เครื่องนับแยกและทอนเหรียญอัตโนมัติ

AUTOMATIC COIN CLASSIFICATION AND CHANGE MACHINE

โดย

นายจีระศักดิ์ เลี้ยงเจริญกิจ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

เครื่องนับแยกและทอนเหรียญอัตโนมัติ

AUTOMATIC COIN CLASSIFICATION AND CHANGE MACHINE

โดย

นายจีระศักดิ์ เลี้ยงเจริญกิจ

อาจารย์ที่ปรึกษา

รศ.ดร.มนัส สังวรศิลป์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

แบบฟอร์มรับรองความพร้อมในการสอบ

เครื่องนับแยกและทอนเหรียญอัตโนมัติ

Automatic Coin Classification and Change Machine

นายจีระศักดิ์ เลี้ยงเจริญกิจ 43015204

โครงงานนี้ได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

.....

(รศ.ดร.มนัส สังวรศิลป์)

รายงาน ปีการศึกษา 2545

ภาควิชา วิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องนับแยกและทอนเหรียญอัตโนมัติ

(Automatic Coin Classification and Change Machine)

ผู้จัดทำ

1. นายจีระศักดิ์ เลี้ยงเจริญกิจ 43015204

.....อาจารย์ที่ปรึกษา

(รศ.ดร.มนัส สังวรศิลป์)

เครื่องนับแยกและทอนเหรียญอัตโนมัติ

นายจีระศักดิ์ เลี้ยงเจริญกิจ
รศ.ดร.มนัส สังวรศิลป์
ปีการศึกษา 2545

บทคัดย่อ

รายงานนี้เสนอการสร้างเครื่องนับแยกและทอนเหรียญอัตโนมัติ โดยเป็นการใช้งานอุปกรณ์ FPGA ซึ่งเป็นอุปกรณ์ในลักษณะของ Programmable Device ที่เราสามารถโปรแกรมตัวอุปกรณ์FPGA ให้เป็นวงจรดิจิตอลใดๆก็ได้ โดยการออกแบบวงจรดิจิตอลภายในอุปกรณ์ FPGA(Field-Programmable Gate Array) นั้นจะใช้การวาดวงจร(Schematic) ร่วมกับภาษาบรรยายเชิงพฤติกรรมของฮาร์ดแวร์ (Hardware Description Language) ด้วยภาษา VHDL(VHSIC Hardware Description Language)

การออกแบบจำเป็นจะต้องรับเหรียญ 1 บาท 5 บาทและ 10 บาทได้ แสดงจำนวนเงินที่รับเข้ามา และแสดงจำนวนเงินทอนได้ ซึ่งการควบคุมการทำงานทั้งหมดจะเป็นแบบอัตโนมัติ ส่วนที่ทำหน้าที่ในการประมวลผลจะถูกควบคุมโดยการใช้อุปกรณ์ FPGA(Field-Programmable Gate Array)

Automatic Coin Classification and Change Machine

Mr. Jerasak Liangjaroenkit

Assoc.Prof.Dr.Manus Sangvorasin (Advisor)

Academic Year 2002

Abstract

This project presents a design of Automatic Coin Classification and Change Machine, which is applied to FPGA chip. FPGA is a Programmable Device, which can be programed to be any variety of digital circuits . To design digital circuit in FPGA(Field-Programmable Gate Array) , we have to draw the schematic along with the VHSIC Hardware Description Language (VHDL) (VHSIC Hardware Description Language)

System can get three types of coin, 1 bath, 5 bath, and 10 bath. It also can display amount of money and the change which automatically classifies coins, and counts the total amount of money. The processing unit is controlled by FPGA chip.

กิตติกรรมประกาศ

ในการทำโครงงานเครื่องนับแยกและทอนเหรียญอัตโนมัติ ประสบความสำเร็จลุล่วงไปได้ด้วยดีเนื่องจากได้รับคำแนะนำตลอดจนแนวคิดในการทำงานจากอาจารย์ที่ปรึกษา รศ.ดร.มนัส สังวรศิลป์ รวมทั้งอาจารย์ทุกท่านในภาควิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่ได้ให้ความรู้รวมไปถึงการแนะนำแนวทางในการแก้ไขปัญหาจากการทดลอง รวมทั้งพี่ๆและเพื่อนๆที่ให้ความช่วยเหลือตลอดมา จึงทำให้สามารถจัดทำโครงงานนี้ได้อย่างสมบูรณ์

ทางผู้จัดทำขอแสดงความขอบคุณทุกท่านอีกครั้ง ณ โอกาสนี้ด้วย

ผู้จัดทำ

สารบัญ

บทที่ 1
บทนำ

ในโครงงานนี้ได้นำเสนอวิธีการสร้างเครื่องนับแยกและทอนเหรียญอัตโนมัติ โดยในโครงงานนี้ได้ใช้เทคโนโลยี FPGA เข้ามาเป็นตัวหลักในการทำงานโดยใช้ภาษา VHDL ในการโปแกรมชิพเพราะรูปแบบของภาษา VHDL ที่เข้าใจได้ง่ายมีโครงสร้างของภาษาที่สามารถปรับเปลี่ยนให้เข้ากับ hardware ได้ง่ายมี statement ให้ใช้งานอยู่หลายตัว สามารถออกแบบในรูปแบบ top-down design ได้และมี library ต่างๆให้เลือกใช้งานมากทำให้ออกแบบได้ง่ายขึ้น เป็นประโยชน์ต่อ programmer

1.1 วัตถุประสงค์

- เพื่อศึกษาการออกแบบและการสร้างเครื่องนับแยกและทอนเหรียญอัตโนมัติ
- เพื่อศึกษาการใช้งานอุปกรณ์ FPGA ในการสร้างเครื่องนับแยกและทอนเหรียญอัตโนมัติ
- เพื่อศึกษาภาษา VHDL ที่ใช้ในอธิบายและการออกแบบวงจรดิจิตอล
- เรียนรู้และใช้งานโปรแกรม Max+plusII เพื่อให้เกิดความรู้ความชำนาญมากขึ้น
- สามารถนำความรู้ที่ได้จากโครงงานนี้ นำไปประยุกต์ใช้งานกับโครงงานอื่นๆ ได้

1.2 การนำเสนอโครงงาน

จะมีการเสนอรายละเอียดของโครงงานโดยสมบูรณ์ ตามหัวข้อต่อไปนี้

- บทที่ 1 บทนำ กล่าวถึงตัวโครงงานและคุณสมบัติการทำงานพื้นฐาน
- บทที่ 2 กล่าวถึงทฤษฎีที่เกี่ยวข้องกับการกรองเชิงตัวเลขด้วย FPGA ซึ่งประกอบด้วยรายละเอียดของอุปกรณ์ FPGA การเขียนโปรแกรมบรรยายเชิงพฤติกรรมด้วยภาษา VHDL
- บทที่ 3 หลักการทำงานและการออกแบบ
- บทที่ 4 ผลการทดลอง
- บทที่ 5 สรุปและวิจารณ์ผลการทดลอง

บทที่ 2
ทฤษฎี และ หลักการ

2.1 เทคโนโลยีของ FPGA

2.1.1 Field-Programmable Gate Array (FPGA)

FPGA เป็นอุปกรณ์ในลักษณะของ Programmable Device ที่เราสามารถโปรแกรมตัวมันให้เป็นวงจรดิจิตอลใดๆก็ได้ สำหรับ FPGA แล้วนับว่าเป็นอุปกรณ์ตัวใหม่ในตระกูลของ ASIC ซึ่งมีการเจริญเติบโตอย่าง รวดเร็วและมีบทบาทที่สำคัญในการเข้ามาแทนที่ระบบอิเล็กทรอนิกส์ที่ใช้ TTL โครงสร้าง ภายในของ FPGA ประกอบไปด้วยอะเรย์ของลอจิกเกทต่างๆมากมาย ซึ่งในปัจจุบันความจุเกทภายในตัวชิพ FPGA ได้เพิ่มขึ้น จากระดับไม่กี่พันตัวจนถึงระดับล้านตัวซึ่งสามารถรองรับวงจรดิจิตอลที่มีความสลับซับซ้อนได้เป็นอย่างดี นอกจากนี้ในด้านการออกแบบพัฒนาและทดสอบก็ทำได้ง่ายซึ่งในปัจจุบัน การออกแบบวงจรโดยใช้ FPGA กำลังเป็นที่นิยมและมีแนวโน้มที่จะนำมาใช้งานมากขึ้นเรื่อยๆ ในปัจจุบันมี FPGA อยู่ 4 ชนิดที่วางขายอยู่ในท้องตลาดได้แก่ Symmetrical Array, Row-Based, Hierarchical PLD และ Sea-of-Gates ซึ่งแต่ละชนิดก็มีลักษณะการเชื่อมต่อภายในและการโปรแกรมที่แตกต่างกันไป นอกจากนี้ในการแบ่งประเภทของ FPGA อาจแบ่งได้ตามเทคโนโลยีที่ใช้ในการโปรแกรม ซึ่งมีอยู่ 2 แบบคือ การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพของตัวชิพ และการโปรแกรม โดยการใช้หน่วยความจำ

1. การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพ

1.1. Fuse เป็นวิธีการ โปรแกรมที่สามารถทำได้เพียงครั้งเดียวเท่านั้น ซึ่งหลังจากที่โปรแกรม

แล้วจุดเชื่อมต่อจะขาดจากกัน

1.2. Anti Fuse เป็นวิธีการโปรแกรมที่คล้ายกับแบบ Fuse แต่ต่างกันที่หลังจากทำการโปรแกรม แล้วจุดเชื่อมต่อจะเชื่อมถึงกัน

2. การโปรแกรมโดยใช้หน่วยความจำ

2.1. EEPROM Based FPGA

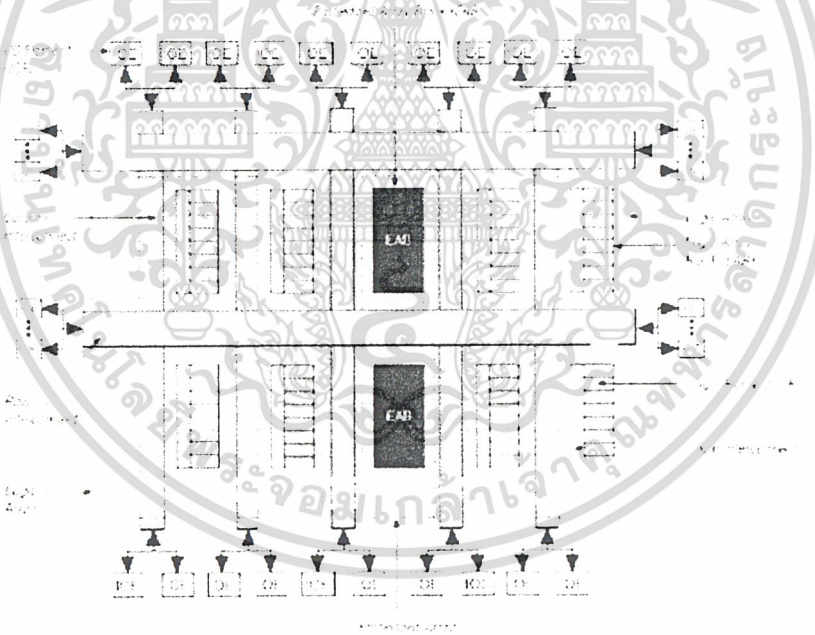
FPGA ที่ใช้การโปรแกรมแบบนี้มักเรียกว่า CPLD ซึ่งเทคโนโลยีที่ใช้จะเหมือนกับ EEPROM ทำให้มีความจุของเกทต่ำ โดยทั่วไปจะน้อยกว่า 20,000 เกทแต่ข้อดีของ EEPROM Based FPGA คือสามารถเก็บข้อมูลที่โปรแกรมลงไปได้โดยไม่จำเป็นต้องมีไฟเลี้ยง และในการโปรแกรมจะใช้ทรานซิสเตอร์ 1 ตัวต่อ 1 บิท ซึ่งการโปรแกรมสามารถทำได้ประมาณ 10,000 ครั้ง

2.2. SRAM Based FPGA

FPGA แบบนี้จะใช้เทคโนโลยีในการโปรแกรมเหมือนกับ SRAM (Static RAM) ทำให้สามารถ โปรแกรมซ้ำได้โดยไม่จำกัดจำนวนครั้ง นอกจากนี้ยังมีความจุของเกทในระดับปานกลางถึงสูงมาก (ประมาณ 10,000 - 1,000,000 เกท) ซึ่งข้อดีของ SRAM Based FPGA คือใช้เวลาในการโปรแกรมน้อย (ระดับ nSec) การโปรแกรมทำได้ง่ายเทียบได้กับการเขียน SRAM ทั่วไป และเหมาะสำหรับการออกแบบวงจรที่มีความสลับซับซ้อน ส่วนข้อเสียคือไม่สามารถเก็บโปรแกรมในภาวะที่ไม่มีไฟเลี้ยงได้ ดังนั้น FPGA ชนิดนี้จึงมักใช้ควบคู่กับ ROM เพื่อเก็บโปรแกรมและทำการโหลดโปรแกรมลงในตัวชิพในขณะที่เริ่มต้นใช้งาน

2.1.2 โครงสร้างภายในของ FPGA

ลักษณะโครงสร้างภายในของ FPGA จะเป็นอะเรย์ของบล็อกลอจิกที่สามารถทำการโปรแกรมได้ดังรูปที่ 2.1



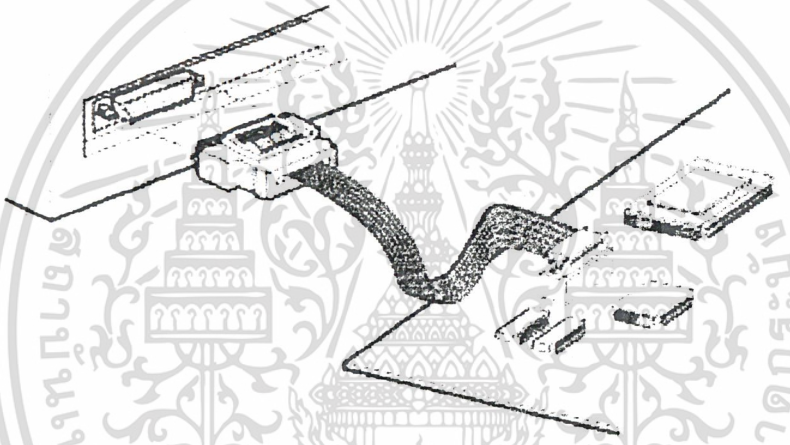
รูปที่ 2.1 โครงสร้างภายในของ FPGA ตระกูล FLEX 10K

2.1.3 ปัจจัยที่ทำให้การออกแบบ FPGA ทำได้ง่ายและสะดวกรวดเร็ว

1. ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพ เพียงแต่มีความรู้เกี่ยวกับขั้นตอนการออกแบบลอจิกก็เพียงพอแล้ว ต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายในรวมถึง ภาษา Assembly ของไมโครโปรเซสเซอร์ตัวนั้นด้วย

