รายการอ้างอิง

ภาษาไทย

กิตติ ภักดีวัฒนะกุล และ กิตติพงษ์ กลมกล่อม. UML – วิเคราะห์และออกแบบระบบเชิงวัตถุ. พิมพ์ครั้งที่ 1. กรุงเทพมหานคร: เคทีพี คอมพ์ แอนด์ คอนซัลท์, 2544.

ศรัณย์ ชัยวรวิทย์กุล. การประยุกต์ใช้แนวคิดเชิงวัตถุในการออกแบบวงจรตรรกะเชิงผสม. วิทยานิพนธ์ปริญญามหาบัณฑิต, ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย, 2546.

ภาษาอังกฤษ

Ashenden, P., and Wilsey, P. Considerations on Object-Oriented Extensions to VHDL. VIUF Conference 1997 : 109-118.

Budd, T. An Introduction to Object-Oriented Programming. 2nd ed. United States of America: Addison Wesley, 1997.

Chaiworawitgul, S., Pitsatorn, P., and Sowanwanichakul, B. An Application of Object Oriented Paradigm (OOP) to Combination Logic Design. The 2003 International MultiConference in Computer Science and Engineering 2003 : 582-587.

Chaiworawitgul, S., Pitsatorn, P., and Sowanwanichakul, B. Applying Object Oriented Concept to Hardware Design. Software Engineering and Applications 2004.

Damasevicius, R., and Stuikys, V. Application of UML for hardware design based on design process model. Design Automation Conference 2004 : 244-249.

Djafri B., and Benzakki J. OOVHDL: object oriented VHDL. VHDL International Users' Forum 1997 : 54-59.

FlexGraphic Software. FlexGraphic for Borland Delphi/C++Builder[Online]. 2003. Available from: http://www.flex-graphics.com.

Hsiao, M. S. International Symposium on Circuits And Systems 1989[Online]. 1989. Available from: http://www.ece.vt.edu/mhsiao/iscas89.html.

Hutchings, B., and Nelson, B. Using General-Purpose Programming Languages for FPGA Design. Conference on Design Automation 2000 : 561-566.

Ing.Büro R.Tschaggelar. Delphi - Graphic Editor / take 2[Online]. 2003. Available from: http://www.ibrtses.com/delphi/graphicedit2.html.

Jacome, M., and Peicoto, H. A Survey of Digital Design Reuse. IEEE Design & Test of Computer 2001 : 98-107.

Jordan Russell. Toolbar2000[Online]. 2000. Available from: http://www.jrsoftware.org/tb2k.php.

Kuhn, T., Oppold, T., Edwards, M., and Kashai, Y. A Framework for Object Oriented Hardware Specification, Verification, and Synthesis. Design Automation Conference 2001 : 413-418.

Kuhn, T., Oppold, T., Edwards, M., and Kashai, Y. Object Oriented Hardware Synthesis and Verification. Design Automation Conference 2001 : 189-194.

Mano, M. M. Digital Design. 3rd ed. United States of America: Prentice-Hall, 2002.

Nebel W., and Schumacher, G. Object-Oriented Hardware Modelling Where to apply and what are the Objects?. EURO-DAC '96 with EURO-VHDL '96 1996 : 428-433.

Priestle, M. Practical Object-Oriented Design with UML. Singapore: McGraw-Hill, 2000.

Schildt, H. Java 2: A Beginner's Guide. United States of America: McGraw-Hill, 2001.

Schumacher, G., and Nebel W. Inheritance Concept for Signals in Object-Oriented Extensions to VHDL. European Design Automation Conference with EURO-VHDL '95 on EURO-DAC 1995 : 428-435.

Vollmar, K. R. Hardware Simulation Tools for Computer Design. Journal of Computing Sciences in Colleges Volume 17 Issue 3 (February 2002) : 231-239.

ภาคผนวก

ภาคผนวก ก
วิธีการใช้เครื่องมือออกแบบ
วงจรตรรกะเชิงลำดับโดยใช้แนวคิดเชิงวัตถุ

เครื่องมือออกแบบวงจรตรรกะเชิงลำดับ โดยใช้แนวคิดเชิงวัตถุที่พัฒนาขึ้นในวิทยานิพนธ์นี้ ประกอบด้วยเครื่องมือทั้งหมด 3 เครื่องมือ ซึ่งพัฒนาขึ้นภายในโปรแกรมบอร์แลนด์เดลไฟ เวอร์ชัน 7.0 เนื้อหาในส่วนนี้จะอธิบายถึงวิธีการใช้เครื่องมือออกแบบวงจรตรรกะเชิงลำดับโดยใช้แนวคิดเชิงวัตถุที่พัฒนาขึ้น ซึ่งแบ่งการอธิบายออกเป็น 5 ส่วน คือ (1) การติดตั้งเครื่องมือที่พัฒนาขึ้นในโปรแกรม (2) การเรียกใช้งานโปรแกรม (3) การใช้งานส่วนออกแบบวงจรแบบกราฟิก (4) การใช้งานเครื่องมือสังเคราะห์วงจร และ (5) การใช้งานเครื่องมือจำลองการทำงาน ดังรายละเอียดดังต่อไปนี้

ก.1 การติดตั้งเครื่องมือที่พัฒนาขึ้นในโปรแกรม

เครื่องมือออกแบบวงจรตรรกะเชิงลำดับ โดยใช้แนวคิดเชิงวัตถุที่พัฒนาขึ้นในวิทยานิพนธ์นี้ เนื่องจากในการพัฒนาเครื่องมือส่วนกราฟิก ได้นำไลบรารีเวกเตอร์กราฟิกของโปรแกรมเดลไฟที่ชื่อ FlexGraphics มาใช้ ดังนั้นผู้ใช้จำเป็นต้องติดตั้งไลบรารีกราฟิกเวกเตอร์ของโปรแกรมเดลไฟก่อนติดตั้งเครื่องมือที่ได้พัฒนาขึ้น ซึ่งมีขั้นตอนทั้งหมด 2 ขั้นตอน คือ

1. ติดตั้งไลบรารี Toolbar2000 (Russell, 2000), RX Library v2.75, GridView Library และ FlexGraphics (FlexGraphic Software, 2003) โดยทำตามคู่มือจากไฟล์คู่มือ
2. คัดลอกโฟลเดอร์ชื่อ FlexEdit ซึ่งเป็นโฟลเดอร์เครื่องมือออกแบบวงจรตรรกะเชิงลำดับที่ถูกพัฒนาขึ้นไปยังตำแหน่งที่ต้องการ

ก.2 การเรียกใช้งานเครื่องมือออกแบบวงจรตรรกะเชิงลำดับ

การเรียกใช้งานเครื่องมือออกแบบวงจรตรรกะเชิงลำดับ โดยใช้แนวคิดเชิงวัตถุที่ถูกพัฒนาขึ้น หากมีการติดตั้งไลบรารีทั้งหมดเรียบร้อยแล้ว ให้ผู้ใช้เปิดไฟล์ FlexEdit.dpr ที่อยู่ในโฟลเดอร์ FlexEdit ที่ได้คัดลอกไว้ หลังจากนั้นให้ผู้ใช้รันโปรแกรมโดยการกด F9 หรือกดปุ่มรัน หรือเลือกเมนู Run->Run

