บทที่ 4

ภาษาวีเอชดีแอล

4.1 บทนำ

จากทฤษฎีการประมวลผลสัญญาณเชิงเลขที่ได้ทำการศึกษาไปแล้วในบทที่ 3 จะพบว่าการสร้างระบบการประมวลผลสัญญาณเชิงเลขสามารถทำได้สองวิธีคือ วิธีแรกสร้างจากซอฟท์แวร์ วิธีนี้ใช้การเขียนโปรแกรมเพื่อทำการประมวลผลเพื่อให้ได้ผลตามอัลกอริทึม (Algorithm) ที่ต้องการ สำหรับตัวประมวลผลอาจจะใช้ตัวประมวลผลที่สร้างขึ้นมาโดยเฉพาะสำหรับการประมวลผลสัญญาณเชิงเลข เช่น Blackfin Processor, TigerSHARC Processor ของบริษัท Analog Device ฯลฯ หรืออาจจะใช้ตัวประมวลผลที่ใช้งานทั่วไป เช่น ตัวประมวลผลตระกูล X86 ที่ใช้ในเครื่องคอมพิวเตอร์ส่วนบุคคลหรืออาจจะใช้ตัวประมวลผลที่ออกแบบมาสำหรับระบบฝังตัว (Embedded System) เช่น PowerPC, XScale, ARM เป็นต้น วิธีที่สองสร้างจากฮาร์ดแวร์ วิธีที่สองนี้จะแตกต่างจากวิธีแรกคือ ใช้การแปลงอัลกอริทึมไปเป็นวงจรเชิงเลขเพื่อทำการประมวลผล ในอดีตการออกแบบจะใช้การต่อวงจรด้วยไอซีลอจิกเกทมาตรฐาน แต่ปัจจุบันอุปกรณ์ประเภทพีแอลดี (Programmable Logic Device: PLD) เช่น ซีพีแอลดี (Complex Programmable Logic Device: CPLD) หรือ เอฟพีจีเอ (Field Programmable Gate Array: FPGA) ได้ถูกพัฒนาให้มีขีดความสามารถสูงขึ้น เช่น จำนวนลอจิกเกทภายในที่มากขึ้น, ค่าการหน่วงเวลาของลอจิกเกท (Propagation Delay Time: t_{pd}) ที่น้อยลงในขณะที่ราคาต่อจำนวนของลอจิกเกทที่ลดลงอย่างมากรวมไปถึงมันมีความสามารถในการโปรแกรมใหม่ (Re-Program) เมื่อการออกแบบมีการปรับปรุงแก้ไข ดังนั้นการออกแบบวงจรเชิงเลขในปัจจุบันจึงเปลี่ยนไปใช้อุปกรณ์ประเภทพีแอลดีเสียเป็นส่วนใหญ่รวมถึงงานวิจัยที่กำลังนำเสนอนี้ด้วย

การออกแบบวงจรอิเล็กทรอนิกส์ในปัจจุบันไม่ว่าจะเป็นวงจรเชิงอุปมานหรือวงจรเชิงเลขได้มีการนำเครื่องคอมพิวเตอร์มาช่วยในการออกแบบที่เรียกว่า Computer Aided Design (CAD) หรือ Electronic Design Automation (EDA) เพื่อลดต้นทุนของการทำเครื่องต้นแบบและลดเวลาของการพัฒนาผลิตภัณฑ์เพื่อออกวางตลาด สำหรับซอร์ฟแวร์ที่ถูกพัฒนาขึ้นเพื่อช่วยในการออกแบบถูกแบ่งออกเป็น 2 กลุ่มตามรูปแบบของการติดต่อกับผู้ใช้ (User Interface: UI) ได้แก่ แบบแรกโหมดรูปภาพ (Graphic Mode) เช่น การออกแบบวงจรเชิงเลขด้วยการเขียนวงจร (Circuit Diagram) และ แบบที่สอง

โหมดตัวหนังสือ (Text Mode) เช่น การออกแบบวงจรเชิงเลขด้วยการเขียน ภาษาบรรยายพฤติกรรมของฮาร์ดแวร์ (Hardware Description Language: HDL) ซึ่งปัจจุบันมีหลายภาษาด้วยกัน เช่น เวรีล็อก (Verilog), วีเอชดีแอล (Very high speed integrated circuit Hardware Description Language: VHDL) สำหรับงานวิจัยที่กำลังนำเสนอได้เลือกใช้ภาษาวีเอชดีแอลในการออกแบบเนื่องจากมีข้อดีหลายประการซึ่งได้กล่าวไปแล้วในบทที่ 1 การออกแบบวงจรเชิงเลขโดยใช้ภาษาวีเอชดีแอลรวมไปถึงภาษาบรรยายพฤติกรรมของฮาร์ดแวร์อื่นๆ สามารถรองรับหลักการออกแบบที่เรียกว่า “การออกแบบจากบนลงล่าง” (Top Down Design) ซึ่งจะกล่าวถึงรายละเอียดในหัวข้อต่อไป

4.2 การออกแบบจากบนลงล่าง (Top Down Design)

แนวความคิดของการออกแบบจากบนลงล่างคือ การออกแบบระบบที่เริ่มจากการเขียนรูปแบบ (Model) จากแนวความคิดโดยสังเขปซึ่งมีรายละเอียดทางไฟฟ้าหรือรายละเอียดเกี่ยวกับการผลิตน้อยมาก จากนั้นจะผ่านกระบวนการหลายขั้นตอนที่ให้ข้อมูลทางไฟฟ้ามากขึ้นเช่น เป็นวงจรหรือลาย (Layout) ที่ใช้ในการผลิตไอซี (IC Fabrication) ไปจนถึงขั้นสุดท้ายคือการประกอบอุปกรณ์ลงแผ่นวงจรพิมพ์ (Printed Circuit Board: PCB) รวมกับอุปกรณ์ชนิดอื่นเพื่อสร้างเป็นผลิตภัณฑ์ การออกแบบจากบนลงล่างสามารถประยุกต์ใช้ได้กับหลายเทคโนโลยีของการผลิตแต่ละขั้นตอนที่กำลังจะกล่าวถึงเป็นวิธีการกลางซึ่งในรายละเอียดอาจจะมีความแตกต่างกันเล็กน้อยทั้งนี้เนื่องจากเทคโนโลยีเป้าหมายที่แตกต่างกันดังกล่าว ขั้นตอนของการออกแบบจากบนลงล่างมีรายละเอียดดังนี้

- System Specification and Analysis เริ่มการออกแบบด้วยการสร้างข้อกำหนดของความต้องการ (Specification) และวิเคราะห์ระบบเพื่อหาแนวความคิดและหลักการในการแก้ปัญหา
- Modeling and Simulation เขียนรูปแบบ (Model) ของระบบที่ต้องการซึ่งได้จากขั้นตอนแรกโดยใช้ภาษาบรรยายพฤติกรรมของฮาร์ดแวร์และจำลองการทำงานเปรียบเทียบผลที่ได้กับคุณสมบัติที่ต้องการ
- Logic and Test Synthesis รูปแบบ (Model) ที่ทำงานถูกต้องจะถูกสังเคราะห์ (Synthesize) ให้อยู่ในรูปของวงจรเชิงเลข (Gate Level) ที่ประกอบด้วยเกทชนิดต่างๆเชื่อมต่อกันหรืออาจจะอยู่ในรูปของเน็ทลีสท์ (Net list) ในขั้นตอนนี้กับการผลิตของบางเทคโนโลยีเช่น Gate Array, Standard Cell หรือ Full Custom IC อาจมีความจำเป็นที่จะต้องสร้างโครงสร้างของวงจรใหม่หลังจากที่ได้สังเคราะห์ไปแล้วในครั้งแรกเช่น อาจจะสร้างวงจรเพื่อทำการทดสอบตัวเอง (Build-In Self Test:

BIST) รวมถึงข้อมูลในการทดสอบ (Test Pattern) หลังการผลิตเป็นเครื่องต้นแบบและเรียกการออกแบบด้วยวิธีการนี้ว่า “Design for Test” (DFT)

