คอมโพเนนท์คลาสเพื่องานจำลองการทำงานระบบดิจิตอลแบบมัลติแพลตฟอร์ม

Component Class for Multi-platform Digital System Simulation

ประดนเดช นีละคุปต์, กษม โคตรอาษา และสุรศักดิ์ สงวนพงษ์
ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์
บางเขน กรุงเทพฯ 10900
โทร (02) 942-8555 (EXT 1415, 1433, 1411) E-Mail: {pom, g4265077, nguan}@ ku.ac.th

บทคัดย่อ

บทความนี้กล่าวถึงการออกแบบคอมโพเนนท์คลาสแบบมัลติแพลตพอร์มที่สามารถขยายเพิ่มเติมได้ และการนำคอมโพเนนท์นี้สร้างต้นแบบของโปรแกรมจำลองการทำงานของระบบดิจิตอลในระดับเกต ที่มีความสามารถเพิ่มเติมอุปกรณ์และส่วนประกอบต่างๆ เข้าไปได้ในภายหลังในรูปแบบของ Plug-in โดยไม่ต้องมีการคอมไพล์โปรแกรมใหม่ การทำงานของโปรแกรมจะทำบนสภาพแวดล้อมแบบกราฟฟิกเพื่อผู้ใช้สามารถจำลองการทำงานของวงจรต่างๆ ได้สะดวก ผู้ใช้สามารถใช้งานโปรแกรมนี้ได้ภายในทุกสภาพแวดล้อมที่มี Java Virtual Machine [1]

Abstract

This article presents a design of component classes of digital circuit in the gate-level based on objected orientation approach. The component classes can be used to as basic elements for simulation of digital circuit behaviors or VHDL based synthesis which support plug-in capability in a multi-platform manner. To make use of this approach, we develop a digital system simulation prototype. A window graphical user interface (GUI) is provided for easily simulating digital circuit. The program can run on any platform that support Java Virtual Machine (JVM).

1. บทนำ

การทำงานพัฒนาทางด้านดิจิตอลนั้น หลังจากออกแบบแล้วจำเป็นต้องทดลองเพื่อตรวจสอบการทำงานของระบบที่ออกแบบ วิธีการหนึ่งคือการสร้างวงจรเหล่านั้นขึ้นจริง แต่วิธีนี้ต้องใช้งบประมาณสูงและสิ้นเปลืองเวลา การนำเอาคอมพิวเตอร์มาช่วยในการจำลองการทำงานของวงจรดิจิตอลก่อนจะช่วยลดงบประมาณและเวลาที่ต้องใช้ลงได้ ในปัจจุบันมีโปรแกรมสำเร็จรูปหลายโปรแกรมที่ใช้ทำหน้าที่นี้ เช่น LogicWorks[2] หรือ WORKVIEW OFFICE[3] ที่ให้ผู้ใช้จำลองการทำงานของวงจรดิจิตอลได้โดยไม่ต้องสร้างวงจรนั้นๆ ขึ้นจริง

อุปกรณ์ในงานดิจิตอลไม่ได้จำกัดเพียงเกตพื้นฐาน แต่มีวงจรรวมต่างๆจำนวนมากออกมาใหม่อยู่เสมอ โปรแกรมจำลองการทำงานควรมีความสามารถเพิ่มเติมอุปกรณ์เข้าไปได้โดยไม่ต้องคอมไพล์ใหม่ และควรทำงานได้กับระบบคอมพิวเตอร์ที่มีอยู่อย่างหลากหลายรูปแบบ