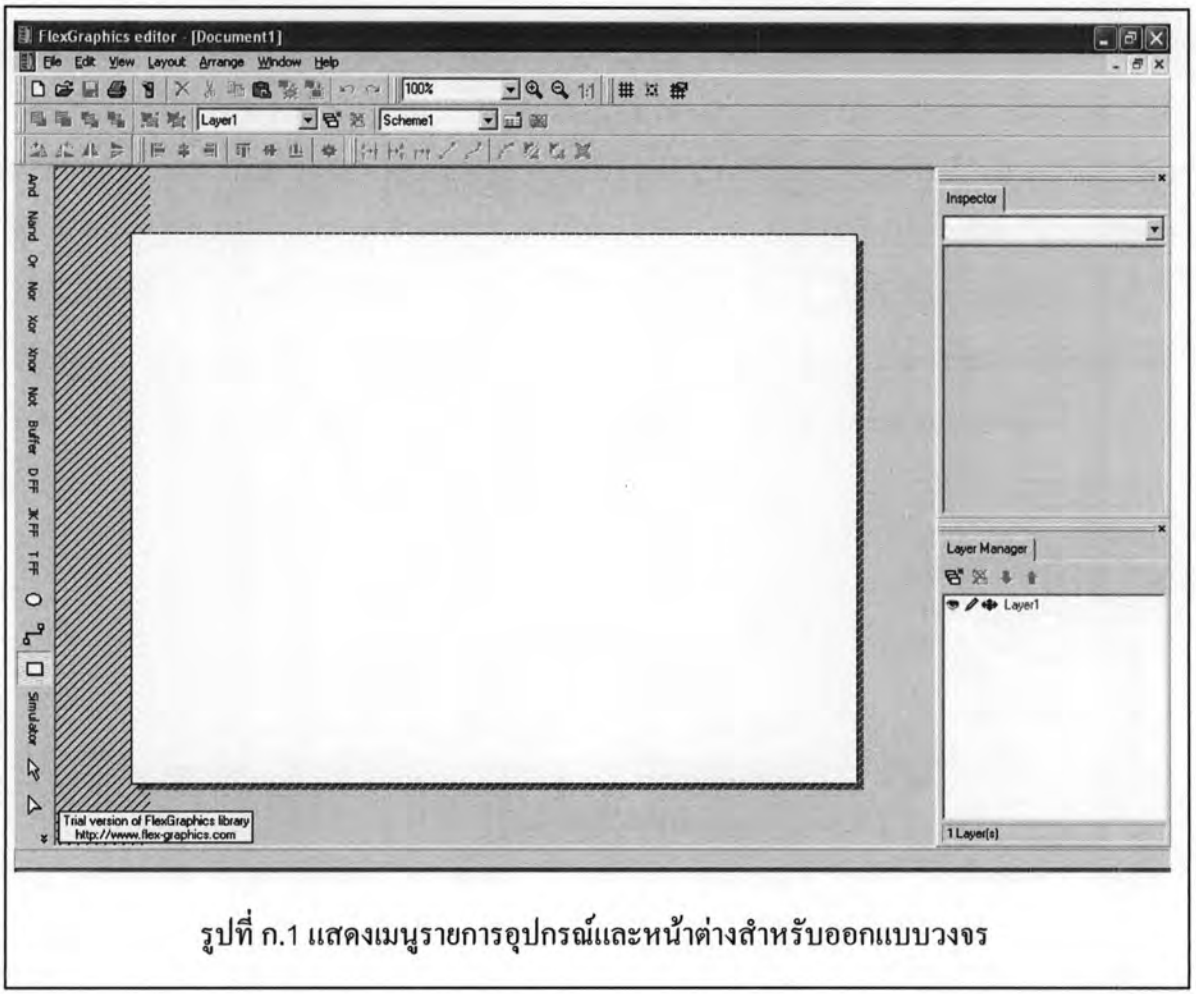
ก.3 การใช้งานส่วนออกแบบวงจรแบบกราฟิก

การใช้งานส่วนออกแบบวงจรแบบกราฟิก สามารถแบ่งการอธิบายออกได้เป็น 4 ส่วน คือ 1) การใช้เกทและฟลิปฟลอปสำหรับออกแบบวงจร 2) การใช้สายสัญญาณสำหรับออกแบบวงจร 3) การ

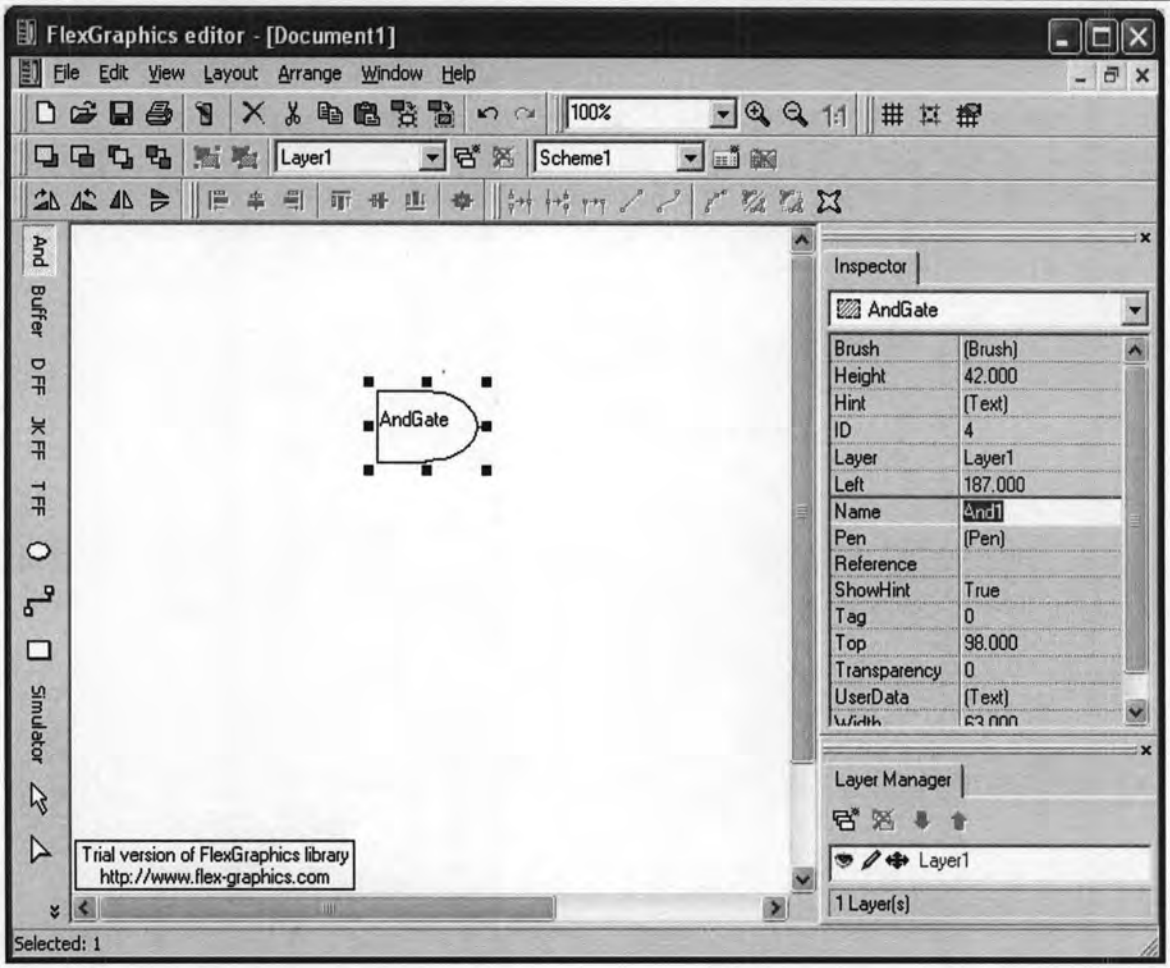
ใช้จุดเชื่อมต่อสำหรับออกแบบวงจร และ 4) การใช้วงจรบล็อกสำหรับออกแบบวงจร ซึ่งมีรายละเอียดดังต่อไปนี้