- Pre-Timing Verification หลังจากได้ทำการสังเคราะห์ระบบให้อยู่ในระดับของเกทหรือเน็ทลีสท์แล้วในขั้นตอนนี้เป็นขั้นตอนของการจำลองการทำงานซึ่งต้องใช้ข้อมูลของเทคโนโลยีเป้าหมายที่ต้องการใช้เช่น เทคโนโลยีเอฟพีจีเอหรือเทคโนโลยีซีพีแอลดีของบริษัท Xilinx, Altera, Actel ฯลฯ ซึ่งประกอบด้วยข้อมูลของการทำงานในระดับฟังก์ชั่นและข้อมูลทางเวลาอันเนื่องมาจากความจริงที่ว่าอุปกรณ์อิเล็กทรอนิกส์ทุกชนิดจะมี Propagation Delay เสมอถึงแม้ว่ามันจะมีค่าน้อยมากคืออยู่ในระดับนาโนวินาที (10^{-9} Second) แต่ถ้าระบบที่ออกแบบเป็นระบบใหญ่ประกอบด้วยเกทจำนวนมากตั้งแต่ 10,000 จนถึงระดับล้านเกท ค่าการหน่วงเวลาสะสมอาจจะทำให้ระบบทำงานผิดฟังก์ชั่นที่ต้องการหรือไม่สามารถทำงานกับความถี่สัญญาณนาฬิกา (Clock) ที่ต้องการได้
- Physical Design and Analysis เป็นขั้นตอนของการผลิตเป็นวงจรจริงเช่น การผลิตเป็นเอสิก, การโปรแกรมข้อมูลของการออกแบบลงเอฟพีจีเอ, ซีพีแอลดี หรือการประกอบอุปกรณ์ลงแผ่นวงจรพิมพ์
- Post-Timing Verification หลังจากได้วงจรจริงจากขั้นตอนที่ผ่านมาขั้นตอนนี้เป็นขั้นตอนทดสอบการทำงานที่คำนึงถึงเวลาและความถูกต้องของฟังก์ชั่นที่ได้จากวงจรจริง เนื่องจากในขั้นตอนนี้จะได้วงจรที่ประกอบด้วย 2 จุดใหญ่ๆจะต้องทำการทดสอบคือ สัญญาณขาเข้าและสัญญาณขาออกซึ่งเป็นจุดที่จะต้องนำไปเชื่อมต่อกับส่วนอื่นๆเพื่อสร้างเป็นระบบหรือผลิตภัณฑ์ที่พร้อมใช้งาน
- System Level Verification ระบบที่กำลังออกแบบเพื่อสร้างเป็นผลิตภัณฑ์อาจจะประกอบด้วยวงจรหลายๆวงจรและอุปกรณ์ชนิดอื่นๆอีกหลายชนิดเช่น อาจจต้องใช้ทั้งอุปกรณ์ดีสครีต (Discrete Device) และวงจรรวม (IC) และอาจจะประกอบด้วยเอสิกและหรือพีแอลดีหลายตัว เป็นต้น ดังนั้นหลังจากการรวมอุปกรณ์ต่างๆเหล่านี้เข้าด้วยกันจึงต้องมีการทดสอบการทำงานอีครั้งหนึ่งซึ่งถือว่าเป็นขั้นตอนการควบคุมคุณภาพของผลิตภัณฑ์อันเป็นขั้นตอนสุดท้ายของการออกแบบจากบนลงล่าง

ประโยชน์ของการออกแบบจากบนลงล่างคือการแก้ไขข้อผิดพลาดหรือการเพิ่มคุณสมบัติใหม่ให้กับระบบที่ออกแบบสามารถทำได้ง่ายเนื่องจากทุกขั้นตอนมีการตรวจสอบความถูกต้องก่อนการออกแบบในระดับต่อไป การตรวจสอบแก้ไขข้อผิดพลาดรวมทั้งการเพิ่มคุณสมบัติใหม่ให้กับระบบจะทำในระดับที่สูงกว่าซึ่งเป็นการง่ายเมื่อเปรียบเทียบกับการแก้ไขหรือเพิ่มเติมในระดับล่าง ยกตัวอย่างเช่น การออกแบบไมโครคอนโทรลเลอร์ ถ้าผู้ออกแบบต้องการสร้างชุดคำสั่งใหม่ การเริ่มแก้ไขในระดับเกทเพื่อให้ได้ชุดคำสั่งใหม่ดังกล่าวจะทำได้ยากมากเนื่องจากมีส่วนเกี่ยวข้องกันหลายส่วน แต่ถ้า

ผู้ออกแบบเริ่มการแก้ไขตั้งแต่ระดับบนสุด (Top Level) เช่นแก้ไขระดับฟังก์ชั่นและทำตามขั้นตอนเดิมเหมือนการออกแบบครั้งแรกจะสามารถทำได้ง่ายกว่า

ในปัจจุบันบริษัทผู้ผลิตซอร์ฟแวร์รวมไปถึงบริษัทผู้ผลิตไอซีได้ทำการพัฒนาโปรแกรมที่เรียกว่า “เครื่องมือสังเคราะห์” (Synthesis Tool) เพื่อรองรับการออกแบบจากบนลงล่าง กล่าวคือมันจะทำการสังเคราะห์จากระดับบนสุดของการออกแบบและให้ไฟล์วัตถุที่สามารถนำไปสร้างเอสิกหรือนำไปโปรแกรมอุปกรณ์ประเภทพีแอลดี ทำให้ผู้ออกแบบไม่ต้องสนใจรายละเอียดในระดับล่างหรือระดับต่ำกว่าระดับที่ทำการออกแบบเช่น การออกแบบระบบเชิงเลขในระดับฟังก์ชั่นผู้ออกแบบไม่ต้องสนใจว่าต้องใช้เกทชนิดอะไรมาต่อเพื่อให้ได้การทำงานตามที่ต้องการหรือการออกแบบวงจรเชิงเลขในระดับโครงสร้าง (ระดับเกท) ผู้ออกแบบไม่ต้องสนใจว่าจะต้องต่อทรานซิสเตอร์อย่างไร (ในกรณีที่ใช้เทคโนโลยีทีทีแอล) จึงจะได้การทำงานตามที่ต้องการ นอกเสียจากว่าระบบที่ออกแบบต้องการ การออกแบบที่ได้ผลดีที่สุด (Optimum) เช่น ทำงานเร็วที่สุดและหรือใช้ทรัพยากรน้อยที่สุด บางโปรแกรมได้เปิดโอกาสให้ผู้ออกแบบสามารถแก้ไขการออกแบบในระดับล่างได้เช่นกัน เช่น โปรแกรมของบริษัท Xilinx หรือ Altera ได้เตรียมเครื่องมือที่เรียกว่า ฟลอร์แพลนเอดิเตอร์ (Floorplane Editor) ซึ่งผู้ออกแบบสามารถแก้ไขการเชื่อมต่อภายในของชิพได้ เป็นต้น

4.3 โครงสร้างและส่วนการออกแบบ

ภาษาวีเอชดีแอลเป็นภาษาบรรยายพฤติกรรมของฮาร์ดแวร์หรืออาจกล่าวได้ว่าเป็นภาษาที่ใช้ในการสร้างระบบหรือสังเคราะห์วงจร (ในยุคเริ่มต้นของการสร้างภาษามีจุดประสงค์เพื่อนำมาสร้างระบบเชิงเลขแต่ในปัจจุบันกำลังพัฒนาให้สามารถบรรยายพฤติกรรมของระบบเชิงอุปมานซึ่งถูกเรียกว่า VHDL-AMS) ซึ่งแตกต่างจากภาษาทั่วไปที่เขียนขึ้นเพื่อใช้งานกับตัวประมวลผล เช่น ภาษาซี, ภาษาแอสเซมบลี ซึ่งเป็นการเขียนโปรแกรมเพื่อจัดการกับข้อมูล ดังนั้นการเขียนจึงถูกเรียกว่าการ “โปรแกรม” (Programming) ในขณะที่การเขียนภาษาวีเอชดีแอลถูกเรียกว่าการ “โมเดล” (Modeling) โดยที่ภาษาทั้งสองชนิดดังกล่าวมีหลักภาษาหรือไวยากรณ์ (Grammar) แตกต่างกันหรืออาจจะมีบางส่วนคล้ายกันบ้าง ดังนั้นก่อนที่จะศึกษาถึงรายละเอียดของการโมเดล (Model) ภาษาวีเอชดีแอล จึงมีความจำเป็นอย่างยิ่งที่จะต้องศึกษารายละเอียดของไวยากรณ์เพื่อการโมเดลที่ถูกต้องตามกฎเกณฑ์ (Syntax) และมีประสิทธิภาพ

