การแสดงผลภาพทางหลอด LED 3สี โดยใช้ FPGA

RGB Display via Three-Colored LED By FPGA

โดย

นาย นิเวศน์ แก้วใจ

นาย พงศ์พันธ์ แถลงนิตย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

การแสดงผลภาพทางหลอด LED 3สี โดยใช้ FPGA

RGB Display via Three-Colored LED By FPGA

โดย

นาย นิเวศน์ แก้วใจ 49015147

นาย พงศ์พันธ์ แฉลงนิตย์ 49015200

อาจารย์ที่ปรึกษา

รศ. ดร. มนัส สังวรศิลป์

ปริญญานิพนธ์นี้สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

ปริญญานิพนธ์ ปีการศึกษา 2551

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การแสดงผลภาพทางหลอด LED 3สี โดยใช้ FPGA (RGB Display via Three-Colored LED By FPGA)

ผู้จัดทำ 1. นาย นิเวศน์ แก้วใจ 49015147

2. นาย พงศ์พันธ์ แถลงนิตย์ 49015200

..... อาจารย์ที่ปรึกษา

(รศ. ดร. มนัส สังวรศิลป์)

การแสดงผลภาพทางหลอดLED 3สี โดยใช้ FPGA

นาย นิเวศน์ แก้วใจ 49015147
นาย พงศ์พันธ์ แถลงนิตย์ 49015200
รศ.ดร. มนัส สังวรศิลป์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2551

บทคัดย่อ

โครงการนี้เป็นการนำเสนอการออกแบบวงจรปรับระดับสีโดยแสดงผลผ่านLED 3 สีซึ่งสามารถปรับระดับความสว่างของสีได้ 4096 ระดับโดยใช้ FPGA เป็นตัวประมวลผลสัญญาณของระดับสี FPGA จะรับสัญญาณจากตัวถอดรหัสสีและทำการเก็บข้อมูลแล้วส่งผ่านข้อมูลให้ IC 5941 เป็นตัวปรับระดับสีแล้วจึงแสดงผลทาง LED3 สี

RGB Display Via Three-Colored LED By FPGA

Mr. Niwet Kaewjai 49015147

Mr. Pongpan Thalaengnit 49015200

Assoc.Prof. Manus Sangvornsilp, Ph.d.

(Advisor)

Academic 2008

Abstract

This project presents the design of color adjustment circuits displayed on three-colored LED. The brightness can be adjusted to 4096 steps using FPGA to process the color level signal. FPGA receivers the signal from the color decoder and stores the data before transferring then to IC TLC 5941 which adjusts the color level and the result is shown on the three-colored LED.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดีเนื่องจากการสนับสนุนและช่วยเหลือจากอาจารย์ที่ปรึกษาโปรเจค รศ.ดร.มนัส สังวรศิลป์ และขอขอบคุณพี่สารัตถ์ที่ให้คำปรึกษาและช่วยเหลือในทุกๆเรื่องของการทำหัวข้อการทำปริญญานิพนธ์ครั้งนี้และนายอนุภาค มาตรมูลที่ช่วยออกแบบและสุดท้ายขอกราบขอบพระคุณบิดามารดาซึ่งเป็นผู้อุปการะในการเรียน ทั้งคอยดูแลและช่วยเหลือทุกอย่างในด้านการศึกษา

ผู้จัดทำ

นาย นิเวศน์ แก้วใจ

นาย พงศ์พันธ์ แถลงนิตย์

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูปภาพ

สารบัญรูปภาพ(ต่อ)