ก.3.1 การใช้เกทสำหรับออกแบบวงจร

รูป ก.1 แสดงส่วนออกแบบวงจรแบบกราฟิก ซึ่งประกอบด้วย รูปร่างหลักของอุปกรณ์สำหรับออกแบบวงจรตรรกะเชิงลำดับซึ่งอยู่ด้านซ้ายมือ และหน้าต่างออกแบบวงจรตรรกะเชิงลำดับ รูปแบบหลักของเกทและฟลิปฟลอปที่ถูกพัฒนาขึ้นในวิทยานิพนธ์นี้จำนวน 11 รูปร่างหลัก คือ 1) AND 2) OR 3) BUFFER 4) NOT 5) NAND 6) NOR 7) XOR 8) XNOR 9) DFF 10) JKFF และ 11) TFF ผู้ใช้สามารถนำเกทและฟลิปฟลอปดังกล่าวมาใช้ในการออกแบบวงจร โดยเมื่อคลิกเลือกอุปกรณ์ที่เมนูด้านซ้ายแล้ว ให้คลิกที่หน้าต่างสำหรับออกแบบวงจรครั้งหนึ่งแล้วลากเกทหรือฟลิปฟลอปที่ต้องการใช้เพื่อกำหนดขนาดของอุปกรณ์ที่จะออกแบบได้ นอกจากนี้ผู้ใช้ยังสามารถตั้งชื่อให้ทุกอุปกรณ์ รวมทั้งสายสัญญาณและวงจรบล็อกได้ที่เมนูด้านขวา ดังแสดงในรูปที่ ก.2



รูปที่ ก.1 แสดงเมนูรายการอุปกรณ์และหน้าต่างสำหรับออกแบบวงจร



รูปที่ ก.2 แสดงการวาดอุปกรณ์แอนด์เกท

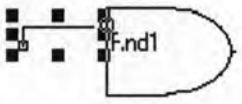
ก.3.2 การใช้สายสัญญาณสำหรับออกแบบวงจร

สายสัญญาณที่ใช้ในเครื่องมือออกแบบวงจรที่ได้พัฒนาขึ้นนี้มีทิศทางการใช้งาน 2 ทิศทาง ได้แก่ทิศแนวตั้ง และทิศแนวนอน ผู้ใช้สามารถลากสายสัญญาณจากจุดเริ่มต้นไปยังจุดสุดท้ายซึ่งจะอยู่ทิศทางเดียวกับจุดเริ่มต้นหรืออยู่คนละทิศทางก็ได้ ซึ่งเครื่องมือจะทำการปรับทิศทางของสายสัญญาณให้เป็นระเบียบ 2 ทิศทางให้อัตโนมัติ และการใช้สายสัญญาณในการออกแบบวงจรตรรกะเชิงลำดับสำหรับเครื่องมือที่พัฒนาขึ้นนี้ สามารถแบ่งอธิบายการใช้สายสัญญาณออกได้เป็น 2 รูปแบบ คือ 1) การใช้สายสัญญาณเพื่อเป็นอินพุตของอุปกรณ์ใดๆ และ 2) การใช้สายสัญญาณเพื่อเป็นเอาต์พุตของอุปกรณ์ใดๆ ดังรายละเอียดต่อไปนี้

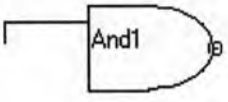
ก.3.2.1 การใช้สายสัญญาณเพื่อเป็นอินพุตของอุปกรณ์ใดๆ เมื่อต้องการเชื่อมต่อสายสัญญาณเข้ากับกับอุปกรณ์ใดๆ เพื่อเป็นอินพุตของอุปกรณ์นั้น เมื่อผู้ใช้เลือกอุปกรณ์สายสัญญาณจากเมนูแล้ว ให้ผู้ใช้คลิกจุดเริ่มต้นของ

สายสัญญาณจุดใดๆ ในหน้าต่างออกแบบวงจร แล้วลากไปปล่อยยังบริเวณเชื่อมต่อระหว่างสายสัญญาณและอุปกรณ์นั้นๆ จุดสังเกตบริเวณเชื่อมต่อระหว่างสายสัญญาณและอุปกรณ์ ก็คือบริเวณที่มีวงกลมสีฟ้าขึ้น เวลาผู้ใช้ลากสายสัญญาณไปทับบริเวณขอบเส้นของอุปกรณ์ที่ต้องการ ดังแสดงในรูปที่ ก.3

ก.3.2.2 การใช้สายสัญญาณเพื่อเป็นเอาต์พุตของอุปกรณ์ใดๆ เมื่อต้องการเชื่อมต่อสายสัญญาณเข้ากับกับอุปกรณ์ใดๆ เพื่อเป็นเอาต์พุตของอุปกรณ์นั้น เมื่อผู้ใช้เลือกอุปกรณ์สายสัญญาณจากเมนูแล้ว ให้ผู้ใช้คลิกจุดเริ่มต้นของสายสัญญาณที่บริเวณเชื่อมต่อระหว่างสายสัญญาณและอุปกรณ์นั้นๆ ซึ่งจุดสังเกตบริเวณเชื่อมต่อระหว่างสายสัญญาณและอุปกรณ์ก็คือบริเวณที่มีวงกลมสีฟ้าขึ้น เวลาผู้ใช้ผ่านเมาส์ไปทับบริเวณขอบเส้นของอุปกรณ์ที่ต้องการ เมื่อคลิกจุดเริ่มต้นของสายสัญญาณแล้ว ให้ลากไปปล่อยยังจุดใดๆ ในหน้าต่างออกแบบวงจร หรือลากไปปล่อยยังอุปกรณ์อื่นๆ เพื่อเป็นอินพุตของอุปกรณ์อื่นๆ ก็ได้ ดังแสดงในรูปที่ ก.4



รูปที่ ก.3 แสดงการเชื่อมต่อสายสัญญาณเพื่อเป็นอินพุตให้แอนด์เกท

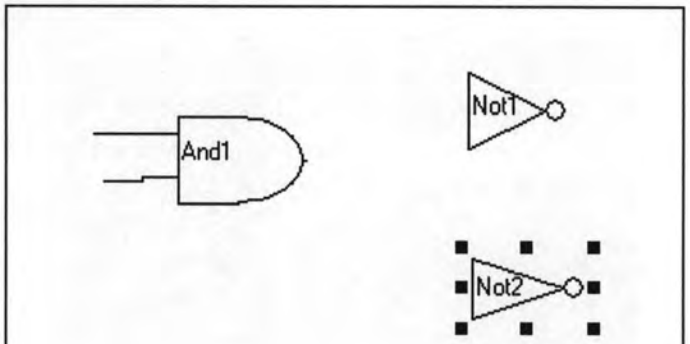


รูปที่ ก.4 แสดงวงกลมสีฟ้าที่จะแสดง
เมื่อมีเมาส์ผ่านตรงบริเวณเชื่อมต่อระหว่างอุปกรณ์กับสายสัญญาณ