2. มีการออกแบบโดยใช้ภาษาในการอธิบายการทำงานของวงจร หรือ HDL (Hardware Description Language) เป็นเครื่องมือในการออกแบบ ซึ่งเป็นวิธีการที่มีความยืดหยุ่นสูง ทำได้รวดเร็ว และไม่จำเป็นต้องทราบถึงลักษณะของวงจรที่ต้องการว่าจะเชื่อมต่อกันอย่างไร เพียงแต่กำหนดลักษณะการทำงานให้มัน จากนั้นตัวซอฟต์แวร์จะทำ Synthesis and Optimize ให้ทั้งหมด นอกจากนี้ภาษาที่ใช้ยังเป็นมาตราฐานเดียวกันสามารถใช้ได้กับชิพทุกตัวและทุกบริษัท

3. การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นาน เพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของ คอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพขณะที่อยู่ในระบบได้ โดยไม่จำเป็นต้องถอดมาโปรแกรมข้างนอก ดังรูปที่ 2.2 และที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสีย ค่าใช้จ่ายเพิ่มแต่อย่างใด



รูปที่ 2.2 การโปรแกรมลงในชิพ

2.1.4 การออกแบบโดยใช้ภาษาอธิบายพฤติกรรมของฮาร์ดแวร์

ในการออกแบบวงจรดิจิตอลนั้นสามารถทำได้โดยการวาดวงจร (Schematic) หรือใช้ภาษาอธิบายพฤติกรรม (Hardware Description Language) ของฮาร์ดแวร์ ในกรณีของการออก แบบวงจรด้วย ASIC ชนิด Full Custom ผู้ออกแบบจะต้องเขียนวงจรด้วย Schematic จากนั้นจะนำวงจรที่ออกแบบไว้ไปทำการจำลองการทำงาน (Simulate) ซึ่งหากผลออกมาเป็นที่พอใจก็จะต้อง Layout เป็นชั้นสาร และในการออกแบบ ASIC ชนิดนี้ผู้ออกแบบจำเป็นจะต้องทราบถึงเทคโนโลยีที่ใช้ในการสร้างด้วย หลังจากได้ Layout ที่สมบูรณ์แล้วจึงจะส่งไปเข้ากระบวนการสร้างไอซีหรือ Fabrication เพื่อสร้างเป็นชิพไอซีออกมา แต่ในการออกแบบวงจรด้วย FPGA โดยการใช้ Schematic หรือใช้ภาษาอธิบายการทำงานของวงจรจะทำ ได้สะดวกกว่า เนื่องจากวิธีการนี้ผู้ออกแบบไม่จำเป็นต้องคำนึงถึงเทคโนโลยีที่จะใช้สร้างไอซีและที่สำคัญ การออก แบบโดยวิธีนี้สามารถ

แก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่า เพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยีสำหรับภาษาที่ใช้ สำหรับอธิบายพฤติกรรมของฮาร์ดแวร์ที่ใช้กันก็มี VHDL, AHDL และ Verilog เป็นต้น ส่วนรายละเอียด ของขั้นตอนในการออกแบบสามารถอธิบายได้ดังนี้

1. การสังเคราะห์วงจร (Logic Synthesis)

ในขั้นตอนนี้จะใช้ซอฟต์แวร์ในการสังเคราะห์วงจร (Synthesis Tools) ทำการสังเคราะห์พฤติกรรม ของวงจรที่ได้จากการออกแบบด้วย Schematic หรือ VHDL ซึ่งต้องทำการตรวจสอบด้วยว่าซอฟต์แวร์ นั้นสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการหรือไม่ ตัวอย่างเช่น FPGA ของบริษัท XILINX และบริษัท ALTERA จะมีซอฟต์แวร์หลายตัวที่สามารถใช้ได้ เช่น Max Plus II ในขั้นตอนนี้ ซอฟต์แวร์สังเคราะห์วงจรจะทำการแปลงโค้ด VHDL และทำการ Optimize เพื่อให้ได้วงจรตาม เทคโนโลยีที่เลือกใช้ในการสังเคราะห์วงจรนั้นวงจรระดับเกท (Gate Level) จะไม่เหมาะสมกับโครงสร้างที่มีอยู่ในอุปกรณ์ FPGA ดังนั้นในการ Optimize ซอฟต์แวร์สังเคราะห์วงจร จะต้องทำการ Optimize ให้ได้เป็นวงจรที่ประกอบ ด้วยกลุ่มของลอจิกที่เหมาะสมกับอุปกรณ์ FPGA นั้นๆจึงทำให้ผล ที่ได้มีประสิทธิภาพและในขั้นตอนการสังเคราะห์วงจรนี้ ผู้ออกแบบสามารถกำหนดข้อบังคับสำหรับโมเดล แต่ละตัวได้ เช่น ข้อบังคับในเรื่องเวลา (Timing Constraints) หรือข้อบังคับในเรื่องของพื้นที่ (Area) หรือกำหนดชนิดและตำแหน่งของ I/O ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอน Optimize เพื่อให้วงจร ที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการ Optimize คือการเทียบ (Mapping) โมเดลให้เข้ากับ เทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างและสถาปัตยกรรมภายในอุปกรณ์ FPGA เมื่อทำ การสังเคราะห์วงจรเสร็จแล้ว ซอฟแวร์การสังเคราะห์วงจรก็จะมีการรายงานผลว่าโมเดลที่ออกแบบไปนั้น เป็นอย่างไร เช่นมีค่าความหน่วง (Delay) เท่าใด ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้าง เมื่อมาถึงขั้น ตอนนี้ ผู้ออกแบบก็จะทราบว่าโมเดลเป็นไปตามข้อบังคับหรือไม่ ถ้าไม่ก็สังเคราะห์ใหม่จนกว่าจะเป็นไปตาม ที่กำหนด

2. การแบ่งวงจร (Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์ เป็นส่วนย่อยๆ สำหรับลงใน CLBs, IOBs หรือองค์ ประกอบอื่นๆ ภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่จะแยกออกจากกัน มีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้ เพื่อลดความหนาแน่นในตอนทำการเชื่อมต่อสัญญาณ (routing) ในขั้นตอนนี้จะใช้ซอฟต์แวร์ทำโดยซอฟต์แวร์จะเทียบส่วนประกอบของวงจรเช่น เกท (gate), ฟลิป-ฟลอบ (flip-flop) ลงในทรัพยากรต่างๆ ที่มีอยู่ในอุปกรณ์ FPGAหลังจากทำขั้นตอน นี้เสร็จแล้วผู้ออกแบบสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGA ไปเท่าไร ส่วนข้อมูลทางเวลานั้นผู้ออกแบบจะทราบเฉพาะ

ความหน่วงภายในแต่ละส่วนเท่านั้น หรือที่เรียกว่าความ หน่วงลอจิก(logic delay) ส่วนซอฟต์แวร์จะรวมเอาซอฟต์แวร์ย่อยอื่นๆ อีก เพื่อให้การทำ PPR (Partitioning Placement & Routing) เป็นไปอย่างต่อเนื่อง

3. การวางอุปกรณ์ (Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Partitioning) มาแล้วว่าควร จะอยู่ ณ ตำแหน่งไหนในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ดีที่สุด เช่นวงจรส่วนไหนควรอยู่ใกล้กัน เพื่อจะ ได้ค้นหาเส้นทางได้ (route) ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมี ความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือ Router ทำการค้นหาเส้นทางสัญญาณได้ไม่หมด การวางอุปกรณ์ที่ดีควรวางส่วนต่างๆให้อยู่ใกล้กันโดยเฉพาะส่วน ที่มีการเชื่อมต่อสัญญาณด้วยกันนอกจากนั้นการกำหนดตำแหน่งขา I/O (I/O pin) ตามตำแหน่งขา I/O ของ FPGA บนแผ่น PCB ก็จะมีผลโดยตรงเลยคือซอฟต์แวร์จะวาง I/O ลงในตำแหน่งที่ผู้ออกแบบกำหนด ซึ่ง บางครั้งตำแหน่งที่กำหนดไปไม่เหมาะสม ดังนั้นการกำหนดขา I/O ควรกำหนดตำแหน่งให้เหมาะสม หรือ ไม่ก็ให้ซอฟต์แวร์จัดการเอง

4. การเชื่อมต่อสัญญาณ (Routing)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA ขั้นตอนนี้จะทำ ต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีซอฟต์แวร์ก็จะทำการเชื่อมต่อสัญญาณได้ไม่หมด (เนื่องจากจำนวนทรัพยากรสำหรับเชื่อมต่อสัญญาณนั้นมีอยู่จำกัด) หรือเกิดความหน่วงเกิน ค่าที่กำหนดในข้อบังคับ ผู้ออกแบบสามารถทำขั้นตอนนี้ได้โดยใช้ซอฟท์แวร์หรือผู้ออกแบบจะทำการ เชื่อมต่อสัญญาณด้วยตนเองก็ได้ แต่ทางที่ดีควรใช้ซอฟต์แวร์ทำดีกว่า นอกจากนั้นการกำหนดข้อบังคับ ทางเวลา จะช่วยให้ผลที่ได้จากการเชื่อมต่อสัญญาณดีขึ้นได้

5. ความหน่วงด้านเวลา (Delay)

ในการทำ FPGA นั้นความหน่วงที่เกิดขึ้นเป็นความหน่วงที่เกิดจากการวางตำแหน่ง (layout) ของอุปกรณ์ ซึ่งผู้ออกแบบไม่สามารถเข้าไปแก้ไขได้ แต่สามารถทำให้มีความหน่วงน้อยที่สุดได้ สำหรับความหน่วง ที่เกิดขึ้นนั้นแยกได้เป็นสองประเภทคือ ความหน่วงลอจิก (Logic delay) เป็นความหน่วงภายในองค์ประกอบของอุปกรณ์ FPGA เองและความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณ (Routing delay) เป็นความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณระหว่างองค์ประกอบภายในอุปกรณ์ FPGA โดยปกติแล้ว ค่าความหน่วงลอจิกไม่ควรเกิน 50% ของค่าความหน่วงที่ยอมรับได้ เพราะความหน่วงที่เกิดจากการ เชื่อมต่อสัญญาณมักจะมีค่ามากกว่าค่าความหน่วงลอจิก ดังนั้นในการวางอุปกรณ์ และเชื่อมต่อสัญญาณ ผู้ออกแบบควรกำหนดข้อบังคับทางเวลาเพื่อให้ซอฟต์แวร์ได้ทำงานอย่างมีประสิทธิภาพเพิ่มขึ้น และเพื่อให้ได้ผลลัพธ์ที่ดีขึ้นค่าความหน่วงที่ได้

หลังจากการวางอุปกรณ์ และเชื่อมต่อสัญญาณแล้วจะมีค่าความ หน่วงที่ค่อนข้างแน่นอน ซึ่งผู้ออกแบบสามารถทราบได้ว่าโมเดลที่ออกแบบนั้น เป็นไปตามข้อกำหนด หรือไม่

6. การจำลองการทำงานของวงจร (Simulation)

ในขั้นตอนนี้เป็นขั้นตอนที่สำคัญอีกขั้นตอนหนึ่ง เพราะเป็นขั้นตอนที่ผู้ออกแบบตรวจสอบฟังก์ชันการ ทำงานของโมเดลว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหนเพื่อจะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้ จะมีซอฟต์แวร์ที่ใช้สำหรับทำการจำลองการทำงานของวงจรที่ใช้อยู่ เช่น Model Sim ของบริษัท Model Technology หรือ Max Plus II ของบริษัท Altera ในการจำลองการทำงานของวงจร ควรทำทุกครั้งหลังจากที่มีการทำแต่ละขั้นตอนหลักเสร็จแล้ว เพื่อจะได้ทราบว่าข้อผิดพลาดของโมเดล เกิดขึ้นตอนไหน จะได้แก้ไขข้อผิดพลาดตรงขั้นตอนนั้นๆ ได้เลย ไม่ต้องมาคอยตรวจหาขั้นตอนที่ทำให้ เกิดข้อผิดพลาด นั่นคือการทำการจำลองการทำงานของวงจร ต้องทำทั้งหลังการเขียนโค้ด, การสังเคราะห์วงจร และการทำ PPR การจำลองการทำงานของวงจรหลังจากที่เขียนโค้ดเสร็จแล้วนั้น ผู้ออก แบบสามารถทราบได้แค่โมเดลทำงานถูกต้องหรือไม่เท่านั้น (functional test) ยังไม่สามารถตรวจ สอบการทำงานในเชิงเวลาได้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่สังเคราะห์เป็นวงจรแล้ว เพื่อตรวจสอบว่าฟังก์ชันการทำงานยังคงถูกต้องหรือไม่ และค่าความหน่วงที่เกิดขึ้นเป็นไปตาม ข้อบังคับหรือไม่ มีข้อผิดพลาดเกิดขึ้นหรือไม่ถ้ามีจะแก้ไขให้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่ทำการวางอุปกรณ์ การเชื่อมต่อสัญญาณ (post layout simulation) แล้วก็มีความสำคัญเช่นกันเพราะผลที่ได้จากการจำลองการทำงานของวงจรในตอนนี้ จะเป็นผลลัพธ์ของโมเดลเลย ซึ่งผู้ออกแบบนอกจากจะตรวจสอบฟังก์ชันการทำงานแล้วยังต้อง ตรวจสอบคุณสมบัติอื่นๆ เช่น ความหน่วงที่ได้จากการทำ PPR ในรูปแบบค่าความหน่วงมาตรฐาน (Standard Delay Format : SDF) ว่าตรงตามที่กำหนดหรือไม่ หรือตรวจสอบว่าวงจรรวม สามารถใช้งานที่ความถี่สูงสุดเท่าไรนั่นเอง ในการจำลองการทำงาน ของวงจรควรใช้ซอฟต์แวร์ตัวเดียวกันตลอดเพื่อจะได้เปรียบเทียบผลที่ได้จากขั้นตอนต่างๆ

7. การโปรแกรมอุปกรณ์ FPGA (Configuration)

หลังจากที่โมเดลผ่านขั้นตอนต่างๆ จนกระทั่งผ่านการทำ PPR (Partitioning, Placement & Routing) แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด (download) ลงในอุปกรณ์ FPGA ได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้เป็นข้อมูลวงจร (configuration data) ซึ่งอยู่ในรูปของบิตสตรีม (bit stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มี ฟังก์ชันการทำงานตามโมเดลที่ผู้ออกแบบต้องการ ซึ่งในขั้นตอนนี้จะใช้วิธีที่แตกต่างกันออกไปสำหรับอุปกรณ์ FPGA ของแต่ละบริษัทผู้ผลิตคือ ในกรณีที่เป็นอุปกรณ์ FPGA ชนิดที่ต้องโปรแกรมโดยวิธี SRAM นั้น ในการใช้งานผู้ออกแบบจะต้องเก็บข้อมูลวงจรไว้ในหน่วยความจำประเภท EPROM หรือ serial PROM ด้วยเพื่อจะใช้งานสะดวกขึ้น คือในการใช้งานโมเดลครั้งต่อไปไม่ ต้องดาวน์โหลดข้อมูลวงจรจากเครื่องคอมพิวเตอร์อีก เพราะมีข้อมูลวงจรเก็บอยู่ในหน่วยความจำอยู่

แล้ว แต่กรณีที่อุปกรณ์ FPGA เป็นชนิดที่โปรแกรมโดยใช้วิธี EPROM หรือ Anti fuse ก็ไม่จำเป็นต้องมีหน่วยความจำสำหรับเก็บข้อมูลวงจร เพราะว่าอุปกรณ์ FPGA ชนิดนี้เมื่อดาวน์โหลดข้อมูลวงจรลงไป ข้อมูลที่ดาวน์โหลดลงไปก็ยังคงอยู่ในอุปกรณ์ FPGA และครั้งต่อไปก็ใช้งานโมเดลที่ออกแบบไว้ได้เลย