สารบัญตาราง

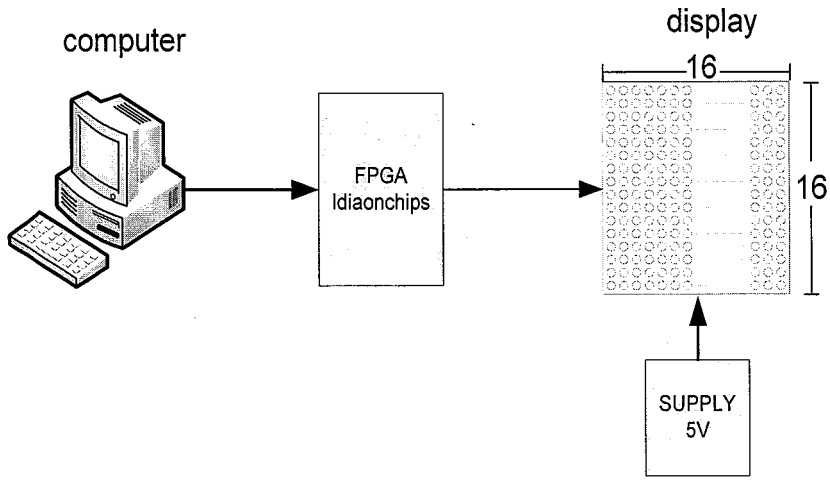
บทที่1

บทนำ

1.1 ความสำคัญและความเป็นมา

ในปัจจุบันเทคโนโลยีได้มีการพัฒนาไปอย่างรวดเร็วและมีความทันสมัยขึ้นเพื่ออำนวยความสะดวกสบายให้กับมนุษย์ ในการพัฒนาเทคโนโลยีทางระบบดิจิตอลให้ทันสมัย การออกแบบระบบจะมีความซับซ้อนมากขึ้นเรื่อยๆและจำเป็นต้องการตัวประมวลผลการทำงานที่มีความเร็วสูงในการประมวลผลของระบบ

ซึ่งผู้จัดทำโครงการนี้ได้เล็งเห็นว่า FPGA (Field programmable gate array) เป็นเทคโนโลยีที่สามรถรองรับการทำงานของระบบที่มีความซับซ้อนได้เป็นอย่างดี ซึ่ง FPGA มีคุณสมบัติกึ่งซอฟท์แวร์และฮาร์ดแวร์กล่าวคือมีสามารถนำมาใช้งานเป็นฮาร์ดแวร์โดยที่สามารถโปรแกรมได้ (Programmable) หรือเปลี่ยนลักษณะของการทำงานต่างๆได้(Reconfigurable) เหมือนซอฟท์แวร์ ทำให้มีความยืดหยุ่นในการใช้งานสูง ดังนั้นจึงได้เกิดโครงการนี้ขึ้นมาเป็นโครงการการแสดงผลภาพทางหลอดLED 3สี โดยใช้ FPGA เป็นตัวรับข้อมูลค่าสีต่างๆจากแหล่งจ่ายภายนอกและทำการติดต่อกับ IC TLC5941 ซึ่งมีคุณสมบัติเป็นตัวขับหลอด LED ขนาด16 Channel สามารถควบคุมความสว่างโดยใช้หลักการ PWM (Pulse Width Modulation) ในการปรับระดับความสว่างของหลอด



รูปที่1.1 แสดงกระบวนการทำงานทั้งหมดของระบบ

1.2 วัตถุประสงค์

1.2.1 เพื่อศึกษาการเขียนโปรแกรมลงใน FPGA และศึกษาการทำงานของ IC TLC5941

1.2.2 เพื่อแสดงการปรับระดับความละเอียดของสีต่างๆได้

1.2.3 เพื่อแสดงการผสมสีระหว่างแม่สีเพื่อให้ได้สีตามที่ต้องการ

1.2.4 เพื่อนำไปประยุกต์พัฒนาเป็นการทำจอ LCD

1.2.5 เพื่อเตรียมความพร้อมก่อนที่จะได้ออกไปทำงาน ทางด้านวิศวกรรมและฝึกทักษะการแก้ปัญหาเฉพาะหน้าในการทำงานจริง

1.3 ขอบเขตของการทำงาน

ศึกษาและทดลองการปรับระดับความละเอียดของสีแดงสีเขียวและสีน้ำเงินและสามารถผสมสีระหว่างแม่สีRGBเข้าด้วยกันเพื่อให้เกิดสีใหม่ขึ้นมาโดยแสดงผลทางหลอดLED

