F2="10" else
75     "0111";
76 -----7 Segment decoder-----
77 DP <= F1Hz  when F2="01" else --Colon
78     F1Hz  when F2="10" else --Colon
79     '0';
80 DECODER_7SEG : ch4vf_BCD_to_7SEGMENT
81     port map(A=>A, Y=>Y);
82 end Behavioral;

```

รูปที่ L2.2 โค้ดงานพิการดิจิตอล

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25 MHz และปุ่มกด PB1-PB2 เป็นอินพุต และมีตัวแสดงผล เขียนเซกเมนต์ DIGIT1-DIGIT4 เป็นอาต์ฟุต กล่าวคือ

SET_HR = PB1 = INPUT = p44	Y(0) = a = OUTPUT = p40	Y(6) = g = OUTPUT = p23
SET_MIN = PB2 = INPUT = p46	Y(1) = b = OUTPUT = p35	Y(7) = dp = OUTPUT = p20
OSC = INPUT = p127	Y(2) = c = OUTPUT = p32	COM(3) = DIGIT4= OUTPUT= p41
	Y(3) = d = OUTPUT = p30	COM(2) = DIGIT3= OUTPUT= p36
	Y(4) = e = OUTPUT = p27	COM(1) = DIGIT2= OUTPUT= p33
	Y(5) = f = OUTPUT = p25	COM(0) = DIGIT1= OUTPUT= p31

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
COM<0>	Output	p31	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
COM<1>	Output	p33	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
COM<2>	Output	p36	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
COM<3>	Output	p41	BANK5	LVCMS33	N/A	3.30			SLOW		Unknown		
DP	Output	p20	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
OSC	Input	p127	BANK0	LVCMS33	N/A	3.30					Unknown		
SET_HR	Input	p44	BANK5	LVCMS33	N/A	3.30					Unknown		
SET_MIN	Input	p46	BANK5	LVCMS33	N/A	3.30					Unknown		
Y<0>	Output	p40	BANK5	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<1>	Output	p35	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<2>	Output	p32	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<3>	Output	p30	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<4>	Output	p27	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<5>	Output	p25	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		
Y<6>	Output	p23	BANK6	LVCMS33	N/A	3.30			SLOW		Unknown		

รูปที่ L2.3 Assign Package Pins

หลังจากโปรแกรม FPGA แล้วให้กดปุ่ม PB2 ไปเรื่อยๆ แล้วบันทึกการกด PB2 ค้างไว้เพื่อตั้งเวลาหลักนาที แล้วให้สังเกตคุณผลที่ตัวแสดงผลเว้นเซกเมนต์ว่าหลักนาทีทำงานถูกหรือไม่ จากนั้นให้กดปุ่ม PB1 ไปเรื่อยๆ แล้วบันทึกการกด PB1 ค้างไว้เพื่อตั้งเวลาหลักชั่วโมงแล้วให้สังเกตคุณผลที่ตัวแสดงผลว่าหลักชั่วโมงทำงานถูกหรือไม่ แล้วทำการบันทึกผลการทดลอง

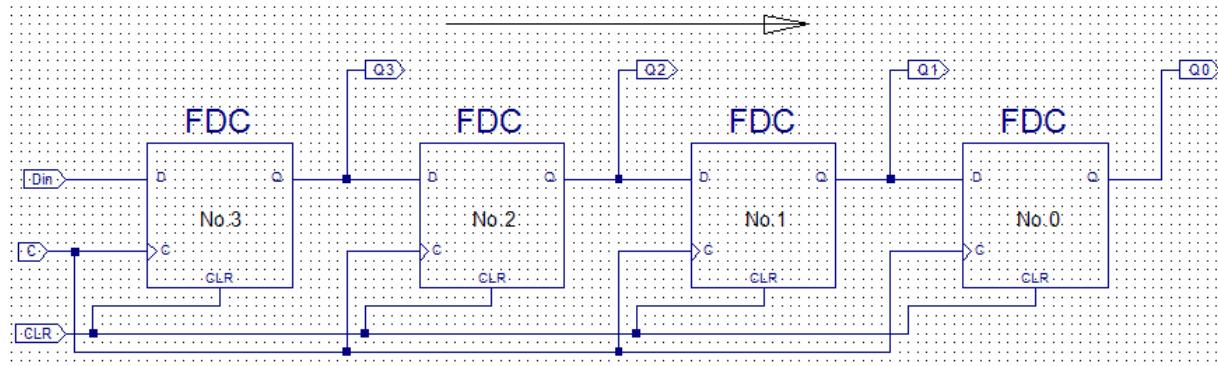
4.5 ชิฟต์รีจิสเตอร์

ชิฟต์รีจิสเตอร์ (Shift register) สามารถแบ่งตามวิธีการรับข้อมูลเข้าและส่งข้อมูลออกได้เป็นดังนี้

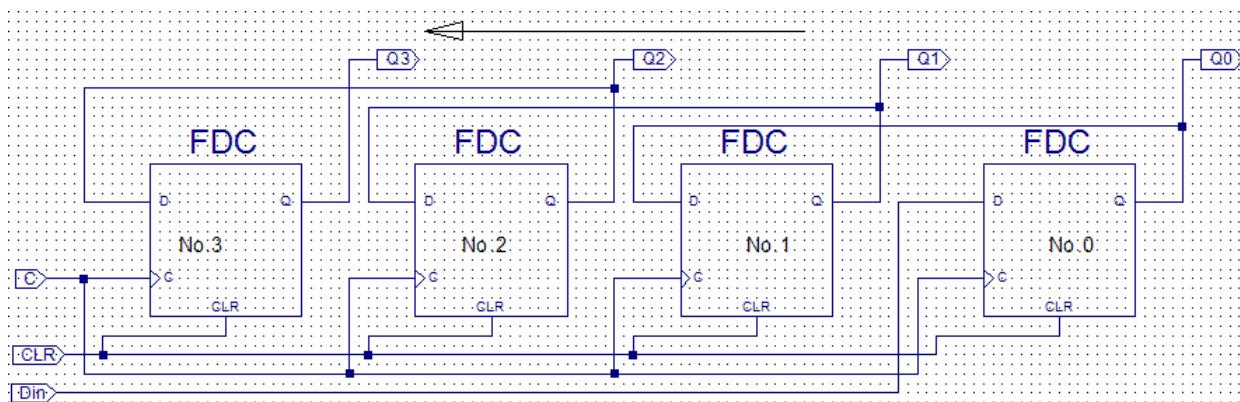
- เข้าแบบอนุกรม–ออกแบบอนุกรม (Serial in–Serial out) และ/หรือขนาน (Serial in–Parallel out)
- เข้าแบบขนาน–ออกแบบอนุกรม (Parallel in–Serial out) และ/หรือขนาน (Parallel in–Parallel out)

ชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรม–ออกแบบอนุกรมและ/หรือขนานแบบเลื่อนข้อมูลไปทางขวา (Shift right) แสดงดังรูปที่ 4.43 โดยจะส่งข้อมูลเข้าแบบอนุกรมที่อินพุต Din และจะเลื่อนข้อมูลไปทางขวา (เลื่อนบิตจาก MSB ไป LSB) ตามจังหวะการทริกด้วยขอบขาขึ้นของสัญญาณนาฬิกา (C = Clock) ส่วนชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรม–ออกแบบอนุกรมและ/หรือขนานแบบเลื่อนข้อมูลไปทางซ้าย (Shift left) แสดงดังรูปที่ 4.44 ซึ่งเป็นการเลื่อนข้อมูลไปทางซ้าย (เลื่อนบิตจาก LSB ไป MSB) ตามจังหวะการทริกด้วยขอบขาขึ้นของสัญญาณนาฬิกา (C = Clock)

การเขียนโค้ด VHDL ของชิฟต์รีจิสเตอร์ข้อมูลไปทางขวาหรือซ้ายในรูปแบบต่างๆ นั้นมีรายละเอียดในบทที่ 2 ตัวอย่างที่ 2.29 และตัวอย่างที่ 2.33



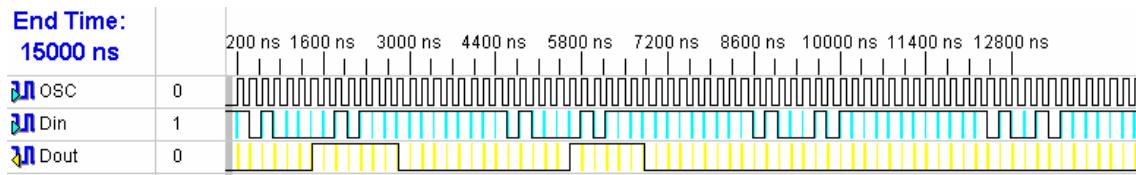
รูปที่ 4.43 ชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรม–ออกแบบอนุกรมและแบบขนานแบบ Shift right



รูปที่ 4.44 ชิฟต์รีจิสเตอร์ชนิดข้อมูลเข้าแบบอนุกรม–ออกแบบอนุกรมและแบบขนานแบบ Shift left

ในการกดคีย์บอร์ดหรือปุ่มกดจะเกิดเบ้า (Bouncing) ในช่วงเวลาที่หน้าตั้งผัสแตกหรือจากกันยังไม่สนิทจะทำให้เกิดพลั๊ส (Pulse) ออกมากหลายลูกทึ้งๆ ที่กดคีย์บอร์ดหรือปุ่มกดเพียงครั้งเดียว ปัญหานี้สามารถแก้ไขได้โดยใช้วงจร โโนโนสเตเบิล (Mono stable) หรือวงจรดีเบาเชอร์ (Debouncer) ดังที่อธิบายในการทดลองที่ 4.2.3 ซึ่งเป็นวงจรดีเบาเชอร์อย่างง่ายและวงจรนี้มีประสิทธิภาพดีเมื่อทำงานโดยอัตโนมัติ ซึ่งในทางปฏิบัตินั้นเรามักจะมีความจำเป็นต้องใช้วงจรดีเบาเชอร์ที่มีประสิทธิภาพสูง

วงจรดีเบาเซอร์ที่ให้ความกว้างเอาต์พุตพัลส์ขึ้นกับเวลาที่กดคីบอร์ดแบบ Active low โดยมีโคลาแกร์เวลาแสดงดังรูปที่ 4.45 ซึ่งเป็นการนำชิตรีจิตเตอร์ไปประยุกต์ใช้งาน โดยวงจรนี้จะทำงานเดียบแบบไอซี MC14490 จึงเหมาะสมสำหรับทำวงจรของคីบอร์ดแบบเบาซ์เลส (Bounceless)



รูปที่ 4.45 โคลาแกร์เวลาของวงจรดีเบาเซอร์ที่ให้ความกว้างเอาต์พุตพัลส์ขึ้นกับเวลาที่กดคីบอร์ด

จากรูปที่ 4.45 จะพบว่าเมื่อกดปุ่ม Din จะให้ลอจิก ‘0’ ต่อเนื่องกัน 4 คาบของสัญญาณนาฬิกาหรือ 4 Clock (จำนวน 4 ขอบขาขึ้นของ Clock) แล้ว Dout = ‘1’ และ Dout จะคงสถานะลอจิก ‘1’ ไปตลอดจนกว่าจะปล่อยปุ่มกด Din ให้กลับเป็นลอจิก ‘1’ ต่อเนื่องกัน 4 คาบของสัญญาณนาฬิกาหรือ 4 Clock แล้ว Dout = ‘0’ เราสามารถเขียนโค้ดวงจรดีเบาเซอร์ได้ดังรูปที่ 4.46

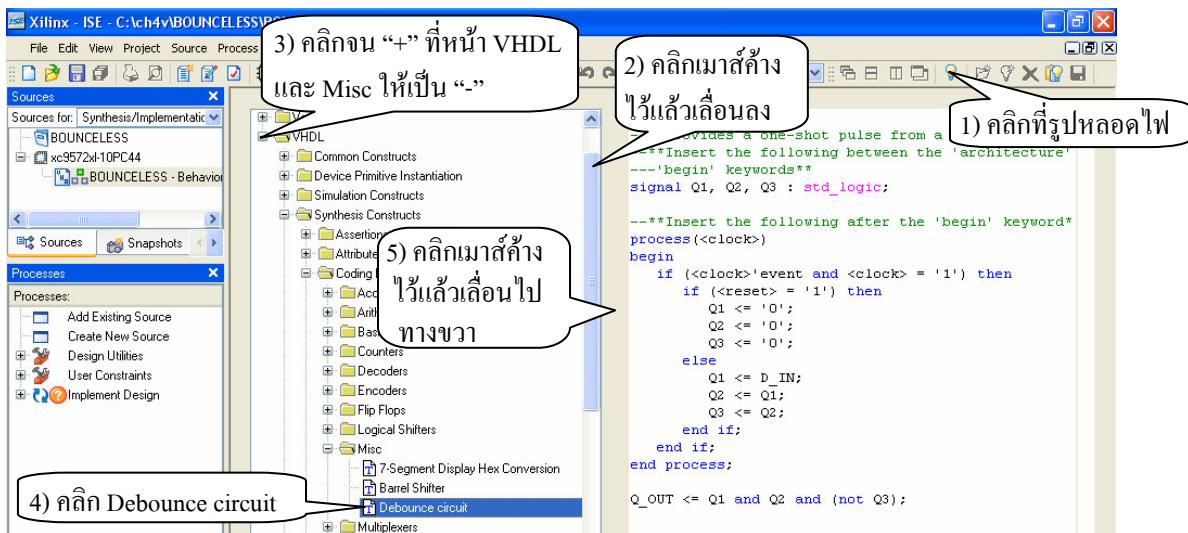
```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity BOUNCELESS is
6     Port ( C,Din : in STD_LOGIC;
7             Dout : out STD_LOGIC);
8 end BOUNCELESS;
9
10 architecture Behavioral of BOUNCELESS is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0):="0000";
12 begin
13     -----Bounceless keyboard-----
14 process(C)
15 begin
16     if (C'event and C='1') then
17         if (not Din)= QT(3) then
18             QT <= QT(3) & QT(3) & QT(3) & QT(3);--Load
19         else QT <= QT(2 downto 0) & (not Din); --Shift left
20         end if;
21     end if;
22 end process;
23 Dout <= QT(3);
24 end Behavioral;
```

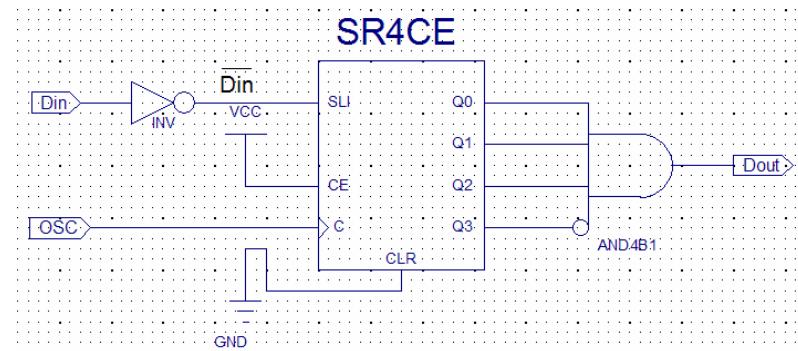
รูปที่ 4.46 โค้ดของวงจรดีเบาเซอร์ที่ให้ความกว้างเอาต์พุตพัลส์ขึ้นกับเวลาที่กดคីบอร์ด

จากรูปที่ 4.45 นี้หากหน้าสัมผัสแตะกันสนิทต่อเนื่องไม่น้อยกว่า 4 Clock (เวลาสูงสุดคือ 4 คาบของสัญญาณนาฬิกา) จึงจะสามารถส่งฟลัสร์ Dout ออกໄປได้ ซึ่งปกติการเกิดเบาซ์จะกินเวลาน้อยกว่า 20 มิลลิวินาที ดังนั้นความถี่ของอสซิลเลเตอร์ จึงไม่ควรเกิน $1/(20\text{มิลลิวินาที}/4) = 200 \text{ Hz}$

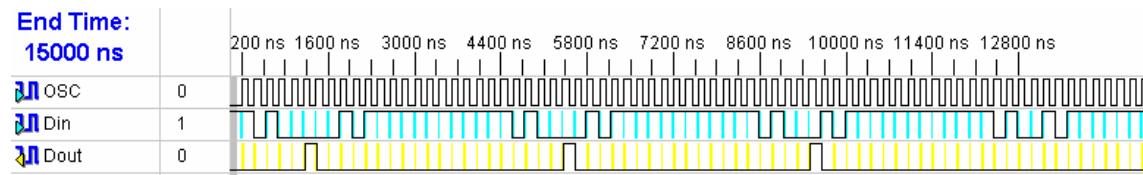
วงจรดีเบาเซอร์สำหรับปุ่มกดอิกลักษณะหนึ่งเป็นวงจรโมโนสเตเบิล (Mono stable) หรือ One-shot ซึ่งโค้ดวงจนี้เขียนไว้ใน Coding Examples ด้วยภาษา VHDL ดังรูปที่ 4.47 วงจนี้คิมไชรีจิตเตอร์หรือพลิปฟล็อก 3 ตัวแต่ผู้เขียนได้ดัดแปลงโดยเพิ่มเป็น 4 ตัวและใช้สัญญาณนาฬิกา (Clock) เพื่อให้มีประสิทธิภาพใกล้เคียงกับวงจรที่เขียนด้วยโค้ดในรูปที่ 4.46 โดยมีดังวงจรแสดงดังรูปที่ 4.48 และมีโคลาแกร์เวลาดังรูปที่ 4.49 ซึ่งวงจนี้จะเลื่อนค่าเอาต์พุตตามจังหวะสัญญาณนาฬิกา (Clock = OSC) โดยเอาต์พุต Dout = ‘1’ เมื่อหน้าสัมผัสแตะกันสนิทต่อเนื่องไม่น้อยกว่า 3 คาบและให้ความกว้างพัลส์คงที่ดังรูปที่ 4.49 ดังนั้นความถี่ Clock จึงไม่ควรเกิน $1/(20\text{มิลลิวินาที}/3) = 150 \text{ Hz}$ โค้ดวงจรโมโนสเตเบิลแสดงดังรูปที่ 4.50



รูปที่ 4.47 แสดงการเข้าไปที่โค้ดของ Debounce circuit ที่อยู่ใน Coding Examples ในหน้าต่าง Xilinx-ISE



รูปที่ 4.48 ผังวงจรโนนิสเตเบิลหรือวงจรดีเบาเซอร์ที่ให้ความกว้างพัลส์คงที่ซึ่งใช้ชิฟต์รีจิสเตอเร็บแบบ Shift left (SR4CE)



รูปที่ 4.49 ไดอะแกรมเวลาของวงจรโนนิสเตเบิลหรือวงจรดีเบาเซอร์ที่ให้ความกว้างพัลส์คงที่

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity MONO_STABLE is
6     Port ( C,Din : in STD_LOGIC;
7             Dout : out STD_LOGIC);
8 end MONO_STABLE;
9
10 architecture Behavioral of MONO_STABLE is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0):="0000";
12 begin
13     -----4Bits Shift register : Shift left-----
14     process(C)
15         begin
16             if (C'event and C='1') then
17                 QT <= (QT(2 downto 0)) & (not Din);
18             end if;
19         end process;
20     -----Dout-----
21     Dout <= QT(0) and QT(1) and QT(2) and (not QT(3));
22 end Behavioral;

```

รูปที่ 4.50 โค้ดของวงจรโนนิสเตเบิลหรือวงจรดีเบาเซอร์ที่ให้ความกว้างพัลส์คงที่

การทดลองที่ 4.5.1 การออกแบบวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวา

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการออกแบบวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวา (Shift right)
- 2) เพื่อสร้างวงจรชิฟต์รีจิสเตอร์โดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวาด้วย CPLD

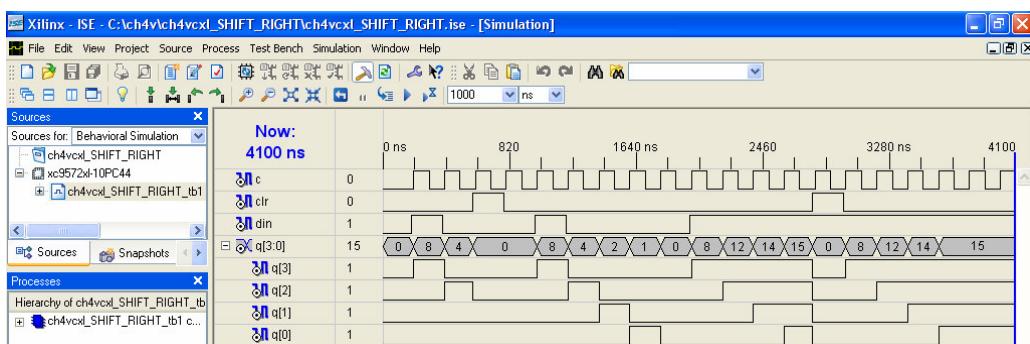
1.1) เขียนโค้ดชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift right ดังรูปที่ L1.1 โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ch4vcx1_SHIFT_RIGHT บันทึกไฟล์แล้ว Check Syntax หากนั่นทำ Behavioral simulation โดยใช้ไฟล์ชื่อ ch4vcx1_SHIFT_RIGHT_tb1 และพิจารณาผล Behavioral simulation ในรูปที่ L1.2 ว่าเป็นไปตามทฤษฎีหรือไม่

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ch4vcx1_SHIFT_RIGHT is
6     Port ( C,CLR,Din : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end ch4vcx1_SHIFT_RIGHT;
9
10 architecture Behavioral of ch4vcx1_SHIFT_RIGHT is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0):="0000";
12 begin
13 process(C,CLR)
14 begin
15     if CLR='1' then QT <= "0000";
16     elsif (C'event and C='1') then QT <= Din & QT(3 downto 1);
17     end if;
18 end process;
19     Q <= QT;
20 end Behavioral;

```

รูปที่ L1.1 โค้ดวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift right ที่มีการประกาศใช้ signal



รูปที่ L1.2 ผล Behavioral simulation ของวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift right

1.2) เขียนโค้ดชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift right ที่รวมวงจรดีเบาเซอร์ไว้แล้วไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_5_1vcx1 นำไฟล์วงจรดีเบาเซอร์ชื่อ ch4vcx1_DEBOUNCE และชิฟต์รีจิสเตอร์ชื่อ ch4vcx1_SHIFT_RIGHT มาทำเป็น Component และเขียนโค้ดดังรูปที่ L1.3 ซึ่งรวมโภคของวงจรนับขั้น 15 บิตเพื่อสร้างความถี่ความถี่ 2 Hz สำหรับเกลียร์วงจรดีเบาเซอร์โดยอัตโนมัติทุกๆ 1/2Hz = 0.5 วินาที

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_5_1vcx1 is
7     Port ( C,CLR,Din,OSC : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex4_5_1vcx1;
10
11 architecture Behavioral of ex4_5_1vcx1 is
12     component ch4vcx1_DEBOUNCER
13         Port ( OSC,D_in,CLR : in STD_LOGIC;
14                 D_out : out STD_LOGIC);
15     end component;
16     component ch4vcx1_SHIFT_RIGHT
17         Port ( C,CLR,Din : in STD_LOGIC;
18                 Q : out STD_LOGIC_VECTOR (3 downto 0));
19     end component;
20     signal CT : STD_LOGIC;
21     signal F : STD_LOGIC_VECTOR (14 downto 0);
22 begin
23     -----15Bits Counter-----
24 process(OSC) --Output F(14)=1Hz,F(13)=2Hz,F(12)=4Hz,F(11)=8Hz
25     begin
26         if OSC'event and OSC='1' then F <= F + 1;
27         end if;
28     end process;
29     -----DEBOUNCER-----
30 DEBOUNCER : ch4vcx1_DEBOUNCER
31     port map(OSC=>OSC,D_in=>C,CLR=>F(13),D_out=>CT);
32     -----SHIFT RESISTER-----
33 SHIFT_RIGHT : ch4vcx1_SHIFT_RIGHT
34     port map(C=>CT,CLR=>CLR,Din=>Din,Q=>Q);
35 end Behavioral;

```

รูปที่ L1.3 โค้ดของวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift right ที่รวมวงจรดีเบาเซอร์ไว้แล้ว

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz ปุ่มกด PB1, PB3(Slide SW1) และ PB4(Slide SW2) เป็นอินพุตและ LED1-LED4 เป็นเอาต์พุต ก่อว่าคือ

$$\begin{array}{lll}
 C = PB1 = \text{INPUT} = p39 & Q(3) = \text{LED1} = \text{OUTPUT} = p38 & Q(1) = \text{LED3} = \text{OUTPUT} = p36 \\
 \text{CLR} = PB3(\text{Slide SW1}) = \text{INPUT} = p42 & Q(2) = \text{LED2} = \text{OUTPUT} = p37 & Q(0) = \text{LED4} = \text{OUTPUT} = p35 \\
 \text{Din} = PB4(\text{Slide SW1}) = \text{INPUT} = p43 & \text{OSC} = \text{OSC} = \text{INPUT} = p5
 \end{array}$$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Logic	Function Block	Macrocell	Slew	Globals
C	Input	p39	2	9		
CLR	Input	p42	2	14		
Din	Input	p43	2	15		
OSC	Input	p5	1	9		
Q<3>	Output	p38	2	8	SLOW	
Q<2>	Output	p37	2	6	SLOW	
Q<1>	Output	p36	2	5	SLOW	
Q<0>	Output	p35	2	2	SLOW	

รูปที่ L1.4 Assign Package Pins

หลังจากโปรแกรม CPLD แล้ว คุณต้องต่อ LED1-LED4 ในตอนเริ่มต้นว่า LED ทุกดวงดับจริงหรือไม่ จากนั้นเลื่อน Slide SW1 ไปที่ตำแหน่ง ON (CLR = '0') และเลื่อน Slide SW2 ไปที่ตำแหน่ง OFF (Din = '1') ให้กดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกด PB1 ถ้าไม่แล้วให้คุณต่อ LED1-LED4 ว่าติดสว่างโดยให้กดอีกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นเลื่อน Slide SW2 ไปที่ตำแหน่ง ON (Din = '0') ให้กดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกด PB1 ถ้าไม่แล้วให้คุณต่อ LED1-LED4 จากนั้นเลื่อน

Slide SW2 ไปที่ตำแหน่ง OFF (Din = '1') อีกครั้ง แล้วให้กดปุ่ม PB1 ไปเรื่อยๆ ลับกันการกด PB1 ค้างไว้ แล้วให้คูลท์ LED1-LED4 แล้วเดือน Slide SW1 ไปที่ตำแหน่ง OFF (CLR= '1') เพื่อเคลียร์เอาต์พุตแล้ว ON แล้วทดลองซ้ำ

2 สร้างวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวาด้วย FPGA

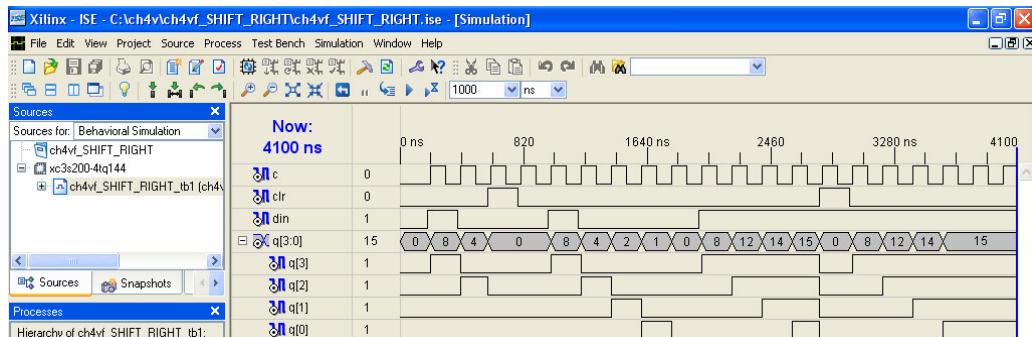
2.1) เขียนโค้ดวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift right ดังรูปที่ L2.1 โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ch4vf_SHIFT_RIGHT จากนั้นให้ทำ Behavioral simulation โดยใช้ไฟล์ชื่อ ch4vf_SHIFT_RIGHT_tb1 แล้วพิจารณาผล Behavioral simulation ในรูปที่ L2.2 ว่าเป็นไปตามทฤษฎีหรือไม่

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ch4vf_SHIFT_RIGHT is
6     Port ( C,CLR,Din : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end ch4vf_SHIFT_RIGHT;
9
10 architecture Behavioral of ch4vf_SHIFT_RIGHT is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0):="0000";
12 begin
13 process(C,CLR)
14 begin
15     if CLR='1' then QT <= "0000";
16     elsif (C'event and C='1') then QT <= Din & QT(3 downto 1);
17     end if;
18 end process;
19 Q <= QT;
20 end Behavioral;

```

รูปที่ L2.1 โค้ดวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift right



รูปที่ L2.2 ผล Behavioral simulation ของวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift right

2.2) เขียนโค้ดชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift right ที่รวมวงจรดีเบาเซอร์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_5_1vf นำไฟล์ชื่อ ch4vf_DEBOUNCE และ ch4vf_SHIFT_RIGHT มาทำเป็น Component เขียนโค้ดดังรูปที่ L2.3 วงจรนับ 24 บิตใช้สร้างความถี่ 2.98 Hz สำหรับเคลียร์วงจรดีเบาเซอร์โดยอัตโนมัติกๆ $1/2.98 \text{ Hz} = 0.34 \text{ วินาที}$

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25 MHz ปุ่มกด PB1, Dip SW1 และ Dip SW2 เป็นอินพุต และ LED L0-L3 เป็นเอาต์พุต กล่าวคือ

$$C = PB1 = INPUT = p44 \quad Q(3) = L3 = OUTPUT = p76 \quad Q(1) = L1 = OUTPUT = p77$$

$$CLR = Dip SW1 = INPUT = p52 \quad Q(2) = L2 = OUTPUT = p69 \quad Q(0) = L0 = OUTPUT = p70$$

$$Din = Dip SW2 = INPUT = p53 \quad OSC = OSC = INPUT = p127$$

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_5_1vf is
7     Port ( C,CLR,Din,OSC : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex4_5_1vf;
10
11 architecture Behavioral of ex4_5_1vf is
12     component ch4vf_DEBOUNCER
13         Port ( OSC,D_in,CLR : in STD_LOGIC;
14                 D_out : out STD_LOGIC);
15     end component;
16     component ch4vf_SHIFT_RIGHT
17         Port ( C,CLR,Din : in STD_LOGIC;
18                 Q : out STD_LOGIC_VECTOR (3 downto 0));
19     end component;
20     signal CT : STD_LOGIC;
21     signal F : STD_LOGIC_VECTOR (23 downto 0);
22 begin
23     -----24Bits Counter-----
24 process(OSC) --Output F(23)=1.49Hz,F(22)=2.98Hz
25     begin
26         if OSC'event and OSC='1' then F <= F + 1;
27     end if;
28 end process;
29     -----DEBOUNCER-----
30 DEBOUNCER : ch4vf_DEBOUNCER
31     port map(OSC=>OSC,D_in=>C,CLR=>F(22),D_out=>CT);
32     -----SHIFT RESISTER-----
33 SHIFT_RIGHT : ch4vf_SHIFT_RIGHT
34     port map(C=>CT,CLR=>CLR,Din=>Din,Q=>Q);
35 end Behavioral;

```

รูปที่ L2.3 โค้ดของวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift right ที่รวมวงจรดีเบาเซอร์ไว้แล้ว

พิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
C	Input	p44	BANK5	LVCMS33	N/A	3.30					Unknown		
CLR	Input	p52	BANK5	LVCMS33	N/A	3.30					Unknown		
Din	Input	p53	BANK5	LVCMS33	N/A	3.30					Unknown		
OSC	Input	p127	BANK0	LVCMS33	N/A	3.30					Unknown		
Q<0>	Output	p70	BANK4	LVCMS33	N/A	3.30			SLOW		Unknown		
Q<1>	Output	p77	BANK3	LVCMS33	N/A	3.30			SLOW		Unknown		
Q<2>	Output	p69	BANK4	LVCMS33	N/A	3.30			SLOW		Unknown		
Q<3>	Output	p76	BANK3	LVCMS33	N/A	3.30			SLOW		Unknown		

รูปที่ L2.4 Assign Package Pins

หลังจากโปรแกรม FPGA แล้ว คุณต้องต่อ LED L0-L3 ในตอนเริ่มต้นว่า LED ทุกดวงดับจริงหรือไม่ จากนั้นเลื่อน Dip SW1 ไปที่ตำแหน่ง ON (CLR = '0') และเลื่อน Dip SW2 ไปที่ตำแหน่ง OFF (Din = '1') ให้กดปุ่ม PB1 ไปเรื่อยๆ слับกับการกด PB1 ค้างไว้ แล้วให้คุณต่อ LED L0-L3 ว่าติดสว่างโดยให้ล้อจิกເອัดพูดเป็นไปตามทฤษฎีหรือไม่ จากนั้นเลื่อน Dip SW2 ไปที่ตำแหน่ง ON (Din = '0') ให้กดปุ่ม PB1 ไปเรื่อยๆ слับกับการกด PB1 ค้างไว้แล้วให้คุณต่อ LED L0-L3 จากนั้นเลื่อน Dip SW2 ไปที่ตำแหน่ง OFF (Din = '1') อีกครั้ง แล้วให้กดปุ่ม PB1 ไปเรื่อยๆ слับกับการกด PB1 ค้างไว้แล้วให้คุณต่อ LED L0-L3 แล้วเลื่อน Dip SW1 ไปที่ตำแหน่ง OFF (CLR = '1') เพื่อเคลียร์เอาต์พูดแล้ว ON แล้วทดสอบซ้ำ

การทดลองที่ 4.5.2 การออกแบบวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางซ้าย

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการออกแบบวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางซ้าย (Shift left)
- 2) เพื่อสร้างวงจรชิฟต์รีจิสเตอร์โดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวาด้วย CPLD

เขียนโค้ดชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift left ดังรูปที่ L1.1 โดยสร้างไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vcx1_SHIFT_LEFT

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.all;
4
5 entity ch4vcx1_SHIFT_LEFT is
6     Port ( C,CLR,Din : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end ch4vcx1_SHIFT_LEFT;
9
10 architecture Behavioral of ch4vcx1_SHIFT_LEFT is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0):="0000";
12 begin
13 process(C,CLR)
14 begin
15     if CLR='1' then QT <= "0000";
16     elsif (C'event and C='1') then QT <= QT(2 downto 0) & Din;
17     end if;
18 end process;
19     Q <= QT;
20 end Behavioral;

```

รูปที่ L1.1 โค้ดวงจรชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift left ที่มีการประกาศใช้ signal

จากนั้นทำการทดลอง CPLD ทำงานเดียวกับการทดลองที่ 4.5.1 ทุกประการ โดยเขียนโค้ดชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift left ที่รวมวงจรดีเบาเซอร์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_5_2vcx1

2 สร้างวงจรชิฟต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางซ้ายด้วย FPGA

นำโค้ดชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift left ในรูปที่ L1.1 มาแก้ชื่อ entity เป็น ch4vf_SHIFT_LEFT โดยสร้างไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vf_SHIFT_LEFT จากนั้นทำการทดลอง FPGA ทำงานเดียวกับการทดลองที่ 4.5.1 ทุกประการ โดยเขียนโค้ดชิฟต์รีจิสเตอร์ 4 บิตแบบ Shift left ที่รวมวงจรดีเบาเซอร์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_5_2vf

การทดลองที่ 4.5.3 การออกแบบวงจรดีเบาเซอร์สำหรับปุ่มกดแบบเบาๆเลส

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจ โค้ดของวงจรดีเบาเซอร์สำหรับปุ่มกดหรือคีย์บอร์ดแบบเบาๆเลส (Bounceless)
- 2) เพื่อสร้างวงจรดีเบาเซอร์ประสิทธิภาพสูง โดยวิธีที่ยืนด้วย โค้ด VHDL และโปรแกรมชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรดีเบาเซอร์สำหรับปุ่มกดแบบเบาๆเลสด้วย CPLD

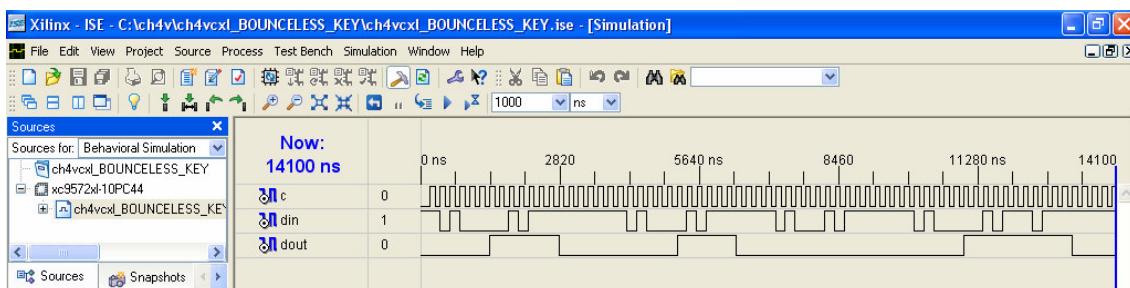
1.1) เขียน โค้ดของวงจรดีเบาเซอร์สำหรับปุ่มกดหรือคีย์บอร์ดแบบเบาๆเลส (Bounceless) ซึ่งให้ความไว้ทางเอกสาร พุตพัลส์ขึ้นกับเวลาที่กดคีย์บอร์ดแสดงดังรูปที่ L1.1 โดยสร้างไฟล์ใน Project Location ชื่อ ch4v จากนั้นกำหนด Project Name และ Source File ชื่อ ch4vcx1_BOUNCELESS_KEY บันทึกไฟล์แล้ว Check Syntax และทำ Behavioral simulation โดยใช้ไฟล์ชื่อ ch4vcx1_BOUNCELESS_KEY_tb1 และให้พิจารณาผล Behavioral simulation ดังรูปที่ L1.2 ว่าเป็นไปตามทฤษฎีหรือไม่

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ch4vcx1_BOUNCELESS_KEY is
6     Port ( C,Din : in STD_LOGIC;
7             Dout : out STD_LOGIC);
8 end ch4vcx1_BOUNCELESS_KEY;
9
10 architecture Behavioral of ch4vcx1_BOUNCELESS_KEY is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0):="0000";
12 begin
13     -----Bounceless keyboard-----
14 process(C)
15 begin
16     if (C'event and C='1') then
17         if (not Din)= QT(3) then QT <= QT(3) & QT(3) & QT(3)& QT(3);
18         else
19             QT <= QT(2 downto 0) & (not Din);
20         end if;
21     end process;
22     Dout <= QT(3);
23 end Behavioral;

```

รูปที่ L1.1 โค้ดของวงจรดีเบาเซอร์สำหรับปุ่มกดหรือคีย์บอร์ดแบบเบาๆเลส (Bounceless)



รูปที่ L1.2 ผล Behavioral simulation ของโค้ดในรูปที่ L1.1

1.2) เขียนโค้ดของวงจรทดสอบว่าชุด IC นี้สามารถรับสัญญาณขาเข้าแบบไม่มีจังหวะ (Bounceless) ได้รูปที่ L1.3 เพื่อป้อนพัลส์ให้กับวงจรนับตัวบินใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_5_3vcx1 นำไฟล์วงจรดีเบาเซอร์ชื่อ ch4vcx1_BOUCLELESS_KEY และไฟล์วงจรนับชื่อ ch4vcx1_c10_up มาทำเป็น Component แล้วทำการเขียนโค้ดซึ่งรวมโค้ดของวงจรนับขึ้น 15 บิตเพื่อสร้างความถี่ 128 Hz จาก F(7) (< 200 Hz) สำหรับจ่ายวงจรดีเบาเซอร์

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_5_3vcx1 is
7     Port ( CLR,Din,OSC : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex4_5_3vcx1;
10
11 architecture Behavioral of ex4_5_3vcx1 is
12     component ch4vcx1_BOUCLELESS_KEY
13         Port ( C,Din : in STD_LOGIC;
14                 Dout : out STD_LOGIC);
15     end component;
16     component ch4vcx1_c10_up
17         Port ( C,CLR : in STD_LOGIC;
18                 Q : out STD_LOGIC_VECTOR (3 downto 0));
19     end component;
20     signal C : STD_LOGIC;
21     signal F : STD_LOGIC_VECTOR (14 downto 0);
22 begin
23     -----15Bits Counter-----
24 process(OSC) --Output F(7)=128Hz,
25 begin
26     if OSC'event and OSC='1' then F <= F + 1;
27     end if;
28 end process;
29     -----DEBOUNCER-----
30 DEBOUNCER : ch4vcx1_BOUCLELESS_KEY
31     port map(C=>F(7),Din=>Din,Dout=>C);
32     -----COUNTER : C10_UP-----
33 COUNTER_10UP : ch4vcx1_c10_up
34     port map(C=>C,CLR=>CLR,Q=>Q);
35 end Behavioral;

```

รูปที่ L1.3 เขียนโค้ดของวงจรทดสอบว่าชุด IC นี้สามารถรับสัญญาณขาเข้าแบบไม่มีจังหวะ (Bounceless)

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz ปุ่มกด PB1 และ PB3(Slide SW1) เป็นอินพุต และ LED1-LED4 เป็นเอาต์พุต กล่าวคือ

Din = PB1 = INPUT = p39 Q(3) = LED1 = OUTPUT = p38 Q(1) = LED3 = OUTPUT = p36

CLR= PB3(Slide SW1) = INPUT= p42 Q(2) = LED2 = OUTPUT = p37 Q(0) = LED4 = OUTPUT = p35

OSC= OSC = OUTPUT = p5

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]						
I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
CLR	Input	p42	2	14		
Din	Input	p39	2	9		
OSC	Input	p5	1	9		
Q<3>	Output	p38	2	8	SLOW	
Q<2>	Output	p37	2	6	SLOW	
Q<1>	Output	p36	2	5	SLOW	
Q<0>	Output	p35	2	2	SLOW	

รูปที่ L1.4 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป CPLD แล้วให้เลื่อน Slide SW1 ไปที่ตำแหน่ง ON (CLR = '0') ให้กดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกดเร็วๆ PB1 แล้วกดค้างไว้แล้วให้คูลท์ LED1-LED4 ตามไปด้วยในขณะที่กดปุ่ม PB1 ว่าให้อาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นเลื่อน Slide SW1 ไปที่ตำแหน่ง OFF (CLR = '1') เพื่อเคลียร์อาต์พุตแล้ว ON แล้วทดลองซ้ำ

2 สร้างวงจรดีเบาเซอร์สำหรับปุ่มกดแบบเบาๆ เลสด้วย FPGA

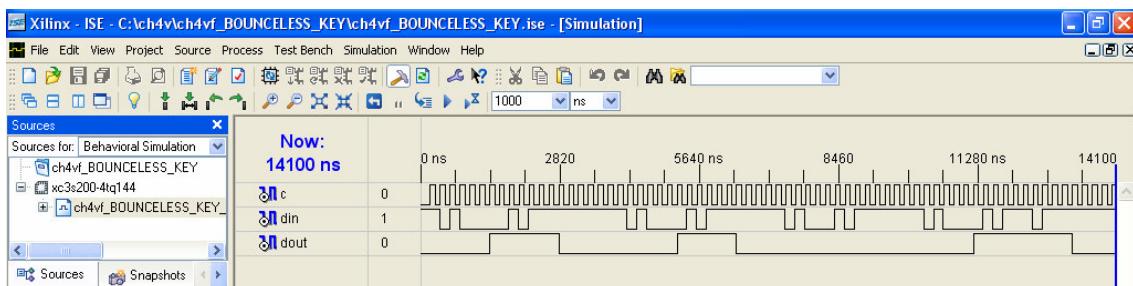
2.1) เขียนโค้ดของวงจรดีเบาเซอร์สำหรับปุ่มกดหรือคีบอร์ดแบบเบาๆ เลส (Bounceless) ซึ่งให้ความกว้างอาต์พุตพัลส์ขึ้นกับเวลาที่กดคีบอร์ดแปดคงดังรูปที่ L2.1 โดยสร้างไฟล์ใน Project Location ชื่อ ch4v จากนั้นกำหนด Project Name และ Source File ชื่อ ch4vf_BOUNCELESS_KEY บันทึกไฟล์แล้ว Check Syntax และทำ Behavioral simulation โดยใช้ไฟล์ชื่อ ch4vf_BOUNCELESS_KEY_tb1 แล้วให้พิจารณาผล Behavioral simulation ดังรูปที่ L2.2 ว่าเป็นไปตามทฤษฎีหรือไม่

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ch4vf_BOUNCELESS_KEY is
6      Port ( C,Din : in STD_LOGIC;
7             Dout : out STD_LOGIC);
8  end ch4vf_BOUNCELESS_KEY;
9
10 architecture Behavioral of ch4vf_BOUNCELESS_KEY is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0):="0000";
12 begin
13-----Bounceless keyboard-----
14 process(C)
15 begin
16     if (C'event and C='1') then
17         if (not Din)= QT(3) then QT <= QT(3) & QT(3) & QT(3) & QT(3);
18         else
19             QT <= QT(2 downto 0) & (not Din);
20         end if;
21     end process;
22     Dout <= QT(3);
23 end Behavioral;

```

รูปที่ L2.1 โค้ดของวงจรดีเบาเซอร์สำหรับปุ่มกดหรือคีบอร์ดแบบเบาๆ เลส (Bounceless)



รูปที่ L2.2 ผล Behavioral simulation ของโค้ดในรูปที่ L2.1

2.2) เขียนโค้ดของวงจรทดสอบวงจรดีเบาเซอร์สำหรับปุ่มกดหรือคีบอร์ดแบบเบาๆ เลส (Bounceless) ดังรูปที่ L2.3 เพื่อป้อนพัลส์ให้กับวงจรนับสิบใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_5_3vf นำไฟล์ร่วงวงจรดีเบาเซอร์ชื่อ ch4vf_BOUNCELESS_KEY และไฟล์ร่วงวงจนับชื่อ ch4vf_c10_up มาทำเป็น Component แล้วทำการเขียนโค้ดซึ่งรวมโค้ดของวงจนับขึ้น 24 บิตเพื่อกรองความถี่ 190.72 Hz จาก F(16) (< 200 Hz) สำหรับจ่ายวงจรดีเบาเซอร์

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_5_3vf is
7     Port ( CLR,Din,OSC : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex4_5_3vf;
10
11 architecture Behavioral of ex4_5_3vf is
12     component ch4vf_BOONCELESS_KEY
13         Port ( C,Din : in STD_LOGIC;
14                 Dout : out STD_LOGIC);
15     end component;
16     component ch4vf_c10_up
17         Port ( C,CLR : in STD_LOGIC;
18                 Q : out STD_LOGIC_VECTOR (3 downto 0));
19     end component;
20     signal C : STD_LOGIC;
21     signal F : STD_LOGIC_VECTOR (14 downto 0);
22 begin
23     -----24Bits Counter-----
24 process(OSC) --Output F(16)=190.72Hz
25     begin
26         if OSC'event and OSC='1' then F <= F + 1;
27         end if;
28     end process;
29     -----DEBOUNCER-----
30 DEBOUNCER : ch4vf_BOONCELESS_KEY
31     port map(C=>F(7),Din=>Din,Dout=>C);
32     -----COUNTER : C10_UP-----
33 COUNTER_10UP : ch4vf_c10_up
34     port map(C=>C,CLR=>CLR,Q=>Q);
35 end Behavioral;

```

รูปที่ L2.3 เขียนโค้ดของวงจรทดสอบบางจุดเชอร์สำหรับปุ่มกดหรือคีย์บอร์ดแบบเบาๆเดส (Bounceless)

การกำหนดขาสัญญาณต่างๆ จะใช้ OSC = 25 MHz ปุ่มกด PB1 และ Dip SW1 เป็นอินพุต มี LED L0-L3 เป็นเอาต์พุต

Din = PB1 = INPUT = p44 Q(3) = L3 = OUTPUT = p76 Q(1) = L1 = OUTPUT = p77

CLR=Dip SW1 = INPUT = p46 Q(2) = L2 = OUTPUT = p69 Q(0) = L0 = OUTPUT = p70

OSC = OSC= INPUT= p127

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLR	Input	p52	BANK LVC MOS33	N/A	3.30						Unknown		
Din	Input	p44	BANK LVC MOS33	N/A	3.30						Unknown		
OSC	Input	p127	BANK LVC MOS33	N/A	3.30						Unknown		
Q<0>	Output	p70	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
Q<1>	Output	p77	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
Q<2>	Output	p69	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		
Q<3>	Output	p76	BANK LVC MOS33	N/A	3.30			SLOW			Unknown		

รูปที่ L2.4 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป FPGA แล้วให้เลื่อน Dip SW1 ไปที่ตำแหน่ง ON (CLR = '0') ให้กดปุ่ม PB1 ไปเรื่อยๆ แล้วกับการกดเร็วๆ PB1 และกดค้างไว้แล้วให้คุณติ่ง LED L0-L3 ตามไปด้วยในขณะที่กดปุ่ม PB1 ว่าให้อาต์พุตเป็นไปตามทฤษฎีหรือไม่ หากนั้นเลื่อน Dip SW1 ไปที่ตำแหน่ง OFF (CLR= '1') เพื่อเคลียร์อาต์พุตแล้ว ON และทดลองซ้ำ

การทดลองที่ 4.5.4 การออกแบบวงจรดีเบาเซอร์สำหรับปุ่มกดแบบโมโนสเตเบิล

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจโค้ดวงจรดีเบาเซอร์สำหรับปุ่มกดแบบโมโนสเตเบิล (Monostable) หรือ One-shot
- 2) เพื่อสร้างวงจรดีเบาเซอร์ประสีทึกภาพสูง โดยจะเขียนด้วยโค้ด VHDL และโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรดีเบาเซอร์สำหรับปุ่มกดแบบโมโนสเตเบิลด้วย CPLD

เขียนโค้ดวงจรดีเบาเซอร์สำหรับปุ่มกดแบบโมโนสเตเบิลหรือ One-shot และดังรูปที่ L1.1 ซึ่งให้ความกว้างเอาต์พุต พลั๊กส์ค์ที่ไวไฟล์ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vcx1_ONE_SHOT

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ch4vcx1_ONE_SHOT is
6     Port ( C,Din : in STD_LOGIC;
7             Dout : out STD_LOGIC);
8 end ch4vcx1_ONE_SHOT;
9
10 architecture Behavioral of ch4vcx1_ONE_SHOT is
11     signal QT : STD_LOGIC_VECTOR (3 downto 0):="0000";
12 begin
13     -----4Bits Shift register : Shift left-----
14 process(C)
15 begin
16     if (C'event and C='1') then
17         QT <= (QT(2 downto 0)) & (not Din);
18     end if;
19 end process;
20     -----Dout-----
21     Dout <= QT(0) and QT(1) and QT(2) and (not QT(3));
22 end Behavioral;

```

รูปที่ L1.1 โค้ดวงจรดีเบาเซอร์สำหรับปุ่มกดแบบโมโนสเตเบิลหรือ One-shot

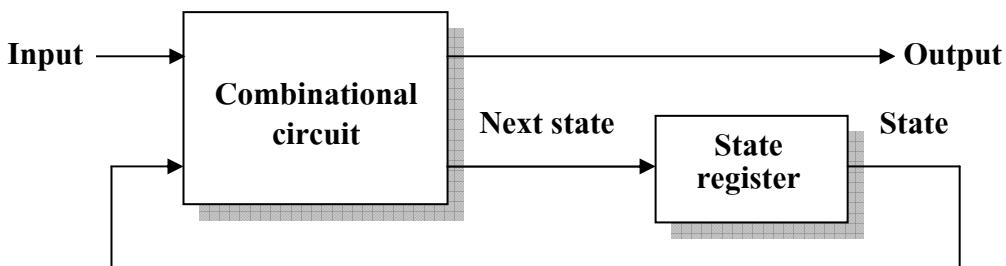
จากนี้ทำการทดลอง CPLD ทำนองเดียวกับการทดลองที่ 4.5.3 ทุกประการ โดยจะเขียนโค้ดของวงจรทดสอบของวงจรดีเบาเซอร์สำหรับปุ่มกดแบบโมโนสเตเบิลหรือ One-shot เพื่อป้อนพลั๊กส์ให้กับวงจรนับสิบใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File ชื่อ ex4_5_4vcx1 นำไฟล์วงจรดีเบาเซอร์ชื่อ ch4vcx1_ONE_SHOT และไฟล์วงจรนับชื่อ ch4vcx1_c10_up มาทำเป็น Component และทำการเขียนโค้ดซึ่งรวมโค้ดของวงจรนับขีน 15 บิตเพื่อใช้ในการสร้างความถี่ 128 Hz จาก F(7) (< 150 Hz) สำหรับจ่าของวงจรดีเบาเซอร์

2 สร้างวงจรดีเบาเซอร์สำหรับปุ่มกดแบบโมโนสเตเบิลด้วย FPGA

นำโค้ดวงจรดีเบาเซอร์สำหรับปุ่มกดแบบโมโนสเตเบิลหรือ One-shot ในรูปที่ L1.1 ซึ่งให้ความกว้างเอาต์พุตพลั๊กส์ค์ที่มาแก้ชื่อ entity เป็น ch4vf_ONE_SHOT โดยสร้างไฟล์ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_5_4vf จากนี้เขียนโค้ดการทดลอง FPGA ทำนองเดียวกับการทดลองที่ 4.5.3 ทุกประการ ซึ่งโค้ดนี้ได้รวมโค้ดของวงจรนับขีน 24 บิตเพื่อสร้างความถี่ 95.36 Hz จาก F(17) (< 150 Hz) สำหรับจ่าของวงจรดีเบาเซอร์แล้ว

4.6 การออกแบบวงจรซีเควนเชียลโดยใช้วิธี Finite state machine

วงจรซีเควนเชียล (Sequential circuit หรือ Sequential machine) จะประกอบด้วย วงจรออมบินเนชัน (Combinational circuit) และหน่วยความจำหรือจิสเตอร์ (State register) โดยที่เอาต์พุตหรือเอาต์พุตปัจจุบัน (Present output) ของวงจรจะขึ้นกับ ข้อมูลที่อยู่ในหน่วยความจำและอาจจะขึ้นกับอินพุตหรืออินพุตปัจจุบัน (Present input) อีกด้วย ซึ่งข้อมูลที่อยู่ในหน่วยความจำจะขึ้นกับอินพุตอดีต (Previous input) บล็อก ไดอะแกรมของวงจรซีเควนเชียลแสดงดังรูปที่ 4.51 ซึ่ง “ค่าของข้อมูลที่เก็บในหน่วยความจำ ปัจจุบัน” เราเรียกว่า “State” หรือ “Present state” หรือ “Current state”



รูปที่ 4.51 บล็อกไดอะแกรมของวงจรซีเควนเชียลโดยย่างง่าย

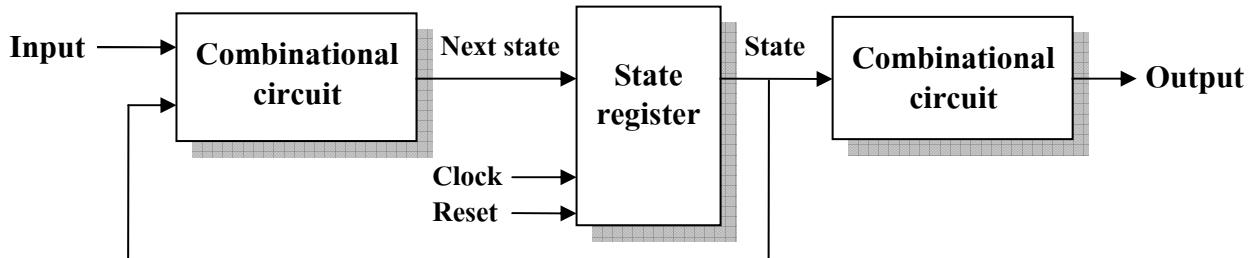
วงจนับและชิฟต์ริจิตเตอร์ที่ได้อธิบายไปแล้วถือเป็นวงจรซีเควนเชียลอย่างง่าย ในข้อ 4.6 นี้จะอธิบายการออกแบบ วงจรซีเควนเชียลอย่างเป็นระบบวิธี ซึ่งหมายความว่าการออกแบบวงจรที่มีความซับซ้อน วงจรซีเควนเชียลแบ่งเป็น 2 ประเภท คือ วงจรซีเควนเชียลแบบซิงไครนัส (Synchronous sequential circuit) และวงจรซีเควนเชียลแบบอะซิงไครนัส (Asynchronous sequential circuit) วงจรซีเควนเชียลแบบซิงไครนัสนั้นจำเป็นต้องใช้สัญญาณนาฬิกา (Clock) จึงเรียกอีกอย่างหนึ่งว่า Clocked sequential machines และเนื่องจากจำนวน State ของวงจรซีเควนเชียลมีจำกัด ดังนั้นวงจรซีเควนเชียลแบบซิงไครนัสอาจเรียกอีกอย่างหนึ่งว่า Finite state machine (FSM)

Xilinx synthesis tool หรือ XST จะรองรับเฉพาะการออกแบบวงจรซีเควนเชียลแบบซิงไครนัสเท่านั้น ดังนั้นในข้อ 4.6 นี้เราจะจึงอธิบายเฉพาะวงจรซีเควนเชียลแบบซิงไครนัส

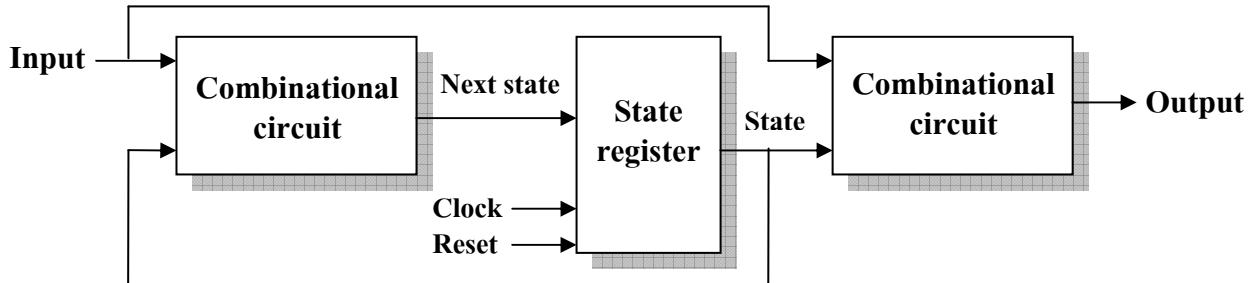
4.6.1 Finite state machine

Finite state machine (FSM) แบ่งตามชนิดของเอาต์พุตได้ 2 ชนิด คือ มาร์ (Moor-type FSM หรือ Moor machine) และ เมียลี (Mealy-type FSM หรือ Mealy machine) แสดงดังรูปที่ 4.52 และรูปที่ 4.53 ตามลำดับ โดยที่ Moor-type FSM นั้นเอาต์พุต (ปัจจุบัน) จะขึ้นกับข้อมูลที่เก็บในหน่วยความจำ (State register) ณ เวลาปัจจุบันหรือ State หรือ Present state เท่านั้น ในขณะที่ Mealy-type FSM นั้นเอาต์พุต (ปัจจุบัน) จะขึ้นกับข้อมูลที่เก็บในหน่วยความจำ (State register) ณ ปัจจุบันและอินพุต (ปัจจุบัน)