2.1.5 เครื่องมือสำหรับการออกแบบ FPGA

จะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้นทำได้สะดวกกว่า ASIC มากเพราะใช้เวลาน้อยกว่ามากด้วย ส่วน สำคัญที่ใช้ในการทำ FPGA คือ ซอฟต์แวร์ที่ใช้ตั้งแต่เขียนโค้ดอธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลดลงใน อุปกรณ์ FPGA ซึ่งซอฟต์แวร์ที่ใช้ต้องเป็น ซอฟต์แวร์ที่ทำงานต่อเนื่องกันได้ สำหรับซอฟต์แวร์ที่ใช้ทำการ จำลองการทำงานของวงจรนั้น ต้องสามารถใช้งานต่อเนื่องกับซอฟต์แวร์ที่ใช้ทั้งระบบ เพราะโมเดลที่ได้จาก การทำขั้นตอนต่างๆ ด้วยซอฟต์แวร์ ต่างๆ ต้องเอามาจำลองการทำงานได้ และในการจำลองการทำงานของ วงจรควรใช้ซอฟต์แวร์ตัวเดียวกันตลอดทั้งระบบ เพื่อจะได้เปรียบเทียบผลได้ง่าย ในอดีตซอฟต์แวร์ส่วนใหญ่ จะใช้งานอยู่บนคอมพิวเตอร์สมรรถนะสูงอย่างเวิร์คสเตชั่น (Workstation) ในปัจจุบันมีการพัฒนาซอฟต์แวร์ ที่ใช้บนพีซี (PC) มากขึ้นซึ่งสามารถลดค่าใช้จ่ายในด้านอุปกรณ์คอมพิวเตอร์

2.2 เทคโนโลยีการออกแบบวงจรลอจิก (VHDL : Logic Design Technology)

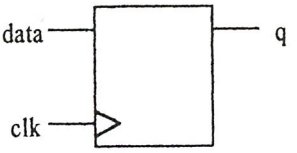
เทคโนโลยีของการออกแบบวงจรลอจิกได้มีการพัฒนาไปอยู่เสมอ จนได้มีการพัฒนาภาษาโปรแกรมที่ใช้ในการออกแบบวงจรลอจิกขึ้นมา VHDL เป็นหนึ่งในนั้นที่ได้มีการพัฒนาขึ้นมา VHDL ย่อมาจาก VHS Hardware Description Language (VHSIC = Very High Speed Integrated Circuit IC) มันก็คือภาษาที่ใช้ในการออกแบบวงจรลอจิกเพื่อผลิตเป็นชิพไอซีขึ้นมา VHDL เป็นภาษาระดับสูงที่ใช้ในการออกแบบระบบและวงจรลอจิกเป็นภาษาที่สามารถออกแบบวงจรได้ในระดับต่างๆ จะประกอบไปด้วยโครงสร้างต่างๆที่ถูกออกแบบขึ้นมา ในระดับที่สูงขึ้นตัวภาษานั้นสามารถที่จะออกแบบระบบโดยไม่คำนึงกระบวนการหรือวิธีการของวงจร เพราะวงจรจะถูกสร้างในรูปแบบของฟังก์ชั่น เราจะพิจารณาเพียงจุดมุ่งหมายของการออกแบบวงจรเท่านั้น เป็นภาษามาตรฐานที่ IEEE ได้ให้การยอมรับ ข้อดีคือเป็นภาษาที่สามารถปรับเปลี่ยนโครงสร้างให้เข้ากับระบบฮาร์ดแวร์ต่างๆได้เป็นอย่างดี และอ่านเข้าใจได้ง่าย

2.2.1 องค์ประกอบที่สำคัญของ VHDL

VHDL มีองค์ประกอบที่สำคัญอยู่ 2 ส่วนคือ Entity และ Architecture

1. Entity เป็นเสมือน Block ที่เราสร้างขึ้นมาเพื่อจะบอกถึงจุดเชื่อมต่อภายนอก (I/O) ว่ามีอะไรบ้าง โดยไม่ต้องมีการอธิบายโครงสร้างภายใน

2 . Architecture ใช้ในการอธิบายโครงสร้างภายในของ Entity



รูปที่ 2.3. แสดง D Filp-Flop

```
entity  dff  is
 port (data,clk:in  std_logic;
     q:out  std_logic);
end  dff;
architecture behav of dff is
 begin
  process  (clk)
   begin
       if (clk'event and clk = '1') then
          q <=  data;
       end  if;
   end  process;
end  behav;
```

โปรแกรมนี้เป็นตัวอย่างของการเขียนโปรแกรมเพื่ออธิบายคุณสมบัติของ D F/F (รูปที่ 2.3) Entity มีการประกาศอินพุทอยู่ 2 ตัวคือ data กับ clk เอาท์พุทคือ q ส่วน Architecture มีการบอกการทำงานของ D F/F ว่าถ้าสัญญาณ clk มีการเปลี่ยนแปลงจาก 0 เป็น 1 ให้มีการส่งค่า data ไป q

2.2.2 รูปแบบของภาษา VHDL

- รูปแบบการเขียน Architecture ของ VHDL

VHDL มีรูปแบบการเขียนอยู่ 3 รูปแบบ

1. Structural Model การอธิบายวงจรโดยใช้โมดูลต่างๆมาเชื่อมต่อกันให้เห็นโครงสร้างภายใน

2. Behavioral Model การอธิบายการทำงานของวงจรในระดับลอจิกเกท

3. Sequential Model การอธิบายขั้นตอนการทำงานโดยใช้ Sequential Statement ทำงานทีละขั้นตอน

ต่อไปนี้จะเป็นตัวอย่างการเขียนโปรแกรมเพื่ออธิบายวงจร RS F/F โดยใช้ Architecture ที่แตกต่างกัน

```
Structural Model        u1: nand2  port  map  (set, q, qb);
                        u2: nand2  port  map  (reset, qb, q);
Behavioral Model        qb<=not (q and set) after 2 ns;
                        q <=not (qb and reset) after 2 ns;
Sequential Model        process (set,reset)
                        begin
                            if  set = '0' and  reset = '1' then
                                q <= '0' after 2 ns;
                                qb <= '1' after 4 ns;
                            if . . .
                            . . .
                        end process;
```

สำหรับรูปแบบการเขียนที่เราจะเลือกใช้นั้น ขึ้นอยู่กับความเหมาะสมของฮาร์ดแวร์ เราสามารถที่จะใช้รูป แบบต่างๆมาร่วมกันได้อยู่ใน Architecture ตัวเดียวกันก็ขึ้นอยู่กับความเหมาะสมเช่นกัน จะเห็นว่าในแต่ละแบบก็มีจุดเด่นของตัวมันเอง แต่ทั้ง 3 แบบก็ให้ผลลัพธ์เหมือนกัน

Sequential มีการอธิบาย Architecture เป็นลำดับขั้นตอนที่ชัดเจน เข้าใจง่าย โดยปกติแล้วโปรแกรมที่อยู่ภายใน Architecture นั้นจะมีการ process พร้อมๆกันทุกบรรทัด (เหมือนวงจรจริงที่มีการเชื่อมต่อสัญญาณกันทั้งวงจร เมื่อมีสัญญาณ input ตัวใดเปลี่ยน output ก็จะมีการเปลี่ยนตามทันที จะมี delay ก็เพียงเล็กน้อย) แต่แบบ Sequential สามารถจะมองเป็นขั้นตอนได้โดยใส่ Process Statement คร่อมเข้าไปในโปรแกรม ส่วน Structural ก็มีการดึง Component NAND มาจะมองเห็นการเชื่อมต่อที่ชัดเจน ส่วน Behavioral นั้นมีการอธิบายคุณสมบัติระดับลอจิกเกท

- Behavioral Modeling

ในหัวข้อที่แล้วได้กล่าวถึง Structural Model ไปบ้างแล้วในหัวข้อนี้เป็น Behavioral Model ซึ่งจะกล่าวถึง Statement และฟังก์ชั่นต่างๆที่นิยมใช้กับ Behavioral Model

1. การใช้ Selected signal assignment และ Conditional signal assignment statement

Statement ที่นิยมใช้อีก 2 แบบสำหรับ Behavioral Model ก็คือ Conditional และ Selected signal assignment โปรแกรมต่อไปนี้จะแสดงตัวอย่างของการใช้ 2 Statement ดังกล่าว

```
with sel select                    sel<=   0 when a = '0' and b = '0' else
  q<=i0 after 10 ns when 0,                1 when a = '1' and b = '0' else
      i1 after 10 ns when 1,               2 when a = '0' and b = '1' else
      i2 after 10 ns when 2,               3 when a = '1' and b = '1' else
      i3 after 10 ns when 3,               4;
     'x' after 10 ns when others;
```

โปรแกรมทางซ้ายคือ Selected signal assignment โดย q จะเลือกส่งค่า i0, i1, i2, i3 ออกไปขึ้นอยู่กับค่าของ sel (sel เป็น integer) ส่วนโปรแกรมทางขวาก็คือ Condition signal assignment โดยตัวแปร sel จะมีค่าเป็น 0, 1, 2, 3 หรือ 4 ขึ้นอยู่กับเงื่อนไขว่าเงื่อนไขใดถูกต้อง

2. การใช้ delay

การใช้ delay ใน VHDL นั้นเป็นการทำงานเลียนแบบการทำงานจริงๆของวงจรที่จะมี delay ในการทำงานอยู่เช่นกัน

```
a <= b;  ( ส่งค่า b ไป a ทันทีโดยไม่มี delay time )
a <= b after 10 ns; ( ส่งค่า b ไป a หลังจากที่ผ่านไปแล้ว 10 ns )
```

VHDL จะมี delay อยู่ 2 แบบ

1. Inertial delay ก่อนจะส่งค่าสัญญาณทางขวาไปยังทางซ้าย โดยสัญญาณทางขวาจะต้องคงค่านั้นได้นานเท่ากับค่า delay ดังนั้น delay แบบนี้สัญญาณของตัวส่งกับตัวรับไม่จำเป็นจะต้องเหมือนกัน ดังตัวอย่างการใช้งาน

```
a <= b after 20 ns;  ( a จะได้รับ ค่าจาก b เมื่อค่าของ b ไม่เกิดการเปลี่ยนแปลงเป็นเวลา 20 ns )
```

2. Transport delay เป็นการส่งค่าสัญญาณทางขวาไปยังทางซ้าย โดยสัญญาณของทั้งตัวส่งและตัวรับจะเหมือนกัน แตกต่างกันก็ตรงที่สัญญาณตัวรับจะช้ากว่าเป็นเวลาเท่ากับที่ delay ไว้ ดังตัวอย่างการใช้งาน

```
a <= transport b after 20 ns; (สัญญาณของ b จะเหมือนกับ a  แต่ b จะมีสัญญาณที่ช้ากว่า a อยู่ 20 ns)
```

- การใช้งาน Generics

Generics เป็นเครื่องมือที่ใช้ในการส่งข้อมูลต่างๆให้ Entity เช่น ค่า delay time,ค่าความจุ, ค่าความต้านทาน, ความกว้างของ data-path หรือว่า ความกว้างของ signal เป็นต้น ใช้งานได้โดยเราจะเพิ่มเติมส่วน Generic นี้แทรกเข้าไปใน Entity Block ดังตัวอย่างโปรแกรมดังนี้

```
entity buff is                          architecture buff1 of buff is
  generic(delay:time;load:integer);       begin
  port(a:in bit; b:out bit);                b<= a after(delay * load);
end buff;                               end buff1;
```

จากตัวอย่างโปรแกรมเป็น buffer ที่มีการส่งค่า delay และ load ให้กับตัว buffer โดยจะนำค่าเหล่านั้นไปใช้ใน architecture ดังตัวอย่าง ภายใน Architecture จะมีการส่งค่าจาก a ไปยัง b โดยมี delay หน่วงไว้โดยค่า delay นั้นเท่ากับ (delay * load) ซึ่งหลังจากที่เราได้เขียน entity buffer ตัวนี้ขึ้นมาแล้วก็สามารถจะนำไปใช้กับ entity ตัวอื่นได้เสมือนการเรียกใช้งาน function โดยเราจะใช้ entity buffer ตัวนี้ใน entity ตัวใดเราก็จะแทรก Component ไว้ใน Architecture ตัวนั้นโดยแทรกไว้ก่อน begin ดังตัวอย่าง

```
architecture test1 of test is
component buff
        generic(delay : time; load : integer);
        port( a : in bit; b : out bit);
end component;
begin
```

เมื่อเราแทรก entity ตัวนั้นลงไปแล้วเราก็สามารถเรียกใช้ได้เสมือนเป็นฟังก์ชั่นตัวหนึ่ง และมีการส่งค่าให้ได้ด้วยดังนี้

u1: buff generic map (13 ns, 2) port map (ina, outb); (เป็นการเรียกใช้ในรูป-แบบ Structure)

- การใช้งาน Block Statements

เป็น Statement ที่เข้ามาช่วยให้ Architecture ของเรานั้นดูเป็นระบบ และสัดส่วนมากขึ้น อีกทั้งยังทำให้เราสามารถประกาศตัวแปรแบบ Local ขึ้นมาใช้ได้ ทำให้ตัวแปรมีการแยกแยะใช้

เป็นสัดส่วน ตัวแปรที่ประกาศไว้ใน Block ก็สามารถใช้ครอบคลุมได้ภายใน Block เท่านั้น (เหมือนในภาษาปาสคาล)

```
[label] : block [ ( condition )]
        [ประกาศตัวแปร]
begin
        -- statements
end block [label];
```

ในแต่ละ Architecture สามารถที่จะมีได้หลาย block ขึ้นอยู่กับผู้เขียนว่าจะแบ่งเป็นสัดส่วนเช่นใด และในแต่ละ block ก็สามารถมี block ซ้อนอยู่ข้างในได้ด้วย โดย condition ที่อยู่หลัง block นั้นคือคุณสมบัติของ Guarded Blocks ที่เพิ่มเติมเข้ามาเพื่อนำเงื่อนไขดังกล่าวไปใช้ใน statement ดูได้จากตัวอย่าง

```
architecture test1 of test is
begin
   g1 : block (clk = '1')
   begin
     q <= guarded d after 5 ns;
   end block g1;
end latch_guard;
```

การทำงานของโปรแกรมก็คือ จะเห็นว่าจะมี keyword GUARDED เพิ่มเข้ามาซึ่ง GUARDED ตัวนี้จะให้ค่าเป็นจริงเมื่อ clk = '1' นอกจากนี้จะเป็นเท็จ ดังนั้น statement d ที่ส่งค่าให้ q จะทำงานก็ต่อเมื่อ Guarded เป็นจริง

2.2.3 Sequential Processing

สำหรับการออกแบบวงจรที่มีความขนาดใหญ่และซับซ้อนนั้น รูปแบบของ Structure และ Behavioral Model นั้นยังไม่เพียงพอ รูปแบบของ Sequential เป็นรูปแบบที่มีความสำคัญมากในการใช้ เพราะมีรูปแบบอย่างเดียวกันกับภาษาระดับสูงอย่าง C หรือ ปาสคาลที่มีการทำงานแบบ Sequential Statement

- Process Statement

การที่จะทำให้ statement ใน VHDL นั้นสามารถทำงานแบบ Sequential ได้นั้นเราจะต้องใช้ statement ที่เรียกว่า Process เข้าไปช่วย ดังตัวอย่างการใช้งาน

```
architecture test1 of test is
begin
  process (a, b)
    variable temp : std_logic;
  begin
    -- sequential statement;
  end process;
    -- statement;
end test1;
```