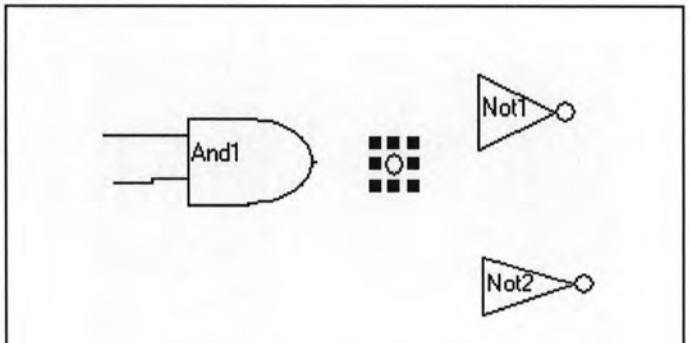
ก.3.3 การใช้จุดเชื่อมต่อสำหรับออกแบบวงจร

จุดเชื่อมต่อใช้ในการออกแบบวงจรเมื่อสายสัญญาณใดๆ ที่ถูกออกแบบในวงจรถูกแยกออกเป็นอินพุตของเกทที่มีจำนวนมากกว่าหนึ่งเกท รูปร่างหลักของจุดเชื่อมต่อคือ การใช้งานจุดเชื่อมต่อสำหรับเครื่องมือออกแบบวงจรที่ได้พัฒนาขึ้นนี้ ทำได้โดยการลากจุดปลายของสายสัญญาณ

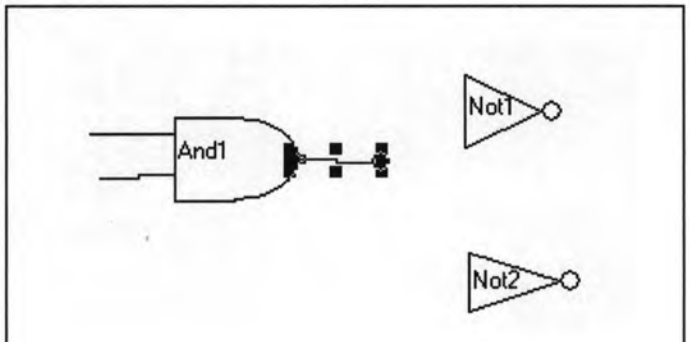
ที่ต้องการแยกออกไปเป็นอินพุตของอุปกรณ์อีกอุปกรณ์หนึ่ง ซึ่งได้ออกแบบไว้แล้วไปเชื่อมต่อเข้ากับจุดเชื่อมต่อที่ได้เลือกมาวาดยังหน้าต่างการออกแบบแล้ว หลังจากนั้นให้นำจุดเริ่มต้นของสายสัญญาณมาเชื่อมต่อเข้ากับจุดเชื่อมต่อในทิศทางที่ผู้ใช้ต้องการออกแบบ ซึ่งจุดเชื่อมต่อจะสามารถนำจุดเริ่มต้นของสายสัญญาณมาเชื่อมต่อเข้ากับจุดเชื่อมต่อได้ 1-3 จุด และผู้ใช้สามารถเลือกจุดเชื่อมต่อเป็นจุดปลายของสายสัญญาณหรือเป็นจุดเริ่มต้นของสายสัญญาณได้ 4 ทิศทาง คือด้านบน ด้านล่าง ด้านซ้าย และด้านขวา ดังแสดงขั้นตอนการใช้งานจุดเชื่อมต่อในรูปที่ ก.5 - ก.8



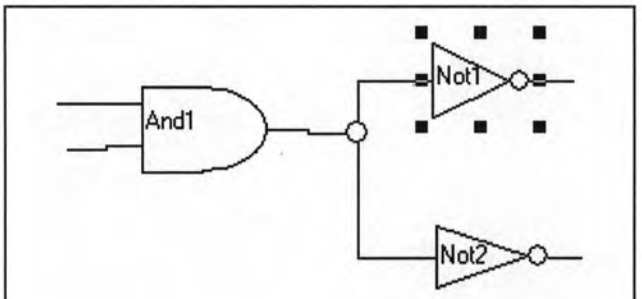
รูปที่ ก.5 แสดงการใช้งานจุดเชื่อมต่อ



รูปที่ ก.6 แสดงวาดจุดเชื่อมต่อ



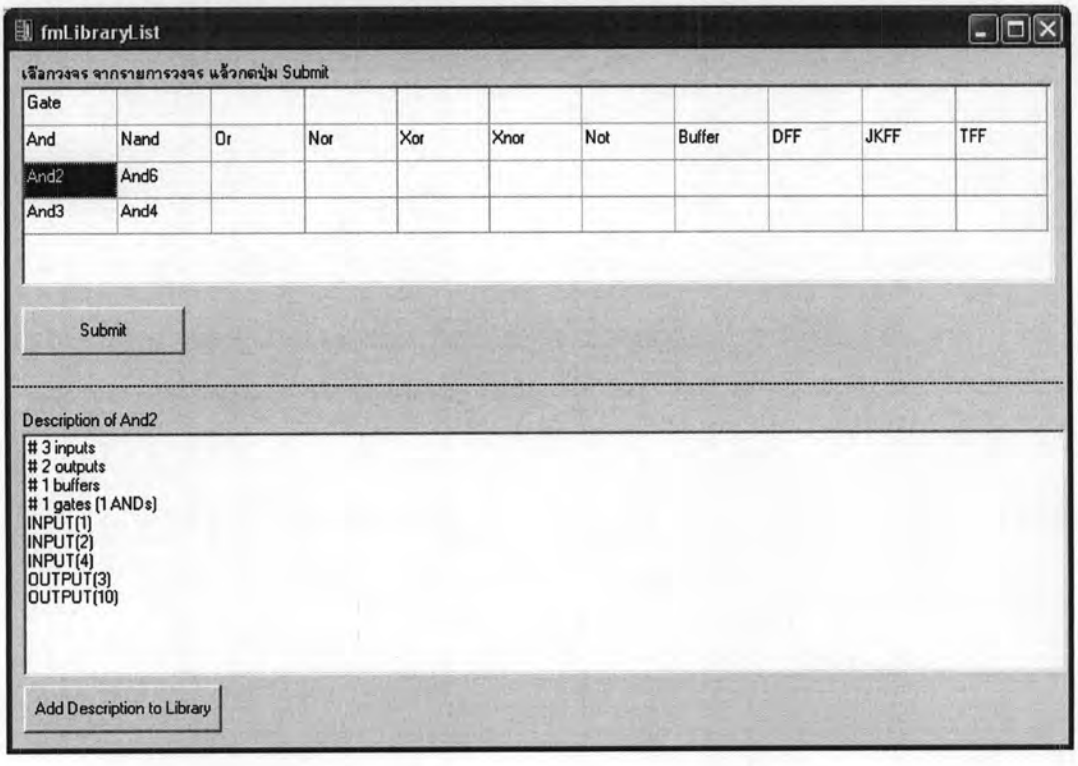
รูปที่ ก.7 แสดงการลากสายสัญญาณเข้าจุดเชื่อมต่อ