2. การจำลองการทำงานของวงจรดิจิตอล

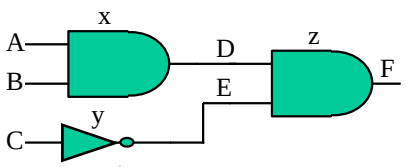
2.1 ระดับของการจำลองการทำงาน

วงจรดิจิตอลสามารถมองได้เป็นหลายระดับ เช่น ระดับบนประกอบไปด้วยแผงวงจรดิจิตอลหลายๆ แผงต่อเข้าด้วยกัน แต่ละส่วนมีหน้าที่เป็นของตัวเอง โดยเรียกระดับนี้ว่าระดับบอร์ด (board level) ในแต่ละแผงวงจรก็จะประกอบไปด้วยชิพวงจรรวมหลายๆ ตัวที่มีหน้าที่เฉพาะอย่าง เช่น เอแอลยูหรือเป็นแลทช์ แต่ละตัวก็จะประกอบไปด้วยเกตพื้นฐานและเรียกระดับนี้จะเรียกว่าระดับโมดูล (module level) ส่วนเกตต่างๆ จะอยู่ในระดับเกต (gate level) และเกตพื้นฐานแต่ละตัวถูกสร้างขึ้นมาจากการประกอบกันของทรานซิสเตอร์จึงเรียกระดับนี้ว่าระดับไฟฟ้า (electrical model) ตัวทรานซิสเตอร์เองก็อาจสร้างมาจากวัสดุต่างๆ เช่น เอ็นมอสกับพีมอส และจัดอยู่ในระดับจีโอเมตริก (geometric level) การจำลองการทำงานของวงจรดิจิตอลจึงสามารถทำได้ในทุกระดับ แต่ในที่นี้จะเน้นเฉพาะระดับเกตขึ้นไปเท่านั้น

2.2 การจำลองการตามสถานการณ์ที่เกิด (Event-Driven)

การจำลองการทำงานวงจรดิจิตอลด้วยคอมพิวเตอร์มีหลายวิธี วิธีหนึ่งที่เหมาะสมซึ่งจะนำมาใช้คือการจำลองการทำงานจากสถานการณ์ที่เกิด (Event-Driven Simulation) วิธีนี้เป็นการอาศัยการเปลี่ยนแปลงที่เกิดขึ้นของระดับลอจิกเป็นตัวกำหนดการทำงาน การเปลี่ยนแปลงทางลอจิกส่งผลให้โปรแกรมจำลองการทำงานปฏิบัติหน้าที่ซึ่งอาจเกิดทันที หรือรอเวลาระยะหนึ่งก่อนจึงจะเกิดการทำงานขึ้นจริง

จากตัวอย่างในรูปที่ 1 การเปลี่ยนแปลงของระดับสัญญาณที่จุด A หรือจุด B ซึ่งเป็นอินพุตของ x อาจทำให้เกิดการเปลี่ยนแปลงของสัญญาณที่จุด D ซึ่งเป็นเอาท์พุทของ x ซึ่งก็เป็นอินพุตของ z ด้วย การเปลี่ยนแปลงนี้อาจทำให้เกิดการเปลี่ยนแปลงของระดับสัญญาณที่จุด F ได้ในเวลาต่อมา วิธีการจำลองการทำงานแบบตามสถานการณ์ที่เกิดนี้จะอาศัยการเปลี่ยนแปลงนี้ในการเลือกว่าจะต้องมีการจำลองการทำงานที่จุดใดของวงจร เนื่องจากวิธีนี้จะทำงานกับเฉพาะส่วนที่เกิดการเปลี่ยนแปลงขึ้นของวงจรเท่านั้น วิธีนี้จึงเป็นวิธีที่มีประสิทธิภาพมาก

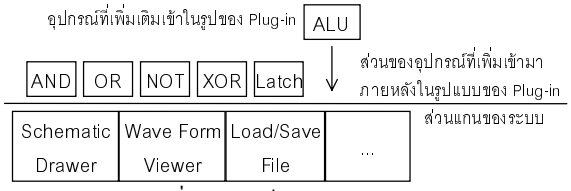


รูปที่ 1 ตัวอย่างวงจรดิจิตอล

3. ระบบ Plug-in

โปรแกรมจำลองการทำงานของวงจรดิจิตอลจำเป็นต้องมีคลังของอุปกรณ์ที่จะนำมาต่อเป็นวงจร อุปกรณ์ต่างๆ เหล่านี้จะถูกสร้างและพัฒนาใหม่ออกมาตลอดเวลา ทำให้ไม่สามารถเตรียมอุปกรณ์เหล่านี้ทั้งหมดเป็นส่วนหนึ่งของโปรแกรมได้ แต่สามารถนำวิธีการสร้าง Plug-in มาใช้เพื่อเพิ่มเติมหรือเปลี่ยนแปลงอุปกรณ์ต่างๆ ได้ดังลักษณะในรูปที่ 2 ก่อนจะทำเช่นนี้ได้เราต้องออกแบบและจัดเตรียมส่วนเชื่อมต่อระหว่างส่วนของโปรแกรมหลักหรือส่วนแกนของระบบ กับส่วนของ Plug-in ที่จะสร้างขึ้นมาเพิ่มเติมต่อไป เพื่อที่โปรแกรมหลักและส่วนที่นำมาต่อเพิ่มจะได้สามารถทำงานร่วมกันได้