บทที่2

ทฤษฎีที่เกี่ยวข้อง

2.1 ทฤษฎี Verilog HDL เบื้องต้น

2.1.1 ภาษาฮาร์ดแวร์ (Hardware Description Language)

ภาษาฮาร์ดแวร์ คือภาษาชั้นสูงที่ใช้ในการอธิบายการทำงานของฮาร์ดแวร์ โดยที่เราไม่จำเป็นต้องแปลงเป็นสมการบูลีน หรือ Schematic ตัวอย่างเช่น ภาษา HDL ส่วนใหญ่สามารถสร้างหรืออธิบาย Finite sate machine สำหรับวงจร Sequential logic ได้ หรือสร้างเป็นตารางค่าความจริง (Truth table) สำหรับวงจรCombinational logic ได้ภาษา HDL ใช้เป็นภาษาหลักในการออกแบบระบบในอุปกรณ์ประเภท Programmable logic device (PLD) หรือComplex PLD (CPLD) และ Field programmable gate array (FPGA) ซึ่งในปัจจุบันมีภาษาลักษณะนี้ใช้งานกันหลาย ๆ ตัว แต่ตัวที่เป็นที่นิยม และรู้จักกันได้แก่ Abel,Palasm, Cupl ที่ใช้สำหรับการออกแบบวงจรที่ไม่ซับซ้อนนักใน PLD หรือ Verilog, VHDL สำหรับการออกแบบระบบที่ซับซ้อนในอุปกรณ์พวก CPLD หรือ FPGA

2.1.2 การออกแบบในระดับต่าง ๆ (Dealing with Different Design Levels)

เนื่องจากวงจรในปัจจุบันเพิ่มความซับซ้อนมากขึ้นเรื่อย ๆ ดังนั้นแนวโน้มของความต้องการในการออกแบบคือ ความสามารถในการเปลี่ยน หรือรองรับลักษณะของ Design entry ได้หลาย ๆ แบบ เช่นสามารถออกแบบได้ทั้งการใช้ภาษาชั้นสูงในการอธิบายระบบ หรือการใช้สมการบูลีน หรือการใช้ Schematicเราสามารถแบ่งระดับของการอธิบายระบบได้เป็นชั้น จากชั้นล่างสุดได้แก่ Silicon level ไปยังชั้นสูงสุดได้แก่ System level โดยที่แต่ละระดับสามารถแยกได้เป็นการอธิบายในลักษณะของโครงสร้าง (Structure)หรือการทำงาน (Behavior)การออกแบบในระดับล่าง ๆ นั้นสามารถกำหนดรายละเอียด หรือคุณสมบัติของวงจรได้ทุกขั้นตอน เริ่มตั้งแต่ขนาดของทรานซิสเตอร์ ที่จะนำมาสร้างเป็นเกต ทำให้เราได้วงจรตามที่กำหนดร้อยเปอร์เซ็นต์ แต่ทว่าผู้ออกแบบจะต้องใช้เวลาในการออกแบบมาก รวมทั้งจำเป็นต้องมีประสบการณ์พอสมควร ส่วนการออกแบบในระดับสูง ๆ นั้นเราสามารถทำการออกแบบได้เร็ว จากการกำหนดการทำงานของระบบในระดับที่สูงขึ้นทำให้ทำได้ง่ายขึ้นหน้าที่ของการสร้างวงจรในระดับล่างลงมาในแต่ละระดับนั้นตัวเครื่องมือจะเป็นตัวรับผิดชอบเอง

การออกแบบด้วยภาษาชั้นสูงอย่าง Verilog สามารถครอบคลุมลักษณะของการออกแบบได้เกือบทั้งทุกระดับ ทั้งทางด้านโครงสร้าง (Structure) หรือลักษณะการทำงาน (Behavior) ซึ่งถือว่า