รูปที่ ก.8 แสดงลากสายสัญญาณออกจากจุดเชื่อมต่อ

ก.3.4 การใช้วงจรบล็อกสำหรับออกแบบวงจร

วงจรบล็อก คือ วงจรที่ถูกออกแบบและสังเคราะห์วงจรไว้แล้ว ซึ่งผู้ใช้สามารถนำวงจรดังกล่าวมาใช้ในการออกแบบวงจรที่มีวงจรบล็อกเป็นส่วนประกอบได้ โดยที่ผู้ใช้ไม่จำเป็นต้องออกแบบวงจรบล็อกใหม่อีกครั้งหนึ่ง สำหรับวงจรบล็อกมีวิธีการใช้งานในเครื่องมือที่ได้พัฒนาขึ้น โดยเริ่มต้นจากการเลือก ☐ และเลือกวงจรจากตารางการถ่ายทอด แล้วมาวาดลงในหน้าต่างของการออกแบบ ดังแสดงในรูปที่ ก.9 และ ก.10 ส่วนรายละเอียดการเชื่อมต่อสายสัญญาณกับวงจรบล็อกได้กล่าวอย่างละเอียดแล้วในบทที่ 4 หัวข้อที่ 4



รูปที่ ก.9 แสดงการเลือกวงจรจากตารางการถ่ายทอดเพื่อสร้างวงจรบล็อก

รูปที่ ก.10 แสดงการวาดวงจรบล็อกลงในหน้าต่างการออกแบบ

ก.4 การใช้งานเครื่องมือสังเคราะห์วงจร

เครื่องมือสังเคราะห์วงจรที่ได้พัฒนาขึ้น จะทำงานอย่างอัตโนมัติหลังจากผู้ใช้ได้ออกแบบวงจรตรรกะเชิงลำดับเสร็จและทำการจัดเก็บวงจร (Save หรือ Save as) เป็นที่เรียบร้อยแล้ว

ก.5 การใช้งานเครื่องมือจำลองการทำงาน Verilogger Pro

เครื่องมือจำลองการทำงาน Verilogger Pro จะสามารถเรียกทำงานได้ก็ต่อเมื่อผู้ใช้ได้ออกแบบวงจรตรรกะเชิงลำดับเสร็จและสังเคราะห์วงจรเป็นที่เรียบร้อยแล้ว ซึ่งผู้ใช้สามารถเรียกใช้เครื่องมือจำลองการทำงาน Verilogger Pro ได้ 2 วิธี ได้แก่

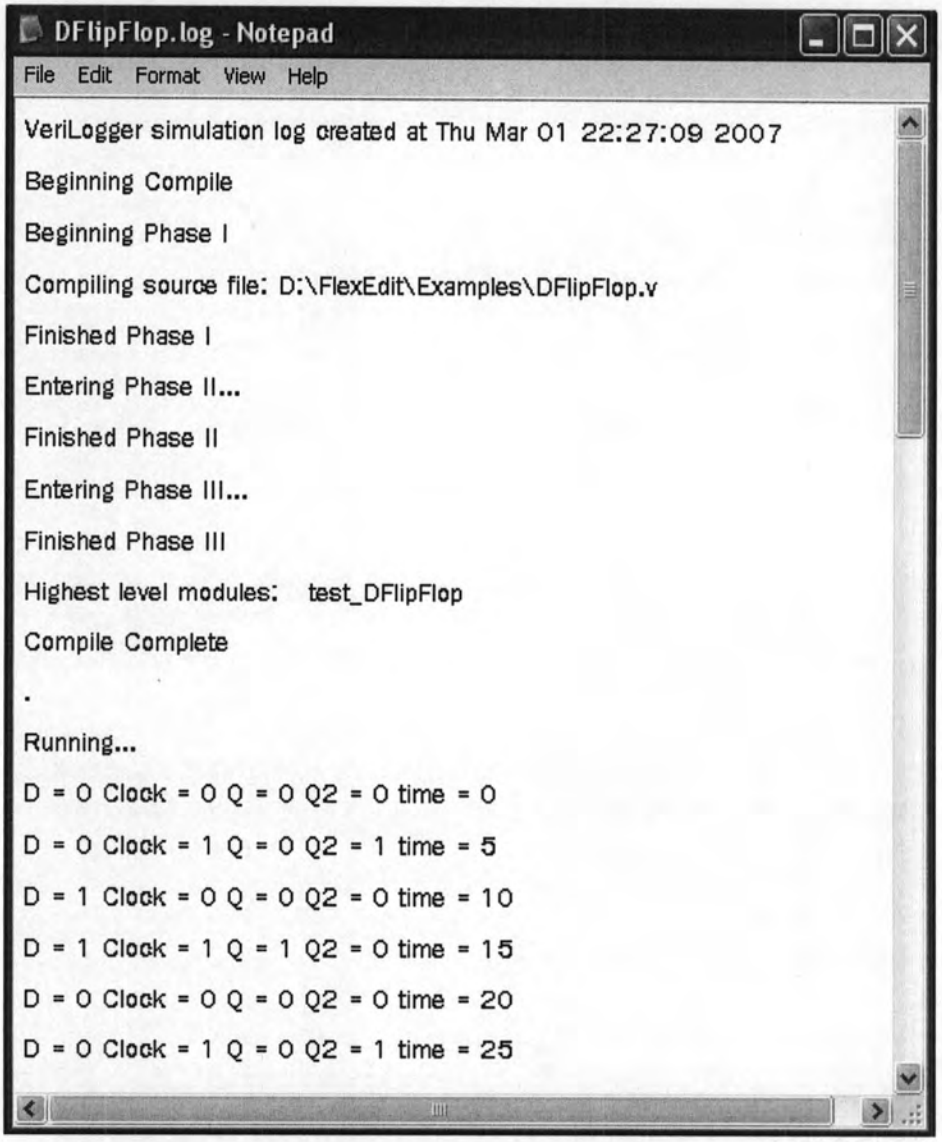
1. เลือกปุ่ม Simulator เลือกชื่อวงจรที่อยู่ในนามสกุล *.v แล้วจะแสดงผลการจำลองการทำงานในรูปแบบของไฟล์ข้อความซึ่งเป็นไฟล์ที่มีชื่อเป็นชื่อวงจรและมีนามสกุลเป็น *.log ดังแสดงในรูป ก.1

2. เรียกโปรแกรม Verilogger Pro โดยเลือก Start->Program->SynaptiCAD->Verilogger Proแล้วเลือกไฟล์ของวงจรที่ต้องการ โดยไฟล์ที่เลือกจะต้องเป็นชื่อวงจรและมีนามสกุลเป็น .v โดยทำตามลำดับดังนี้