4.3.1 ลักษณะของรูปแบบ (Model Styles)

ลักษณะของการเขียนรูปแบบ (Model) ด้วยภาษาวีเอชดีแอลสามารถแบ่งออกได้ดังนี้

- Behavioral Model เรียกอีกอย่างหนึ่งว่า Algorithmic Description เป็นรูปแบบ (Model) ที่บรรยายพฤติกรรมของระบบเชิงเลขซึ่งแสดงความสำพันธ์ระหว่างสัญญาณขาเข้าและสัญญาณขาออก เช่น ถ้าสัญญาณขาเข้ามีค่าตามเงื่อนไขตามที่กำหนดจะให้สัญญาณขาออกเป็นอย่างไร มีรูปแบบคล้ายการเขียนโปรแกรมด้วยภาษาระดับสูงทั่วไปเช่น ภาษาซี, ภาษาปาสคาล ฯลฯ ในการโมเดลภาษาวีเอชดีแอลแบบนี้จะไม่ให้รายละเอียดทางด้านโครงสร้างของฮาร์ดแวร์

- Dataflow Model เรียกอีกอย่างหนึ่งว่า Register Transfer Level (RTL) เป็นรูปแบบ (Model) ที่ให้รายละเอียดทางโครงสร้างของฮาร์ดแวร์มากขึ้นกล่าวคือ การโมเดลจะแสดงการดำเนินการเพื่อให้ได้มาซึ่งสัญญาณขาออกที่ต้องการยกตัวอย่างเช่น Output = ((not InputC) and InputA) or (InputC and InputB) จากตัวอย่างจะเห็นว่าเป็นการสร้างรีจีสเตอร์ (Register) นั่นเองซึ่งในความเป็นจริงรีจีสเตอร์เกิดจากการต่อกันของเกทเพื่อให้ได้ผลอย่างใดอย่างหนึ่งของสัญญาณขาออกเช่น ท็อกเกิลฟลิปฟลอฟ (Toggle Flip Flop) ค่าสัญญาณขาออกของมันจะกลับสถานะทุกๆคาบของสัญญาณนาฬิกา เป็นต้น

- Structural Model เป็นรูปแบบที่แสดงการเชื่อมต่อกันระหว่างอุปกรณ์ต่างๆที่ประกอบกันขึ้นเป็นวงจรหรือระบบเชิงเลขและสามารถเรียกอีกอย่างหนึ่งว่า Netlist Representation

- Mixed Level Model เป็นการโมเดลที่ใช้รูปแบบทั้งสามชนิดแรกรวมกันในการออกแบบวงจรหรือระบบเชิงเลข ซึ่งเป็นความอ่อนตัวของภาษาวีเอชดีแอลที่เปิดโอกาสให้ผู้ออกแบบสามารถใช้รูปแบบผสมกับวงจรหรือระบบเดียวกันได้ โดยเฉพาะกับระบบเชิงเลขขนาดใหญ่ซึ่งบางครั้งแต่ละส่วนการทำงานต้องการรายละเอียดที่ต่างกันซึ่งอาจจะเกิดจากวัตถุประสงค์หรือข้อจำกัดของการออกแบบ

4.3.2 คำศัพท์ที่ใช้ในภาษาวีเอชดีแอล (VHDL Term)

การเขียนรูปแบบของระบบเชิงเลขด้วยภาษาวีเอชดีแอลจะมีศัพท์เทคนิคเฉพาะ ดังนั้นในส่วนนี้จะอธิบายศัพท์บางคำที่เกี่ยวของกับการโมเดลภาษาวีเอชดีแอลกับงานวิจัยที่กำลังนำเสนอ

- Concurrency เป็นชุดคำสั่ง (Statements) ที่ทำงานพร้อมกัน (แข่งขนาน) คือ มันจะทำงานพร้อมกันในเวลาเดียวกันทุกครั้งเมื่อมีการเปลี่ยนแปลงค่าของสัญญาณและทำงานอิสระต่อกัน ซึ่งเป็นคุณสมบัติที่เป็นความจริงทางฟิสิกส์ของวงจรอิเล็กทรอนิกส์

- Sequential เป็นชุดคำสั่งที่ทำงานแบบเรียงลำดับขั้นจากบนลงล่าง คล้ายกับการเขียนโปรแกรมภาษาที่ใช้งานกับตัวประมวลผลทั่วไปเช่น ภาษาซีหรือแอสเซมบลี ชุดคำสั่งที่เป็น Sequential นี้จะใช้ในโปรแกรมย่อยและใน Process ซึ่งจะกล่าวถึงรายละเอียดในหัวข้อต่อไป

- Driver สัญญาณ (Signal) ต่างๆในภาษาวีเอชดีแอลจะถูกควบคุมด้วยตัวขับ (Driver) กล่าวคือสัญญาณจะได้รับค่าระดับของสัญญาณใหม่ได้ด้วยตัวขับนี้เอง

- Transaction เมื่อมีการกำหนดค่าๆหนึ่งให้กับสัญญาณโดยที่ค่าใหม่ที่สัญญาณได้รับอาจจะมีผลหรือไม่มีผลทำให้เกิดการเปลี่ยนแปลงระดับของสัญญาณก็ได้เช่น สัญญาณถูกเปลี่ยนจากลอจิก “0” เป็น ลอจิก ”0” ถือว่าเกิด Transaction กับสัญญาณ

- Event คือการเปลี่ยนแปลงค่าระดับของสัญญาณจากระดับหนึ่งไปสู่ระดับหนึ่งเช่น เปลี่ยนจากลอจิก “0” เป็น ลอจิก ”1” ถือว่าเกิด Event ฉะนั้นจะเห็นว่าการเกิด Event จะเกิด Transaction ด้วยเสมอ แต่การเกิด Transaction อาจไม่เกิด Event

- Sensitivity List คือรายชื่อของสัญญาณต่างๆที่ใช้กระตุ้นการทำงานของชุดคำสั่ง Concurrent เมื่อเกิด Event ขึ้นกับสัญญาณตัวใดตัวหนึ่งหรือหลายๆตัวพร้อมกันในรายชื่อของ Sensitivity List

- Object ในภาษาวีเอชดีแอลคำว่า Object ใช้เขียนเพื่อบ่งบอกถึงองค์ประกอบส่วนหนึ่งของรูปแบบ (Model) สามารถแบ่งออกได้เป็นสามชั้น (Class) ด้วยกันคือ

  - Constant หรือค่าคงที่ เป็น Object ประเภทหนึ่งที่เมื่อกำหนดค่าเริ่มต้นแล้วไม่สามารถเปลี่ยนแปลงแก้ไขได้ สามารถประกาศใช้ได้ในส่วนประกาศ (Declaration) ของรูปแบบ

  - Signal คือ Object ประเภทหนึ่งที่สามารถเปลี่ยนแปลงค่าได้ การเปลี่ยนแปลงค่าเกิดจากการขับของตัวขับ (Driver) แต่ Signal จะสามารถรับค่าได้เพียงค่าเดียวเท่านั้น ณ เวลาใดเวลาหนึ่ง Signal สามารถประกาศใช้ได้ในส่วนของ Concurrent Body ดังนั้นมันจึงสามารถถูกนำไปใช้ได้ตลอดโครงสร้างของรูปแบบ (Model) หรือที่เรียกว่า Global Object