การสร้างระบบ Plug-in ขึ้นกับระบบที่ใช้ เช่นในเครื่องที่ทำงานด้วยระบบปฏิบัติการวินโดวส์อาจใช้ Dynamic Link Library (DLL) เป็นตัวเก็บส่วนของ Plug-in หรือในระบบปฏิบัติการยูนิกซ์อาจสร้างระบบ Plug-in ได้โดยใช้ Shared Library ส่วนในภาษาจาวานั้นสามารถสร้าง Plug-in ได้โดยใช้คลาสธรรมดา เนื่องจากสามารถนำคลาสที่ไม่รู้จักขณะคอมไพล์ หรือคลาสที่สร้างขึ้นมาในภายหลังเข้ามาทำงานกับโปรแกรมได้โดยตรง โดยไม่ต้องคอมไพล์โปรแกรมหลักใหม่



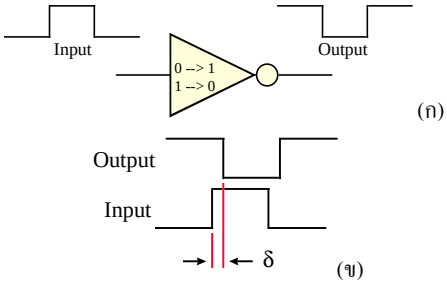
รูปที่ 2 อุปกรณ์ในรูปของ Plug-in

4. การใช้ OOP กับโปรแกรมจำลองการทำงานวงจรดิจิตอล

วงจรดิจิตอลจริงๆ ประกอบไปด้วยอุปกรณ์หลายๆ ตัวนำมาต่อเชื่อมเข้าด้วยกัน จึงจำเป็นต้องเก็บการเชื่อมโยงกันของอุปกรณ์ต่างๆ เอาไว้เพื่อที่จะสามารถส่งการเปลี่ยนแปลงของระดับสัญญาณไปให้อุปกรณ์ที่เกี่ยวข้องกับสัญญาณนั้นได้ โดยทั่วไปแล้วเอาท์พุตของอุปกรณ์หนึ่งก็จะเป็นอินพุตของอุปกรณ์อื่นๆ และอินพุตของอุปกรณ์นั้นๆ ก็มักจะต่อกับเอาท์พุตของอุปกรณ์อื่นเช่นกัน สิ่งจำเป็นคือต้องมีวิธีการที่เหมาะสมสำหรับเก็บการเชื่อมต่อกันของอุปกรณ์ต่างๆ เอาไว้ เพื่อที่จะสามารถส่งต่อสัญญาณระหว่างอุปกรณ์ได้ถูกต้องและมีประสิทธิภาพ

แนวคิดการเขียนโปรแกรมเชิงวัตถุ (OOP) จะมองโปรแกรมเป็นรูปแบบของวัตถุที่มีพฤติกรรมเฉพาะของตัวเอง และมีความสัมพันธ์กับวัตถุอื่น การมองปัญหาในลักษณะนี้นำมาใช้กับปัญหาการจำลองวงจรดิจิตอลได้เป็นอย่างดี กล่าวคือถ้ามองชิ้นส่วนอุปกรณ์ดิจิตอลแต่ละชิ้นเป็นอ็อบเจ็ค อุปกรณ์แต่ละชนิดก็จะมีคุณสมบัติเป็นของตัวเอง การนำอุปกรณ์มาต่อเชื่อมกันจะเหมือนกับการประกอบอ็อบเจ็คเข้าด้วยกัน