เป็นประโยชน์สำหรับการเปลี่ยนหรือย้าย (Transfer) การออกแบบไปยังระดับต่าง ๆ ในแต่ละเฟสของการออกแบบ หรือการทำเอกสารประกอบในการออกแบบ เนื่องจากความเป็นเอกภาพของการออกแบบนั่นเอง

2.1.3 ส่วนประกอบของระบบ (The Anatomy of a System)

2.1.3.1 วงจร และโมดูล (System, Circuit and Module)

ภาษา HDL ถูกใช้ในการอธิบายลักษณะของวงจร หรือระบบอิเล็กทรอนิกส์ ซึ่งโดยทั่วไปแล้วระบบ(System) จะถูกอ้างอิงในภาษา HDL เหล่านี้ในลักษณะเป็น Entity โดยที่ส่วนประกอบของระบบนั้น ได้แก่วงจร (Circuit) ต่าง ๆ ที่ทำงานสอดคล้องกัน ให้ได้เป็นผลลัพธ์อย่างที่ผู้ออกแบบต้องการ แต่อย่างไรก็ตามทั้งความหมายของวงจร หรือระบบนั้นสามารถนำมาใช้แทนกันได้ แล้วแต่การมองของผู้ออกแบบแต่ละคนข้อกำหนดของระบบ (System specification) นั้นจะต้องสามารถอธิบายองค์ประกอบ (Elements) ต่าง ๆในระบบ และความเกี่ยวข้องระหว่างกันให้ได้ สำหรับ Verilog เองนั้น ระบบจะถูกกำหนดได้ด้วยคำสั่ง**module** และจบด้วย **endmodule** เมื่อไรก็ตามที่ระบบทำงาน ระบบจะรับอินพุทข้อมูลเข้ามา และสร้างเอาท์พุทตามฟังก์ชันที่ถูกกำหนดขึ้นมาในระบบนั้น ๆ หรือมองอีกนัยหนึ่งก็คือ ระบบทำการติดต่อ (Interface) กับสิ่งต่าง ๆ รอบตัวมันผ่านทางพอร์ตอินพุท และพอร์ตเอาท์พุท ถ้าปราศจากการอินเทอร์เฟส ระบบก็จะไม่สามารถทำงานได้ ลองคิดดูง่ายๆ ว่าถ้าเรามีคอมพิวเตอร์โดยปราศจากเมาส์ หรือคีย์บอร์ด (ซึ่งถือว่าเป็นอินพุทของระบบ) และมอนิเตอร์(เอาท์พุทของระบบ) นั้นจะเป็นอย่างไรในระบบต่าง ๆ นั้นเราสามารถแยกได้เป็นสองส่วนหลัก ๆ คือ Body และ Interface หน้าที่ของ Body คือทำการประมวลผลข้อมูลที่รับเข้าไป และส่งออกมา ผ่านทางส่วนของการ Interface นอกจากนี้ระบบบางระบบอาจมีส่วนเพิ่มเติม หรือที่เรียกว่า Add-on ที่จะมาใช้งานควบคู่กับระบบที่มีอยู่ ซึ่งถ้าเทียบกับเครื่องคอมพิวเตอร์เครื่องหนึ่งแล้ว ส่วนของ Body คือ Main board นั่นเอง ส่วนของInterface คือ เมาส์และคีย์บอร์ด และในส่วนของ Add-onอาจจะเป็น Sound card หรือ Video card เป็นต้น

2.1.3.2 Interface และ Body ในโมดูล Verilog (Interface and Body in Verilog Module)

เนื่องจาก **module** ใน Verilog ใช้แทนระบบดิจิตอล ดังนั้น **module** จะต้องสามารถแทนส่วนประกอบของระบบได้ นั่นคือ Body และ Interfaceพอร์ตของโมดูลจะถูกกำหนดไว้ในวงเล็บหลังจากชื่อของโมดูล โดยส่วนนี้จะทำหน้าที่เป็น Interface ของระบบ โดยที่รายละเอียดต่าง ๆ ของแต่ละพอร์ต (Direction and type) เช่นเป็นอินพุท หรือเอาท์พุทพอร์ต ขนาดกี่บิต จะถูกกำหนดข้างล่างต่อมา หลังจากวงเล็บปิด คำสั่ง **'include** เสมือนกับ Add-onของระบบที่จะเรียกไฟล์อื่นมาใช้งาน หลังจากนั้นจะเป็นส่วนของ Body ที่จะกำหนดการทำงาน หรือเรียกใช้โมดูลย่อยที่มีอยู่