  - Variable หรือตัวแปร เป็น Object ที่สามารถเปลี่ยนแปลงค่าได้แต่จะเก็บค่าได้เพียงค่าเดียวเท่านั้นในขณะเวลาหนึ่ง ตามมาตรฐานของ IEEE 1076-1987 ตัวแปรสามารถประกาศใช้ได้เฉพาะในส่วนของ Sequential Body เท่านั้นทำให้มันถูกสามารถใช้ได้เฉพาะในขอบเขตที่ประกาศเท่านั้น แต่มาตรฐาน IEEE 1076-1993 ซึ่งเป็นมาตรฐานใหม่ของภาษาวีเอชดีแอล สามารถประกาศเป็นตัวแปรชนิด Global ได้

- Data Type คือประเภทข้อมูล ได้แก่ Type ของ Object ซึ่งเป็นตัวกำหนดว่าค่าใดบ้างในกลุ่มของค่า (Set of Value) สามารถที่จะกำหนดให้กับ Object ได้ นอกจากนั้น Type ยังเป็นตัวกำหนดการทำงานในลักษณะต่างๆ ของ Object นั้นๆ ประเภทข้อมูลแบ่งออกเป็น 4 ประเภทคือ

  - Scalar ได้แก่ตัวเลขซึ่งในภาษาวีเอชดีแอลมี่ตัวเลขจำนวนเต็มบวก (Integer) เช่น 1, 30, 100 เป็นต้น และเลขจำนวนจริง (Real) เช่น 1.0, 1E2 เป็นต้น

- Enumeration กลุ่มของค่าประเภทนี้ได้แก่ ตัวหนังสือหรือชื่อต่างๆ

- Physical ได้แก่หน่วยทางฟิสิกส์ในระบบ SI

- File เป็นประเภทของข้อมูลภายนอกสามารถมีค่าได้หลายๆอย่าง

Predefined	Operators
Logical Operators:	NOT, AND, OR, NAND, NOR, XOR
Operand TYPE:	BIT, BOOLEAN
Result TYPE:	BIT, BOOLEAN
Relational Operators:	=, /=, <, <=, >, >=
Operand TYPE:	TYPE ใดๆ ก็ได้
Result TYPE:	BOOLEAN
Arithmetic Operator:	+, -, *, /, **, MOD, REM, ABS
Operand TYPE:	INTEGER, REAL, Physical
Result TYPE:	INTEGER, REAL, Physical
Concatenation Operators:	&
Operand TYPE:	array ของ TYPE ทุกประเภท
Result TYPE:	array ของ TYPE ทุกประเภท

รูปที่ 4.1 แสดงตัวดำเนินการและผลที่ได้จากการดำเนินการในภาษาวีเอชดีแอล

- Predefined Type คือประเภทของ Object ที่ได้กำหนดไว้แล้ว ได้แก่ Type ที่กำหนดไว้ใน Package ชื่อ Standard และกำหนดโดย IEEE ว่าจะต้องมีในระบบที่ใช้พัฒนาภาษาวีเอชดีแอลดังนั้นจึงไม่จำเป็นต้องประกาศใช้ในทุกๆรูปแบบที่เขียนขึ้น Type ประเภทนี้ได้แก่

- Boolean คือกลุ่มของค่า False และ True

- Bit คือกลุ่มของค่า “0” และ “1”

- Integer คือกลุ่มของค่าตั้งแต่ -214748347 ถึง 214748347

- Real คือกลุ่มของค่า -1.0E38 ถึง 1.05E38

- Character คือกลุ่มของค่า พยัญชนะ “A”-“Z”, “a”-”z” อักษรหรือเครื่องหมายพิเศษและตัวอักษรควบคุม

- Time ได้แก่หน่วยเวลาที่มีค่าพื้นฐานเป็นวินาที

- Severity Level คือกลุ่มของค่า Note, Warning, Error, Failure

- Subtype คือประเภทย่อยของข้อมูล สำหรับ Type ที่กำหนดไว้แล้วสามารถที่จะแบ่งออกเป็นกลุ่มย่อยลงไปได้อีก โดยที่องค์ประกอบของ Type ใหม่นี้จะเป็นส่วนหนึ่งของ Type เดิมหรือที่เรียกว่า Subtype

- Operator คือตัวดำเนินการเนื่องจาก Type ของ Object จะเป็นตัวบ่งบอกถึง Operator ที่ Object นั้นๆสามารถกระทำได้ ตลอดจน Type ของผลลัพธ์ที่ได้จากการทำงาน ในภาษาวีเอชดีแอลแบ่งตัวดำเนินการออกได้เป็นประเภทต่างๆแสดงดังรูปที่ 4.1

- Testbenches คือรูปแบบ (Model) ที่เขียนขึ้นเพื่อสร้างสัญญาณทดสอบเพื่อป้อนเป็นสัญญาณขาเข้าให้กับรูปแบบที่กำลังออกแบบในการจำลองการทำงาน

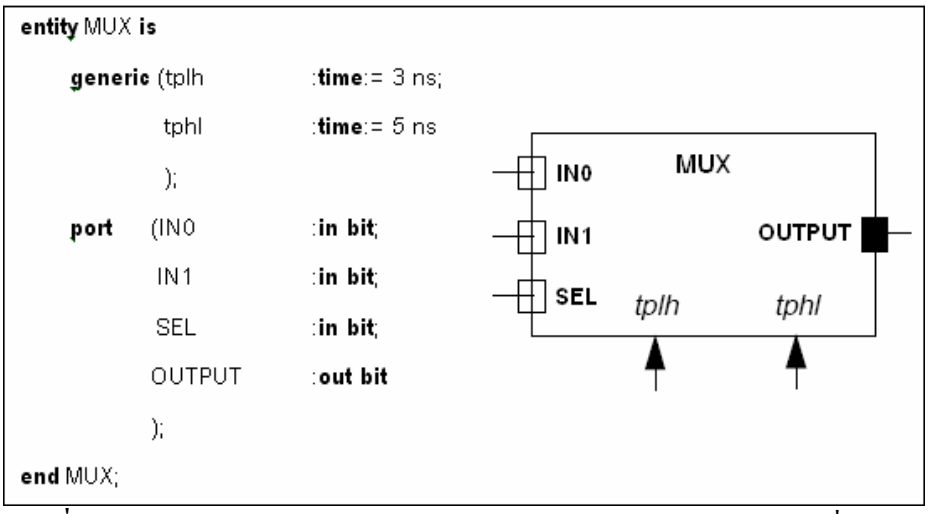
4.3.3 หน่วยของแบบ (Design Units)

จากที่ได้กล่าวไปแล้วในข้างต้นว่าภาษาวีเอชดีแอลถูกออกแบบมาเพื่อสร้างระบบ มันจึงมีความแตกต่างในลักษณะของโครงสร้างเมื่อเปรียบเทียบกับภาษาคอมพิวเตอร์ระดับสูงทั่วไป ดังนั้นก่อนที่จะทำการศึกษาการเขียนรูปแบบ (Modeling) จึงมีความจำเป็นอย่างยิ่งที่จะต้องแนะนำให้รู้จักหน่วยต่างๆของแบบ (Design Units) ที่ใช้ในภาษาและถือได้ว่าเป็นขั้นตอนแรกที่สำคัญที่สุดของการเรียนรู้การใช้ภาษาวีเอชดีแอลในการเขียนรูปแบบบรรยายพฤติกรรมของระบบเชิงเลข ส่วนประกอบต่างๆที่สำคัญและเป็นพื้นฐานของการเขียนรูปแบบระบบเชิงเลขที่สำคัญมี 4 หน่วยคือ

- Entity Design Unit
- Architecture Design Unit
- Package Design Unit
- Configuration Design Unit