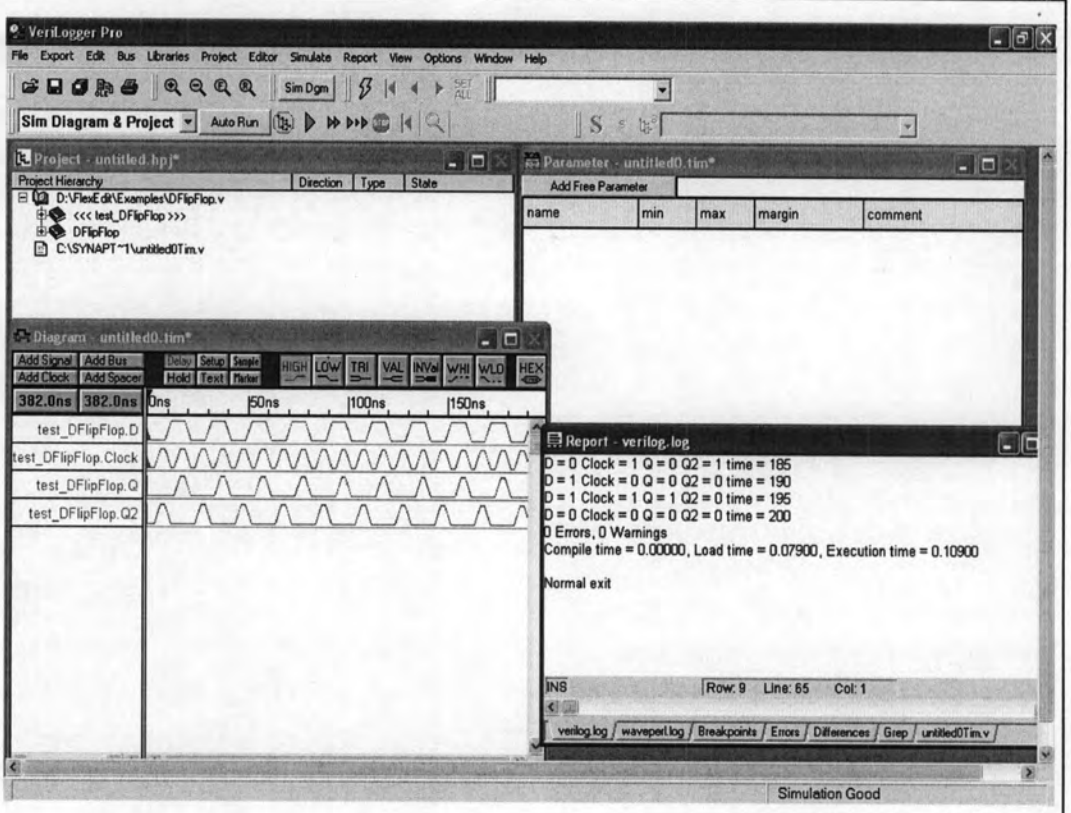
2.1 เลือก Project-> Add File(s)

2.2 กดปุ่ม  เพื่อคอมไพล์

2.3 กดปุ่ม  เพื่อรันการจำลองการทำงาน จะได้ผลการทำงานในรูปแบบกราฟิกหรือ waveform และในรูปแบบไฟล์ข้อความได้ดังรูปที่ ก.2



รูปที่ ก.11 แสดงผลการจำลองการทำงานในรูปไฟล์ข้อความ



รูปที่ ก.12 แสดงผลการจำลองการทำงานของวงจรบล็อกดีฟลิปฟลอปโดยใช้ Verilogger Pro

ภาคผนวก ข
ผลงานตีพิมพ์

งานประชุมวิชาการระบบสาระสนเทศ (Information Systems: New Generations – ISNG 2005) เมืองลาสเวกัส รัฐเนวาดา ประเทศสหรัฐอเมริกา ในบทความเรื่อง On treating Sequential Logic Design as an Aspect-Oriented Concept

On treating Sequential Logic Design as an Aspect-Oriented Concept

Petcharat Burapathana, Proadpran Pitsatorn, and Boonchai Sowanwanichkul
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University
Bangkok, 10330, Thailand
Petcharat.B@student.chula.ac.th, Proadpran.P@chula.ac.th, Boonchai@cp.eng.chula.ac.th

Abstract

Hardware Design is a complex task that demands hardware designer to redesign the circuit every time the specification is changed. Previous research offered the view of OO reuse, inheritance, and encapsulation to help hardware designers ease their jobs and embed such functionalities into the tool. To contribute to this line of work, this research proposes the extension of Aspect Oriented concept to the design of hardware, particularly the sequential logic design, hoping to increase design performance.

Keywords:
Aspect-oriented concept, Crosscutting, Sequential logic design, Hardware design, Gate, and Object-oriented concept

1. Introduction

Hardware design encompasses combinational and sequential logic design. Combinational logic design is the most basic level in hardware design process. This level uses eight basic gates such as AND, OR, NAND, etc. to represent Boolean expression. On the other hand, sequential logic design adds in feedback and clock in the design process. Aside from eight fundamental gates in combinational logic design, it combines feedback that connects to the storage elements and the clock [1]. The storage elements used in sequential circuits are flip-flops. Three types of flip-flops are D flip-flop, JK flip-flop and T flip-flop. Sequential circuit employs clock signals that manipulate all flip-flops at discrete instance of time [2].

Today hardware design process is more complex because various more functions are needed in many hardware circuits. There are many researches and tools in hardware design and those researches and tools are developed rapidly. The example is the extension to hardware description language. Although there are many researches and tools, hardware designers always face one of the major problems that is redesigning from scratch when only a few of hardware specifications are changed [3]. A more recent research offered a new method in helping hardware designer by applying software development concept to hardware design. In software design process, Object-oriented (OO) is a successful design technique. It considers everything as an object. [3] OO has three important properties; encapsulation, inheritance and reusability. Existing research proves that these three properties can be applied to hardware design [1, 3].

Aspect-oriented concept is a new technology in software development that deals with the separation of crosscutting concerns that are not available in object-oriented concept. Although object oriented concept has succeeded in modeling and implementing complex software systems, it has its problems with the maintenance of codes because it becomes increasingly difficult to cleanly separate concerns into modules. An attempt to do a minor change in the program may require several updates to a large number of unrelated modules [4].

This paper presents an idea to treat gates as aspects and proposes another view of hardware design for helping hardware designers to reduce redundancy, difficulty and promise adaptability, customizability and reusability in hardware design process.

Section 2 introduces previous works on how software design concepts are applied to hardware design. Section 3 and Section 4 introduce object-oriented and aspect-oriented concept. Section 5 presents the idea on how to apply aspect concept to sequential logic design, and the last section is a conclusion of the paper.

2. Previous work

Previous work:Chaiworawitgul et al.,2004 has been done on treating gate as an object [3], whereby encapsulation, inheritance, and reusability characteristics are embedded into the design tool. More researches :Nebel and Schumacher, 1996 and Jacome and Peicoto,2001 also applied software design strategies to hardware design [5,6]. These previous works focused on OO capabilities. Netinant et al.,2000 [7] was dedicated to the design of adaptable operating systems using aspect-oriented concept. And finally, Chaiworawitgul et al.,2003 [1] offered a design tool that encompasses OO functionalities into combination logic design.

It is obvious that there is no research, to date, that focuses particularly on sequential logic design using AOP concept. The expected outcome is to improve reusability and modularity of hardware design process.

3. Object-oriented concepts

Object-oriented programming (OOP) has become the mainstream programming paradigm where real world problems are decomposed into objects that abstract behavior and data in a single unit [4]. Objects communicate by sending and receiving messages as shown in Figure 1.

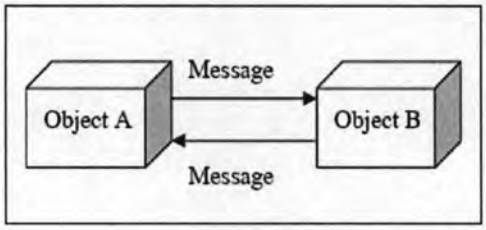


Figure 1 - Two Objects communicate by messages.

3.1. Encapsulation

In encapsulation, a combination of attribute and behavior, is used to generate a class. Attribute is an operation that uses at least one attribute in its own class [3]. Figure 2 shows a class structure in a Unified Modeling Language (UML) format.