หากพิจารณาการจำลองการทำงานของอุปกรณ์หนึ่งชิ้นซึ่งโดยทั่วไปจะมีทั้งอินพุตและเอาท์พุตที่รับสัญญาณดิจิตอลเข้าและส่งออกจากอุปกรณ์นั้นๆ เมื่อเกิดการเปลี่ยนแปลงของสัญญาณหรือข้อมูลเข้าที่อินพุตอาจทำให้เกิดการเปลี่ยนแปลงที่เอาท์พุตในช่วงเวลาถัดไป เช่นในรูปที่ 3 สัญญาณเข้าที่อินพุตของเกตที่เป็นอินเวอร์เตอร์ตัวหนึ่งเปลี่ยนจาก 0 เป็น 1 จะทำให้เกิดการเปลี่ยนแปลงที่เอาท์พุตของเกตตัวนั้นจาก 1 เป็น 0 ที่เวลาห่างออกไปเท่ากับ δ ซึ่งเป็นเวลาหน่วงของเกตตัวนั้น ก่อนที่จะจำลองเหตุการณ์นี้ได้จะต้องทราบคุณสมบัติของเกตว่าสัญญาณเอาท์พุตจะตรงกันข้ามกับสัญญาณที่อินพุต และการเปลี่ยนแปลงระดับสัญญาณจะใช้ช่วงเวลาหนึ่ง ก่อนที่สัญญาณนั้นจะปรากฏที่เอาท์พุต



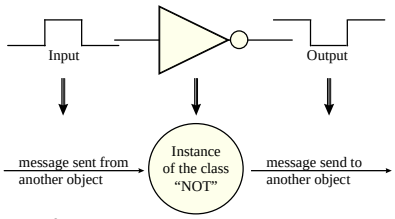
รูปที่ 3 (ก) ลักษณะสัญญาณที่ผ่านเกตอินเวอเตอร์
(ข) เวลาหน่วงที่เกิดขึ้นจากเกต

อุปกรณ์ดิจิตอลแต่ละชนิดมีพฤติกรรมที่ตอบสนองต่ออินพุตแตกต่างกัน การออกแบบโดยใช้ OOP ทำให้ไม่จำเป็นต้องสนใจถึงพฤติกรรมที่เฉพาะตัวของอุปกรณ์แต่ละชนิด และไม่ต้องสนใจรายละเอียดของการสร้างโปรแกรมส่วนนั้นว่าจะเป็นแบบใด หรือจะต้องใช้วิธีการใดเพื่อให้อุปกรณ์มีคุณสมบัติตามที่ต้องการ แต่สิ่งแรกที่สนใจคือการที่อุปกรณ์เหล่านั้นประกอบสิ่งใดที่เหมือนกันบ้าง เช่น เราไม่สนใจว่าเกต NAND และเกต AND เมื่อป้อนค่าอินพุตที่เหมือนกันจะให้เอาท์พุตต่างกันหรือไม่อย่างไร หรือ เราจะไม่สนใจว่าเราจะสร้างคลาสของเกต NAND ด้วยการเก็บตารางค่าความจริง หรือจะใช้การเขียนโปรแกรมย่อยเฉพาะสำหรับเกตนั้น แต่เราจะสนใจที่ทั้งเกต NAND และเกต AND หรืออุปกรณ์ใดๆ จะมีช่องสำหรับต่อกับอุปกรณ์อื่นๆ เหมือนๆ กัน ซึ่งช่องนั้นอาจใช้เพื่อรับสัญญาณเข้า หรือส่งสัญญาณออก หรือทั้งสองอย่าง ถ้าอุปกรณ์นั้นมีช่องที่ใช้รับสัญญาณเข้า ถ้าสัญญาณที่เข้าเกิดเปลี่ยนแปลงก็อาจทำให้สัญญาณออกเปลี่ยนแปลงได้ การเปลี่ยนแปลงจะเกิดขึ้นตามแต่เวลาหน่วงของอุปกรณ์ สิ่งที่อุปกรณ์ต่างๆ มีเหมือนๆ กันเหล่านี้จะถูกนำมาเป็นซูเปอร์คลาสของอุปกรณ์ทางดิจิตอลทั้งหมด