ในรูปที่ 4.52 และรูปที่ 4.53 นั้นจะมี Clock และอาจมี Reset เพิ่มเข้ามาที่ State register ซึ่งในการออกแบบ FSM นั้น คำว่า Reset ในที่นี้จะมีความหมายเดียวกับ Clear หรือ Asynchronous clear ซึ่งเป็นข้อยกเว้นในหนังสือเล่มนี้เพื่อให้สอดคล้อง กับตำราของต่างประเทศที่มักจะใช้คำนี้ เช่นเดียวกัน เมื่อป้อนอินพุต (สำม) ให้กับ Finite state machine แล้วมีการทริก State register (ซึ่งอาจจะเป็น D Flip-flop หรือ JK Flip-flop) ด้วย Clock ก็จะทำให้วงจร มีการเปลี่ยน State เป็น State ใหม่ เอาต์พุต ของ State ใหม่นี้จะถูกป้อนกลับไปที่อินพุตในส่วนของวงจรออมบินเนชัน (ด้านซ้าย) ทำให้ได้ค่า Next state ใหม่และค่าเอาต์พุต ใหม่ จากนั้นวงจรจะวนกลับมาทำงานในลักษณะเป็นวัฏจักรต่อไปเรื่อยๆ



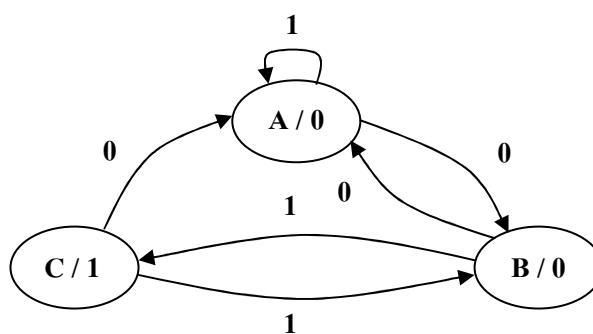
รูปที่ 4.52 บล็อกไซด์แกรมของ Moore-type FSM



รูปที่ 4.53 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 4.54 และรูปที่ 4.55 ตามลำดับ ในรูปที่ 4.54 นั้น ตัวอักษรภายในวงกลมเป็นชื่อ State และตัวเลขเป็นค่าเอาต์พุต เช่น ที่ State A วงจร มีเอาต์พุต = 0 การเปลี่ยน State (State transition) จะเป็นตามทิศทางของลูกศร โดยส่วนทางจะเป็น State ปัจจุบัน (Present state) และหัวจะเป็น State ถัดไป (Next state) ตัวเลขที่กำกับใกล้เส้น (Transition) จะเป็นค่าอินพุต การทำงานของวงจร เช่น ถ้าเริ่มต้นที่ State A (เอาต์พุต Z = 0) ถ้า อินพุต X = 0 และมี Clock ทริกแล้วจะจะเปลี่ยนไปที่ State B (เอาต์พุต Z = 0) ถ้าอินพุต X = 1 และมี Clock ทริกแล้วจะจะเปลี่ยนไปที่ State C (เอาต์พุต Z = 1) ถ้าอินพุต X = 1 และมี Clock ทริกแล้วจะจะเปลี่ยนกลับไปที่ State B (ที่ State B เอาต์พุต Z = 0) เป็นต้น

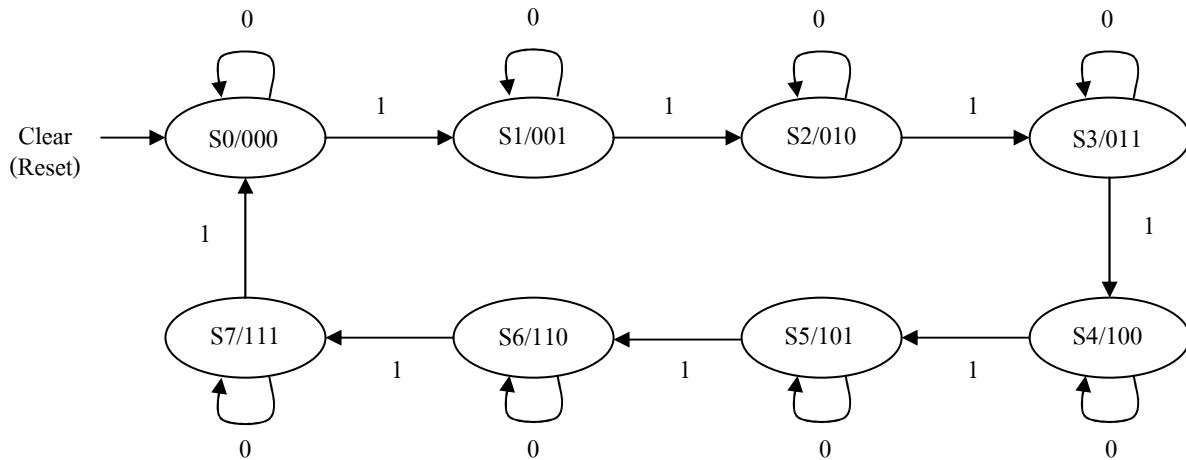


รูปที่ 4.54 State diagram ของ Moore-type FSM ที่มี X เป็นอินพุตและ Z เป็นเอาต์พุต

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

รูปที่ 4.55 State table ของ Moore-type FSM ที่มี X เป็นอินพุตและ Z เป็นเอาต์พุต

ตัวอย่างที่ 4.1 ให้เขียน State diagram และ State table ของวงจรนับ 8 แบบซิงโครนัสแบบบันทึกที่มีขา Clock enable input (CE) โดยที่ Z เป็นเอาต์พุตแบบอะเรย์ที่ประกอบด้วย $Z(2)$, $Z(1)$ และ $Z(0)$ วงจรจะนับเมื่อ $CE = 1$ และจะ Clear (Reset) $Z = "000"$ เมื่อขา Clear = '1' ซึ่งตัวอย่างนี้จะเป็น Moor-type FSM เพราะว่าเอาต์พุต Z ของวงจรนับ 8 ขึ้นกับ State เท่านั้น (ค่าเอาต์พุตไม่ได้แปรตามค่าอินพุต CE) จากนั้นจึงสร้าง State diagram และ State table แสดงดังรูปที่ 4.56 และรูปที่ 4.57 ตามลำดับ

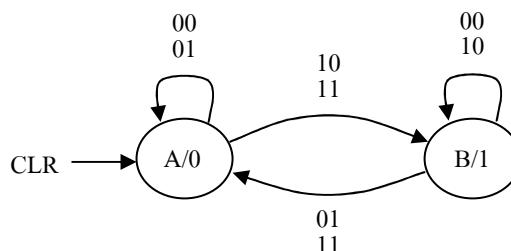


รูปที่ 4.56 State diagram ของวงจรนับ 8

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

รูปที่ 4.57 State table ของ Moor-type FSM

ตัวอย่างที่ 4.2 ให้เขียน State diagram และ State table ของ JK Flip-flop ที่มี J และ K เป็นอินพุตและมี Q เป็นเอาต์พุต วงจรจะ Clear ($Q = '0'$) เมื่อ CLR = '1' และจะเปลี่ยน State ทุกครั้งที่ทริกเกอร์ด้วยขอบนาฬิกาของ Clock (CLK) และเนื่องจากเอาต์พุตของ JK Flip-flop ขึ้นกับ State เท่านั้น (ค่าเอาต์พุตไม่ได้แปรตามค่าอินพุต J และ K) เราจึงเขียน State diagram แบบ Moor-type FSM ได้ดังรูปที่ 4.58 และเขียน State table ของ JK Flip-flop แสดงดังรูปที่ 4.59 และเพื่อความสะดวกเรากำหนดให้อินพุต X เป็นแบบอะเรย์ 2 มิต โดยที่ $X(1) = J$ และ $X(0) = K$



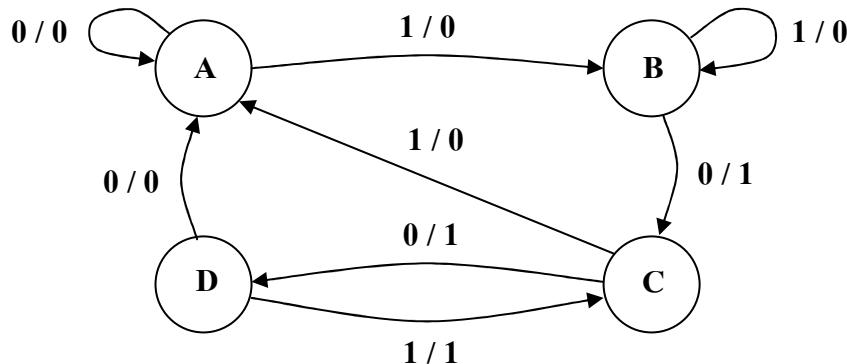
รูปที่ 4.58 State diagram ของ JK Flip-flop (Moor-type FSM)

Present state	Next state				Output Q
	X = 00	X = 01	X = 10	X = 11	
A	A	A	B	B	0
B	B	A	B	A	1

รูปที่ 4.59 State table ของช่อง JK Flip-flop (Moor-type FSM)

4.6.3 State diagram และ State table ของ Mealy-type FSM

การสร้าง State diagram และ State table ของ Mealy-type FSM ในการออกแบบวงจรซีเควนเชียล เพื่อให้เกิดความเข้าใจง่ายขึ้น ให้ผู้อ่านพิจารณาตัวอย่างดังในรูปที่ 4.60 และรูปที่ 4.61 ตามลำดับ ในรูปที่ 4.60 นั้นจะมีวงกลมแสดง State ต่างๆ โดยมีตัวอักษรที่อยู่ภายในเป็นชื่อ State การเปลี่ยน State (State transition) จะเป็นตามทิศของหัวลูกศร โดยที่ทางจะเป็น State ปัจจุบัน (Present state) ในขณะที่หัวลูกศรจะเป็น State ถัดไป (Next state) ตัวเลขที่กำกับไว้ใกล้เส้น (Transition) ด้านซ้ายจะเป็นอินพุตและด้านขวาจะเป็นเอาต์พุต การทำงานของวงจร เช่น ถ้าเริ่มต้นที่ State A ถ้าอินพุต X = 1 แล้วเอาต์พุต Z = 0 และเมื่อมี Clock ทริกเกอร์แล้วจะจะเปลี่ยนไปที่ State B ที่ State B ถ้าอินพุต X = 0 แล้วเอาต์พุต Z = 1 และเมื่อมี Clock ทริกเกอร์แล้วจะจะเปลี่ยนไปที่ State C ที่ State C ถ้าอินพุต X = 1 แล้วเอาต์พุต Z = 0 และเมื่อมี Clock ทริกเกอร์แล้วจะจะคงอยู่ที่ State A เราจะเห็นได้ว่า Mealy-type FSM นั้นเอาต์พุตจะขึ้นกับค่า State และอินพุต

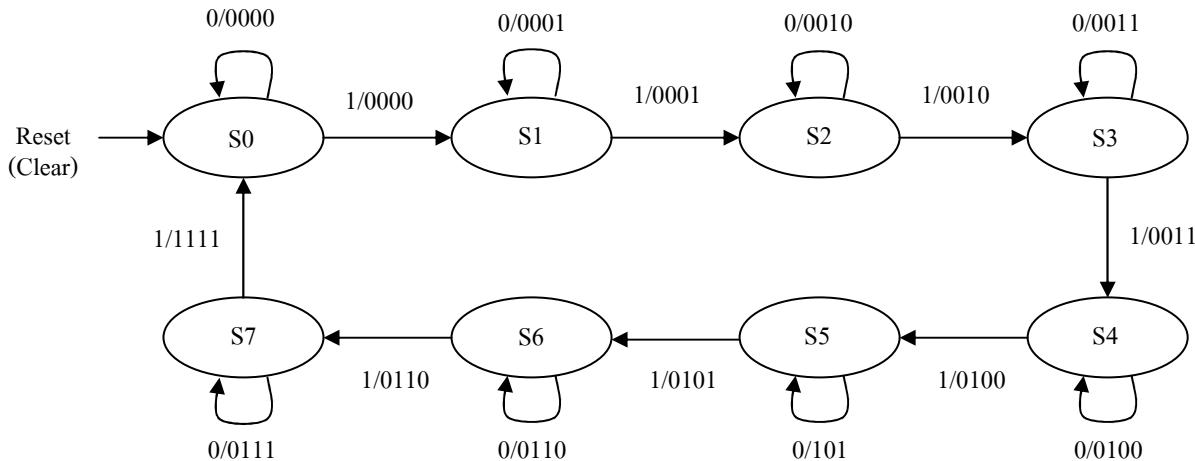


รูปที่ 4.60 State diagram ของ Mealy-type FSM ที่มี X เป็นอินพุตและ Z เป็นเอาต์พุต

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

รูปที่ 4.61 State table ของ Mealy-type FSM ที่มี X เป็นอินพุตและ Z เป็นเอาต์พุต

ตัวอย่างที่ 4.3 ให้เขียน State diagram และ State table ของวงจรนับ 8 แบบชิงไครนัสแบบนับขึ้นที่มี Clock enable input (CE) = X มีเอาต์พุต Z เป็นแบบอะเรย์ซึ่งประกอบด้วย Z(3), Z(2), Z(1) และ Z(0) โดยที่ Q = Z(2 downto 0) และ CEO = Z(3) วงจรนี้จะนับเมื่ออินพุต CE หรือ X = '1' และจะ Clear (Reset) Z = "0000" เมื่อ Clear = '1' เมื่อวงจรนับถึงค่าสูงสุดถ้า X = '1' แล้ว Z(3) หรือ Clock enable output (CEO) = '1' ยกเว้นถ้า X = '0' และ Z(3) = '0' ดังนั้นวงจรนี้เป็น Mealy-type FSM จากนั้นจึงสร้าง State diagram และ State table และแสดงดังรูปที่ 4.62 และรูปที่ 4.63 ตามลำดับ



รูปที่ 4.62 State diagram ของ Mealy-type FSM (วงจรนับ 8) ที่มี X เป็นอินพุตและ Z เป็นเอาต์พุต

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

รูปที่ 4.63 State table ของ Mealy-type FSM (วงจรนับ 8) ที่มี X เป็นอินพุตและ Z เป็นเอาต์พุต

4.6.4 ขั้นตอนการออกแบบ Finite state machine

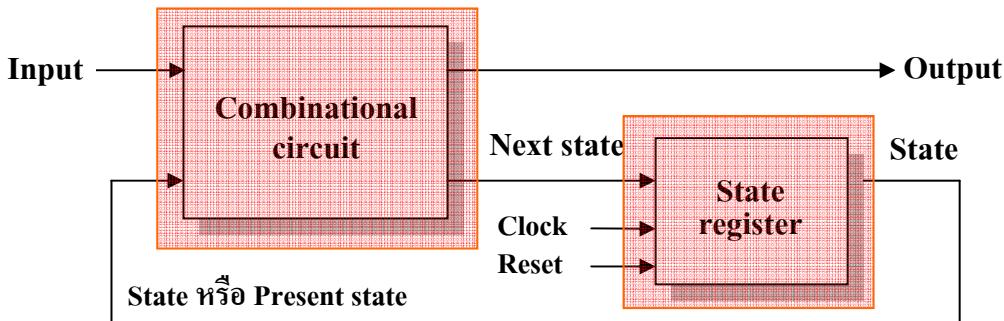
การออกแบบวงจรชีวภาพแบบชิงโกรนัสหรือ Finite state machine (FSM) สามารถสรุปได้ดังต่อไปนี้

- 1) หารายละเอียดของวงจรชีวภาพแบบชิงโกรนัสที่ต้องการออกแบบ แล้วนำข้อมูลที่ได้ไปสร้าง State diagram และ/หรือ State table ซึ่งอาจเป็น Moor-type FSM หรือ Mealy-type FSM ถ้ามี State ที่ซ้ำซ้อนกันก็ทำการลดจำนวน State (State reduction) ก่อนเพื่อประหยัดจำนวน Flip-flop ซึ่งวิธีการลด State สามารถอ่านได้จากหนังสือออกแบบดิจิตอลทั่วๆ ไป
- 2) กำหนดค่าให้กับ State (State assignment) แต่ละ State ลงใน State table เรียกว่า State-assigned table แล้วกำหนดชนิดและจำนวนฟลิป-ฟล๊อปที่ใช้กับวงจรที่ออกแบบ จากนั้นนำ State-assigned table ไปสร้างตารางความจริงเพื่อนำไปหาสมการบูลินของวงจร Next state ที่ใช้ป้อนให้กับอินพุตของ Flip-flop และหาสมการบูลินของเอาต์พุต
- 3) ใช้สมการบูลินที่ได้มาเป็นข้อมูลในการสร้างเป็นวงจรตามที่เราต้องการออกแบบ

ซึ่งรายละเอียดในการออกแบบวงจรชีวภาพแบบชิงโกรนัสหรือ Finite state machine (FSM) นั้นผู้อ่านสามารถค้นคว้าเพิ่มเติมได้จากหนังสือออกแบบวงจรดิจิตอลทั่วๆ ไปได้

4.6.5 การออกแบบ Finite state machine โดยใช้ซอฟต์แวร์ชุด

การออกแบบ Finite state machine ไม่ว่าจะเป็น Moor-type FSM หรือ Mealy-type FSM นั้นสามารถเขียนโค้ดโดยแยกวงจรเป็น 2 ส่วน คือ วงจรคอมบินेशัน (Combinational circuit) และรีจิสเตอร์ (State register) และคงดังรูปที่ 4.64 รูปแบบการเขียนโค้ด VHDL มี 2 แบบคือ แบบธรรมชาติและแบบมีรีจิสเตอร์แล็ตช์ค่าเอ้าต์พุต (Registered output) เพื่อซิงโครไนซ์กับ Clock ซึ่งอาจมีความจำเป็นด้องใช้เพื่อกำกับหากลิตช์ (Glitch) ที่เอ้าต์พุตของ Mealy-type FSM เมื่อจากการเปลี่ยนแปลงที่อินพุต



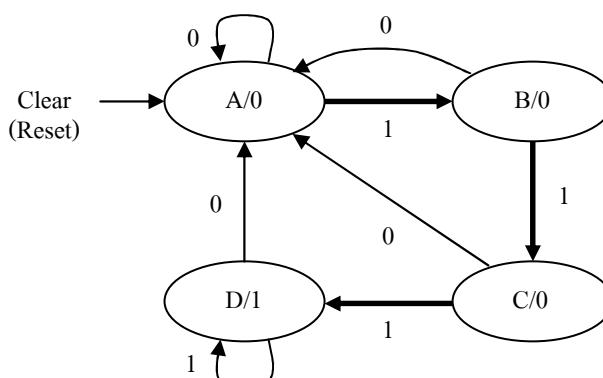
รูปที่ 4.64 บล็อกไกด์ของวงจรซีเควนเชียลออกบ่าย่างง่าย

ตัวอย่างที่ 4.4 ออกแบบวงจรตรวจจับลำดับ (Sequence detector) เป็น Moor-type FSM (หรือ Mealy-type FSM ได้เช่นกัน) มี X เป็นอินพุตและ Z เป็นเอ้าต์พุต โดยที่วงจรนี้จะให้อเอ้าต์พุต Z = '1' ก็ต่อเมื่อตรวจจับลอกอินพุต X = '1' จำนวน 3 ตัวติดต่อกัน

จากโจทย์ความสามารถเขียน State diagram และ State table และคงดังรูปที่ 4.65 และรูปที่ 4.656 โดยมีหลักการคิดดังนี้

- 1) ถ้า A เป็น State เริ่มต้น เมื่อตรวจจับอินพุต X = '0' ได้ ก็ให้ State อญ্তิ State A เช่นเดิม
- 2) เมื่อตรวจจับอินพุต X = '1' ตัวแรกได้แล้ว ให้วงจรเปลี่ยน State ไปที่ State B
- 3) ที่ State B เมื่อตรวจจับอินพุต X = '1' ตัวที่ 2 ได้แล้ว ให้เปลี่ยน State ไปที่ State C
- 4) ที่ State C เมื่อตรวจจับอินพุต X = '1' ตัวที่ 3 ได้แล้ว ให้เปลี่ยน State ไปที่ State D
- 5) จากนั้นให้เขียนทั้ง 4 State นี้เป็นมาเก็บอ่อนพร้อมเขียนเส้นที่มีลูกศรชี้ไปข้าง State ตัดไป จากนั้นจึงค่อยๆ เขียน State diagram และ State table ให้เป็นตามข้อกำหนดจนแล้วเสร็จ จะได้รูปที่ 4.65 และตารางในรูปที่ 4.66 ตามลำดับ

จากรูปที่ 4.65 หรือตารางในรูปที่ 4.66 นำไปเขียนโค้ดได้ดังรูปที่ 4.67 ผลการสังเคราะห์วงจรจะได้รูปที่ 4.68 และผลจำลองการทำงานของโค้ด VHDL ที่เป็น Moor-type FSM ของวงจรตรวจจับลำดับแสดงดังรูปที่ 4.69



รูปที่ 4.65 State diagram ของวงจรตรวจจับลำดับ

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

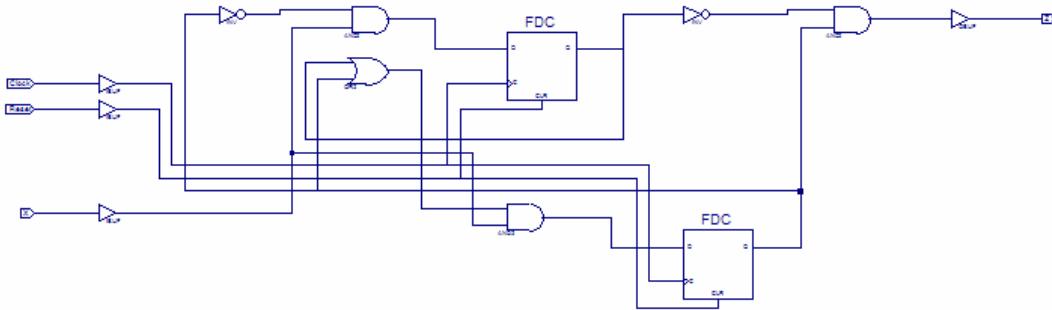
รูปที่ 4.66 State table ของ Moor-type FSM ของวงจรตรวจสอบคำดับ

```

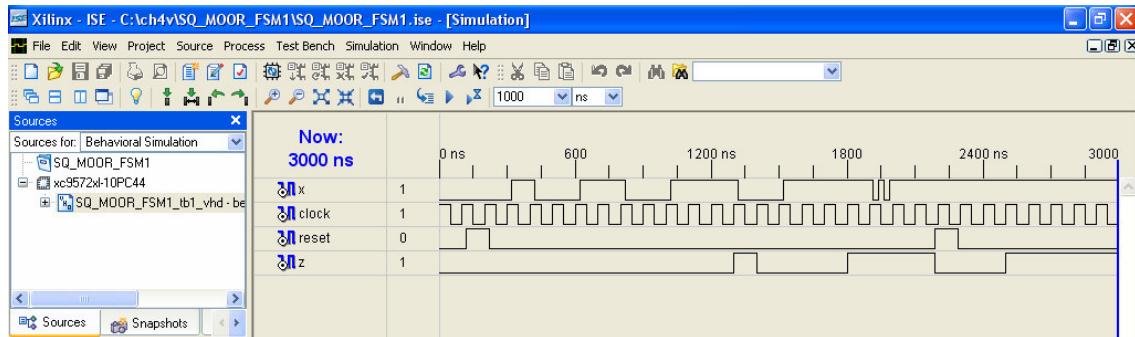
2 -- Moor-type FSM.
3
4 library IEEE;
5 use IEEE.STD_LOGIC_1164.ALL;
6
7 entity SQ_MOOR_FSM1 is
8     Port ( X,Clock,Reset : in STD_LOGIC;
9             Z : out STD_LOGIC);
10 end SQ_MOOR_FSM1;
11
12 architecture Behavioral of SQ_MOOR_FSM1 is
13     type State_type is (A,B,C,D);
14     signal State,Next_state : State_type ;
15 begin
16
17 P1_combination : process (X,State)
18 begin
19     case State is
20         when A =>
21             Z <= '0';
22             if X ='1' then
23                 Next_state <= B;
24             else
25                 Next_state <= A;
26             end if;
27         when B =>
28             Z <= '0';
29             if X ='1' then
30                 Next_state <= C;
31             else
32                 Next_state <= A;
33             end if;
34         when C =>
35             Z <= '0';
36             if X ='1' then
37                 Next_state <= D;
38             else
39                 Next_state <= A;
40             end if;
41         when D =>
42             Z <= '1';
43             if X ='1' then
44                 Next_state <= D;
45             else
46                 Next_state <= A;
47             end if;
48     end case;
49 end process P1_combination;
50
51 P2_state_resister : process (Clock,Reset)
52 begin
53     if (Reset ='1') then
54         State <= A;
55     elsif (Clock'event and Clock ='1') then
56         State <= Next_state;
57     end if;
58 end process P2_state_resister;
59
60 end Behavioral;

```

รูปที่ 4.67 โค้ด VHDL ที่เป็น Moor-type FSM ของวงจรตรวจสอบคำดับ



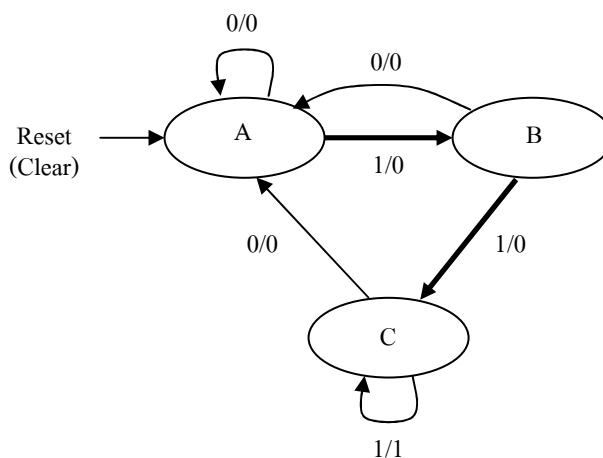
รูปที่ 4.68 ผลการ สังเคราะห์วงจรตรวจจับลำดับ



รูปที่ 4.69 ตัวอย่างผลจำลองการทำงานของโค้ด VHDL ที่เป็น Moor-type FSM ของวงจรตรวจจับลำดับ

ตัวอย่างที่ 4.5 ออกแบบวงจรตรวจจับลำดับ (Sequence detector) เป็น Mealy-type FSM มี X เป็นอินพุต และ Z เป็นเอาต์พุต โดยที่วงจรนี้จะให้ $Z = 1$ เมื่อตรวจจับลอกซิก $X = 1$ จำนวน 3 ตัวติดต่อกัน จากโจทย์เราราบารณาเพิ่ม State diagram และ State table แสดงดังรูปที่ 4.70 และรูปที่ 4.71 ตามลำดับ โดยหลักการคิดในการเขียน State diagram เป็นดังนี้

- 1) ถ้า A เป็น State เริ่มต้น เมื่อตรวจจับอินพุต $X = 0$ ได้ ก็ให้ State อยู่ที่ State A เช่นเดิม
- 2) เมื่อตรวจจับอินพุต $X = 1$ ตัวแรกได้แล้ว ให้วางเปลี่ยน State เป็น State B
- 3) ที่ State B เมื่อตรวจจับอินพุต $X = 1$ ตัวที่ 2 ได้แล้ว ให้เปลี่ยน State เป็น State C
- 4) ที่ State C เมื่อตรวจจับอินพุต $X = 1$ ตัวที่ 3 แล้วเอาต์พุต $Z = 1$ ทันที และให้ค่าว่า State เปลี่ยนเป็น State ใด?
- 5) จากนั้นให้เพิ่งทั้ง 3 State นี้ขึ้นมา ก่อน เขียนเส้นที่มีลูกศรชี้ไปยัง State ถัดไปพร้อมกับกำหนดค่าอินพุต/เอาต์พุต แล้วจึงค่อยๆ เขียน State diagram ให้เป็นตามข้อกำหนดจนแล้วเสร็จ จากนั้นจึงเขียน State table



รูปที่ 4.70 State diagram ของวงจรตรวจจับลำดับ

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

รูปที่ 4.71 State table ของ Mealy-type FSM ของวงจรตรวจจับลำดับ

นำ State diagram และ State table ในรูปที่ 4.70 และรูปที่ 4.71 ไปเขียนโค้ดที่เป็น FSM แบบชั้นดานและ FSM แบบที่มี Registered output เพื่อทำให้อาพุตที่ได้ซิงโกรไนซ์กับ Clock และดังรูปที่ 4.72 และรูปที่ 4.73 ผลการสังเคราะห์วงจรจะได้รูปที่ 4.74a) และรูปที่ 4.74b) และด้านข้างผลจำลองการทำงานของโค้ด VHDL ที่เป็น Mealy-type FSM ของวงจรตรวจจับลำดับ และดังรูปที่ 4.75a) และรูปที่ 4.75b) ตามลำดับ

```

2  -- Mealy-type FSM.
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  entity SQ_MEALY_FSM1 is
8      Port ( X,Clock,Reset : in STD_LOGIC;
9             Z : out STD_LOGIC);
10 end SQ_MEALY_FSM1;
11
12 architecture Behavioral of SQ_MEALY_FSM1 is
13     type State_type is (A,B,C);
14     signal State,Next_state : State_type ;
15 begin
16
17 P1_combination : process (X,State)
18     begin
19         case State is
20             when A =>
21                 if X ='1' then
22                     Next_state <= B;
23                     Z <= '0';
24                 else
25                     Next_state <= A;
26                     Z <= '0';
27                 end if;
28             when B =>
29                 if X ='1' then
30                     Next_state <= C;
31                     Z <= '0';
32                 else
33                     Next_state <= A;
34                     Z <= '0';
35                 end if;
36             when C =>
37                 if X ='1' then
38                     Next_state <= C;
39                     Z <= '1';
40                 else
41                     Next_state <= A;
42                     Z <= '0';
43                 end if;
44             end case;
45 end process P1_combination;
46
47 P2_state_resister : process (Clock,Reset)
48     begin

```

```

49      if (Reset ='1') then
50          State <= A;
51      elsif (Clock'event and Clock ='1') then
52          State <= Next_state;
53      end if;
54  end process P2_state_resister;
55
56 end Behavioral;

```

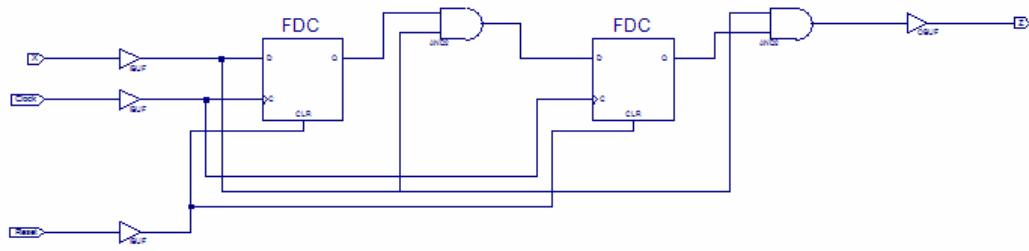
รูปที่ 4.72 โค้ด VHDL ที่เป็น Mealy-type FSM แบบธรรมชาตของวงจรตรวจจับคำดับ

```

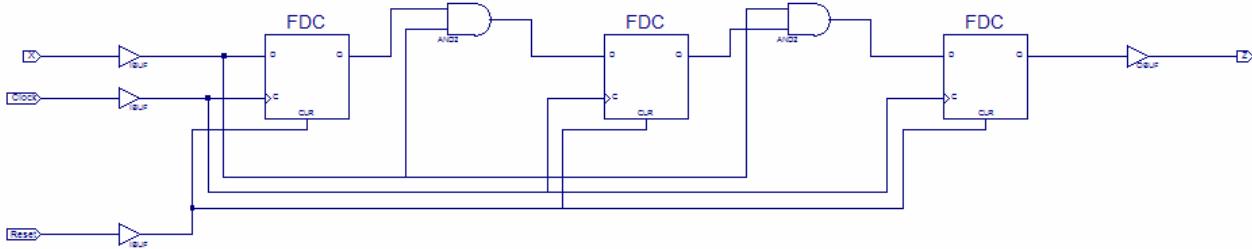
2 -- Mealy-type FSM with registered output.
3
4 library IEEE;
5 use IEEE.STD_LOGIC_1164.ALL;
6
7 entity SQ_MEALY_FSM1 is
8     Port ( X,Clock,Reset : in STD_LOGIC;
9             Z : out STD_LOGIC);
10    end SQ_MEALY_FSM1;
11
12 architecture Behavioral of SQ_MEALY_FSM1 is
13     type State_type is (A,B,C);
14     signal State,Next_state : State_type ;
15     signal Z_temp : STD_LOGIC;
16 begin
17
18 P1_combination : process (X,State)
19 begin
20     case State is
21         when A =>
22             if X ='1' then
23                 Next_state <= B;
24                 Z_temp <= '0';
25             else
26                 Next_state <= A;
27                 Z_temp <= '0';
28             end if;
29         when B =>
30             if X ='1' then
31                 Next_state <= C;
32                 Z_temp <= '0';
33             else
34                 Next_state <= A;
35                 Z_temp <= '0';
36             end if;
37         when C =>
38             if X ='1' then
39                 Next_state <= C;
40                 Z_temp <= '1';
41             else
42                 Next_state <= A;
43                 Z_temp <= '0';
44             end if;
45         end case;
46    end process P1_combination;
47
48 P2_state_resister : process (Clock,Reset)
49 begin
50     if (Reset ='1') then
51         State <= A;      --Clear state resister
52         Z <= '0';        --Clear output resister
53     elsif (Clock'event and Clock ='1') then
54         State <= Next_state;
55         Z <= Z_temp;
56     end if;
57  end process P2_state_resister;
58
59 end Behavioral;

```

รูปที่ 4.73 โค้ด VHDL ที่เป็น Mealy-type FSM แบบที่มี Registered output ของวงจรตรวจจับคำดับ

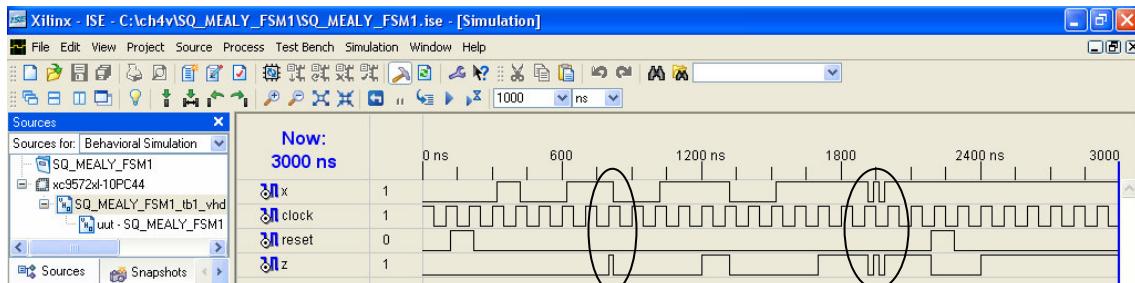


4.74a) ผลการสังเคราะห์วงจรที่เป็น Mealy-type FSM แบบชรรรมดา

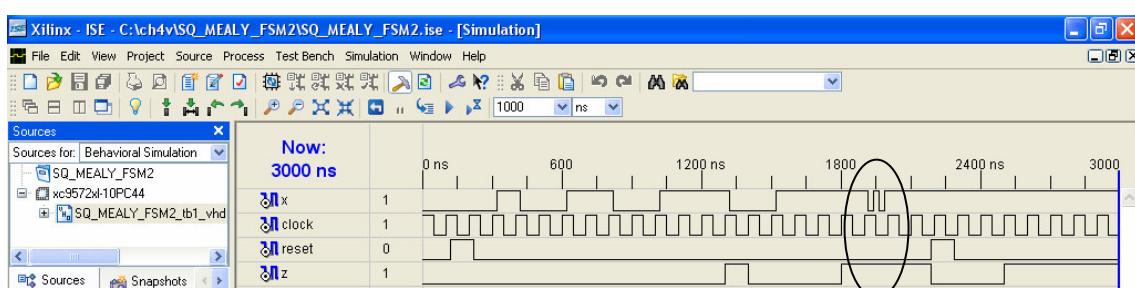


4.74b) ผลการสังเคราะห์วงจรที่เป็น Mealy-type FSM แบบที่มี Registered output

รูปที่ 4.74 ผลการสังเคราะห์วงจรที่เป็น Mealy-type FSM ทั้ง 2 แบบของวงจรตรวจจับลำดับ



4.75a) ผลจำลองการทำงาน Mealy-type FSM แบบชรรรมดาจะเกิดกลิตช์ที่เอาต์พุต



4.75b) ผลจำลองการทำงาน Mealy-type FSM แบบ Registered output ซึ่งไม่เกิดกลิตช์แต่เอาพุตเอาต์จะล่าช้าอยู่ 1 Clock

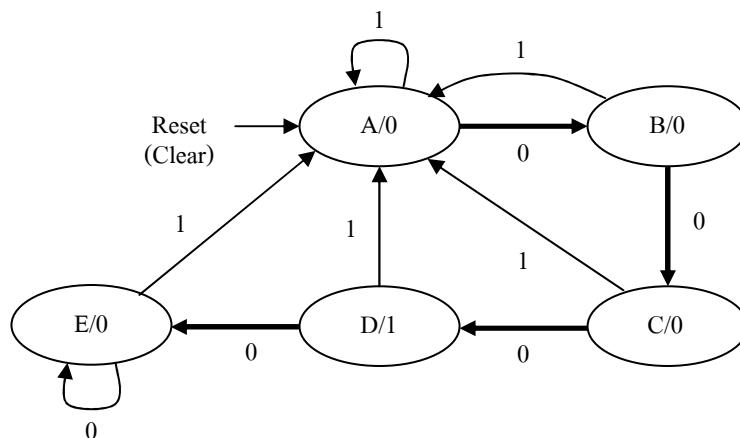
รูปที่ 4.75 ผลจำลองการทำงานของโก้ด VHDL ที่เป็น Mealy-type FSM ทั้ง 2 แบบของวงจรตรวจจับลำดับ

จากรูปที่ 4.75a) จะพบว่า Mealy-type FSM แบบชรรรมดาจะเกิดกลิตช์ที่เอาต์พุต (บริเวณวงกลม) แต่จากรูปที่ 4.75b) จะพบว่าใน Mealy-type FSM แบบ Registered output จะไม่เกิดกลิตช์ (บริเวณวงกลม) แต่เอาต์พุตจะช้า (Delay) กว่า Mealy-type FSM แบบชรרรมดาอยู่ 1 Clock ซึ่ง Mealy-type FSM แบบ Registered output ในรูปที่ 4.75b) นี้จะให้ผลลัพธ์เหมือน Moor-type FSM ในรูปที่ 4.69

ตัวอย่างที่ 4.6 ออกแบบวงจร โมโนสเตเบิล (Monostable) ประสาทชีพสูง ที่มี X เป็นอินพุต Active low และ Z เป็นเอาต์พุต โดยที่วงจรนี้จะให้อาต์พุต Z = '1' กว้างเพียง 1 คาบของ Clock ก็ต่อเมื่อตรวจสอบจับล็อก X = '0' ติดต่อกันไม่น้อยกว่า 3 คาบ

จากโจทย์เรามารอเขียน State diagram และ State table ของเป็น Moor-type FSM และรูปที่ 4.76a) และรูปที่ 4.76b) ตามลำดับ โดยที่ “-” คือ Don't care หลักการคิดในการเขียน State diagram เป็นดังนี้

- 1) ถ้า A เป็น State เริ่มต้น เมื่อตรวจสอบอินพุต X = '1' ให้ State อญญาติ State A เช่นเดิม
- 2) เมื่อตรวจสอบจับอินพุต X = '0' ตัวแรกได้แล้ว ให้วงจรเปลี่ยน State เป็น State B
- 3) ที่ State B เมื่อตรวจสอบจับอินพุต X = '0' ตัวที่ 2 ได้แล้ว ให้เปลี่ยน State เป็น State C
- 4) ที่ State C เมื่อตรวจสอบจับอินพุต X = '0' ตัวที่ 3 ได้แล้ว ให้เปลี่ยน State เป็น State D
- 5) ที่ State D เมื่อตรวจสอบจับอินพุต X = '0' ตัวถัดๆ ไปแล้ว ให้เปลี่ยน State เป็น State E
- 6) จากนั้นให้เขียนทั้ง 5 State นี้ขึ้นมา ก่อน เขียนเส้นที่มีลูกศรชี้ไปยัง State ถัดไปพร้อมกับกำหนดค่าอินพุต จากนั้น แล้วจึงค่อยๆ เขียน State diagram ให้เป็นตามข้อกำหนดจนแล้วเสร็จ จากนั้นจึงเขียน State table



4.76a) State diagram ของวงจร โมโนสเตเบิล

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

4.76b) State table ของ Moor-type FSM ของวงจร โมโนสเตเบิล

รูปที่ 4.76 State diagram และ State table ของ Moor-type FSM ของวงจร โมโนสเตเบิล

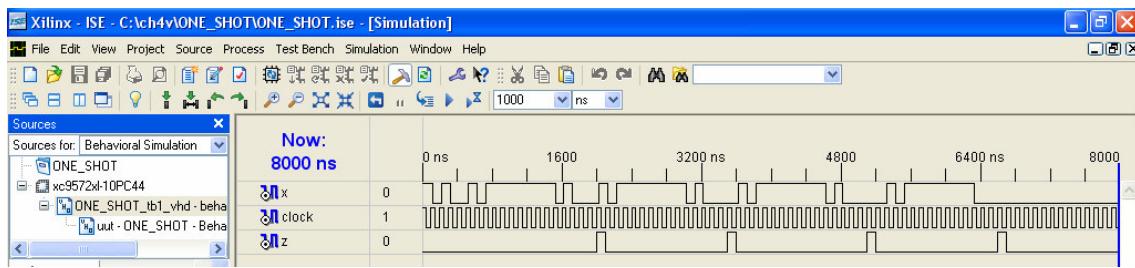
นำ State diagram และ State table ในรูปที่ 4.76a) และรูปที่ 4.76b) ไปเขียนโค้ดที่เป็น FSM ได้ดังรูปที่ 4.77 ตัวอย่าง ผลจำลองการทำงาน โค้ด VHDL ที่เป็น Moor-type FSM ของวงจร โมโนสเตเบิลแสดงดังรูปที่ 4.78 ซึ่งจะทำงานเหมือนกับวงจร โมโนสเตเบิลที่ได้อธิบายไปแล้วในหัวข้อ 4.5 รูปที่ 4.47 ซึ่งผู้อ่านจะเห็นได้ว่าการออกแบบวงจรซึ่งความเรียลไทม์โดยใช้ Finite state machine นั้นจะเป็นการออกแบบวงจรอย่างเป็นระบบวิธีโดยไม่จำเป็นต้องให้ทักษะและประสบการณ์ในการออกแบบวงจร ดังที่ได้อธิบายในรูปที่ 4.47 และขั้นตอนหลัก Flip-flop อีกด้วย ซึ่งอาจจะใช้ Flip-flop เพียง 3 ตัวเท่านั้น (จำนวน State = 5 $\leq 2^3$)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ONE_SHOT is
6     Port ( X,Clock : in STD_LOGIC;
7             Z : out STD_LOGIC);
8 end ONE_SHOT;
9
10 architecture Behavioral of ONE_SHOT is
11     type State_type is (A,B,C,D,E);
12     signal State : State_type;
13     signal Next_state : State_type ;
14 begin
15
16 P1_combination : process (X,State)
17 begin
18     case State is
19         when A =>
20             Z <= '0';
21             if X ='0' then
22                 Next_state <= B;
23             else
24                 Next_state <= A;
25             end if;
26         when B =>
27             Z <= '0';
28             if X ='0' then
29                 Next_state <= C;
30             else
31                 Next_state <= A;
32             end if;
33         when C =>
34             Z <= '0';
35             if X ='0' then
36                 Next_state <= D;
37             else
38                 Next_state <= A;
39             end if;
40         when D =>
41             Z <= '1';
42             if X ='0' then
43                 Next_state <= E;
44             else
45                 Next_state <= A;
46             end if;
47         when E =>
48             Z <= '0';
49             if X ='0' then
50                 Next_state <= E;
51             else
52                 Next_state <= A;
53             end if;
54     end case;
55 end process P1_combination;
56
57 P2_state_resister : process (Clock)
58 begin
59     if  (Clock'event and Clock ='1')  then
60         State <= Next_state;
61     end if;
62 end process P2_state_resister;
63 end Behavioral;

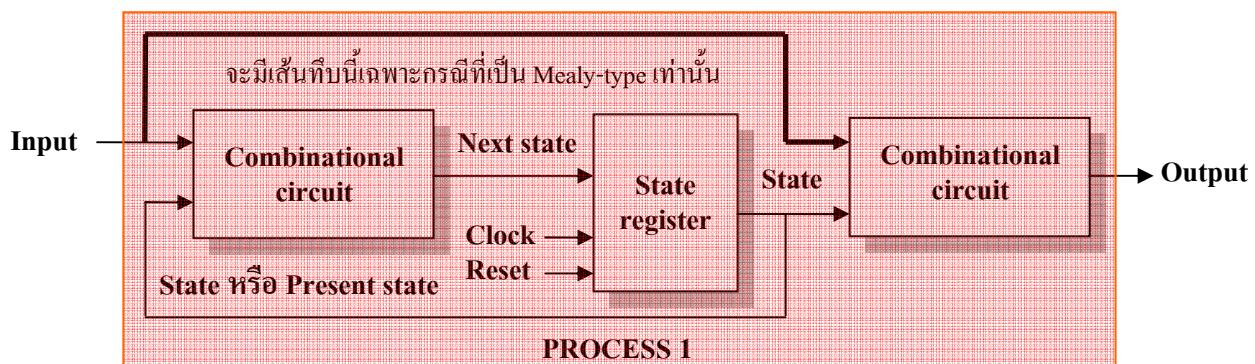
```

รูปที่ 4.77 โค้ด VHDL ที่เป็น Moore-type FSMของวงจรไมโครสเกตเบด

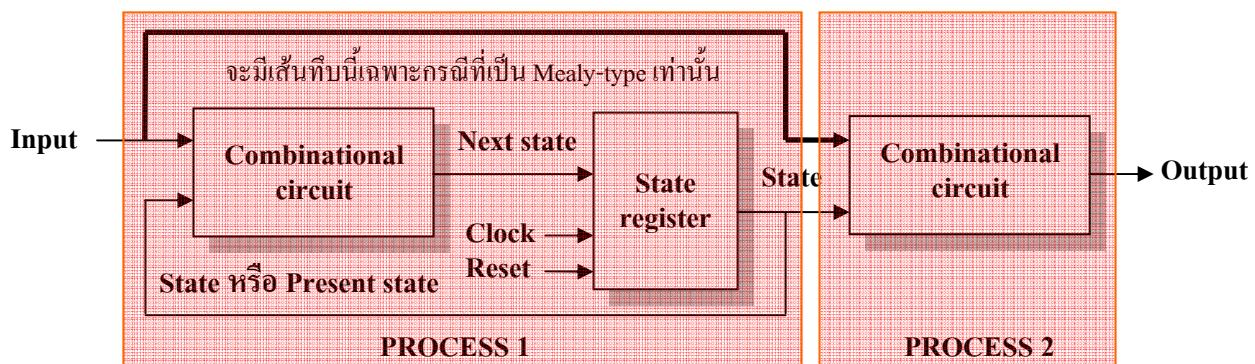


รูปที่ 4.78 ตัวอย่างผลจำลองการทำงานของโค้ด VHDL ที่เป็น Moore-type FSM ของวงจรโนโนสเตเบิล

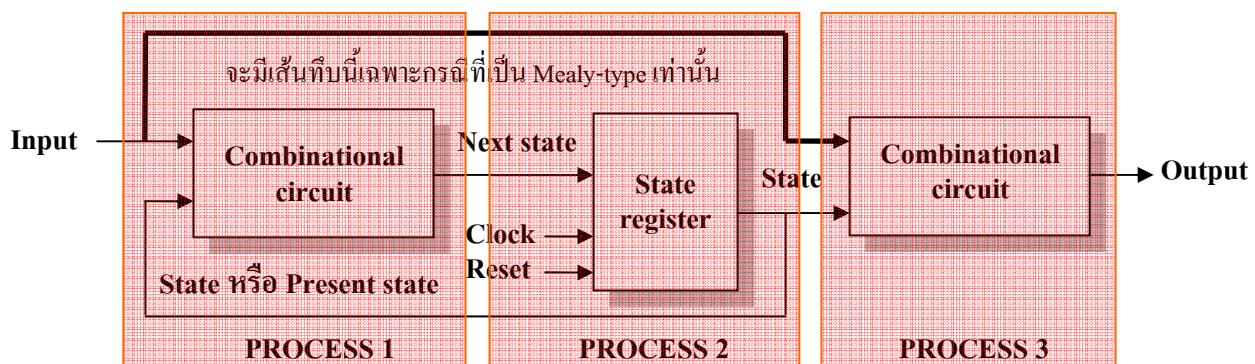
การเขียนโค้ดของ FSM มีหลายส่วนๆ เราอาจเขียนโค้ดโดยแยกวงจรเป็นส่วนๆ ดังรูปที่ 4.79 ถึงรูปที่ 4.81 โดยที่แต่ละ Process จะแทนส่วนของวงจรย่อย จึงเรียกว่าเป็น FSM with 1 process, FSM with 2 processes และ FSM with 3 processes ตามลำดับ โค้ดแบบ FSM with 1 process นั้นจะเป็นแบบ Registered output เพื่อแลตซ์ค่าเอาต์พุตให้ซิงค์ไนซ์กับ Clock



รูปที่ 4.79 บล็อกไอดีอะแกรมของ FSM with 1 process (with registered output)



รูปที่ 4.80 บล็อกไอดีอะแกรมของ FSM with 2 processes



รูปที่ 4.81 บล็อกไอดีอะแกรมของ FSM with 3 processes

เพื่อให้เห็นสไตร์การเขียนโค้ดทั้ง 3 วิธี เราจะนำตัวอย่างที่ 4.4 และตัวอย่างที่ 4.5 มาเขียนใหม่ โดยในตัวอย่างที่ 4.4 ซึ่งเป็น Moor-type FSM นั้นเราจะเขียนโค้ดแบบ FSM with 2 processes, FSM with 3 processes และ FSM with 1 process และดังรูปที่ 4.82a) ถึงรูปที่ 4.82c) ส่วนตัวอย่างที่ 4.5 ซึ่งเป็น Mealy-type FSM นั้นเราจะเขียนโค้ดแบบ FSM with 2 processes, FSM with 3 processes และ FSM with 1 process และดังรูปที่ 4.83a) ถึงรูปที่ 4.83c) ตามลำดับ ซึ่ง FSM with 1 process ในรูปที่ 4.82c) และรูปที่ 4.83c) นั้นจะเป็นแบบ Registered output

```

2  -- Moor-type FSM with 2 processes.
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  entity SQ_MOOR_FSM_2P is
8      Port ( X,Clock,Reset : in STD_LOGIC;
9             Z : out STD_LOGIC);
10 end SQ_MOOR_FSM_2P ;
11
12 architecture Behavioral of SQ_MOOR_FSM_2P is
13     type State_type is (A,B,C,D);
14     signal State : State_type ;
15 begin
16 process1: process (Clock,Reset)
17 begin
18     if (Reset ='1') then
19         State <= A;
20     elsif (Clock'event and Clock='1') then
21         case State is
22             when A =>
23                 if X ='1' then
24                     State <= B;
25                 else
26                     State <= A;
27                 end if;
28             when B =>
29                 if X ='1' then
30                     State <= C;
31                 else
32                     State <= A;
33                 end if;
34             when C =>
35                 if X ='1' then
36                     State <= D;
37                 else
38                     State <= A;
39                 end if;
40             when D =>
41                 if X ='1' then
42                     State <= D;
43                 else
44                     State <= A;
45                 end if;
46             end case;
47         end if;
48     end process process1;
49
50 process2 : process (State)
51 begin
52     case State is
53         when A => Z <= '0';
54         when B => Z <= '0';
55         when C => Z <= '0';
56         when D => Z <= '1';
57     end case;
58 end process process2;
59
60 end Behavioral;
```

4.82a) โค้ด VHDL ที่ปั่น Moor-type FSM with 2 processes ของวงจรตราจั๊บลำดับ

```

2  -- Moor-type FSM with 3 processes.
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  entity SQ_MOOR_FSM_3P is
8      Port ( X,Clock,Reset : in STD_LOGIC;
9             Z : out STD_LOGIC);
10 end SQ_MOOR_FSM_3P ;
11
12 architecture Behavioral of SQ_MOOR_FSM_3P is
13     type State_type is (A,B,C,D);
14     signal State, Next_state : State_type ;
15 begin
16 process1: process (State,X)
17 begin
18     case State is
19         when A =>
20             if X ='1' then
21                 Next_state <= B;
22             else
23                 Next_state <= A;
24             end if;
25         when B =>
26             if X ='1' then
27                 Next_state <= C;
28             else
29                 Next_state <= A;
30             end if;
31         when C =>
32             if X ='1' then
33                 Next_state <= D;
34             else
35                 Next_state <= A;
36             end if;
37         when D =>
38             if X ='1' then
39                 Next_state <= D;
40             else
41                 Next_state <= A;
42             end if;
43     end case;
44 end process process1;
45
46 process2: process (Clock,Reset)
47 begin
48     if (Reset ='1') then
49         State <= A;
50     elsif (Clock'event and Clock='1') then
51         State <= Next_state;
52     end if;
53 end process process2;
54
55 process3 : process (State)
56 begin
57     case State is
58         when A => Z <= '0';
59         when B => Z <= '0';
60         when C => Z <= '0';
61         when D => Z <= '1';
62     end case;
63 end process process3;
64
65 end Behavioral;

```

4.82b) โค้ด VHDL ที่เขียน Moor-type FSM with 3 processes ของวงจรตรวจข้อมูลคำนับ

```

2  -- Moor-type FSM with 1 process (with output register).
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  entity SQ_MOOR_FSM_1P  is
8      Port ( X,Clock,Reset : in  STD_LOGIC;
9             Z : out  STD_LOGIC);
10 end SQ_MOOR_FSM_1P ;
11
12 architecture Behavioral of SQ_MOOR_FSM_1P  is
13     type State_type is (A,B,C,D);
14     signal State : State_type ;
15 begin
16 process1: process (Clock,Reset)
17 begin
18     if (Reset ='1') then
19         State <= A;
20         Z <= '0';
21     elsif (Clock'event and Clock='1') then
22         case State is
23             when A =>
24                 Z <= '0';
25                 if X ='1' then
26                     State <= B;
27                 else
28                     State <= A;
29                 end if;
30             when B =>
31                 Z <= '0';
32                 if X ='1' then
33                     State <= C;
34                 else
35                     State <= A;
36                 end if;
37             when C =>
38                 Z <= '0';
39                 if X ='1' then
40                     State <= D;
41                 else
42                     State <= A;
43                 end if;
44             when D =>
45                 Z <= '1';
46                 if X ='1' then
47                     State <= D;
48                 else
49                     State <= A;
50                 end if;
51         end case;
52     end if;
53 end process process1;

```

4.82c) โค้ด VHDL ที่เป็น Moor-type FSM with 1 process (with registered output) ของวงจรตรวจขับคำดับ

รูปที่ 4.82 โค้ด VHDL ที่เป็น Moor-type FSM ของวงจรตรวจขับคำดับ

```

2  -- Mealy-type FSM with 2 processes.
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  entity SQ_MEALY_FSM_2P  is
8      Port ( X,Clock,Reset : in  STD_LOGIC;
9             Z : out  STD_LOGIC);
10 end SQ_MEALY_FSM_2P ;
11
12 architecture Behavioral of SQ_MEALY_FSM_2P  is
13     type State_type is (A,B,C);
14     signal State : State_type ;

```

```

15 begin
16 process1: process (Clock,Reset)
17 begin
18     if (Reset ='1') then
19         State <= A;
20     elsif (Clock'event and Clock='1') then
21         case State is
22             when A =>
23                 if X ='1' then
24                     State <= B;
25                 else
26                     State <= A;
27                 end if;
28             when B =>
29                 if X ='1' then
30                     State <= C;
31                 else
32                     State <= A;
33                 end if;
34             when C =>
35                 if X ='1' then
36                     State <= C;
37                 else
38                     State <= A;
39                 end if;
40             end case;
41         end if;
42     end process process1;
43
44 process2 : process (State)
45 begin
46     case State is
47         when A => Z <= '0';
48         when B => Z <= '0';
49         when C =>
50             if X ='1' then
51                 Z <= '1';
52             else
53                 Z <= '0';
54             end if;
55     end case;
56 end process process2;
57
58 end Behavioral;

```

4.83a) โค้ด VHDL ที่เป็น Mealy-type FSM with 2 processes ของวงจรตรวจจับคำดับ

```

2 -- Mealy-type FSM with 3 processes.
3
4 library IEEE;
5 use IEEE.STD_LOGIC_1164.ALL;
6
7 entity SQ_MEALY_FSM_3P  is
8     Port ( X,Clock,Reset : in  STD_LOGIC;
9            Z : out  STD_LOGIC);
10 end SQ_MEALY_FSM_3P ;
11
12 architecture Behavioral of SQ_MEALY_FSM_3P  is
13     type State_type is (A,B,C);
14     signal State, Next_state : State_type ;
15 begin
16 process1: process (State,X)
17 begin
18     case State is
19         when A =>
20             if X ='1' then
21                 Next_state <= B;
22             else
23                 Next_state <= A;
24             end if;

```

```

25      when B =>
26          if X ='1' then
27              Next_state <= C;
28          else
29              Next_state <= A;
30          end if;
31      when C =>
32          if X ='1' then
33              Next_state <= C;
34          else
35              Next_state <= A;
36          end if;
37      end case;
38  end process process1;
39
40 process2: process (Clock,Reset)
41 begin
42     if (Reset ='1') then
43         State <= A;
44     elsif (Clock'event and Clock='1') then
45         State <= Next_state;
46     end if;
47 end process process2;
48
49 process3 : process (State)
50 begin
51     case State is
52         when A => Z <= '0';
53         when B => Z <= '0';
54         when C =>
55             if X ='1' then
56                 Z <= '1';
57             else
58                 Z <= '0';
59             end if;
60     end case;
61 end process process3;
62
63 end Behavioral;

```

4.83b) โค้ด VHDL ที่เป็น Mealy-type FSM with 3 processes ของจริงจะขับคำดับ