แล้วมาใช้งานในระบบ ซึ่งมีวิธีการกำหนดการทำงานแตกต่างกันสามแบบคือ Structural, Dataflow หรือ Behavioral

2.1.4 โครงสร้างของโมดูล Module (The Structure of a Module)

สำหรับโมดูล ใน Verilog หมายถึง บล็อกพื้นฐานที่สร้างขึ้นมาสำหรับการกำหนดคุณสมบัติของระบบ ทั้งระบบเล็ก ๆ ที่เป็นเกตเพียงตัวเดียว หรือระบบใหญ่ ที่ประกอบด้วยโมดูลย่อย ๆ หลาย ๆ โมดูล อย่างไรก็ตามไม่ว่าระบบเล็ก หรือใหญ่ ที่อธิบายด้วยโมดูลนั้น จะต้องมีโครงสร้างเหมือนกันทุก ๆ โมดูล จะต้องเริ่มต้นด้วยคำสงวน (Reserve words) คือ **module** ตามด้วยชื่อของโมดูล และจบด้วยคำสงวน **endmodule** (สังเกตว่า ไม่มีช่องว่างระหว่างคำว่า end กับคำว่า module) ภายในตัวโมดูลจะประกอบด้วยสามส่วนดังที่กล่าวมาแล้วข้างต้น คือ

- Module Name : ชื่อที่เรียกโมดูลหรือตั้งไว้เพื่ออ้างอิงการเรียกใช้โมดูล
- Interface : ประกอบด้วยพอร์ต และการกำหนดตัวพารามิเตอร์ของโมดูล
- Body : จะเป็นการกำหนดการทำงานของตัวโมดูล
- Optional add-on : สามารถกำหนดได้ทุกที่ในโมดูล เพื่อเรียกใช้ส่วนประกอบอื่น ๆ ที่เขียนไว้แล้วเป็นไฟล์ มาใช้เพิ่มเติม โดยใช้คำสั่ง 'include'

2.1.4.1 ชื่อของโมดูล (The Name of a Module)

หลักของการตั้งชื่อโมดูลคือ กะทัดรัด และมีความหมายที่สามารถเข้าใจได้ง่าย ที่เป็นการอธิบายว่าโมดูลตัวนั้นทำหน้าที่อะไร เนื่องจากข้อกำหนดของภาษา Verilog จะเป็นภาษาที่คำนึงถึงขนาดตัวอักษรที่ใช้งาน (Case sensitive) ซึ่งไม่เหมือนกับ VHDL ที่ตัวอักษรเล็กใหญ่สามารถใช้แทนกันได้ ดังนั้นเราสามารถใช้ลักษณะตัวอักษรเล็กใหญ่สลับกันเพื่อให้อ่านได้ง่ายขึ้นตัวอย่างเช่น FastBinaryCntr เป็นต้น (สังเกตว่าคำสงวนใน Verilog จะใช้ตัวเล็กเท่านั้น)
โดยสรุป มีข้อกำหนดในภาษา Verilog ในการตั้งชื่อโมดูลดังนี้

- ประกอบด้วย ตัวอักษรภาษาอังกฤษ (Letter) ตัวเลข (Digit) เครื่องหมาย $ (Dollar sign) และ_ (Underscore) เท่านั้น
- ต้องเริ่มต้นด้วยตัวอักษร หรือ Underscore เท่านั้น
- ไม่สามารถเว้นว่างระหว่างตัวอักษรหรือเครื่องหมายได้
- ตัวอักษรเล็ก ใหญ่มีความหมายต่างกัน (Case sensitive)
- ต้องไม่เป็นคำสงวน