4.3.3.1 Entity Design Unit

หน่วยของแบบส่วนนี้เป็นส่วนที่ใช้กำหนดจุดต่อ, ประเภทของค่า, และทิศทางการไหลของสัญญาณว่าเป็นสัญญาณขาเข้าหรือขาออกของแต่ละจุดต่อหรือช่องทาง (Port) ของรูปแบบ เพื่อการติดต่อระหว่างโลกภายนอกกับรูปแบบที่เขียนขึ้น อธิบายให้เข้าใจง่ายๆก็คือสมมติให้รูปแบบที่กำลังออกแบบเป็นไอซีมัลติเพล็ก หน่วย Entity จะใช้ในการกำหนดชื่อของขาของไอซีที่กำลังออกแบบและกำหนดแต่ละขาว่าเป็นขาออกหรือขาเข้า นอกจากนั้นยังสามารถใส่ข้อมูลเกี่ยวกับเวลาเช่น Propagation Delay Time: t_{pd} ฯลฯ เพื่อใช้ในการจำลองการทำงานของระบบ

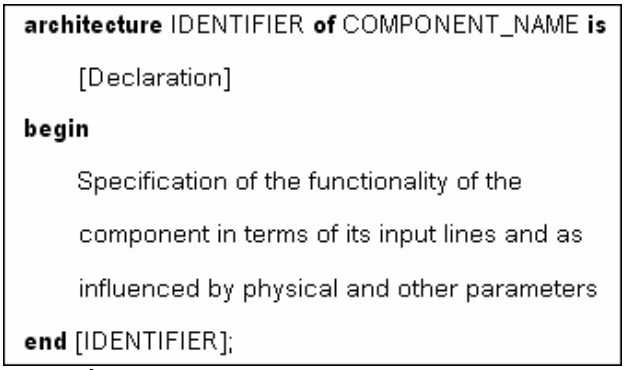


รูปที่ 4.2 (ซ้าย) แสดงโครงสร้างของหน่วย Entity (ขวา) แสดงมุมมองการเชื่อมต่อ

จากรูปที่ 4.2 เป็นรูปแบบ (Model) ของอุปกรณ์ชื่อมัลติเพล็กโดยชื่อ (MUX) จะเขียนอยู่หลังคำสั่ง entity คำสั่ง generic ใช้กำหนดค่าคงที่ tphl และ tplh ซึ่งเป็น Object ประเภทเวลามีค่าตายตัว (Default Value) อยู่ที่ 3 และ 5 nS. ตามลำดับ คำสั่ง port ใช้กำหนดชื่อ, ประเภทและทิศทางของสัญญาณสำหรับรูปแบบนี้จะเห็นว่ามีช่องทาง (Port) การเชื่อมต่อทั้งหมดสี่ช่องทางแบ่งออกเป็นสามช่องทางขาเข้า คือ IN0, IN1, SEL และ หนึ่งช่องทางขาออกคือ OUTPUT ทุกช่องทางเป็นชนิดบิทหรือมีขนาด 1 เส้นของสัญญาณดังนั้นค่าระดับของสัญญาณเป็นได้แค่สองค่าคือ “0” และ ”1”

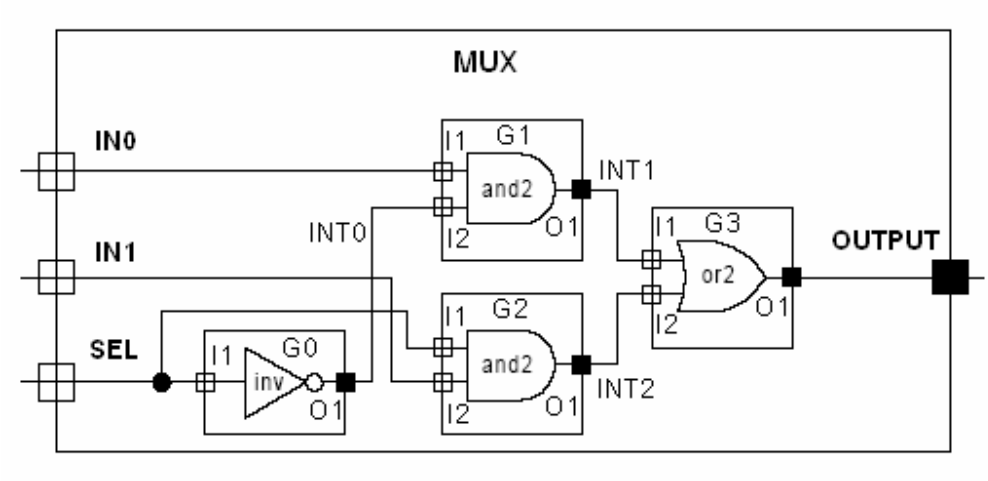
4.3.3.2 Architecture Design Unit

เป็นส่วนที่ใช้เขียนบรรยายกำหนดพฤติกรรมของรูปแบบซึ่งลักษณะของการเขียนรูป (Model Styles) แบบสามารถแบ่งออกได้เป็น 4 ลักษณะดังได้กล่าวไปแล้วในข้างต้น



รูปที่ 4.3 แสดงโครงสร้างของหน่วย Architecture

การเขียนเริ่มต้นด้วยคำสั่ง architecture และตามด้วยชื่อของหน่วย architecture นั้นๆ ซึ่งอาจจะตั้งตามลักษณะของรูปแบบที่กำลังบรรยายเช่น BEHAVE, RTL, STRUCT หรือ MIXED เป็นต้น จากนั้นจะเป็นชื่ออุปกรณ์ซึ่งอยู่หลังคำสั่ง of เป็นตัวที่บอกว่า architecture นั้นๆใช้บรรยาย entity design unit ใด ส่วนที่อยู่ระหว่างคำสั่ง architecture และ begin เป็นส่วนเผื่อเลือก (Option) ใช้ประกาศกำหนดค่าต่างๆที่จะนำไปใช้ภายใน architecture นั้นๆเช่น สัญญาณ (Signal), ค่าคงที่ (Constant), โปรแกรมย่อย (ฟังก์ชั่นและโปรซีเยอร์) และ อุปกรณ์ (Component) ส่วนที่ใช้บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้าและออกของรูปแบบซึ่งถูกกำหนดโดยการใช้คำสั่ง port ในหน่วย entity จะถูกบรรยายในบริเวณเนื้อที่ระหว่างคำสั่ง begin กับ end และที่สำคัญชุดคำสั่งที่ใช้ภายในบริเวณนี้จะเป็นชุดคำสั่งแบบแข่งขนาน (Concurrent Statement) เท่านั้น หน่วย architecture จะถูกปิดท้ายด้วยคำสั่ง end ตามด้วยชื่อของหน่วยซึ่งเป็นส่วนเผื่อเลือก



รูปที่ 4.4 แสดงวงจรมัลติเพล็กขนาด 2:1

```
architecture RTL of MUX is
begin
      OUTPUT <= ((not SEL) and IN0) or (SEL and IN1);
end RTL;
```

รูปที่ 4.5 แสดงหน่วย Architecture ที่ใช้ Model Style แบบ Register Transfer Level (RTL)

รูปที่ 4.5 เป็นตัวอย่างการเขียนรูปแบบ (Model) ของหน่วย Architecture สำหรับวงจรมัลติเพล็กตามรูปที่ 4.4 ส่วนที่บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า (IN0, IN1, SEL) และ

ข้อมูลที่ไหลออก (OUTPUT) ใช้ชุดคำสั่งแบบแข่งขนานเพียงชุดเดียว การบรรยายในลักษณะนี้เรียกว่า Data Flow Description หรือ Register Transfer Level (RTL)

```
architecture STRUCT of MUX is
        component inv
                port (I1 :in bit; O1 :out bit);
        end component;
        component and2
                port (I1,I2 :in bit; O1 :out bit);
        end component;
        component or2
                port (I1,I2 :in bit; O1 :out bit);
        end component;
        signal INT0, INT1, INT2          :bit;
begin
        G0: inv;
        port map (I1 => SEL, O1 => INT0);
        G1: and2;
        port map (I1 => IN0, I2 => INT0, O1 => INT1);
        G2: and2;
        port map (I1 => SEL, I2 => IN1, O1 => INT2);
        G3: or2;
        port map (I1 => INT1, I2 => INT2, O1 => OUTPUT);
end STRUCT;
```