```

2 -- Mealy-type FSM with 1 process (with output register).
3
4 library IEEE;
5 use IEEE.STD_LOGIC_1164.ALL;
6
7 entity SQ_MEALY_FSM_1P is
8     Port ( X,Clock,Reset : in STD_LOGIC;
9             Z : out STD_LOGIC);
10 end SQ_MEALY_FSM_1P ;
11
12 architecture Behavioral of SQ_MEALY_FSM_1P is
13     type State_type is (A,B,C);
14     signal State : State_type ;
15 begin
16 process1: process (Clock,Reset)
17 begin
18     if (Reset ='1') then
19         State <= A;
20         Z <= '0';
21     elsif (Clock'event and Clock='1') then
22         case State is
23             when A =>
24                 Z <= '0';
25                 if X ='1' then
26                     State <= B;
27                 else
28                     State <= A;
29                 end if;

```

```

30      when B =>
31          Z <= '0';
32          if X ='1' then
33              State <= C;
34          else
35              State <= A;
36          end if;
37      when C =>
38          if X ='1' then
39              State <= C;
40              Z <= '1';
41          else
42              State <= A;
43              Z <= '0';
44          end if;
45      end case;
46  end if;
47 end process process1;
48
49 end Behavioral;

```

4.83c) โค้ด VHDL ที่เป็น Mealy-type FSM with 1 process (with registered output) ของวงจรตรวจจับคำดับ

รูปที่ 4.83 โค้ด VHDL ที่เป็น Mealy-type FSM ของวงจรตรวจจับคำดับ

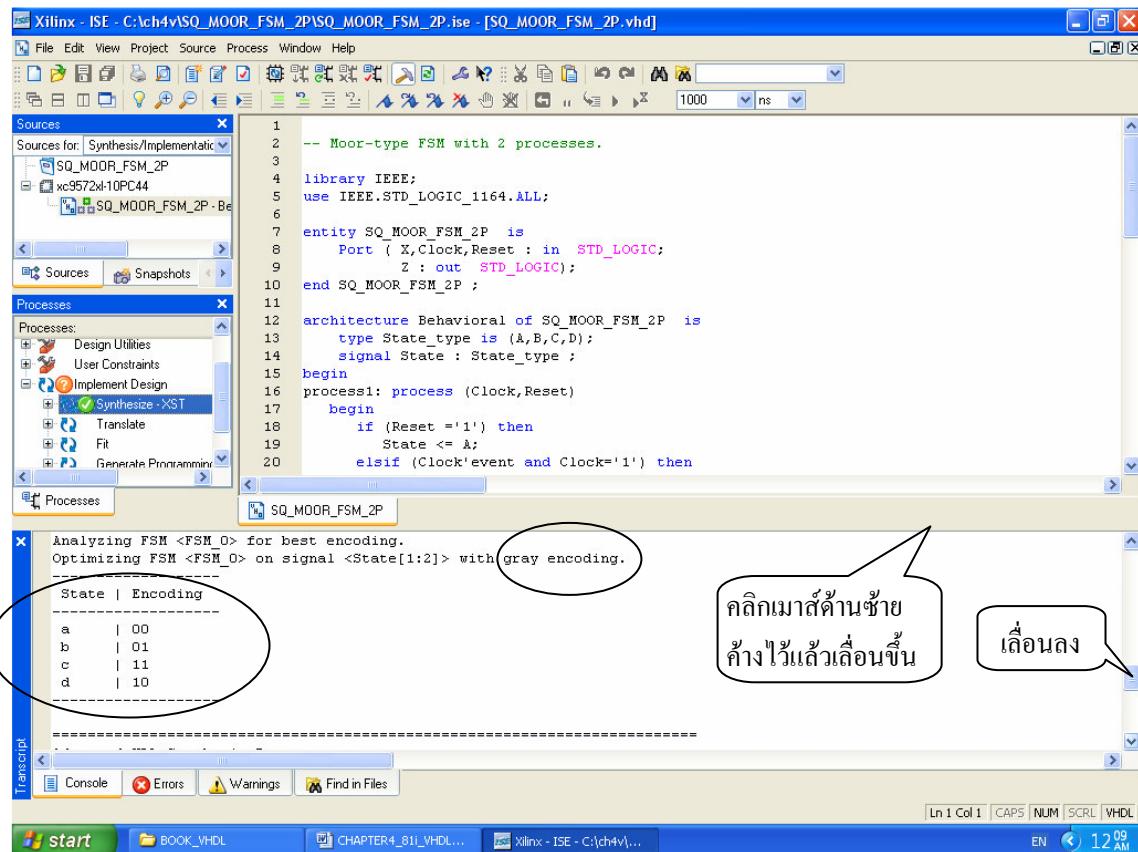
4.6.6 State encoding

การกำหนดค่าให้กับ State (State assignment หรือ State encoding) ในการออกแบบ FSM ขนาดใหญ่ด้วย CPLD และ FPGA นั้นจะไม่เหมือนกับวิธีทั่วไป การเน้นประหดจำนวน Flip-flop และลดขนาดวงจรคอมบินेशันหรือจำนวนเกต (Area) อาจทำให้ไม่ได้แรงที่มีสมรรถนะที่ดี (Speed) การทำ State encoding จึงควรคำนึงถึงอุปกรณ์พื้นฐานที่มีอยู่ภายใน ไอชีด้วย เพื่อให้ได้แรงที่ประหดและได้สมรรถนะที่ดี การสร้าง FSM ขนาดใหญ่โดยใช้ CPLD นั้นการทำ State encoding เป็นเลข รหัสไบนาเรีย (Binary) หรือเกรย์ (Gray) เพื่อประหด Flip-flop เนื่องจาก CPLD มีจำนวน Flip-flop จำกัด ซึ่งจะต่างจาก FPGA ที่มี Flip-flop เป็นจำนวนมาก การสร้าง FSM ขนาดใหญ่โดยใช้ FPGA จึงควรทำ State encoding แบบ “One-hot” เพื่อทำให้วงจรคอมบินेशันมีขนาดเล็กลงและได้สมรรถนะที่ดี ตัวอย่างการเข้ารหัสแบบต่างๆ ที่ใช้ทำ State encoding แสดงดังรูปที่ 4.84

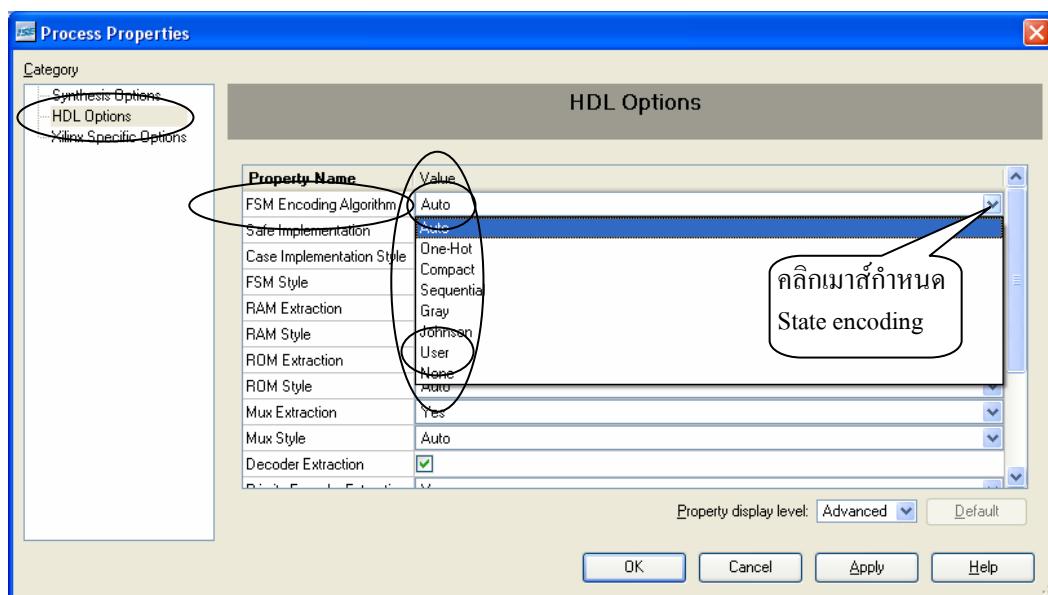
No.	Binary	Gray	One-hot
0	000	000	00000001
1	001	001	00000010
2	010	011	00000100
3	011	010	00001000
4	100	110	00010000
5	101	111	00100000
6	110	101	01000000
7	111	100	10000000

รูปที่ 4.84 ตารางแสดงการเข้ารหัสแบบต่างๆ ที่ใช้ทำ State encoding

Xilinx Synthesis Tool หรือ XST ได้กำหนดการทำ State encoding ไว้ล่วงหน้า (Default) เป็นแบบ โดยอัตโนมัติ (Auto) โดย XST พยายามเลือกทำวิธีที่ให้ผลลัพธ์ดีที่สุด (Optimize) ตัวอย่างการทำ State encoding ของวงจรตรวจจับคำดับในรูปที่ 4.82a) โดยอัตโนมัติแสดงดังรูปที่ 4.85 ซึ่งในกรณีนี้จะทำ State encoding เป็นรหัสเกรย์ แต่ยังไร์คามด้วยข้อจำกัดบางประการของซอฟต์แวร์ทุกจึงอาจไม่ได้ผลลัพธ์ตามที่เราต้องการ เช่น Speed เป็นต้น ดังนั้นซอฟต์แวร์ทุก XST จึงยอมให้ผู้ออกแบบเลือกวิธีทำ State encoding ได้เองดังรูปที่ 4.86 โดยคลิกที่ User ถ้าต้องการประหดฟล็อปให้คลิกที่ Compact

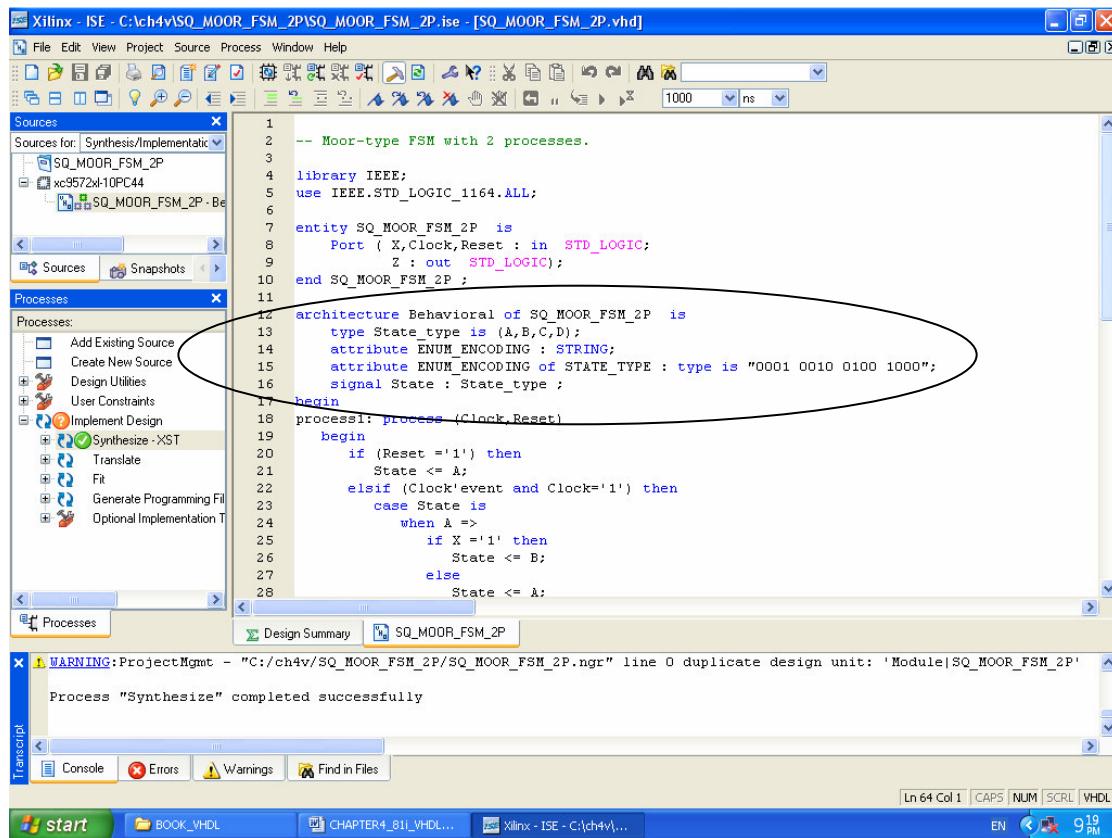


รูปที่ 4.85 การทำ State encoding โดยอัตโนมัติของໂโคଡ VHDL ที่เป็น Moore-type FSM with 2 process ของวงจรตรวจขับคำดับ



รูปที่ 4.86 รายการที่ XST ยอนให้ผู้ออกแบบเลือกทำ State encoding ได้

สำหรับซอฟต์แวร์ทุก XST และของบริษัทผู้พัฒนาฯ บางรายจะใช้วิธีกำหนด State encoding โดยใช้ User defined attribute ตัวอย่างการทำ State encoding ในໂโคଡ VHDL ของวงจรตรวจขับคำดับในรูปที่ 4.82a) โดยผู้ใช้จะเป็นผู้กำหนด State encoding เอง (เป็นแบบ One-hot โดยการเขียนໂโคଡ VHDL เพิ่มเข้าไปในบรรทัดที่ 14-15) และคงดังในรูปที่ 4.87 ซอฟต์แวร์ทุก XST จะยอนให้ผู้ใช้กำหนด State encoding ได้โดยคลิกที่ User ในรูปที่ 4.86



รูปที่ 4.87 การกำหนด State encoding โดยผู้ใช้เป็นผู้กำหนดลงในโค้ด VHDL ของจริงตรวจจับลำดับ

4.6.7 Incompletely specified state table

FSM ที่อธิบายมาทั้งหมดคือ State table แบบ Completely specified กล่าวคือไม่มี Don't care “-” ปรากฏอยู่ในคอลัมน์ Next state และ/หรือ Output แต่ยังไรมีความในทางปฏิบัติ FSM อาจมี Next state และ/หรือ Output เป็น Don't care “-” FSM นี้จึงมี State table เป็นแบบ Incompletely specified ตัวอย่าง State table แบบ Incompletely specified แสดงดังรูปที่ 4.88

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

รูปที่ 4.88 ตัวอย่าง State table ของ Mealy-type FSM แบบ Imcomplete

State table ที่เป็นแบบ Incompletely specified และแบบ Completely specified ในทางปฏิบัตินั้นอาจมี State ที่ซ้ำซ้อนกัน การลดจำนวน State จะทำให้จำนวนเร็วสุดๆ และเกตที่ต้องใช้ลดลง วิธีการลดจำนวน State เช่น วิธีลดจำนวน State โดยใช้ Implication table และ Merger diagram เป็นต้น ซึ่งวิธีการลด State ผู้อ่านสามารถอ่านได้จากหนังสือออกแบบดิจิตอลทั่วไป

การทดลองที่ 4.6.1 ออกแบบวงจรนับ 10 แบบซิงโครนัสโดยใช้ชีวีชี FSM

วัตถุประสงค์

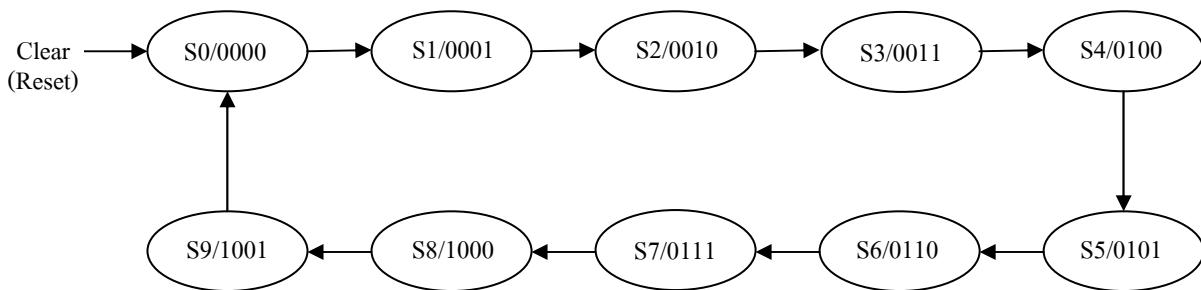
- 1) เพื่อออกแบบวงจรนับ 10 แบบซิงโครนัสโดยใช้ชีวีชี FSM
- 2) เพื่อสร้างวงจรนับ 10 ที่เป็นวงจรนับขึ้น โดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรนับ 10 แบบซิงโครนัสแบบขึ้นด้วย CPLD

1.1) สร้าง State diagram และ State table ดังรูปที่ L1.1 และสร้างไฟล์ของໄก์ดวงจรนับ 10 แบบนับขึ้นดังรูปที่ L1.2 ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vcxl_c10_up_FSM บันทึกไฟล์แล้ว Check syntax ทำ Behavioral simulation โดยใช้ไฟล์ชื่อ ch4vcxl_c10_up_FSM_tb1 และพิจารณาผลในรูปที่ L1.3 ว่าเป็นไปตามทฤษฎีหรือไม่



L1.1a) State diagram ของวงจรนับ 10 (ไม่มีอินพุตโดยตรง)

Present state	Next state	Output Z
S0	S1	0000
S1	S2	0001
S2	S3	0010
S3	S4	0011
S4	S5	0100
S5	S6	0101
S6	S7	0110
S7	S8	0111
S8	S9	1000
S9	S0	1001

L1.1b) State table ของ Moor-type FSM ของวงจรนับ 10 (ไม่มีอินพุตโดยตรง)

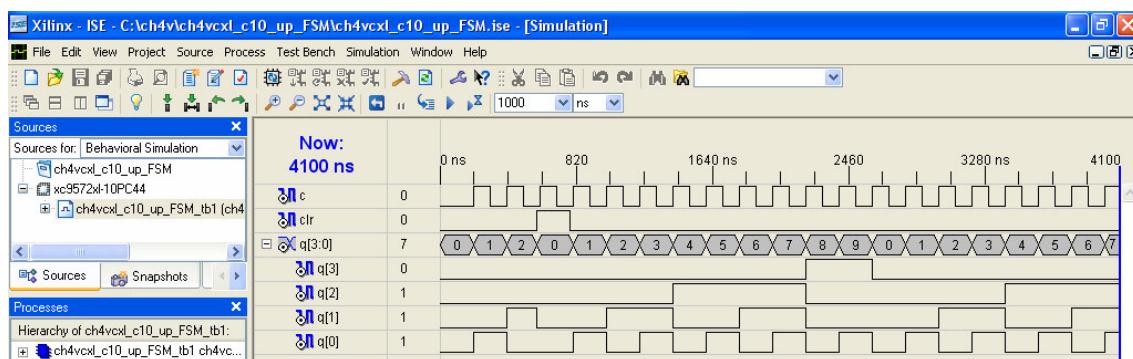
รูปที่ L1.1 State diagram และ State table ของ Moor-type FSM ของวงจรนับ 10 แบบนับขึ้น

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ch4vcxl_c10_up_FSM is
6     Port ( C,CLR : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8 end ch4vcxl_c10_up_FSM;
9
10 architecture Behavioral of ch4vcxl_c10_up_FSM is
11     type State_type is (S0,S1,S2,S3,S4,S5,S6,S7,S8,S9);
12     signal State,Next_state : State_type ;
13 begin
14
15 P1_combination : process (State)
16     begin
17         case State is
18             when S0 => Q <= "0000";
19                 Next_state <= S1;
20             when S1 => Q <= "0001";
21                 Next_state <= S2;
22             when S2 => Q <= "0010";
23                 Next_state <= S3;
24             when S3 => Q <= "0011";
25                 Next_state <= S4;
26             when S4 => Q <= "0100";
27                 Next_state <= S5;
28             when S5 => Q <= "0101";
29                 Next_state <= S6;
30             when S6 => Q <= "0110";
31                 Next_state <= S7;
32             when S7 => Q <= "0111";
33                 Next_state <= S8;
34             when S8 => Q <= "1000";
35                 Next_state <= S9;
36             when S9 => Q <= "1001";
37                 Next_state <= S0;
38         end case;
39     end process P1_combination;
40
41 P2_state_register : process (C,CLR)
42     begin
43         if (CLR ='1') then
44             State <= S0;
45         elsif (C'event and C ='1') then
46             State <= Next_state;
47         end if;
48     end process P2_state_register;
49
50 end Behavioral;

```

รูปที่ L1.2 โค้ดวงจรนับ 10 แบบนับขึ้น โดยใช้วิธี FSM



รูปที่ L1.3 ผล Behavioral simulation ของโค้ดวงจรนับ 10 แบบนับขึ้น โดยใช้วิธี FSM

1.2) เขียนโค้ดของวงจรนับ 10 แบบนับขึ้นที่รวมวงจรดีเบาเซอร์ไว้แล้วไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_6_1vcx1 และเขียนโค้ดของวงจรนับขึ้น 15 บิตโดยเลือกใช้ความถี่ 2 Hz ใน การเคลื่บไหวของวงจรดีเบาเซอร์โดยอัตโนมัติทุกๆ $1/2\text{Hz} = 0.5$ วินาที จากนั้นนำไฟล์วงจรดีเบาเซอร์ชื่อ ch4vcx1_debounce และวงจรนับชื่อ ch4vcx1_c10_up_FSM มาทำเป็น Component เสร็จแล้วจะได้ดังรูปที่ L1.3

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_6_1vcx1 is
7     Port ( CLR,Din,OSC : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex4_6_1vcx1;
10
11 architecture Behavioral of ex4_6_1vcx1 is
12     component ch4vcx1_DEBOUNCER
13         Port ( OSC,D_in,CLR : in STD_LOGIC;
14                 D_out : out STD_LOGIC);
15     end component;
16     component ch4vcx1_c10_up_FSM
17         Port ( C,CLR : in STD_LOGIC;
18                 Q : out STD_LOGIC_VECTOR (3 downto 0));
19     end component;
20     signal C : STD_LOGIC;
21     signal F : STD_LOGIC_VECTOR (14 downto 0);
22 begin
23     -----15Bits Counter-----
24 process(OSC) --Output F(14)=1Hz,F(13)=2Hz,F(12)=4Hz,F(11)=8Hz
25     begin
26         if OSC'event and OSC='1' then F <= F + 1;
27         end if;
28     end process;
29     -----DEBOUNCER-----
30 DEBOUNCER : ch4vcx1_DEBOUNCER
31     port map(OSC=>OSC,D_in=>Din,CLR=>F(13),D_out=>C);
32     -----COUNTER : C10_UP_FSM-----
33 COUNTER_10UP : ch4vcx1_c10_up_FSM
34     port map(C=>C,CLR=>CLR,Q=>Q);
35 end Behavioral;

```

รูปที่ L1.3 โค้ดของวงจรนับ 10 แบบนับขึ้นที่รวมวงจรดีเบาเซอร์ไว้แล้ว

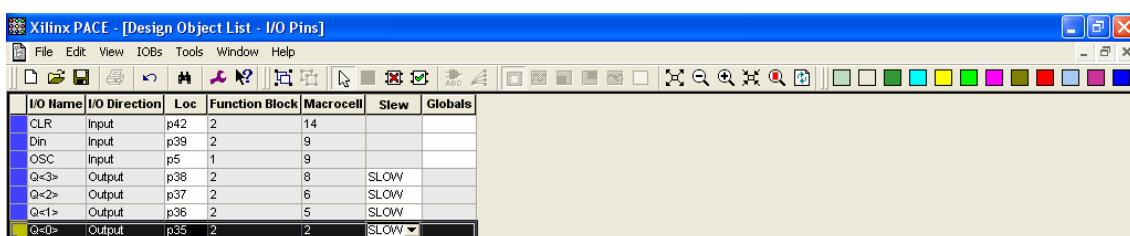
การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz ปุ่มกด PB1 และ PB3(Slide SW1) เป็นอินพุต และ LED1-LED4 เป็นเอาต์พุต ก่อร่างกาย

Din = PB1 = INPUT = p39 Q(3) = LED1 = OUTPUT = p38 Q(1) = LED3 = OUTPUT = p36

CLR= PB3(Slide SW1) = INPUT= p42 Q(2) = LED2 = OUTPUT = p37 Q(0) = LED4 = OUTPUT = p35

OSC= OSC = OUTPUT = p5

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



รูปที่ L1.4 Assign Package Pins

หลังจากโปรแกรมวงจรล็อกตัวยัง CPLD แล้วเลื่อน Slide SW1 ไปที่ตำแหน่ง ON (CLR = '0') ให้กดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกด PB1 ถ้าไว้แล้วให้คุณกดที่ LED1-LED4 ว่าติดสว่างโดยให้ออกເອຕີ່ພຸດເປັນໄປຄາມທຸກໆຮ້ອມ ຈາກນີ້ເລືອນ Slide SW1 ไปที่ตำแหน่ง OFF (CLR= '1') เพื่อເຄີຍເອຕີ່ພຸດແລ້ວ ON ແລ້ວทำการทดสอบໜ້າແລ້ວກິດກະທຸກຄົງ

2 สร้างวงจรนับ 10 แบบซิงໂຄຣນັສແບບນັບຂຶ້ນດ້ວຍ FPGA

2.1) สร้างวงจรนับ 10 แบบນັບຂຶ້ນດ້ວຍ FPGA ນັນຈະເໝືອກັນການກົດກະທຸກຄົງ CPLD ໃນຫຼື 1 ຖຸກປະການ ໂດຍຮາແກ້ໄຂໂຄດໃນຮູບປີ L1.2 ເພີ້ມ Entity ເປັນ ch4vf_c10_up_FSM ແລ້ວສ້າງໄຟລ໌ໄວ້ໃນ Project Location ຫຼື ch4v ກໍາເນັດ Project Name ແລະ Source File ຫຼື ch4vf_c10_up_FSM ບັນທຶກໄຟລ໌ແລ້ວ Check syntax ແລະ ທຳ Behavioral simulation ໂດຍໃຊ້ໄຟລ໌ຂີ້ວ່າ ch4vf_c10_up_FSM_tb1 ແລະ ພິຈານາພລວ່າເປັນໄປຄາມທຸກໆຮ້ອມ ໂດຍ

2.2) ເປັນໂຄດຂອງງານຈະກົດຂຶ້ນທີ່ຮ່ວມງານຈົດເບາເຊອຮ້ໄວ້ແລ້ວໄວ້ໃນ Project Location ຫຼື ch4v ກໍາເນັດ Project Name ແລະ Source File ຫຼື ex4_6_1vf ເປັນໂຄດກົດຈະກົດຂຶ້ນ 24 ບົດໂດຍເລືອກໃຊ້ຄວາມຄື 2.98 Hz ເພື່ອເຄີຍຮ່ວມງານຈົດເບາເຊອຮ້ໄດຍ ອັດໂນມັດຖຸກໆ 0.34 ວິນາທີ ນຳໄຟລ໌ຮ່ວມງານຈົດເບາເຊອຮ້ຂີ້ວ່າ ch4vf_debounce ແລະ ຈະກົດຂຶ້ນຂີ້ວ່າ ch4vf_c10_up ມາທຳເປັນ Component ເສົ່ງແລ້ວຈະໄດ້ດັງຮູບປີ L2.1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_6_1vf is
7     Port ( CLR,Din,OSC : in STD_LOGIC;
8             Q : out STD_LOGIC_VECTOR (3 downto 0));
9 end ex4_6_1vf;
10
11 architecture Behavioral of ex4_6_1vf is
12     component ch4vf_DEBOUNCER
13         Port ( OSC,D_in,CLR : in STD_LOGIC;
14                 D_out : out STD_LOGIC);
15     end component;
16     component ch4vf_c10_up_FSM
17         Port ( C,CLR : in STD_LOGIC;
18                 Q : out STD_LOGIC_VECTOR (3 downto 0));
19     end component;
20     signal C : STD_LOGIC;
21     signal F : STD_LOGIC_VECTOR (23 downto 0);
22 begin
23     -----24Bits Counter-----
24 process(OSC) --Output F(23)=1.49Hz,F(22)=2.98Hz,F(21)=5.96Hz
25     begin
26         if OSC'event and OSC='1' then F <= F + 1;
27         end if;
28     end process;
29     -----DEBOUNCER-----
30 DEBOUNCER : ch4vf_DEBOUNCER
31     port map(OSC=>OSC,D_in=>Din,CLR=>F (22),D_out=>C);
32     -----COUNTER : C10_UP-----
33 COUNTER_10UP : ch4vf_c10_up_FSM
34     port map(C=>C,CLR=>CLR,Q=>Q);
35 end Behavioral;

```

ຮູບປີ L2.1 ໂດຍຂອງງານຈະກົດຂຶ້ນທີ່ຮ່ວມງານຈົດເບາເຊອຮ້ໄວ້ແລ້ວ

ການກໍາເນັດຂາສົ່ງຄູາມຕ່າງໆ ຈະໃຊ້ OSC = 25 MHz ປຸ່ມກົດ PB1 ແລະ Dip SW1 ເປັນອິນພຸດ ມີ LED L0-L3 ເປັນເອຕີ່ພຸດ

Din = PB1 = INPUT = p44 Q(3) = L3 = OUTPUT = p76 Q(1) = L1 = OUTPUT = p77

CLR= Dip SW1 = INPUT = p52 Q(2) = L2 = OUTPUT = p69 Q(0) = L0 = OUTPUT = p70

OSC = OSC= INPUT= p127

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLR	Input	p52	BANK LVCMOS33	N/A	3.30						Unknown		
Din	Input	p44	BANK LVCMOS33	N/A	3.30						Unknown		
OSC	Input	p127	BANK LVCMOS33	N/A	3.30						Unknown		
Q<0>	Output	p70	BANK LVCMOS33	N/A	3.30				SLOW		Unknown		
Q<1>	Output	p77	BANK LVCMOS33	N/A	3.30				SLOW		Unknown		
Q<2>	Output	p69	BANK LVCMOS33	N/A	3.30				SLOW		Unknown		
Q<3>	Output	p76	BANK LVCMOS33	N/A	3.30				SLOW		Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป FPGA แล้วเลื่อน Dip SW1 ไปที่ตำแหน่ง ON (CLR = ‘0’) ให้กดปุ่ม PB1 ไปเรื่อยๆ ลับกับการกด PB1 ถ้าไม่แล้วให้คุณตั้ง LED L0-L3 ว่าคิดสว่างโดยให้ออกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นเลื่อน Dip SW1 ไปที่ตำแหน่ง OFF (CLR= ‘1’) เพื่อเคลียร์เอาต์พุตแล้ว ON แล้วทำการทดสอบซ้ำและบันทึกผลการทดลอง

แบบฝึกหัด

- 1) จาก State diagram และ State table ในรูปที่ L1.1 ให้นักศึกษาเขียนโค้ดวงจรนับ 10 แบบชิงโกรนัสแบบนับขึ้น โดยใช้ชีวิช FSM with 2 processes และ FSM with 3 processes
- 2) ให้นักศึกษาเขียน State diagram และ State table และ โค้ดของวงจรนับแบบชิงโกรนัสที่เป็นวงจรนับ 10 แบบนับขึ้น โดยใช้ชีวิช FSM with 2 processes และ FSM with 3 processes ซึ่งวงจรนี้จะนับเมื่อ Clock enable input หรือ CE = ‘1’ และจะ Clear ค่าเอาต์พุต Q = “0000” เมื่อ CLR = ‘1’ และเมื่อวงจรนับถึง 9 คือ “1001” แล้ว Clock enable Output หรือ CEO = ‘1’ (คำแนะนำในการออกแบบวงจรนี้ ให้หลักการทำองเดียวกันกับดัวอย่างที่ 4.2)

การทดลองที่ 4.6.2 วงจรตรวจจับลำดับแบบ Moor-type FSM

วัตถุประสงค์

- 1) เพื่อออกแบบวงจรตรวจจับลำดับ (Sequence detector) แบบ Moor-type FSM with 2 processes
- 2) เพื่อสร้างวงจรตรวจจับลำดับโดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรตรวจจับลำดับด้วย CPLD

- 1.1) สร้างวงจรตรวจจับลำดับเป็น Moor-type FSM ตามตัวอย่างที่ 4.4 โดยที่วงจรจะให้ออกตัว Z = '1' ก็ต่อเมื่อตรวจจับอินพุต X = '1' จำนวน 3 ตัวติดต่อกัน นำโค้ดในรูปที่ 4.82a มาเขียนใหม่ (แก้ชื่อ Entity เป็น ch4vcxl_Sequencer) โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vcxl_Sequencer ทำการบันทึกไฟล์และ Check syntax ทำ Behavioral simulation โดยใช้ Source File ชื่อ ch4vcxl_tb1 และให้คุณตรวจสอบว่าเป็นตามทฤษฎีหรือไม่
- 1.2) นำโค้ดวงจรโ้มโนสเตเบิล (One shot) ในรูปที่ 4.77 ซึ่งเป็นโ้มโนสเตเบิลประสาทเชิง串มาเขียนโค้ดใหม่ (แก้ชื่อ Entity เป็น ch4vcxl_ONE_SHOT_FSM) เพื่อเก็บไว้ทำ Component โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v และกำหนด Project Name และ Source File เป็นชื่อ ch4vcxl_ONE_SHOT_FSM ทำการบันทึกไฟล์และ Check syntax
- 1.3) เขียนโค้ดวงจรตรวจจับลำดับที่รวมวงจรโ้มโนสเตเบิลไว้แล้ว (เพื่อ Debouncing ตัวข้อมูล Clock) ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_6_2vcxl และเขียนโค้ดวงจรนับ 8 บิตเพื่อสร้างความถี่ 128 Hz < 150 Hz ป้อนให้กับวงจรโ้มโนสเตเบิล จากนั้นนำไฟล์ชื่อ ch4vcxl_ONE_SHOT_FSM และชื่อ ch4vcxl_Sequencer มาทำเป็น Component และจะได้โค้ด Moor-type FSM with 2 processes ของวงจรตรวจจับลำดับที่รวมวงจรโ้มโนสเตเบิลดังรูปที่ L1.1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_6_2vcxl is
7     Port ( CLK,CLR,OSC,X : in STD_LOGIC;
8             Q : out STD_LOGIC);
9 end ex4_6_2vcxl;
10
11 architecture Behavioral of ex4_6_2vcxl is
12     component ch4vcxl_ONE_SHOT_FSM is
13         Port ( X,Clock : in STD_LOGIC;
14                 Z : out STD_LOGIC);
15     end component;
16     component ch4vcxl_Sequencer  is
17         Port ( X,Clock,Reset : in STD_LOGIC;
18                 Z : out STD_LOGIC);
19     end component;
20     signal C : STD_LOGIC;
21     signal F : STD_LOGIC_VECTOR (7 downto 0):= (others => '0');
22 begin
23     -----8Bits Counter-----
24     process(OSC) --OSC=32.768kHz, Output F(7)=128Hz < 150Hz
25     begin
26         if OSC'event and OSC='1' then F <= F + 1;
27         end if;
28     end process;

```

(ต่อ)

```

29 -----ONE_SHOT-----
30 ONE_SHOT : ch4vcx1_ONE_SHOT_FSM
31   port map( X => CLK,Clock => F(7),Z => C);
32 -----COUNTER : C10_UP_FSM-----
33 Sequencer : ch4vcx1_Sequencer
34   port map(X => X,Clock => C,Reset => CLR,Z => Q);
35
36 end Behavioral;

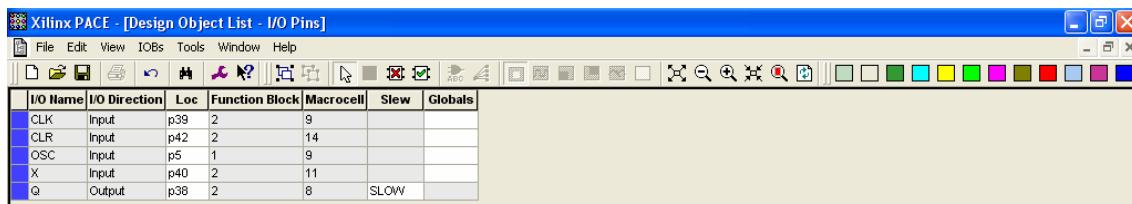
```

ຮູບທີ L1.1 ໂຄສໍາ Moor-type FSM with 2 processes ຂອງຈົດຕະວັນລຳດັບທີ່ຮ່ວມງານໂນໂນສເຕເບີດ

ການກຳທຳນັດຂາສັ້ນຍານຕ່າງໆ ຈະໃຊ້ອອສື່ຈຸລາເຕອຣ໌ OSC = 32.768 kHz ແລະ ບຸ້ມກົດ PB1, PB2 ແລະ PB3(Slide SW1) ເປັນ ອິນພຸດ ມີ LED4 ເປັນເອົາຕົ້ມ ກລ່າວກືອ

CLK = PB1 = INPUT = p39 CLR = PB3(Slide SW1) = INPUT = p42 Z = LED4 = OUTPUT = p38
X = PB2 = INPUT = p40 OSC = OSC = INPUT = p5

ໂດຍພິມຟີ່ໃນ Assign Package Pins ສຽງປັດຈິງນີ້



ຮູບທີ L1.2 Assign Package Pins

ໜັງຈາກໂປຣແກຣມງານຮລົງໝຶກ CPLD ແລ້ວເລືອນ Slide SW1 ໄປທີ່ຕຳແໜ່ງ ON (CLR = '0') ແລ້ວໃຫ້ກົດລອງກົບປຸ່ມ PB1 ຊ້າງ ທີ່ລະ 1 ຄວັງໄປຈົນກະທຳໆ LED4 ຕິດສ່ວາງ ຈາກນັ້ນກົດ PB2 (X = '0') ດັ່ງໄວ້ແລ້ວກົບປຸ່ມ PB1 1 ຄວັງແລ້ວໃຫ້ຄູພລື່ໆ LED4 ປລ່ອຍື່ນກົດ PB2 (X = '1') ແລ້ວກົບປຸ່ມ PB1 3 ຄວັງແລ້ວໃຫ້ຄູພລື່ໆ LED4 ອີກຮັງ ແລ້ວກົບປຸ່ມ PB2 ຕ່ອໄປ 3 ຄວັງແລ້ວໃຫ້ຄູພລື່ໆ LED4 ວ່າ LED4 ດັບຕາມຈົງຫວະກາດ PB2 ອີກຮັງ ອີກຮັງ ຈາກນັ້ນເລືອນ Slide SW1 ໄປທີ່ຕຳແໜ່ງ OFF ແລ້ວໃຫ້ຄູພລື່ໆ LED4 ເລືອນ Slide SW1 ໄປທີ່ຕຳແໜ່ງ ON ອີກຮັງແລ້ວກົບປຸ່ມ PB1 ຕ່ອໄປ 3 ຄວັງ ແລ້ວໃຫ້ພິຈານາຄູ່ໆ LED4 ວ່າເອົາຕົ້ມທີ່ໄດ້ເປັນຕາມທຖນຸ້ງຫຼືໄວ້ໄມ່

2 ສ້າງງານຈົດຕະວັນລຳດັບທີ່ວ່າງ

2.1) ສ້າງງານຈົດຕະວັນລຳດັບເປັນ Moor-type FSM ດາມຕ້ວອຍ່າງທີ່ 4.4 ໂດຍທີ່ວົງຈະໄຫ້ເອົາຕົ້ມ Z = '1' ກີ່ຕ່ອມື່ອຈົດຕະວັນລຳດັບອິນພຸດ X = '1' ຈຳນວນ 3 ຕົວຕິດຕ່ອກັນ ນຳໄກສໍາໃນຮູບທີ່ 4.82a) ນາເຈີນໃໝ່ (ແກ້ໜ້ອ Entity ເປັນ ch4vf_Sequencer) ໂດຍສ້າງໄຟລ໌ໄວ້ໃນ Project Location ຂໍ້ອ ch4v ກໍາທັນ Project Name ແລະ Source File ຂໍ້ອ ch4vf_Sequencer ທຳກຳກັບກຳທັນທີ່ໄຟລ໌ແລະ Check syntax ທຳ Behavioral simulation ໂດຍໃຊ້ Source File ຂໍ້ອ ch4vf_Sequencer_tb1 ແລ້ວໃຫ້ຄູພລື່ໆເປັນຕາມທຖນຸ້ງຫຼືໄວ້ໄມ່

2.2) ນຳໄກສໍາຈົດຕະວັນໂນໂນສເຕເບີດ (One shot) ໃນຮູບທີ່ 4.77 ຜົ່ງເປັນໄມ້ໂນໂນສເຕເບີດປະສົງສົງມາເຈີນໄກສໍາໃໝ່ (ແກ້ໜ້ອ Entity ເປັນ ch4vf_ONE_SHOT_FSM) ເພື່ອເກີ່ມໄວ້ທຳ Component ໂດຍສ້າງໄຟລ໌ໄວ້ໃນ Project Location ຂໍ້ອ ch4v ແລ້ວກຳທັນ Project Name ແລະ Source File ເປັນຂໍ້ອ ch4vf_ONE_SHOT_FSM ທຳກຳກັບກຳທັນທີ່ໄຟລ໌ແລະ Check syntax

2.3) ເນື້ອໄກສໍາຈົດຕະວັນລຳດັບທີ່ຮ່ວມງານໂນໂນສເຕເບີດໄວ້ແລ້ວ (ເພື່ອ Debouncing ສັ້ນຍານ Clock) ໄວໃນ Project Location ຂໍ້ອ ch4v ກໍາທັນ Project Name ແລະ Source File ຂໍ້ອ ex4_6_2vf ແລ້ວເຈີນໄກສໍາຈົດຕະວັນ 18 ບົດເພື່ອສ້າງຄວາມຄື $95.37 \text{ Hz} < 150 \text{ Hz}$ ປຶ້ນໄກ້ກັບງານໂນໂນສເຕເບີດ ຈາກນັ້ນນຳໄຟລ໌ຂໍ້ອ ch4vf_ONE_SHOT_FSM ແລະ ຂໍ້ອ ch4vf_Sequencer ມາທຳເປັນ Component ເສົ່ງແລ້ວຈະໄດ້ໄກສໍາ Moor-type FSM with 2 processes ຂອງຈົດຕະວັນລຳດັບທີ່ຮ່ວມງານໂນໂນສເຕເບີດດັ່ງຮູບທີ່ L2.1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_6_2vf is
7     Port ( CLK,CLR,OSC,X : in STD_LOGIC;
8             Q : out STD_LOGIC);
9 end ex4_6_2vf;
10
11 architecture Behavioral of ex4_6_2vf is
12     component ch4vf_ONE_SHOT_FSM is
13         Port ( X,Clock : in STD_LOGIC;
14                 Z : out STD_LOGIC);
15     end component;
16     component ch4vf_Sequencer is
17         Port ( X,Clock,Reset : in STD_LOGIC;
18                 Z : out STD_LOGIC);
19     end component;
20     signal C : STD_LOGIC;
21     signal F : STD_LOGIC_VECTOR (17 downto 0):= (others => '0');
22 begin
23     -----18Bits Counter-----
24 process(OSC) --OSC=25MHz, Output F(17)=95.37Hz < 150Hz
25     begin
26         if OSC'event and OSC='1' then F <= F + 1;
27     end if;
28 end process;
29     -----ONE_SHOT-----
30 ONE_SHOT : ch4vf_ONE_SHOT_FSM
31     port map( X => CLK,Clock => F(17),Z => C );
32     -----COUNTER : C10_UP_FSM-----
33 Sequencer : ch4vf_Sequencer
34     port map(X => X,Clock => C,Reset => CLR,Z => Q );
35
36 end Behavioral;

```

รูปที่ L2.1 ไฟล์ Moor-type FSM with 2 processes ของวงจรตรวจขั้นลำดับที่รวมวงจรไม้ในสเตมเบิล

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25 MHz และปุ่มกด PB1, PB2 และ Dip SW1 เป็นอินพุต และมี LED L3 เป็นเอาต์พุต กล่าวคือ

CLK = PB1 = INPUT = p44 CLR = Dip SW1 = INPUT = p52 Z = LED L3 = OUTPUT = p76

X = PB2 = INPUT = p46 OSC = OSC = INPUT = p127

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLK	Input	p44	BANK	LVC MOS33	N/A	3.30					Unknown		
CLR	Input	p52	BANK	LVC MOS33	N/A	3.30					Unknown		
OSC	Input	p127	BANK	LVC MOS33	N/A	3.30					Unknown		
Q	Output	p76	BANK	LVC MOS33	N/A	3.30			SLOW		Unknown		
X	Input	p46	BANK	LVC MOS33	N/A	3.30					Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป FPGA แล้วเลื่อน Dip SW1 ไปที่ตำแหน่ง ON (CLR = '0') แล้วให้ทดลองกดปุ่ม PB1 ช้าๆ ทีละ 1 ครั้ง ไปจนกระทั่ง LED L3 ติดสว่าง จากนั้นกด PB2 (X = '0') ค้างไว้แล้วกดปุ่ม PB1 1 ครั้งแล้วให้คูplot ที่ LED L3 ปล่อยปุ่มกด PB2 (X = '1') แล้วกดปุ่ม PB1 3 ครั้งแล้วให้คูplot ที่ LED L3 อีกครั้ง แล้วกดปุ่ม PB2 ต่อไป 3 ครั้งแล้วให้คูplot ที่ LED L3 ว่า LED L3 ดับตามจังหวะการกด PB2 หรือไม่ จากนั้นเลื่อน Dip SW1 ไปที่ตำแหน่ง OFF แล้วคูplot LED L3 เลื่อน Dip SW1 ไปที่ตำแหน่ง ON อีกครั้งแล้วกดปุ่ม PB1 ต่อไป 3 ครั้ง แล้วให้คูplot LED L3 ว่าเอาต์พุตที่ได้เป็นตามทฤษฎีหรือไม่

การทดลองที่ 4.6.3 วงจรตรวจจับลำดับแบบ Mealy-type FSM

วัตถุประสงค์

- 1) เพื่อออกแบบวงจรตรวจจับลำดับ (Sequence detector) แบบ Mealy-type FSM with 2 processes
- 2) เพื่อสร้างวงจรตรวจจับลำดับโดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรตรวจจับลำดับด้วย CPLD

- 1.1) สร้างวงจรตรวจจับลำดับเป็น Mealy-type FSM ตามตัวอย่างที่ 4.5 โดยที่วงจรจะให้ออกพุต $Z = '1'$ ก็ต่อเมื่อตรวจจับอินพุต $X = '1'$ จำนวน 3 ตัวติดต่อกัน นำโค้ดในรูปที่ 4.83a มาเขียนใหม่ (แก้ชื่อ Entity เป็น ch4vcx1_SQ_MEALY2P) โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vcx1_SQ_MEALY2P บันทึกไฟล์และ Check syntax ทำ Behavioral simulation โดยใช้ Source File ชื่อ ch4vcx1_SQ_MEALY2P_tb1 และให้คุณลักษณะเป็นตามทฤษฎีหรือไม่
- 1.2) เขียนโค้ดวงจรตรวจจับลำดับที่รวมวงจรโรมโนนสเตเบิลไว้แล้ว (เพื่อ Debouncing สัญญาณ Clock) ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_6_3vcx1 และเพิ่ยงโค้ดวงจรนับ 8 บิตเพื่อสร้างความถี่ 128 Hz < 150 Hz ป้อนให้กับวงจรโรมโนนสเตเบิล นำไฟล์ชื่อ ch4vcx1_ONE_SHOT_FSM และชื่อ ch4vcx1_SQ_MEALY2P มาทำเป็น Component เตรียมเลือก "ได้โค้ด Mealy-type FSM with 2 processes ของวงจรตรวจจับลำดับที่รวมวงจรโรมโนนสเตเบิลดังรูปที่ L1.1"

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_6_3vcx1 is
7   Port ( CLK,CLR,OSC,X : in STD_LOGIC;
8         Q : out STD_LOGIC);
9 end ex4_6_3vcx1;
10
11 architecture Behavioral of ex4_6_3vcx1 is
12   component ch4vcx1_ONE_SHOT_FSM is
13     Port ( X,Clock : in STD_LOGIC;
14            Z : out STD_LOGIC);
15   end component;
16   component ch4vcx1_SQ_MEALY2P is
17     Port ( X,Clock,Reset : in STD_LOGIC;
18            Z : out STD_LOGIC);
19   end component;
20   signal C : STD_LOGIC;
21   signal F : STD_LOGIC_VECTOR (7 downto 0):= (others => '0');
22 begin
23   -----8Bits Counter-----
24   process(OSC) --OSC=32.768kHz, Output F(7)=128Hz < 150Hz
25   begin
26     if OSC'event and OSC='1' then F <= F + 1;
27     end if;
28   end process;
29   -----ONE_SHOT-----
30   ONE_SHOT : ch4vcx1_ONE_SHOT_FSM
31   port map( X => CLK,Clock => F(7),Z => C);
32   -----COUNTER : C10_UP_FSM-----
33   Sequencer : ch4vcx1_SQ_MEALY2P
34   port map(X => X,Clock => C,Reset => CLR,Z => Q);
35
36 end Behavioral;

```

รูปที่ L1.1 โค้ด Mealy-type FSM with 2 processes ของวงจรตรวจจับลำดับที่รวมวงจรโรมโนนสเตเบิล

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1, PB2 และ PB3(Slide SW1) เป็นอินพุต มี LED4 เป็นเอาต์พุต กล่าวคือ

$$\begin{aligned} \text{CLK} &= \text{PB1} = \text{INPUT} = \text{p39} & \text{CLR} &= \text{PB3}(\text{Slide SW1}) = \text{INPUT} = \text{p42} & Z &= \text{LED4} = \text{OUTPUT} = \text{p38} \\ X &= \text{PB2} = \text{INPUT} = \text{p40} & \text{OSC} &= \text{OSC} = \text{INPUT} = \text{p5} \end{aligned}$$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
CLK	Input	p39	2	9		
CLR	Input	p42	2	14		
OSC	Input	p5	1	9		
X	Input	p40	2	11		
Q	Output	p38	2	8	SLOW	

รูปที่ L1.2 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป CPLD แล้วเลื่อน Slide SW1 ไปที่ตำแหน่ง ON (CLR = '0') แล้วให้กดลงกดปุ่ม PB1 ช้าๆ ทีละ 1 ครั้ง ไปจนกระทั่ง LED4 ติดสว่าง จากนั้นกด PB2 (X = '0') ค้างไว้แล้วกดปุ่ม PB1 1 ครั้งแล้วให้คุณลองที่ LED4 ปล่อยปุ่มกด PB2 (X = '1') แล้วกดปุ่ม PB1 2 ครั้งแล้วให้คุณลองที่ LED4 อีกครั้ง แล้วกดปุ่ม PB2 ต่อไป 3 ครั้งแล้วให้คุณลองที่ LED4 ว่า LED4 ดับตามจังหวะการกด PB2 หรือไม่ จากนั้นเลื่อน Slide SW1 ไปที่ตำแหน่ง OFF แล้วให้คุณลองที่ LED4 เลื่อน Slide SW1 ไปที่ตำแหน่ง ON อีกครั้งแล้วกดปุ่ม PB1 ต่อไป 2 ครั้งแล้วให้คุณลองที่ LED4 ว่าเป็นไปตามทฤษฎีหรือไม่

2 สร้างวงจรตรวจจับคำดับด้วย FPGA

- 1.1) สร้างวงจรตรวจจับคำดับเป็น Mealy-type FSM ตามตัวอย่างที่ 4.5 โดยที่วงจรจะให้อาต์พุต $Z = '1'$ ก็ต่อเมื่อตรวจจับอินพุต $X = '1'$ จำนวน 3 ตัวติดต่อกัน นำโค้ดในรูปที่ 4.83a มาเขียนใหม่ (แก้ชื่อ Entity เป็น ch4vf_SQ_MEALY2P) โดยสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vf_SQ_MEALY2P บันทึกไฟล์และ Check syntax ทำ Behavioral simulation โดยใช้ Source File ชื่อ ch4vf_SQ_MEALY2P_tb1 แล้วให้คุณว่าเป็นตามทฤษฎีหรือไม่
- 1.2) เขียนโค้ดวงจรตรวจจับคำดับที่รวมวงจรโมโนโนสเตเบิลไว้แล้ว (เพื่อ Debouncing สัญญาณ Clock) ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_6_3vf และเขียนโค้ดวงจรนับ 18 บิตเพื่อสร้างความถี่ $95.37 \text{ Hz} < 150 \text{ Hz}$ ป้อนให้กับวงจรโมโนโนสเตเบิล นำไฟล์ชื่อ ch4vf_ONE_SHOT_FSM และชื่อ ch4vf_SQ_MEALY2P มาทำเป็น Component เสร็จแล้วจะได้โค้ด Mealy-type FSM with 2 processes ของวงจรตรวจจับคำดับที่รวมวงจรโมโนโนสเตเบิลดังรูปที่ L2.1

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_6_3vf is
7   Port ( CLK,CLR,OSC,X : in STD_LOGIC;
8         Q : out STD_LOGIC);
9 end ex4_6_3vf;
10
11 architecture Behavioral of ex4_6_3vf is
12   component ch4vf_ONE_SHOT_FSM is
13     Port ( X,Clock : in STD_LOGIC;
14            Z : out STD_LOGIC);
15   end component;
16   component ch4vf_SQ_MEALY2P is
17     Port ( X,Clock,Reset : in STD_LOGIC;
18            Z : out STD_LOGIC);
19   end component;

```

(ต่อ)

```

20      signal C : STD_LOGIC;
21      signal F : STD_LOGIC_VECTOR (17 downto 0) := (others => '0');
22 begin
23 -----18Bits Counter-----
24 process(OSC) --OSC=25kHz, Output F(17)=95.37Hz < 150Hz
25 begin
26     if OSC'event and OSC='1' then F <= F + 1;
27     end if;
28 end process;
29 -----ONE_SHOT-----
30 ONE_SHOT : ch4vf_ONE_SHOT_FSM
31     port map( X => CLK,Clock => F(17),Z => C);
32 -----COUNTER : C10_UP_FSM-----
33 Sequencer : ch4vf_SO_MEALY2P
34     port map(X => X,Clock => C,Reset => CLR,Z => Q);
35
36 end Behavioral;

```

รูปที่ L2.1 โค้ด Mealy-type FSM with 2 processes ของวงจรตรวจขับลำดับที่รวมวงจร โนมโนมสเตเบิล

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25 MHz และปุ่มกด PB1, PB2 และ Dip SW1 เป็นอินพุต และมี LED L3 เป็นเอาต์พุต กล่าวว่าคือ

CLK = PB1 = INPUT = p44	CLR = Dip SW1 = INPUT = p52	Z = LED L3 = OUTPUT = p76
X = PB2 = INPUT = p46	OSC = OSC = INPUT = p127	

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLK	Input	p44	BANK LVC MOS33	N/A	3.30					Unknown			
CLR	Input	p52	BANK LVC MOS33	N/A	3.30					Unknown			
OSC	Input	p127	BANK LVC MOS33	N/A	3.30					Unknown			
Q	Output	p76	BANK LVC MOS33	N/A	3.30			SLOW		Unknown			
X	Input	p46	BANK LVC MOS33	N/A	3.30					Unknown			

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป FPGA แล้วเลื่อน Dip SW1 ไปที่ตำแหน่ง ON (CLR = '0') แล้วให้ทดลองกดปุ่ม PB1 ช้าๆ ทีละ 1 ครั้ง ไปจนกระทั่ง LED L3 ติดสว่าง จากนั้นกด PB2 (X = '0') ถ้าງไว้แล้วกดปุ่ม PB1 1 ครั้งแล้วให้คูล์เพลที่ LED L3 ปล่อยปุ่มกด PB2 (X = '1') แล้วกดปุ่ม PB1 2 ครั้งแล้วให้คูล์เพลที่ LED L3 อีกครั้ง แล้วกดปุ่ม PB2 ต่อไป 3 ครั้งแล้วให้คูล์เพลที่ LED L3 ว่า LED L3 ดับตามจังหวะการกด PB2 หรือไม่ จากนั้นเลื่อน Dip SW1 ไปที่ตำแหน่ง OFF แล้วให้คูล์เพลที่ LED L3 เลื่อน Dip SW1 ไปที่ตำแหน่ง ON อีกครั้งแล้วกดปุ่ม PB1 ต่อไป 2 ครั้งแล้วให้คูล์เพลที่ LED L3 ว่าเป็นไปตามทฤษฎีหรือไม่

แบบฝึกหัด

- กรณีของ Mealy-type FSM ใน การทดลองที่ 4.6.3 นั้น เมื่อกดปุ่ม PB2 แล้วทำให้อเอต์พุต LED ดับตามจังหวะการกด PB2 ทุกครั้ง ถ้า่ว่า นักศึกษาจะแก้ปัญหาเอาต์พุตที่ผิดพลาดของ Mealy-type FSM ในขณะที่อินพุตเกิดการเปลี่ยนแปลงложิกน้อยๆ ไง
- ให้นักศึกษาเขียนโค้ดของวงจร Mealy-type FSM ใน การทดลองที่ 4.6.3 ใหม่เป็น Mealy-type FSM with Registered output (Mealy-type FSM with 1 process) เสร็จแล้วให้ทดลองโดยเปรียบเทียบกับการทดลองที่ 4.6.2 ว่าให้ผลลัพธ์เหมือนกันหรือไม่

การทดลองที่ 4.6.4 หน่วยควบคุมเครื่องขายน้ำอัดลมอัตโนมัติ

วัตถุประสงค์

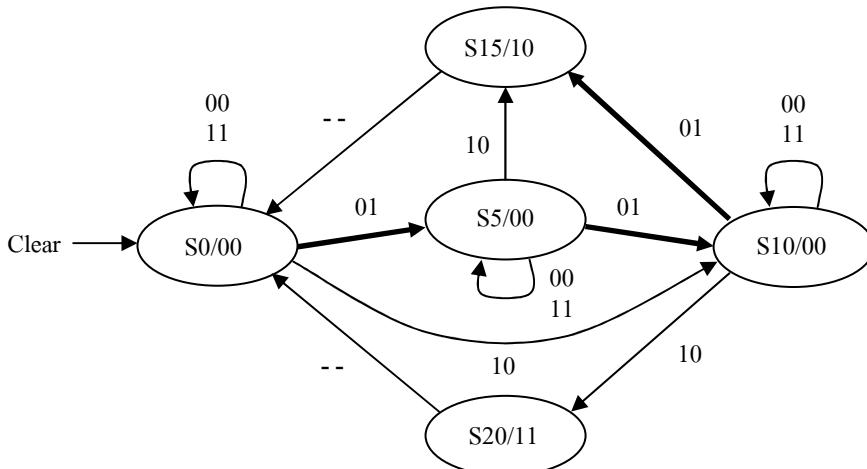
- 1) เพื่อออกแบบหน่วยควบคุม (Control unit) เครื่องขายน้ำอัดลมอัตโนมัติ (Vending machine)
- 2) เพื่อสร้างหน่วยควบคุมเครื่องขายน้ำอัดลมอัตโนมัติโดยเขียนโค้ด VHDL และโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรหน่วยควบคุมเครื่องขายน้ำอัดลมอัตโนมัติด้วย CPLD