2.1.4.2 ส่วน Interface ในโมดูล (The Module Interface)

ในการที่จะกำหนดส่วนของการ Interface ในตัวโมดูล จะต้องประกอบด้วยสองส่วนดังนี้

- Port list: เป็นรายชื่อของพอร์ตที่จะใช้ในตัวโมดูล ซึ่งกำหนดอยู่ในวงเล็บหลังจากชื่อของโมดูลโดยมีเครื่องหมาย Comma (,) ระหว่างพอร์ตต่าง ๆ
- Port declaration: เป็นส่วนที่กำหนดขึ้นมาเพื่ออธิบายคุณลักษณะของพอร์ตต่าง ๆ ที่อยู่ใน

Port list โดยจะเป็นข้อมูลของทิศทางว่าเป็นอินพุท หรือเอาท์พุท หรือทั้งสอง และความกว้างของพอร์ต เป็นต้นสำหรับชื่อของพอร์ตควรตั้งในเหมาะสม เพื่อให้ง่ายในการอ่านทำความเข้าใจ หรืออาจจะเพิ่ม Commentตามหลังแต่ละพอร์ตโดยใช้ One-line comment (//)

2.1.4.3 การเขียนในส่วนของ Body (The Body of a Module)

เนื่องจากความหลากหลายของวงจรที่เราต้องการออกแบบ ทั้งในด้านการทำงาน และในด้านโครงสร้างการออกแบบ ซึ่งบางทีในวงจรชนิดเดียวกันสามารถออกแบบได้หลายวิธีเป็นต้น ดังนั้นภาษาที่ดีควรจะสนับสนุนลักษณะการออกแบบ หรือลักษณะการเขียนอธิบายวงจร (Coding style) ได้หลายแบบ ทั้งแบบที่เหมาะสมสำหรับวงจรเล็ก ๆ ไม่กี่เกต แต่บางทีวิธีการนี้อาจไม่เหมาะสมกับวงจรใหญ่ ๆ ที่มีหลายพันเกตเนื่องจากทำความเข้าใจได้ยากสำหรับภาษา Verilog HDL สามารถมีวิธีการเขียนในการอธิบายวงจร แบ่งได้เป็น 3 ลักษณะตามความต้องการในวงจรแต่ละระดับ (Hierarchy) หรือความถนัดที่แตกต่างกันของผู้ออกแบบ ได้แก่

- Structural design : เป็นการใช้อุปกรณ์ที่มีอยู่แล้วในไลบรารีมาตรฐาน หรือที่เรียกว่า Primitivesและโมดูลย่อย ๆ มาต่อกันโครงสร้าง (Structure) แบบพอร์ตต่อพอร์ต เพื่อให้เป็นระบบขึ้นมา
- Dataflow style : เป็นการส่งผ่านค่าเอาท์พุทที่ขึ้นอยู่กับค่าของอินพุท หลังจากผ่านตัวกระทำหรือฟังก์ชันต่าง ๆ โดยสามารถเห็นเป็นการไหลของข้อมูล (Data flow) ว่าค่าในแต่ละจุดในวงจรนั้นมาจากไหน
- Behavioral style : เป็นการเขียนลักษณะการทำงาน (Behavior) ของวงจรได้โดยตรง

2.1.4.4 การเขียนข้อสังเกต หรือหมายเหตุใน Verilog (Comments in Verilog)

วิธีการเขียนโค้ด (Coding style) ที่ดีนั้นจำเป็นจะต้องคำนึงถึงผู้อ่านในการทำความเข้าใจส่วนต่าง ๆ ของการเขียนด้วย หรือแม้กระทั่งตัวผู้ออกแบบเองในการที่จะกลับมาตรวจสอบ แก้ไขส่วนต่าง ๆ ในภายหลัง ดังนั้นลักษณะการเขียนที่ดีนอกจากจะต้องเขียนให้อ่านง่าย เป็นสัดส่วนแล้ว