อุปกรณ์ดิจิตอลที่ถูกแทนด้วยอ็อบเจ็คจะรับและส่งสัญญาณผ่านการส่งแมสเสจ เมื่ออ็อบเจ็คที่แทนอุปกรณ์ดิจิตอลตัวได้ได้รับแมสเสจ ก็เปรียบเสมือนมีสัญญาณเข้าที่อุปกรณ์ตัวนั้น ซึ่งอุปกรณ์ตัวนั้นก็จะทำงานตามหน้าที่ ถ้าหากเกิดการเปลี่ยนแปลงสัญญาณที่เอาท์พุตก็จะส่งการเปลี่ยนแปลงนั้นออกไปโดยส่งแมสเสจไปให้อ็อบเจ็คอื่นต่อไป

เนื่องจากเราต้องคำนึงถึงช่วงเวลาที่จะปล่อยแมสเสจออกไปด้วย เราจึงต้องอาศัยอ็อบเจ็คอื่นที่ทำหน้าที่ควบคุมจังหวะเวลาการทำงาน

ของแต่ละอ็อบเจ็คที่ทำงานร่วมกันให้ทำงานในจังหวะที่ถูกต้อง รูปที่ 4 เป็นตัวอย่างการแทนอุปกรณ์ด้วยอ็อบเจ็ค



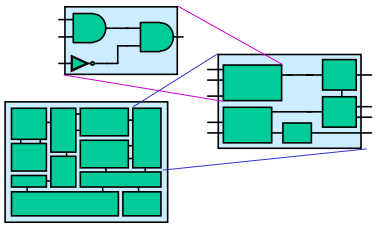
รูปที่ 4 การใช้อ็อบเจ็คแทนอุปกรณ์ดิจิตอล

การใช้คุณสมบัติโพลีมอร์ฟีสช่วยให้ไม่ต้องตรวจสอบว่าอ็อบเจ็คที่เป็นตัวรับแมสเสจนั้นเป็นอ็อบเจ็คของอุปกรณ์ชนิดใด เพียงแต่อ็อบเจ็คนั้นจะต้องเป็นอ็อบเจ็คที่สืบทอดมาจากคลาสของอุปกรณ์ดิจิตอลเท่านั้น ซึ่งการทำงานของอ็อบเจ็คก็จะเป็นไปตามที่ได้กำหนดในคลาสของอ็อบเจ็คนั้น

ในที่นี้ได้ออกแบบให้อุปกรณ์ทุกตัวสืบทอดมาจากคลาส `LComponent` ซึ่งเป็นคลาสนามธรรมที่เป็นตัวแทนของอุปกรณ์ดิจิตอล ตัวคลาส `LComponent` จะกำหนดการทำงานพื้นฐานต่างๆ ที่อุปกรณ์ทุกตัวต้องมี เช่น กำหนดการขอรายละเอียดเกี่ยวกับขาของอุปกรณ์ว่ามีจำนวนเท่าใด แต่ละขามีชื่อว่าอะไร ตัวอุปกรณ์เองมีชื่อใด รวมถึงการที่อุปกรณ์นั้นๆ จะต้องสามารถรายงานได้ว่าการเปลี่ยนแปลงระดับของสัญญาณของอุปกรณ์นั้นจะเกิดขึ้นอีกครั้งเมื่อใด

สำหรับอุปกรณ์แต่ละตัว ขาต่างๆ ของอุปกรณ์จะถูกแทนด้วยคลาส `Pin` ซึ่งเป็นส่วนที่ใช้ติดต่อกับภายนอก การเชื่อมต่อระหว่าง `Pin` ของอุปกรณ์จะใช้คลาส `Wire` เป็นตัวจัดเก็บการเชื่อมต่อระหว่างขา และทำหน้าที่ส่งการเปลี่ยนแปลงของสัญญาณให้กับอุปกรณ์ที่ต่ออยู่ด้วย

วงจรดิจิตอลที่ซับซ้อนจะประกอบไปด้วยวงจรที่เล็กกว่าหลายๆ วงจรประกอบกันดังเช่นในรูปที่ 5 เราออกแบบคลาส `ComponentContainer` ซึ่งสืบทอดมาจากคลาส `LComponent` เป็นตัวเก็บวงจรย่อยเหล่านั้นไว้ภายใน โดยภายนอกจะมองเห็นอ็อบเจ็คของคลาสนี้เป็นเสมือนอ็อบเจ็คของอุปกรณ์อื่นๆ ซึ่งอ็อบเจ็คของคลาสนี้สามารถประกอบกับอ็อบเจ็คอื่น เพื่อสร้างวงจรที่ซับซ้อนขึ้นไปอีกได้