การทดลองนี้ เราสร้างเฉพาะหน่วยควบคุมของเครื่องขายน้ำอัดลมอัตโนมัติ (อ่าย่างง่าย) ซึ่งรับสัญญาณอินพุตจากวงจรรับเรียบวนนาค 5 บาทและ 10 บาทที่จะต้องหยุดหรือยกกระง吝ะหรือยกเท่านั้น เมื่อหยุดหรือยกครบ 15 บาทเครื่องจะปล่อยน้ำอัดลมออกมา 1 กระป๋องพร้อมเงินทอนอีก 5 บาท เราสามารถเขียน State diagram และ State table ได้ดังรูปที่ L1.1 และรูปที่ L1.2 ตามลำดับ กำหนดให้ X และ Z เป็นอินพุตและเอาต์พุตแบบเรียร์ โดย X(1) และ X(0) เป็นอินพุตที่สามารถหยุดหรือยก 10 บาทและ 5 บาทได้ครั้งละหรือยก ในขณะที่ Z(1) และ Z(0) จะเป็นเอาต์พุตของวงจรปล่อยน้ำอัดลมและวงจรทอนเงิน 5 บาทตามลำดับ โดยที่ “-” คือ Don't care



รูปที่ L1.1 State diagram (Moor-type FSM) วงจรหน่วยควบคุมของเครื่องขายน้ำอัดลมอัตโนมัติ (อ่าย่างง่าย)

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 L1.2 State table วงจรหน่วยควบคุมของเครื่องขายน้ำอัดลมอัตโนมัติ (อ่ายางง่าย)

หลักคิดในการเขียน State diagram เป็นดังนี้

- ถ้า State S0 เป็น State เริ่มต้น เมื่อไม่มีการหยอดเหรียญ อินพุต X = “00” ให้ State อยู่ที่ State S0 เช่นเดิม
 - ที่ State S0 เมื่อมีการหยอดเหรียญ 5 บาท อินพุต X = “01” แล้วทำให้วงจรเปลี่ยน State เป็น State S5
 - ที่ State S5 เมื่อมีการหยอดเหรียญ 5 บาท อินพุต X = “01” แล้วทำให้วงจรเปลี่ยน State เป็น State S10
 - ที่ State S10 เมื่อมีการหยอดเหรียญ 5 บาท อินพุต X = “01” แล้วทำให้วงจรเปลี่ยน State เป็น State S15 จากนั้น เรายังเขียน State diagram ทั้ง 4 State คือ S0, S5, S10 และ S15 ขึ้นมาก่อน ซึ่งที่ State S15 นี้เราตั้งชื่อวงจรจะ ปล่อยน้ำอัดลมออกมา เสร็จแล้วจะกลับไปที่ State S0
 - ที่ State S0 เมื่อมีการหยอดเหรียญ 10 บาท อินพุต X = “10” แล้วทำให้วงจรเปลี่ยน State เป็น State S10
 - ที่ State S10 เมื่อมีการหยอดเหรียญ 10 บาท อินพุต X = “10” แล้วทำให้วงจรเปลี่ยน State เป็น State S20 จากนั้นให้เราเขียน State 20 เพิ่มเข้าไปใน State diagram ซึ่งที่ State S20 นี้เราตั้งชื่อวงจรจะ ปล่อยน้ำอัดลม ออกมากลับและทอนเงิน 5 บาท เสร็จแล้วจะกลับไปที่ State S0
 - ที่ State S5 เมื่อมีการหยอดเหรียญ 10 บาท อินพุต X = “10” แล้วทำให้วงจรเปลี่ยน State เป็น State S15 ซึ่งที่ State S15 นี้เราตั้งชื่อวงจรจะ ปล่อยน้ำอัดลมออกมา เสร็จแล้วจะกลับไปที่ State S0
 - เขียนส៊านที่มีลูกศรชี้ไปยัง State ตัดไปพร้อมกับกำหนดค่าอินพุต จากนั้นจึงค่อยๆ เขียน State diagram และ State table จนแล้วเสร็จ วงจรนี้เรายังหยอดเหรียญพร้อมกันไม่ได้ อินพุต X = “11” จึงไม่มี ดังนั้น State diagram และ State diagram ของผู้ออกแบบแต่ละคนอาจต่างกันบ้างเล็กน้อยเพราการตั้งสมมุติฐานอาจจะ ไม่เหมือนกัน

ขั้นตอนการเปลี่ยนโถคัดของวงจรหน่วยความจำของเครื่องขยายบันทึกคลุมอัตโนมัติและการทดสอบเป็นดังนี้

1.1) เขียนโค้ด VHDL วงจรหน่วยความจำของเครื่องขายน้ำอัดลมอัตโนมัติเป็น Moor-type FSM with 2 processes ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vcx1_vending ดังรูปที่ L1.3 บันทึกไฟล์และ Check syntax จากนั้นทำ Behavioral simulation โดยใช้ Source File ชื่อ ch4vcx1_vending_tb1 และให้คุณจำลองการทำงานดังรูปที่ L1.4 ว่า เป็นไปตามทฤษฎีหรือไม่

```
-----Control unit of vending machine-----
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ch4vcx1_vending is
6     Port ( C,Clear : in STD_LOGIC;
7             X : in STD_LOGIC_VECTOR (1 downto 0);
8             Z : out STD_LOGIC_VECTOR (1 downto 0));
9 end ch4vcx1_vending;
10
11 architecture Behavioral of ch4vcx1_vending is
12     type State_type is (S0,S5,S10,S15,S20);
13     signal State : State_type ;
14 begin
15
16 process1: process (C,Clear)
17 begin
18     if (Clear ='1') then
19         State <= S0;
20     elsif (C'event and C='1') then
21         case State is
22             when S0 =>
23                 if X = "00" then
24                     State <= S0;
```

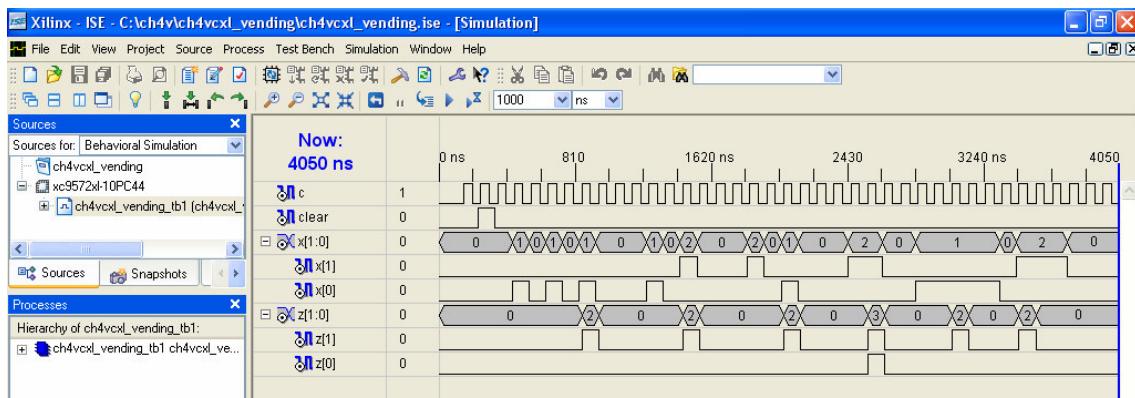
(୪୦)

```

28         elsif X = "01" then
29             State <= S5;
30         elsif X = "10" then
31             State <= S10;
32         else
33             State <= S0;
34         end if;
35     when S5 =>
36         if X = "00" then
37             State <= S5;
38         elsif X = "01" then
39             State <= S10;
40         elsif X = "10" then
41             State <= S15;
42         else
43             State <= S5;
44         end if;
45     when S10 =>
46         if X = "00" then
47             State <= S10;
48         elsif X = "01" then
49             State <= S15;
50         elsif X = "10" then
51             State <= S20;
52         else
53             State <= S10;
54         end if;
55     when S15 =>
56         if X = "00" then
57             State <= S0;
58         elsif X = "01" then
59             State <= S0;
60         elsif X = "10" then
61             State <= S0;
62         else
63             State <= S0;
64         end if;
65     when S20 =>
66         if X = "00" then
67             State <= S0;
68         elsif X = "01" then
69             State <= S0;
70         elsif X = "10" then
71             State <= S0;
72         else
73             State <= S0;
74         end if;
75     end case;
76 end if;
77 end process process1;
78
79 process2 : process (State)
80 begin
81     case State is
82         when S0 => Z <= "00";
83         when S5 => Z <= "00";
84         when S10 => Z <= "00";
85         when S15 => Z <= "10";
86         when S20 => Z <= "11";
87     end case;
88 end process process2;
89 end Behavioral;

```

รูปที่ L1.3 ได้ด VHDL วงจรหน่วยควบคุมของเครื่องข่ายนี้อัดรวมอัตโนมัติที่เป็น Moor-type FSM with 2 processes



รูปที่ L1.4 ผล Behavioral simulation ของโก้วงจรหน่วยความคุณของเครื่องขายน้ำอัดลมอัตโนมัติ

1.2) เขียนโก้วงจรเพื่อใช้ทดสอบวงจรหน่วยความคุณของเครื่องขายน้ำอัดลมอัตโนมัติที่รวมวงจรโมโนสเตเบิลไว้แล้ว (เพื่อใช้ Debouncing ตัวญี่ปุ่น Clock) ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_6_4vcx1 แล้ว เขียนโก้วงจรรับ 8 บิตเพื่อสร้างความถี่ 128 Hz นำไฟล์ชื่อ ch4vcxl_ONE_SHOT_FSM และไฟล์ชื่อ ch4vcxl_vending มาทำเป็น Component เสร็จแล้วจะได้โก้วงจรหน่วยความคุณของเครื่องขายน้ำอัดลมอัตโนมัติที่รวมวงจรโมโนสเตเบิลดังรูปที่ L1.5

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex4_6_4vcx1 is
7     Port ( CLK,CLR,OSC : in STD_LOGIC;
8             X : in STD_LOGIC_VECTOR (1 downto 0);
9             Q : out STD_LOGIC_VECTOR (1 downto 0));
10 end ex4_6_4vcx1;
11
12 architecture Behavioral of ex4_6_4vcx1 is
13     component ch4vcxl_ONE_SHOT_FSM is
14         Port ( X,Clock : in STD_LOGIC;
15                 Z : out STD_LOGIC);
16     end component;
17     component ch4vcxl_vending is
18         Port ( C,Clear : in STD_LOGIC;
19                 X : in STD_LOGIC_VECTOR (1 downto 0);
20                 Z : out STD_LOGIC_VECTOR (1 downto 0));
21     end component;
22     signal C : STD_LOGIC;
23     signal F : STD_LOGIC_VECTOR (7 downto 0):= (others => '0');
24 begin
25     -----8Bits Counter-----
26 process(OSC) --OSC=32.768kHz, Output F(7)=128Hz < 150Hz
27     begin
28         if OSC'event and OSC='1' then F <= F + 1;
29     end if;
30 end process;
31     -----ONE_SHOT-----
32 ONE_SHOT : ch4vcxl_ONE_SHOT_FSM
33     port map( X => CLK,Clock => F(7),Z => C);
34     -----Vending-----
35 Sequencer : ch4vcxl_vending
36     port map(C => C,Clear=> CLR,X => X,Z => Q);
37
38 end Behavioral;

```

รูปที่ L1.5 โก้วงจรเพื่อใช้ทดสอบหน่วยความคุณของเครื่องขายน้ำอัดลมอัตโนมัติ

การกำหนดขาสัญญาณต่างๆ จะใช้อสัชโนเลเตอร์ OSC = 32.768 kHz และ PB1, PB3(Slide SW1)-PB5(Slide SW1) เป็นอินพุต มี LED3-LED4 เป็นเอาต์พุต ก่อร่างกาย

CLK = PB1 = INPUT = p39

Z(1) = LED3 = OUTPUT = p36

X(1) = PB3(Slide SW1) = INPUT = p42

Z(0) = LED4 = OUTPUT = p35

X(0) = PB4(Slide SW2) = INPUT = p43

CLR = PB5(Slide SW3) = INPUT = p44

OSC = OSC = INPUT = p5

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
CLK	Input	p39	2	9		
CLR	Input	p44	2	17		
OSC	Input	p5	1	9		
X<1>	Input	p42	2	14		
X<0>	Input	p43	2	15		
Q<1>	Output	p36	2	5	SLOW	
Q<0>	Output	p35	2	2	SLOW	

รูปที่ L1.6 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป CPLD แล้วเลื่อน Slide SW3 ไปที่ตำแหน่ง ON (CLR = '0') แล้วทำการทดสอบดังนี้

1) หยดหยดเหรียญ 5 บาทเพียงอย่างเดียว เลื่อน Slide SW1 ไปที่ตำแหน่ง ON (X(1) = '0') เลื่อน Slide SW2 ไปที่ตำแหน่ง OFF (X(0) = '1') แล้วกดปุ่ม PB1 ครั้ง ที่จะครั้ง (จำนวนครั้งที่กดจะเท่ากับจำนวนเหรียญ 5 บาทที่หยด) จนกระทิ้ง LED3 ติดสว่าง (เครื่องจะปล่อยกระแสป้องน้ำอัดลมออกมา) เสรีจแล้วเลื่อน Slide SW2 ไปที่ตำแหน่ง ON (X(0) = '0') กดปุ่ม PB1 อีก 1 ครั้งแล้ว LED3 ดับและพร้อมที่จะขยต่อไป ให้สังเกตว่าเครื่องมีการกินเหรียญต่อเนื่องเกิน 15 บาท

2) หยดหยดเหรียญ 10 บาทเพียงอย่างเดียว เลื่อน Slide SW1 ไปที่ตำแหน่ง OFF (X(1) = '1') เลื่อน Slide SW2 ไปที่ตำแหน่ง ON (X(0) = '0') แล้วกดปุ่ม PB1 ครั้ง ที่จะครั้ง (จำนวนครั้งที่กดจะเท่ากับจำนวนเหรียญ 10 บาทที่หยด) จนกระทิ้ง LED3 และ LED4 ติดสว่าง (เครื่องจะปล่อยกระแสป้องน้ำอัดลมและthonเงิน 5 บาทออกมา) เสรีจแล้วเลื่อน Slide SW1 ไปที่ตำแหน่ง ON (X(1) = '0') แล้วกดปุ่ม PB1 อีก 1 ครั้งแล้ว LED3 และ LED4 จะดับ และพร้อมที่จะขยต่อไป ให้สังเกตว่าเครื่องมีการกินเหรียญในกรณีใส่เหรียญต่อเนื่องเกิน 20 บาท

3) หยดเหรียญ 5 บาทและเหรียญ 10 บาท เลื่อน Slide SW1 ไปที่ตำแหน่ง ON (X(1) = '0') เลื่อน Slide SW2 ไปที่ตำแหน่ง OFF (X(0) = '1') และกดปุ่ม PB1 1 ครั้ง แล้วเลื่อน Slide SW1 ไปที่ตำแหน่ง OFF (X(1) = '1') เลื่อน Slide SW2 ไปที่ตำแหน่ง ON (X(0) = '0') และกดปุ่ม PB1 อีก 1 ครั้ง แล้วให้สังเกตว่า LED3 ติดสว่างหรือไม่ เสรีจแล้วเลื่อน Slide SW1 ไปที่ตำแหน่ง ON (X(1) = '0') กดปุ่ม PB1 อีก 1 ครั้งแล้ว LED3 ดับและพร้อมที่จะขยต่อไป

4) หยดเหรียญ 10 บาทและเหรียญ 5 บาท เลื่อน Slide SW1 ไปที่ตำแหน่ง OFF (X(1) = '1') เลื่อน Slide SW2 ไปที่ตำแหน่ง ON (X(0) = '0') และกดปุ่ม PB1 1 ครั้ง แล้วเลื่อน Slide SW1 ไปที่ตำแหน่ง ON (X(1) = '0') เลื่อน Slide SW2 ไปที่ตำแหน่ง OFF (X(0) = '1') และกดปุ่ม PB1 อีก 1 ครั้ง แล้วให้สังเกตว่า LED3 ติดสว่างหรือไม่ เสรีจแล้วเลื่อน Slide SW2 ไปที่ตำแหน่ง ON (X(0) = '0') กดปุ่ม PB1 อีก 1 ครั้งแล้ว LED3 ดับและพร้อมที่จะขยต่อไป

2 สร้างวงจรหน่วยความคุณเครื่องขายน้ำอัดลมอัตโนมัติด้วย FPGA

การเขียนโค้ดวงจรหน่วยความคุณของเครื่องขายน้ำอัดลมอัตโนมัติด้วย FPGA จะเหมือนกับ CPLD ทุกประการและการทดลองเป็นดังนี้

2.1) เขียนโค้ด VHDL วงจรหน่วยความคุณของเครื่องขายน้ำอัดลมอัตโนมัติเป็น Moor-type FSM with 2 processes ไว้ใน Project Location ชื่อ ch4v เมื่อตอนในรูปที่ L1.3 แต่กำหนด Project Name และ Source File เป็นชื่อ ch4vf_vending ทำการบันทึกไฟล์ และ Check syntax จากนั้นทำ Behavioral simulation เมื่อตอนในรูปที่ L1.4 แต่ใช้ Source File ชื่อ ch4vf_vending_tb1 แล้วให้คุณทดลองการทำงานว่าเป็นไปตามทฤษฎีหรือไม่

2.2) เขียนโค้ดวงจรเพื่อใช้ทดสอบวงจรหน่วยความคุณของเครื่องขายน้ำอัดลมอัตโนมัติที่รวมวงจรไมโครโคนิกส์เดียว (เพื่อใช้ Debouncing สัญญาณ Clock) ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_6_4vf แล้วเขียนโค้ดวงจนับ 18 บิตเพื่อสร้างความถี่ 95.37 Hz นำไฟล์ชื่อ ch4vf_ONE_SHOT_FSM และไฟล์ชื่อ ch4vf_vending มาทำเป็น Component เสร็จแล้วจะได้โค้ดวงจรหน่วยความคุณของเครื่องขายน้ำอัดลมอัตโนมัติที่รวมวงจรไมโครโคนิกส์เดียวดังรูปที่ L2.1

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity ex4_6_4vf is
7      Port ( CLK,CLR,OSC : in STD_LOGIC;
8             X : in STD_LOGIC_VECTOR (1 downto 0);
9             Q : out STD_LOGIC_VECTOR (1 downto 0));
10 end ex4_6_4vf;
11
12 architecture Behavioral of ex4_6_4vf is
13     component ch4vf_ONE_SHOT_FSM is
14         Port ( X,Clock : in STD_LOGIC;
15                 Z : out STD_LOGIC);
16     end component;
17     component ch4vf_vending is
18         Port ( C,Clear : in STD_LOGIC;
19                 X : in STD_LOGIC_VECTOR (1 downto 0);
20                 Z : out STD_LOGIC_VECTOR (1 downto 0));
21     end component;
22     signal C : STD_LOGIC;
23     signal F : STD_LOGIC_VECTOR (17 downto 0):= (others => '0');
24 begin
25     --18Bits Counter--
26 process(OSC) --OSC=25kHz, Output F(17)=95.37Hz < 150Hz
27     begin
28         if OSC'event and OSC='1' then F <= F + 1;
29     end if;
30 end process;
31     --ONE_SHOT-----
32 ONE_SHOT : ch4vf_ONE_SHOT_FSM
33     port map( X => CLK,Clock => F(7),Z => C);
34     --Vending-----
35 Sequencer : ch4vf_vending
36     port map(C => C,Clear=> CLR,X => X,Z => Q);
37
38 end Behavioral;

```

รูปที่ L2.1 โค้ดวงจรเพื่อใช้ทดสอบหน่วยความคุณของเครื่องขายน้ำอัดลมอัตโนมัติ

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 32.768 kHz และ PB1, Dip SW1-Dip SW3 เป็นอินพุต มี LED L0-L1 เป็นเอาต์พุต กล่าวคือ

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

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

$$\text{X}(1) = \text{Dip SW1} = \text{INPUT} = \text{p52}$$

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

X(0) = Dip SW2 = INPUT = p33

CLR = Dip SW3 = INPUT = p55

OSC = OSC = INPUT = p127

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLK	Input	p44	BANK	LVCMS33	N/A	3.30				Unknown		
CLR	Input	p55	BANK	LVCMS33	N/A	3.30				Unknown		
OSC	Input	p127	BANK	LVCMS33	N/A	3.30				Unknown		
Q<0>	Output	p70	BANK	LVCMS33	N/A	3.30		SLOW		Unknown		
Q<1>	Output	p77	BANK	LVCMS33	N/A	3.30		SLOW		Unknown		
X<0>	Input	p53	BANK	LVCMS33	N/A	3.30				Unknown		
X<1>	Input	p52	BANK	LVCMS33	N/A	3.30				Unknown		

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิพ FPGA แล้วเลื่อน Dip SW3 ไปที่ตำแหน่ง ON (CLR = '0') แล้วทำการทดสอบดังนี้

1) หยอดเหรียญ 5 บาทเพียงอย่างเดียว เลื่อน Dip SW1 ไปที่ตำแหน่ง ON (X(1) = '0') เลื่อน Dip SW2 ไปที่ตำแหน่ง OFF (X(0) = '1') แล้วกดปุ่ม PB1 ช้าๆ ทีละครั้ง (จำนวนครั้งที่กดจะเท่ากับจำนวนเหรียญ 5 บาทที่หยอด) จนกระทิ้ง LED L1 ติดสว่าง (เครื่องจะปล่อยกระแสป้องน้ำอัดลมออกมมา) เสร็จแล้วเลื่อน Dip SW2 ไปที่ตำแหน่ง ON (X(0) = '0') กดปุ่ม PB1 อีก 1 ครั้งแล้ว LED L1 ดับและพร้อมจะขายต่อไป ให้สังเกตว่าเครื่องมีการกินเหรียญในการณ์ใส่เหรียญต่อเนื่องเกิน 15 บาท

2) หยอดเหรียญ 10 บาทเพียงอย่างเดียว เลื่อน Dip SW1 ไปที่ตำแหน่ง OFF (X(1) = '1') เลื่อน Dip SW2 ไปที่ตำแหน่ง ON (X(0) = '0') แล้วกดปุ่ม PB1 ช้าๆ ทีละครั้ง (จำนวนครั้งที่กดจะเท่ากับจำนวนเหรียญ 10 บาทที่หยอด) จนกระทิ้ง LED3 และ LED4 ติดสว่าง (เครื่องจะปล่อยกระแสป้องน้ำอัดลมและthonเงิน 5 บาทออกมมา) เสร็จแล้วเลื่อน Dip SW1 ไปที่ตำแหน่ง ON (X(1) = '0') แล้วกดปุ่ม PB1 อีก 1 ครั้งแล้ว LED L1 และ L0 จะดับ และพร้อมจะขายต่อไป ให้สังเกตว่าเครื่องมีการกินเหรียญในการณ์ใส่เหรียญต่อเนื่องเกิน 20 บาท

3) หยอดเหรียญ 5 บาทและเหรียญ 10 บาท เลื่อน Dip SW1 ไปที่ตำแหน่ง ON (X(1) = '0') เลื่อน Dip SW2 ไปที่ตำแหน่ง OFF (X(0) = '1') และกดปุ่ม PB1 1 ครั้ง แล้วเลื่อน Dip SW1 ไปที่ตำแหน่ง OFF (X(1) = '1') เลื่อน Dip SW2 ไปที่ตำแหน่ง ON (X(0) = '0') และกดปุ่ม PB1 อีก 1 ครั้ง แล้วให้สังเกตว่า LED L1 ติดสว่างหรือไม่ เสร็จแล้วเลื่อน Dip SW1 ไปที่ตำแหน่ง ON (X(1) = '0') กดปุ่ม PB1 อีก 1 ครั้งแล้ว LED L1 ดับและพร้อมจะขายต่อไป

4) หยอดเหรียญ 10 บาทและเหรียญ 5 บาท เลื่อน Dip SW1 ไปที่ตำแหน่ง OFF (X(1) = '1') เลื่อน Dip SW2 ไปที่ตำแหน่ง ON (X(0) = '0') และกดปุ่ม PB1 1 ครั้ง แล้วเลื่อน Dip SW1 ไปที่ตำแหน่ง ON (X(1) = '0') เลื่อน Dip SW2 ไปที่ตำแหน่ง OFF (X(0) = '1') และกดปุ่ม PB1 อีก 1 ครั้ง แล้วให้สังเกตว่า LED L1 ติดสว่างหรือไม่ เสร็จแล้วเลื่อน Dip SW2 ไปที่ตำแหน่ง ON (X(0) = '0') กดปุ่ม PB1 อีก 1 ครั้งแล้ว LED L1 ดับและพร้อมจะขายต่อไป

แบบฝึกหัด

- ให้นักศึกษาเขียนโค้ด VHDL หน่วยควบคุมของเครื่องขายน้ำอัดลมอัตโนมัติใหม่ เพื่อแก้ไขปัญหาการกินเหรียญในกรณีใส่เหรียญต่อเนื่องตามที่เราได้ทดลองไปแล้ว โดยให้เขียนโค้ดโดยใช้วิธี FSM with 2 processes และ FSM with 3 processes

4.7 ROM และ RAM

หน่วยความจำ (Memory) สำหรับงานด้านคอมพิวเตอร์มีหลากหลายชนิด แต่ที่เป็นหน่วยความจำพื้นฐานได้แก่ ROM (Read Only Memory) และ RAM (Random Access Memory) โดยที่ ROM จะเป็นหน่วยความจำแบบตายตัวที่สามารถอ่านค่า ออกมากได้เพียงอย่างเดียวและข้อมูลจะคงอยู่เมื่อไม่มีไฟเลี้ยงอยู่ก็ตาม ส่วน RAM จะเป็นหน่วยความจำที่เขียนข้อมูลลงไปใหม่ หรืออ่านข้อมูลออกมาได้และข้อมูลจะหายไปเมื่อไม่มีไฟเลี้ยง

การเขียนโค้ดของ ROM มีรายละเอียดในตัวอย่างที่ 2.3 ของบทที่ 2 การกำหนดค่าที่อยู่ใน ROM ซึ่งมีค่าที่แน่นอน (Literals) แสดงตัวอย่างดังรูปที่ 4.89 และผลการทำงานแสดงดังรูปที่ 4.90 โดยที่โค้ดรูปที่ 4.89 นี้จะแสดงการใส่ค่าเริ่มต้นใน ROM เช่น Address ที่ 11-15 ใส่ค่า “1A” ที่เป็นเลขฐาน 16 ก็จะให้ผลเช่นเดียวกับการใส่ค่า “00011010” ที่เป็นเลขฐาน 2

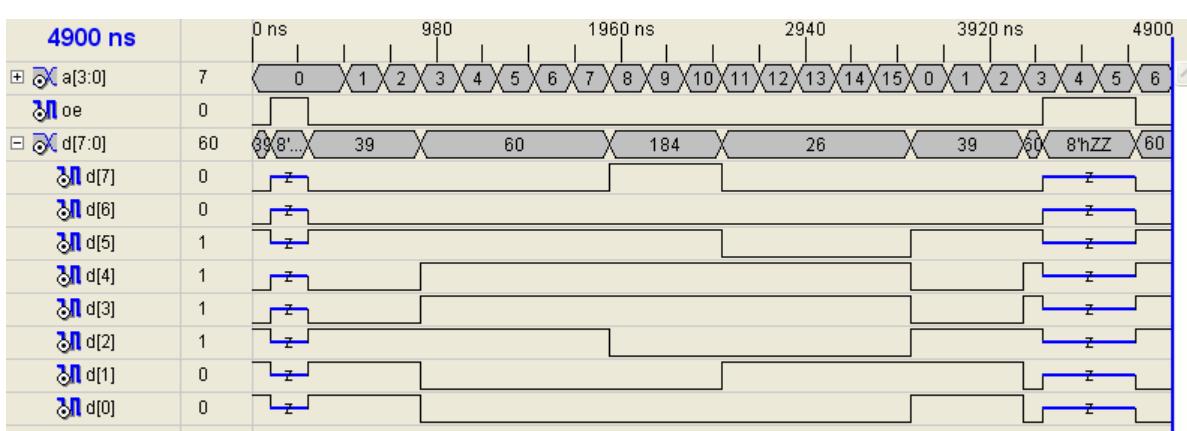
การกำหนดค่าให้กับ ROM สามารถทำได้โดยการประกาศใช้ constant ในบรรทัดที่ 14 หรือประกาศใช้ signal ที่ให้ผลเหมือนกัน เพราะค่าของ ROM ถูกกำหนดค่าตั้งแต่เริ่มต้น

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ROM16x8BIT is
7     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);      --Address
8             OE : in STD_LOGIC;                      --Output enable
9             D : out STD_LOGIC_VECTOR (7 downto 0)); --Output data
10 end ROM16x8BIT;
11
12 architecture Behavioral of ROM16x8BIT is
13     type ROM_type is array (15 downto 0) of std_logic_vector (7 downto 0);
14     constant ROM_ADDR_N : ROM_type := (15 downto 11 => X"1A", -- X = HEX.
15                                         10 downto 8 => X"B8",
16                                         7 downto 3 => X"3C",
17                                         2 downto 0 => X"27");
18 begin
19     process (A,OE)
20         begin
21             if OE = '1' then
22                 D <= (others => 'Z');
23             else
24                 D <= ROM_ADDR_N( conv_integer(A) );
25             end if;
26     end process;
27 end Behavioral;

```

รูปที่ 4.89 โค้ด VHDL ของ ROM ขนาด 16x8 บิต



รูปที่ 4.90 โค้ด VHDL ของ ROM ขนาด 16x8 บิต

การเขียนโค้ด RAM แบบชิงโครนัสขนาด 16x4 บิตมีตัวอย่างแสดงดังรูปที่ 4.91 สำหรับการใส่ค่าเริ่มต้นให้กับ RAM มีตัวอย่างแสดงดังรูปที่ 4.92 และผลสังเคราะห์วงจรโดยใช้ FPGA (Spartan3) จะได้ดังรูปที่ 4.93 โดยที่การทำงานของ RAM แบบชิงโกรนัส ถ้า WE = '1' ก็จะเป็นการเขียนข้อมูล แต่ถ้า WE = '0' ก็จะเป็นการอ่านข้อมูล และขอให้สังเกตว่าในกรณีของ RAM นั้นบรรทัดที่ 15 จะต้องประกาศเป็น signal เนื่องจาก signal จะต้อง Update ค่าได้เมื่อเราเขียนข้อมูลลง RAM สำหรับในรูปที่ 4.94 โค้ดของ RAM ขนาด 16x4 บิตที่มีขา Chip enable

```

2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_unsigned.all;
5
6 entity RAM16x4BIT is
7     port (CLK : in std_logic;
8           WE : in std_logic;
9           ADDR : in std_logic_vector(3 downto 0);
10          DATA : inout std_logic_vector(3 downto 0));
11 end RAM16x4BIT;
12
13 architecture Behavioral of RAM16x4BIT is
14 type RAM_type is array (15 downto 0) of std_logic_vector (3 downto 0);
15 signal RAM_ADDR_N : RAM_type;
16 begin
17     process (CLK,WE)
18     begin
19         if WE = '1' then
20             DATA <= (others => 'Z');
21             if CLK'event and CLK = '1' then
22                 RAM_ADDR_N(conv_integer(ADDR)) <= DATA;
23             end if;
24         else DATA <= RAM_ADDR_N(conv_integer(ADDR));
25         end if;
26     end process;
27 end Behavioral;

```

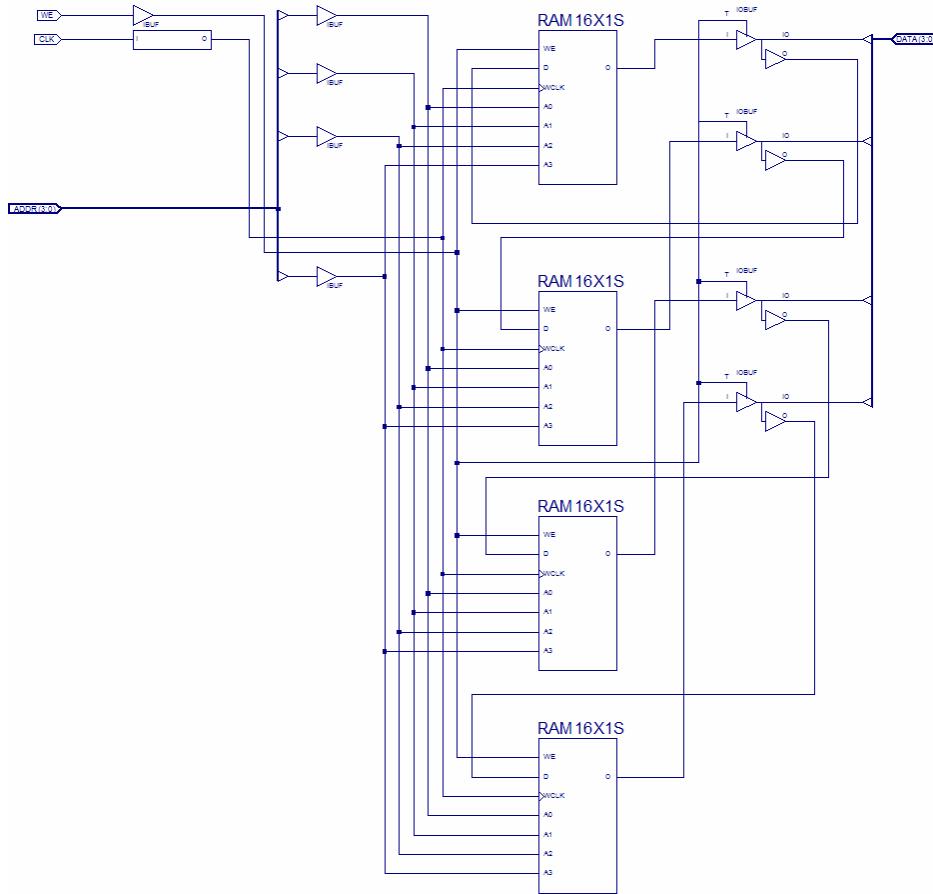
รูปที่ 4.91 โค้ดของ RAM ขนาด 16x4 บิต

```

2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_unsigned.all;
5
6 entity RAM16x4BIT is
7     port (CLK : in std_logic;
8           WE : in std_logic;
9           ADDR : in std_logic_vector(3 downto 0);
10          DATA : inout std_logic_vector(3 downto 0));
11 end RAM16x4BIT;
12
13 architecture Behavioral of RAM16x4BIT is
14 type RAM_type is array (15 downto 0) of std_logic_vector (3 downto 0);
15 signal RAM_ADDR_N : RAM_type :=("0111","0110","0101","0100",
16                                     "0011","0010","0001","0000",
17                                     "1111","1110","1101","1100",
18                                     "1011","1010","1001","1000");
19 begin
20     process (CLK,WE)
21     begin
22         if WE = '1' then
23             DATA <= (others => 'Z');
24             if CLK'event and CLK = '1' then
25                 RAM_ADDR_N(conv_integer(ADDR)) <= DATA;
26             end if;
27         else DATA <= RAM_ADDR_N(conv_integer(ADDR));
28         end if;
29     end process;
30 end Behavioral;

```

รูปที่ 4.92 โค้ดของ RAM ขนาด 16x4 บิตที่มีการใส่ค่าเริ่มต้น (Initial value)



รูปที่ 4.93 ผลลัพธ์เคราะห์วงจรของโลคัล RAM ขนาด 16x4 บิต โดยใช้ FPGA (Spartan3)

```

2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_unsigned.all;
5
6 entity RAM16x4BIT is
7     port (CLK : in std_logic;
8             EN,WE : in std_logic; --EN=Chip enable, WE= write enable
9             ADDR : in std_logic_vector(3 downto 0);
10            DATA : inout std_logic_vector(3 downto 0));
11 end RAM16x4BIT;
12
13 architecture Behavioral of RAM16x4BIT is
14 type RAM_type is array (15 downto 0) of std_logic_vector (3 downto 0);
15 signal RAM_ADDR_N : RAM_type;
16 begin
17     process (CLK,EN,WE)
18     begin
19         if EN = '1' and WE = '1' then
20             DATA <= (others => 'Z');
21             if CLK'event and CLK = '1' then
22                 RAM_ADDR_N(conv_integer(ADDR)) <= DATA;
23             end if;
24         elsif EN = '1' and WE = '0' then
25             DATA <= RAM_ADDR_N(conv_integer(ADDR));
26         else
27             DATA <= (others => 'Z');
28         end if;
29     end process;
30 end Behavioral;

```

รูปที่ 4.94 โลคัลของ RAM ขนาด 16x4 บิตที่มีขา Chip enable

การทดลองที่ 4.7.1 ROM

วัตถุประสงค์

- 1) เพื่อออกแบบหน่วยความจำประเภท ROM
- 2) เพื่อสร้าง ROM โดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้าง ROM ด้วย CPLD

นำโค้ด ROM ขนาด 16×8 บิตในรูปที่ 4.89 มาแก้ชื่อ Entity เป็น ex4_7_1vcx1 และสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_7_1vcx1 บันทึกไฟล์และ Check syntax และทำ Behavioral simulation เขียนเดียวกับรูปที่ 4.90 โดยใช้ไฟล์ชื่อ ex4_7_1vcx1_tb1 และพิจารณาผลว่าเป็นไปตามทฤษฎีหรือไม่

การกำหนดขาสัญญาณต่างๆ ของ CPLD เป็นดังนี้

$OE = PB1 = INPUT = p39$	$Q(3) = LED1 = OUTPUT = p38$	$Q(7) = MN1 = OUTPUT = p7$
$A(3) = PB3 (Slide SW1) = INPUT = p42$	$Q(2) = LED2 = OUTPUT = p37$	$Q(6) = MN2 = OUTPUT = p6$
$A(2) = PB4 (Slide SW2) = INPUT = p43$	$Q(1) = LED3 = OUTPUT = p36$	$Q(5) = MN3 = OUTPUT = p4$
$A(1) = PB5 (Slide SW3) = INPUT = p44$	$Q(0) = LED4 = OUTPUT = p35$	$Q(4) = MN4 = OUTPUT = p3$
$A(0) = PB6 (Slide SW4) = INPUT = p1$		

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
$A<3>$	Input	p42	2	14		
$A<2>$	Input	p43	2	15		
$A<1>$	Input	p44	2	17		
$A<0>$	Input	p1	1	2		
OE	Input	p39	2	9		
$D<7>$	Output	p7	1	14	SLOW	
$D<6>$	Output	p6	1	11	SLOW	
$D<5>$	Output	p4	1	8	SLOW	
$D<4>$	Output	p3	1	6	SLOW	
$D<3>$	Output	p38	2	8	SLOW	
$D<2>$	Output	p37	2	6	SLOW	
$D<1>$	Output	p36	2	5	SLOW	
$D<0>$	Output	p35	2	2	SLOW	

รูปที่ L1.1 Assign Package Pins

หลังจากโปรแกรม CPLD ให้กดปุ่ม PB1 และปล่อยสลับกันแล้วดูผลที่ LED1-LED4 และ Mn1-MN4 ว่าติดสว่างโดยให้กลอจิกເອົາດີພຸດເປັນໄປຕາມທѹຍຖື້ງໃໝ່ ຈາກນັ້ນให้กดปຸ່ມ PB1 ຄ້າງໄວ້ และເລື່ອນ Slide SW1- Slide SW4 ໄປທີ່ ON ຖຸກຕົວຫຼືອ Address A = “0000” และปล่อยสลับກັນແລ້ວดູຜົບທີ່ LED1-LED4 และ Mn1-MN4 ຈາກໃຫ້ทดลองໜ້າແຕ່ເປົ້າຢືນ Address ເປັນຄ່າອື່ນ ໂດຍ ON ຫຼື OFF ຂອງ Slide SW1- Slide SW4 ເປັນຄ່າ Address ພັດໄປເຮືອຍໆ ຈາກນີ້ A = “1111” ແລ້ວນັ້ນທີ່ກົດການทดลอง

2 สร้าง ROM ด้วย FPGA

นำโค้ด ROM ขนาด 16x8 มิติในรูปที่ 4.89 มาแก้ชื่อ Entity เป็น ex4_7_1vf และสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_7_1vf บันทึกไฟล์และ Check syntax และทำ Behavioral simulation เช่นเดียวกับรูปที่ 4.90 โดยใช้ไฟล์ชื่อ ex4_7_1vf_tb1 และพิจารณาผลว่าเป็นไปตามทฤษฎีหรือไม่

การกำหนดขาสัญญาณต่างๆ ของ FPGA เป็นดังนี้

OE = PB1 = INPUT = p44	Q(3) = L3 = OUTPUT = p76	Q(7) = L7 = OUTPUT = p78
A(3) = Dip SW1 = INPUT= p52	Q(2) = L2 = OUTPUT = p69	Q(6) = L6 = OUTPUT = p73
A(2) = Dip SW2 = INPUT= p53	Q(1) = L1 = OUTPUT = p77	Q(5) = L5 = OUTPUT = p79
A(1) = Dip SW3 = INPUT= p55	Q(0) = L0 = OUTPUT = p70	Q(4) = L4 = OUTPUT = p74
A(0) = Dip SW4 = INPUT= p56		

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
A<0>	Input	p56	BANK	LVCMOS33	N/A	3.30					Unknown		
A<1>	Input	p55	BANK	LVCMOS33	N/A	3.30					Unknown		
A<2>	Input	p53	BANK	LVCMOS33	N/A	3.30					Unknown		
A<3>	Input	p52	BANK	LVCMOS33	N/A	3.30					Unknown		
D<0>	Output	p70	BANK	LVCMOS33	N/A	3.30		SLOW			Unknown		
D<1>	Output	p77	BANK	LVCMOS33	N/A	3.30		SLOW			Unknown		
D<2>	Output	p69	BANK	LVCMOS33	N/A	3.30		SLOW			Unknown		
D<3>	Output	p76	BANK	LVCMOS33	N/A	3.30		SLOW			Unknown		
D<4>	Output	p74	BANK	LVCMOS33	N/A	3.30		SLOW			Unknown		
D<5>	Output	p79	BANK	LVCMOS33	N/A	3.30		SLOW			Unknown		
D<6>	Output	p73	BANK	LVCMOS33	N/A	3.30		SLOW			Unknown		
D<7>	Output	p78	BANK	LVCMOS33	N/A	3.30		SLOW			Unknown		
OE	Input	p44	BANK	LVCMOS33	N/A	3.30					Unknown		

รูปที่ L2.1 Assign Package Pins

หลังจากโปรแกรม FPGA ให้กดปุ่ม PB1 แล้วปล่อยสลับกันแล้วคูลท์ LED L0-L7 ว่าติดสว่างโดยให้ลองเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นให้กดปุ่ม PB1 ค้างไว้ แล้วเลื่อน Dip SW1-Dip SW4 ไปที่ ON ทุกตัวหรือ Address A = “0000” แล้วปล่อยสลับกันแล้วคูลท์ LED L0-L7 จากให้ทดลองซ้ำแต่เปลี่ยน Address เป็นค่าอื่น โดย ON หรือ OFF ของ Dip SW1- Dip SW4 เป็นค่า Adress ตัดไปเรื่อยๆ จนถึง A = “1111” แล้วบันทึกผลการทดลอง

การทดลองที่ 4.7.2 RAM

วัตถุประสงค์

- 1) เพื่อออกแบบหน่วยความจำประเภท RAM แบบชิ้งโคร์นัส
- 2) เพื่อสร้าง RAM แบบชิ้งโคร์นัสโดยวิธีเขียนด้วยโค้ด VHDL และโปรแกรมลงชิป CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้าง RAM ด้วย CPLD

นำโค้ด RAM แบบชิ้งโคร์นัส 16x4 บิตในรูปที่ 4.91 มาแก้ไขเป็น 8x4 บิต และแก้ไขชื่อ Entity เป็น ex4_7_2vcx1 และจะได้ังรูปที่ L1.1 แล้วสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_7_2vcx1 บันทึกไฟล์และ Check syntax

```

2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_unsigned.all;
5
6 entity ex4_7_2vcx1 is
7     port (CLK : in std_logic;
8           WE : in std_logic;
9           ADDR : in std_logic_vector(2 downto 0);
10          DATA : inout std_logic_vector(3 downto 0));
11 end ex4_7_2vcx1;
12
13 architecture Behavioral of ex4_7_2vcx1 is
14     type RAM_type is array (7 downto 0) of std_logic_vector (3 downto 0);
15     signal RAM_ADDR_N : RAM_type;
16 begin
17     process (CLK,WE)
18     begin
19         if WE = '1' then
20             DATA <= (others => 'Z');
21             if CLK'event and CLK = '1' then
22                 RAM_ADDR_N(conv_integer(ADDR)) <= DATA;
23             end if;
24         else DATA <= RAM_ADDR_N(conv_integer(ADDR));
25         end if;
26     end process;
27 end Behavioral;

```

รูปที่ L1.1 โค้ด RAM แบบชิ้งโคร์นัส 8x4 บิต

การกำหนดขาสัญญาณต่างๆ ของ CPLD เป็นดังนี้

ADDR(2)= PB3(Slide SW1)= INPUT= p42	DATA(3) = IO6,MN1 = INPUT/OUTPUT = p7
ADDR(1)= PB4(Slide SW2)= INPUT= p43	DATA(2) = IO7,MN2 = INPUT/OUTPUT = p6
ADDR(0)= PB5(Slide SW3)= INPUT= p44	DATA(1) = IO8,MN3 = INPUT/OUTPUT = p4
WE = PB6(Slide SW4)= INPUT= p1	DATA(0) = IO9,MN4 = INPUT/OUTPUT = p3
CLK = PB1 = INPUT= p39	

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
CLK	Input	p39	2	9		
WE	Input	p1	1	2		
ADDR<2>	Input	p42	2	14		
ADDR<1>	Input	p43	2	15		
ADDR<0>	Input	p44	2	17		
DATA<3>	InOut	p7	1	14	SLOW	
DATA<2>	InOut	p6	1	11	SLOW	
DATA<1>	InOut	p4	1	8	SLOW	
DATA<0>	InOut	p3	1	6	SLOW	

รูปที่ L1.1 Assign Package Pins

หลังจากโปรแกรม CPLD แล้วเขียน RAM โดย OFF Slide SW4 (WE = '1') แล้วเริ่มเขียนจาก Address ADDR = "000" ไปถึง "111" ซึ่งจะป้อน DATA เรียงลำดับดังนี้ "0111", "0110", "0101", "0100", "0011", "0010", "0001" และ "1000" โดยจะป้อน DATA ที่ IO6-IO9 ของคอนเนคเตอร์ K1 กล่าวคือ เช่น ถ้า DATA = "0111" ก็ให้เสียบเฉพาะจัมเปอร์ของ IO6 (เข้ากับ GND) ถ้า DATA = "0110" ก็ให้เสียบเฉพาะจัมเปอร์ของ IO6 และ IO9 (เข้ากับ GND) เป็นต้น ซึ่งถ้าไม่เสียบจัมเปอร์จะให้ล็อกจิก '1' (IO6-IO9 ต่ออยู่กับ Pull up resistor) การป้อนค่าที่ Address ต่างๆ นั้นมีเช็คค่า DATA ที่ K1 แล้วให้ ON หรือ OFF Slide SW1-Slide SW3 ให้เป็น Address ที่ต้องการและกดปุ่ม PB1 (CLK) แล้วปล่อย (ทริกตอนที่เป็นขอบขาขึ้น) การกำหนด Address เช่น ADDR = "000" ทำได้โดย ON Slide SW1 และ Slide SW2 แต่ OFF Slide SW3 เป็นต้น และเมื่อป้อน DATA จนครบทุกถ้าแล้วให้อ่านข้อมูลจาก RAM ซึ่งทำได้โดย ON Slide SW4 (WE = '0') แล้วเช็คค่า Address โดยเริ่มอ่านค่าจาก ADDR = "111" จนไปถึง ADDR = "000" (โดยไม่ต้องกดปุ่ม PB1) พร้อมกับดูผลที่ LED1-LED4 และ Mn1-MN4 ว่าให้ล็อกเอาต์พดเป็นไปตามทฤษฎีหรือไม่ แล้วบันทึกผลการทดลอง

2 สร้าง RAM ด้วย FPGA

นำโค้ด RAM ในรูปที่ 4.91 มาแก้ไขเป็น 8x4 บิตและแก้ไขชื่อ Entity เป็น ex4_7_2vf จะได้ดังรูปที่ L2.1 แล้วสร้างไฟล์ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_7_2vf บันทึกไฟล์และ Check syntax

```

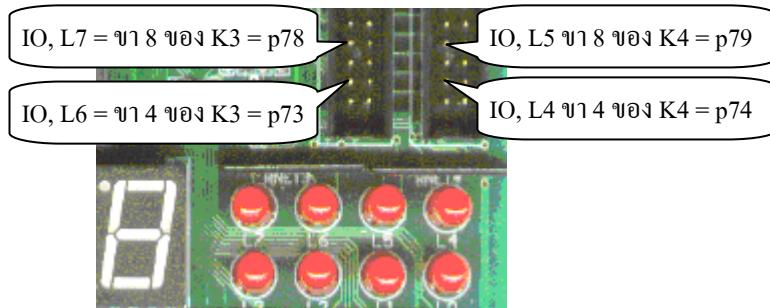
2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_unsigned.all;
5
6 entity ex4_7_2vf is
7   port (CLK : in std_logic;
8         WE : in std_logic;
9         ADDR : in std_logic_vector(2 downto 0);
10        DATA : inout std_logic_vector(3 downto 0));
11 end ex4_7_2vf;
12
13 architecture Behavioral of ex4_7_2vf is
14   type RAM_type is array (7 downto 0) of std_logic_vector (3 downto 0);
15   signal RAM_ADDR_N : RAM_type;
16 begin
17   process (CLK,WE)
18   begin
19     if WE = '1' then
20       DATA <= (others => 'Z');
21       if CLK'event and CLK = '1' then
22         RAM_ADDR_N( conv_integer(ADDR) ) <= DATA;
23       end if;
24     else
25       DATA <= RAM_ADDR_N( conv_integer(ADDR) );
26     end if;
27   end process;
28 end Behavioral;

```

รูปที่ L2.1 โค้ด RAM แบบชิ้งโครนัส 8x4 บิต

การกำหนดขาสัญญาณต่างๆ ของ FPGA เป็นดังนี้

ADDR(2)= Dip SW1 = INPUT= p52	DATA(3) = IO, L7 = INPUT/OUTPUT = p78
ADDR(1)= Dip SW2 = INPUT= p53	DATA(2) = IO, L6 = INPUT/OUTPUT = p73
ADDR(0)= Dip SW3 = INPUT= p55	DATA(1) = IO, L5 = INPUT/OUTPUT = p79
WE = Dip SW4 = INPUT= p56	DATA(0) = IO, L4 = INPUT/OUTPUT = p74
CLK = PB1 = INPUT= p44	



รูปที่ L2.2 ตำแหน่งขาคอนเนกเตอร์ K3 และ K4 โดยที่ ขา (Pin) 1 (ด้านล่างขวา) ของ K3 และ K4 จะเป็น Vcc = +3.3V

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]													
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
ADDR<0>	Input	p55	BANK	LVCMS33	N/A	3.30					Unknown		
ADDR<1>	Input	p53	BANK	LVCMS33	N/A	3.30					Unknown		
ADDR<2>	Input	p52	BANK	LVCMS33	N/A	3.30					Unknown		
CLK	Input	p44	BANK	LVCMS33	N/A	3.30					Unknown		
DATA<0>	InOut	p74	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
DATA<1>	InOut	p79	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
DATA<2>	InOut	p73	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
DATA<3>	InOut	p78	BANK	LVCMS33	N/A	3.30			SLOW		Unknown		
WE	Input	p56	BANK	LVCMS33	V/A	3.30					Unknown		

รูปที่ L2.3 Assign Package Pins

หลังจากโปรแกรม FPGA และวิ่ง RAM โดย OFF Dip SW4 (WE = ‘1’) แล้วเริ่มเขียนจาก Address ADDR = “000” ไปถึง “111” ซึ่งจะป้อน DATA เรียงลำดับดังนี้ “0111”, “0110”, “0101”, “0100”, “0011”, “0010”, “0001” และ “1000” โดยจะป้อน DATA ที่ขา 4 และ 8 ของคอนเนกเตอร์ K3 และ K4 กล่าวคือ เช่น ถ้า DATA = “0111” ที่ให้ต่อ GND (เช่น ขา 3 ของ K3 หรือ K4) เข้ากับ DATA(3) (คือขา 8 ของ K3) และต่อ Vcc = +3.3V (จากขา 1 ของ K3 หรือ K4) เข้ากับ DATA(2), DATA(1), DATA(0) (คือขา 4 ของ K3, ขา 8 ของ K4 และขา 4 ของ K4 ตามลำดับ) เป็นต้น การป้อนค่าที่ Address ต่างๆ นั้นมีอีกด้วย DATA แล้วให้ ON หรือ OFF Dip SW1-Slide SW3 ให้เป็น Address ที่ต้องการและกดปุ่ม PB1 (CLK) และปล่อย (ทริกตอนที่เป็นขอบขาขึ้น) การกำหนด Address เช่น ถ้าเป็น ADDR = “000” ทำได้โดย ON Dip SW1-Dip SW3 ทุกตัว แต่ถ้าเป็น ADDR = “001” ทำได้โดย ON Dip SW1 และ Dip SW2 แต่ OFF Dip SW3 เป็นต้น และเมื่อป้อน DATA จนครบทุกค่าแล้วให้อ่านข้อมูลจาก RAM ซึ่งสามารถทำได้โดย ON Dip SW4 (WE = ‘0’) และเซ็ตค่า Address โดยเริ่มอ่านจาก ADDR = “111” ไปถึง ADDR = “000” (โดยไม่ต้องกดปุ่ม PB1) พร้อมกับดูผลที่ LED L0-L7 ว่าติดสว่างโดยให้ก็อจิกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ แล้วบันทึกผลการทดลอง

บทที่ 5

Package, Library และ Subprogram

5.1 Package

แพคเกจ (Package) เป็นที่รวบรวมโค้ดย่อยต่างๆ ที่อาจจะเป็น ค่าคงที่ (Constant) นิยามของชนิดข้อมูล การประมวลผล ใช้คอมโพเนนต์ (Component declaration) และ/หรือ โปรแกรมย่อย (Subprogram) ใน Package อาจเขียนหลาขอxygen ไว้ใน Package เดียวกันได้ ซึ่งสิ่งที่อยู่ใน Package นั้นเป็นของส่วนกลางหรือกองกลางที่ใช้ร่วมกัน การเขียนโค้ดของวงจรหรือแบบจำลองหรือโมเดลต่างๆ นั้นสามารถเรียกโค้ดย่อยจาก Package ต่างๆ ไปใช้โดยไม่ต้องเสียเวลาเขียนโค้ดซ้ำ โดยที่ Package ต่างๆ นั้นจะถูกเก็บรวบรวมไว้ในไลบรารี (Library) ต่างๆ

Package อาจประกอบด้วย 2 ส่วน คือ ประกาศใช้แพคเกจ (Package declaration) และ แพคเกจบอดี (Package body) โดยที่ส่วนของบอดีนั้นอาจจะมีหรือไม่มีก็ได้

5.1.1 การประกาศใช้ Package

การประกาศใช้ Package (Package declaration) มีรูปแบบการประกาศใช้ดังนี้

```
package PACKAGE_NAME is
    {PACKAGE_DECLARATIVE_ITEM}
end [ package ] [ PACKAGE_NAME ] ;
```

โดยที่ 1) PACKAGE_DECLARATIVE_ITEM เป็นดังนี้

- use clause (เมื่อมีการเรียกใช้จาก Package อื่น)
- Subprogram declaration
- Type declaration และ/หรือ Subtype declaration
- Constant declaration และ/หรือ Signal declaration
- Component declaration

2) {...}หมายถึงจะมีหรือไม่มีก็ได้หรืออาจมีได้หลายตัว แต่ถ้าเป็น [...] หมายถึงจะมีหรือไม่มีก็ได้

5.1.2 Package body

แพคเกจบอดี (Package body) มีรูปแบบการประกาศใช้ดังนี้

```
package body PACKAGE_NAME is
    {PACKAGE_BODY_DECLARATIVE_ITEM}
end [ package body ] [ PACKAGE_NAME ] ;
```

โดยที่ 1) PACKAGE_BODY_DECLARATIVE_ITEM เป็นดังนี้

- use clause
- Subprogram declaration และ/หรือ Subprogram body
- Type declaration
- Constant declaration
- Component declaration

2) {...}หมายถึงจะมีหรือไม่มีได้หรืออาจมีได้หลายตัว แต่ถ้าเป็น [...] หมายถึงจะมีหรือไม่มีได้

5.1.3 การเรียกใช้ Package

การเรียกใช้ Package ในกรณีนี้จะแตกต่างจากข้อ 2.10.1 เล็กน้อย เมื่อจาก Package ที่เราเขียนขึ้นนั้นจะถูกคอมไพล์ เก็บไว้ใน Library ซึ่ง work ซึ่ง Library นี้จะถูกเรียกใช้โดยอัตโนมัติที่กำหนดไว้ล่วงหน้าไว้แล้ว (Default) จึงไม่จำเป็นต้องเรียกใช้ Library ซึ่ง work ซึ่งอีก โดยจะเรียกใช้เฉพาะชื่อ Package เท่านั้นและจะต้องเขียนไว้ที่ด้านบนสุดของโค้ด การเรียกใช้ Package (Using the package) จะใช้คำสั่ง Use (Use clause) มีรูปแบบการเขียนที่ว่าไปดังนี้

```
use WORK.PACKAGE_NAME.all;
```

ตัวอย่างที่ 5.1 ฝึกเขียน Package และเรียกใช้ Package โดยเขียนโค้ดของวงจรบวก 3 บิตที่แสดงผลทางเลขฐานเทอมต์ 1 หลักในตัวอย่างที่ 2.14 เสียใหม่ โดยนำโค้ดของวงจรบวก 2 บิตในรูปที่ 2.31b) และ โค้ดของวงจรดอร์หัสตัวแสดงผลทางเลขฐานเทอมต์ในรูปที่ 2.31c) มาทำเป็น Component โดยประกาศใช้ Component ทั้ง 2 วงจรไว้ใน Package ชื่อ PACK_ADDER3BIT_7SEG แสดงดังรูปที่ 5.1a) จากนั้นให้เขียนโค้ดของวงจรบวกขนาด 3 บิตโดยเรียกใช้ Package ชื่อ PACK_ADDER3BIT_7SEG ที่อยู่ใน Library ซึ่ง work แสดงดังรูปที่ 5.1b) ซึ่งจะเห็นได้ว่าการเขียนโค้ดในรูปที่ 5.1b) จะสั้นกว่าโค้ดในรูปที่ 2.31d)

ขอให้สังเกตว่าก่อนการเขียน Package ในรูปที่ 5.1a) นั้นเราจะต้องเขียนโค้ดของวงจรบวก 2 บิตในรูปที่ 2.31b) และ โค้ดของวงจรดอร์หัสตัวแสดงผลทางเลขฐานเทอมต์ในรูปที่ 2.31c) (ในบทที่ 2) ให้เสร็จเรียบร้อยเสียก่อน

```
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.all;
4
5 package PACK_ADDER3BIT_7SEG is
6
7   component ADDER_2BIT
8     generic ( N : integer := 2);
9     port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
10           C : out STD_LOGIC_VECTOR (N downto 0));
11   end component;
12   component HEX_TO_7SEGMENT
13     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
14            Y : out STD_LOGIC_VECTOR (6 downto 0));
15   end component;
16
17 end PACK_ADDER3BIT_7SEG;
```

ต้องเรียกใช้ std_logic_1164 package เพราะใช้กับชนิดข้อมูล std_logic_vector

บรรทัดที่ 7-15 เป็นการประกาศใช้คุณสมบัติ

5.1a) การประกาศใช้คุณสมบัติไว้ใน Package

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use WORK.PACK_ADDER3BIT_7SEG.ALL; -- เรียกใช้ Package ชื่อ PACK_ADDER3BIT_7SEG
5                                         ที่อยู่ใน Library ชื่อ work
6 entity ADDER3BIT_1DIGIT is
7     Port ( A,B : in STD_LOGIC_VECTOR (2 downto 0);
8             Y : out STD_LOGIC_VECTOR (6 downto 0));
9 end ADDER3BIT_1DIGIT;
10
11 architecture Behavioral of ADDER3BIT_1DIGIT is
12     signal C : STD_LOGIC_VECTOR (3 downto 0); -- Signal C ใช้ต่อจากเอกสารพูด
13 begin
14     -- IC1=ADDER3BIT --
15     IC1 : ADDER_2BIT generic map (N => 3)      -- ADDER_2BIT=>ADDER_3BIT
16                     port map (A,B,C);           -- Positional association
17
18     -- IC2=7SEGMENT DECODER --
19     IC2 : HEX_TO_7SEGMENT port map (A => C, Y => Y); -- Named association
20
21 end Behavioral;

```

5.1b) โค้ด VHDL ของวงจรบวกขนาด 3 บิตที่แสดงผลทางเขียนเชกเม้นต์จำนวน 1 หลักที่ใช้ Component

รูปที่ 5.1 โค้ด VHDL ของวงจรบวกขนาด 3 บิตที่แสดงผลทางเขียนเชกเม้นต์จำนวน 1 หลัก

ตัวอย่างที่ 5.2 ให้เขียนโค้ดตัวอย่างที่ 5.1 ใหม่โดยรวมโค้ดของ Package ไว้ในไฟล์ของโค้ดหลักของวงจรบวก 3 บิตแสดงดังรูปที่ 5.2

```

2 -----Package code-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.all;
5
6 package PACK_ADDER3BIT_7SEG is
7
8     component ADDER_2BIT
9         generic ( N : integer := 2);
10        port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
11                C : out STD_LOGIC_VECTOR (N downto 0));
12    end component;
13    component HEX_TO_7SEGMENT
14        Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
15                Y : out STD_LOGIC_VECTOR (6 downto 0));
16    end component;
17
18 end PACK_ADDER3BIT_7SEG;
19
20 -----Main code-----
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23 use WORK.PACK_ADDER3BIT_7SEG.ALL;
24
25 entity a3_1d is
26     Port ( A,B : in STD_LOGIC_VECTOR (2 downto 0);
27             Y : out STD_LOGIC_VECTOR (6 downto 0));
28 end a3_1d;
29
30 architecture Behavioral of a3_1d is
31     signal C : STD_LOGIC_VECTOR (3 downto 0);
32 begin
33     -- IC1=ADDER3BIT --
34     IC1 : ADDER_2BIT generic map (N => 3)      -- ADDER_2BIT=>ADDER_3BIT
35                     port map (A,B,C);           -- Positional association
36
37     -- IC2=7SEGMENT DECODER --
38     IC2 : HEX_TO_7SEGMENT port map (A => C, Y => Y); -- Named association
39
40 end Behavioral;

```

รูปที่ 5.2 การเขียนโค้ดของ Package ในรูปที่ 5.1a) รวมไว้ในไฟล์ของโค้ดหลักของวงจรบวก 3 บิต

ตัวอย่างที่ 5.3 ฝึกประกาศใช้ Constant ใน Package โดยให้ประกาศใช้ Constant ของ N = 2 ของโโค้ดวงจรบวก 2 บิตในรูปที่ 2.31b) (ในบทที่ 2) ไว้ใน Package ชื่อ PACK_CONSTANT_N และดังรูปที่ 5.3a) ดังนั้นโโค้ดวงจรบวก N ในรูปที่ 5.3b) จะกลายเป็นโโค้ดของวงจรบวก 2 บิตทันทีที่เราเรียกใช้ package ชื่อ PACK_CONSTANT_N ซึ่งในที่นี่เราจะเขียนไว้ในบรรทัดที่ 5

```

2  --library IEEE;
3  --use IEEE.STD_LOGIC_1164.all;
4
5  package PACK_CONSTANT_N  is
6
7      constant N : integer := 2;
8
9  end PACK_CONSTANT_N;

```

N เป็นชนิดข้อมูล Integer ที่อยู่ใน Package ชื่อ Standard (IEEE Std 1076) ซึ่งไม่ต้องเรียกใช้ Package นี้เนื่องจากประกาศใช้โดย Default

5.3a) การประกาศใช้ Constant ไว้ใน Package โดยกำหนดให้ N = 2

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5 use WORK.PACK_CONSTANT_N.ALL;
6
7 entity ADDER_NBIT is
8     generic ( N : integer );
9     port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
10            C : out STD_LOGIC_VECTOR (N downto 0));
11 end ADDER_NBIT;
12
13 architecture BEHAVIORAL of ADDER_NBIT is
14 begin
15
16     C <= ('0' & A) + ('0' & B);
17
18 end BEHAVIORAL;

```

ต้องเรียกใช้ package ชื่อ STD_LOGIC_UNSIGNED ! เพราะต้องใช้ “+” กับชนิดข้อมูล std_logic_vector
ค่า N = 2 จึงต้องเรียกใช้ package ชื่อ PACK_CONSTANT_N

5.3b) โโค้ด VHDL ของวงจรบวกขนาด N = 2 บิตที่เขียนในรูปฟอร์มทั่วไป (วิธีนี้ไม่ต้องกำหนด N = 2 ที่ Generic)

รูปที่ 5.3 โโค้ด VHDL ของวงจรบวกขนาด 2 บิตที่เขียนในรูปฟอร์มทั่วไปโดยกำหนด N = 2 ไว้ใน package

ขอให้ผู้อ่านสังเกตว่าในตัวอย่างที่ 5.1 ถึงตัวอย่างที่ 5.3 นี้เป็นการเขียนโโค้ดที่มีเฉพาะส่วนประกาศใช้ Package เท่านั้น

5.2 Library

ไลบรารี (Library) เป็นส่วนที่เก็บรวบรวม Package ต่างๆ ซึ่งในการเขียนโโค้ดเพื่อออกแบบวงจรหรือไมโครเดลต่างๆ โดยทั่วๆ ไปแล้วจะต้องเรียกใช้ Package จากไลบรารีต่างๆ ดังนี้

- เรียกใช้ Package ชื่อ standard จาก Library ชื่อ std ซึ่งเป็น Resource library ที่ใช้ในภาษา VHDL
- เรียกใช้ Package ชื่อ work จาก Library ชื่อ work ซึ่งเป็นที่เก็บสิ่งที่เราออกแบบหรือถูกสร้างขึ้นโดยซอฟต์แวร์ทุก
- เรียกใช้ Package ชื่อ std_logic_1164 จาก Library ชื่อ ieee เมื่อใช้ชนิดข้อมูล std_logic หรือ std_logic_vector

การเรียกใช้ Library มีรูปแบบการเขียนทั่วไปดังนี้

```
library LIBRARY_NAME ;
```

โดยทั่วไปแล้วการเรียกใช้ Package ต่างๆ ที่อยู่ใน Library นั้นจะต้องเขียนไว้ที่ด้านบนสุดของโค้ด การเรียกใช้ Library และ Package (Using the package) จะใช้คำสั่ง Use (Use clause) มีรูปแบบการเขียนทั่วๆ ไปดังนี้

```
library LIBRARY_NAME ;
use LIBRARY_NAME . PACKAGE_NAME . PACKAGE_PARTS ;
```

ตัวอย่างที่เราพบเห็นจนชินตาแล้วก็คือ เมื่อใช้ชนิดข้อมูล std_logic หรือ std_logic_vector นั้นจะต้องเรียกใช้ Package ชื่อ std_logic_1164 ที่อยู่ใน Library ชื่อ ieee ซึ่งแสดงในบรรทัดที่ 2 และบรรทัดที่ 3 ดังตัวอย่างในรูปที่ 5.4 ซึ่งจะแตกต่างจาก การใช้ชนิดข้อมูลตาม IEEE Std 1076 เข่น bit แสดงดังรูปที่ 5.5 ที่เราอาจจะเกิดคำถามในใจว่าส่วนที่เรียกใช้ Library และ Package ในรูปที่ 5.5 นั้นหายไปไหน?