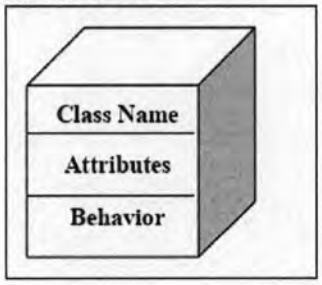


Figure 2 - Class structure in UML

A class represents a group of similar objects. Hence every object is an instance of a class.

3.2. Inheritance

Inheritance in OO requires that the behavior and data associated with child classes are always an extension of the properties associated with base classes. This property results in reducing time and effort when specification is changed or a new assimilable class is designed [3]. Figure 3 shows an example of inheritance.

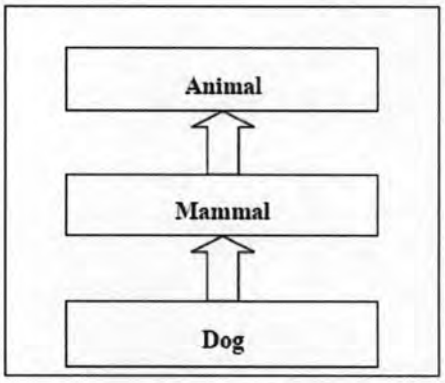


Figure 3 - An example of inheritance

3.3. Reusability

Details of any existing classes may be modified in order to create new classes. One can also integrate many existing classes to create a new class. These are examples of reusability. Classes may be reused in two ways. Firstly, classes can be reused directly without any modification. Designers may not know how a existing class was designed or implemented. They only needs to know only how this class communicates with other classes. Secondly, the existing classes may be reused indirectly. Modifications are made to the behaviors or to the data in order to generate a new class.

4. Aspect-oriented concept

OOP encourages software re-use by providing design and language constructs for reusability, encapsulation, inheritance, and polymorphism. Although OOP has met great success in modeling and implementing complex software systems, it has its problems. Programmers may face some problems with maintaining their code because it becomes increasingly difficult to cleanly separate concerns into modules. An attempt to do a minor change in the program design may require several updates to a large number of unrelated modules. Aspect-oriented concept is a new technology for separating crosscutting concerns into single units called aspects. An aspect is a modular unit of crosscutting implementation. It encapsulates behaviors that affect multiple classes into reusable modules. We start by implementing our project using our object-oriented language such as JAVA and then we deal separately with crosscutting concerns in our code by implementing aspects. Finally, both the code and aspects are combined into a final executable form using an aspect weaver. As a result, a single aspect

can contribute to the implementation of a number of methods, modules, or objects, increasing both reusability and maintainability of the code. Figure 4 explains the weaving process. The original code doesn't need to know about any functionality the aspect has added; it needs only to be recompiled without the aspect to regain the original functionality [4].

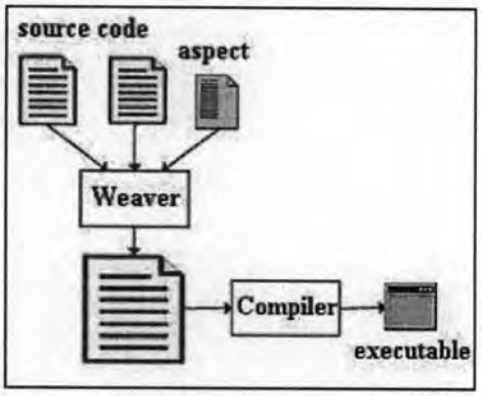


Figure 4 - Aspect Weaver

In that way, aspect-oriented concept complements object-oriented concept, not replacing it, by facilitating another type of modularity that pulls together the widespread implementation of a crosscutting concern into a single unit [4]. The example of aspect-oriented tool is AspectJ. It is an extension of the Java language for aspect-oriented. AspectJ has important terminologies: crosscutting, join point, pointcut, advice, and introduction [8].

4.1. Crosscutting

Crosscutting is how to characterize a concern that spans multiple units of object-oriented modularity. Crosscutting concerns resist modularization using normal object-oriented construct. Figure 5 presents an example of crosscutting of a simple figure editor. There are two concrete classes of figure element; points and lines. The concern is that the screen manager should be notified whenever a figure element moves. This requires every method that moves a figure element to perform the notification. The dotted line in the figure shows the DisplayUpdating that cuts across other boxes.

4.2. Join point

A join point is a well-defined point in the flow of a program. AspectJ has several kinds of join points:

- Method/constructor call join points
- Method/constructor execution join points
- Field get and set join points
- Exception handler execution join points
- Static and dynamic initialization join points

For example, a method call join point is a point in the flow when a method is called, and when that method call returns. The lifetime of the method call join point includes all the actions that comprise six method calls, starting after all arguments are evaluated up to and including normal or abrupt returns. Each method call itself is one join point.

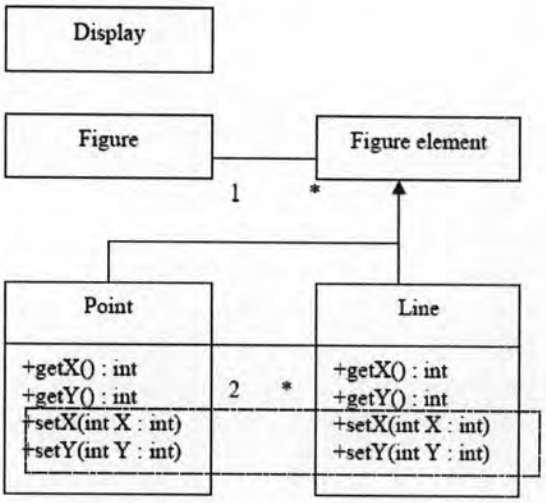


Figure 5 - An example of crosscutting

4.3. Pointcut

A pointcut selects particular join points by filtering out a subset of all join points, based on defined criteria. The criteria can be explicit function names, or function names specified by wildcards. Pointcuts can be composed using logical operators. Customized pointcuts can be defined, and pointcuts can identify join points from different classes. Examples of pointcuts are as follows:

- call(void Point.setX(int)) identifies any call to Point.setX(int).
- Call(void Point.setX(int)) || call(void Point.setX(int)) identifies any call to either Point setX or Point setY.
- call(public * Figure.*(..)) identifies any public method defined on Figure.
- Pointcut move():
  call(void Point.setX(int)) ||
  call(void Point.setY(int)) ||
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point));
  Define a pointcut called move. Notice that this pointcut crosscuts different classes.

4.4. Advice

Pointcuts are used in the definition of advice. An advice in AspectJ is used to define additional code

that should be executed at join points. AspectJ has the following advices:

4.4.1. Before advice

The code executes when a join point is reached but before the computation proceeds.

4.4.2. After advice

The code executes after the computation of under a join point has completed, but before the exit of that join point.

4.4.3. Around advice

The code executes when the join point is reached and conditions specified in the advice are satisfied.

For example, an advice after(): move() { System.out.println("moved"); } prints a message immediately after a point of a line is moved.

4.5. Introduction

An introduction in AspectJ introduces new members to classes and therefore changes inheritance relationship between classes. Unlike advices, introduction takes effect at the compilation time. Introduction is a useful construct when we want to add new concern to some existing classes, such as adding timestamp to a data structure, yet we want to have clear separation of these crosscutting concerns.