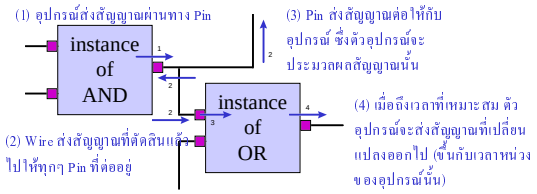
รูปที่ 5 วงจรย่อยๆ ประกอบรวมเป็นส่วนประกอบของวงจรที่ใหญ่ขึ้น

5. หน้าที่และการทำงานของอ็อบเจ็คต่างๆ

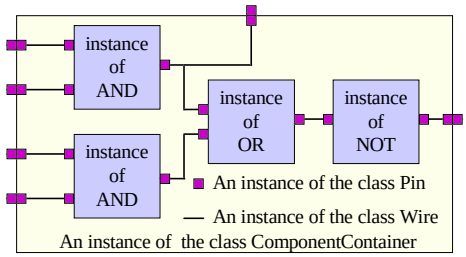
อ็อบเจ็คของคลาสต่างๆ เช่นอ็อบเจ็คของคลาส `Pin` คลาส `Wire` สืบคลาสของคลาส `LComponent` (ซึ่งอาจหมายถึงคลาสที่แทนตัวอุปกรณ์จริงๆ) และคลาส `ComponentContainer` จะถูกนำมาประกอบรวมกันเข้าเป็นวงจร การประกอบรวมกันของอ็อบเจ็คต่างๆ เหล่านี้จะถูกทำผ่านส่วนติดต่อกับผู้ใช้ โดยให้ผู้ใช้นำส่วนประกอบต่างๆ มาประกอบกันเข้าเป็นวงจรตามที่ต้องการ เพื่อที่จะนำไปจำลองการทำงานต่อไป โดยคลาสที่มีความสำคัญต่อการจำลองการทำงานในโครงงานนี้ได้แก่ คลาส `LComponent` คลาส `Pin` คลาส Wire คลาส `ComponentContainer` และสืบคลาสของคลาส `LComponent`

ขั้นตอนการจำลองการทำงานโดยการส่งแมสเสจถึงกันของอ็อบเจ็คต่างๆ ที่ประกอบกันเข้าเป็นวงจรแบบคร่าวๆ แสดงได้ดังรูปที่ 6

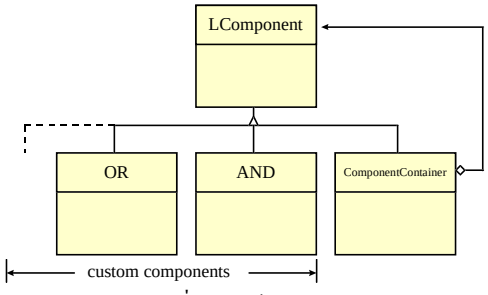
คลาส `ComponentContainer` เป็นคลาสที่แทนวงจรด้วยอุปกรณ์ตัวหนึ่ง วงจรจริงๆ จะถูกเก็บไว้ภายในอ็อบเจ็คของคลาสนี้ โดยตัวคลาสนี้จะทำหน้าที่เป็นเสมือนอุปกรณ์ตัวหนึ่งที่มีคุณสมบัติเช่นเดียวกับวงจรที่เก็บไว้ภายในดังในรูปที่ 7 คลาสไดอะแกรมจะเป็นดังรูปที่ 8