```
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DECODER2TO4 is
6     port ( AO,A1 : in STD_LOGIC;
7             DO,D1,D2,D3 : out STD_LOGIC);
8 end DECODER2TO4;
9
10 architecture BEHAVIORAL of DECODER2TO4 is
11 begin
12     DO <= (not A1)and(not AO);
13     D1 <= (not A1)and AO;
14     D2 <= A1 and(not AO);
15     D3 <= A1 and AO;
16 end BEHAVIORAL;
```

รูปที่ 5.4 โค้ด VHDL ของวงจร 2 to 4 Decoder ที่เขียนโดยใช้ชนิดข้อมูล std_logic

```
2 entity DECODER2TO4 is
3     port ( AO,A1 : in bit;
4             DO,D1,D2,D3 : out bit);
5 end DECODER2TO4;
6
7 architecture BEHAVIORAL of DECODER2TO4 is
8 begin
9     DO <= (not A1)and(not AO);
10    D1 <= (not A1)and AO;
11    D2 <= A1 and(not AO);
12    D3 <= A1 and AO;
13 end BEHAVIORAL;
```

รูปที่ 5.5 โค้ด VHDL ของวงจร 2 to 4 Decoder ที่เขียนโดยใช้ชนิดข้อมูล bit

ในการเขียนโค้ด VHDL สำหรับการออกแบบวงจรหรือโมเดลต่างๆ นั้น Package ชื่อ standard ที่อยู่ใน Library ชื่อ std และ Package ชื่อ work ที่อยู่ใน Library ชื่อ work จะถูกเรียกใช้โดยอัตโนมัติโดยกำหนดค่าว่างหน้าไว้แล้ว (Default) จึงไม่จำเป็นต้องเขียนส่วนที่เรียกใช้ Package ทั้ง 2 Package ดังกล่าวแต่อย่างใด

ใน Library ชื่อ ieee นั้นนอกจากจะมี Package ชื่อ std_logic_1164 แล้วยังมี Package อื่นๆ ของ IEEE เช่น numeric_std และ numeric_bit เป็นต้น นอกจากนี้แล้วยังมี Package ของ Synopsys ที่ไม่ใช่ Package มาตรฐาน (Non-standard Package) แต่เป็นที่นิยมใช้กันอย่างแพร่หลายรวมอยู่ด้วย ได้แก่ std_logic_signed, std_logic_unsigned และ std_logic_arith เป็นต้น

5.3 Subprograms

โปรแกรมย่อย (Subprogram) เป็นโค้ดย่อที่ถูกเรียกใช้บ่อย (Commonly used) โดยจะเขียนไว้ที่ส่วนกลาง เมื่อต้องการจะใช้ก็เรียกใช้ได้โดยไม่ต้องเสียเวลาวนนั่งเขียนโค้ดซ้ำๆ โดยทั่วไปแล้วการเขียนเป็น Subprogram จะได้เปรียบกว่าการเขียนเป็น Component คือ จะไม่ทำให้เกิดเป็นระดับใหม่ (New level) ของการออกแบบที่เป็นลำดับชั้น (Design hierarchy) ซึ่งขึ้นกับซอฟต์แวร์ทุกที่ใช้ เมื่อเขียน Subprogram ไว้ใน Package แล้ว Subprogram นี้จะเป็นโค้ดย่อของส่วนกลางที่ใช้ร่วมกัน ซึ่งอาจถูกเรียกใช้ Subprogram (Subprogram call) โดยโดยชอบของวงจรหรือโมเดลใดๆ ก็ได้ แต่ถ้าเขียน Subprogram ไว้ใน Architecture หรือ Process และก็จะถูกเรียกใช้ได้เฉพาะส่วนนั้นเท่านั้น คำสั่งที่ใช้ใน Subprogram จะเป็นคำสั่งชีเควนเชียล (Sequential statement) Subprogram มี 2 ชนิด คือ Function และ Procedure

5.3.1 Function

ฟังก์ชัน (Function) เป็นโปรแกรมย่อยที่ส่งค่าอาต์พุตกลับ (Return) ได้เพียงชุดเดียว และเมื่อเขียน Function ไว้ใน Package นั้นเราจะต้องเขียนทั้งส่วนของ Declaration และส่วนของ Body แต่ถ้าเขียนไว้ใน Architecture หรือ Process ก็จะมีเฉพาะส่วนของ Body เท่านั้น

- รูปแบบการประกาศใช้ Function หรือ Function declaration (เขียนไว้เฉพาะใน Package declaration) เป็นดังนี้

```
function FUNCTION_NAME ([object_class] OBJECT_NAME : OBJECT_TYPE
                      { ; [object_class] OBJECT_NAME : OBJECT_TYPE } )
                     return TYPE_NAME ;
```

โดยที่ [object_class] ใน Function declaration (และ Function body) อาจจะเป็น Constant หรือ Signal และถ้าไม่มีการระบุ [object_class] ก็จะเป็น Constant (ซึ่งถูกกำหนดล่วงหน้าไว้แล้วหรือถูก Default ไว้แล้ว) ส่วน OBJECT_NAME คือ ชื่อของ OBJECT และ TYPE_NAME คือ ชนิดข้อมูลของค่าผลลัพธ์ที่ส่งกลับ (Return)

- Function body เป็น Body ของ Function ที่เขียนไว้ในส่วน Body ของ Package หรือเขียนไว้ในส่วน Declaration ของ Architecture หรือ Declaration ของ Process ซึ่งมีรูปแบบการเขียนดังนี้

```
function FUNCTION_NAME ([object_class] OBJECT_NAME : OBJECT_TYPE
                      { ; [object_class] OBJECT_NAME : OBJECT_TYPE } )
                     return TYPE_NAME is
                     { FUNCTION_DECLARATIVE_ITEM }
begin
{ SEQUENTIAL_STATEMENT}
end [function] [FUNCTION_NAME] ;
```

โดยที่ 1) FUNCTION_DECLARATIVE_ITEM เป็นส่วนของ Declaration ดังนี้

- use clause
- Subprogram declaration และ Subprogram body ที่อยู่ภายใต้
- Type declaration และ/หรือ Subtype declaration
- Constant declaration และ/หรือ Variable declaration
- Attribute declaration และ/หรือ Attribute specification

2) SEQUENTIAL_STATEMENT เป็นคำสั่งซึ่ควนเขียน

3) คำสั่ง return เป็นคำสั่งซึ่ควนเขียนที่ใช้ในการสั่งให้ออกจาก Function ซึ่งมีรูปแบบการเขียนดังนี้

return (expression);

โดยที่ expression คือ สมการ หรือ Object หรือ ค่าต่างๆ (Litteral)

4) {...}หมายถึงจะมีหรือไม่มีก็ได้หรืออาจมีได้หลายตัว แต่ถ้าเป็น [...]หมายถึงจะมีหรือไม่มีก็ได้

- Function call เป็นการเรียกใช้ Function ในลักษณะที่เป็น Expression หรือสมการ ซึ่งในการเขียน Gödel VHDL ที่ผ่านมา นั้นเราใช้คลิ่นน้ำโดยตลอด โดยที่เราไม่รู้ว่าสิ่งที่เรานำขึ้นมาคือ Function call หรือการเรียกใช้ Function เช่น ฟังก์ชัน แปลง (Conversion functions) ชนิดข้อมูล และ Operator ต่างๆ ซึ่งก็คือ Function รูปแบบหนึ่ง เป็นต้น ตัวอย่างเช่น

A_int <= to_integer(A); -- Function call : Conversion function "to_integer (expression)"

QT := QT + 1; -- Function call : Operator "+"

รูปแบบทั่วไปของการเรียกใช้ Function เป็นดังนี้

FUNCTION_NAME ([OBJECT_TYPE =>] expression
{, [OBJECT_TYPE =>] expression});

ตัวอย่างที่ 5.4 ฝึกเขียน Function ที่ใช้แปลงชนิดข้อมูล integer เป็น std_logic_vector ไว้ใน Architecture, ใน Process และใน Package เพื่อใช้แทนการเรียกใช้ Package ชื่อ std_logic_unsigned ดังรูปที่ 5.6a) รูปที่ 5.6b) และรูปที่ 5.6d) ตามลำดับ โดยการเขียน Gödel วงจรบัน 16 (4 มิติ) แบบนับฐาน 2

ในรูปที่ 5.6a) แสดงการเขียน Function ไว้ใน Architecture โดยจะต้องเขียนไว้ในส่วนของ DECLARATIVE_ITEM (ดูข้อ 2.10.3) รูปที่ 5.6b) และการเขียน Function ไว้ใน Process โดยจะต้องเขียนไว้ในส่วนของ PROCESS DECLARATIVE PART (ดูข้อ 2.13.7) รูปที่ 5.6c) และการเขียน Function ไว้ใน Package และรูปที่ 5.6d) และการเรียกใช้ Package ที่มีเขียนขึ้นมาไว้เอง

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_NBITS is
6     generic (N : integer := 4);
7     port ( C,CLR : in STD_LOGIC;
8            Q : out STD_LOGIC_VECTOR (N-1 downto 0));
9 end COUNTER_NBITS;
10
11 architecture Behavioral of COUNTER_NBITS is
12     function STD_VECTOR (SIZE : integer; VALUE : integer) return std_logic_vector is
13         variable VTEMP : std_logic_vector (SIZE-1 downto 0);
14         begin
15             for i in 0 to VTEMP'high loop
16                 if (VALUE/(2**i) mod 2) = 1 then VTEMP(i) := '1';
17                 else VTEMP(i) := '0';
18                 end if;
19             end loop;
20             return VTEMP;      --Return value
21     end STD_VECTOR;
22     signal X : integer range 0 to (N**2)-1 ;
23 begin
24     process(C,CLR)          --counter : 0-15
25     begin
26         if CLR='1' then X <= 0;
27         elsif C'event and C='1' then
28             if (X >= (N**2)-1) then X <= 0;
29             else X <= X + 1;
30             end if;
31         end if;
32     end process;
33     Q <= STD_VECTOR (N,X); --Integer to binary decoder
34 end Behavioral;

```

5.6a) โค้ดวงจรนับ 4 บิตที่มีการเขียน Function ไว้ใน Architecture

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_NBITS is
6     generic (N : integer := 4);
7     port ( C,CLR : in STD_LOGIC;
8            Q : out STD_LOGIC_VECTOR (N-1 downto 0));
9 end COUNTER_NBITS;
10
11 architecture Behavioral of COUNTER_NBITS is
12 begin
13     process(C,CLR)          --counter : 0-15
14         function STD_VECTOR (SIZE : integer; VALUE : integer) return std_logic_vector is
15             variable VTEMP : std_logic_vector (SIZE-1 downto 0);
16             begin
17                 for i in 0 to VTEMP'high loop
18                     if (VALUE/(2**i) mod 2) = 1 then VTEMP(i) := '1';
19                     else VTEMP(i) := '0';
20                     end if;
21                 end loop;
22                 return VTEMP;      --Return value
23         end STD_VECTOR;
24         variable X : integer range 0 to (N**2)-1 ;
25     begin
26         if CLR='1' then X := 0;
27         elsif C'event and C='1' then
28             if (X >= (N**2)-1) then X := 0;
29             else X := X + 1;
30             end if;
31         end if;
32         Q <= STD_VECTOR (N,X); --Integer to binary decoder
33     end process;
34 end Behavioral;

```

5.6b) โค้ดวงจรนับ 4 บิตที่มีการเขียน Function ไว้ใน Process

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.all;
4
5 package PACK_TYPE_CONV is
6
7     function STD_VECTOR (SIZE : integer; VALUE : integer) return std_logic_vector;
8
9 end PACK_TYPE_CONV;
10
11 package body PACK_TYPE_CONV is
12
13     function STD_VECTOR (SIZE : integer; VALUE : integer) return std_logic_vector is
14         variable VTEMP : std_logic_vector (SIZE-1 downto 0);
15         begin
16             for i in 0 to VTEMP'high loop
17                 if (VALUE/(2**i) mod 2) = 1 then VTEMP(i) := '1';
18                 else VTEMP(i) := '0';
19                 end if;
20             end loop;
21             return VTEMP;           --Return value
22     end STD_VECTOR;
23
24 end PACK_TYPE_CONV;

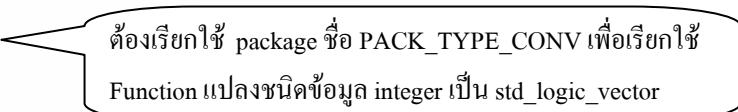
```

5.6c) โค้ดของ Function ที่ใช้แปลงชนิดข้อมูล integer เป็น std_logic_vector ที่ไว้ภายใน Package

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use WORK.PACK_TYPE_CONV.ALL;
5
6 entity COUNTER_NBITS is
7     generic (N : integer := 4);
8     port ( C,CLR : in STD_LOGIC;
9            Q : out STD_LOGIC_VECTOR (N-1 downto 0));
10 end COUNTER_NBITS;
11
12 architecture Behavioral of COUNTER_NBITS is
13     signal X : integer range 0 to (N**2)-1 ;
14 begin
15     process(C,CLR)          --counter : 0-15
16     begin
17         if CLR='1' then X <= 0;
18         elsif C'event and C='1' then
19             if (X >= (N**2)-1) then X <= 0;
20             else X <= X + 1;
21             end if;
22         end if;
23     end process;
24     Q <= STD_VECTOR (N,X); --Integer to binary decoder
25 end Behavioral;

```



ต้องเรียกใช้ package ชื่อ PACK_TYPE_CONV เพื่อเรียกใช้ Function แปลงชนิดข้อมูล integer เป็น std_logic_vector

5.6d) โค้ดวงจรนับ 4 บิตที่มีการเรียกใช้ Package ที่มี Function แปลงชนิดข้อมูล integer เป็น std_logic_vector อยู่ภายใน

รูปที่ 5.6 โค้ดวงจรนับ 4 บิตที่มีการเรียกใช้ Function แปลงชนิดข้อมูล integer เป็น std_logic_vector

จากโค้ดในรูปที่ 5.6 นั้นเราสามารถเขียนโค้ดแปลงชนิดข้อมูลโดยใช้ Process แทน Function ได้เช่นกันโดยแสดงดังรูปที่ 5.7 โดยคำสั่ง Process 1 คำสั่งจะแทนตำแหน่งที่มีการเรียกใช้ Function ได้ 1 ตำแหน่ง ดังนั้นในกรณีที่มีการเขียนโค้ดและต้องมีการเรียกใช้ Function หากยตำแหน่ง การเรียกใช้ Function จึงมีลักษณะกว่าและทำให้เขียนโค้ดสั้นกว่า

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_NBITS is
6   generic (N : integer := 4);
7   port ( C,CLR : in STD_LOGIC;
8         Q : out STD_LOGIC_VECTOR (N-1 downto 0));
9 end COUNTER_NBITS;
10
11 architecture Behavioral of COUNTER_NBITS is
12   signal X : integer range 0 to (N**2)-1 ;
13 begin
14   process(C,CLR)           --counter : 0-15
15   begin
16     if CLR='1' then X <= 0;
17     elsif C'event and C='1' then
18       if (X >= (N**2)-1) then X <= 0;
19       else X <= X + 1;
20       end if;
21     end if;
22   end process;
23
24   process(X)           --Integer to std_logic_vector conversion
25   variable VTEMP : std_logic_vector (N-1 downto 0);
26 begin
27   for i in 0 to VTEMP'high loop
28     if (X/(2**i) mod 2) = 1 then VTEMP(i) := '1';
29     else VTEMP(i) := '0';
30     end if;
31   end loop;
32   Q <= VTEMP;
33   end process;
34 end Behavioral;

```

รูปที่ 5.7 โค้ดวงจรนับ 4 บิตที่ใช้คำสั่ง Process ในการแปลงชนิดข้อมูล integer เป็น std_logic_vector

และในทำนองเดียวกันจากโค้ดในรูปที่ 5.6 เราสามารถเขียนโค้ดแปลงชนิดข้อมูล integer เป็น std_logic_vector โดยใช้ Component ดังรูปที่ 5.8 แทน Function ได้เช่นกัน โดยที่โค้ดของวงจรนับ 4 บิตที่ได้แสดงดังรูปที่ 5.9

```

----- Component : Integer to std_logic_vector conversion -----
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity STD_VECTOR is
6   generic (N : integer := 16);
7   port ( DATA_IN : in integer range 0 to (2**N)-1;
8         DATA_OUT : out STD_LOGIC_VECTOR (N-1 downto 0));
9 end STD_VECTOR;
10
11 architecture Behavioral of STD_VECTOR is
12 begin
13   process(DATA_IN)           --Integer to std_logic_vector conversion
14     variable VTEMP : std_logic_vector (N-1 downto 0);
15     begin
16       for i in 0 to VTEMP'high loop
17         if (DATA_IN/(2**i) mod 2) = 1 then VTEMP(i) := '1';
18         else VTEMP(i) := '0';
19         end if;
20       end loop;
21       DATA_OUT <= VTEMP;
22     end process;
23   end Behavioral;

```

5.8a) Component ที่ใช้แปลงชนิดข้อมูล integer เป็น std_logic_vector

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_NBITS is
6     generic (N : integer := 4);
7     port ( C,CLR : in STD_LOGIC;
8            Q : out STD_LOGIC_VECTOR (N-1 downto 0));
9 end COUNTER_NBITS;
10
11 architecture Behavioral of COUNTER_NBITS is
12     component STD_VECTOR
13         generic (N : integer := 16);
14         port ( DATA_IN : in integer range 0 to (2**N)-1;
15                DATA_OUT : out STD_LOGIC_VECTOR (N-1 downto 0));
16     end component;
17     signal X : integer range 0 to (N**2)-1 ;
18 begin
19     process(C,CLR)          --counter : 0-15
20     begin
21         if CLR='1' then X <= 0;
22         elsif C'event and C='1' then
23             if (X >= (N**2)-1) then X <= 0;
24             else X <= X + 1;
25             end if;
26         end if;
27     end process;
28     --Integer to std_logic_vector conversion
29 PROCESS_CONV : STD_VECTOR generic map (N => 4)
30             port map (DATA_IN => X,
31                         DATA_OUT => Q);
32 end Behavioral;

```

5.8b) โค้ดวงจรนับ 4 บิตที่ใช้ Component แปลงชนิดข้อมูล integer เป็น std_logic_vector

รูปที่ 5.8 โค้ดวงจรนับ 4 บิตที่ใช้ Component ในการแปลงชนิดข้อมูล integer เป็น std_logic_vector

- Function overloading คือ Function ที่มีชื่อซ้ำกัน เช่น Function ของ Logical operator ตาม IEEE Std 1164 ดังรูปที่ 5.9 มีชื่อ Function ไปซ้ำกับชื่อของ Logical operator ตาม IEEE Std 1076 จึงทำให้ Operator นี้ถูกขยายขอบเขตให้ใช้ได้ กับชนิดข้อมูล เช่น bit, bit_vector, std_logic (std_ulogic) และ std_logic_vector (std_logic_vector) เป็นต้น นั่นก็ แสดงว่า Operator นี้ได้ทำ “เกิน” หรือ Overload จากหน้าที่ที่ได้定义ไว้เดิมที่ได้กำหนดล่วงหน้าไว้แล้ว (Predefined) ใน IEEE Std 1076 ดังนั้นเราจึงเรียก Operator เหล่านี้ว่า Overloaded operator หรือ Overloaded function

```

-- overloaded logical operators ( with optimizing hints )

FUNCTION "and" ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
BEGIN
    RETURN (and_table(l, r));
END "and";
FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
BEGIN
    RETURN (not_table ( and_table(l, r)));
END "nand";
FUNCTION "or" ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
BEGIN
    RETURN (or_table(l, r));
END "or";
FUNCTION "nor" ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
BEGIN
    RETURN (not_table ( or_table( l, r )));
END "nor";

```

รูปที่ 5.9 ตัวอย่างบางส่วนของ Function body ของ Logical operator ตาม IEEE Std 1164

ตัวอย่างที่ 5.5 ฝึกเขียน Function ของ Overloaded operator ที่เป็นเครื่องหมาย “+” และ “-” เพื่อใช้กับวงจรบาก-ลับ 4 บิตที่เป็นชนิดข้อมูล std_logic_vector ไว้ใน Package ชื่อ PACK_ARITH_OPERATOR เพื่อใช้แทน Package ชื่อ std_logic_unsigned โดยรูปที่ 5.10a) แสดงการเขียน Function ไว้ใน Package และรูปที่ 5.10b) แสดงการเรียกใช้ Package

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.all;
4 use IEEE.NUMERIC_STD.all;
5
6 package PACK_ARITH_OPERATOR is
7     function "+" (L,R: std_logic_vector) return std_logic_vector;
8     function "-" (L,R: std_logic_vector) return std_logic_vector;
9 end PACK_ARITH_OPERATOR;
10
11 package body PACK_ARITH_OPERATOR is
12     function "+" (L,R: std_logic_vector) return std_logic_vector is
13         variable RESULT : std_logic_vector ((L'length - 1) downto 0);
14     begin
15         RESULT := std_logic_vector(unsigned(L) + unsigned(R));
16         return RESULT;
17     end "+";
18
19     function "-" (L,R: std_logic_vector) return std_logic_vector is
20         variable RESULT : std_logic_vector ((L'length - 1) downto 0);
21     begin
22         RESULT := std_logic_vector(unsigned(L) - unsigned(R));
23         return RESULT;
24     end "-";
25 end PACK_ARITH_OPERATOR;

```

5.10a) การเขียน Function ของ Overloaded operator ที่เป็นเครื่องหมาย “+” และ “-” ไว้ใน Package

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use WORK.PACK_ARITH_OPERATOR.ALL;
5
6 entity ADD_SUB_4BIT_PACK is
7     Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
8            OPER : in STD_LOGIC;
9            C : out STD_LOGIC_VECTOR (4 downto 0));
10 end ADD_SUB_4BIT_PACK;
11
12 architecture Behavioral of ADD_SUB_4BIT_PACK is
13 begin
14     C <= ('0' &A) + ('0' &B) when OPER = '0' else
15     ('0' &A) - ('0' &B);
16 end Behavioral;

```

5.10b) การเรียกใช้ Function ของ Overloaded operator ที่เป็นเครื่องหมาย “+” และ “-” ไว้ใน Package

รูปที่ 5.10 โค้ดวงจรบาก-ลับที่มีการเรียกใช้ใน Package ชื่อ PACK_ARITH_OPERATOR ที่เราเขียนขึ้นมาใช้เอง

5.3.2 Procedure

Procedure เป็นโปรแกรมย่อยที่ส่งค่ากลับ (Return) ได้หลายค่าโดยผ่านทาง Interface หรือไม่ส่งค่ากลับก็ได้ เมื่อเขียน Procedure ไว้ใน Package นั้นเราจะต้องเขียนทั้งส่วนของ Declaration และส่วนของ Body แต่ถ้าเขียนไว้ใน Architecture หรือ Process ก็จะมีเฉพาะส่วนของ Body เท่านั้น

- รูปแบบการประกาศใช้ Procedure หรือ Procedure declaration (เขียนไว้เฉพาะใน Package declaration) เป็นดังนี้

```
procedure PROCEDURE_NAME ([object_class] OBJECT_NAME : [MODE] OBJECT_TYPE
{ ; [object_class] OBJECT_NAME : [MODE] OBJECT_TYPE } );
```

โดยที่ [object_class] อาจเป็น Constant, Signal, Variable หรือ file ถ้า MODE เป็น in และไม่มีการระบุ [object_class] ก็ให้อ่านว่าเป็น Constant แต่ถ้า MODE เป็น inout หรือ out และไม่มีการระบุ [object_class] ก็ให้อ่านว่าเป็น Variable

- Procedure body เป็นส่วนที่เขียนไว้ใน Body ของ Package หรือในส่วน Declaration ของ Architecture หรือในส่วน Declaration ของ Process มีรูปแบบการเขียนดังนี้

```
procedure PROCEDURE_NAME ([object_class] OBJECT_NAME : [MODE] OBJECT_TYPE
{ ; [object_class] OBJECT_NAME : [MODE] OBJECT_TYPE } ) is
{ PROCEDURE_DECLARATIVE_ITEM }
begin
{ SEQUENTIAL_STATEMENT}
end [procedure] [PROCEDURE_NAME];
```

โดยที่ 1) PROCEDURE_DECLARATIVE_ITEM เป็นส่วนของ Declaration ดังนี้

- use clause
- Subprogram declaration และ Subprogram body ที่อยู่ภายใต้
- Type declaration และ/หรือ Subtype declaration
- Constant declaration และ/หรือ Variable declaration
- Attribute declaration และ/หรือ Attribute specification

2) SEQUENTIAL_STATEMENT เป็นคำสั่งซึ่คานเชียล

3) คำสั่ง return; (ไม่มี expression) เป็นคำสั่งซึ่คานเชียลที่ใช้ในการสั่งให้ออกจาก Procedure (ซึ่งจะต่างจาก Function)

4) {...} หมายถึงจะมีหรือไม่มีได้หรืออาจมีได้หลายตัว แต่ถ้าเป็น [...] หมายถึงจะมีหรือไม่มีได้

- Procedure call เป็นการเรียกใช้ Procedure ในลักษณะคำสั่ง (Statement) Concurrent procedure calls และ Sequential procedure calls จะเหมือนกัน ซึ่งรูปแบบทั่วไปการเรียกใช้ Procedure เป็นดังนี้

```
PROCEDURE_NAME ([ NAME => ] EXPRESSION { , [ NAME => ] EXPRESSION } );
```

ตัวอย่างที่ 5.6 ฝึกการเขียน Function และ Procedure ของวงจร Full adder 1 บิตไว้ใน Package เพื่อนำไปสร้างเป็นวงจรvak 4 บิตแสดงดังรูปที่ 5.11a) และรูปที่ 5.11b) ตามลำดับ

```

2 -----Package : FUNCTION-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 package PACK_FUNCTION_ADDER is
7     function ADD(A,B, Cin : STD_LOGIC ) return STD_LOGIC_VECTOR;
8 end PACK_FUNCTION_ADDER;
9
10 package body PACK_FUNCTION_ADDER is
11     function ADD(A,B, Cin : STD_LOGIC ) return STD_LOGIC_VECTOR is
12         variable S, Cout : STD_LOGIC;
13         variable RESULT : STD_LOGIC_VECTOR (1 downto 0);
14     begin
15         S := A xor B xor Cin;
16         Cout := (A and B) or (A and Cin) or (B and Cin);
17         RESULT := Cout & S;
18         return RESULT;
19     end ADD;
20 end PACK_FUNCTION_ADDER;
21
22 -----Main code-----
23 library IEEE;
24 use IEEE.STD_LOGIC_1164.ALL;
25 use work.PACK_FUNCTION_ADDER.all;
26
27 entity ADD4BIT is
28 port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
29        Cin : in STD_LOGIC;
30        S : out STD_LOGIC_VECTOR (3 downto 0);
31        Cout: out STD_LOGIC );
32 end ADD4BIT;
33
34 architecture Behavioral of ADD4BIT is
35     signal S0, S1, S2, S3 : STD_LOGIC_VECTOR (1 downto 0);
36 begin
37     S0 <= ADD ( A(0), B(0), Cin );
38     S1 <= ADD ( A(1), B(1), S0(1) );
39     S2 <= ADD ( A(2), B(2), S1(1) );
40     S3 <= ADD ( A(3), B(3), S2(1) );
41     S <= S3(0) & S2(0) & S1(0) & S0(0);
42     Cout <= S3(1);
43 end Behavioral;

```

5.11a) โค้ดของ Function ที่ไว้ภายใน Package ที่เป็นรวมไว้ในโค้ดหลัก (Main code) ของวงจรบวก 4 บิต

```

2 -----Package : PROCEDURE-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 package PACK_PROCEDURE_ADDER is
7     procedure ADD(A,B, CIN : in STD_LOGIC; C : out STD_LOGIC_VECTOR (1 downto 0) );
8 end PACK_PROCEDURE_ADDER;
9
10 package body PACK_PROCEDURE_ADDER is
11     procedure ADD(A,B, Cin : in STD_LOGIC; C : out STD_LOGIC_VECTOR (1 downto 0) ) is
12         variable S, Cout : STD_LOGIC;
13     begin
14         S := A xor B xor CIN;
15         COUT := (A and B) or (A and Cin) or (B and Cin);
16         C := Cout & S;
17     end ADD;
18 end PACK_PROCEDURE_ADDER;
19
20 -----Main code-----
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23 use work.PACK_PROCEDURE_ADDER.all;
24

```

```

25 entity ADD4BIT is
26 port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
27      Cin : in STD_LOGIC;
28      S : out STD_LOGIC_VECTOR (3 downto 0);
29      Cout : out STD_LOGIC);
30 end ADD4BIT;
31
32 architecture Behavioral of ADD4BIT is
33 begin
34   process (A,B,Cin)
35     variable S0, S1, S2, S3 : STD_LOGIC_VECTOR (1 downto 0);
36   begin
37     ADD ( A(0), B(0), Cin, S0 );
38     ADD ( A(1), B(1), S0(1), S1 );
39     ADD ( A(2), B(2), S1(1), S2 );
40     ADD ( A(3), B(3), S2(1), S3 );
41     S <= S3(0) & S2(0) & S1(0) & S0(0);
42     Cout <= S3(1);
43   end process;
44 end Behavioral;

```

5.11b) โค้ดของ Procedure ที่ไว้ภายใน Package ที่เขียนรวมไว้ในโค้ดหลัก (Main code) ของวงจรบวก 4 บิต

รูปที่ 5.11 โค้ดของวงจร Full adder 1 บิตที่เขียนไว้ใน Package ที่เขียนรวมไว้ในโค้ดหลัก (Main code) ของวงจรบวก 4 บิต

จากโค้ดของวงจร Full adder 1 บิตในรูปที่ 5.11a) นั้นเราจะเห็นได้อย่างชัดเจนว่า Function call ในบรรทัดที่ 37-40 มีลักษณะเป็นสมการหรือ Expression ในขณะที่รูปที่ 5.11b) นั้นเราจะเห็นได้อย่างชัดเจนว่า Procedure call ในบรรทัดที่ 37-40 มีลักษณะเป็นคำสั่งหรือ Statement

การทดลองที่ 5.1 การประมวลใช้ Constant ไว้ใน Package

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับการประมวลใช้ Constant ไว้ใน Package และการเรียกใช้ Package
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างวงจรบวก 3 บิตและโปรแกรมลงชิป CPLD และ FPGA

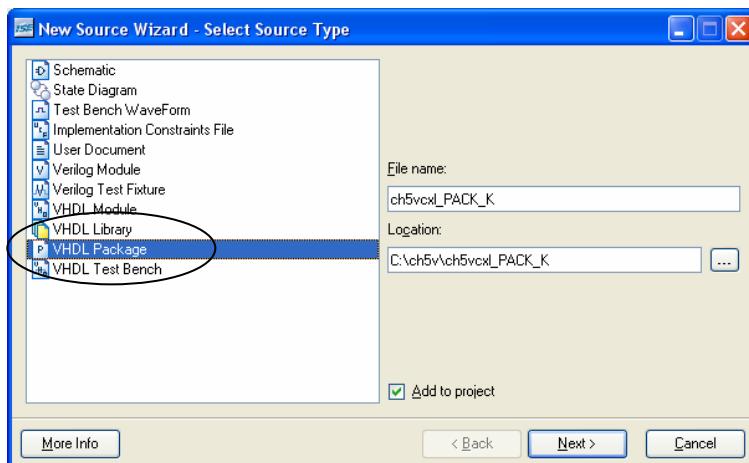
อุปกรณ์ทดลอง

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

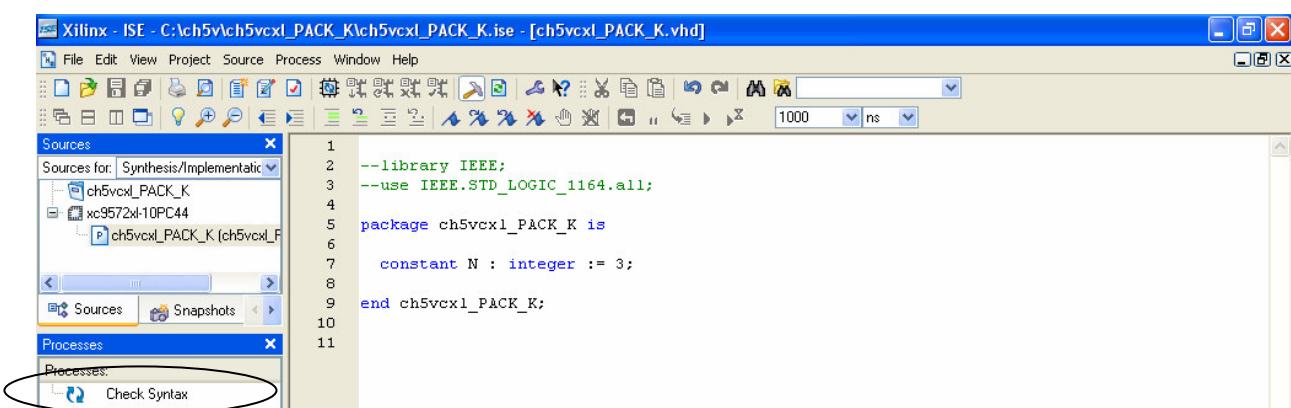
1 ออกแบบวงจรบวก 3 บิตด้วย CPLD

การทดลองนี้จะฝึกการประมวลใช้ Constant ไว้ใน Package โดยการเขียนโค้ดวงจรบวก 3 บิตจากวงจรบวก N บิต โดยจะประมวลใช้ Constant $N = 3$ ไว้ใน Package การสร้าง Package นี้จะเหมือนกับขั้นตอน Design entry ทุกประการ

1.1) สร้าง Package ใน Folder ชื่อ ch5v มี Project Name และ Source File ชื่อ ch5vcxl_K โดยที่หน้าต่าง New Source Wizard ให้คลิกที่ VHDL Package ดังรูปที่ L1.1 คลิก Next คลิก Finish คลิก Next 2 ครั้ง คลิก Finish ทำการแก้ไขโค้ดดังรูปที่ L1.2 คลิก บันทึกไฟล์ คลิกขวาที่ Check Syntax และคลิก Run ถ้าไม่มีข้อผิดพลาดก็ถือว่าการสร้าง Package แล้วเสร็จสมบูรณ์



รูปที่ L1.1 หน้าต่าง New Source Wizard-Select Source Type



รูปที่ L1.2 โค้ด VHDL ของ Package ที่ประมวลใช้ Constant โดย $N = 3$

1.2) เขียนโค้ดวงจรบวก 3 บิตจากวงจรบวก N บิต สร้างไฟล์ใน Folder ชื่อ ch5v โดยใช้ Project Name และ Source File ชื่อ ex5_1vcx1 เมื่อเขียนโค้ดแล้วเสร็จจะได้ดังรูปที่ L1.3 โดยในรูปที่ L1.3 นี้จะยังสังเคราะห์วงจรไม่ได้จนกว่าเราจะทำการ Add Source ไฟล์ของ Package ชื่อ ch5vcx1_K เข้าไปเลียก่อน

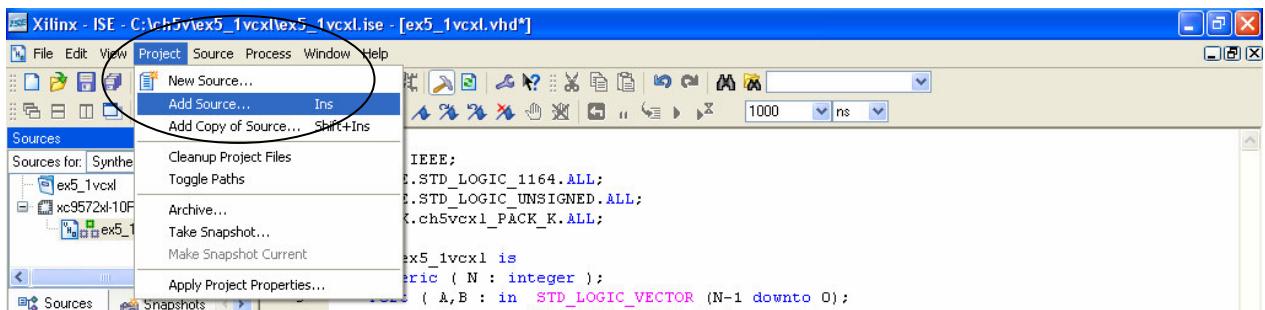
```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_UNSIGNED.ALL;
4 use WORK.ch5vcx1_PACK_K.ALL;
5
6 entity ex5_1vcx1 is
7     generic ( N : integer );
8     port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
9             C : out STD_LOGIC_VECTOR (N downto 0));
10 end ex5_1vcx1;
11
12 architecture Behavioral of ex5_1vcx1 is
13 begin
14
15     C <= ('0' & A) + ('0' & B);
16
17 end Behavioral;
18

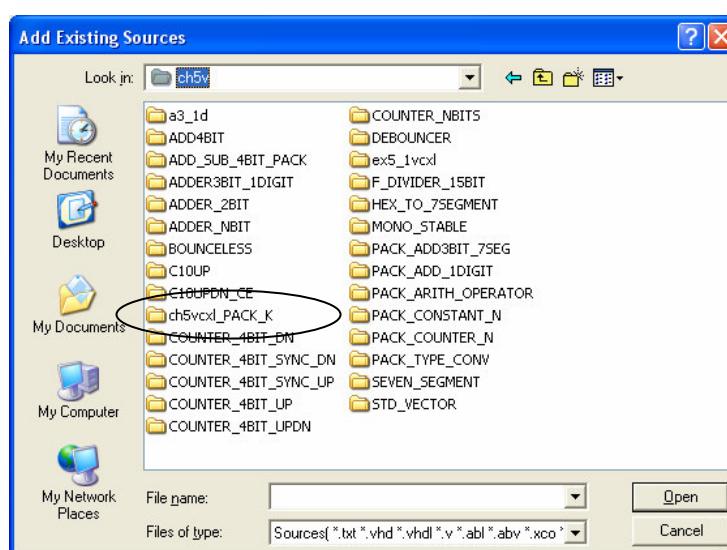
```

รูปที่ L1.3 โค้ดของวงจรบวก N บิต

1.3) วิธี Add Source ในภายหลังนั้นได้อธิบายไปแล้วในบทที่ 2 ทำได้โดยคลิก Add Source ดังรูปที่ L1.4 แล้วจะได้นำมาต่อ Add Existing Source จากนั้นให้คลิกไฟล์ Package ชื่อ ch5vcx1_PACK_K ที่อยู่ใน Folder ชื่อ ch5v ทั้งรูปที่ L1.5

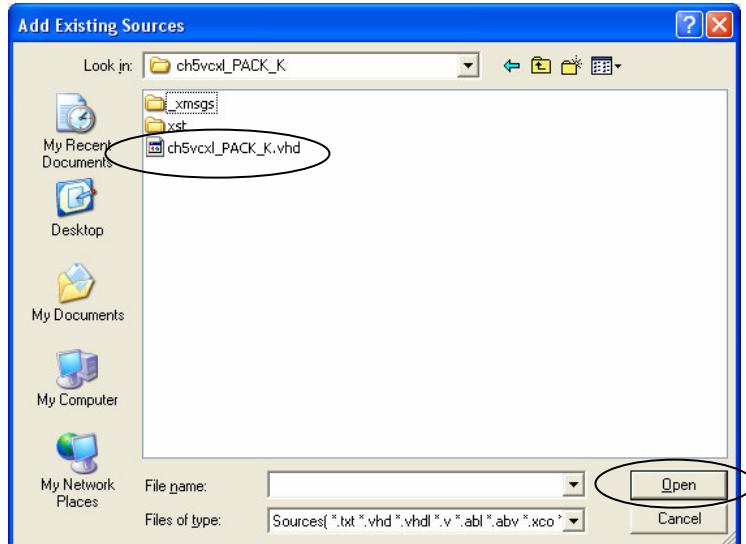


รูปที่ L1.4 เริ่มขั้นตอนการ Add Source

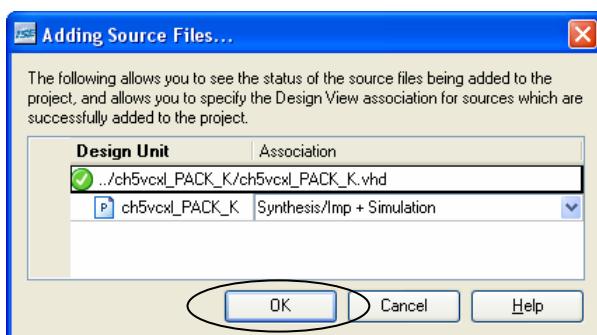


รูปที่ L1.5 หน้าต่าง Add Existing Source (ตำแหน่งไฟล์อาจอยู่ที่อื่นของ Folder ชื่อ ch5v ก็ได้)

ในรูปที่ L1.5 ดับเบลคลิกไฟล์ชื่อ ch5vcxl_PACK_K แล้วจะได้ดังรูปที่ L1.6 คลิกที่ไฟล์ชื่อ ch5vcxl_PACK_K.vhd และคลิก Open จะได้ดังรูปที่ L1.7 คลิกปุ่ม OK ก็จะถือว่าการ Add Source ไฟล์ Package ชื่อ ch5vcxl_K แล้วเสร็จ คลิก บันทึกไฟล์และทำการตรวจสอบความถูกต้อง (Check Syntax) ถ้าไม่มีข้อผิดพลาดก็ถือว่าขั้นตอน Design entry แล้วเสร็จ



รูปที่ L1.6 หน้าต่าง Add Existing Source



รูปที่ L1.7 หน้าต่าง Adding Source files

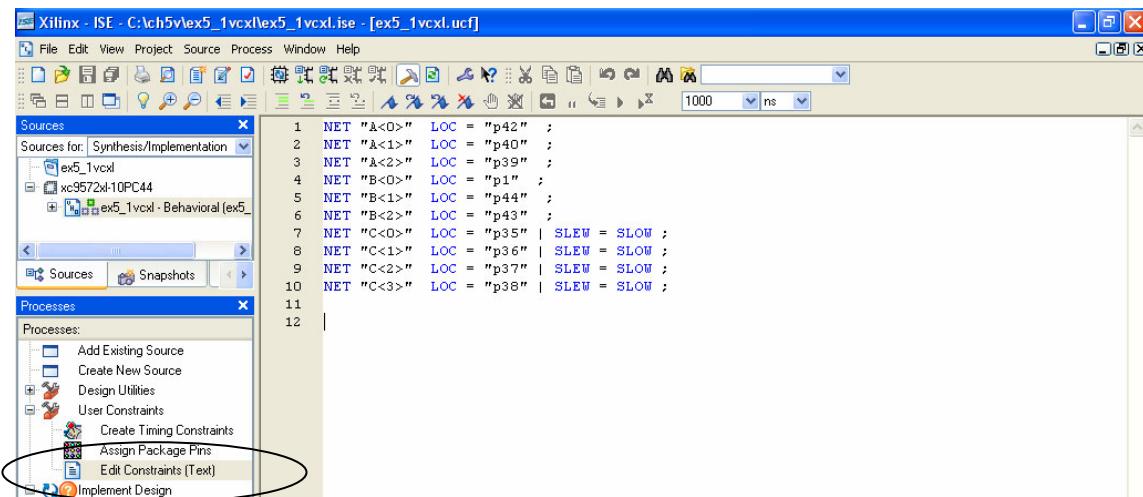
การกำหนดขาสัญญาณต่างๆ ของ CPLD จะใช้ปุ่มกด PB1-PB6 เป็นอินพุตและ LED1–LED4 เป็นเอาต์พุต กล่าวคือ

A(2)= PB1 = INPUT = p39	B(2)= PB4(Slide SW2)= INPUT= p43	C(3)= LED1= OUTPUT = p38
A(1)= PB2 = INPUT = p40	B(1)= PB5(Slide SW3)= INPUT= p44	C(2)= LED2 = OUTPUT = p37
A(0)= PB3(Slide SW1)= INPUT= p42	B(0)= PB6(Slide SW4)= INPUT = p1	C(1)= LED3 = OUTPUT = p36

C(0)= LED4 = OUTPUT = p35

จากนั้นให้พิมพ์ขาสัญญาณต่างๆ ใน Assign Package Pins แต่ถ้าการกำหนดขาสัญญาณต่างๆ ของ CPLD ผ่านทาง Assign Package Pins ไปที่หน้าต่าง PACE ไม่ได้ ก็ให้กำหนดขาสัญญาณต่างๆ ของ CPLD ผ่านทาง Edit Constraints (Text) แทน โดยดับเบลคลิกที่ Edit Constraints (Text) จากนั้นพิมพ์ขาสัญญาณต่างๆ ดังรูปที่ L1.8 แล้วคลิก บันทึกไฟล์

หลังจากโปรแกรมจะที่ออกแบบลงชิป CPLD แล้วให้ทดสอบกับปุ่ม PB1–PB6 และให้สังเกตคุณลักษณะที่ LED1–LED4 ว่าให้ลองเอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงทำการบันทึกผลการทดลอง

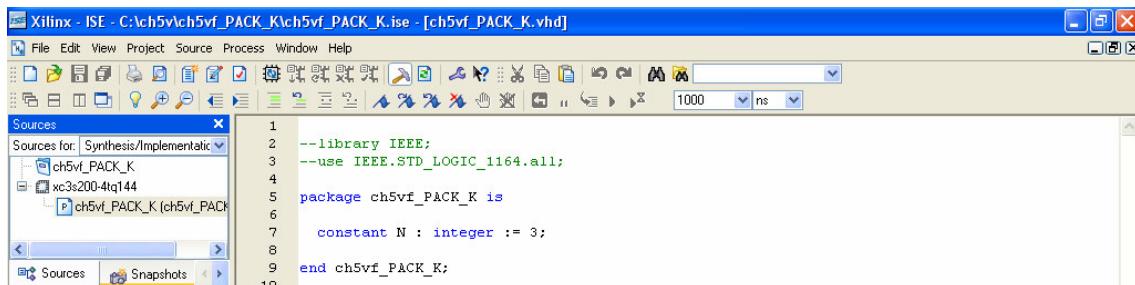


รูปที่ L1.8 การกำหนดขาสัญญาณต่างๆ ของ CPLD ผ่านทาง Edit Constraints (Text)

2 ออกแบบวงจรบวก 3 บิตด้วย FPGA

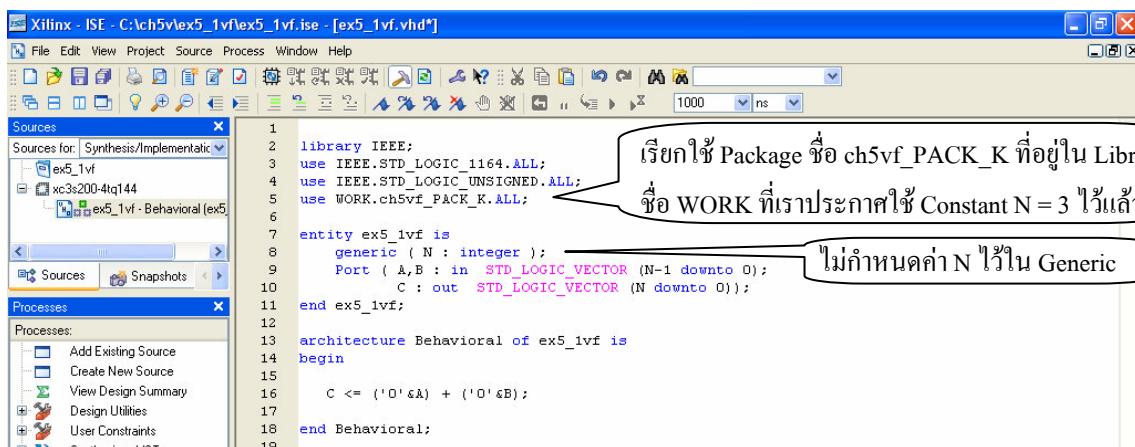
การทดลองนี้จะฝึกการประกาศใช้ Constant ไว้ใน Package โดยการเขียนโค้ดวงจรบวก 3 บิตจากวงจรบวก N บิต โดยจะประกาศใช้ Constant N = 3 ไว้ใน Package โดยที่ขั้นตอนการทดลอง FPGA จะเหมือนกับ CPLD ในข้อ 1 ทุกประการ

2.1) สร้าง Package ใน Folder ชื่อ ch5v มี Project Name และ Source File ชื่อ ch5vf_K โดยเขียนโค้ดดังรูปที่ L2.1



รูปที่ L2.1 โค้ด VHDL ของ Package ที่ประกาศใช้ Constant โดย N = 3

2.2) สร้างไฟล์โดยเขียนโค้ดวงจรบวก 2 บิตจากวงจรบวก N บิตไว้ใน Folder ชื่อ ch5v โดยใช้ Project Name และ Source File ชื่อ ex5_1vf เมื่อเขียนโค้ดแล้วเสร็จจะได้ดังรูปที่ L2.2

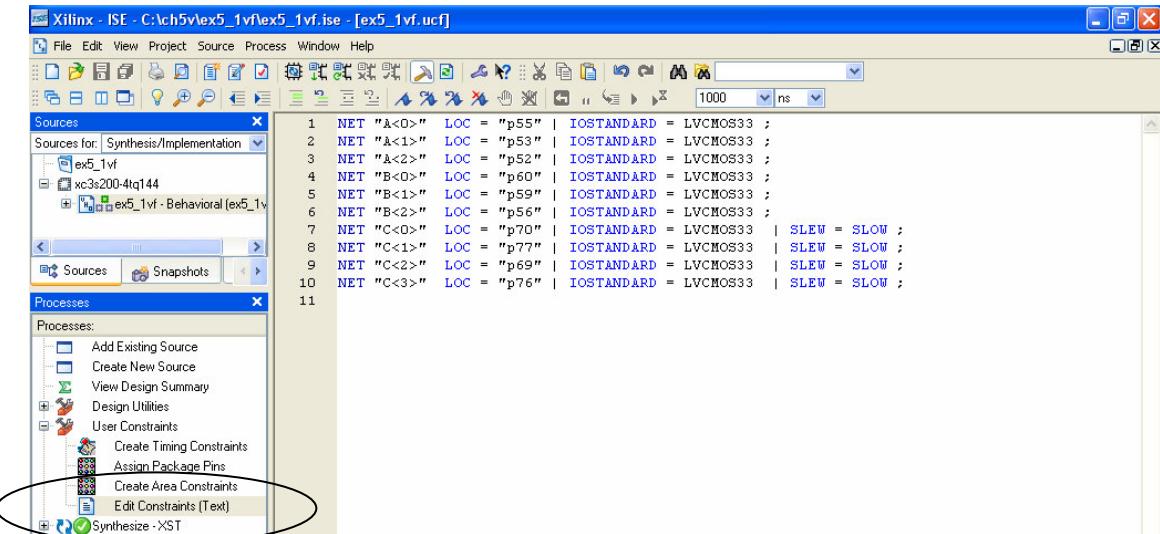


2.3) ทำการ Add Source ไฟล์ Package ชื่อ ch5vf_PACK_K ที่อยู่ใน Folder ชื่อ ch5v เช้าไปในไฟล์ของโค้ดของวงจรบวก 3 บิต ชื่อ ex5_1vf ที่ทำไว้แล้วในข้อ 2.2

การกำหนดขาสัญญาณต่างๆ ของ FPGA นี้ จะใช้ Dip SW1-Dip SW6 เป็นอินพุต และ LED L0-L3 เป็นเอาต์พุต กล่าวคือ

A(2) = Dip SW1 = INPUT = p52	B(2) = Dip SW4 = INPUT = p56	C(3) = L3 = OUTPUT = p76
A(1) = Dip SW2 = INPUT = p53	B(1) = Dip SW5 = INPUT = p59	C(2) = L2 = OUTPUT = p69
A(0) = Dip SW3 = INPUT = p55	B(0) = Dip SW6 = INPUT = p60	C(1) = L1 = OUTPUT = p77
		C(0) = L0 = OUTPUT = p70

จากนี้ให้พิมพ์ขาสัญญาณต่างๆ ใน Assign Package Pins แต่ถ้าการกำหนดขาสัญญาณต่างๆ ของ FPGA ผ่านทาง Assign Package Pins ไปที่หน้าต่าง PACE ไม่ได้ ก็ให้กำหนดขาสัญญาณต่างๆ ของ FPGA ผ่านทาง Edit Constraints (Text) แทน โดยดับเบิลคลิกที่ Edit Constraints (Text) จากนั้นพิมพ์ขาสัญญาณต่างๆ ดังรูปที่ L2.3 แล้วคลิก บันทึกไฟล์



รูปที่ L2.3 การกำหนดขาสัญญาณต่างๆ ของ CPLD ผ่านทาง Edit Constraints (Text)

หลังจากโปรแกรมจะที่ออกแบบลงชิป FPGA แล้วให้ทดสอบ ON-OFF Dip SW1-Dip SW6 และให้สังเกตคุณที่ LED L0-L3 ว่าให้หลอดจิกเอาต์พุตติดสว่างเป็นไปตามทฤษฎีหรือไม่ หากนั้นจึงทำการบันทึกผลการทดสอบ

การทดลองที่ 5.2 การประมวลใช้ Component ไว้ใน Package

วัตถุประสงค์

- 1) เพื่อฝึกการประมวลใช้ Component ไว้ใน Package และการเรียกใช้ Package
- 2) เพื่อทดลองใช้ ISE WebPACK 8.1i สร้างวงจรบวก 3 บิตและโปรแกรมลงชิป CPLD และ FPGA

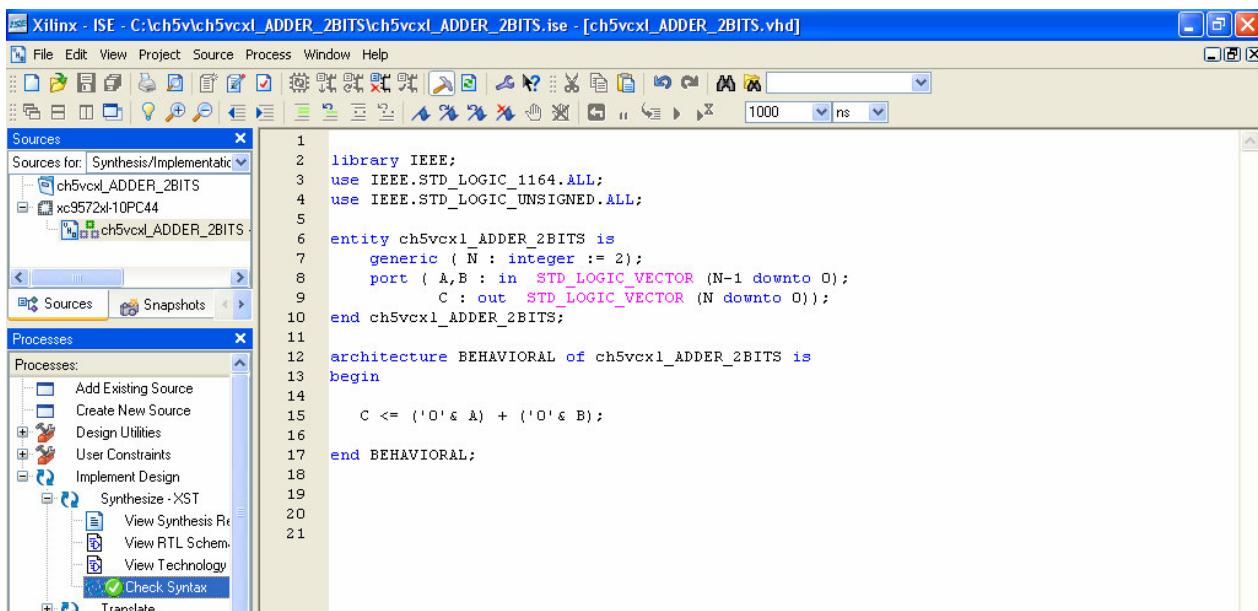
อุปกรณ์ทดลอง

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

1 วงจรบวก 3 บิตที่แสดงผลทางเขียนเชกเมนต์ 1 หลักด้วย CPLD

การสร้าง Package และเรียกใช้ Package โดยเขียนโค้ดของวงจรบวก 3 บิตที่แสดงผลทางเขียนเชกเมนต์ มีขั้นตอนดังนี้

- 1.1) เขียนโค้ดของวงจรบวก 2 บิตและวงจรบวก 3 บิตที่แสดงผลทางเขียนเชกเมนต์และรูปที่ L1.1 และรูปที่ L1.2 สร้างไฟล์ใน Folder ชื่อ ch5v โดยใช้ Project Name และ Source File ชื่อ ch5vcx1_ADDER_2BITS และชื่อ ch5vcx1_HEX_TO_7SEGMENT ตามลำดับ โดยที่รูปที่ L1.1 นี้จะเขียนโค้ดของวงจรบวก 2 บิตในรูปของวงจรบวก N บิตโดยมี $N = 2$ ซึ่งวงจรนี้จะเป็นวงจรบวก 3 บิตก็ต่อเมื่อกำหนดค่า $N = 3$
- 1.2) นำโค้ดในรูปที่ L1.1 และรูปที่ L1.2 มาประมวลใช้ Component ใน Package สร้างไฟล์ใน Folder ชื่อ ch5v โดยใช้ Project Name และ Source File ชื่อ ch5vcx1_PACK_ADD_7SEG เมื่อสร้าง Package เสร็จแล้วจะได้รูปที่ L1.3 คลิก บันทึกไฟล์ จากนั้นคลิกขวาที่ Check Syntax และคลิก Run ถ้าไม่มีข้อผิดพลาดถือว่าขั้นตอนการสร้าง Package เสร็จสมบูรณ์



รูปที่ L1.1 โค้ดของวงจรบวก N บิต

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ch5vcxl_HEX_TO_7SEGMENT is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (6 downto 0));
end ch5vcxl_HEX_TO_7SEGMENT;

architecture Behavioral of ch5vcxl_HEX_TO_7SEGMENT is
begin
    --gfedcba
    Y <= "1110001" when A="1111" else --F
        "1110001" when A="1110" else --E      Y(0)=a
        "1011110" when A="1101" else --d      ---
        "0111001" when A="1100" else --C Y(5)=f | Y(1)=b
        "1111100" when A="1011" else --b      --- Y(6)=g
        "1110111" when A="1010" else --A Y(4)=e | Y(2)=c
        "1101111" when A="1001" else --9      ---
        "1111111" when A="1000" else --8      Y(3)=d
        "0000111" when A="0111" else --7
        "1111101" when A="0110" else --6
        "1101101" when A="0101" else --5
        "1100110" when A="0100" else --4
        "1001111" when A="0011" else --3
        "1011011" when A="0010" else --2
        "0000110" when A="0001" else --1
        "0111111";                         --0
end Behavioral;