รูปที่ 4.6 แสดงหน่วย Architecture ที่ใช้ Model Style แบบ Structural Description

รูปที่ 4.6 เป็นการเขียนรูปแบบ (Model) ในลักษณะ Structure โดยใช้สัญญาณ (Signal) ทั้งหมด 3 เส้นคือ INT0, INT1, INT2 ในการเชื่อมต่อระหว่างลอจิกเกทกับจุดต่อ (Port)

```
architecture BEHAVE of MUX is
begin
    process (IN0, IN1, SEL)
    begin
        if (SEL = '0') then
            OUTPUT <= IN0;
        else
            OUTPUT <= IN1;
        end if;
    end process;
end BEHAVE;
```

รูปที่ 4.7 แสดงหน่วย Architecture ที่ใช้ Model Style แบบ Behavioral Description

จากรูปที่ 4.7 จะเห็นว่าการบรรยายใช้การแสดงความสัมพันธ์ระหว่างสัญญาณขาเข้ากับขาออกโดยการเขียนเงื่อนไข ในการบรรยายไม่ได้ให้รายละเอียดของโครงสร้างทางฮาร์ดแวร์ การบรรยายรูปแบบในลักษณะนี้เรียกว่า Behavioral Model หรือ Algorithmic Description

4.3.3.3 Package Design Unit

ข้อมูลที่เป็นประโยชน์ในการเขียนรูปแบบรวมไปถึงโปรแกรมย่อยสามารถนำไปเก็บไว้ในส่วนที่เรียกว่า Package ได้ ข้อมูลที่ถูกเก็บไว้นี้สามารถเรียกใช้ได้จากหน่วยอื่นๆเช่น หน่วย Entity, หน่วย Architecture, หรือจากหน่วย Package อื่นๆโดยใช้คำสั่ง Use ข้อมูลที่สามารถประกาศหรือเก็บไว้ในหน่วย Package ได้แก่

- Subprogram
- Types
- Constant
- Signal
- Aliases
- Attributes
- Component
- Disconnection Specification

หน่วย Package แบ่งออกเป็นสองส่วนคือ Package Declaration และ Package Body

- Package Body

เป็นส่วนที่ใช้บรรยายฟังก์ชั่นการทำงานของโปรแกรมย่อยซึ่งประกอบด้วยชุดคำสั่งลำดับ (Sequential Statement) รวมไปถึงการกำหนดค่าให้ Deferred Constants ที่ถูกประกาศไว้ในส่วนของ Package Declaration

- Package Declaration

เป็นส่วนที่ใช้ประกาศเพื่อทำให้หน่วยอื่นสามารถมองเห็นสิ่งที่เก็บไว้ในหน่วย Package และสามารถนำสิ่งต่างๆเหล่านั้นไปใช้งานได้ ถือได้ว่าเป็นส่วนที่สำคัญที่สุดของหน่วย Package กล่าวคือ รูปแบบที่กำลังเขียนอยู่จะไม่สามารถนำค่าหรือพฤติกรรมที่เก็บไว้ในส่วนของ Package Body ไปใช้งานได้ถ้ามันไม่ถูกประกาศในส่วนประกาศนี้ ซึ่งอาจจะเปรียบได้ว่าเป็นส่วนที่ใช้ในการติดต่อระหว่างหน่วย Package กับโลกภายนอกนั่นเอง

```
-- Package Declaration
package PACK_FUNC is
      function MEAN (A, B, C :real) return real;
end PACK_FUNC;
-- Package Body
package body PACK_FUNC is
      function MEAN (A, B, C :real) return real is
      begin
            return (A+B+C)/3.0;
      end MEAN;
end PACK_FUNC;
```

รูปที่ 4.8 แสดงหน่วย Package ซึ่งภายในประกอบด้วยโปรแกรมย่อยชื่อ Mean

4.3.3.4 Configuration Design Unit

ในหนึ่งระบบเชิงเลขที่ใช้ภาษาวีเอชดีแอลในการอกแบบสามารถมีหน่วย Entity ได้เพียงหนึ่งหน่วยเท่านั้น ในขณะที่ในหนึ่งหน่วยของ Entity สามารถมีหน่วย Architecture ที่เป็นหน่วยรองได้หลายหน่วยแต่สามารถเลือกใช้ได้เพียงหนึ่งรูปแบบเท่านั้น ดังนั้นในการออกแบบจะต้องใช้หน่วย Configuration เพื่อกำหนดว่าหน่วย Entity นั้นๆต้องการใช้หน่วย Architecture หน่วยใด

```
entity and2 is
    generic (ttl_delay      :time:= 3 ns);
    port     (IN1           :in bit;
              IN2           :in bit;
              OUTPUT        :out bit
             );
end and2;
architecture RTL of and2 is
begin
    OUTPUT <= IN1 and IN2 after ttl_delay;
end RTL;
architecture BEHAVE of and2 is
begin
    process (IN1,IN2)
        if (IN1 = '1' and IN2 = '1') then
            OUTPUT <= '1' after ttl_delay;
        else
            OUTPUT <= '0';
        end if;
    end process;
end BEHAVE;
configuration EXAMPLE of and2 is
    for RTL
    end for;
end EXAMPLE;
```

รูปที่ 4.9 แสดงรูปแบบ (Model) ที่มีหน่วย Architecture สองหน่วย

รูปที่ 4.9 เป็นรูปแบบ (Model) ของระบบเชิงเลขอย่างง่ายในการสร้างแอนเกตขนาดสองอินพุท ประกอบด้วยหน่วย Entity หนึ่งหน่วยชื่อ and2 และหน่วย Architecture สองหน่วยซึ่งหมายความว่ารูปแบบนี้มีลักษณะการเขียนรูปแบบ (Model Styles) สองแบบคือ แบบ Register Transfer Level (RTL) และแบบ Behavioral แต่ดังที่ได้กล่าวไปแล้วในข้างต้นว่าหนึ่งระบบจะสามารถเลือกใช้หน่วย Architecture ได้เพียงหนึ่งรูปแบบเท่านั้น ดังตัวอย่างตามรูปที่ 4.9 หน่วย Architecture ชื่อ RTL ได้ถูกกำหนดให้ใช้กับระบบนี้ด้วยคำสั่ง for RTL ซึ่งอยู่ภายในหน่วย Configuration

4.4 ตัวอย่างการออกแบบระบบเชิงเลขด้วยภาษาวีเอชดีแอล

การวิจัยคิดค้นอัลกอริทึมใหม่ๆต้องการระบบการคำนวณที่มีประสิทธิภาพมากขึ้นรวมไปถึงความต้องการเพื่อให้สามารถสร้างได้จริงในทางปฏิบัติ (Realization) เป็นผลทำให้การออกแบบระบบเชิงเลขมีความซับซ้อนมากขึ้น ในอดีตการออกแบบระบบเชิงเลขใช้การเขียนวงจร แต่เมื่อระบบที่ต้องการมีความซับซ้อนมากขึ้นการออกแบบโดยการเขียนวงจรที่ประกอบด้วยลอจิกเกตจำนวน 100,000 เกต คงทำได้ยากมาก การออกแบบด้วยภาษาบรรยายพฤติกรรมของฮาร์ดแวร์จึงได้มีการพัฒนาขึ้น อย่างไรก็ตามถึงแม้ว่าผู้ออกแบบระบบที่ใช้ภาษาบรรยายพฤติกรรมของฮาร์ดแวร์อาจไม่มีความจำเป็นที่จะต้องมีความรู้ในระดับโครงสร้าง (ระดับเกต) แต่ความเข้าใจพื้นฐานในระดับโครงสร้างจะมีส่วนช่วยให้การออกแบบระบบมีประสิทธิภาพมากขึ้น เช่น ใช้ทรัพยากรน้อย, ทำงานเร็ว ฯลฯ เป็นต้น ดังนั้นหัวข้อนี้จะกล่าวถึงตัวอย่างของการออกแบบระบบเชิงเลขพื้นฐานโดยใช้ภาษาวีเอชดีแอลเพื่อเป็นแนวทางสำหรับการศึกษาในระดับสูงต่อไป