รูปที่ 6 การส่งแมสเสจระหว่างอุปกรณ์



รูปที่ 7 การแทนวงจรดิจิตอลด้วยความสัมพันธ์ระหว่างอ็อบเจ็ค



รูปที่ 8 คลาสไดอะแกรม

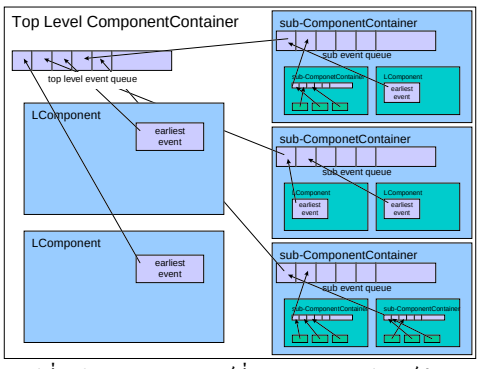
ในโครงงานนี้มีอุปกรณ์พื้นฐานบางส่วนที่ได้สร้างไว้แล้ว โดยอุปกรณ์เหล่านี้ทุกตัวสืบทอดมาจากคลาส `LComponent` (อาจสืบทอดตรง หรือสืบทอดผ่านคลาสอื่น) เช่น ตัวป้อนสัญญาณ, เกตพื้นฐาน, ฟลิป-ฟล็อป, เคาน์เตอร์, หน่วยความจำ และพีแอลดีเป็นต้น

6. คิวลำดับเหตุการณ์ (Event Queue)

อุปกรณ์แต่ละตัวมีช่วงเวลาที่จะเกิดการเปลี่ยนแปลงในครั้งต่อไปต่างกัน การจำลองการทำงานต้องไม่ทำให้ลำดับของการเกิดการเปลี่ยนแปลงนั้นผิดไป วิธีการหนึ่งก็คือการนำช่วงเวลาที่จะเกิดการเปลี่ยนแปลงของอุปกรณ์ทั้งหมดเข้ามารวมกัน แล้วจัดเรียงตามลำดับของช่วงเวลาที่จะเกิดเหตุการณ์จากก่อนไปหลัง เหตุการณ์ที่ถูกจัดไว้ก่อนจะเป็นเหตุการณ์ที่เกิดก่อน และเหตุการณ์ที่อยู่หน้าสุดก็คือเหตุการณ์ที่จะเกิดขึ้นในการจำลองการทำงานเป็นอันดับแรก

อ็อบเจ็คของคลาส ComponentContainer ซึ่งทำตัวเป็นเสมือนอุปกรณ์ธรรมดาตัวหนึ่งในวงจร เป็นคลาสที่เก็บอุปกรณ์และการเชื่อมโยงต่างๆ ไว้ การเก็บเหตุการณ์ในคลาสนี้จึงมีลักษณะภายนอกเหมือนอุปกรณ์อื่นๆ คือจะแสดงเฉพาะเหตุการณ์ที่ใกล้ที่สุดที่จะเกิดขึ้นเท่านั้น ตัวอ็อบเจ็คของคลาสนี้ในชั้นนอกจะเรียกขอช่วงเวลาการเกิดเหตุการณ์ของอุปกรณ์แต่ละตัวภายในทุกตัวมาจัดเรียง แล้วนำเวลาที่จะเกิดเหตุการณ์ที่จะเกิดขึ้นก่อนมาเป็นเวลาที่จะเกิดเหตุการณ์ของตัวเอง การทำเช่นนี้อาศัยการจัดเรียงลำดับเหตุการณ์ที่จะเกิดขึ้นก่อนหลัง ซึ่งจำเป็นต้องมีการจัดลำดับใหม่ภายหลังจากที่อุปกรณ์ตัวใดตัวหนึ่งตอบสนองต่อเหตุการณ์หนึ่งๆไปแล้ว หรือการตอบสนองต่อเหตุการณ์ที่เกิดขึ้นใหม่ทำให้ช่วงเวลาการเกิดเหตุการณ์เปลี่ยนแปลงไป