จากตัวอย่างการใช้งานจะเห็นว่า Process ที่แทรกเข้าไปใน Architecture นั้นจะทำให้ในส่วน statement ที่ถูกคั่นด้วย begin และ end process นั้นจะมีการทำงานเป็น step ที่ละบรรทัดทันที ซึ่งก็คือรูปแบบของ Sequential Processing ส่วน statement ที่อยู่ใน architecture เดียวกันแต่อยู่นอก block process ก็ยังคงมีการทำงานแบบปกติก็คือ process พร้อมกันทุกบรรทัด นอกจากนี้จากโปรแกรมจะว่าข้างหลัง process ยังมี (a,b) ใส่ไว้ ในวงเล็บนั้นก็คือสัญญาณที่ใช้ใน architecture ตัว architecture จะเข้าไปทำใน block process นี้ก็ต่อเมื่อสัญญาณ a หรือ b มีการเปลี่ยนแปลงเท่านั้น หรือที่เรียกว่า Event ทำให้ในการ simulate นั้นโปรแกรมไม่ต้องเข้าไปทำงานใน process ตลอดเวลาโดยไม่จำเป็น

- Sequential Statements

นอกจาก process statement แล้วก็ยังมี statement ที่จำเป็นอยู่อีกมาก VHDL มี Sequential Statements ให้ใช้งานอยู่ 5 ตัวดังนี้ if, case, loop, assert, wait ให้เราได้เลือกใช้งานในงานต่างๆ มีรายละเอียดและรูปแบบดังต่อไปนี้

รูปแบบ IF statement

```
IF condition THEN
    Sequence Statements
[ELSEIF condition THEN Sequence Statements ]
```

```
[ELSE  Sequence Statements ]
END IF;
```

if statement ใช้ในการตรวจสอบเงื่อนไขถ้า condition เป็นจริงก็จะทำใน sequential statement ของ then ถ้าไม่ก็จะไปทำใน sequential statement ของ else แทน ยกตัวอย่างการใช้งานง่ายๆ ถ้าเรามีเงื่อนไข ถ้า a = '1' ก็จะส่งค่า c ให้ b นอกจากนี้ก็จะส่งค่า d ให้ b เขียนเป็นโปรแกรมได้ดังนี้

```
if ( a = '1')  then b <= c;  else b <= d;  end if;
```

รูปแบบ Case Statements

```
Case expression IS
   WHEN choices [| choices ] => sequence statements
   WHEN choices [| choices ] => sequence statements
   . . .
END CASE;
```

case statement ใช้ในการตรวจตรวจสอบว่า expression (ตัวแปรที่จะนำมาเปรียบเทียบ) นั้นมีค่าเท่ากับตัวเลือกใด โปรแกรมก็จะเข้าไปทำ statement ในตัวเลือกนั้น แตกต่างกับ if statement ตรงที่ case นั้นจะใช้กับกรณีที่เงื่อนไขนั้นเป็นการเปรียบเทียบกับตัวแปรตัวเดียวกัน และมีหลายเงื่อนไข

รูปแบบของ LOOP Statements

VHDL มี loop ให้ใช้งานอยู่ 2 แบบ คือ while และ for การใช้ loop นั้นมีประโยชน์มาก สำหรับข้อมูลที่เรามีหลายชุดเราไม่จำเป็นต้องเขียนโปรแกรมซ้ำๆกัน มีรูปแบบของการใช้งานดังนี้

```
WHILE  condition LOOP
   Sequential Statements;
END LOOP
```

```
FOR  identifier IN start_value TO finish_value LOOP
      Sequential Statements;
END LOOP;
```

ใน loop statement ยังมี statement อีก 2 ตัวที่ช่วยให้การทำงานของ loop นั้นมีความยืดหยุ่นขึ้นคือ next และ exit statement เมื่อเราเพิ่ม NEXT; เข้าไปใน loop เมื่อโปรแกรมได้ทำงานมาถึง statement ที่มี NEXT อยู่ โปรแกรมก็จะกระโดดกลับไปยังส่วนหัวของ loop ทันทีเพื่อทำงานใน

step ถัดมาต่อ สำหรับ exit statement เมื่อเราเพิ่ม EXIT; เข้าไปเมื่อโปรแกรมทำงานมาเจอ EXIT แล้วก็จะทำให้หลุดออกจาก LOOP ทันทีไปทำงานนอก loop ต่อไป

2.2.4 Wait Statements

Wait statement คือ การหน่วงเวลาใน sequential statement ที่จะทำให้โปรแกรมนั้นหยุดทำงานอยู่ใน statement นั้นเป็นเวลาตามที่กำหนดไว้ หรือตามเงื่อนไขที่ตั้งไว้ มีรูปแบบการหน่วงให้ใช้อยู่ 4 แบบคือ wait on signal, wait until condition, wait for time_expression, และ multiple wait condition Statement เหล่านี้เป็น statement ที่มีประโยชน์มากอีก statement หนึ่งมีการทำงานของแต่ละแบบดังนี้

- Wait on signal

เมื่อโปรแกรมทำงานมาถึง statement นี้จะเป็นการหน่วงไว้กับที่ไปจนกว่าจะมีการเปลี่ยนแปลงของ signal ที่กำหนดให้ เช่น Wait on a,b; คือจะรออยู่กับที่จนกว่าค่าของ a หรือ b จะมีการเปลี่ยนแปลง

- Wait until condition

เมื่อโปรแกรมทำงานมาถึง statement นี้จะเป็นการหน่วงไว้กับที่ไปจนกว่าเงื่อนไขนั้นๆจะเป็นจริงก็จะหลุดจากการหน่วง เช่น Wait until ((a='1') and (b='1')); คือจะหน่วงไปเรื่อยๆ จนกว่า
a และ b จะมีค่า '1' ก็จะหลุดจากการหน่วง

- Wait for time_expression

เป็นการหน่วงตามเวลาที่เรากำหนดไว้ เช่น Wait for 10 ns; คือเป็นการหน่วงไว้เป็นเวลา 10 ns;

- Multiple Wait condition

เป็นการใช้การหน่วงทั้ง 3 แบบมาร่วมกันเช่นดังตัวอย่าง

Wait on a, b until c='1' for 2 usec;

คือจะหน่วงไว้จนกว่า a หรือ b จะมีการเปลี่ยนแปลงค่า และ c จะต้องเป็น 1 ถ้าไม่มีเงื่อนไขแบบนี้เกิดขึ้นก็จะหน่วงไว้นานสุดที่ 2 usec โดยการใช้ Multiple Wait นี้มีเงื่อนไขการใช้อยู่ด้วย คือสามารถจะนำมา wait แบบต่างๆมาใช้ร่วมกันได้ 4 แบบ โดยเวลาใช้ก็ต้องมีเรียงลำดับด้วยดังนี้
1) on until for 2) on until 3) on for 4) until for

2.2.5 ชนิดของข้อมูล (Data types)

- Object Types

VHDL ก็มีคุณสมบัติต่างๆเช่นเดียวกับภาษาระดับสูง มีการแบ่งชนิดของตัวแปรให้ใช้ในภาษาอยู่ด้วย 3 ชนิดด้วยกันคือ

1 Signal ใช้เป็นสัญญาณเพื่อแสดงการเชื่อมต่อกันระหว่าง Component ต่างๆ

2 Variable ใช้เป็น Temp เก็บข้อมูลชั่วคราวภายใน entity โดยใช้ในการ process

3 Constant ชื่อที่มีการกำหนดค่าคงที่ให้ ไว้ใช้ใน program

signal มีรูปแบบการใช้งานดังนี้

```
SIGNAL signal_name : signal_type [:= initial_value];
```

การใช้งาน Signal นอกจากจะเป็นการประกาศชนิดของสัญญาณแล้ว ยังสามารถจะกำหนดค่าเริ่มต้นให้กับสัญญาณนั้นได้ด้วยดังตัวอย่างเป็นการประกาศสัญญาณ vcc ให้มีชนิดเป็น std_logic และมีค่าเริ่มต้นเป็น 1

```
SIGNAL  vcc : std_logic := '1';
```

(* std_logic คือ ชนิดของตัวแปรอีกชนิดหนึ่งมีด้วยกันทั้งหมด 9 ค่าคือ U, X, 0, 1, Z, W, L, H, -)

Variables มีรูปแบบการใช้งานดังนี้

```
VARIABLE variable_name [,variable_name] : variable_type [:= value];
```

การใช้งาน Variables ก็มีลักษณะคล้ายกับ signal คือสามารถกำหนดค่าเริ่มต้นให้กับตัวแปรได้ด้วย ดังตัวอย่าง กำหนดให้ delay เป็น time มีค่าเริ่มต้นที่ 2 ns

```
VARIABLE  delay : time := 2 ns;
```

Constants มีรูปแบบการใช้งานดังนี้

```
CONSTANT constant_name [,constant_name] : type_name [:= value];
```

การใช้งาน Constant คือจะต้องมีการกำหนดค่าเริ่มต้นให้กับตัวแปรเพื่อเป็นค่าคงที่นำไปใช้ในโปรแกรมได้ ดังตัวอย่างเป็นการกำหนดตัวแปร pi มีชนิดเป็น real มีค่าคงที่เท่ากับ 3.1414

```
CONSTANT pi : real := 3.1414;
```

- ชนิดของข้อมูล (Data types)

ชนิดของตัวแปรที่เราใช้ประกาศใน Signal, Variable, หรือ Constant แบ่งออกมาได้เป็น 4 ประเภทใหญ่ๆ คือ Scalar, Composite, Access และ File Types

Scalar Types นั้นเป็นตัวแปรแบบพื้นฐานที่ใช้กันบ่อยเช่น integer, real เป็นต้น Composite เป็นชนิดที่ใช้สร้าง array และ record ส่วน Access types เป็นตัวแปรรูปแบบของ Pointer เหมือนกับภาษาโปรแกรมทั่วไป File เป็นตัวแปรที่เกี่ยวกับไฟล์

1. Scalar Types

Scalar Types แบ่งย่อยออกเป็น 4 ชนิดคือ Integer, Real, Enumerated, และ Physical types การใช้งานตัวแปรเหล่านี้จะต้องใส่ค่าให้ถูกต้องต้องใส่ค่าให้ถูกชนิดด้วยเช่น Integer ห้ามเอาค่า Real ใส่ให้เป็นต้น

Integer อ้างถึงข้อมูลตัวเลขจำนวนเต็มในช่วง -2,147,483,647 ถึง +2,147,483,647

Real อ้างถึงข้อมูลตัวเลขจำนวนจริงที่เป็นทศนิยมในช่วง -1.0E+38 ถึง +1.0E+38

Enumerated เป็นชนิดที่มีประสิทธิภาพมากอีกชนิดหนึ่ง เพราะเป็นชนิดที่มีการกำหนดขอบเขต หรือว่า set ของกลุ่มข้อมูลที่จะอ้างถึงขึ้นมาได้เองโดย Programmer ยกตัวอย่างถ้าเราต้องการสร้าง type ข้อมูลที่อ้างถึงข้อมูลได้ 3 ตัวคือ red, green และ yellow เพื่อใช้กับโปรแกรมไฟจราจรเพื่อทำให้มองโปรแกรมได้ง่าย

```
TYPE t_state IS ( red, green, yellow );
```

Physical เป็นการกำหนดตัวแปรที่ประกอบด้วย 2 ส่วน ส่วนแรกกำหนดขอบเขตของตัวแปร ส่วนที่สองกำหนดหน่วยขึ้นมาใช้งาน ดังตัวอย่างจะเป็นการกำหนดชนิดข้อมูลที่ชื่อว่า current มีขอบเขตที่ 0 ถึง 1000000000 และมีหน่วยให้ใช้งานอยู่ 3 หน่วยคือ na, ua และ ma ตัวแปรแบบนี้เหมาะที่จะใช้กับข้อมูลที่มีความละเอียดสูง มีตัวเลขอยู่หลายหลัก เมื่อมาใช้หน่วยช่วยจะทำให้อ้างถึงข้อมูลเหล่านั้นได้ง่าย และ สั้นกว่า

```
Type  current is range 0 to 1000000000
 units
        na;
        ua = 1000 na;
        ma = 1000 ua;
end units;
```

2. Composite Types

เป็นชนิดที่มีการเก็บข้อมูลที่มีลักษณะเป็นกลุ่มจึงแยกย่อยได้เป็น 2 ชนิดคือ array และ record สำหรับข้อมูลชนิด array แล้วช่วยได้มากในการทำ ROM หรือ RAM

Array รูปแบบของ array มิติเดียว

TYPE name IS Array (start TO stop) OF ชนิดของตัวแปร

ตัวอย่างการประกาศ Array มิติเดียว data_bus เป็น type ที่มีขนาด 4 bit มีชนิด signal เป็น bit

```
TYPE data_bus IS Array ( 0 TO 3 ) OF bit;
```

ตัวอย่างการประกาศตัวแปรให้ x มีชนิดเป็น data_bus

```
VARIABLE x : data_bus;
```

ตัวอย่างการเข้าถึง Array แบบทีละตำแหน่ง `x(1) := '1';`

ตัวอย่างการเข้าถึง Array แบบเป็นกลุ่ม `x := ('1', '0', '1', '0');`

Multidimensional Arrays เป็นการใช้ array หลายมิติ Array 2 มิตินิยมใช้ในการออกแบบ rom และ ram

ตัวอย่างการประกาศ Array หลายมิติ

```
TYPE mem_data IS Array (0 TO 32, 0 TO 32 ) OF std_logic;
```

ตัวอย่างการเข้าถึง Array หลายมิติ `X(1,1) := '1';`

Record

Record คือการจะมองกลุ่มของข้อมูลชนิดต่างๆ เข้าด้วยกันเป็น Object เดียว ดังนั้น Record จะประกอบไปด้วยข้อมูลชนิดต่างๆอยู่ เราสามารถที่จะใช้ Record ซ้อน Record ได้เหมือนกับ Programming Language ทั่วๆไป

ตัวอย่างการประกาศ

```
type test_record is
    record
        d : integer;
        o : real;
    end record;
variable test : test_record;
```

เราสามารถอ้างถึงข้อมูลได้โดยใส่จุดไว้ข้างหลังตัวแปร และตามด้วยชื่อฟิลด์ใน record เช่น

```
test.d := 5;
test.o := 10.5;
```

3. Access Types

Access มีรูปแบบเหมือนกับ pointer ในภาษาปาสคาลสามารถนำ Pointer นั้นไป Link เชื่อมต่อกันเป็น Link List ได้ด้วย มี Function ที่ใช้กับ Access Types นี้อยู่ 2 ตัวคือ 1. NEW ใช้ในการ Allocate ตำแหน่งใน Memory 2. DEALLOCATE ใช้ในการยกเลิกการใช้งานของ Pointer

ตัวอย่างการใช้

```
TYPE  fifo  IS ARRAY (0 TO 3) OF std_logic;
 TYPE fifo_access IS ACCESS fifo;
 VARIABLE  fifo_ptr : fifo_access := NULL;
 fifo_ptr := new fifo;
 fifo_ptr.ALL := ('0','0','1','1');
```

4. File Types

เป็นข้อมูลที่ประกาศขึ้นเพื่อใช้ในการติดต่อกับไฟล์

รูปแบบการประกาศ

```
TYPE file_ name IS FILE OF file_ type;
```

file มี Function ให้ใช้งานอยู่ 3 Function

READ (file,data) (file คือชื่อ fileที่จะอ่านข้อมูล ; data คือตัวแปรที่จะรับค่าในไฟล์กลับ)