The example of aspect-oriented is shown in Figure 5. It presents an object-oriented JAVA language named TestClass.java that prints the words "Hello, AOP".

```
public class TestClass {
  public void sayHello () {
    System.out.println ("Hello, AOP");
  }

  public static void main (String[] args) {
    sayHello ();
  }
}
```

Listing 1: TestClass.java

Figure 5 - An example of object-oriented

Then we revise TestClass.java by using AspectJ in aspect-oriented named MyAspect.aj shown in Figure 6. The result of AspectJ execution is when method named "sayHello" is called, the word "TestClass", the name of join point and the word "start..." are printed.

```
public aspect MyAspect {
public pointcut sayMethodCall (): call (public void
                                TestClass.sayHello() );
before(): sayMethodCall() {
System.out.println("\n TestClass." +
 thisJoinPointStaticPart.getSignature().getName()+
"start..." );
}
}
```

Listing 2: MyAspect.aj

Figure 6 - An example of aspect-oriented modifying from object-oriented

5. Applying Aspect-oriented concept to Sequential logic design

Sequential logic is a combination of eight fundamental gates in combinational logic, feedback that connects with storage or memory elements, and clock signal. Most storage elements are called flip-flops. A flip-flop is a binary storage device storing one bit of information. The block diagram of a sequential circuit is shown in Figure 7.

The outputs can come either from the combinational circuit or from flip-flops or both. The flip-flops receive their inputs from the combinational circuit and also from clock signal with pulses as shown in Figure 8.

There are three types of flip-flops that are widely used in hardware design; D flip-flop, JK flip-flop, and T flip-flop. Each type of the flip-flop performs in different function. All flip-flops are shown in Figure 9,10 and 11 respectively.

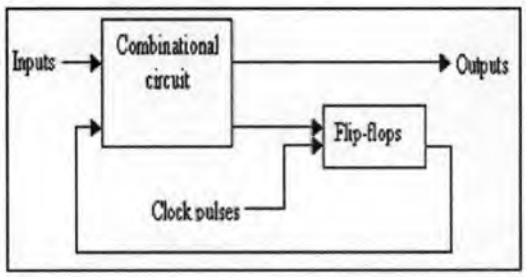


Figure 7 - Block diagram of sequential circuit

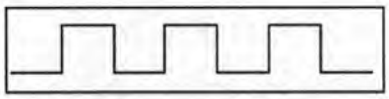


Figure 8 - Timing diagram of clock pulses

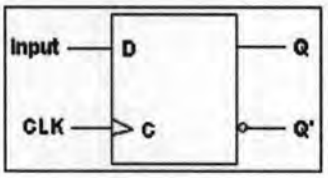


Figure 9 - D flip-flop

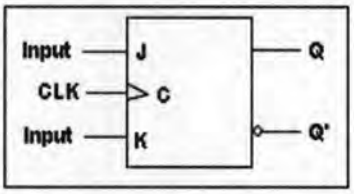


Figure 10 - JK flip-flop

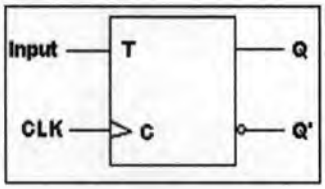


Figure 11 - T flip-flop

The existing research showed that the clock could be reused directly. Clock object is created from clock class to be used in a sequential circuit design. Flip-flop supports reusability in direct and indirect ways. In direct ways, flip-flop object is created from the existing flip-flop class, and can be used in the design of a sequential circuit. To generate a new kind of flip-flop, an existing flip-flop class is extended to a new class. Changing of attributes or behaviors can be done through the override property. The outcome of reusability in indirect way is a new class. The new class will be directly reused or indirectly reused to create a new flip-flop class. Any flip-flop class may be considered as an aspect.

The operation of flip-flop is determined by clock pulses. Therefore clock class is redundant in all sequential logic circuits. We can separate this redundancy into single module called aspect, following aspect-oriented concept. Along the same line, clock pulses may be viewed as crosscutting concerns. When hardware designers design or redesign the circuits, the weaving process will take place. Hardware designers would only have to worry about these crosscutting concerns, such that they are able to modify, change or improve any attributes or behaviors from this module. This results in a better reusability and maintainability in hardware design process whereby the conventional design lacks. The difference from reusability in OO is that if we would like to redesign sequential logic design, we always create a new class to use in redesign. However, in aspect-oriented we can define join point, pointcut and one or more of three advices to redesign clock class such as clock speed and timing.

6. Conclusion and Future works

This paper proposes a new idea for hardware design by treating flip-flops as aspects, and clock pulses as crosscuts. Implementation and experiment are yet to be carried out. And a tool that embraces this idea will be developed. Should this concept works, the process of hardware design would become easier and faster.

We hope to prove that aspect-oriented concept can be applied to hardware design process in high-level design. Finally we want to show that aspect-oriented concept may be implemented in the world of hardware.

7. References

[1] S. Chaiworawitgul, P. Pitsatorn, and B. Sowanwanichakul, "An application of Object Oriented Paradigm (OOP) to Combination Logic Dessign", *The 2003 International MultiConference in Computer Science and Engineering 2003*, 582-587.

[2] M. Norris Mano, "Digital design", Upper Daddle River, NJ: Prentice Hall, 2002.

[3] S. Chaiworawitgul, P. Pitsatorn, and B. Sowanwanichakul, "Applying Object Oriented Concept to Hardware Design", *The International Conference on Information Systems- New Generations (ISNG)*, 2004.

[4] Yasser EL-Manzalawy, "Aspect Oriented Programming", 2004. Available: http://www.developer.com/design/article.php/330894.

[5] Nebel, W. and Schumacher G., "Object-Oriented Hardware Modeling Where to apply and what are the objects?", *EURO-DAC '96 with EURO-VHDL '96*, 428-433.

[6] Jacome, M. and Peicoto, H., "A Survey of Digital Design Reuse. *IEEE Design & Test of Computer 2001*, 98-107.

[7] P. Netinant, C. A. Constantinides, T. Elrad and M. E. Fayad, "Aspect-oriented frameworks (poster session): the design of adaptable operating systems", *Proc. Object-oriented programming, systems, languages, and applications (Addendum) 2000*, 61-62.

[8] Ken Wing Kuen Lee, "An Introduction to Aspect-Oriented Programming", 2002. Available: http://www.cs.ust.hk/~scc/comp610e/assignment/reading04.pdf.

[9] Laura Ricci, "Aspect-oriented programming", 2002. Available: http://www.disi.unige.it/person/ReggioG/ISII01/AOP.ppt.

[10] Budd, T., "An Introduction to Object-Oriented Programming", 2nd ed. United States of America: Addison Wesley, 1997.

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวเพชรรัตน์ บูรพาธนะ เกิดเมื่อวันที่ 7 ธันวาคม พ.ศ.2525 ที่กรุงเทพมหานคร สำเร็จการศึกษาหลักสูตรวิศวกรรมศาสตรบัณฑิต (วศ.บ.) สาขาวิศวกรรมคอมพิวเตอร์ จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์ เมื่อปีการศึกษา 2546 และเข้าศึกษาต่อหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต (วศ.ม.) สาขาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2547 ขณะศึกษาได้มีโอกาสไปเสนอผลงานเรื่อง On treating Sequential Logic Design as an Aspect-Oriented Concept ในงานประชุมวิชาการระบบสาระสนเทศ (Information Systems: New Generations – ISNG 2005) เมืองลาสเวกัส รัฐเนวาดา ประเทศสหรัฐอเมริกา