```

รูปที่ L1.2 โค้ดของวงจรดอร์หัสตัวแสดงผลเลขฐานเซกเมนต์

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

package ch5vcxl_PACK_ADD_7SEG is

    component ch5vcxl_ADDER_2BITS
        generic ( N : integer := 2 );
        port ( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
               C : out STD_LOGIC_VECTOR (N downto 0));
    end component;
    component ch5vcxl_HEX_TO_7SEGMENT
        Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
               Y : out STD_LOGIC_VECTOR (6 downto 0));
    end component;
end ch5vcxl_PACK_ADD_7SEG;

```

รูปที่ L1.3 การประกาศใช้ Component ใน Package

1.3) ขั้นตอนการออกแบบวงจร (Design entry) ก็จะเหมือนกับการออกแบบทั่วๆ ไป โดยสร้างไฟล์ใน Folder ชื่อ ch5v โดยใช้ Project Name และ Source File ชื่อ ex5_2vcxl เมื่อเสร็จแล้วจะได้ดังรูปที่ L1.4 โค้ดในรูปที่ L1.4 นี้จะยังสังเคราะห์วงจรไม่ได้จนกว่าเราจะทำการ Add Source ไฟล์ของ Package ชื่อ ch5vcxl_PACK_ADD_7SEG และไฟล์ของ Component เข้าไปทั้งหมดเสียก่อน ซึ่งจากโค้ดรูปที่ L1.4 ขอให้ผู้อ่านสังเกตว่าในบรรทัดที่ 3 เราต้องกำหนด N => 3 เพราะว่าเป็นวงจรบวก 3 บิตไม่ใช่วงจรบวก 2 บิตตามที่เรากำหนดค่า N = 2 ในคำสั่ง Generic

1.4) ทำการ Add Source ซึ่งในกรณีนี้จะมีทั้งหมด 3 ไฟล์ คือไฟล์ Package ชื่อ ch5vcxl_PACK_ADD_7SEG และไฟล์ของ Component ชื่อ ch5vcxl_ADDER_2BITS และ ch5vcxl_HEX_TO_7SEGMENT ที่อยู่ใน Folder ชื่อ ch5v เมื่อ Add Source เรียบร้อยแล้วให้บันทึกไฟล์และ Check Syntax ถ้าไม่มีข้อผิดพลาดถือว่าขั้นตอน Design entry ของวงจรบวก 3 บิตที่แสดงผลทางฐานเซกเมนต์ 1 หลักแล้วเสร็จสมบูรณ์

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use WORK.ch5vcx1_PACK_ADD_7SEG.ALL;
5
6 entity ex5_2vcx1 is
7     Port ( A,B : in STD_LOGIC_VECTOR (2 downto 0);
8             Y : out STD_LOGIC_VECTOR (6 downto 0);
9             COMMON : out STD_LOGIC);
10 end ex5_2vcx1;
11
12 architecture Behavioral of ex5_2vcx1 is
13     signal C : STD_LOGIC_VECTOR (3 downto 0);
14 begin
15     COMMON <= '0';
16     -----
17     IC1 : ch5vcx1_ADDER_2BITS generic map (N => 3)      --ADDER_2BIT=>ADDER_3BIT
18         port map (A,B,C);
19     -----
20     IC2 : ch5vcx1_HEX_TO_7SEGMENT port map (C,Y);
21
22 end Behavioral;

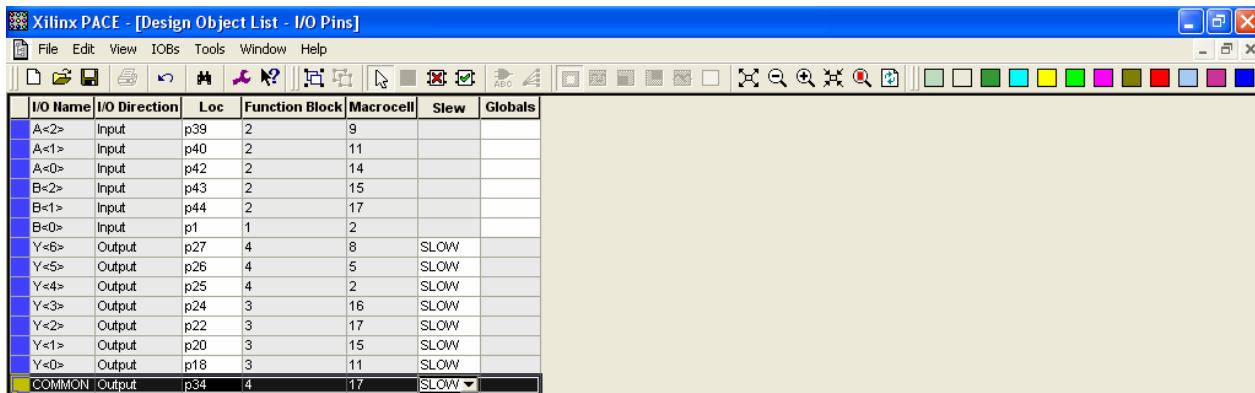
```

รูปที่ L1.4 โค้ด VHDL ของวงจรบวก 3 บิตที่แสดงผลทางชีวนเซกเมนต์

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB6 เป็นอินพุต และตัวแสดงผลชีวนเซกเมนต์แบบแอดเดอร์ 3 บิตที่ 1 (คือ DIGIT1) และขา COMMON หลักที่ 1 เป็นเอาต์พุต กล่าวว่าคือ

A(2) = PB1 = INPUT = p42	Y(0) = a = OUTPUT = p27
A(1) = PB2 = INPUT = p43	Y(1) = b = OUTPUT = p26
A(0) = PB3(Slide SW1) = INPUT = p42	Y(2) = c = OUTPUT = p25
B(2) = PB4(Slide SW2) = INPUT = p43	Y(3) = d = OUTPUT = p24
B(1) = PB5(Slide SW3) = INPUT = p44	Y(5) = f = OUTPUT = p20
B(0) = PB6(Slide SW4) = INPUT = p1	Y(4) = e = OUTPUT = p22
COMMON = DIGIT1 = OUTPUT = p34	Y(6) = g = OUTPUT = p18

โดยพิมพ์ใน Assign Package Pins ดังรูปที่ L1.6 สรุปได้ดังนี้



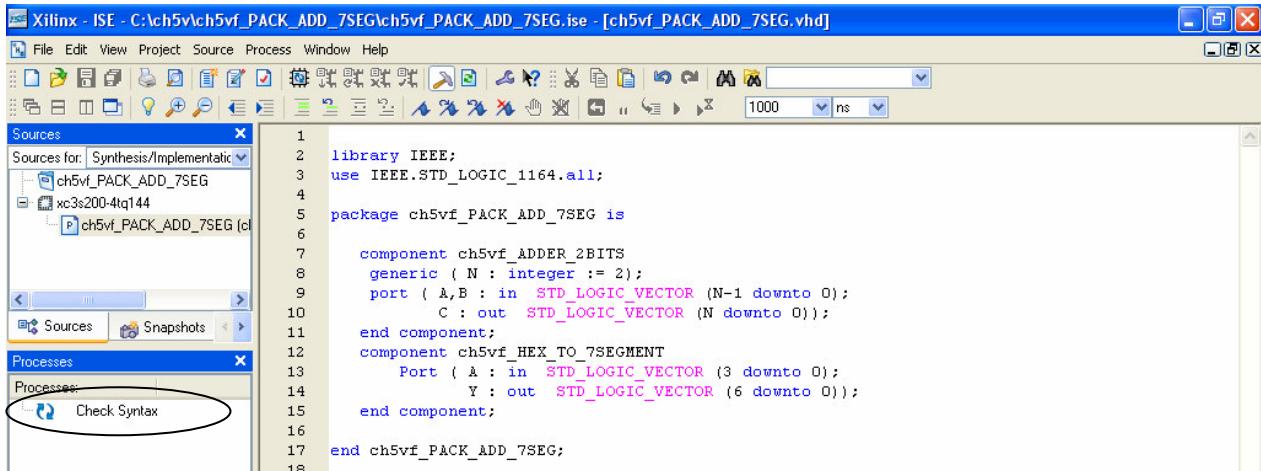
รูปที่ L1.6 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิป CPLD แล้วให้ทดสอบกับปุ่ม PB1-PB6 และให้สังเกตผลที่ตัวแสดงผลชีวนเซกเมนต์ว่ามีส่วนติดสว่างโดยให้ออกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นจึงบันทึกผลการทดลอง

2 วงจรบวก 3 บิตที่แสดงผลทางเลขฐานเทกเมนต์ 1 หลักด้วย FPGA

ฝึกการสร้าง Package และเรียกใช้ Package โดยเขียนโค้ดวงจรบวก 3 บิตที่แสดงผลทางเลขฐานเทกเมนต์ โดยที่ขึ้นตอนการทดลอง FPGA จะเหมือนกับ CPLD ในข้อ 1 ทุกประการ

- 2.1) เขียนโค้ดของวงจรบวก 2 บิตและวงจรอัตโนมัติที่แสดงผลทางเลขฐานเทกเมนต์ สร้างไฟล์ใน Folder ชื่อ ch5v โดยใช้ Project Name และ Source File ชื่อ ch5vf_ADDER_2BITS และชื่อ ch5vf_HEX_TO_7SEGMENT ตามลำดับ
- 2.2) นำโค้ดในข้อ 2.1) มาประยุกต์ใช้ Component ใน Package สร้างไฟล์ใน Folder ชื่อ ch5v โดยใช้ Project Name และ Source File ชื่อ ch5vf_PACK_ADD_7SEG เมื่อสร้าง Package เสร็จแล้วจะได้รูปที่ L2.1



รูปที่ L2.1 การประยุกต์ใช้ Component ใน Package

- 2.3) ขั้นตอนการออกแบบวงจร (Design entry) ก็จะเหมือนกับการออกแบบทั่วๆ ไป โดยสร้างไฟล์ใน Folder ชื่อ ch5v โดยใช้ Project Name และ Source File ชื่อ ex5_2vf เมื่อเสร็จแล้วจะได้รูปที่ L2.2

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use WORK.ch5vf_PACK_ADD_7SEG.ALL;

entity ex5_2vf is
    Port ( A,B : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC_VECTOR (6 downto 0);
           COMMON : out STD_LOGIC);
end ex5_2vf;
begin
    architecture Behavioral of ex5_2vf is
        signal C : STD_LOGIC_VECTOR (3 downto 0);
    begin
        COMMON <= '0';
        -----
        IC1 : ch5vf_ADDER_2BITS generic map ( N => 3 )      -- ADDER_2BIT=>ADDER_3BIT
              port map ( A,B,C );
        -----
        IC2 : ch5vf_HEX_TO_7SEGMENT port map ( C,Y );
    end Behavioral;

```

รูปที่ L2.2 โค้ด VHDL ของวงจรบวก 3 บิตที่แสดงผลทางเลขฐานเทกเมนต์

- 2.4) ทำการ Add Source ซึ่งในกรณีจะมีพื้นที่ 3 ไฟล์ คือ ไฟล์ Package ชื่อ ch5vcxl_PACK_K และไฟล์ของ Component ชื่อ ch5vcf_ADDER_2BITS และ ch5vf_HEX_TO_7SEGMENT ที่อยู่ใน Folder ชื่อ ch5v

การกำหนดขาสัญญาณต่างๆ จะใช้ Dip SW1-Dip SW6 เป็นอินพุต และตัวแสดงผลเซเวนเซกเมนต์แบบแคร็อกคร่าวม หลักที่ 1 (คือ DIGIT1) และขา COMMON หลักที่ 1 เป็นเอาต์พุต ก่อร่างคือ

A(2) = Dip SW1 = INPUT = p52	Y(0) = a = OUTPUT = p40
A(1) = Dip SW2 = INPUT = p53	Y(1) = b = OUTPUT = p35
A(0) = Dip SW3 = INPUT = p55	Y(2) = c = OUTPUT = p32
B(2) = Dip SW4 = INPUT = p56	Y(3) = d = OUTPUT = p30
B(1) = Dip SW5 = INPUT = p59	Y(5) = f = OUTPUT = p27
B(0) = Dip SW6 = INPUT = p60	Y(4) = e = OUTPUT = p25
COMMON = DIGIT1 = OUTPUT = p31	Y(6) = g = OUTPUT = p23

โดยพิมพ์ใน Assign Package Pins ดังรูปที่ L2.3 สรุปได้ดังนี้

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
A<0>	Input	p55	BANK4	LVCMS33	N/A	3.30					Unknown		
A<1>	Input	p53	BANK5	LVCMS33	N/A	3.30					Unknown		
A<2>	Input	p52	BANK5	LVCMS33	N/A	3.30					Unknown		
B<0>	Input	p60	BANK4	LVCMS33	N/A	3.30					Unknown		
B<1>	Input	p59	BANK4	LVCMS33	N/A	3.30					Unknown		
B<2>	Input	p56	BANK4	LVCMS33	N/A	3.30					Unknown		
COMMON	Output	p31	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<0>	Output	p40	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<1>	Output	p35	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<2>	Output	p32	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<3>	Output	p30	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<4>	Output	p27	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<5>	Output	p25	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		
Y<6>	Output	p23	BANK6	LVCMS33	N/A	3.30		SLOW			Unknown		

รูปที่ L2.3 Assign Package Pins

หลังจากโปรแกรมจะลงชิป FPGA แล้วให้ทดลอง ON-OFF Dip SW1-Dip SW6 และให้สังเกตคุณที่ตัวแสดงผลเซเวนเซกเมนต์ว่ามีส่วนติดสว่างโดยให้ลองเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ หากนั่นจึงบันทึกผลการทดลอง

การทดลองที่ 5.3 การเขียน Function

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับการเขียน Function ไว้ในส่วนของ Architecture, Process และ Package
- 2) เพื่อใช้ ISE WebPACK 8.1i สร้างวงจรนับ 16 (4 บิต) แบบนับขึ้น

อุปกรณ์ทดลอง

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

1 ออกแบบวงจรนับ 16 (4 บิต) แบบนับขึ้นด้วย CPLD

การทดลองนี้เราจะเขียนโค้ดวงจรนับ 16 (4 บิต) แบบนับขึ้นจากตัวอย่างที่ 5.4 โดยเขียน Function เพื่อใช้แปลงชนิดข้อมูล integer เป็น std_logic_vector ไว้ใน Architecture, ใน Process และใน Package เพื่อใช้แทนการเรียกใช้ Package ชื่อ std_logic_unsigned และดังรูปที่ 5.6a) รูปที่ 5.6b) และรูปที่ 5.6d) ตามลำดับ ในการเขียนโค้ดวงจรนับ 16 (4 บิต) แบบนับขึ้น

1.1) เขียนโค้ดไว้ใน Folder ชื่อ ch5v และใช้ Project Name และ Source File ชื่อ ex5_3vcxl โดยนำโค้ดในรูปที่ 5.6a มาแก้ไขโดยเปลี่ยนชื่อ Entity เป็น ex5_3vcxl เสร็จแล้วจะได้ดังรูปที่ L1.1 ซึ่งในกรณีนี้จะเป็นการเขียน Function ไว้ใน Architecture คลิก บันทึกไฟล์ คลิกขวาที่ Check Syntax และคลิก Run ถ้าไม่มีข้อผิดพลาดถือว่าขั้นตอน Design entry แล้วเสร็จจากนั้นจำลองการทำงานโดยใช้ Source File ชื่อ ex5_3vcxl_tb1 ได้ดังรูปที่ L1.2 และคุณจะเป็นตามทฤษฎีหรือไม่

```

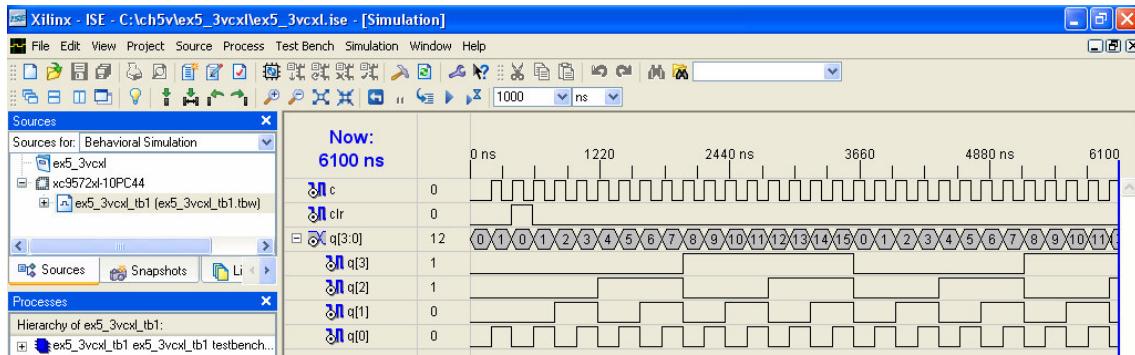
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ex5_3vcxl is
    generic (N : integer := 4);
    port ( C,CLR : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (N-1 downto 0));
end ex5_3vcxl;

architecture Behavioral of ex5_3vcxl is
    function STD_VECTOR (SIZE : integer; VALUE : integer) return std_logic_vector is
        variable VTEMP : std_logic_vector (SIZE-1 downto 0);
        begin
            for i in 0 to VTEMP'high loop
                if (VALUE/(2**i) mod 2) = 1 then VTEMP(i) := '1';
                else VTEMP(i) := '0';
                end if;
            end loop;
            return VTEMP; --Return value
        end STD_VECTOR;
        signal X : integer range 0 to (N**2)-1 ;
begin
    process(C,CLR)          --counter : 0-15
    begin
        if CLR='1' then X <= 0;
        elsif C'event and C='1' then
            if (X >= (N**2)-1) then X <= 0;
            else X <= X + 1;
            end if;
        end if;
    end process;
    Q <= STD_VECTOR (N,X); --Integer to binary converter
end Behavioral;

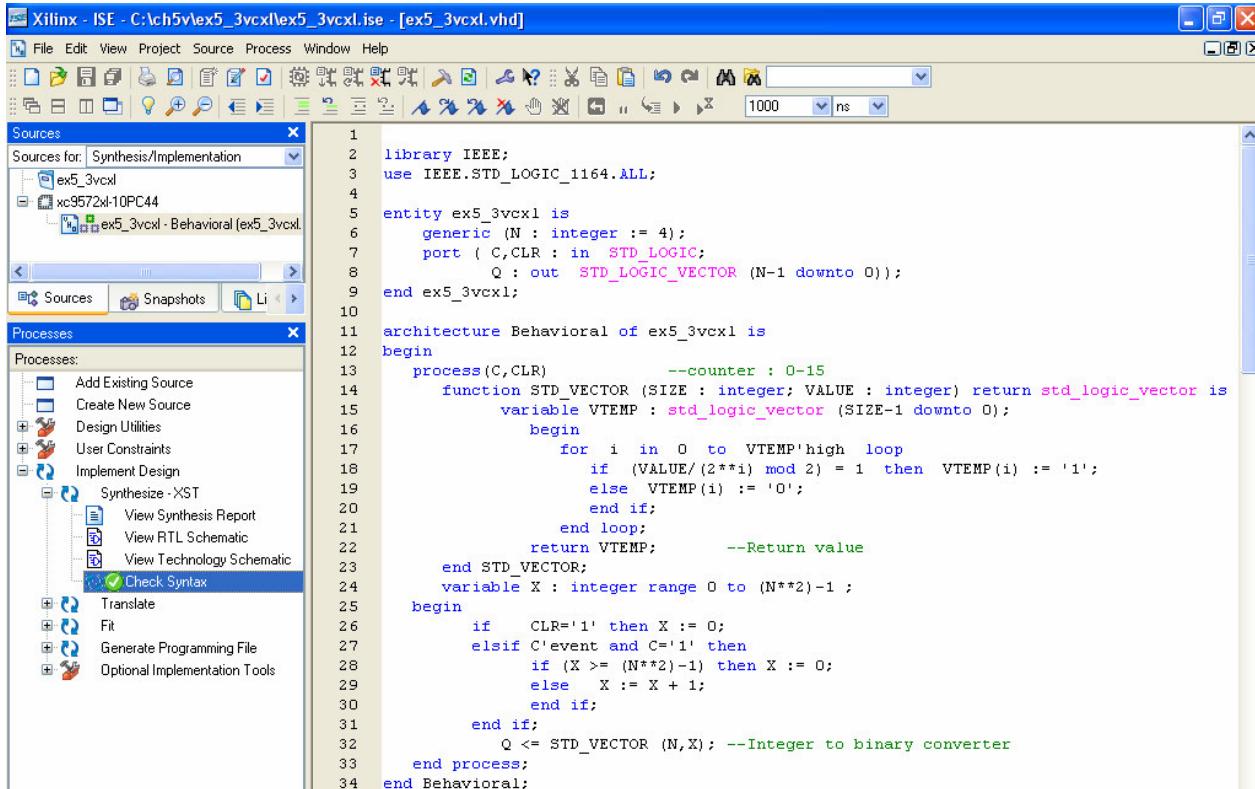
```

รูปที่ L1.1 โค้ดวงจรนับ 4 บิตที่เขียน Function ไว้ใน Architecture



รูปที่ L1.2 ผล Simulation ของโค้ดงานนับ 4 บิตที่เขียน Function ไว้ใน Architecture

1.2 แก้ไขโค้ดในรูปที่ L1.1 ใหม่โดยเขียน Function ไว้ใน Process เสร็จแล้วจะได้ดังรูปที่ L1.3 คลิก บันทึกไฟล์ คลิกขวาที่ Check Syntax แล้วคลิก Run ถ้าไม่มีข้อผิดพลาดถือว่าขั้นตอน Design entry แล้วเสร็จสมบูรณ์ จากนั้นจำลองการทำงานโดยใช้ Source File ชื่อ ex5_3vcxl_tb1 เช่นเดิม แล้วจะได้ดังรูปที่ L1.2 เช่นเดิมหรือไม่



รูปที่ L1.3 โค้ดงานนับ 4 บิตที่เขียน Function ไว้ใน Process

1.3 เขียน Function ไว้ใน Pakage โดยในที่นี่เราจะใช้ Folder ชื่อ ch5v และใช้ Project Name และ Source File ชื่อ ch5vcxl_PACK_TYPE_CONV เสร็จแล้วจะได้ดังรูปที่ L1.4 คลิก บันทึกไฟล์ คลิกขวาที่ Check Syntax แล้วคลิก Run ถ้าไม่มีข้อผิดพลาดถือว่าขั้นตอน Design entry ของ Package แล้วเสร็จ

ทำการแก้ไขโค้ดในรูปที่ L1.3 ใหม่ได้ดังรูปที่ L1.5 แล้ว Add source File ชื่อ ch5vcxl_PACK_TYPE_CONV เสร็จแล้วคลิก บันทึกไฟล์ คลิกขวาที่ Check Syntax แล้วคลิก Run ถ้าไม่มีข้อผิดพลาดถือว่าขั้นตอน Design entry แล้วเสร็จ จากนั้นจำลองการทำงานโดยใช้ Source File ชื่อ ex5_3vcxl_tb1 เช่นเดิม แล้วจะได้ดังรูปที่ L1.2 เช่นเดิมหรือไม่

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
package ch5vcxl_PACK_TYPE_CONV is
    function STD_VECTOR (SIZE : integer; VALUE : integer) return std_logic_vector;
end ch5vcxl_PACK_TYPE_CONV;
package body ch5vcxl_PACK_TYPE_CONV is
    function STD_VECTOR (SIZE : integer; VALUE : integer) return std_logic_vector is
        variable VTEMP : std_logic_vector (SIZE-1 downto 0);
        begin
            for i in 0 to VTEMP'high loop
                if (VALUE/(2**i) mod 2) = 1 then VTEMP(i) := '1';
                else VTEMP(i) := '0';
                end if;
            end loop;
            return VTEMP; --Return value
    end STD_VECTOR;
end ch5vcxl_PACK_TYPE_CONV;

```

รูปที่ L1.4 โค้ดของ Function ที่ใช้แปลงชนิดข้อมูล integer เป็น std_logic_vector ที่ไว้ภายใน Package

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use WORK.ch5vcxl_PACK_TYPE_CONV.ALL;
entity ex5_3vcxl is
    generic (N : integer := 4);
    port ( C,CLR : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (N-1 downto 0));
end ex5_3vcxl;
architecture Behavioral of ex5_3vcxl is
    signal X : integer range 0 to (N**2)-1;
begin
    process(C,CLR)
        begin
            if CLR='1' then X <= 0;
            elsif C'event and C='1' then
                if (X >= (N**2)-1) then X <= 0;
                else X <= X + 1;
                end if;
            end if;
    end process;
    Q <= STD_VECTOR (N,X); --Integer to binary converter
end Behavioral;

```

รูปที่ L1.5 โค้ดของรันนับ 4 บิตที่มีการเรียกใช้ Package ที่มี Function แปลงชนิดข้อมูล integer เป็น std_logic_vector

2 ออกแบบวงจรนับ 16 (4 บิต) แบบนับขึ้นด้วย FPGA

ขั้นตอนการทดลอง FPGA จะเหมือนกับ CPLD ในข้อ 1 ทุกประการ เพียงแต่เปลี่ยนชื่อไฟล์ที่มีคำว่า “vcxl” เป็น “vf”

หมายเหตุ ขอให้ผู้อ่านสังเกตว่าในการทดลองที่ 5.3 นี้เราจะทำการจำลองการทำงานของวงจรเพียงอย่างเดียวโดยไม่มีการกำหนดขาและค่าวน้ำหลอดไฟลงใน CPLD หรือ FPGA แต่อย่างใด เพราะว่าเรากำลังออกแบบวงจรนับ 4 บิต ดังนั้นวงจรนี้จะทำงานไม่ถูกต้องเนื่องจาก Bouncing จากปุ่มกด การที่จะทำให้วงจรนับนี้ทำงานได้ถูกต้องจำเป็นต้องใช้วงจรโน�สตเตเบิล

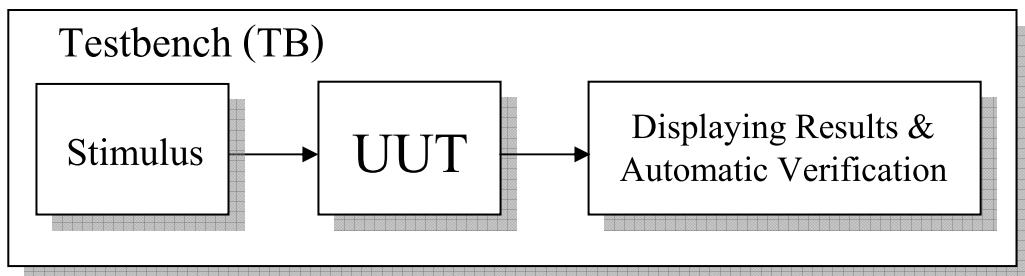
บทที่ 6

Testbench และ Package TEXTIO

6.1 Testbench

Testbench เป็นเครื่องมือทางซอฟต์แวร์ที่ใช้ภาษา HDL (VHDL หรือ Verilog) เขียนขึ้นเพื่อใช้ในการตรวจความถูกต้อง (Verification) โค้ดของวงจรดิจิตอลที่ออกแบบด้วยการจำลองการทำงาน (Simulation) ต่อไปเราเรียกโค้ดของวงจรหรือระบบดิจิตอลที่ออกแบบว่า Unit under test (UUT) หรือ Design under test (DUT) Testbench จะทำหน้าที่สร้างสัญญาณทดสอบ (Stimulus) หรือ Test vector ได้โดยอิสระเพื่อป้อนให้กับ UUT หรือ DUT ที่อยู่ในรูปของ Component โดยอาศัยพุตที่ได้จะถูกนำไปแสดงผลในรูปแบบต่างๆ เช่น รูปคลื่นสัญญาณ (Waveform) หรือ รายงานข้อผิดพลาด หรือไฟล์ข้อมูล เป็นต้น โดยที่การทำงานทั้งหมดนี้จะกระทำการใน Testbench เอง ดังนั้น Entity ของ Testbench จึงไม่มี I/O Port ที่ติดต่อกันวงจรภายนอก

การเขียน Testbench ที่มีความซับซ้อนนั้นเราอาจนำเอาต์พุตจริง (Actual results) ไปเปรียบเทียบกับเอาต์พุตตามทฤษฎี ซึ่งเป็นเอาต์พุตที่คาดการว่าจะได้ (Expected results) ถ้าหากผลที่ได้ไม่ตรงกันก็แสดงว่ามีข้อผิดพลาดเกิดขึ้น และให้รายงานผลโดยอัตโนมัติ (ทางหน้าต่าง Transcript ที่อยู่ด้านล่างสุดของหน้าต่าง Xilinx-ISE) ซึ่งวิธีนี้จะดีกว่าการสังเกตข้อผิดพลาดจากรูปคลื่นสัญญาณเอาต์พุตที่ได้จากวิชั่นจำลองการทำงานโดยตรง โดยใช้ Waveform Editor ในการทดลองในบทที่ 3 และบทที่ 4 ซึ่งถ้าเป็นวงจรขนาดใหญ่ก็จะยิ่งทำให้เสียเวลามาก บล็อก ໂດະແກນของ Testbench แสดงดังรูปที่ 6.1



รูปที่ 6.1 บล็อกໂດະແກນของ Testbench

คำสั่งเบื้องต้นที่ควรทราบเพิ่มเติมในการเขียน Testbench ได้แก่ คำสั่ง wait, wait for และ assert ซึ่งมีความหมายดังนี้

- คำสั่ง wait เป็นคำสั่งให้ Process หยุดทำงาน
- คำสั่ง wait for เป็นคำสั่งให้ Process หยุดรอตามเวลาที่กำหนด เช่น wait for 100 ns;
- คำสั่ง assert เป็นคำสั่งที่ทำงานเมื่อเงื่อนไขมูลคุณ (Condition) เป็นเท็จ ใน การเขียน Testbench นั้นเราจะใช้ร่วมกับคำสั่ง report และ severity เพื่อรายงานระดับความรุนแรงของข้อผิดพลาด โดยมีรูปแบบการเขียนดังนี้

```

assert CONDITION_EXPRESSION
report TEXT_STRING
severity SEVERITY_LEVEL;
  
```

โดยที่ ชนิดข้อมูล SEVERITY_LEVEL จะบอกระดับความรุนแรงดังนี้ NOTE, WARNING, ERROR และ FAILURE

ตัวอย่าง 6.1 โค้ดแอนด์เกต 2 อินพุตแสดงดังรูปที่ 6.2a) มี Testbench และแสดงดังรูปที่ 6.2b) โดยผลการทำงานแสดงดังรูปที่ 6.2c) และรูปที่ 6.2d) โดยรูปที่ 6.2d) นั้นจะไม่มีการใส่ค่าเริ่มต้นให้ Signal A และ Signal B (บรรทัดที่ 14-15) ที่ป้อนให้อินพุต

```

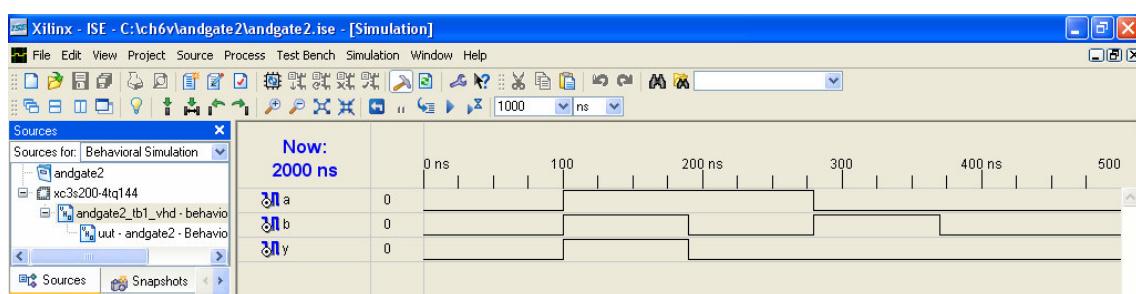
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity andgate2 is
6     Port ( A,B : in STD_LOGIC;
7             Y : out STD_LOGIC);
8 end andgate2;
9
10 architecture Behavioral of andgate2 is
11 begin
12     Y <= A and B;
13 end Behavioral;
```

6.2a) โค้ดของวงจรแอนด์เกต 2 อินพุต

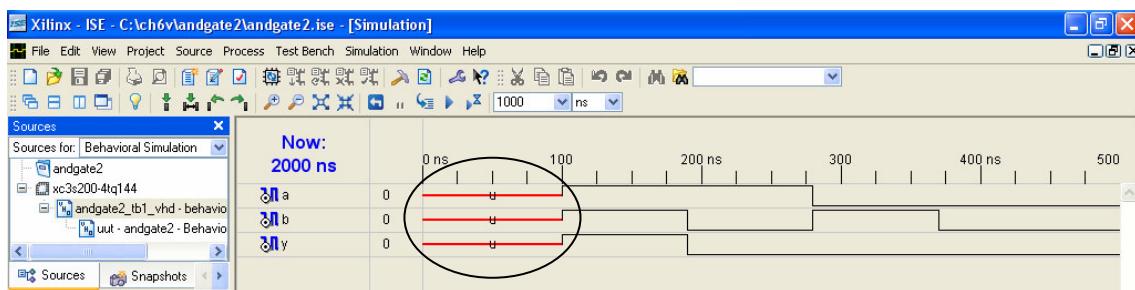
```

2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4
5 ENTITY andgate2_tb1_vhd IS
6 END andgate2_tb1_vhd;
7
8 ARCHITECTURE behavior OF andgate2_tb1_vhd IS
9     COMPONENT andgate2    -- Component Declaration for the UUT
10    PORT(A : IN std_logic;
11          B : IN std_logic;
12          Y : OUT std_logic);
13 END COMPONENT;
14    SIGNAL A : std_logic := '0'; --Input
15    SIGNAL B : std_logic := '0'; --Input
16    SIGNAL Y : std_logic;        --Output
17 BEGIN
18
19     uut: andgate2 PORT MAP(A => A,B => B,Y => Y); -- Instantiate the UUT
20
21     tb : PROCESS
22     BEGIN
23         wait for 100 ns;
24         A <= '1';
25         B <= '1';
26         wait for 90 ns;
27         A <= '1';
28         B <= '0';
29         wait for 90 ns;
30         A <= '0';
31         B <= '1';
32         wait for 90 ns;
33         A <= '0';
34         B <= '0';
35         wait; -- will wait forever
36     END PROCESS tb;
37 END;
```

6.2b) โค้ดของ Testbench ที่ใช้ตรวจสอบความถูกต้องของแอนด์เกตแบบ 2 อินพุต



6.2c) Waveform ของเอาต์พุตที่ได้จากผลการทำงานที่แสดงผลทางหน้าต่างหลัก



6.2d) Waveform ของเอาต์พุตที่ได้จากผลจำลองการทำงาน ซึ่งเราไม่ได้ใส่ค่าเริ่มต้นให้ Signal A และ Signal B

รูปที่ 6.2 โค้ด VHDL และ Testbench ของแอนด์เกต 2 อินพุต

จากบรรทัดที่ 5-6 (รูปที่ 6.2b) จะเห็นได้ว่า Testbench จะไม่มี Port เนื่องจากไม่มีส่วนใดที่ติดต่อกับวงจรภายนอก โค้ดบรรทัดที่ 14-16 แสดงการใช้ Signal ชื่อต่อสัญญาณระหว่างตัวสร้างสัญญาณทดสอบ (Stimulus) กับ UUT ที่เป็น Component โดยที่ Signal นี้จะใช้ชื่อดีบวกับ Port ของ UUT และเนื่องจาก Signal A และ Signal B เพราะเป็นชนิดข้อมูล std_logic (มีอโลจิก 9 สถานะ) การไม่ใส่ค่าเริ่มต้น (Initial value) ให้กับ Signal จะทำให้ล็อกอิจิกเป็น 'U' (Uninitialized) แสดงในรูปที่ 6.2d)

การสร้างสัญญาณทดสอบหรือ Test vector ในรูปที่ 6.2b) บรรทัดที่ 21-36 จะใช้คำสั่ง wait for โดยมี Time_expression คือ 100 ns และ 90 ns ตามลำดับ ซึ่ง Time_expression นี้จะเป็นชนิดข้อมูล time

จากรูปที่ 6.2a) ในบรรทัดที่ 12 ถ้าเขียนโมเดลของเกตเป็น $Y \leq A \text{ and } B \text{ after } 20 \text{ ns}$; (เกตมีเวลาล่าช้าที่เอาต์พุตหลังจากป้อนอินพุต หรือ Propagation delay time (tp) = 20 ns) และเขียน Testbench ใหม่ที่มีการตรวจสอบข้อผิดพลาดแสดงดังรูปที่ 6.3 ซึ่งหลังจากป้อนอินพุตแล้ว 10 ns (wait for gatedelay;) ถ้าตรวจพบว่าเงื่อนไขบูลินในคำสั่ง assert เป็นเท็จแล้วให้ใช้คำสั่ง report เพื่อรายงานข้อผิดพลาดและบอกระดับความรุนแรง (Severity) ทางหน้าต่าง Transcript (ด้านล่างสุด) ซึ่งผลที่ได้แสดงดังรูปที่ 6.4 ซึ่งรายงานว่า “at 110.000 ns: Error: Test1 failed!” และ “at 210.000 ns: Error: Test2 failed!”

```

2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4
5 ENTITY andgate2_tb1_vhd IS
6 END andgate2_tb1_vhd;
7
8 ARCHITECTURE behavior OF andgate2_tb1_vhd IS
9   COMPONENT andgate2           -- Component Declaration for UUT
10    PORT (A : IN std_logic;
11          B : IN std_logic;
12          Y : OUT std_logic);
13  END COMPONENT;
14  SIGNAL A : std_logic := '0';--Inputs
15  SIGNAL B : std_logic := '0';--Inputs
16  SIGNAL C : std_logic := '0';--Inputs
17  SIGNAL Y : std_logic;        --Outputs
18 BEGIN
19
20   uut: andgate2 PORT MAP(A => A,B => B,Y => Y);-- Instantiate the UUT
21
22 tb : PROCESS
23   constant gatedelay : time := 10 ns;
24 BEGIN
25   wait for 100ns;
26   A <= '1';
27   B <= '1';
28   wait for gatedelay;
29   assert (Y = '1') report "Test1 failed!" severity ERROR;
30   wait for 90ns;

```

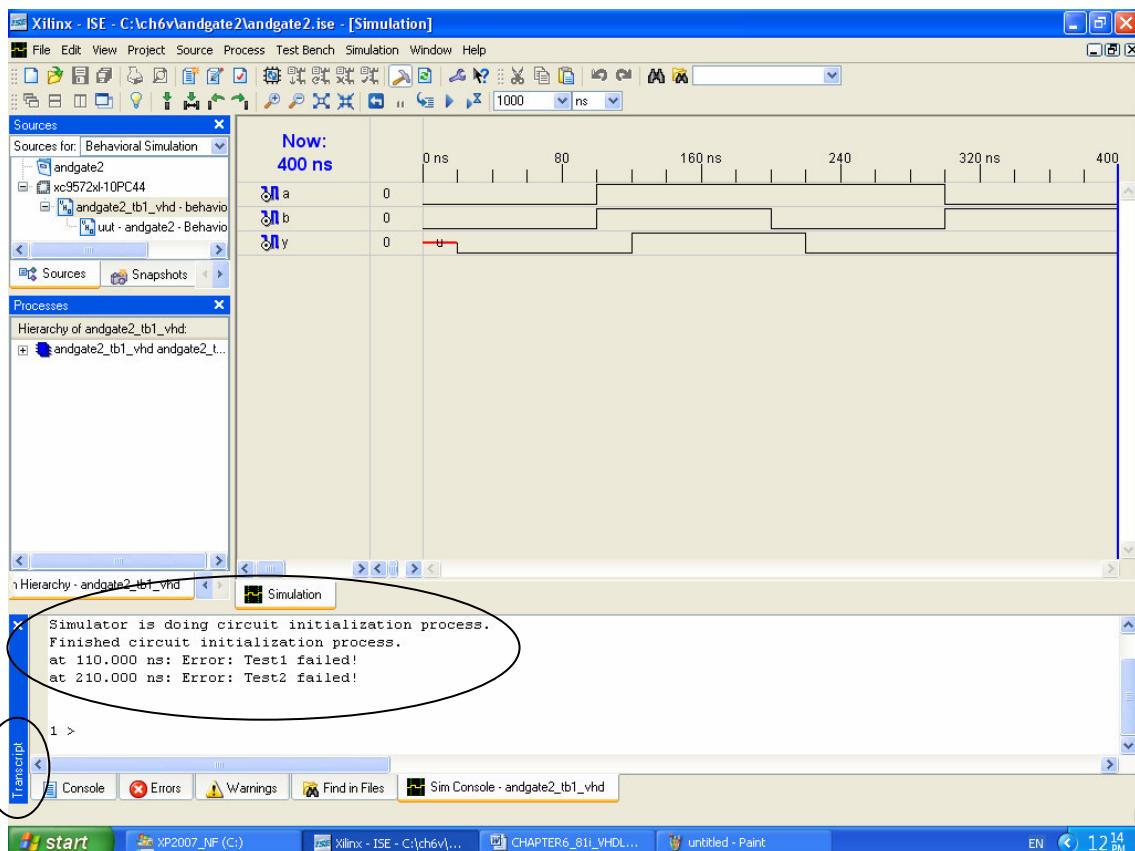
(ต่อ)

```

31      A <= '1';
32      B <= '0';
33      wait for gatedelay;
34      assert (Y = '0') report "Test2 failed!" severity ERROR;
35      wait for 90ns;
36      A <= '0';
37      B <= '1';
38      wait for gatedelay;
39      assert (Y = '0') report "Test3 failed!" severity ERROR;
40      wait for 90ns;
41      A <= '0';
42      B <= '0';
43      wait for gatedelay;
44      assert (Y = '0') report "Test4 failed!" severity ERROR;
45      wait; -- will wait forever
46 END PROCESS tb;
47 END behavior;

```

รูปที่ 6.3 โค้ด Testbench ที่ใช้ตรวจสอบความถูกต้องของแอนด์เกต 2 อินพุตที่มี Propagation delay time ≤ 10 ns



รูปที่ 6.4 รายงานข้อผิดพลาดทางหน้าต่าง Transcript เนื่องจากเกตมี Propagation delay time มาเกินไป

6.1.1 การสร้าง Testbench

จากตัวอย่างที่ 6.1 เราได้เห็นภาพเกี่ยวกับ Testbench กันบ้างแล้ว ในการเขียน Testbench นั้นเราไม่จำเป็นต้องคำนึงถึงเรื่องการสังเคราะห์วงจร บางคำสั่งที่ใช้อาจจะนำไปสังเคราะห์ไม่ได้ Testbench จะประกอบด้วยส่วนต่างๆ ดังนี้

- Entity declaration และ Architecture เป็นส่วนประกาศใช้ Entity (ที่ไม่ประกาศใช้ Port) และ Architecture body
- Signal declaration เป็นส่วนประกาศใช้ Signal ที่ใช้เชื่อมต่อส่วนที่ใช้สร้างสัญญาณทดสอบ (Stimulus) เข้ากับ Port ของ Component ของ UUT ที่เรา拿来ไปทดสอบ ซึ่ง Signal นี้โดยทั่วไปแล้วจะใช้ชื่อเดียวกับ Port ของ UUT

- Instantiation of top-level design จะเป็นการประกาศใช้และใช้ Component ของ UUT ที่เราทำไปทดสอบ
- Provide stimulus เป็นส่วนที่ใช้สร้างสัญญาณทดสอบหรือ Test vector รวมทั้งสัญญาณ Clock และ Clear เป็นต้น

6.1.2 การสร้างสัญญาณนาฬิกา

การสร้างสัญญาณนาฬิกา (Clock) ชี้ว่า Clock มีความจำเป็นสำหรับวงจรซึ่วีเควนเชียล ตัวอย่างเช่น

วิธีที่ 1 เช่นถ้า constant Period : TIME := 200 ns; -- Declare a clock period constant
แล้วจะได้ Clock <= not Clock after Period / 2; -- Clock generation

หรือถ้าไม่ประกาศใช้ Constant จะได้

Clock <= not Clock after 100 ns; -- Clock generation : Period = (100+100) ns, F = 5 MHz

วิธีที่ 2 Process

```
constant Period : TIME := 200 ns; -- Declare a clock period constant
begin
    Clock <= '1';
    wait for (Period / 2);
    Clock <= '0';
    wait for (Period / 2);
end process;
```

ขอให้ผู้อ่านสังเกตว่าคำสั่ง process ในตัวอย่างของวิธีที่ 2 นี้จะไม่มี Sensitivity list เพราะใช้กับคำสั่ง wait

6.1.3 การสร้างสัญญาณทดสอบ

การสร้างสัญญาณทดสอบ (Stimulus) หรือ Test vector เพื่อป้อนให้กับ Component ของ UUT ตัวอย่างเช่น

```
Stimulus1 : process
begin
    Reset <= '1';
    wait for 100 ns;
    CE <= '1';
    Reset <= '0';
    wait for 800 ns;
    CE <= '0';
    wait for 100 ns;
    wait ;
end process Stimulus1;
```

การสร้างสัญญาณทดสอบหรือ Test vector อาจสร้างได้ด้วยคำสั่งต่างๆ ได้หลายวิธี เช่น

S <= "00",	(เริ่มต้น S = "00")
"01" after 30 ns,	(หลังจากเวลาผ่านไป 30 ns แล้ว S = "01")
"11" after 120 ns,	(หลังจากเวลาผ่านไป 120 ns แล้ว S = "11")
"01" after 150 ns;	(หลังจากเวลาผ่านไป 150 ns แล้ว S = "01")

ซึ่งการสร้างสัญญาณทดสอบในตัวอย่างที่กล่าวมานี้จะคล้ายกับการใช้คำสั่ง wait for แต่จะมีความแตกต่างตรงที่การใช้คำสั่ง wait for เป็นการสร้างสัญญาณต่อเนื่องไปเรื่อยๆ เท่ากับเวลาที่กำหนดในคำสั่ง wait for

นอกจากนี้แล้วเราอาจจะสร้างสัญญาณทดสอบจากข้อมูลของเราที่ประกาศใช้ Constant ไว้แล้ว หรืออาจจะอ่านจากไฟล์ที่อยู่ภายนอกก็ได้ การเขียน-อ่านไฟล์จากภายนอกสามารถทำได้โดยการเรียกใช้ Package ชื่อ TEXTIO และในการพิมพ์ของชนิดข้อมูล std_logic และชนิดข้อมูล std_logic_vector นั้นจะเรียกใช้ Package ชื่อ STD_LOGIC_TEXTIO ของบริษัท Synopsys

ตัวอย่างที่ 6.2 สร้าง Testbench อย่างจ่ายของ D Flip-flop แบบ Asynchronous clear โดยที่ C เป็น Clock และ CE เป็นขา Clock enable input โค้ดมีโค้ดแสดงดังรูปที่ 6.5a) Testbench และแสดงผลรูปที่ 6.5b) และเอาต์พุต Waveform และแสดงดังรูปที่ 6.5c)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DFF_EN is
6   Port ( C,CLR,D,CE : in STD_LOGIC;
7          Q : out STD_LOGIC);
8 end DFF_EN;
9
10 architecture Behavioral of DFF_EN is
11   signal QT : STD_LOGIC:='0';
12 begin
13 process(C,CLR)
14 begin
15   if CLR='1' then QT <= '0';
16   elsif C'event and C='1' then
17     if CE='1' then QT <= D;
18     end if;
19   end if;
20 end process;
21 Q <= QT;
22 end Behavioral;

```

6.5a) โค้ดของวงจร D Flip-flop !!แบบ Asynchronous clear

```

2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4
5 ENTITY DFF_EN_TB_vhd IS
6 END DFF_EN_TB_vhd;
7
8 ARCHITECTURE behavior OF DFF_EN_TB_vhd IS
9   -- Component Declaration for the Unit Under Test (UUT)
10  COMPONENT DFF_EN
11    PORT(
12      C : IN std_logic;
13      CLR : IN std_logic;
14      D : IN std_logic;
15      CE : IN std_logic;
16      Q : OUT std_logic
17    );
18  END COMPONENT;

```

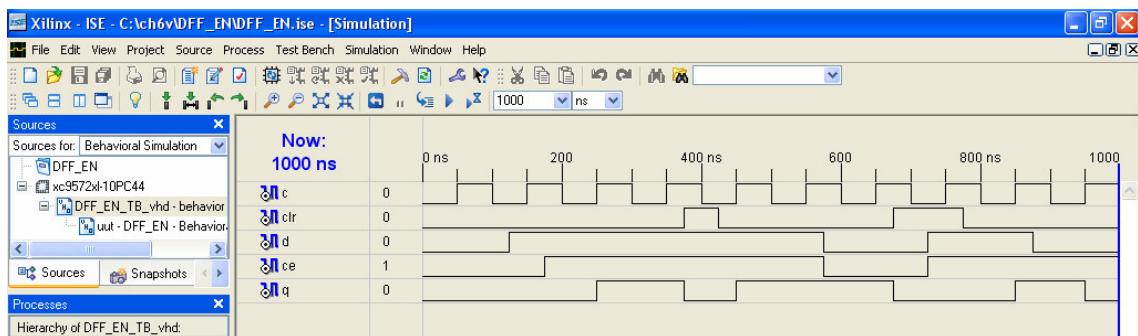
(ต่อ)

```

19      --Inputs
20      SIGNAL C :  std_logic := '0';
21      SIGNAL CLR :  std_logic := '0';
22      SIGNAL D :  std_logic := '0';
23      SIGNAL CE :  std_logic := '0';
24      --Outputs
25      SIGNAL Q :  std_logic;
26 BEGIN
27      -- Instantiate the Unit Under Test (UUT)
28      uut: DFF_EN PORT MAP(
29          C => C,
30          CLR => CLR,
31          D => D,
32          CE => CE,
33          Q => Q
34      );
35      --Generating clock
36      Clock_gen : PROCESS
37          BEGIN
38              C <= '0';
39              wait for 50 ns;
40              C <= '1';
41              wait for 50 ns;
42      END PROCESS Clock_gen;
43
44      --Providing stimulus
45      CLR <= '0', '1' after 375 ns, '0' after 425 ns, '1' after 675 ns, '0' after 775 ns;
46
47      tb : PROCESS
48          BEGIN
49              wait for 100 ns; -- Wait 100 ns for global reset to finish
50              wait for 25 ns;
51              D <= '1';
52              wait for 50 ns;
53              CE <= '1';
54              wait until C'event and C='1';
55              wait for 325 ns;
56              D <= '0';
57              CE <= '0';
58              wait for 150 ns;
59              D <= '1';
60              CE <= '1';
61              wait for 150 ns;
62              D <= '0';
63              wait;-- will wait forever
64      END PROCESS tb;
65 END behavior;