WRITE (file,data) (file คือชื่อ fileที่จะอ่านข้อมูล ; data คือตัวแปรที่เก็บค่าที่จะเขียนลงไปในไฟล์)

ENDFILE (file) (file คือชื่อ fileที่ปิด)

5.Subtypes

เป็นการนำชนิดของข้อมูลที่มีอยู่ดึงมาใช้เพียงบางส่วน มีข้อดีคือ ไม่ต้องกำหนด type ขึ้นมาใหม่, ใช้ขอบเขตของข้อมูลเท่าที่จำเป็น, เข้าใจขอบเขตการใช้งานได้ง่าย เช่นชนิดจำนวนเต็มบวก natural จาก type integer


```
SUBTYPE  natural  IS  integer  RANGE  0  to  +2,147,483,647;
```

2.2.6 เทคนิคการใช้งาน VHDL

- การใช้งาน signal ชนิด std_logic

Std_logic เป็นสัญญาณอีกชนิดหนึ่งที่นิยมใช้ซึ่งมีอยู่ด้วยกันทั้งหมด 9 ค่าคือ (U, X, 0, 1, Z, W, L, H, -) ดังนั้นในการใช้งานเราต้องคำนึงอยู่เสมอว่าไม่ได้มีเพียงค่า 1 กับ 0 ดังนั้นในโปรแกรมของเรานั้นจะต้องมีการใช้ other อยู่เสมอ (others เป็น keyword ตัวหนึ่งที่จะให้ค่าที่เหลือที่เราไม่ได้อ้างถึง) ดังตัวอย่าง กำหนดให้ s เป็น array ขนาด 2 bit ชนิด std_logic

```
case  s  is
    when "00" => muxout <= c;
    when "01" => muxout <= d;
    when "10" => muxout <= e;
    when others  => muxout <= f;
end case;
```

จากโปรแกรมถ้าค่า s ไม่ได้เท่ากับ "00" หรือ "01" หรือ "10" แล้ว โปรแกรมจะส่งค่า f ให้ muxout ทันที

นอกจากการนำ others มาช่วยแล้ว เรายังสามารถใช้ don't care เข้ามาช่วยได้อีกด้วยในกรณีที่เราสนใจข้อมูลเฉพาะบางบิท เช่น ถ้าเรามีข้อมูลอยู่ 6 บิท แต่ต้องการตรวจสอบเฉพาะ 2 บิท หลังว่าเป็น 1 ทั้งคู่หรือไม่ ก็จะต้องนำข้อมูลนั้นไปเปรียบเทียบกับ "----11" เครื่องหมายลบแทน don't care การใช้งานอย่างนี้จะมีประโยชน์มากในการเปรียบเทียบเพราะเราไม่จำเป็นต้องแยกออกมาเปรียบเทียบกันทีละบิท

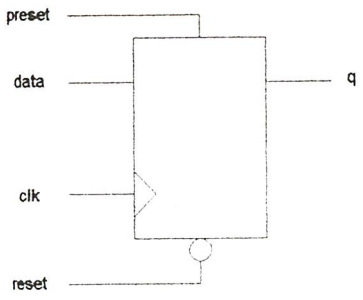
- การใช้งาน VHDL กับ Flip-Flops

Flip-Flop จะทำงานได้นั้นจะต้องใช้ clk เข้ามาช่วยซึ่งการทำงานของ flip-flop นั้นอาศัยการทำงานของขอบขาขึ้น หรือ ขอบขาลงของ clk สามารถประยุกต์ใช้ได้ดังนี้

(clk'event and clk = '1') แทนขอบขาขึ้นของ clk

(clk'event and clk = '0') แทนขอบขาลงของ clk

ยกตัวอย่างการออกแบบ D Flip-Flop ที่มี preset และ reset

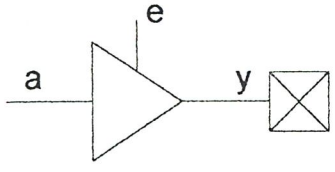


รูปที่ 2.4. แสดง D Flip-Flop มี preset และ reset

```
entity dff_async is     ( entity block )
  port (data, clk, reset, preset : in std_logic;
     q : out std_logic);
  end dff_async;
  architecture behav of dff_async is
  begin
  process (clk, reset, preset ) begin
    if (reset ='0') begin              ( reset ค่า q เมื่อ reset เป็น '0' )
          q <= '0';
    elsif (preset ='1') then                ( set ค่า q เมื่อ preset เป็น '1' )
          q <= '1';
    elsif(clk'event and clk='1') then  (ส่ง data เมื่อมี clk เข้ามา )
          q <= data;
        end if;
  end process;
  end behav;
```

- **การออกแบบ buffer แบบต่างๆ**

1. Tri-State



รูปที่ 2.5 แสดง Tri-State

การออกแบบ buffer ที่มี 3 สถานะโดยมี enable เป็นตัวควบคุมสามารถเขียนเป็น entity ได้ดังนี้

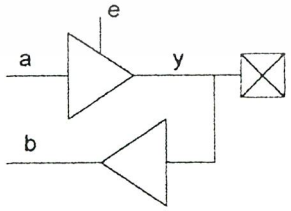
```
entity  tristate  is
port  (e, a : in std_logic;
       y : out std_logic);
end tristate;
```

Architecture นั้นสามารถเขียนได้หลายแบบดังตัวอย่างทั้ง 2 แบบต่อไปนี้

```
architecture tri of tristate is
 begin
  process (e, a)
    begin
      if  e = '1' then
       y <= a;
      else
       y<= 'z';
      end if;
   end process;
end tri;
```

```
architecture tri of tristate is
 begin
    y<=a when(e= '1') else 'z'
 end tri;
```

2. Bi-Directional Buffer



รูปที่ 2.6. แสดง Bi-Directional Buffer

เป็น buffer 2 ทิศทางสามารถเขียนเป็นโปรแกรมส่วน Architecture ได้ดังนี้

```
architecture bi of bidir is
begin
 process (e, a)
   begin
        case c is
          when '1' => y <= a;
          when '0' => y <= 'z';
          when  other => y <= 'x';
        end case;
 end process;
 b <= y;
end bi;
```

2.2.7. ข้อดีของ VHDL

1. รูปแบบของภาษาที่เข้าใจได้ง่าย
2. มีโครงสร้างของภาษาที่สามารถปรับเปลี่ยนให้เข้ากับ hardware ได้ง่าย
3. มี statement ให้ใช้งานอยู่หลายตัว
4. สามารถออกแบบในรูปแบบ top-down design ได้
5. มี library ต่างๆให้เลือกใช้งานได้ทำให้ออกแบบได้ง่ายขึ้น เป็นประโยชน์ต่อ programmer

2.2.8 VHDL ช่วยในการออกแบบได้อย่างไรบ้าง

1. การออกแบบวงจรสามารถทำได้โดยง่าย
2. เมื่อมีการแก้ไข หรือ เปลี่ยนแปลงวงจรสามารถทำได้โดยง่าย เพียงโปรแกรมวงจรใหม่ลงไป ไม่ต้องมีการเปลี่ยนแปลงตัว hardware
3. การทดสอบไม่จำเป็นต้องทดลองกับวงจรจริง สามารถใช้การ simulate ทดลองดูผลลัพธ์ของวงจรที่เราออกแบบได้ว่าถูกต้องหรือไม่

บทที่ 3
หลักการทำงานและการออกแบบ

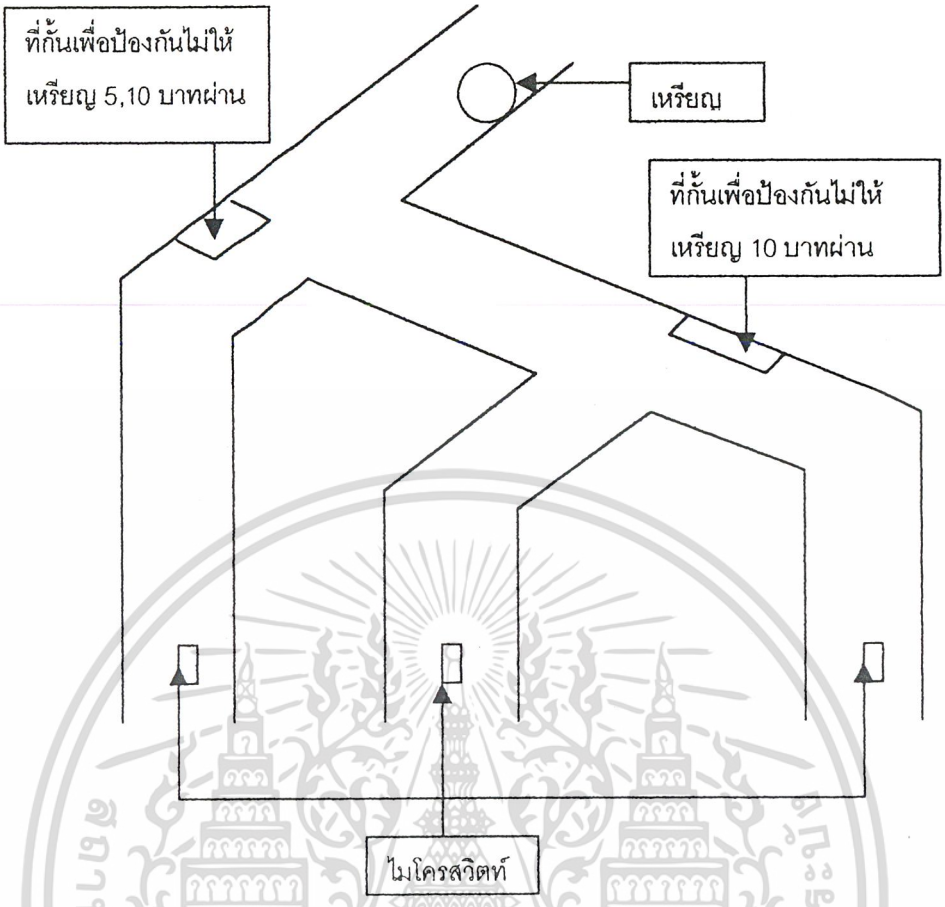
3.1 หลักการของเครื่องนับแยกและทอนเหรียญอัตโนมัติโดยรวม

เมื่อเราทำการหยอดเหรียญลงไปในชุดแยกเหรียญ ชุดแยกเหรียญก็จะทำการแยกเหรียญโดยอาศัยที่ความแตกต่างของขนาดเหรียญ และเมื่อเหรียญที่ถูกแยกแล้ว เหรียญก็จะไหลไปตามรางที่ได้ทำการแยกไว้ ซึ่งจะมีด้วยกันสามขนาด คือ รางของเหรียญ 1 บาท 5 บาท และ 10 บาท ซึ่งรางแต่ละราง จะมีไมโครสวิตซ์ติดอยู่ ซึ่งจะทำหน้าที่ในการส่งสัญญาณลอจิก 0 ไปยังชุดประมวลผลซึ่งใช้อุปกรณ์ FPGA เป็นตัวประมวลผล และจะแสดงผลออกทาง 7 Segment ซึ่งจะแสดงผลเป็นจำนวนเงินรวมที่รับเข้ามา จำนวนเงินรวมที่ต้องทอนและจำนวนเหรียญ 1,5,10 บาท ที่ต้องใช้ทอน ซึ่งในส่วนของจำนวนเหรียญที่ต้องใช้ทอน เราจะใช้การแสดงผลทาง LED แทนการทอนเหรียญจริง เมื่อเรากดสวิตซ์เงินทอน ก็จะมีการแสดงผลออกทาง LED ซึ่ง LED จะสว่างแล้วดับตามจำนวนเหรียญที่ต้องใช้ทอนของแต่ละเหรียญ

3.2 หลักการทำงานของแต่ละส่วน มีดังนี้คือ

3.2.1 ชุดรับเหรียญและแยกเหรียญ

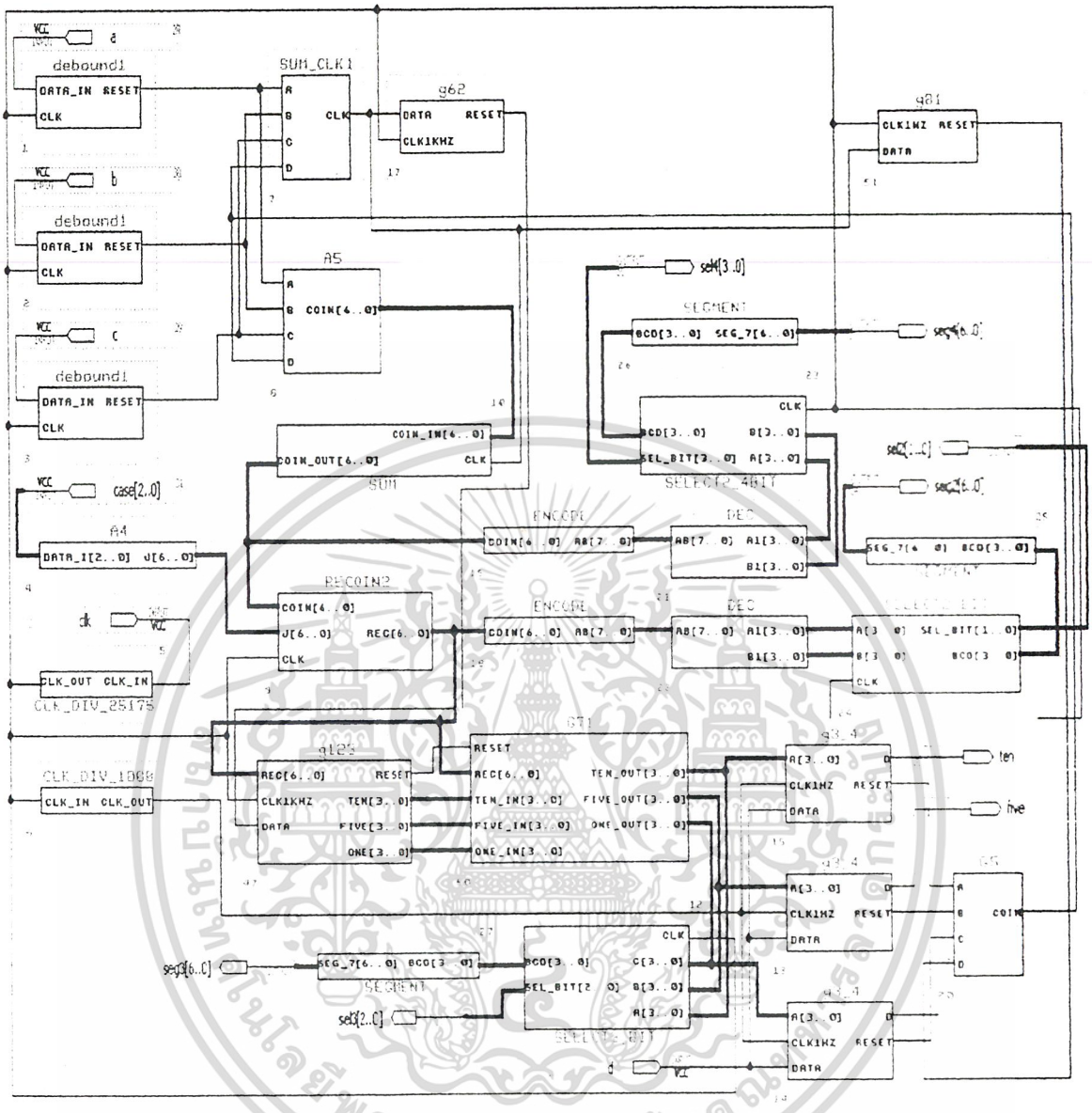
เป็นชุดที่ใช้สำหรับรับเหรียญและทำการแยกเหรียญ เมื่อมีการหยอดเหรียญเข้ามา เหรียญจะถูกแยกให้ไหลไปตามรางของมันเอง โดยจะเป็นการแยกที่ขนาดของเหรียญ และเมื่อเหรียญไหลไปตามรางของมันแล้ว ในแต่ละรางจะมีไมโครสวิตซ์ติดอยู่ เพื่อทำการส่งสัญญาณลอจิก 0 ไปยังส่วนของการประมวลผล เพื่อทำการตรวจสอบว่า เหรียญที่หยอดเข้ามาเป็นเหรียญ 1บาท 5 บาท หรือ 10 บาท แล้วคิดจำนวนเงินรวม และเงินทอนออกมา ซึ่งจะแสดงชุดรับเหรียญและแยกเหรียญได้ดังรูป