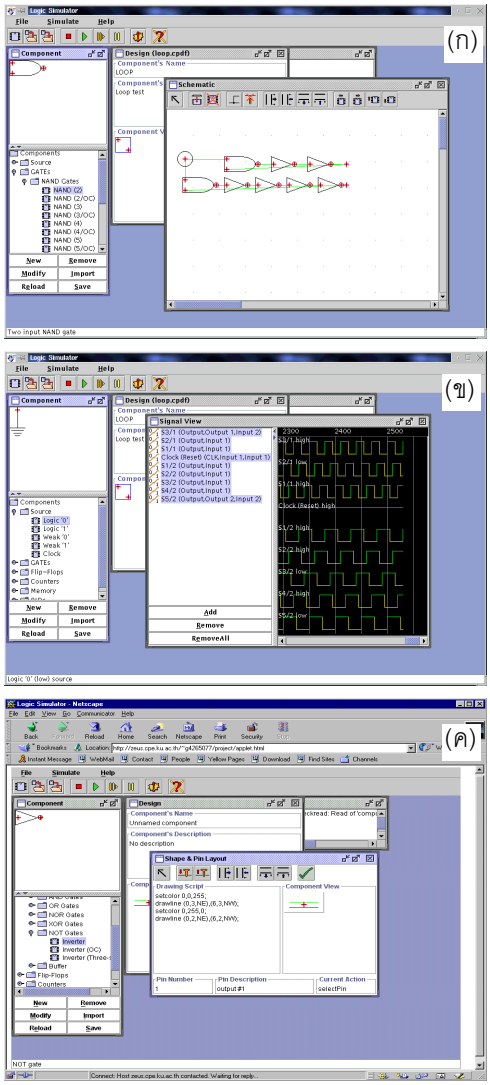
แทนที่จะให้อ็อบเจ็คของคลาส `ComponentContainer` เป็นผู้ถามอุปกรณ์ภายในทุกๆ ตัวว่าตัวใดจะเกิดเหตุการณ์เมื่อใด จะให้อุปกรณ์ทุกๆ ตัวที่เกิดการเปลี่ยนแปลงเวลาการเกิดเหตุการณ์เป็นผู้รายงานความเปลี่ยนแปลงนี้ไปยังอุปกรณ์ที่บรรจุอุปกรณ์ตัวนั้นๆ ไว้แทน ซึ่งส่งผลให้การจัดเรียงลำดับเหตุการณ์จะเกิดขึ้นเมื่อจำเป็น และเฉพาะอุปกรณ์ที่จำเป็นเท่านั้นดังในรูปที่ 9



รูปที่ 9 คิวลำดับเหตุการณ์ที่กระจายตามอุปกรณ์ต่างๆ

7. บทสรุป

จากการออกแบบคอมโพเนนท์คลาสทำให้เราได้คอมโพเนนท์คลาสที่เป็นรูปแบบพื้นฐานของอุปกรณ์ดิจิตอลที่สามารถนำมาสืบทอดหรือขยายเป็นอุปกรณ์ทางดิจิตอลใดๆ ได้ และได้ต้นแบบของโปรแกรมจำลองการทำงานของระบบดิจิตอลที่สามารถใช้งานได้ในหลายๆ ระบบ ทั้งในรูปของจาวาแอพเพล็ทและจาวาแอพลิเคชันดังในรูปที่ 10 ที่สามารถพัฒนาและเพิ่มเติมอุปกรณ์ต่างๆ ได้ในรูปแบบของ plug-in การใช้ OOP ในการออกแบบ ทำให้การพัฒนาเพิ่มเติมเป็นไปได้ง่าย



รูปที่ 10 (ก) การสร้างวงจร (ข) ผลลัพท์ของการจำลอง (ค) การทำงานเป็นแอพเพล็ต

เอกสารอ้างอิง

[1] *Java(TM) Technology Home Page*, http://java.sun.com/, Sun Microsystem, Inc.

[2] *LogicWorks Home Page*, http://www.capilano.com/LogicWorks/, Capilano Computing Systems Ltd.

[3] *VIEWLOGIC HOME*, http://www.viewlogic.com/, Viewlogic Systems, Inc.

[4] Alan W. Shaw, *Logic Circuit Design*, Saunders Collage Publishing, USA, 1991.

[5] Alexander Miczo, *Digital Logic Testing and Simulation*, John Wiley & Sons, New York, Singapore, 1987.

[6] Bruce Eckel, *Thinking in Java*, Prentice Hall Inc, New Jersey, 1998.

[7] David Flanagan, *Java in a Nutshell*, O'Reilly & Associates, Inc, USA, 1996.

[8] D. Lewin and D. Protheroe, *Design of Logic Systems*, Chapman & Hall, UK, 1992.