```

6.5b) Testbench ของวงจร D Flip-flop แบบ Asynchronous clear ซึ่งเราสร้างสัญญาณทดสอบให้ดูอย่างรูปแบบ



6.5c) เอาท์พุต Waveform ของวงจร D Flip-flop แบบ Asynchronous clear

รูปที่ 6.5 โค้ดและ Testbench ของวงจร D Flip-flop แบบ Asynchronous clear

6.1.4 การแสดงผลและตรวจสอบข้อผิดพลาดโดยอัตโนมัติ

การตรวจสอบเพื่อหาข้อผิดพลาดโดยอัตโนมัติหมายความว่าระบบที่มีขนาดใหญ่ ซึ่งจะช่วยลดการเสียเวลาลงไปได้มากและลดความผิดพลาดที่เกิดขึ้นเนื่องจากคน การตรวจสอบความถูกต้องโดยอัตโนมัติแบบ Self-checking testbench เป็นการตรวจสอบความถูกต้องวิธีหนึ่งที่นิยมใช้กันมาก ซึ่งเรารอจะนำเสนอค่าพื้นจริง (Actual results) ไปเปรียบเทียบกับเอาต์พุตตามทฤษฎีซึ่งเป็นเอาต์พุตที่คาดการว่าจะได้ (Expected results) ถ้าหากผลที่ได้ไม่ตรงกันก็แสดงว่ามีข้อผิดพลาดเกิดขึ้น และให้รายงานผลโดยอัตโนมัติ (ทางหน้าต่าง Transcript ที่อยู่ด้านล่างของหน้าต่าง Xilinx-ISE) การสร้าง Self-checking testbench นี้ หมายความว่าสามารถนำไปเปรียบเทียบกันตอนที่เป็นเวลาของข้อมูลหรืออาจกำหนดต่อๆ กัน ทั้งนี้สามารถนำ去ใช้ในการออกแบบซิงโครนัส (Synchronous design) เพราะเอาต์พุตที่ได้จะริงแคละเอาต์พุตที่คาดการว่าจะได้ในนั้น สามารถนำไปเปรียบเทียบกันตอนที่เป็นเวลาของข้อมูลหรืออาจกำหนดต่อๆ กัน ทั้งนี้สามารถนำ去ใช้ในการออกแบบซิงโครนัส (Synchronous design) เพราะเอาต์พุตที่ได้จะริงแคละเอาต์พุตที่คาดการว่าจะได้ในนั้น

ตัวอย่างที่ 6.3 ฝึกการเขียน Testbench ของโค้ด VHDL วงจรบวก (แบบไม่คิดเครื่องหมายหรือ Unsigned) 4 บิต โดยในรูปที่ 6.6a) จะเป็นการเขียนโค้ดโดยเรียกใช้ Package ชื่อ std_logic_unsigned ของ Synopsys โดยที่ Testbench ของวงจรบวกนี้มีการสร้างสัญญาณทดสอบ (Stimulus) หรือ Test vector จากข้อมูลคงเรียบที่ประกาศใช้ Constant แสดงดังรูปที่ 6.6b) ซึ่งในตัวอย่างนี้เราจะแสดงการเลือกทดสอบเฉพาะบางค่าเท่านั้น เพื่อประหยัดเวลาในการทดสอบและสร้างไฟล์ข้อมูลทดสอบ โดยรูปที่ 6.6c) นั้นจะแสดงเอาต์พุต Waveform และรายงานผลทางหน้าต่าง Transcript (ด้านล่างสุด) ของหน้าต่าง Xilinx-ISE

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ADDER4BIT is
7 generic(N : integer :=4);
8 port( A,B : in std_logic_vector(N-1 downto 0);
9       CARRY : out std_logic;
10      SUM : out std_logic_vector(N-1 downto 0));
11 end ADDER4BIT;
12
13 architecture Behavioral of ADDER4BIT is
14   signal C : std_logic_vector(N downto 0);
15 begin
16   C <= ('0' &A)+('0' &B);
17   SUM <= C(N-1 downto 0) after 5ns;
18   CARRY <= C(N) after 5ns;
19 end Behavioral;

```

6.6a) โค้ด VHDL ของวงจรบวก 4 บิต

```

2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4
5 ENTITY ADDER4BIT_TB_vhd IS
6 END ADDER4BIT_TB_vhd;
7
8 ARCHITECTURE behavior OF ADDER4BIT_TB_vhd IS
9 -- Component Declaration for the Unit Under Test (UUT)
10  COMPONENT ADDER4BIT
11    PORT(
12      A : IN std_logic_vector(3 downto 0);
13      B : IN std_logic_vector(3 downto 0);
14      CARRY : OUT std_logic;
15      SUM : OUT std_logic_vector(3 downto 0)
16    );
17  END COMPONENT;
18 --Inputs
19  SIGNAL A : std_logic_vector(3 downto 0) := (others=>'0');
20  SIGNAL B : std_logic_vector(3 downto 0) := (others=>'0');

```

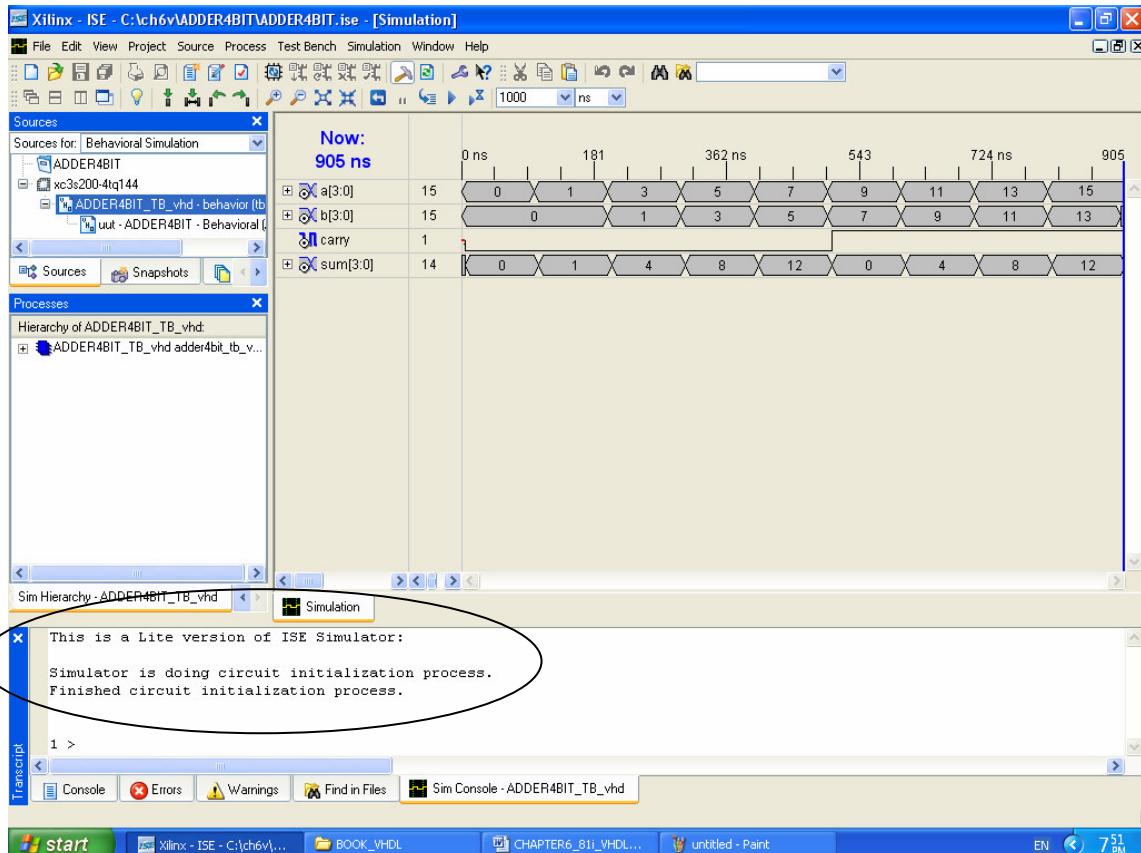
(ต่อ)

```

21 --Outputs
22   SIGNAL CARRY : std_logic;
23   SIGNAL SUM : std_logic_vector(3 downto 0);
24   TYPE adder_array IS ARRAY(0 to 9) of std_logic_vector(3 downto 0);
25   TYPE carry_array IS ARRAY(0 to 9) of std_logic;
26   CONSTANT Ain : adder_array := ("0000","0001","0011","0101","0111",
27                                     "1001","1011","1101","1111","1111");
28   CONSTANT Bin : adder_array := ("0000","0000","0001","0011","0101",
29                                     "0111","1001","1011","1101","1111");
30   CONSTANT SUMout : adder_array :=("0000","0001","0100","1000","1100",
31                                     "0000","0100","1000","1100","1110");
32   CONSTANT CARRYout : carry_array :=( '0',    '0',    '0',    '0',    '0',
33                                     '1',    '1',    '1',    '1',    '1');
34 BEGIN
35 --Instantiate the Unit Under Test (UUT)
36   uut: ADDER4BIT PORT MAP(A => A,B => B,CARRY => CARRY,SUM => SUM);
37 --Stimulus
38   tb : PROCESS
39     BEGIN
40       FOR i IN adder_array'RANGE LOOP
41         A <= Ain(i);
42         B <= Bin(i);
43         WAIT FOR 10ns;
44         ASSERT SUM = SUMout(i) REPORT "SUM failed!" SEVERITY error;
45         ASSERT CARRY = CARRYout(i) REPORT " CARRY failed!" SEVERITY error;
46         WAIT FOR 90ns;
47     END LOOP;
48     wait; -- will wait forever
49   END PROCESS tb;
50 END behavior;

```

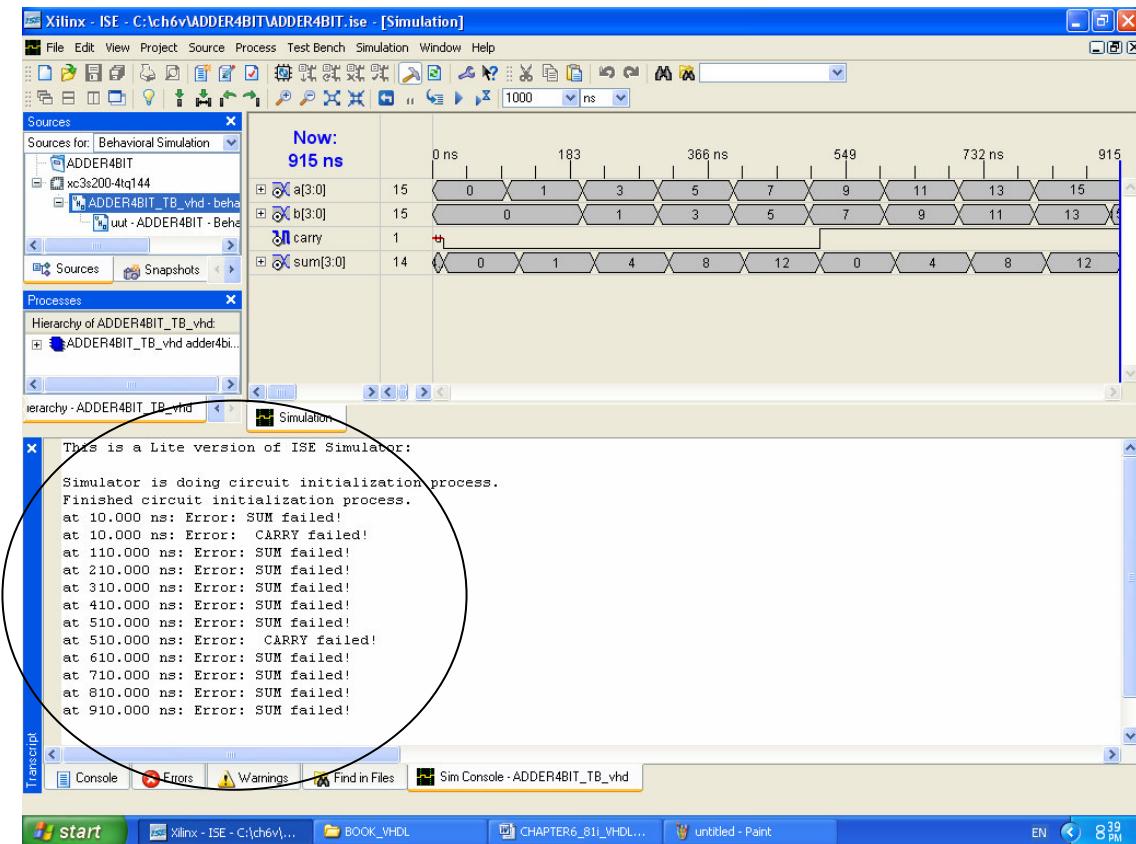
6.6b) Testbench ที่ใช้ตรวจสอบความถูกต้องของวงจรบวก 4 บิต



6.6c) Waveform และรายงานผลทางหน้าต่าง Transcript (ที่อยู่ด้านล่างสุด) ซึ่งไม่มีข้อผิดพลาด

รูปที่ 6.6 โค้ด VHDL ของวงจรบวก 4 บิต Testbench และผลจำลองการทำงาน

โค้ดบรรทัดที่ 17-18 ในรูปที่ 6.6a จะมีคำว่า “after 5 ns” ซึ่งในการสังเคราะห์วงจรนั้นซอฟต์แวร์ทูล XST (ที่ติดตั้งใน WebPACK) จะละเว้นโดยไม่สนใจ Delay นี้ แต่ ISE Simulator จะตีความว่า Propagation delay time = 5 ns และถ้าเราแก้ไขค่า Propagation delay time จาก “after 5 ns” เป็น “after 15 ns” แล้วจะได้ผลจำลองการทำงานแสดงดังรูปที่ 6.7



รูปที่ 6.7 Waveform และข้อผิดพลาดที่หน้าต่าง Transcript เนื่องจากวงจร มี Propagation delay time มากเกินไป

การทดลองที่ 6.1.1 Testbench ของบอร์ดทดลองทิศทาง

วัตถุประสงค์

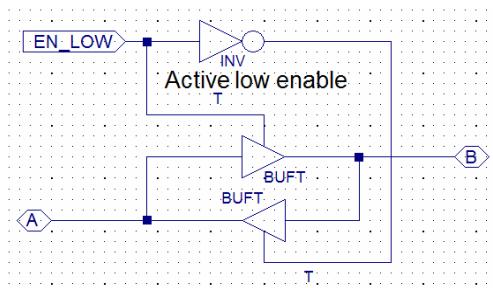
- 1) เพื่อทำความเข้าใจเกี่ยวกับการสร้าง Testbench เพื่อทำ Behavioral Simulation ของบอร์ดทดลองทิศทาง
- 2) เพื่อทดลองใช้ซอฟต์แวร์ทุก ISE WebPACK 8.1i สร้างบอร์ดทดลองทิศทางโดยใช้ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างบอร์ดทดลองทิศทางด้วย CPLD

- 1.1) ก่อนเข้าโปรแกรม ISE WebPACK ให้สร้าง Folder ชื่อ ch6v ไว้ในไดร์ฟ C (ถ้าไม่มี) เนื่องจากต้องบอร์ดทดลองที่มีผังวงจรดังรูปที่ L1.1 ไว้ใน Project Location ชื่อ ch6v แล้วกำหนดให้ Project Name และ Source File ชื่อ ex6_1_1vcx1 โดยโคลด์ของบอร์ดทดลองแสดงดังรูปที่ L1.2 แล้วคลิก บันทึกไฟล์และ Check Syntax



รูปที่ L1.1 ผังวงจรบอร์ดทดลองทิศทาง

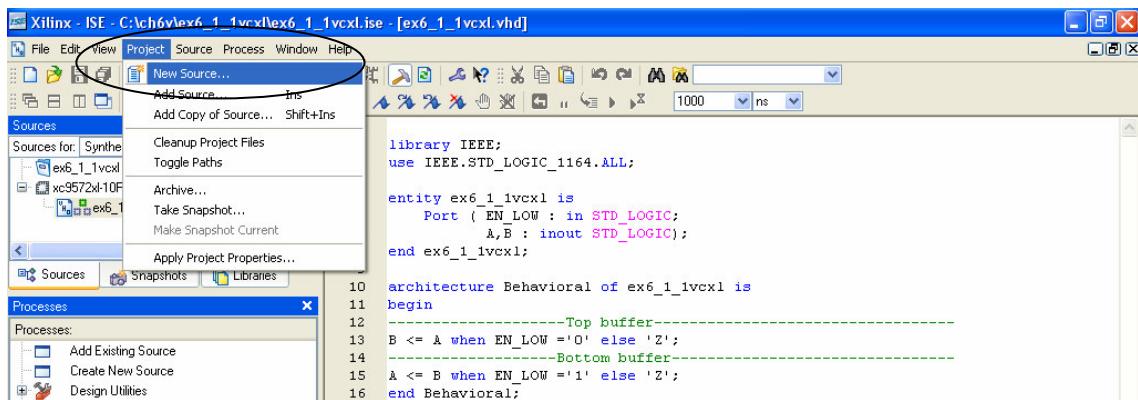
```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex6_1_2vcx1 is
6     Port ( EN_LOW : in STD_LOGIC;
7             A,B : inout STD_LOGIC);
8 end ex6_1_2vcx1;
9
10 architecture Behavioral of ex6_1_2vcx1 is
11 begin
12     -----Top buffer-----
13     B <= A when EN_LOW = '0' else 'Z';
14     -----Bottom buffer-----
15     A <= B when EN_LOW = '1' else 'Z';
16 end Behavioral;

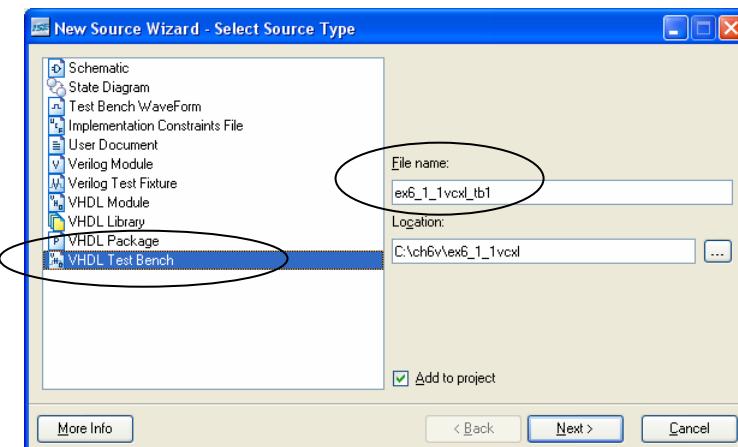
```

รูปที่ L1.2 โคลด์ VHDL ของวงจรบอร์ดทดลองทิศทาง

- 1.2) สร้าง Testbench คลิก Project->New Source ดังรูปที่ L1.3 แล้วพิมพ์ ex6_1_1vcx1_tb1 ในช่อง File name และคลิก VHDL Test Bench ดังรูปที่ L1.4 คลิก Next 2 ครั้ง คลิก Finish และจะได้หน้าต่าง Xilinx-ISE ที่ใช้เขียน Testbench จากนั้นแก้ไขโคลด์เป็นดังรูปที่ L1.5 คลิก บันทึกไฟล์ คลิก แล้วคลิกที่ Behavioral Simulation และคลิกชื่อไฟล์ ex6_1_1vcx1_tb1_vhdl ในหน้าต่าง Source คลิก “+” หน้า Xilinx ISE Simulator ในหน้าต่าง Processes จะเป็น “-” แล้วคลิกขวาที่ Check Syntax และคลิก Run ดังในรูปที่ L1.6 ถ้าปรากฏ ที่หน้า Check Syntax ถือว่าผ่าน แต่ถ้าปรากฏ ถือว่าไม่ผ่านและให้แก้ไขโคลด์ Testbench ใหม่



รูปที่ L1.3 ขั้นตอนเริ่มสร้าง Source file ของ Testbench



รูปที่ L1.4 ขั้นตอนพิมพ์ชื่อ Source file และคลิกสร้าง Testbench

```

2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4
5 ENTITY ex6_1_lvcxl_tb1_vhd IS
6 END ex6_1_lvcxl_tb1_vhd;
7
8 ARCHITECTURE behavior OF ex6_1_lvcxl_tb1_vhd IS
9     -- Component Declaration for the Unit Under Test (UUT)
10    COMPONENT ex6_1_lvcxl
11        PORT(EN_LOW : IN std_logic;
12              A : INOUT std_logic;
13              B : INOUT std_logic);
14    END COMPONENT;
15    --Inputs
16    SIGNAL EN_LOW :  std_logic := '0';
17    --BiDirs
18    SIGNAL A :  std_logic;
19    SIGNAL B :  std_logic;
20
21 BEGIN
22     -- Instantiate the Unit Under Test (UUT)
23     uut: ex6_1_lvcxl PORT MAP(EN_LOW => EN_LOW,A => A,B => B);
24
25     tb : PROCESS
26     BEGIN
27         EN_LOW <= '1'; -- B->A
28         A <= 'Z';      -- Multiple driver : ('Z','1') = '1'
29         B <= '1';      -- B=Input
30         wait for 100 ns;
31         EN_LOW <= '0'; -- A->B
32         A <= '1';      -- A=Input
33         B <= 'Z';      -- Multiple driver : ('Z','1') = '1'
34         wait for 100 ns;

```

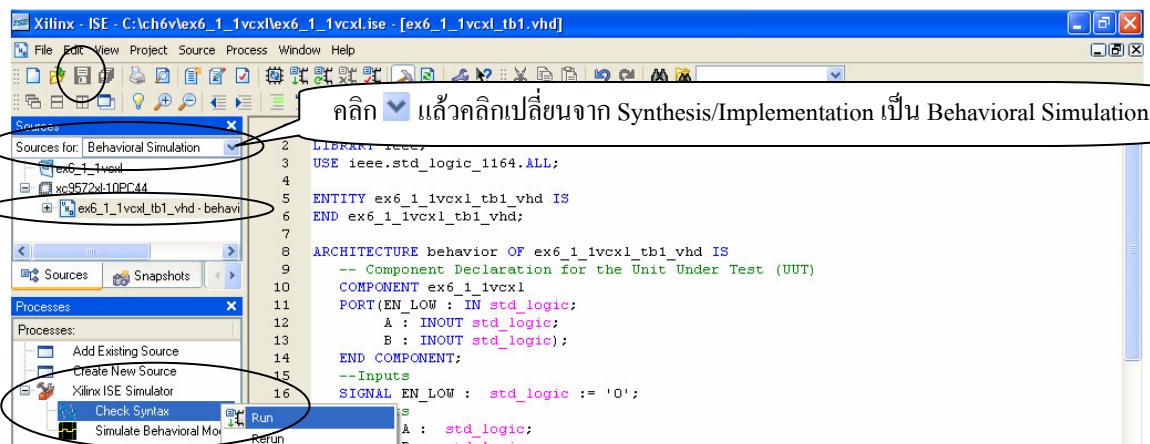
(ต่อ)

```

34      A <= '0';      -- A->B, A=Input
35      B <= 'Z';      -- Multiple driver : ('Z','0') = '0'
36      wait for 100 ns;
37      EN_LOW <= '1'; -- B->A
38      A <= 'Z';      -- Multiple driver : ('Z','0') = '0'
39      B <= '0';      -- B=Input
40      wait for 100 ns;
41      A <= 'Z';      -- Multiple driver : ('Z','Z') = 'Z'
42      B <= 'Z';      -- B->A,B=Input
43      wait for 100 ns;
44      A <= '0';      -- Multiple driver : ('0','1') = 'X'
45      B <= '1';      -- B->A,B=Input
46      wait for 100 ns;
47      A <= '1';      -- Multiple driver : ('1','1') = '1'
48      B <= '1';      -- B->A,B=Input
49      wait for 100 ns;
50      EN_LOW <= '0'; -- A->B
51      A <= '1';      -- A=Input
52      B <= '0';      -- Multiple driver : ('0','1') = 'X'
53      wait for 200 ns;
54      wait; -- will wait forever
55  END PROCESS tb;
56 END behavior;

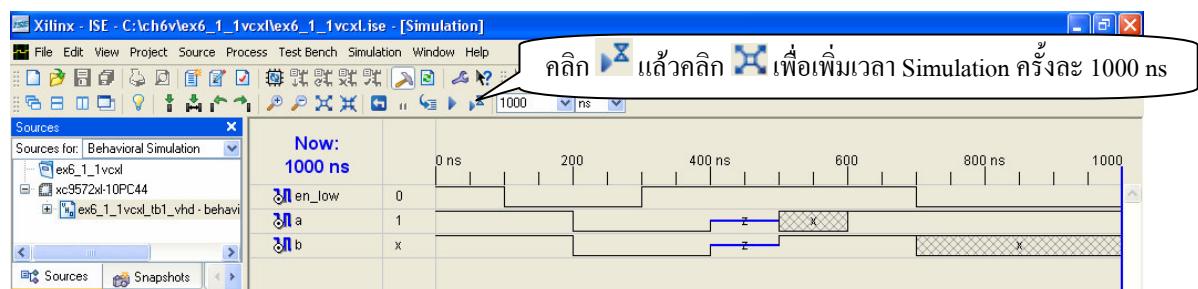
```

รูปที่ L1.5 โค้ดของ Testbench ที่เขียนเสร็จสมบูรณ์แล้ว



รูปที่ L1.6 ขั้นตอนคลิกเปลี่ยนจาก Synthesis/Implementation เป็น Behavioral Simulation

- 1.3) ทำ Behavioral simulation เลื่อนมาส่องไปด้านล่างของ Check Syntax ในรูปที่ L1.6 คลิกขวาที่ Simulate Behavioral Model และคลิก Run แล้วจะได้ดังรูปที่ L1.7 ให้พิจารณาผลการทำงานที่ได้ว่าเป็นไปตามทฤษฎีหรือไม่ จากนั้นคลิก ✕ (สีดำ) และคลิก Yes และคลิก ✕ (สีดำ) อีกครั้งเพื่อปิด Testbench กลับไปที่หน้าต่าง Xilinx-ISE ที่หน้าต่าง Source ให้คลิก ✓ แล้วคลิกเปลี่ยนจาก Behavioral Simulation กลับไปเป็น Synthesis/Implementation ตามเดิม ซึ่งการเขียนโค้ดคงจะย่อหรือโก้ด้าหัวรับใช้ทำเป็น Component นั้นเรารอจะเบินและตรวจสอบจนถึงขั้นตอนนี้ก็ถือว่าเพียงพอแล้ว แต่ยังไร์ก์ตามถ้าเราต้องการนำ้งงานนี้ไปดำเนินการ荷ลง CPLD ก็ให้ทำขั้นตอน Synthesis, Implementation และ Generate Programming File ต่อไป



รูปที่ L1.7 ผล Behavioral simulation ของวงจรบีฟอร์ส่องทิศทางที่ได้จากการเขียน Testbench

2 สร้างบอร์ดทดลองที่ติดต่อ FPGA

2.1) ก่อนเข้าโปรแกรม ISE WebPACK ให้สร้าง Folder ชื่อ ch6v ไว้ในไดรฟ์ C (ถ้าไม่มี) เขียนโค้ดของบอร์ดทดลองที่มีผังวงจรดังรูปที่ L1.1 ไว้ใน Project Location ชื่อ ch6v แล้วกำหนดให้ Project Name และ Source File ชื่อ ex6_1_1vf โดยโค้ดของบอร์ดทดลองแสดงดังรูปที่ L2.1 แล้วคลิก บันทึกไฟล์และ Check Syntax

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity ex6_1_1vf is
6     Port ( EN_LOW : in STD_LOGIC;
7             A,B : inout STD_LOGIC);
8 end ex6_1_1vf;
9
10 architecture Behavioral of ex6_1_1vf is
11 begin
12     -----Top buffer-----
13     B <= A when EN_LOW = '0' else 'Z';
14     -----Bottom buffer-----
15     A <= B when EN_LOW = '1' else 'Z';
16 end Behavioral;

```

รูปที่ L2.1 โค้ด VHDL ของวงจรบอร์ดทดลองที่ติดต่อ FPGA

2.2) การสร้าง Testbench ในกรณีที่เป็น FPGA จะมีขั้นตอนเหมือน CPLD ทุกประการ ตามขั้นตอนที่ได้ทำการทดลองในข้อ 1 โดยที่เราใช้ Source file ชื่อ ex6_1_1vf_tb1 จากนั้นจึงเขียน Testbench เป็นดังรูปที่ L2.2 บันทึกไฟล์และ Check Syntax

```

2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4
5 ENTITY ex6_1_1vf_tb1_vhd IS
6 END ex6_1_1vf_tb1_vhd;
7
8 ARCHITECTURE behavior OF ex6_1_1vf_tb1_vhd IS
9     -- Component Declaration for the Unit Under Test (UUT)
10    COMPONENT ex6_1_1vf
11        PORT(EN_LOW : IN std_logic;
12              A : INOUT std_logic;
13              B : INOUT std_logic);
14    END COMPONENT;
15    --Inputs
16    SIGNAL EN_LOW :  std_logic := '0';
17    --BiDirs
18    SIGNAL A :  std_logic;
19    SIGNAL B :  std_logic;
20
21 BEGIN
22     -- Instantiate the Unit Under Test (UUT)
23     uut: ex6_1_1vf PORT MAP(EN_LOW => EN_LOW,A => A,B => B);
24
25     tb : PROCESS
26     BEGIN
27         EN_LOW <= '1'; -- B->A
28         A <= 'Z';      -- Multiple driver : ('Z','1') = '1'
29         B <= '1';      -- B=Input
30         wait for 100 ns;
31         EN_LOW <= '0'; -- A->B
32         A <= '1';      -- A=Input
33         B <= 'Z';      -- Multiple driver : ('Z','1') = '1'
34         wait for 100 ns;
35         A <= '0';      -- A->B,A=Input
36         B <= 'Z';      -- Multiple driver : ('Z','0') = '0'
37         wait for 100 ns;

```

(ต่อ)

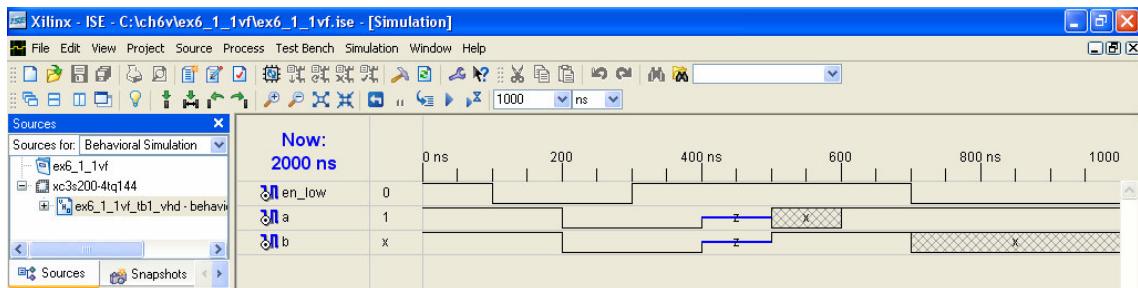
```

37      EN_LOW <= '1'; -- B->A
38      A <= 'Z';    -- Multiple driver : ('Z','0') = '0'
39      B <= '0';    -- B=Input
40      wait for 100 ns;
41      A <= 'Z';    -- Multiple driver : ('Z','Z') = 'Z'
42      B <= 'Z';    -- B->A,B=Input
43      wait for 100 ns;
44      A <= '0';    -- Multiple driver : ('0','1') = 'X'
45      B <= '1';    -- B->A,B=Input
46      wait for 100 ns;
47      A <= '1';    -- Multiple driver : ('1','1') = '1'
48      B <= '1';    -- B->A,B=Input
49      wait for 100 ns;
50      EN_LOW <= '0'; -- A->B
51      A <= '1';    -- A=Input
52      B <= '0';    -- Multiple driver : ('0','1') = 'X'
53      wait for 200 ns;
54      wait; -- will wait forever
55  END PROCESS tb;
56 END behavior;

```

รูปที่ L2.2 โค้ดของ Testbench ที่เขียนเสร็จสมบูรณ์แล้ว

2.3) ทำ Behavioral simulation เสร็จแล้วจะได้รูปที่ L2.3 ให้พิจารณาผลการทำงานที่ได้ว่าเป็นไปตามทฤษฎีไม่จากนั้นปิด Testbench กลับไปที่หน้าต่าง Xilinx-ISE ซึ่งการเขียนโค้ดของเรายังคงอยู่หรือโค้ดสำหรับใช้ทำเป็น Component นั้นเราอาจจะเพิ่มและตรวจสอบจนถึงขั้นตอนนี้ก็ถือว่าเพียงพอแล้ว แต่ยังไหร่ก็ตามถ้าเราต้องการนำ้งงานนี้ไปดาวน์โหลดลง FPGA ก็ให้ทำขั้นตอน Synthesis, Implementation และ Generate Programming File ต่อไป



รูปที่ L2.3 ผล Behavioral simulation ของวงจรบัญชีฟอร์ส่องทิศทางที่ได้จากการเขียน Testbench

หมายเหตุ หลังจากทำ Behavioral simulation แล้วเรามารอๆกันไปทำการสังเคราะห์วงจร ทำการกำหนดขา CPLD หรือ FPGA ทำการ Implementation และทำการ Generate programming file และดาวน์โหลดตามขั้นตอนปกติ

การทดลองที่ 6.1.2 Testbench ของวงจรตรวจจับลำดับที่เป็น Mealy-type FSM

วัตถุประสงค์

- 1) เพื่อสร้าง Testbench และทำ Behavioral Simulation ของวงจรตรวจจับลำดับ (Sequence detector)
- 2) เพื่อตรวจสอบการเกิด Glitch ที่อาจพุ่ง ซึ่งวิธีการ Simulation โดยใช้ Waveform Editor ตรวจไม่พบ
- 3) เพื่อทดลองใช้ซอฟต์แวร์ทุก ISE WebPACK 8.1i สร้างวงจรตรวจจับลำดับ โดยใช้ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรตรวจจับลำดับด้วย CPLD

1.1) ออกแบบวงจรตรวจจับลำดับ (Sequence detector) เป็น Mealy-type FSM ตามตัวอย่างที่ 4.5 ของบทที่ 4 ซึ่งมี X เป็นอินพุต และ Z เป็นเอาต์พุต โดยที่วงจรนี้จะให้ Z = '1' เมื่อตรวจจับลอกอิกิ X = '1' จำนวน 3 ตัวติดต่อกัน จากนั้นให้นำโคล็ควงจรนี้ในรูปที่ 4.72 ของบทที่ 4 มาเขียนใหม่ไว้ใน Project Location ชื่อ ch6v และกำหนดให้ Project Name และ Source File ชื่อ ex6_1_2vcx1 โดยจะแก้ชื่อ Entity เป็น ex6_1_2vcx1 โดยคดีที่ได้แสดงดังรูปที่ L1.1 ทำการบันทึกไฟล์และ Check Syntax

```

2 ----- Mealy-type FSM.-----
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 entity ex6_1_2vcx1 is
7     Port ( X,Clock,Reset : in STD_LOGIC;
8             Z : out STD_LOGIC);
9 end ex6_1_2vcx1;
10
11 architecture Behavioral of ex6_1_2vcx1 is
12     type State_type is (A,B,C);
13     signal State,Next_state : State_type ;
14 begin
15     P1_combination : process (X,State)
16     begin
17         case State is
18             when A =>
19                 if X ='1' then
20                     Next_state <= B;
21                     Z <= '0';
22                 else
23                     Next_state <= A;
24                     Z <= '0';
25                 end if;
26             when B =>
27                 if X ='1' then
28                     Next_state <= C;
29                     Z <= '0';
30                 else
31                     Next_state <= A;
32                     Z <= '0';
33                 end if;
34             when C =>
35                 if X ='1' then
36                     Next_state <= C;
37                     Z <= '1';
38                 else
39                     Next_state <= A;
40                     Z <= '0';

```

(ต่อ)

```

41           end if;
42       end case;
43   end process P1_combination;
44
45   P2_state_resister : process (Clock,Reset)
46   begin
47       if (Reset = '1') then
48           State <= A;
49       elsif (Clock'event and Clock = '1') then
50           State <= Next_state;
51       end if;
52   end process P2_state_resister;
53 end Behavioral;

```

รูปที่ L1.1 โค้ดวงจรตรวจสอบลำดับที่เป็น Mealy-type FSM

1.2) สร้าง Testbench โดยมีขั้นตอนเหมือนกับการทดลองที่ 6.1.1 ทุกประการ โดยครั้งนี้ใน Source file ชื่อ ex6_1_2vcx1_tb1 เสร็จแล้วจะได้ดังรูปที่ L1.2 บันทึกไฟล์และ Check Syntax โค้ดของ Testbench

```

2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4
5 ENTITY ex6_1_2vcx1_tb1_vhd IS
6 END ex6_1_2vcx1_tb1_vhd;
7
8 ARCHITECTURE behavior OF ex6_1_2vcx1_tb1_vhd IS
9     -- Component Declaration for the Unit Under Test (UUT)
10    COMPONENT ex6_1_2vcx1
11        PORT(X : IN std_logic;
12              Clock : IN std_logic;
13              Reset : IN std_logic;
14              Z : OUT std_logic);
15    END COMPONENT;
16    --Inputs
17    SIGNAL X : std_logic := '0';
18    SIGNAL Clock : std_logic := '0';
19    SIGNAL Reset : std_logic := '0';
20    --Outputs
21    SIGNAL Z : std_logic;
22 BEGIN
23     -- Instantiate the Unit Under Test (UUT)
24     uut: ex6_1_2vcx1 PORT MAP(X => X,Clock => Clock,Reset => Reset,Z => Z);
25
26     Clock_gen : PROCESS --Generating clock
27     BEGIN
28         Clock <= '1';
29         wait for 50 ns;
30         Clock <= '0';
31         wait for 50 ns;
32
33     END PROCESS Clock_gen;
34
35     tb : PROCESS      -- Stimulus.
36     BEGIN
37         wait for 100 ns; -- Wait 100 ns for global reset to finish
38         wait for 20 ns;
39         Reset <= '1';
40         wait for 100 ns;
41         Reset <= '0';
42         wait for 100 ns;
43         X <= '1';
44         wait for 100 ns;
45         X <= '0';
46         wait for 200 ns;
47         X <= '1';
48         wait for 200 ns;
49         X <= '0';
50         wait for 200 ns;

```

(ต่อ)

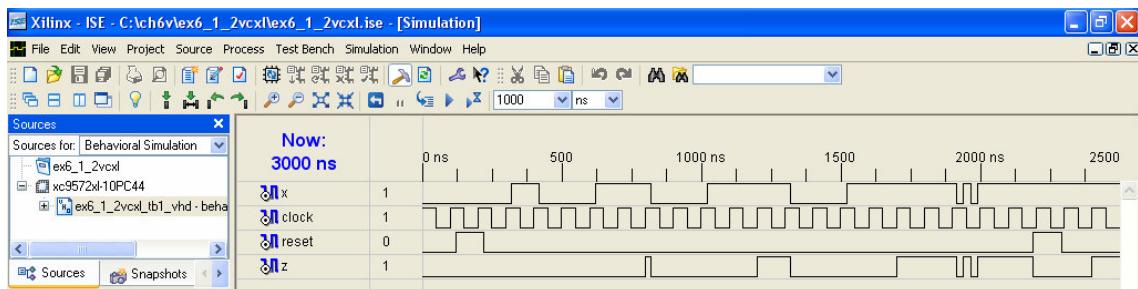
```

51      X <= '1';
52      wait for 300 ns;
53      X <= '0';
54      wait for 200 ns;
55      X <= '1';
56      wait for 395 ns;
57      X <= '0';
58      wait for 25 ns;
59      X <= '1';
60      wait for 25 ns;
61      X <= '0';
62      wait for 25 ns;
63      X <= '1';
64      wait for 200 ns;
65      Reset <= '1';
66      wait for 100 ns;
67      Reset <= '0';
68      wait; -- will wait forever
69  END PROCESS tb;
70
END behavior;

```

รูปที่ L1.2 โค้ด Testbench ของวงจรตรวจจับลำดับที่เป็น Mealy-type FSM

1.3) ทำ Behavioral simulation จะได้ดังรูปที่ L1.3 ให้พิจารณาว่าผลที่ได้ว่าเป็นไปตามทฤษฎีหรือไม่



รูปที่ L1.3 ผล Behavioral simulation ของวงจรตรวจจับคำนับที่เป็น Mealy-type FSM ที่ได้จากการเขียน Testbench

2 สร้างวงจรตรวจจับลำดับด้วย FPGA

2.1) ออกแบบวงจรตรวจจับลำดับ (Sequence detector) เป็น Mealy-type FSM ตามทัวร์ย่างที่ 4.5 ในบทที่ 4 ซึ่งมี X เป็นอินพุต และ Z เป็นเอาต์พุต โดยที่วงจรนี้จะให้ Z = '1' เมื่อตรวจจับลอกิจ X = '1' จำนวน 3 ตัวติดต่อกัน การสร้างวงจรตรวจจับลำดับด้วย FPGA จะมีขั้นตอนเหมือน CPLD ทุกประการ จากนั้นให้นำโค้ดวงจรนี้ไปรูปที่ 4.72 ของบทที่ 4 มาเพิ่นใหม่ไว้ใน Project Location ชื่อ ch6v แล้วกำหนดให้ Project Name และ Source File ชื่อ ex6_1_2vf โดยจะแก้ชื่อ Entity เป็น ex6_1_2vf ทำการบันทึกไฟล์และ Check Syntax

2.2) ขั้นตอนสร้าง Testbench จะเหมือน CPLD ทุกประการ โดยใช้ไฟล์ชื่อ ex6_1_vf_tb1 และเราจะแก้ไขด้านรูปที่ L1.2 โดยจะแก้คำว่า ex6_1_vcxl เป็น ex6_1_vf และ ex6_1_vcxl_tb1 เป็น ex6_1_vf_tb1

2.3) ขั้นตอนทำ Behavioral simulation จะเหมือน CPLD ทุกประการ แล้วให้พิจารณาผลที่ได้ว่าเป็นไปตามทฤษฎีหรือไม่

6.2 การอ่านและเขียนไฟล์ข้อมูลเบื้องต้น

ภาษา VHDL จะไม่มีคำสั่งแสดงผลโดยตรงแต่จะมีเฉพาะคำสั่งอ่านและเขียนไฟล์จากภายนอกโดยการเรียกใช้ Package ชื่อ TEXTIO (เรียกดตาม IEEE Std 1076 ว่า Package TEXTIO) ที่อยู่ในไลบรารีชื่อ Standard ของ IEEE Std 1076 คำสั่งเหล่านี้มีประโยชน์ในการใช้ใส่ค่าเริ่มต้น (Initial value) ให้ RAM และใช้สร้างสัญญาณทดสอบหรือ Test vector โดยการอ่านค่าจากไฟล์ที่อยู่ภายนอก โดยที่ VHDL93 จะต่างกับ VHDL87 ตรงที่ไม่มีคำสั่งเปิด-ปิดไฟล์โดยตรง ซึ่งในทางปฏิบัติอาจไม่จำเป็นต้องใช้

การนิยามของชนิดข้อมูลด้วยการประกาศใช้ชนิดข้อมูล (Type declaration) โดยกำหนดล่วงหน้า (Predefined) ไว้แล้ว ใน Package TEXTIO ของ IEEE Std 1076 การนิยามของชนิดข้อมูลที่สำคัญในกรณีนี้ได้แก่

```
type LINE is access STRING; -- A LINE is a pointer to a STRING value.
```

```
type TEXT is file of STRING; -- A file of variable-length ASCII records.
```

โดยที่ access และ file คือ ชนิดข้อมูล access และชนิดข้อมูล file ตามลำดับ

การประกาศใช้ชนิดข้อมูล (Type declaration) ของชนิดข้อมูล File โดยผู้ใช้งาน (User defined) ซึ่งมาตรฐาน IEEE Std 1076 ไม่ได้นิยามไว้ล่วงหน้า (Predefined) นั้นจะมีรูปแบบดังนี้

```
type FILE_TYPE_NAME is file of TYPE_NAME;
```

โดยที่ TYPE_NAME คือ ชนิดข้อมูล ได้แก่ Scalar type, Record type หรือ Constrained array subtype

การประกาศใช้อbject (Object declaration) ประเภท File (File declearation) ตาม IEEE Std 1076-1993 และ IEEE Std 1076-1987 นั้นจะมีความแตกต่างกัน แต่ XST รองรับการใช้งานทั้ง 2 รูปแบบ โดยมีรูปแบบเป็นดังนี้

```
file INPUT_OBJECT_FILE : text open READ_MODE is "INPUT_FILE.text"; -- VHDL93
```

```
file OUTPUT_OBJECT_FILE : text open WRITE_MODE is "OUTPUT_FILE.text"; -- VHDL93
```

```
file INPUT_OBJECT_FILE : text in is "INPUT_FILE.text"; -- VHDL87
```

```
file OUTPUT_OBJECT_FILE : text out is "OUTPUT_FILE.text"; -- VHDL87
```

โดยที่ INPUT_FILE.text หรือ OUTPUT_FILE.text คือ ไฟล์ข้อมูลที่อ่านเขียนด้วยโปรแกรม Notepad ชื่นนามสกุล อาจใช้ชื่ออื่นๆ ได้ ส่วน INPUT_OBJECT_FILE และ OUTPUT_OBJECT_FILE จะเป็นที่เก็บไฟล์ชั่วคราว

คำสั่งจัดการเกี่ยวกับไฟล์ข้อมูลจะเขียนใน Procedure เพื่อใช้อ่านหรือเขียนเรียงตามลำดับ โดยที่ STD_LOGIC_TEXTIO package (ของบริษัท Synopsys) ที่อยู่ในไลบรารี ieee จะเป็น Overloaded procedure ของ standard TEXTIO คำสั่งที่ใช้บ่อยได้แก่

```
readline(INPUT_OBJECT_FILE, INPUT_LINE); เป็นการสั่งอ่านบรรทัดใหม่จากที่เก็บไฟล์อินพุตชั่วคราว
```

```
read(INPUT_LINE, VALUE); -- Reads a new object from the line.
```

```
write(OUTPUT_LINE, VALUE); -- Writes a new object into the line.
```

```
writeline(OUTPUT_OBJECT_FILE, OUTPUT_LINE); เป็นการสั่งเขียนบรรทัดใหม่จากที่เก็บไฟล์เอาต์พุตชั่วคราว
```

```
endfile(FILE_NAME); -- Returns boolean true if the end of file is reached.
```

การเรียกใช้ Package TEXTIO ที่อยู่ในไลบรารีชื่อ Standard ของ IEEE Std 1076 หรือ STD_LOGIC_TEXTIO package ของบริษัท Synopsys ที่อยู่ในไลบรารี ieee (ซึ่งเป็น Overloaded procedure ของ standard TEXTIO) เป็นตามลำดับดังนี้

```
use STD.TEXTIO.all; หรือ use IEEE.STD_LOGIC_TEXTIO.all; ตามลำดับ
```

ตัวอย่างที่ 6.4 ໂຄ້ງຈາງຈະບວກ 4 ປົດມື Propagation delay time = 5 ns ແລະ ດັດກຳດັ່ງນີ້ (Figure 6.8a) ມີ Testbench ທີ່ສ້າງສໍາຜູນທົດສອບ ຈາກໄຟລ໌ຂໍ້ມູນທີ່ອູ່ງການອກແສດງດັ່ງນີ້ (Figure 6.8b) ໃນຕົວຢ່າງນີ້ຈະແສດງການເລືອກທດສອບເພາະບາງຄ່າທ່ານີ້ ໂດຍໄຟລ໌ຂອງອິນພຸດ ແລະ ໄຟລ໌ຂອງເອົາດຸດື່ກ່ຽວຂ້ອງກຳນົດທີ່ຈະເປັນໄຟລ໌ຂອງຄໍາ Integer (ເປັນເລບຮູ້ານ 10) ແລະ ດັດກຳດັ່ງນີ້ (Figure 6.10c) ແລະ (Figure 6.10d) ຕາມ ລຳດັບ ໄຟລ໌ນີ້ຈະເຫັນແລະອ່ານດ້ວຍໂປຣແກຣມ Notepad ແລະໃຊ້ໄຟລ໌ໆນາມສກຸດ .txt ຜົ່ງໄຟລ໌ຂອງອິນພຸດໃນຮູ້ (Figure 6.10c) ນັ້ນ ຄອລົມນີ້ ທີ່ 1 ເປັນ A ຄອລົມນີ້ 2 ເປັນ B ແລະ ຄອລົມນີ້ 3 ເປັນຜລັພີ່ທີ່ຄາດວ່າຈະໄດ້ (Expected result) ຮັງລັບພົມຕາມກຸມົງລື ສ່ວນໃນ ຮູ້ (Figure 6.10e) ນັ້ນຈະແສດງເອົາດຸດື່ Waveform ແລະ ລາຍງານຜລາທາງໜ້າຕ່າງ Transcript (ທີ່ອູ່ດ້ານລ່າງສຸດ) ຂອງໜ້າຕ່າງ Xilinx-ISE

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ADDER4BIT_TEXTIO is
7     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
8             B : in STD_LOGIC_VECTOR (3 downto 0);
9             C : out STD_LOGIC_VECTOR (4 downto 0));
10 end ADDER4BIT_TEXTIO;
11
12 architecture Behavioral of ADDER4BIT_TEXTIO is
13 begin
14
15     C <= (('0' &A)+('0' &B)) after 5ns; --Propagation delay time = 5 ns
16
17 end Behavioral;

```

6.8a) ໂຄງ ພິບ VHDL ຂອງຈາງຈະບວກ 4 ປົດ

```

2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4 USE ieee.std_logic_arith.all;
5 USE ieee.std_logic_unsigned.all;
6
7 USE std.textio.all;
8
9 ENTITY ADDRE4BIT_TEXTIO_tb1_vhd IS
10 END ADDRE4BIT_TEXTIO_tb1_vhd;
11
12 ARCHITECTURE behavior OF ADDRE4BIT_TEXTIO_tb1_vhd IS
13     -- Component Declaration for the Unit Under Test (UUT)
14     COMPONENT ADDER4BIT_TEXTIO
15         PORT(A : IN std_logic_vector(3 downto 0);
16               B : IN std_logic_vector(3 downto 0);
17               C : OUT std_logic_vector(4 downto 0));
18     END COMPONENT;
19     --Inputs
20     SIGNAL A : std_logic_vector(3 downto 0) := (others=>'0');
21     SIGNAL B : std_logic_vector(3 downto 0) := (others=>'0');
22     --Outputs
23     SIGNAL C : std_logic_vector(4 downto 0);
24 BEGIN
25     -- Instantiate the Unit Under Test (UUT)
26     uut: ADDER4BIT_TEXTIO PORT MAP(A => A,B => B,C => C);
27     -- Stimulus
28     process
29     -- File declaration and variable declaration
30     -- FILE file_ABCin : TEXT IS IN "data_ABC.txt"; --VHDL87
31     -- FILE file_ABCin : TEXT OPEN READ_MODE IS "data_ABC.txt"; --VHDL93
32     -- FILE file_SUM : TEXT IS OUT "data_SUM.txt"; --VHDL87
33     -- FILE file_SUM : TEXT OPEN WRITE_MODE IS "data_SUM.txt"; --VHDL93
34     variable line_ABCin : LINE;
35     variable line_SUM : LINE;
36     variable Ain,Bin,Cin,Cout : integer;
37     variable SUM : std_logic_vector(4 downto 0);
38     begin
39         while NOT ENDFILE(file_ABCin) loop
40             READLINE(file_ABCin,line_ABCin); --Get line of input file.

```

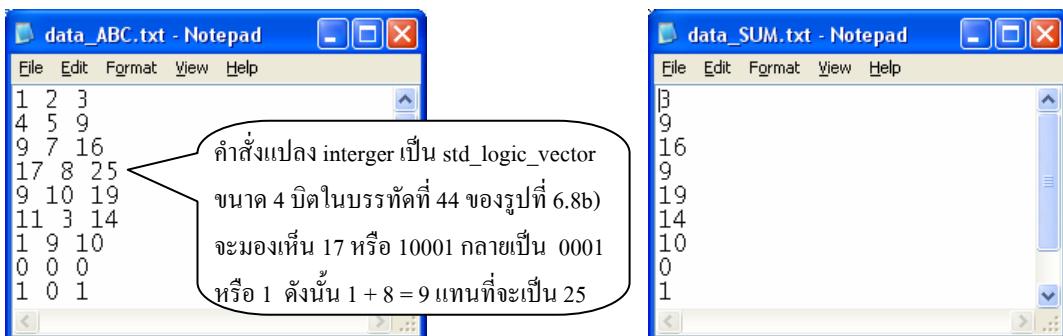
(ຕ້ອ)

```

41           READ(line_ABCin,Ain);          --Get 1st operand.
42           READ(line_ABCin,Bin);          --Get 2nd operand.
43           READ(line_ABCin,Cin);          --Get expected result.
44           A <= conv_std_logic_vector(Ain,4);
45           B <= conv_std_logic_vector(Bin,4);
46           SUM := conv_std_logic_vector(Cin,5);
47           wait for 10 ns;
48           Cout := conv_integer(C);
49           WRITE(line_SUM,Cout);          --Save result to line.
50           WRITELINE(file_SUM,line_SUM);    --Write line to output file.
51           ASSERT C = SUM REPORT "Test failed!" SEVERITY error;
52           wait for 30 ns;
53       end loop;
54   wait; -- will wait forever
55 end process;
56 END behavior;

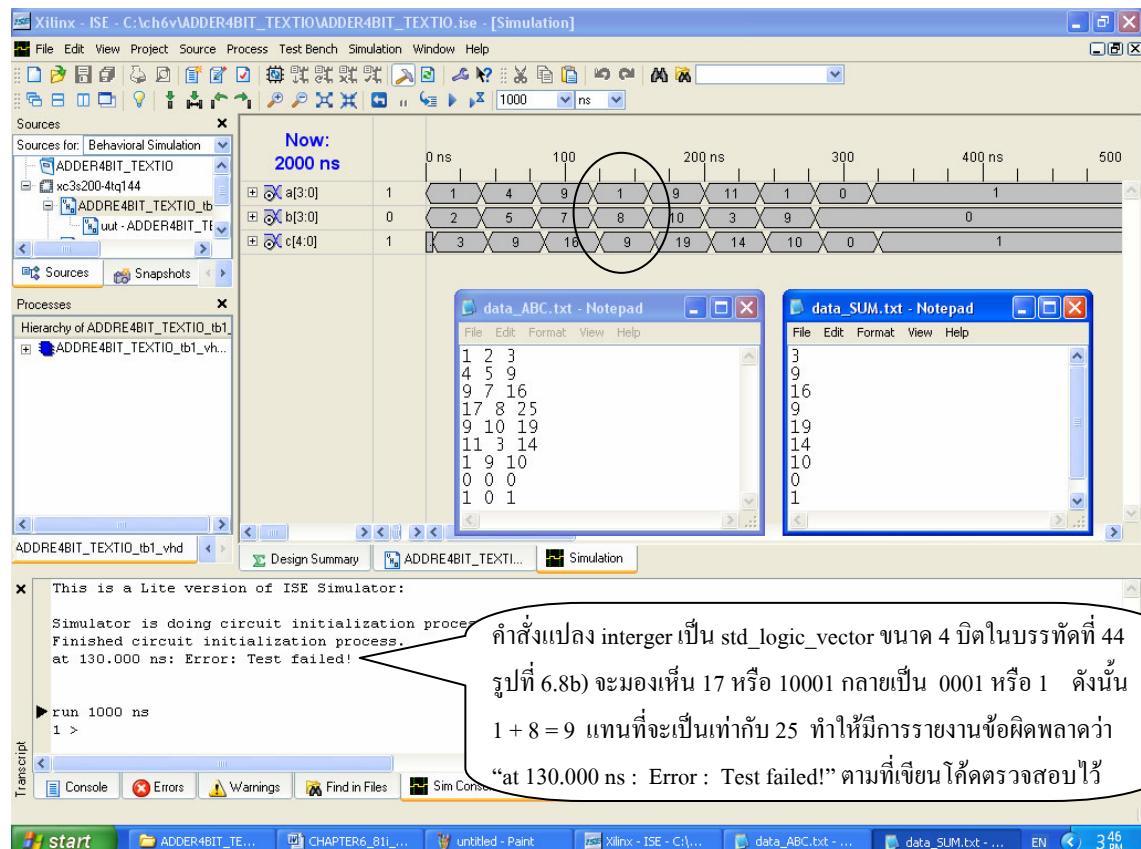
```

6.8b) Testbench ของวงจรบวก 4 บิตที่มีการสร้างสัญญาณทดสอบจากไฟล์ข้อมูลที่อยู่ภายนอก



รูปที่ 6.8c) ไฟล์ integer ของอินพุตชื่อ data_ABC.txt

รูปที่ 6.8d) ไฟล์ integer ของเอาต์พุตชื่อ data_SUM.txt



6.8e) Waveform เอาต์พุตและไฟล์อินพุตและเอาต์พุต (หลังจากป้อนอินพุต 10 ns หรือ wait for 10 ns;) จะให้ผลตรงกัน

รูปที่ 6.8 ໂຄດของวงจรบวก 4 บิตและ Testbench รวมทั้งไฟล์ข้อมูลและผลจำลองการทำงาน

จากรูปที่ 6.8b) ในบรรทัดที่ 4 ต้องเรียกใช้ std_logic_arith package เพราะใช้คำสั่ง conv_std_logic_vector ส่วนในบรรทัดที่ 5 ต้องเรียกใช้ std_logic_unsigned package เพราะใช้คำสั่ง conv_integer และในบรรทัดที่ 7 ต้องเรียกใช้ Package TEXTIO เพราะเราใช้คำสั่งเกี่ยวกับการอ่าน-เขียนไฟล์จากภายนอก (ที่เขียนด้วยโปรแกรม Notepad)

จากรูปที่ 6.8b) ในบรรทัดที่ 31 และบรรทัดที่ 33 เป็นตัวอย่างการประกาศใช้ออบเจคต์ประเภทไฟล์ (File declaration) ที่เขียนตาม VHDL93 และส่วนที่เราใส่ Comment ไว้นั้นจะเป็นวิธีเขียนตาม VHDL87 ซึ่ง XST จะรองรับการเขียนทั้ง 2 วิธี เนื่องจากการอ่าน-เขียนข้อมูลจะเป็นชีวนิรเมชล เราจึงไม่สามารถเลือกอ่านค่าข้อมูลตัวใดตัวหนึ่งหรือบรรทัดใดบรรทัดหนึ่งในลักษณะสุ่มได้ ดังนั้นในบรรทัดที่ 40 จึงเป็นการสั่งให้อ่านข้อมูลที่ละบรรทัด ในขณะที่บรรทัดที่ 41-43 เป็นการสั่งให้อ่านค่าของข้อมูลที่ละค่าในบรรทัดนั้น และในทำนองเดียวกับบรรทัดที่ 49 จะเป็นการสั่งให้เขียนข้อมูลลงในบรรทัดและบรรทัดที่ 50 จะเป็นการเขียนข้อมูลบรรทัดนั้นลงไฟล์ และจากบรรทัดที่ 47 เราใช้คำสั่ง wait for 10 ns; เพราะว่าวงจรบวก 4 บิตมีค่า Propagation delay time = 5 ns ดังนั้นหลังจากป้อนอินพุตเราจะต้องรอเป็นเวลา ≥ 5 ns แล้วจะจบวงจรก็จะให้อาต์พุตค่าใหม่

ในการทำนองเดียวกันเราสามารถเขียน Testbench ที่สร้างสัญญาณทดสอบจากไฟล์ชนิดข้อมูล std_logic_vector ได้เช่นกัน ซึ่งสามารถเขียน Testbench ของโค้ดวงจรบวก 4 บิตในรูปที่ 6.8a) ได้ดังรูปที่ 6.9a) โดยที่ไฟล์ของอินพุตที่เป็นไฟล์ของค่าชนิดข้อมูล std_logic_vector แสดงดังรูปที่ 6.9b) และไฟล์ของเอาต์พุตของค่าที่เป็น std_logic_vector แสดงดังรูปที่ 6.9c) ตามลำดับ ส่วนในรูปที่ 6.9d) นั้นจะแสดงเอาต์พุต Waveform และรายงานผลทางหน้าต่าง Transcript ของหน้าต่าง Xilinx-ISE

จากรูปที่ 6.9a) ในบรรทัดที่ 5-6 นั้นต้องเรียกใช้ Package TEXTIO และ Package ชื่อ STD_LOGIC_TEXTIO เพราะเราใช้คำสั่งเกี่ยวกับการอ่าน-เขียนไฟล์จากภายนอกที่เป็นชนิดข้อมูล std_logic_vector แม้ว่า Package ชื่อ STD_LOGIC_TEXTIO จะคอมไพล์ไว้ในไลบรารี ieee ก็ตามแต่ก็เป็น Non-standard package