รูปที่ 3.1 แสดงชุดรับเหรียญและแยกเหรียญ

3.2.2 ชุดประมวลผล

ในส่วนนี้จะใช้ ชิพ FLEX 10K ซึ่งอยู่ในบอร์ด MASTER FLEX-A01 เป็นตัวประมวลผล ซึ่งภายในชิพ FLEX 10K จะถูกโปรแกรมให้ทำการตรวจสอบว่า ชุดแยกเหรียญได้ทำการส่งสัญญาณของเหรียญใดมา ถ้าเป็นเหรียญบาทก็จะทำการบวกเพิ่มจากค่าเดิมอีกหนึ่ง ถ้าเป็นเหรียญห้าบาทก็จะทำการบวกเพิ่มจากค่าเดิมอีกห้า และถ้าเป็นเหรียญสิบบาทก็จะทำการบวกเพิ่มจากค่าเดิมอีกสิบ ในขณะเดียวกันจะทำการคิดจำนวนเงินทอนและจำนวนเหรียญที่ต้องใช้ทอนด้วย ซึ่งเราจะมีการป้อนค่าเงินจำนวนหนึ่งเข้าไป เพื่อทำการคิดจำนวนเงินทอนออกมา โดยจะใช้ดิฟสวิทช์ซึ่งมีอยู่ในบอร์ด MASTER FLEX-A01 เป็นการเซ็ทค่าจำนวนเงินขึ้นมา เพื่อให้โปรแกรมทำการประมวลผลดังที่ได้กล่าวมาข้างต้น ซึ่งโปรแกรมที่ใช้สามารถแสดงเป็นบล็อกการทำงานได้ดังรูป



รูปที่ 3.2 บล็อกไดอะแกรมแสดงการทำงาน

จากบล็อกไดอะแกรมสามารถอธิบายการทำงานในแต่ละส่วนได้ดังนี้

Clk จะรับสัญญาณนาฬิกาความถี่ 25.175 MHz ที่ผลิตมาจากออสซิเลเตอร์เพื่อนำมาใช้ในการทำงานของอุปกรณ์ FPGA

CLK_DIV_25175 ทำหน้าที่หารความถี่ที่รับมาจาก Clk ให้มีค่าเหลือ 1KHz

CLK_DIV_1000 ทำหน้าที่หารความถี่ที่รับมาจาก CLK_DIV_25175 ให้มีค่าเหลือ 1Hz

DEBOUND1 ทำหน้าที่แก้ไขความผิดพลาดที่เกิดขึ้นเนื่องจากการเชื่อมต่อกันไม่สนิทของหน้าสัมผัสสวิทซ์

A4 ทำหน้าที่กำหนดจำนวนเงินขึ้นมาซึ่งมีอยู่ 3 ค่าคือ 38 บาท 45 บาท 64 บาท

SUM_CLK1 ทำหน้าที่กำหนดจังหวะการทำงานของบล็อก SUM เนื่องจากการทำงานของบล็อก A5 มีความผิดเพี้ยน

A5 ทำหน้าที่รวมจำนวนเงินที่รับเข้ามาจากการหยอดเหรียญ

SUM ทำหน้าที่ปล่อยสัญญาณเอาท์พุทที่รับมาจาก A5 ตามจังหวะการทำงานของ SUM_CLK1

RECOIN2 ทำหน้าที่คิดจำนวนเงินทอน

ENCODE และ DECODE ทำหน้าที่เข้ารหัสจากเลขจำนวนเต็มให้เป็นเลข 8 บิทและทำการถอดรหัสจากเลข 8 บิทให้เป็นเลขจำนวนเต็ม 2 หลัก

G123 ทำหน้าที่คิดจำนวนเหรียญแต่ละเหรียญที่ต้องใช้ทอน

G62 ทำหน้าที่ส่งสัญญาณไปควบคุมให้บล็อก G123 เริ่มทำงาน

G71 ทำหน้าที่ส่งสัญญาณเอาท์พุทของบล็อก G123 ไปแสดงผลทาง 7-Segment และส่งไปยังบล็อก G3_4 หลังจากที่บล็อก G123 ได้ประมวลผลเสร็จแล้ว

G3_4 ทำหน้าที่ส่งสัญญาณไปควบคุมให้ relay ทำงาน

G5 ทำหน้าที่ส่งสัญญาณลอจิก 0 ไปเซ็ทค่าจำนวนเงินรวมที่รับเข้ามากลับเป็นศูนย์เหมือนในสภาวะเริ่มต้น

G81 ทำหน้าที่ส่งสัญญาณไปยังบล็อก G5 เพื่อเซ็ทค่าให้อยู่ในสภาวะลอจิก 1

SELECT_BIT และ SEGMENT ต่างๆจะ ทำหน้าที่แสดงจำนวนเงินที่รับเข้ามา จำนวนเงินทอนและจำนวนเหรียญที่ต้องใช้ทอนในแต่ละเหรียญ โดยแสดงผลออกทาง 7-Segment

ในแต่ละบล็อกการทำงานจะมีการเขียนโครงสร้างการทำงานภายในแต่ละบล็อกด้วยภาษา VHDL เพื่ออธิบายการทำงานเชิงพฤติกรรมของแต่ละบล็อกตามต้องการและรายละเอียดของโครงสร้างในแต่ละบล็อกการทำงานมีดังนี้

โครงสร้างของบล็อก CLK_DIV_25175

```
library ieee;
use ieee.std_logic_1164.all;
entity clk_div_25175 is
port(clk_in:in std_logic;
                clk_out:out std_logic
                );
end clk_div_25175;
```

```
architecture rtl of clk_div_25175 is
begin
 process(clk_in)
  variable temp_1:integer range 0 to 12587;
  variable temp_2:std_logic;
                    begin
                     if clk_in'event and  clk_in='1' then
                          if temp_1=12587 then
                                temp_1:=0;
                                temp_2:=not temp_2;
                          else
                                temp_1:=temp_1+1;
                          end if;
                                clk_out<=temp_2;
                     end if;
  end process;
end rtl;
```

โครงสร้างของบล็อก CLK_DIV_1000

```
library ieee;
use ieee.std_logic_1164.all;
entity clk_div_1000 is
port(clk_in:in std_logic;
      clk_out:out std_logic
      );
end clk_div_1000;
architecture rtl of clk_div_1000 is
begin
  process(clk_in)
   variable temp_1:integer range 0 to 499;
   variable temp_2:std_logic;
                    begin
```

```
        if clk_in'event and  clk_in='1' then
            if temp_1=499 then
                temp_1:=0;
                temp_2:=not temp_2;
            else
                temp_1:=temp_1+1;
            end if;
                clk_out<=temp_2;
        end if;
  end process;
end rtl;
```

โครงสร้างของบล็อก DEBOUND1

```
library ieee;
use ieee.std_logic_1164.all;
entity debound_1_2 is
port(reset:in std_logic;
        data_in:in std_logic;
        count_control:out std_logic
        );
end debound_1_2;
architecture rtl of debound_1_2 is
begin
 process(reset,data_in)
        begin
            if reset='0' then
                count_control<='0';
            elsif data_in'event and data_in='0' then
                count_control<='1';
            end if;
  end process;
  end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity debound_1_1 is
port(clk:in std_logic;
                count_control:in std_logic;
                data_in:in std_logic;
                reset:out std_logic
               );
end debound_1_1;
architecture rtl of debound_1_1 is
signal  count_temp:integer range 0 to 31;
begin
 process(clk)
                begin
                    if clk'event and clk='0' then
                     if count_control='1' then
                                  count_temp<=count_temp+1;
                     else
                                  count_temp<=0;
                     end if;
                    end if;
                   if clk'event and clk='1' then
                            case count_temp is
                                  when  24 =>   if data_in='1' then
                                                      reset<= '0';
                                                end if;
                                 when others  => reset<= '1';
                           end case;
                  end if;

  end process;
  end rtl;
```

โครงสร้างของบล็อก A5

```
library ieee;
use ieee.std_logic_1164.all;
entity a5 is
port (a : in std_logic;
                b : in std_logic;
                c : in std_logic;
                d : in std_logic;
               coin: out integer range 0 to 99
                );
end a5;
architecture rtl of a5 is
signal  count1: integer range 0 to 99;
signal  count2: integer range 0 to 99;
signal  count3: integer range 0 to 99;
begin
process(a,d)
begin
  if d='0' then
   count1<=0;
  elsif a'event and a='0' then
   count1<=count1+1;
  end if;
end process;

process(b,d)
begin
  if d='0' then
   count2<=0;
  elsif b'event and b='0' then
   count2<=count2+5;
  end if;
```

```
end process;

process(c,d)
begin
 if d='0' then
  count3<=0;
 elsif c'event and c='0' then
  count3<=count3+10;
 end if;
end process;
                    coin<=count1+count2+count3;
end rtl;
```

โครงสร้างของบล็อก SUM_CLK1

```
library ieee;
use ieee.std_logic_1164.all;
entity sum_clk1 is
port(a:in std_logic;
                    b:in std_logic;
                    c:in std_logic;
                    d:in std_logic;
                    clk:out std_logic
                    );
end sum_clk1;
architecture rtl of sum_clk1 is
begin
 clk<='1' when a='1' and b='1' and c='1' and d='1' else
                     '0';
end rtl;
```

โครงสร้างของบล็อก A4

```
library ieee;
use ieee.std_logic_1164.all;
entity a4 is
port( data_i: in std_logic_vector(2 downto 0);
                    j       : out integer range 0 to 99 );
end a4;
architecture RTL of a4 is
begin
                process(data_i)
                variable m : integer range 0 to 99;
                begin
                 case data_i is
                        when "110" =>m:=38;
                        when "101" =>m:=45;
                        when "011" =>m:=64;
                   when others =>m:=0;
                 end case;
                            j<=m;
                end process;
end rtl;
```

โครงสร้างของบล็อก SUM

```
library ieee;
use ieee.std_logic_1164.all;
entity sum is
port(clk:in std_logic;
    coin_in:in integer range 0 to 99;
                 coin_out:out integer range 0 to 99
                );
end sum;
architecture rtl of sum is
```

```
begin
 process(clk)
                    begin
 if clk'event and clk='1' then
                     coin_out<=coin_in;
 end if;
 end process;
end rtl;
```

โครงสร้างของบล็อก RECOIN2

```
library ieee;
use ieee.std_logic_1164.all;
entity recoin2 is
port (coin: in integer range 0 to 99;
                      j : in integer range 0 to 99;
                       clk: in std_logic;
      rec: out integer range 0 to 99
                     );
end recoin2;
architecture rtl of recoin2 is
begin
                    process(clk)
                    variable temp:integer range 0 to 99;
                    begin
                          if clk'event and clk='1' then
                                if  coin<j then
                                      temp:=coin;
                                elsif coin>=j then
                                      temp:=coin-j;
                                end if;
                                      rec<=temp;
                          end if;
```

```
                end process;
end rtl;
```

โครงสร้างของบล็อก ENCODE

```
library ieee;
use ieee.std_logic_1164.all;
entity encode is
port(coin : in integer range 0 to 99;
                ab  : out std_logic_vector(7 downto 0)
   );
end encode;
architecture rtl of encode is
begin
                with coin select
                ab<="00000000" when 0,
                        "00000001" when 1,
                        "00000010" when 2,
                        "00000011" when 3,
                        "00000100" when 4,
                        "00000101" when 5,
                        "00000110" when 6,
                        "00000111" when 7,
                        "00001000" when 8,
                        "00001001" when 9,

                        "00010000" when 10,
                        "00010001" when 11,
                        "00010010" when 12,
                        "00010011" when 13,
                        "00010100" when 14,
                        "00010101" when 15,
                        "00010110" when 16,
```

```
"00010111" when 17,
"00011000" when 18,
"00011001" when 19,

"00100000" when 20,
"00100001" when 21,
"00100010" when 22,
"00100011" when 23,
"00100100" when 24,
"00100101" when 25,
"00100110" when 26,
"00100111" when 27,
"00101000" when 28,
"00101001" when 29,

"00110000" when 30,
"00110001" when 31,
"00110010" when 32,
"00110011" when 33,
"00110100" when 34,
"00110101" when 35,
"00110110" when 36,
"00110111" when 37,
"00111000" when 38,
"00111001" when 39,

"01000000" when 40,
"01000001" when 41,
"01000010" when 42,
"01000011" when 43,
"01000100" when 44,
"01000101" when 45,
```

```
"01000110" when 46,
"01000111" when 47,
"01001000" when 48,
"01001001" when 49,

"01010000" when 50,
"01010001" when 51,
"01010010" when 52,
"01010011" when 53,
"01010100" when 54,
"01010101" when 55,
"01010110" when 56,
"01010111" when 57,
"01011000" when 58,
"01011001" when 59,

"01100000" when 60,
"01100001" when 61,
"01100010" when 62,
"01100011" when 63,
"01100100" when 64,
"01100101" when 65,
"01100110" when 66,
"01100111" when 67,
"01101000" when 68,
"01101001" when 69,

"01110000" when 70,
"01110001" when 71,
"01110010" when 72,
"01110011" when 73,
"01110100" when 74,
```

```
"01110101" when 75,
"01110110" when 76,
"01110111" when 77,
"01111000" when 78,
"01111001" when 79,

"10000000" when 80,
"10000001" when 81,
"10000010" when 82,
"10000011" when 83,
"10000100" when 84,
"10000101" when 85,
"10000110" when 86,
"10000111" when 87,
"10001000" when 88,
"10001001" when 89,

"10010000" when 90,
"10010001" when 91,
"10010010" when 92,
"10010011" when 93,
"10010100" when 94,
"10010101" when 95,
"10010110" when 96,
"10010111" when 97,
"10011000" when 98,
"10011001" when 99;

end rtl;
```