ควรจะต้องมีคำอธิบายสั้น ๆ ในแต่ละส่วนของโปรแกรมด้วยสำหรับภาษา Verilog นั้นจะอนุญาตให้เขียนหมายเหตุในตัวโปรแกรม โดยที่ตัวคอมไพเลอร์จะไม่นำไปประมวลผลแต่อย่างใด นั้นคือ

- One-line comment : เริ่มต้นด้วย // และจบด้วยการจบบรรทัด (end-of-line)
- Block comment : จะเริ่มต้นด้วย /* จนกระทั่งถึงจุดที่มีเครื่องหมาย */ ทำให้สามารถกำหนดหมายเหตุได้ครอบคลุมหลาย ๆ บรรทัดเท่าที่ต้องการ

2.1.5 สัญญาณในภาษา Verilog (Signals in Verilog)

2.1.5.1 ค่าที่เป็นไปได้ของสัญญาณ (Available Values of Signals)

ค่าของสัญญาณที่เป็นไปได้ในภาษา Verilog มีเพียงแค่ 4 ค่าตามลักษณะของสัญญาณ และความเป็นไปได้จริง ๆ ในวงจรอิเล็กทรอนิกส์ ซึ่งได้แก่ '0', '1', 'X', 'Z' โดยที่

- '0' คือค่าลอจิกศูนย์ (Logic zero) หรือลอจิกต่ำ(Logic low) หรือค่าที่ไม่จริง (False condition)จากการเปรียบเทียบ
- '1' คือค่าลอจิกหนึ่ง (Logic one) หรือลอจิกสูง(Logic high) หรือค่าที่เป็นจริง (True condition)จากการเปรียบเทียบ
- 'x' หรือ 'X' เป็นลอจิกไม่ทราบค่า (Logic unknown) ที่อาจเกิดจากการไม่ลงรอยกัน(Conflict) ของสัญญาณที่จุด ๆ นั้นที่อาจเป็นไปได้ทั้ง '0' '1' 'Z'
- 'z' หรือ 'Z' เป็นลอจิกที่เกิดจากการเปิดวงจร(Open circuit) ทำให้เกิดสภาวะ High impedance ณ จุดนั้น ๆ

สังเกตได้ว่า ถึงแม้ภาษา Verilog จะเป็น Case sensitive แต่สำหรับสภาวะ unknown หรือ high impedance สามารถใช้ได้ทั้งตัวเล็ก และตัวใหญ่

2.1.5.2 ประเภทของสัญญาณ (Class of Signals)

แต่ละสัญญาณในภาษา Verilog สามารถจัดให้อยู่ในสองประเภทของสัญญาณ คือ nets และ registers

- **Nets** แทนการเชื่อมต่อระหว่างส่วนต่าง ๆ ของวงจร มันไม่สามารถเก็บ (Storage) ค่าของ

สัญญาณไว้ได้ และค่าที่ปรากฏบน nets กำหนดได้โดยตัวขับ (Driver) หรือแหล่งจ่าย (Source) ดังนั้นเมื่อตัดแหล่งจ่ายออกไปจาก nets ก็จะเกิดสภาวะ High impedance 'Z' (ไม่ได้ต่อกับแหล่งจ่ายใด ๆ) เกิดขึ้นที่ nets

- **Registers** ต่างกับ nets คือสามารถเก็บค่าของสัญญาณได้ โดยยังคงค่าเดิมเมื่อไม่ได้ต่อกับ

แหล่งขับ ค่าที่ถูกกำหนดก่อนหน้านี้ สามารถถูกแทนได้ด้วยสัญญาณใหม่ที่เพิ่งเข้ามา ดังนั้น สัญญาณประเภท Registers ก็คล้ายกับรีจีสเตอร์ หรือฟลิปฟลอป (Flip-flop) ที่ใช้งานในวงจรดิจิตอลทั่วไป แต่ต่างกันตรงที่ไม่จำเป็นต้องใช้สัญญาณนาฬิกา (Clock) ในการเก็บค่า