4.4.1 การออกแบบวงจรเชิงเลขแบบ Combination

วงจรเชิงเลขแบบ Combination คือวงจรที่ให้ค่าของสัญญาณขาออกขณะนั้นขึ้นอยู่กับค่าของสัญญาณขาเข้าขณะนั้นโดยไม่สนใจสัญญาณขาเข้าสถานะก่อนหน้า [16] ซึ่งแตกต่างกับวงจรเชิงเลขแบบ Sequential ซึ่งจะกล่าวถึงรายละเอียดในหัวข้อต่อไป

4.4.1.1 ตัวถอดรหัสจาก 2 เป็น 4 (2 to 4 Decoder)

ใช้สำหรับการถอดรหัสสัญญาณที่ถูกเข้ารหัสมาในรูปแบบเดียวกันเช่น เลขฐานสองขนาด n บิตจะสามารถแทนค่าได้ทั้งหมดเท่ากับ 2^n ค่าเป็นต้น รูปที่ 4.10 แสดงรูปแบบขอตัวถอดรหัสซึ่งมีสัญญาณขาเข้าชื่อ A ขนาด 2 บิต (กำหนดโดยคำสั่ง 1 downto 0) มี Type เป็น std_ulogic_vector และสัญญาณขาออกชื่อ Z มีขนาด 4 บิต การทำงานใช้การตรวจสอบเงื่อนไขของสัญญาณขาเข้าโดยคำสั่ง when ถ้าสัญญาณขาเข้ามีเงื่อนไขตรงตามที่กำหนดสัญญาณขาออกจะถูกขับตามค่าที่กำหนดไว้ในเครื่องหมายคำพูดที่อยู่ด้านซ้าย แต่ถ้าสัญญาณไม่ตรงกับทั้งสี่เงื่อนไขที่กำหนดสัญญาณขาออกจะถูกขับด้วยค่าที่ไม่ทราบสถานะ (“XXXX”)

4.4.1.2 ตัวมัลติเพล็ก (Multiplexers)

ตัวมัลติเพล็กใช้สำหรับเลือกสัญญาณขาเข้าที่มีอยู่หลายๆสัญญาณเพื่อให้ออกที่สัญญาณขาออกที่มีอยู่เพียงสัญญาณเดียว ตัวอย่างของรูปแบบแสดงดังรูปที่ 4.11 เป็นมัลติเพล็กขนาด 4:1 (4 สัญญาณขาเข้าและ 1 สัญญาณขาออก) สัญญาณขาเข้า A, B, C, D และสัญญาณขาออก Y มี Type เป็น std_ulogic ในขณะที่สัญญาณชื่อ SEL ซึ่งใช้ในการเลือกสัญญาณขาเข้ามี Type เป็น std_ulogic_vector ขนาด 2 บิต การทำงานใช้การตรวจเงื่อนไขของสัญญาณ SEL ซึ่งอยู่ด้านขวาคำสั่ง when สัญญาณขา

ออกจะถูกขับด้วยสัญญาณขาเข้าเมื่อสัญญาณ SEL ตรงกับเงื่อนไขที่อยู่ในบรรทัดเดียวกัน แต่ถ้าสัญญาณไม่ตรงกับทั้งสี่เงื่อนไขที่กำหนดสัญญาณขาออกจะถูกขับด้วยค่าที่ไม่ทราบสถานะ (“X”)

```
library ieee;
use ieee.std_logic_1164.all;
entity DECODER is
    port    ( A     :in std_ulogic_vector(1 downto 0);
              Z     :out std_ulogic_vector(3 downto 0)
            );
end DECODER;
architecture BEHAVE of DECODER is
begin
    Z <= "0001" when A = "00" else
    Z <= "0010" when A = "01" else
    Z <= "0100" when A = "10" else
    Z <= "1000" when A = "11" else
    "XXXX";
end BEHAVE;
```

รูปที่ 4.10 แสดงรูปแบบของตัวถอดรหัสจาก 2 เป็น 4

4.4.1.3 ตัวบวก (Adder)

ตัวอย่างที่กำลังนำเสนอเป็นตัวบวกขนาด 1 บิต ใช้การเขียนรูปแบบในลักษณะของโครงสร้าง สัญญาณขาเข้าประกอบด้วย A, B, และ CIN มี Type เป็น std_ulogic และสัญญาณขาออกประกอบด้วย SUM และ COUT

4.4.2 การออกแบบวงจรเชิงเลขแบบ Sequential

ในความเป็นจริงระบบเชิงเลขส่วนใหญ่เป็นแบบซีเควนเชียล (Sequential) ซึ่งค่าของสัญญาณขาออกขึ้นอยู่กับค่าของสัญญาณขาเข้าปัจจุบันและค่าของสัญญาณขาเข้าก่อนหน้า ระบบจะถูกเรียกว่า อะซิงโครนัส (Asynchronous) เมื่อสัญญาณขาออกเปลี่ยนค่าทันทีเมื่อสัญญาณขาเข้าเปลี่ยนแปลงสถานะ แต่ถ้าระบบต้องใช้สัญญาณนาฬิกา (Clock) ในการควบคุมจังหวะของการทำงานจะถูกเรียกว่าเป็นระบบซิงโครนัส (Synchronous) อุปกรณ์ที่ใช้สัญญาณนาฬิกาในการควบคุมการทำงานจึงเป็นส่วนประกอบที่สำคัญมาก ดังนั้นก่อนที่จะกล่าวถึงการออกแบบระบบซีเควนเชียลจะแสดงตัวอย่างของการเขียนรูปแบบในการสร้างอุปกรณ์ที่จำเป็นในระบบเสียก่อน

```
library ieee;
use ieee.std_logic_1164.all;
entity MUX is
    port    ( A, B, C, D    :in std_ulogic;
              SEL           :in std_ulogic_vector(1 downto 0);
              Y             :out std_ulogic
            );
end MUX;
architecture BEHAVE of MUX is
begin
    with SEL select
    Y <= A when "00",
         B when "01",
         C when "10",
         D when "11",
         'X' when others;
end BEHAVE;
```

รูปที่ 4.11 แสดงรูปแบบของตัวมัลติเพล็กขนาด 4:1

4.4.2.1 เอส-อาร์ แลทช์ (SR Latch)

อุปกรณ์สองประเภทที่มีการทำงานคล้ายกันมากได้แก่ แลทช์ (Latch) และ ฟลิปฟล็อป แต่ในที่นี้จะกำหนดนิยามของแลทช์ว่าเป็นหน่วยความจำที่ทำงานโดยใช้ระดับของสัญญาณ (Level Sensitive) ส่วนฟลิปฟล็อปคือหน่วยความจำที่ทำงานโดยใช้ขอบของสัญญาณ (Edge Sensitive) รูปที่ 4.13 แสดงรูปแบบของเอสอาร์แลชท์

4.4.2.2 ดี-ฟลิปฟล็อป (D-Flip-Flop)

การทำงานของดีฟลิปฟล็อปคือ สัญญาณขาเข้า (D) จะถูกส่งออกไปที่สัญญาณขาออก (Q) เมื่อสัญญาณนาฬิกาเปลี่ยนสถานะหรืออาจกล่าวได้ว่ามันทำงานโดยใช้ขอบของสัญญาณ ตามรูปแบบแสดงดังรูปที่ 4.14 ใช้ขอบขาขึ้นในการกระตุ้นการทำงานโดยใช้ฟังก์ชั่น rising_edge

```
library ieee;
use ieee.std_logic_1164.all;
entity FULLADDER is
      port     ( A, B, CIN      :in std_ulogic;
                 SUM, COUT   :out std_ulogic
               );
end FULLADDER;
architecture STRUCT of FULLADDER is
begin
      SUM <= A xor B xor CIN;
      COUT <= (A and B) or (A and CIN) or (b and CIN);
end STRUCT;
```

รูปที่ 4.12 แสดงรูปแบบของตัวบวกขนาด 1 บิต

```
library ieee;
use ieee.std_logic_1164.all;
entity SR_LATCH is
      port     ( S, R            :in std_ulogic;
                 Q, QBAR       :buffer std_ulogic
               );
end SR_LATCH;
architecture RTL of SR_LATCH is
begin
      Q <= '1' when R = '0' else
           '0' when S = '0' else
           Q;
      QBAR <= '1' when S = '0' else
              '0' when R = '0' else
              QBAR;
end RTL;
```