โครงสร้างของบล็อก DECODE

```
library ieee;
use ieee.std_logic_1164.all;
entity dec is
port(ab : in std_logic_vector(7 downto 0);
    a1 : out integer range 0 to 9;
    b1 : out integer range 0 to 9
   );
end dec;
architecture rtl of dec is
begin
                process(ab)
                variable a : integer range 0 to 9;
                variable b : integer range 0 to 9;
                begin
                        case ab(7 downto 4) is
                                when"0000"=>a:=0;
                                when"0001"=>a:=1;
                                when"0010"=>a:=2;
                                when"0011"=>a:=3;
                                when"0100"=>a:=4;
                                when"0101"=>a:=5;
                                when"0110"=>a:=6;
                                when"0111"=>a:=7;
                                when"1000"=>a:=8;
                                when"1001"=>a:=9;
                                when others=>null;
                        end case;
                                        a1<=a;
                        case ab(3 downto 0) is
                                when"0000"=>b:=0;
                                when"0001"=>b:=1;
```

```
                when"0010"=>b:=2;
                when"0011"=>b:=3;
                when"0100"=>b:=4;
                when"0101"=>b:=5;
                when"0110"=>b:=6;
                when"0111"=>b:=7;
                when"1000"=>b:=8;
                when"1001"=>b:=9;
                when others=>null;
            end case;
                    b1<=b;
        end process;
end rtl;
```

โครงสร้างของบล็อก G123

```
library ieee;
use ieee.std_logic_1164.all;
entity g11 is
port(clk:in std_logic;
        rec:in integer range 0 to 99;
        con:in std_logic;
        reset:out std_logic;
        ten_out:out integer range 0 to 9;
        five_out:out integer range 0 to 9;
        one_out:out integer range 0 to 9
   );
end g11;
architecture rtl of g11 is
signal ten:integer range 0 to 9;
signal five:integer range 0 to 9;
signal one:integer range 0 to 9;
begin
```

```
process(con,clk)
variable money:integer range 0 to 99;
begin
if con='0' then
        reset<='1';
elsif (clk'event)and(clk='1')then
        if money=0 then
                money:=rec;
                ten<=0;
                five<=0;
                one<=0;
        elsif money>10 then
                money:=money-10;
                ten<=ten+1;
        elsif money>5 then
                money:=money-5;
                five<=five+1;
        else one<=money;
                money:=0;
                reset<='0';
        end if;
end if;
end process;
        ten_out<=ten;
        five_out<=five;
        one_out<=one;
end rtl;
library ieee;
use ieee.std_logic_1164.all;
entity g2 is
port(reset:in std_logic;
        clk_in:in std_logic;
```

```
                data_in:in std_logic;
                con_out:out std_logic;
                clk_out:out std_logic
                );
end g2;
architecture rtl of g2 is
signal con:std_logic;
begin
                process(reset,data_in)
                begin
                if reset='0' then
                        con<='0';
                elsif data_in'event and data_in='1' then
                        con<='1';
                end if;
                end process;
                        con_out<=con;
                        clk_out<=clk_in and con;
end rtl;
```

โครงสร้างของบล็อก G62

```
library ieee;
use ieee.std_logic_1164.all;
entity g6 is
port(reset:in std_logic;
                data_in:in std_logic;
                count_control:out std_logic
                );
end g6;
architecture rtl of g6 is
begin
 process(reset,data_in)
```

```
                begin
                        if reset='0' then
                                count_control<='0';
                        elsif data_in'event and data_in='1' then
                                count_control<='1';
                        end if;
 end process;
end rtl;
library ieee;
use ieee.std_logic_1164.all;
entity g61 is
port(clk:in std_logic;
                count_control:in std_logic;
                data_in:in std_logic;
                reset:out std_logic
                );
end g61;
architecture rtl of g61 is
signal  count_temp:integer range 0 to 9;
begin
 process(clk)
                begin
                        if clk'event and clk='0' then
                         if count_control='1' then
                                        count_temp<=count_temp+1;
                         else
                                        count_temp<=0;
                         end if;
                        end if;
                        if clk'event and clk='1' then
                                case count_temp is
                                        when  8 =>   if data_in='1' then
```

```
                                        reset<= '0';
                                    end if;
                                when others  => reset<= '1';
                            end case;
                        end if;

end process;
end rtl;
```

โครงสร้างของบล็อก G71

```
library ieee;
use ieee.std_logic_1164.all;
entity g71 is
port(reset:in std_logic;
                rec:in integer range 0 to 99;
    ten_in:in integer range 0 to 9;
                five_in:in integer range 0 to 9;
                one_in:in integer range 0 to 9;
                ten_out:out integer range 0 to 9;
                five_out:out integer range 0 to 9;
                one_out:out integer range 0 to 9
                );
end g71;
architecture rtl of g71 is
begin
 process(rec,reset)
                begin
                if rec=0 then
                        ten_out<=0;
                        five_out<=0;
                        one_out<=0;
                elsif reset'event and reset='1' then
```

```
                ten_out<=ten_in;
                five_out<=five_in;
                one_out<=one_in;
            end if;
end process;
end rtl;
```

โครงสร้างของบล็อก G3_4

```
library ieee;
use ieee.std_logic_1164.all;
entity g3 is
port(clk:in std_logic;
        con:in std_logic;
        a  :in integer range 0 to 9;
        d  :out std_logic;
        reset:out std_logic
        );
end g3;
architecture rtl of g3 is
signal c :std_logic;
begin
        process(con,clk)
        variable count:integer range 0 to 9;
        begin
            if con='0' then
                reset<='1';
            elsif clk'event and clk='0' then
                if count=a then
                    c<='0';
                    count:=0;
                    reset<='0';
                else count:=count+1;
```

```
                    c<='1';
                end if;
            end if;
        end process;
            d<= clk and c;
end rtl;
library ieee;
use ieee.std_logic_1164.all;
entity g4 is
port(reset:in std_logic;
        clk_in:in std_logic;
        data_in:in std_logic;
        con_out:out std_logic;
        clk_out:out std_logic
        );
end g4;
architecture rtl of g4 is
signal con:std_logic;
begin
        process(reset,data_in)
        begin
        if reset='0' then
            con<='0';
        elsif data_in'event and data_in='0' then
            con<='1';
        end if;
        end process;
            con_out<=con;
            clk_out<=clk_in and con;
end rtl;
```

โครงสร้างของบล็อก G5

```
library ieee;
use ieee.std_logic_1164.all;
entity g5 is
port (a : in std_logic;
                b : in std_logic;
                c : in std_logic;
                d : in std_logic;
               coin: out std_logic
                );
end g5;
architecture rtl of g5 is
signal  count1: std_logic;
signal  count2: std_logic;
signal  count3: std_logic;
begin
process(a,d)
begin
  if d='0' then
   count1<='1';
  elsif a'event and a='0' then
   count1<='0';
  end if;
end process;

process(b,d)
begin
  if d='0' then
   count2<='1';
  elsif b'event and b='0' then
   count2<='0';
  end if;
```

```
end process;

process(c,d)
begin
 if d='0' then
  count3<='1';
 elsif c'event and c='0' then
  count3<='0';
 end if;
end process;
                coin<=count1 or count2 or count3;
end rtl;
```

โครงสร้างของบล็อก G81

```
library ieee;
use ieee.std_logic_1164.all;
entity g8 is
port(clk:in std_logic;
                count_control:in std_logic;
                reset:out std_logic
                );
end g8;
architecture rtl of g8 is
signal  count_temp:integer range 0 to 9;
begin
 process(clk)
                begin
                        if clk'event and clk='0' then
                         if count_control='1' then
                                        count_temp<=count_temp+1;
                                else
                                        count_temp<=0;
```

```
                end if;
            end if;
            if clk'event and clk='1' then
                case count_temp is
                    when  8 => reset<= '0';
                    when others => reset<= '1';
                end case;
            end if;
end process;
end rtl;
library ieee;
use ieee.std_logic_1164.all;
entity debound_1_2 is
port(reset:in std_logic;
        data_in:in std_logic;
        count_control:out std_logic
        );
end debound_1_2;
architecture rtl of debound_1_2 is
begin
 process(reset,data_in)
        begin
            if reset='0' then
                count_control<='0';
            elsif data_in'event and data_in='0' then
                count_control<='1';
            end if;
 end process;
end rtl;
```

โครงสร้างของบล็อก SELECT_BIT

```
library ieee;
use ieee.std_logic_1164.all;
entity select2_bit is
port(a:in integer range 0 to 9;
                b:in integer range 0 to 9;
                clk:in std_logic;
                sel_bit:out std_logic_vector(1 downto 0);
                bcd:out integer range 0 to 9
                );
end select2_bit;
architecture rtl of select2_bit is
signal temp:std_logic;
begin
 process(clk)
                begin
                if  clk'event and clk='1' then
                 if a=0 then
                        temp<='1';
                else
                        temp<=not temp;
                 end if;
                end if;
 end process;
 process(clk,temp)
  begin
    if clk'event and clk='0' then
                  if temp='0' then
                        bcd<=a;
                        sel_bit<="01";
                  else
                        bcd<=b;
```

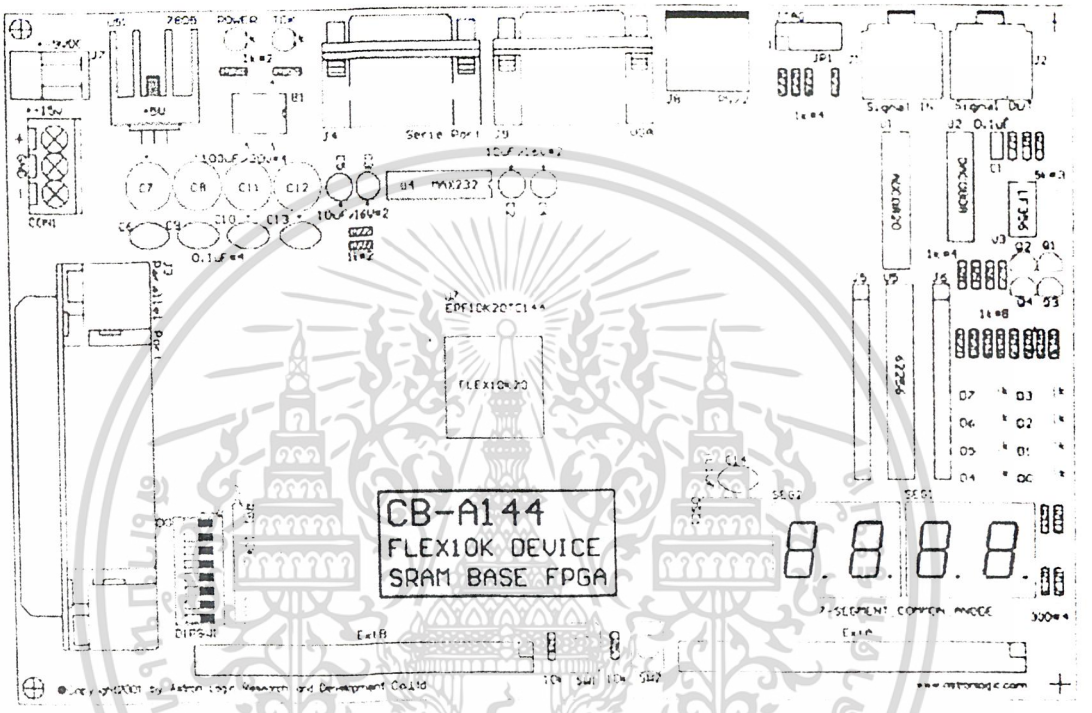
```
                    sel_bit<="10";
                end if;
                end if;
end process;
end rtl;
```

โครงสร้างของบล็อก SEGMENT

```
library ieee;
use ieee.std_logic_1164.all;
entity segment is
port(  bcd  :in integer range 0 to 9;
                seg_7 :out std_logic_vector(6 downto 0)
                );
end segment;
architecture rtl of segment is
begin
                with bcd select
                seg_7<="1001111"when 1,
                                "0010010"when 2,
                                "0000110"when 3,
                                "1001100"when 4,
                                "0100100"when 5,
                                "0100000"when 6,
                                "0001111"when 7,
                                "0000000"when 8,
                                "0000100"when 9,
                                "0000001"when 0;
end rtl;
```

เมื่อได้ทำการเขียนโครงสร้างภายในของแต่ละบล็อกแล้วจึงทำการเรียบเรียง(Complier) และจำลองการทำงาน(Simulate) เมื่อผลการทำงานแต่ละบล็อกเป็นไปตามที่ต้องการแล้วก็ให้นำบล็อกการทำงานมาต่อรวมกันเพื่อให้เป็นบล็อกการทำงานที่สมบูรณ์ของเครื่องนับแยกและทอน

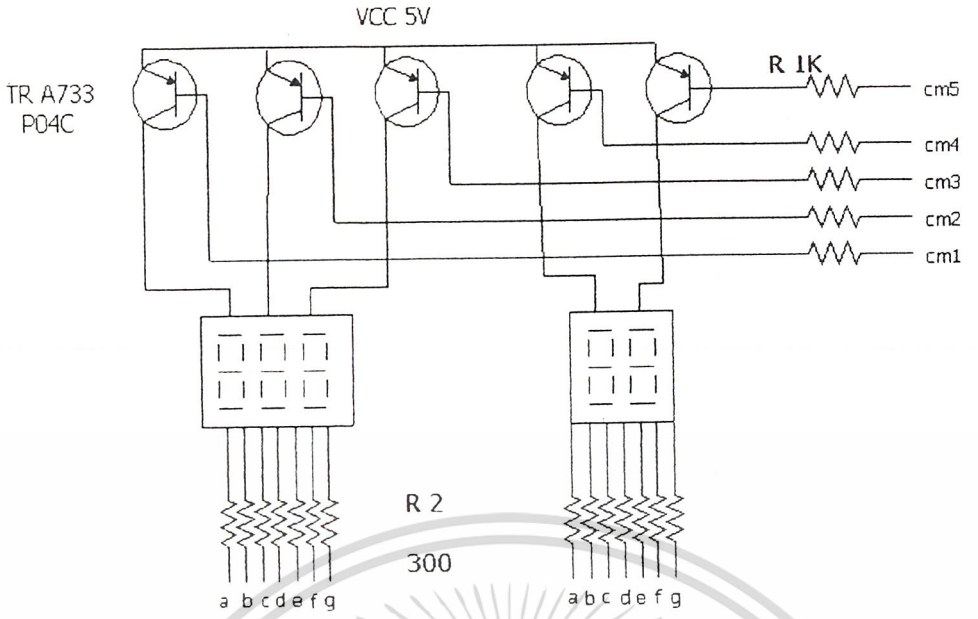
เหรียญอัตโนมัติ หลังจากนั้นจึงทำการจำลองการทำงานของทั้งระบบเพื่อดูการทำงานที่ได้ว่าเป็นไปตามที่ต้องการหรือไม่และมีข้อบกพร่องส่วนใดบ้างหรือไม่ หลังจากที่ทำการปรับปรุงได้ผลจำลองการทำงานเป็นไปตามที่ต้องการแล้ว ในขั้นตอนสุดท้ายก็จะเป็นการป้อนโปรแกรมลงในอุปกรณ์ FPGA เพื่อทดสอบและดูผลการทำงานจริงเป็นอันดับต่อไป ซึ่งจะใช้บอร์ด MASTER FLEX-A01 ดังรูป