```

2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4
5 USE std.textio.all;
6 USE ieee.std_logic_textio.all;
7
8 ENTITY ADDRE4BIT_TEXTIO_tb2_vhd IS
9 END ADDRE4BIT_TEXTIO_tb2_vhd;
10
11 ARCHITECTURE behavior OF ADDRE4BIT_TEXTIO_tb2_vhd IS
12   -- Component Declaration for the Unit Under Test (UUT)
13   COMPONENT ADDER4BIT_TEXTIO
14     PORT(A : IN std_logic_vector(3 downto 0);
15           B : IN std_logic_vector(3 downto 0);
16           C : OUT std_logic_vector(4 downto 0));
17   END COMPONENT;
18   --Inputs
19   SIGNAL A : std_logic_vector(3 downto 0) := (others=>'0');
20   SIGNAL B : std_logic_vector(3 downto 0) := (others=>'0');
21   --Outputs
22   SIGNAL C : std_logic_vector(4 downto 0);
23 BEGIN
24   -- Instantiate the Unit Under Test (UUT)
25   uut: ADDER4BIT_TEXTIO PORT MAP(A => A,B => B,C => C);
26   -- Stimulus
27   process
28     -- File declaration and variable declaration
29     FILE file_ABCin : TEXT IS IN "data_ABCin.txt";          --VHDL87
30     FILE file_ABCin : TEXT OPEN READ_MODE IS "data_ABCin.txt"; --VHDL93
31     FILE file_SUM : TEXT IS OUT "data_SUMout.txt";          --VHDL87
32     FILE file_SUM : TEXT OPEN WRITE_MODE IS "data_SUMout.txt"; --VHDL93
33     variable line_ABCin : LINE;
34     variable line_SUM : LINE;
35     variable Ain,Bin : std_logic_vector(3 downto 0);

```

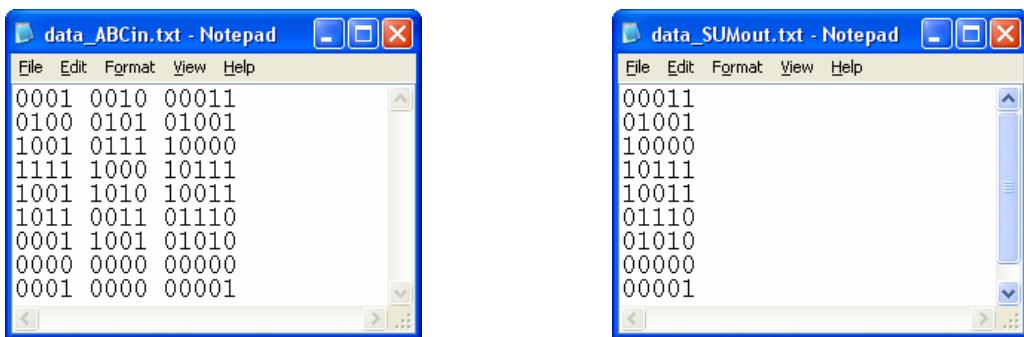
(ต่อ)

```

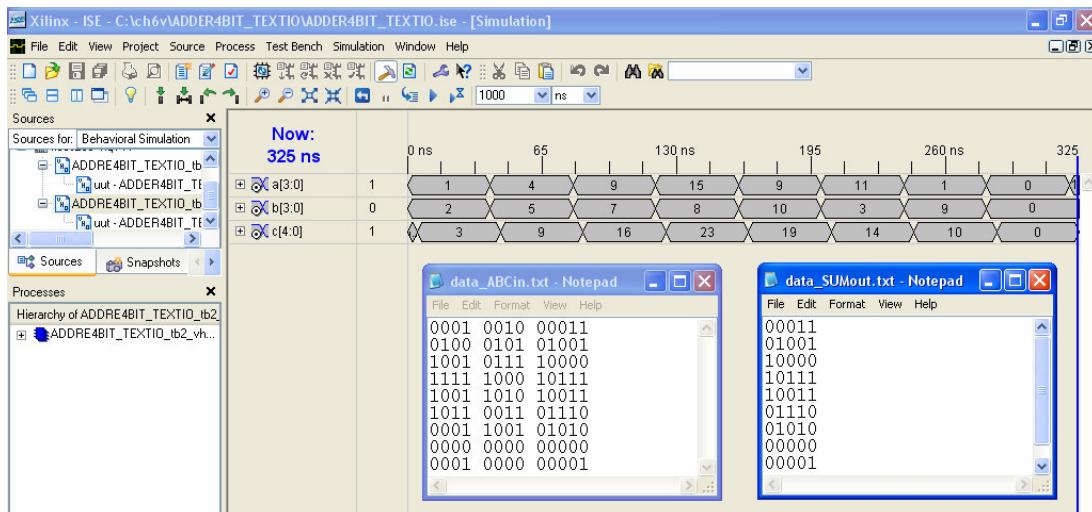
36     variable Cin,Cout : std_logic_vector(4 downto 0);
37 begin
38     while NOT ENDFILE(file_ABCin) loop
39         READLINE(file_ABCin,line_ABCin); --Get line of input file.
40         READ(line_ABCin,Ain); --Get 1st operand.
41         READ(line_ABCin,Bin); --Get 2nd operand.
42         READ(line_ABCin,Cin); --Get expected result.
43         A <= Ain;
44         B <= Bin;
45         wait for 10 ns;
46         WRITE(line_SUM,C); --Save result to line.
47         WRITELINE(file_SUM,line_SUM); --Write line to output file.
48         ASSERT C = Cin REPORT "Test failed!" SEVERITY error;
49         wait for 30 ns;
50     end loop;
51     wait; -- will wait forever
52 end process;
53 END behavior;

```

6.9a) Testbench ของวงจรบวก 4 บิตที่มีการสร้างสัญญาณทดสอบจากไฟล์ชนิดข้อมูล std_logic_vector



6.9b) ไฟล์ชนิดข้อมูล std_logic_vector ชื่อ data_ABCin.txt 6.9c) ไฟล์ชนิดข้อมูล std_logic_vector ชื่อ data_SUMout.txt



6.9d) Waveform เอาท์พุตและไฟล์อินพุตและเอาต์พุต (หลังจากป้อนอินพุต 10 ns หรือ wait for 10 ns;) จะให้ผลตรงกัน

รูปที่ 6.9 Testbench รวมทั้งไฟล์ข้อมูลและผลจำลองการทำงานของวงจรบวก 4 บิต

ตัวอย่างที่ 6.5 ฝึกเขียนโค้ดในการใส่ค่าเริ่มต้น (Initial value) ให้กับ RAM โดยเรียกใช้ STD_LOGIC_TEXTIO Package ซึ่งในตัวอย่างนี้จะใส่ค่าเริ่มต้นให้กับ RAM 4 บิตขนาด 8 Word แสดงดังรูปที่ 6.10a) โดยเราให้อ่านค่าเริ่มต้นที่เป็นชนิดข้อมูล std_logic_vector ทุกค่าจากไฟล์ชื่อ RAM_INITIAL_DATA.txt ที่เราเขียนโดยใช้โปรแกรม Notepad และดังรูปที่ 6.10b)

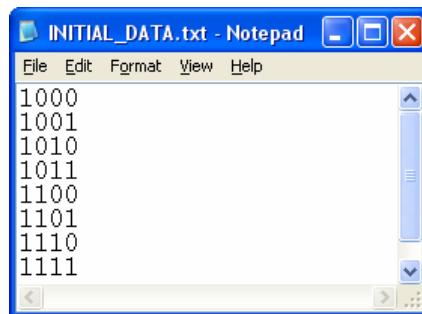
หลังจากใส่ค่าเริ่มต้นให้กับ RAM แล้ว จากนั้นเราจะจำลองการทำงานของ RAM โดยให้อ่านค่าเอาต์พุตจาก RAM ทุกค่าจะได้ดังรูปที่ 6.10c) ซึ่งจะพบว่าค่าเริ่มต้นที่เขียนไว้ในไฟล์จะตรงกับค่าเอาต์พุตทุกๆ Address

```

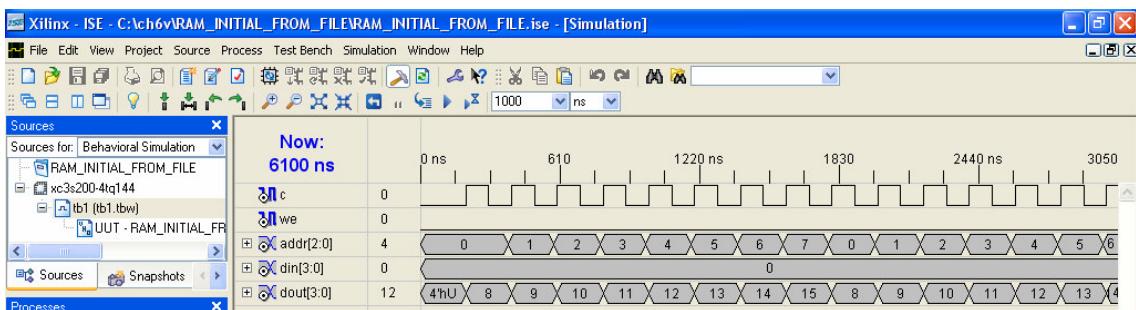
2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_unsigned.all;
5 use std.textio.all;
6 use ieee.std_logic_textio.all;
7
8 entity RAM_INITIAL_FROM_FILE is
9     port(C,WE : in std_logic;
10         ADDR : in std_logic_vector(2 downto 0);
11         Din : in std_logic_vector(3 downto 0);
12         Dout : out std_logic_vector(3 downto 0));
13 end RAM_INITIAL_FROM_FILE;
14
15 architecture Behavioral of RAM_INITIAL_FROM_FILE is
16     type RamType is array(0 to 7) of std_logic_vector(3 downto 0);
17     function InitRamFromFile (RamFileName : in string) return RamType is
18         FILE RamFile : text is in RamFileName;
19         variable RamFileLine : line;
20         variable INITIAL_DATA : RamType;
21     begin
22         for I in RamType'range loop
23             readline (RamFile, RamFileLine);
24             read (RamFileLine, INITIAL_DATA(I));
25         end loop;
26         return INITIAL_DATA;
27     end function;
28     signal RAM : RamType := InitRamFromFile("INITIAL_DATA.txt");--Function call
29 begin
30     process (C)
31     begin
32         if C'event and C = '1' then
33             if WE = '1' then
34                 RAM(conv_integer(ADDR)) <= Din;
35             end if;
36             Dout <= RAM(conv_integer(ADDR));
37         end if;
38     end process;
39 end Behavioral;

```

6.10a) โค้ด VHDL ของ RAM ขนาด 4 บิต 8 Word



6.10b) ค่าเริ่มต้นที่เขียนไว้ในไฟล์ชื่อ INITIAL_DATA.txt



6.10c) Waveform เอาท์พุตที่ได้จากการทำลองการทำงานและไฟล์อินพุตจะให้ผลตรงกัน

รูปที่ 6.10 โค้ด VHDL ของ RAM ขนาด 4 บิต 8 Word พร้อมกับไฟล์ค่าเริ่มต้น

การทดลองที่ 6.2.1 Testbench ของวงจรบวก 4 บิต

วัตถุประสงค์

- 1) เพื่อสร้าง Testbench และทำ Behavioral Simulation ของวงจรบวก 4 บิต
- 2) เพื่อทดลองใช้ซอฟต์แวร์ทุกตัว ISE WebPACK 8.1i สร้างวงจรบวก 4 บิต โดยใช้ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรบวก 4 บิตด้วย CPLD

- 1.1) ออกแบบวงจรบวก 4 บิต ในรูปที่ 6.8a ของตัวอย่างที่ 6.3 แล้วนำโค้ดวงจรนี้ในรูปที่ 6.8a ไปเขียนใหม่ไว้ใน Project Location ชื่อ ch6v แล้วกำหนดให้ Project Name และ Source File ชื่อ ex6_2_1vcxl โดยแก้ชื่อ Entity เป็น ex6_2_1vcxl และดังรูปที่ L1.1 ทำการบันทึกไฟล์และ Check Syntax จากนั้นนำโค้ด Testbench ในรูปที่ 6.8b ไปเขียนใหม่ไว้ใน Source File ชื่อ ex6_2_1vcxl_tb1 และโดยแก้ชื่อ Entity เป็น ex6_2_1vcxl_tb1 และดังรูปที่ L1.2 ทำการบันทึกไฟล์และ Check Syntax

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity ex6_2_1vcxl is
7     Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
8             B : in STD_LOGIC_VECTOR (3 downto 0);
9             C : out STD_LOGIC_VECTOR (4 downto 0));
10 end ex6_2_1vcxl;
11
12 architecture Behavioral of ex6_2_1vcxl is
13 begin
14
15     C <= (('0' &A)+('0' &B));
16
17 end Behavioral;

```

รูปที่ L1.1 โค้ด VHDL ของวงจรบวก 4 บิต (คำว่า after 5 ns ไม่ต้องเขียนเพรา XST จะละเว้น (Ignored))

```

2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4 USE ieee.std_logic_arith.all;
5 USE ieee.std_logic_unsigned.all;
6
7 USE std.textio.all;
8
9 ENTITY ex6_2_1vcxl_tb1_vhd IS
10 END ex6_2_1vcxl_tb1_vhd;
11
12 ARCHITECTURE behavior OF ex6_2_1vcxl_tb1_vhd IS
13     -- Component Declaration for the Unit Under Test (UUT)
14     COMPONENT ex6_2_1vcxl
15         PORT(A : IN std_logic_vector(3 downto 0);
16               B : IN std_logic_vector(3 downto 0);
17               C : OUT std_logic_vector(4 downto 0));
18     END COMPONENT;

```

(ต่อ)

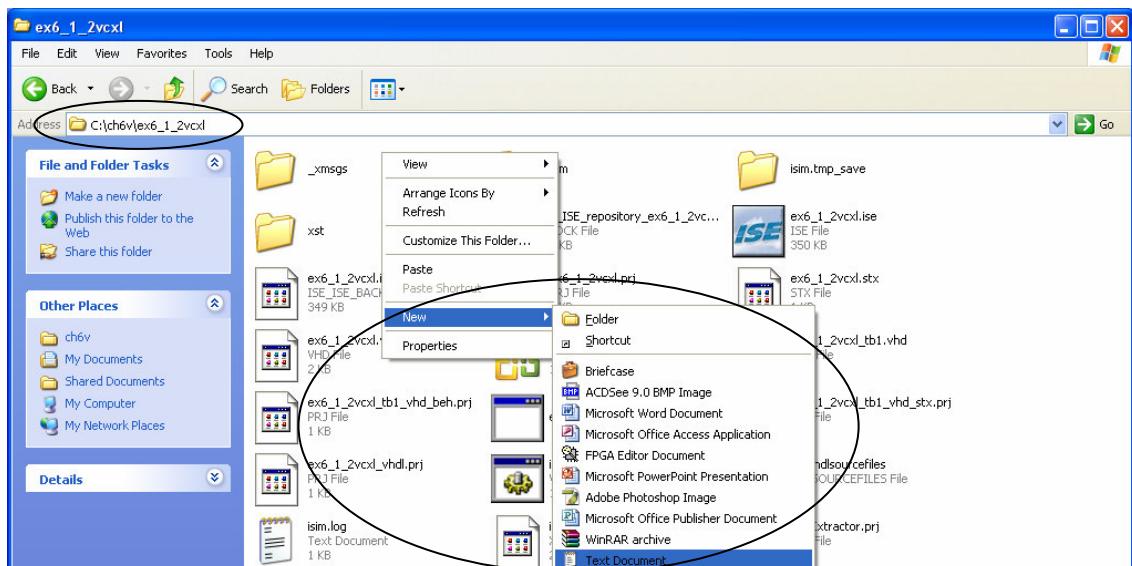
```

19      --Inputs
20      SIGNAL A :  std_logic_vector(3 downto 0) := (others=>'0');
21      SIGNAL B :  std_logic_vector(3 downto 0) := (others=>'0');
22      --Outputs
23      SIGNAL C :  std_logic_vector(4 downto 0);
24 BEGIN
25     -- Instantiate the Unit Under Test (UUT)
26     uut: ex6_2_1vcx1 PORT MAP(A => A,B => B,C => C);
27     -- Stimulus
28     process
29       -- File declaration and variable declaration
30       FILE file_ABCin : TEXT IS IN "data_ABC.txt";          --VHDL87
31       FILE file_ABCin : TEXT OPEN READ_MODE IS "data_ABC.txt"; --VHDL93
32       FILE file_SUM : TEXT IS OUT "data_SUM.txt";           --VHDL87
33       FILE file_SUM : TEXT OPEN WRITE_MODE IS "data_SUM.txt"; --VHDL93
34       variable line_ABCin : LINE;
35       variable line_SUM : LINE;
36       variable Ain,Bin,Cin,Cout : integer;
37       variable SUM : std_logic_vector(4 downto 0);
38 begin
39     while NOT ENDFILE(file_ABCin)  loop
40       READLINE(file_ABCin,line_ABCin);    --Get line of input file.
41       READ(line_ABCin,Ain);             --Get 1st operand.
42       READ(line_ABCin,Bin);            --Get 2nd operand.
43       READ(line_ABCin,Cin);            --Get expected result.
44       A <= conv_std_logic_vector(Ain,4);
45       B <= conv_std_logic_vector(Bin,4);
46       SUM := conv_std_logic_vector(Cin,5);
47       wait for 10 ns;                --Wait the propagation delay time.
48       Cout := conv_integer(C);
49       WRITE(line_SUM,Cout);           --Save result to line.
50       WRITELINE(file_SUM,line_SUM);   --Write line to output file.
51       ASSERT C = SUM REPORT "Test failed!" SEVERITY error;
52       wait for 30 ns;
53     end loop;
54     wait; -- will wait forever
55   end process;
56 END behavior;

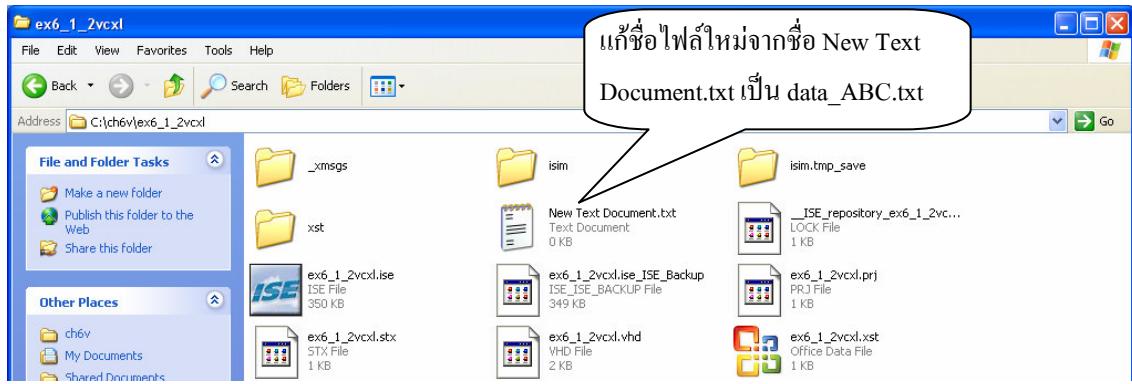
```

รูปที่ L1.2 Testbench ของวงจรบวก 4 บิตที่มีการสร้างสัญญาณทดสอบจากไฟล์ข้อมูลที่อยู่ภายนอก

1.2) การสร้างไฟล์อินพุตและไฟล์เอาต์พุตที่อยู่ภายนอกจะใช้โปรแกรม Notepad โดยเข้าไปที่ Folder ชื่อ C:\ch6v\ex6_2_1vcx1 จากนั้นเลือกมาสีไปที่พื้นที่ว่าง คลิกขวาแล้วเลือกมาสีไปที่ New -> Text Document ดังรูปที่ L1.3 แล้วคลิกที่ Text Document ดังรูปที่ L1.4 แล้วคลิกแก้ไขข้อไฟล์จาก New Text Document.txt เป็นข้อ data_ABC.txt

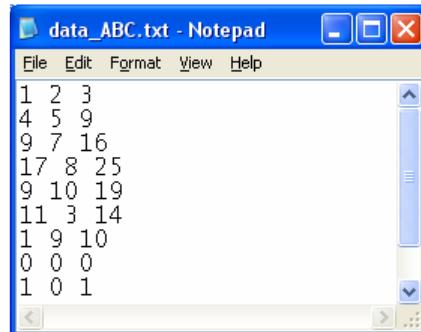


รูปที่ L1.3 ขั้นตอนการสร้างไฟล์อินพุตและไฟล์เอาต์พุตด้วยโปรแกรม Notepad

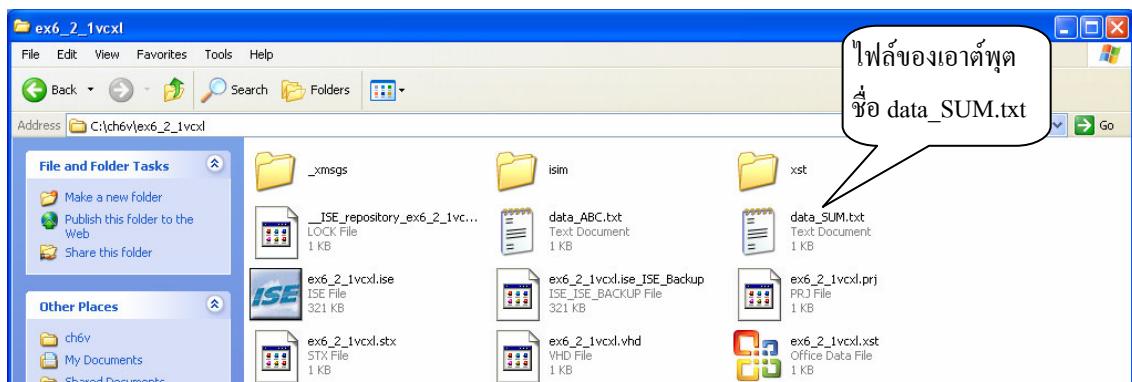


รูปที่ L1.4 ไฟล์ที่สร้างใหม่ชื่อ New Text Document.txt

หลังจากที่แก้ไขไฟล์จากชื่อ New Text Document.txt เป็น data_ABC.txt เสร็จแล้วให้ดับเบิลคลิกที่ชื่อ data_ABC.txt เพื่อเปิดไฟล์และพิมพ์ค่าของอินพุตดังรูปที่ L1.5 แล้วบันทึกไฟล์โดยคลิก File -> Save แล้วปิดโปรแกรม Notepad จากนั้นสร้างไฟล์อ่าดพุตชื่อ data_SUM.txt !! แสดงดังในรูปที่ L1.6 ซึ่งใช้เป็นไฟล์สำหรับรับค่าผลลัพธ์จากการบวก

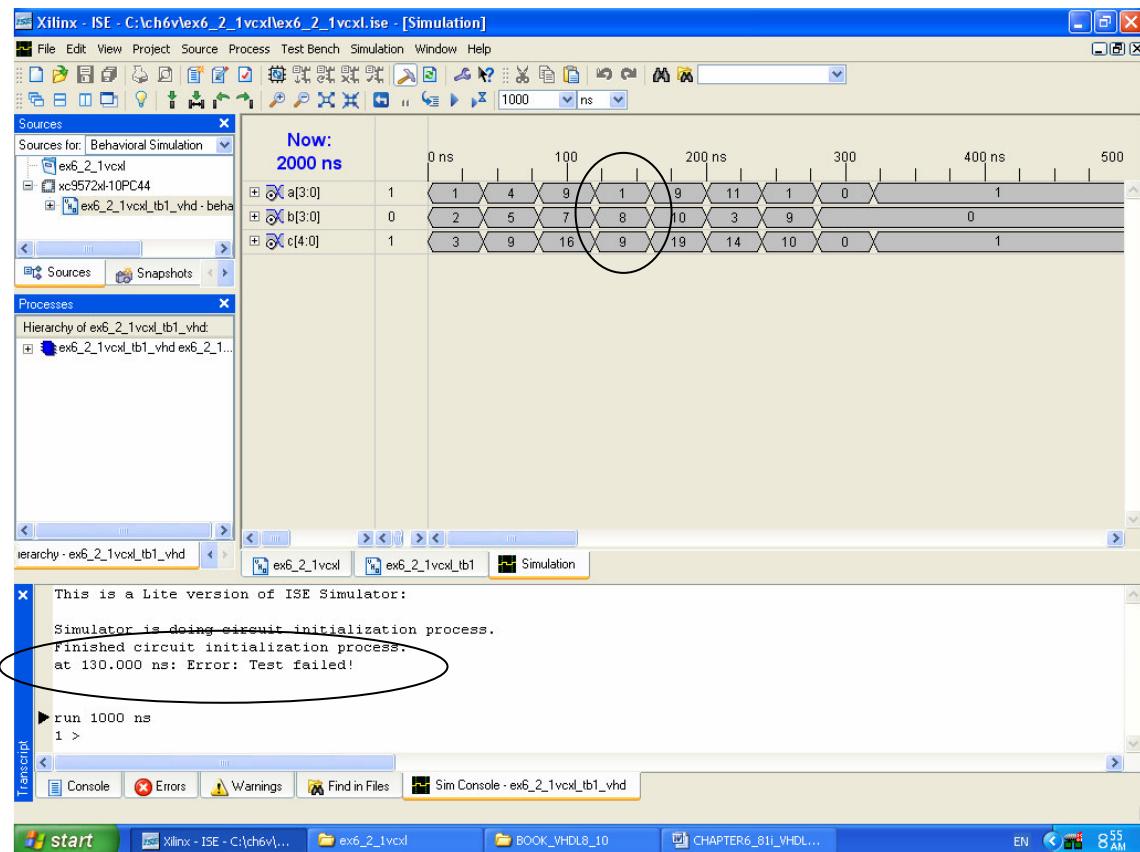


รูปที่ L1.5 ไฟล์ค่า integer ของอินพุตชื่อ data_ABC.txt

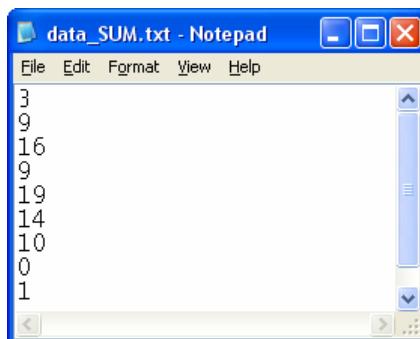


รูปที่ L1.6 ไฟล์ที่สร้างใหม่ชื่อ data_SUM.txt ซึ่งเป็นไฟล์สำหรับรับค่าผลลัพธ์จากการบวก

1.3) ทำ Behavioral Simulation เสร็จแล้วจะได้ผลจำลองการทำงานดังรูปที่ L1.7 ซึ่งในหน้าต่าง Transcript นั้นจะรายงานความผิดพลาดว่า “at 130.000 ns : Error : Test failed!” และให้พิจารณาว่าผลจำลองการทำงานที่ทางหน้าต่างหลักและหน้าต่าง Transcript เป็นตามทฤษฎีหรือไม่ จากนั้นให้เข้าไปคุ้มคลัพธ์ที่ไฟล์อ่าดพุตโดยเข้าไปที่ Folder ชื่อ C:\ch6v\ex6_2_1vcxl และดับเบิลคลิกที่ไฟล์ชื่อ data_SUM.txt แล้วจะได้ดังรูปที่ L1.8 และให้พิจารณาว่าผลจำลองการทำงานที่ปรากฏในไฟล์เป็นตามทฤษฎีหรือไม่



รูปที่ L1.7 Waveform ของเอาต์พุต (หลังจากป้อนอินพุต 10 ns หรือ wait for 10 ns;)



รูปที่ L1.8 ไฟล์ integer ของเอาต์พุตชื่อ data_SUM.txt

1.4) ทำขั้นตอน Synthesis เสร็จแล้วทำการ Design Implementation และ Generate Programming File ตามลำดับ โดยที่ในขั้นตอน Design Implementation นั้นให้กำหนดขาสัญญาณต่างๆ ดังนี้

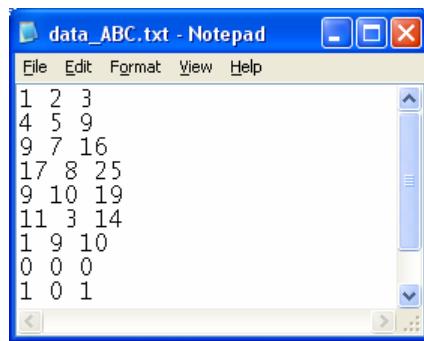
$A(3) = PB1 = INPUT = p39$	$B(3) = PB5 = INPUT = p44$	$C(4) = MN4 = OUTPUT = p3$
$A(2) = PB2 = INPUT = p40$	$B(2) = PB6 = INPUT = p1$	$C(3) = LED1 = OUTPUT = p38$
$A(1) = PB3 = INPUT = p42$	$B(1) = I/O6 of K1 = INPUT = p7$	$C(2) = LED2 = OUTPUT = p37$
$A(0) = PB4 = INPUT = p43$	$B(0) = I/O7 of K1 = INPUT = p6$	$C(1) = LED3 = OUTPUT = p36$
		$C(0) = LED4 = OUTPUT = p35$

หลังจากโปรแกรมวงจรลง CPLD แล้วให้ทำการทดสอบว่าได้ผลบวกเป็นไปตามทฤษฎีหรือไม่

2 สร้างวงจรบวก 4 บิตด้วย FPGA

ออกแบบวงจรบวก 4 บิต ในรูปที่ 6.8a) ของตัวอย่างที่ 6.4 จะมีขั้นตอนต่างๆ เหมือนกับการออกแบบด้วย CPLD ทุกประการที่ได้ทำการทดลองไปแล้วในข้อ 1 โดยนำโค้ดวงจรในรูปที่ L1.1 ไปเขียนใหม่ไว้ใน Project Location ชื่อ ch6v และกำหนดให้ Project Name และ Source File ชื่อ ex6_2_1vf โดยแก้ชื่อ Entity เป็น ex6_2_1vf ส่วนโค้ด Testbench ในรูปที่ L1.2 ไปเขียนใหม่ไว้ใน Source File ชื่อ ex6_2_1vf_tb1 และโดยแก้ชื่อ Entity เป็น ex6_2_1vf_tb1

การสร้างไฟล์อินพุตที่อยู่ภายนอกจะใช้โปรแกรม Notepad โดยเข้าไปที่ Folder ชื่อ C:\ch6v\ex6_2_1vf จากนั้นสร้างไฟล์ใหม่ชื่อ data_ABC.txt เพื่อใช้สำหรับเก็บข้อมูลอินพุตและพิมพ์ค่าของอินพุตดังรูปที่ L2.1 แล้วบันทึกไฟล์และปิดโปรแกรม Notepad จากนั้นสร้างไฟล์เอ้าท์พุตชื่อ data_SUM.txt เพื่อให้เป็นไฟล์สำหรับรับค่าผลลัพธ์จากการบวก



รูปที่ L2.1 ไฟล์ค่า integer ของอินพุตชื่อ data ABC.txt

ทำ Behavioral Simulation เครื่องแล้วให้พิจารณาว่าผลจำลองการทำงานที่ทางหน้าต่างหลักและหน้าต่าง Transcript เป็นตามทฤษฎีหรือไม่ จากนั้นเข้าไปดูผลลัพธ์ที่ไฟล์เอกสารชื่อ data_SUM.txt และให้พิจารณาว่าผลจำลองการทำงานที่ปรากฏในไฟล์เป็นตามทฤษฎีหรือไม่

จากนั้นทำการ Synthesis เสร็จแล้วทำการ Design Implementation และ Generate Programming File ตามลำดับ โดยที่ในขั้นตอน Design Implementation นั้นให้กำหนดขาสัญญาณต่างๆ ดังนี้

A(3) = Dip SW1 = INPUT = p52	B(3) = Dip SW5 = INPUT = p59	C(4) = L4 = OUTPUT = p74
A(2) = Dip SW2 = INPUT = p53	B(2) = Dip SW6 = INPUT = p60	C(3) = L3 = OUTPUT = p76
A(1) = Dip SW3 = INPUT = p55	B(1) = Dip SW7 = INPUT = p63	C(2) = L2 = OUTPUT = p69
A(0) = Dip SW4 = INPUT = p56	B(0) = Dip SW8 = INPUT = p68	C(1) = L1 = OUTPUT = p77
		C(0) = L0 = OUTPUT = p70

หลังจากโปรแกรมวงจรลง FPGA แล้วให้ทำการทดสอบว่าได้ผลบวกเป็นไปตามทฤษฎีหรือไม่

ບຣຣມານຸກຮມ

- 1) IEEE Std 1076-1993 : IEEE Standard VHDL Language Reference Manual.
- 2) IEEE Std 1076-1987 : IEEE Standard VHDL Language Reference Manual.
- 3) IEEE Std 1164-1993 : IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164).
- 4) IEEE Std 1076.3-1997 : IEEE Standard VHDL Synthesis Packages.
- 5) Mentor Graphics VHDL Reference Manual, July 1994.
- 6) FPGA Compiler II / FPGA Express, VHDL Reference Manual Version 1999.05, Synopsys Inc., May 1999.
- 7) VHDL Language Reference, Summary Technical Reference, TR0114 (v1.2), Altium Ltd., September 20, 2005.
- 8) Volnei A. Pedroni, Circuit Design with VHDL, MIT Press, 2004.
- 9) Stephen Brown, Zvonko Vranesic, Fundamentals of Digital Logic with VHDL Design, McGraw-Hill Professional, 2004.
- 10) Sunggu Lee, Advanced Digital Logic Design Using VHDL, State Machines, and Synthesis for FPGAs, 2006.
- 11) Charles H. Roth, Jr., Fundamentals of Logic Design, 2006.
- 12) ISE 8.1i Quick Start Tutorial, Xilinx Inc.
- 13) XST User Guide 8.1i, Xilinx Inc.
- 14) Synthesis and Simulation Design Guide 8.1i, Xilinx Inc.
- 15) ISE 8.2 In-Depth Tutorial, Xilinx Inc.
- 16) Charles H. Roth, Jr., Lizy Kurian John, Digital systems design using VHDL, Toronto, Thomson, 2008.
- 17) Douglas L. Perry VHDL, VHDL : Programming By Example, McGraw-Hill Professional, 2002.

ออกแบบไอซีดิจิตอลด้วย FPGA และ CPLD ภาคปฏิบัติโดยใช้ภาษา VHDL

แผนกวิชาของอุปกรณ์ที่ใช้เทคโนโลยีชั้นสูงไม่ว่าจะเป็น เครื่องมือทางทหาร การแพทย์ ระบบเครือข่าย และสื่อสาร ในปัจจุบัน มีขนาดเล็กและมีประสิทธิภาพสูง คนที่เคยชื่อมเครื่องมือเหล่านี้บ่อยครั้งจะพบว่ามี FPGA หรือ CPLD เป็นส่วนประกอบที่ติดตั้งอยู่บนแผงวงจร สิ่งเหล่านี้เป็นเครื่องยืนยันได้ว่า FPGA และ/หรือ CPLD เข้ามามีบทบาทอย่างมากทางด้านการออกแบบไอซี (IC Design) ในปัจจุบัน หนังสือเล่มนี้จะอธิบายเกี่ยวกับกระบวนการออกแบบไอซีดิจิตอลด้วย FPGA และ CPLD โดยใช้ภาษา VHDL ซึ่งเป็นภาษาระดับสูง โดยจะยกตัวอย่างการออกแบบวงจรต่างๆ ด้วยวิธีการเขียนโค้ดหากาทลายสีต่อเพื่อให้ผู้อ่านได้ศึกษาและทำความเข้าใจ และสามารถทดลองได้จริงทุกการทดลอง ได้แก่ ลอจิกเกต แลดตช์ ฟลิปฟลופ วงจรคอมบินेशัน และวงจรซีเควนเชียล รวมทั้งวงจรที่ใช้งานจริงในทางปฏิบัติ หนังสือเล่มนี้จึงเหมาะสมสำหรับนักศึกษา อาจารย์ และผู้สนใจทางด้านการออกแบบวงจรดิจิตอลภาคปฏิบัติตั้งแต่ระดับพื้นฐานจนถึงระดับสูง โดยมีใบงานการทดลองต่างๆ มากกว่า 40 การทดลอง

ประวัติผู้เขียน

นาย ณรงค์ ทองนิม สำเร็จการศึกษาวิศวกรรมศาสตรบัณฑิต สาขาวิชกรรมไฟฟ้า มหาวิทยาลัยสงขลานครินทร์ ปี 2529 และ วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชกรรมไฟฟ้ากำลัง จุฬาลงกรณ์มหาวิทยาลัย ปี 2533 ปัจจุบันเป็น กรรมการผู้จัดการ บริษัท เอเพก อินสตรูเมนต์ จำกัด มีผลงานวิชาการทางด้าน FPGA ในต่างประเทศจำนวน 3 บทความ งานทางด้านวิชาการ เป็นอดีตอาจารย์พิเศษ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยธรรมศาสตร์ และ เป็นอดีตนักวิจัย ศูนย์เชี่ยวชาญพิเศษเฉพาะด้านเทคโนโลยีไฟฟ้ากำลัง จุฬาลงกรณ์มหาวิทยาลัย

E-mail : support@ailogictechnology.com

นายเจริญ วงศ์ชุมยืน สำเร็จการศึกษาวิศวกรรมศาสตรบัณฑิต (เกียรตินิยมอันดับ 1) สาขาวิชกรรมคอมพิวเตอร์ ปี 2541 และ วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชกรรมไฟฟ้า ปี 2544 จากสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง เป็นอาจารย์ประจำ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง เป็นอาจารย์พิเศษ คณะวิศวกรรมศาสตร์หลักสถาบัน มีผลงานด้านวิชาการระดับนานาชาติจำนวน 5 บทความ และมีประสบการณ์ในการออกแบบและตรวจสอบความถูกต้อง (Design verification) ไมโครโปรเซสเซอร์ ARM7 ที่เขียนด้วยภาษา VHDL

E-mail : kvocharo@kmitl.ac.th

คำนำ

ดิจิตอลเข้ามายืนหนาทอย่างมากในชีวิตประจำวันทุกวันนี้ อุปกรณ์และเครื่องใช้ต่างๆ ล้วนมีดิจิตอลเข้าไปเกี่ยวข้องไม่ว่าจะเป็นคอมพิวเตอร์ เครื่องใช้ภายในบ้าน ภายนอกบ้าน ทางทหาร ทางการแพทย์ ระบบเครือข่าย และ สื่อสาร เป็นต้น

การออกแบบชาร์ดแวร์คิจิตอลในปัจจุบันนั้นเปลี่ยนแปลงอย่างรวดเร็ว นับวันวงจรคิจิตอลจะมีขนาดใหญ่และซับซ้อนขึ้นเรื่อยๆ ทางเลือกในการออกแบบ IC เพื่อผลิตชาร์ดแวร์คิจิตอลจำนวนมากกว่ากันเป็นล้านชิ้นควรเป็นการออกแบบ IC ด้วย ASIC (Application-specific Integrated circuit) ซึ่งต้นทุนต่อหน่วยถูกมาก แต่ถ้าเป็นการสร้างต้นแบบหรือผลิตไปถึงระดับหลักแสนชิ้น FPGA (Field Programmable Gate Array) หรือ CPLD (Complex Programmable Logic Device) จะเป็นทางเลือกที่ดีกว่า เนื่องจากต้นทุนเริ่มต้นต่ำกว่ามากและเมื่อเกิดข้อผิดพลาดก็สามารถทำการแก้ไขได้ง่าย

ผู้ประกอบการ หรือ นักวิจัยทางคณิตศาสตร์ที่มีความตั้งใจที่จะออกแบบวงจรดิจิตอลเองแต่หาชิปไฮบริดที่ตรงกับความต้องการไม่ได้ ทำให้นักคณิตศาสตร์ต้องล้มเลิกความตั้งใจ ทางคณิตศาสตร์สัญญาการทำไม้แพ่งวงจรของอุปกรณ์ที่ใช้เทคโนโลยีชิ้นสูงเช่นมีนาคเล็กมากแต่กลับมีประสิทธิภาพสูง ความสำเร็จดังกล่าววนเวียนส่วนหนึ่งมาจากการออกแบบโดยนักด้านการออกแบบ IC (IC Design) คนที่เคยซ่อนอุปกรณ์ที่ใช้เทคโนโลยีชิ้นสูงบอยครึ่งจะพบว่ามี FPGA หรือ CPLD เป็นส่วนประกอบที่ติดตั้งอยู่บนแพ่งวงจร สิ่งเหล่านี้เป็นเครื่องยืนยันได้ว่า FPGA และ/หรือ CPLD เข้ามามีบทบาทอย่างมากทางด้านการออกแบบ IC ดิจิตอลในปัจจุบัน โดยที่ประเทศไทยได้เริ่มใช้ FPGA ในงานวิจัยเมื่อประมาณ 10 กว่าปีมานี้ทั้งๆ ที่มีการผลิต FPGA ประมาณกว่า 20 ปีแล้ว ปัญหานั้นที่เกิดขึ้นในระยะเริ่มแรก คือ ซอฟต์แวร์ที่ไม่สามารถเขียนโปรแกรม FPGA ได้ราบรื่นมาก และคอมพิวเตอร์ในสมัยนั้นยังทำงานได้ช้า

หนังสือ “ออกแบบไอซีดิจิตอลด้วย FPGA และ CPLD ภาคปฏิบัติ” นี้จะอธิบายเกี่ยวกับการออกแบบวงจรดิจิตอลและสามารถทำการทดลองจริงในภาคปฏิบัติได้ทุกการทดลอง โดยมีการทดลองต่างๆ มากกว่า 40 การทดลอง โดยเริ่มตั้งแต่การออกแบบดิจิตอลขั้นพื้นฐาน เช่น ลอจิกเกต แลตช์ ฟลิปฟล็อป วงจรคอมบินेशัน และ วงจรรีเซเวอร์ชั่น เป็นต้น ไปถึงวงจรซึ่งใช้งานได้จริง โดยที่วงจรเหล่านี้ถูกออกแบบรวมไว้ในไอซี FPGA หรือ CPLD เพียงตัวเดียวเท่านั้น จึงสมอ่อนเป็นการสร้างไอซีเบอร์ใหม่ไว้ใช้งาน ซึ่งไม่มีขายโดยทั่วไป เราเรียกกระบวนการนี้ว่า “การออกแบบไอซี” ซึ่งแน่นอนว่าเราจำกัดเนื้อหาของหนังสือเล่มนี้อยู่ที่การออกแบบไอซีดิจิตอลโดยใช้ FPGA หรือ CPLD เท่านั้น

หนังสือเล่มนี้จะเป็นการออกแบบวงจรคิจิตอคติ้วยภาษา VHDL ที่เป็นการออกแบบด้วยภาระดับสูง ซึ่งเน้นการใช้ความสามารถของซอฟต์แวร์ทูลกอปั่งเป็นระบบวิธีที่ใช้กันในทางปฏิบัติ โดยอธิบายและยกตัวอย่างการออกแบบวงจรต่างๆ ด้วย การเขียน โค้ดหลากหลายลักษณะ ไม่เพื่อให้ผู้อ่านได้ศึกษาและทำความเข้าใจแทนการใช้คำบรรยาย ทำให้เข้าใจได้ง่ายขึ้น

ผู้เขียนหวังเป็นอย่างยิ่งว่าหนังสือเล่มนี้จะช่วยเติมเต็มความรู้ด้าน “การออกแบบໄออีซี” ซึ่งเป็นอุตสาหกรรม “ด้านน้ำ” ภายในประเทศไทยที่ “มีมูลค่าทรัพย์สินทางปัญญาสูง” และอุตสาหกรรมต่อเนื่องด้านอิเล็กทรอนิกส์และคอมพิวเตอร์ฮาร์ดแวร์ให้มีความเจริญรุ่งเรืองต่อไป

นาย ณรงค์ ทองนิม

นายเจริญ วงศ์ชุ่มເບີນ

สารบัญ

บทที่ 1 การออกแบบวงจรดิจิตอลด้วย FPGA และ CPLD.....	9
1.1 ไอซีมาตรฐาน.....	9
1.2 CPLD.....	10
1.3 FPGA.....	11
1.4 ข้อเปรียบเทียบระหว่าง FPGA และ CPLD กับไมโครคอนโทรลเลอร์.....	12
1.5 กระบวนการออกแบบวงจรดิจิตอลด้วย FPGA หรือ CPLD.....	12
1.6 ขั้นตอนการออกแบบวงจรดิจิตอลด้วย FPGA หรือ CPLD.....	15
1.7 การติดตั้งซอฟต์แวร์ทุก.....	18
1.8 บอร์ดทดลองที่ใช้.....	21
1.8.1 CPLD Explorer XC9572XL.....	21
1.8.2 CPLD Explorer XC9572.....	25
1.8.3 FPGA Discovery-III XC3S200.....	26
1.8.4 บอร์ดทดลองรุ่นอื่นๆ.....	35
บทที่ 2 ภาษา VHDL.....	36
2.1 บทนำ.....	36
2.2 VHDL พื้นฐาน.....	36
2.3 กฎเกณฑ์ทั่วไป.....	40
2.4 การตั้งชื่อ.....	41
2.5 ประเภทของพอร์ต.....	42
2.6 การเขียนโค้ด VHDL เมื่องต้นโดยใช้ชนิดข้อมูล std_logic และ std_logic_vector.....	42
2.7 ออบเจ็คต์.....	43
2.8 ชนิดข้อมูลและชนิดข้อมูลย่อย.....	46
2.8.1 ชนิดข้อมูลตามมาตรฐาน IEEE Std 1076.....	47
2.8.2 ชนิดข้อมูลตามมาตรฐาน IEEE Std 1164-1993.....	50
2.8.3 ชนิดข้อมูลตามมาตรฐาน IEEE Std 1076.3-1997.....	53
2.9 ตัวดำเนินการ.....	56
2.9.1 ตัวดำเนินการตรรกะ.....	57
2.9.2 ตัวดำเนินการความสัมพันธ์.....	58
2.9.3 ตัวดำเนินการรวมข้อมูล.....	58
2.9.4 ตัวดำเนินการคณิตศาสตร์.....	58
2.9.5 การแปลงชนิดข้อมูล.....	69
2.9.6 ตัวดำเนินการเลื่อน.....	62

2.10 รูปแบบการเขียนโค๊ด VHDL ทั่วไป.....	65
2.10.1 การเรียกใช้ Package.....	65
2.10.2 ประกาศใช้ออนติคี.....	65
2.10.3 อาร์เทกเชอร์.....	66
2.11 Attribute.....	68
2.11.1 Predefined attributes.....	69
2.11.2 User defined attributes.....	71
2.12 คำสั่งที่ใช้ในการเขียนโค๊ด VHDL.....	71
2.13 คำสั่งคอนเคอร์เรนต์.....	72
2.13.1 Simple signal assignment statement.....	72
2.13.2 Selected signal assignment statement.....	72
2.13.3 Conditional signal assignment statement.....	73
2.13.4 Component instantiation statement.....	82
2.13.5 Generate statement.....	86
2.13.6 Block statement.....	90
2.13.7 Process statement.....	91
2.13.8 Concurrent procedure calls.....	92
2.14 คำสั่งชีวนะชีบล.....	92
2.14.1 Signal assignment statement.....	92
2.14.2 Variable assignment statement.....	92
2.14.3 if statement.....	93
2.14.4 case statement.....	93
2.14.5 Wait statement.....	116
2.14.6 Loop statement.....	119
2.15 ตัวอย่างการออกแบบวงจรที่ใช้งานจริง.....	121
2.16 การใช้ซอฟต์แวร์ทูลอกแบบวงจรดิจิตอลโดยใช้ CPLD.....	124
2.16.1 ขั้นตอนการออกแบบวงจร (Design entry).....	124
2.16.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification).....	133
2.16.3 การสังเคราะห์วงจร (Design synthesis).....	137
2.16.4 Design Implementation.....	140
2.16.5 การโปรแกรมชุดวงจรชิป CPLD.....	144
2.16.6 การเปิดไฟล์ของ CPLD ที่เคยออกแบบไว้แล้วมาใช้งาน.....	146
2.17 การใช้ซอฟต์แวร์ทูลอกแบบวงจรดิจิตอลโดยใช้ FPGA.....	148
2.17.1 ขั้นตอนการออกแบบวงจร (Design entry).....	148
2.17.2 การตรวจสอบความถูกต้องของวงจรที่ออกแบบ (Design Verification).....	156
2.17.3 การสังเคราะห์วงจร (Design synthesis).....	160
2.17.4 Design Implementation.....	163

2.17.5 การโปรแกรมข้อมูลวงจรลงชิป.....	166
2.17.6 การเปิดไฟล์ของ FPGA ที่เกยออกแบบໄว้แล้วมาใช้งาน.....	175
2.18 การใช้ซอฟต์แวร์ทูลออกแบบวงจรดิจิตอลที่มีคอมโพเนนต์รวมอยู่ด้วยโดยใช้ CPLD.....	175
2.18.1 ขั้นตอนการสร้างไฟล์คอมโพเนนต์.....	175
2.18.2 ขั้นตอนการสร้างโค้ดหลัก (Main code) สำหรับการออกแบบวงจรดิจิตอล โดยใช้ CPLD.....	178
2.19 การใช้ซอฟต์แวร์ทูลออกแบบวงจรดิจิตอลที่มีคอมโพเนนต์รวมอยู่ด้วยโดยใช้ FPGA.....	189
2.20 การกำหนด Property ให้กับซอฟต์แวร์ทูลในการออกแบบวงจรดิจิตอล.....	192
2.21 Level of abstraction.....	193
2.22 Transport delay และ Inertial delay.....	196
2.23 การเขียนโค้ดเพื่อการสังเคราะห์วงจร.....	197
บทที่ 3 ลอกิกเกตและวงจรคอมบินেชัน.....	198
3.1 ลอกิกเกต.....	198
การทดลองที่ 3.1.1 ลอกิกเกตต่างๆ.....	203
การทดลองที่ 3.1.2 Multiple-drivers.....	206
การทดลองที่ 3.1.3 บัฟเฟอร์สองทิศทาง.....	209
3.2 วงจรบวก-ลบ.....	211
การทดลองที่ 3.2.1 วงจร Half adder.....	213
การทดลองที่ 3.2.2 วงจร Full adder.....	215
การทดลองที่ 3.2.3 วงจรบวก-ลบขนาดหลายบิตโดยใช้ Arithmetic operators.....	217
3.3 วงจรดอครหัส.....	220
การทดลองที่ 3.3.1 วงจร 2 to 4 Decoder.....	223
การทดลองที่ 3.3.2 วงจรดอครหัสตัวแสดงผลเซกเมนต์.....	225
การทดลองที่ 3.3.3 วงจรบวกเลข 3 บิตที่แสดงผลทางตัวแสดงผลเซกเมนต์.....	228
3.4 การสร้างวงจรเบรี่ยนเทียบ.....	230
การทดลองที่ 3.4.1 การสร้างวงจรเบรี่ยนเทียบ.....	231
3.5 วงจรเข้ารหัส.....	233
การทดลองที่ 3.5.1 วงจรเข้ารหัส.....	234
3.6 วงจรมัลติเพล็กเซอร์.....	236
การทดลองที่ 3.6.1 วงจร 4 to 1 Multiplexer.....	237
3.7 วงจรคิมัลติเพล็กเซอร์.....	239
การทดลองที่ 3.7.1 วงจรคิมัลติเพล็กเซอร์.....	240
บทที่ 4 แลดต์ ฟลิปฟลอป และวงจรรีคอนเซยล.....	242
4.1 แลดต์.....	242
การทดลองที่ 4.1.1 แลดต์.....	244

4.2 ฟลิปฟลอป.....	247
การทดลองที่ 4.2.1 D Flip-Flop แบบอะซิงโกรนัสเคลียร์.....	253
การทดลองที่ 4.2.2 D Flip-Flop แบบซิงโกรนัสเซต.....	255
การทดลองที่ 4.2.3 วงจรดีเบาเซอร์อย่างง่าย.....	257
การทดลองที่ 4.2.4 JK Flip-Flop และ T Flip-Flop แบบอะซิงโกรนัสเคลียร์.....	260
4.3 วงจรนับเลขไบนาเรีย (Binary counter).....	264
4.3.1 วงจรนับแบบอะซิงโกรนัส (Asynchronous counter).....	264
4.3.2 วงจรนับแบบซิงโกรนัส (Synchronous counter).....	267
การทดลองที่ 4.3.1 การสร้างวงจรนับขึ้นแบบริปเปิล.....	273
การทดลองที่ 4.3.2 การสร้างวงจรนับลงแบบริปเปิล.....	280
การทดลองที่ 4.3.3 การสร้างวงจรนับขึ้นแบบซิงโกรนัส.....	281
การทดลองที่ 4.3.4 การสร้างวงจรนับลงแบบซิงโกรนัส.....	285
การทดลองที่ 4.3.5 ใช้ความสามารถของซอฟต์แวร์ทูลช่วยในการออกแบบวงจรนับ.....	286
4.4 การออกแบบวงจรนับ N.....	291
การทดลองที่ 4.4.1 การสร้างวงจรนับ 10 แบบซิงโกรนัสที่เป็นวงจรนับขึ้น.....	296
การทดลองที่ 4.4.2 การสร้างวงจรนับ 10 ที่เป็นวงจรนับขึ้น-นับลงแบบคากเดค 1 หลัก.....	300
การทดลองที่ 4.4.3 การสร้างวงจรนับเลขฐานสิบที่เป็นวงจรนับขึ้น-นับลง 2 หลัก.....	306
การทดลองที่ 4.4.4 การสร้างวงจรนับเลขฐานสิบที่เป็นวงจรนับขึ้น-นับลง 4 หลัก.....	311
การทดลองที่ 4.4.5 นาฬิกาดิจิตอลแบบตั้งเวลาหลักนาทีและชั่วโมงได้.....	316
4.5 ชิปต์รีจิสเตอร์.....	322
การทดลองที่ 4.5.1 การออกแบบวงจรชิปต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางขวา.....	325
การทดลองที่ 4.5.2 การออกแบบวงจรชิปต์รีจิสเตอร์แบบเลื่อนข้อมูลไปทางซ้าย.....	329
การทดลองที่ 4.5.3 การออกแบบวงจรดีเบาเซอร์สำหรับปุ่มกดแบบเบาซ์เดส.....	330
การทดลองที่ 4.5.4 การออกแบบวงจรดีเบาเซอร์สำหรับปุ่มกดแบบโนโนสเตเบิล.....	334
4.6 การออกแบบวงจรชีควนเชียลโดยใช้ชีวี Finite state machine.....	335
4.6.1 Finite state machine.....	335
4.6.2 State diagram และ State table ของ Moor-type FSM.....	336
4.6.3 State diagram และ State table ของ Mealy-type FSM.....	338
4.6.4 ขั้นตอนการออกแบบ Finite state machine.....	339
4.6.5 การออกแบบ Finite state machine โดยใช้ซอฟต์แวร์ทูล.....	340
4.6.6 State encoding.....	354
4.6.7 Incompletely specified state table.....	356
การทดลองที่ 4.6.1 ออกแบบวงจรนับ 10 แบบซิงโกรนัสโดยใช้ชีวี FSM.....	357
การทดลองที่ 4.6.2 วงจรตรวจจับลำดับแบบ Moor-type FSM.....	362
การทดลองที่ 4.6.3 วงจรตรวจจับลำดับแบบ Mealy-type FSM.....	365
การทดลองที่ 4.6.4 หน่วยควบคุมเครื่องขายน้ำอัดลมอัตโนมัติ.....	368

4.7 ROM และ RAM.....	375
การทดลองที่ 4.7.1 ROM.....	378
การทดลองที่ 4.7.2 RAM.....	380
บทที่ 5 Package, Library และ Subprogram.....	383
5.1 Package.....	383
5.1.1 การประกาศใช้ Package.....	383
5.1.2 Package body.....	383
5.1.3 การเรียกใช้ Package.....	384
5.2 Library.....	386
5.3 Subprograms.....	388
5.3.1 Function.....	388
5.3.2 Procedure.....	394
การทดลองที่ 5.1 การประกาศใช้ Constant ไว้ใน Package.....	398
การทดลองที่ 5.2 การประกาศใช้ Component ไว้ใน Package.....	403
การทดลองที่ 5.3 การเขียน Function.....	408
บทที่ 6 Testbench และ Package TEXTIO.....	411
6.1 Testbench.....	411
6.1.1 การสร้าง Testbench.....	414
6.1.2 การสร้างสัญญาณนาฬิกา.....	415
6.1.3 การสร้างสัญญาณทดสอบ.....	415
6.1.4 การแสดงผลและตรวจสอบข้อผิดพลาดโดยอัตโนมัติ.....	418
การทดลองที่ 6.1.1 Testbench ของบันไฟฟอร์ส่องทิศทาง.....	421
การทดลองที่ 6.1.2 Testbench ของวงจรตรวจจับลำดับที่เป็น Mealy-type FSM.....	426
6.2 การอ่านและเขียนไฟล์ข้อมูลเบื้องต้น.....	429
การทดลองที่ 6.2.1 Testbench ของวงจรบวก 4 บิต.....	435
บรรณานุกรม.....	440