รูปที่ 4.13 แสดงรูปแบบของเอสอาร์แลชท์

```
library ieee;
use ieee.std_logic_1164.all;
entity DFF is
    port    (D, CLOCK, RESET    :in bit;
             Q                  :out bit
            );
end DFF;
architecture RTL of DFF is
begin
P0:
process (D, CLOCK, RESET)
begin
    if (RESET = '0') then
            Q <= '0';
    elsif rising_edge (CLOCK) then
            Q <= D;
    end if;
end process
P0;
end RTL;
```

รูปที่ 4.14 แสดงรูปแบบของดีฟลิปฟล็อป

4.4.2.3 หน่วยความจำชนิดรอม (Read Only Memory: ROM)

รูปแบบของรอมสามารถสร้างได้จากอาร์เรย์ของค่าคงที่ ตัวอย่างแสดงดังรูปที่ 4.15 เป็นรอมที่มีขนาดของคำ (Word Size) เท่ากับ 8 บิตและมีความจุ 4 ไบท์ (Capacity) กำหนดโดยค่าคงที่ชื่อ ROM ซึ่งมี Type เป็น ROM_ARRAY ในการอ่านค่าของแต่ละตำแหน่งภายในทำได้โดยการส่งตำแหน่งเข้าไปที่ ADDRESS ซึ่งมี Type เป็น integer มีความกว้างเท่ากับ 2 บิต ข้อมูลจะถูกขับออกที่สัญญาณขาออก DATA ซึ่งมีขนาดเท่ากับขนาดของคำ

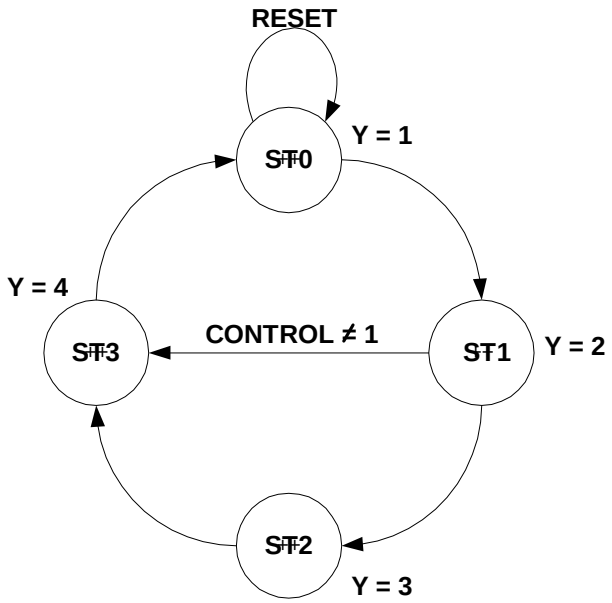
4.4.2.4 ไฟไนท์สเตทแมชีนส์ (Finite State Machines: FSM)

เป็นระบบเชิงเลขซึ่งมีการทำงานเป็นลำดับที่เฉพาะเจาะจงหรือมีจำนวนสถานะ (State) ที่จำกัด ไฟไนท์สเตทแมชีนส์ถูกนำมาประยุกต์ใช้งานอย่างกว้างขวางโดยเฉพาะกับตัวควบคุมที่ทำหน้าที่ควบคุมการทำงานของระบบเชิงเลขชนิดอื่นๆ [17] ยกตัวอย่างเช่น SDRAM Controller, IIC Bus Controller ฯลฯ ตัวอย่างรูปแบบของไฟไนท์สเตทแมชีนส์อย่างง่ายแสดงดังรูปที่ 4.17 ซึ่งบรรยาย

พฤติกรรมของ สเตทไดอะแกรมตามรูปที่ 4.16 ระบบประกอบด้วยสัญญาณขาเข้าที่เป็นสัญญาณควบคุมทั้งหมด 3 สัญญาณได้แก่ CLOCK, RESET, CONTROL และสัญญาณขาออกของระบบคือ Y ซึ่งเป็นสัญญาณขาออกชนิด Moore การทำงานของระบบมีดังนี้ ในกรณีที่สัญญาณ RESET มีค่าเป็น “1” สัญญาณขาออกจะมีค่าเท่ากับ 1 (ST0) แต่ในกรณีที่สัญญาณ RESET มีค่าเป็นค่าอื่นๆ สถานะ (State) จะถูกเปลี่ยนในช่วงขอบขาขึ้นของสัญญาณ CLOCK วนจาก ST0, ST1, ST2, ST3 และกลับมาที่ ST0 พิเศษที่ ST1 ระบบจะมีการตรวจสัญญาณ CONTROL ว่าเป็น “1” หรือไม่ ถ้าเป็น “1” สถานะจะเปลี่ยนไปเป็น ST2 แต่ถ้าเป็นค่าอื่นสถานะจะกระโดดไปที่ ST3 (โดยไม่ผ่าน ST2) รูปแบบแสดงดังรูปที่ 4.17 ถูกแบ่งออกเป็น 3 ส่วนได้แก่ ส่วนของ Next State, Current State, ทั้งสองส่วนนี้ทำงานแบบแข่งขนานเนื่องจากใช้ชุดคำสั่ง process และซิงโครนัส ส่วนที่สามเป็นส่วนของสัญญาณขาออก (Y) ซึ่งเป็นชนิด Combination

```
library ieee;
use ieee.std_logic_1164.all;
entity ROM_4x8 is
    port   (ADDRESS   :in integer range 0 to 3;
            DATA      :out std_ulogic_vector (7 downto 0)
           );
end ROM_4x8;
architecture RTL of ROM_4x8 is
    type ROM_ARRAY is array (0 to 3)
    of std_ulogic_vector (7 downto 0);
    constant ROM :ROM_ARRAY :=  (
                                "11001100",
                                "00110011",
                                "11110000",
                                "00001111",
                                );
begin
    DATA <= ROM (ADDRESS);
end RTL;
```

รูปที่ 4.15 แสดงรูปแบบของหน่วยความจำรอม



รูปที่ 4.16 แสดงสเตทไดอะแกรมของไฟไนท์สเตทแมชีนส์

```
library ieee;
use ieee.std_logic_1164.all;
entity FSM is
        port    ( CLOCK, RESET, CONTROL       :in std_logic;
                  Y                           :out integer range 0 to 4
                );
end FSM;
architecture RTL of FSM is
        type STATE_TYPE is (ST0, ST1, ST2, ST3);
        signal CURRENT_STATE, NEXT_STATE  :STATE_TYPE;
begin
```

รูปที่ 4.17ก แสดงรูปแบบของไฟไนท์สเตทแมชีนส์

```
--Next State Logic
COMB:process(CONTROL, CURRENT_STATE)
begin
        case CURRENT_STATE is
                when ST0 =>
                        NEXT_STATE <= ST1;
                when ST1 =>
                        if (CONTROL = '1') then
                                NEXT_STATE <= ST2;
                        else
                                NEXT_STATE <= ST3;
                        end if;
                when ST2 =>
                        NEXT_STATE <= ST3;
                when ST3 =>
                        NEXT_STATE <= ST0;
                when others =>
                        NEXT_STATE <= ST0;
        end case;
end process COMB;
```

รูปที่ 4.17ข แสดงรูปแบบของไฟไนท์สเตทแมชีนส์ (ต่อ)

```
--Current State Logic
SEQ:process(CLOCK, RESET)
begin
        if (RESET = '1') then
                CURRENT_STATE <= ST0;
        elsif rising_edge (CLOCK) then
                CURRENT_STATE <= NEXT_STATE;
        end if;
end process SEQ;
--Output Logic
with CURRENT_STATE select
        Y <= 1 when ST0,
                2 when ST1,
                3 when ST2,
                4 when ST3,
                1 when others;
end RTL;
```

รูปที่ 4.17ค แสดงรูปแบบของไฟไนท์สเตทแมชีนส์ (ต่อ)