รูปที่ 3.3 แสดงลักษณะของบอร์ด MASTER FLEX-A01

3.2.3 ชุดแสดงผล

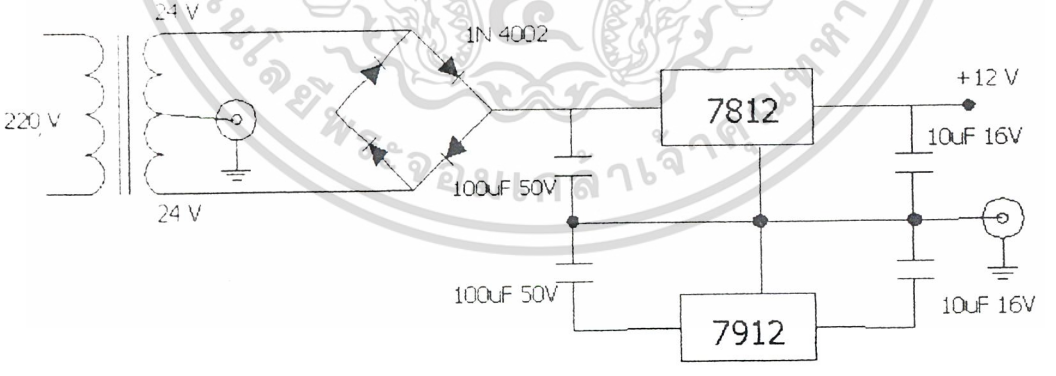
เมื่อชิพ FLEX 10K ทำการประมวลผลเสร็จแล้ว จะทำการส่งข้อมูลที่เป็นจำนวนเงินรวมที่หยอดเข้ามา จำนวนเงินทอน และจำนวนเหรียญที่ต้องใช้ทอน แสดงผลออกทาง 7-Segment และเมื่อทำการกดสวิทซ์เงินทอน RELAY จะทำงานตามจำนวนเหรียญที่ต้องใช้ทอนในแต่ละเหรียญ และจะแสดงผลออกทาง LED ซึ่งทั้ง LED และ RELAY จะมีอยู่อย่างละ 3 ตัว สำหรับเหรียญ1 บาท 5 บาท และ10 บาท ซึ่งวงจรจะแสดงดังรูป



รูปที่ 3.4 แสดงลักษณะวงจรของชุดแสดงผล

3.2.4 ชุดแหล่งจ่ายไฟ

ชุดแหล่งจ่ายไฟ ทำหน้าที่จ่ายไฟตรง 12 โวลท์ ให้กับบอร์ด MASTER FLEX-A01 ซึ่งจะมีลักษณะของวงจรแสดงดังรูป



รูปที่ 3.5 แสดงลักษณะวงจรของชุดแหล่งจ่ายไฟ

บทที่ 4 ผลการทดลอง

การทดลองการแยกเหรียญ

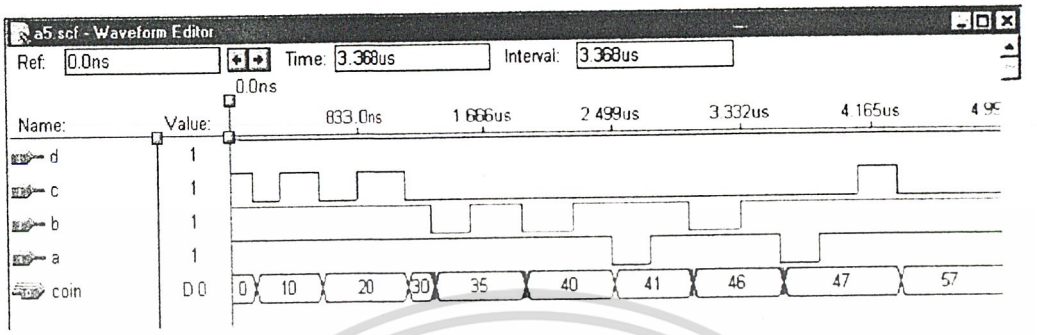
การแยกเหรียญที่ถูกต้อง ชุดที่ทำการแยกเหรียญ จะทำการแยกเหรียญตามขนาดเส้นผ่านศูนย์กลาง โดยเมื่อทำการหยอดเหรียญลงมา เหรียญจะไหลลงมาตามราง ซึ่งช่องแรก จะเป็นขนาดเส้นผ่านศูนย์กลางของเหรียญ 1 บาท ช่องต่อมาก็จะเป็นขนาดเส้นผ่านศูนย์กลางของเหรียญ5 บาท และช่องสุดท้ายก็จะเป็นขนาดเส้นผ่านศูนย์กลางของเหรียญ10 บาท นั่นก็หมายความว่า เหรียญจะสามารถไหลไปตามรางที่มีขนาดเท่ากับตัวเหรียญเองเท่านั้น โดยการทดลอง จะทำการหยอดเหรียญ 10 บาท 5 บาท 1 บาท จำนวนชนิดละ 10 ครั้ง ชุดแยกเหรียญสามารถแยกเหรียญได้ตรงกับที่ได้ออกแบบไว้

ตารางแสดงผลการทดลองการแยกเหรียญ

จำนวนครั้ง	เหรียญ 1 บาท	เหรียญ 5 บาท	เหรียญ 10 บาท
1	ผ่าน	ผ่าน	ผ่าน
2	ผ่าน	ผ่าน	ผ่าน
3	ผ่าน	ผ่าน	ผ่าน
4	ผ่าน	ผ่าน	ผ่าน
5	ผ่าน	ผ่าน	ผ่าน
6	ผ่าน	ผ่าน	ผ่าน
7	ผ่าน	ผ่าน	ผ่าน
8	ไม่ผ่าน	ผ่าน	ผ่าน
9	ผ่าน	ผ่าน	ผ่าน
10	ผ่าน	ผ่าน	ผ่าน
เปอร์เซ็นความถูกต้อง(%)	90	100	100

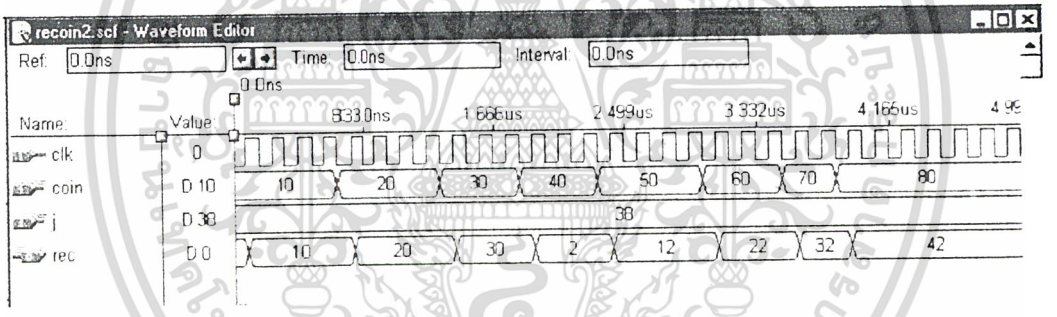
หมายเหตุ จากตารางการทดลอง "ผ่าน" หมายถึง สามารถแยกเหรียญได้ถูกต้อง เหรียญไหลไปตามรางที่ได้กำหนดไว้ "ไม่ผ่าน" หมายถึง ไม่สามารถแยกเหรียญได้ เหรียญไหลผิดรางซึ่งจะส่งผลทำให้การประมวลผลผิดพลาด

การทดลองการนับจำนวนเงินที่หยอดเข้ามา

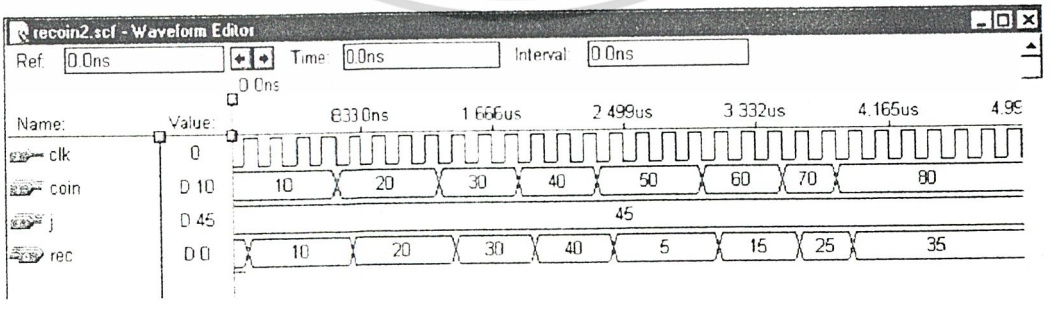


รูปที่ 4.1 แสดงผลการ Simulated ของการคิดจำนวนเงินรวมที่หยอดเข้ามา

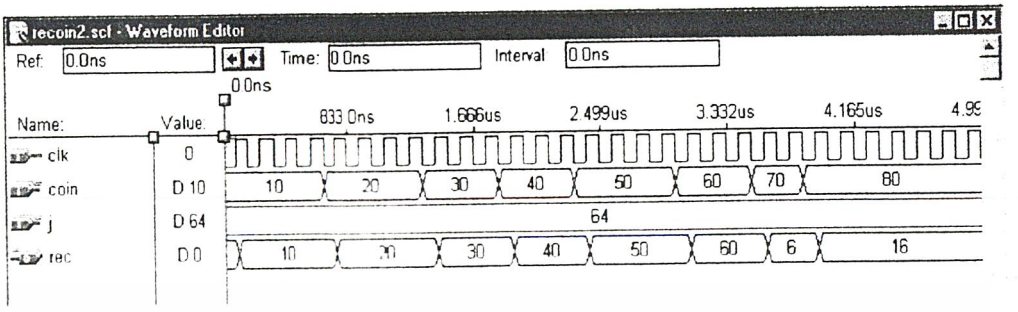
การทดลองการคิดจำนวนเงินทอน



รูปที่ 4.2 แสดงผลการ Simulated ของการคิดจำนวนเงินทอนในกรณีที่เซ็ทค่าเงินเป็น 38 บาท

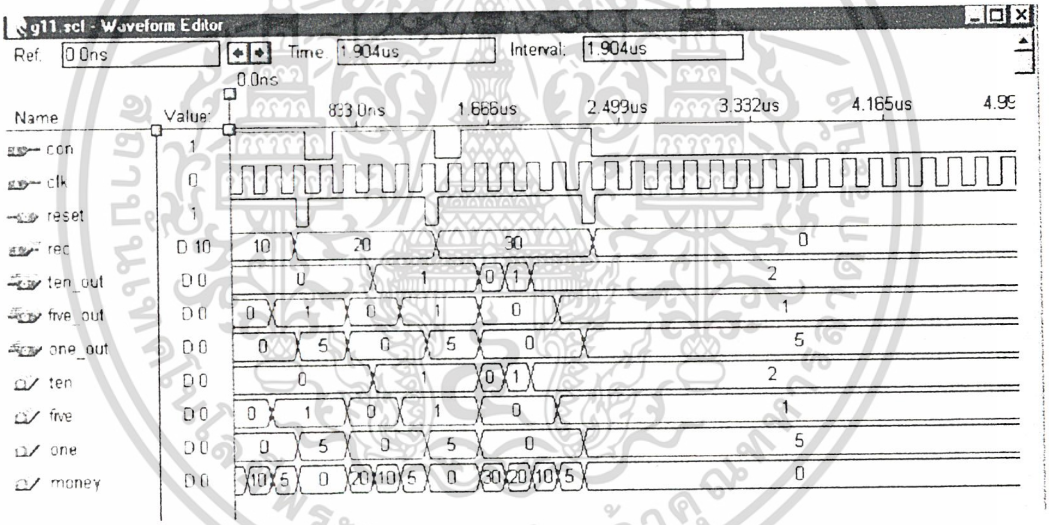


รูปที่ 4.3 แสดงผลการ Simulated ของการคิดจำนวนเงินทอนในกรณีที่เซ็ทค่าเงินเป็น 45 บาท



รูปที่ 4.4 แสดงผลการ Simulated ของการคิดจำนวนเงินทอนในกรณีที่เซ็ทค่าเงินเป็น 64 บาท

การทดลองการคิดจำนวนเหรียญที่ใช้ทอน



รูปที่ 4.5 แสดงผลการ Simulated ของการคิดจำนวนเหรียญที่ต้องใช้ทอน

บทที่ 5 สรุปและวิจารณ์ผลการทดลอง

5.1 สรุปผลการทดลอง

5.1.1 ชุดการแยกเหรียญ

จากทดลองพบว่า ชุดแยกเหรียญ จะทำการแยกเหรียญตามขนาดเส้นผ่านศูนย์กลางของเหรียญ 1 บาท 5 บาท 10 บาท และในการหยอดเหรียญแต่ละเหรียญนั้น ค่าเส้นผ่านศูนย์กลางที่ได้จากการหยอดแต่ละครั้ง มีเสถียรภาพต่ำ ทั้งนี้อาจเนื่องมาจาก ลักษณะของรางที่ไม่เรียบ และการหยอดที่แตกต่างกันจึงทำให้ผลที่ได้เกิดความผิดพลาด

5.1.2 ชุดประมวลผล

จากผลการ Simulated พบว่า ชุดประมวลผลได้ทำการประมวลผลออกมาตามโปรแกรมที่ได้กำหนดไว้ ข้อมูลที่ได้ถูกต้อง ไม่มีความผิดพลาดและมีการแสดงผลโดยใช้ 7-Segment เพื่อแสดงจำนวนเงินที่หยอดเข้ามา จำนวนเงินทอน และจำนวนเหรียญ 1บท 5 บาท 10บาท ที่จะต้องใช้ทอน ส่วนของการทอนเหรียญออกมาจะใช้การแสดงผลเป็น LED แทน

5.2 ปัญหาและอุปสรรคที่เกิดขึ้นในการออกแบบสร้างเครื่องนับแยกและทอนเหรียญอัตโนมัติ

5.2.1ในการออกแบบสร้างรางที่ใช้ในการแยกเหรียญ

ในการออกแบบสร้างรางที่ใช้ในการแยกเหรียญ ส่งผลให้การแยกเหรียญมีประสิทธิภาพต่ำลง ทั้งนี้เนื่องจากเป็นการสร้างโดยมิได้อาศัยเครื่องจักรในการสร้าง ทำให้ลักษณะของรางไม่เรียบ รวมถึงองศาการเอียงของราง ซึ่งต้องออกแบบให้เหมาะสม จึงจะทำการแยกเหรียญได้อย่างมีประสิทธิภาพ

5.2.2 การใช้อุปกรณ์ FPGA ในการควบคุม

การใช้อุปกรณ์ FPGA ในการควบคุม จะต้องทำการศึกษา ค้นคว้าเกี่ยวกับอุปกรณ์ FPGA อย่างละเอียด ซึ่งต้องอาศัยเวลาในการศึกษาและทดลองจนเกิดความชำนาญ

5.3 แนวทางในการพัฒนาและปรับปรุงโครงงาน

สำหรับโครงงานนี้สามารถนำไปเป็นต้นแบบ เพื่อพัฒนาเป็นเครื่องนับแยกและทอนเหรียญอัตโนมัติที่มีความถูกต้องสูง สำหรับในชุดของการแยกเหรียญ ควรจะทำภาคตรวจสอบวัสดุที่ใช้ทำเหรียญ เพื่อสามารถตรวจสอบว่าเป็นเหรียญจริงหรือปลอมได้ และลักษณะของรางที่ใช้ในการแยกเหรียญควรทำให้มีลักษณะที่มีความเรียบมาก และมีองศาการเอียงของรางที่เหมาะสม เพื่อเพิ่มประสิทธิภาพในการแยกเหรียญได้แม่นยำขึ้น และสามารถนำไปใช้งานได้จริงในอนาคต

หนังสืออ้างอิง

1. Frank A. Scarpino,Ph.D., "VHDL And AHDL Digital System Implementation" , Prentice Hall PRT , 316p. ,1998

2. Stefan Sjholm And Lennart Lindh, "VHDL for Designers " , Prentice Hall, 473p., 1997