2.1.5.3 สัญญาณสเกลล่าร์ และเวกเตอร์ (Scalar Signals and Vectors)

ถึงตอนนี้เราได้พูดถึงเฉพาะสัญญาณที่มีเส้นเดียว หรือบิตเดียว ที่เราอาจเรียกว่า Scalar signal ซึ่งมีค่าสัญญาณทางลอจิกค่าเดียว ณ เวลาใดเวลาหนึ่ง ตัวอย่างเช่น สัญญาณนาฬิกาที่ใช้เป็นจังหวะ (Synchronize)ของการทำงานในระบบดิจิตอลหลาย ๆ ระบบจำเป็นต้องมีกลุ่มของสัญญาณหลาย ๆ เส้นที่เรียกว่า บัส หรือเวกเตอร์ (Buses or vectors) ซึ่งจะทำการรับส่งข้อมูลที่ประกอบด้วยค่าทางลอจิกที่อยู่ในแต่ละเส้นหลาย ๆ ค่า ตัวอย่างที่มักจะเห็นกันเป็นประจำก็คือ ระบบไมโครโปรเซสเซอร์ เมื่อได้ยินถึง 32-bit microprocessor นั่นหมายถึงว่าบัสข้อมูลมีขนาด 32 bit หรือ 32 เส้นแต่ละบิตในบัสสามารถเข้าถึงได้ทั้งการอ่าน และการเขียน โดยการใช้ตัวชี้(Index) เช่น data[31:0], address[31:0]

2.1.5.4 การกำหนดสัญญาณ (Signal Specification)

ก่อนที่จะทำการใช้ประเภทของสัญญาณในภาษา Verilog เราจำเป็นต้องมีการกำหนด(Declaration)ชนิด และชื่อก่อน ในที่นี้จะพูดถึงการกำหนดสัญญาณประเภท internal ที่อยู่ในส่วนของ body ของโมดูลเท่านั้น ส่วนสัญญาณประเภท external ที่อยู่ในส่วนของการเชื่อมต่อ(Interface) จะพูดถึงในส่วนต่อไปกฎ หรือไวยากรณ์ของการกำหนดการใช้สัญญาณสามารถทำได้โดยง่ายคือ บอกชนิดของสัญญาณก่อนว่าเป็น **wire, tri, wand,**...แล้วตามด้วยชื่อ โดยอาจใส่พร้อม ๆ กันหลาย ๆ ชื่อได้โดยใส่เครื่องหมาย ,(comma) กั้นระหว่างชื่อ เช่น **wire** a, b, c, d; เป็นต้น
สิ่งที่สะดวกอย่างหนึ่งของภาษา Verilog ก็คือเราสามารถกำหนดการใช้สัญญาณได้ทุกที่ บรรทัดใดก็ได้ในส่วนของ body แต่ต้องกำหนดไว้ก่อนการใช้งาน

2.1.5.5 การกำหนดการใช้งานแบบเวกเตอร์ (Vector Specification)

ในการกำหนดการใช้งานแบบเวกเตอร์ เราไม่จำเป็นต้องมีชนิดของสัญญาณขึ้นมาใหม่สำหรับเวกเตอร์แท้จริงแล้ว สัญญาณสเกลล่าร์นั้นอาจถือได้ว่าเป็นกรณีเฉพาะของสัญญาณแบบเวกเตอร์ที่มีเส้นเดียวโดยที่บิตสำคัญมากสุด MSB (Most significant bit) กับบิตสำคัญน้อยสุด LSB (Least significant bit) นั้นอยู่ในตำแหน่งเดียวกันเมื่อสัญญาณแบบเวกเตอร์ประกอบด้วยหลาย ๆ บิต ซึ่งในแต่ละบิตจะต้องสามารถเข้าถึงได้โดยการอาศัยตัวชี้ (Index) สำหรับภาษา Verilog นั้นเรา**สามารถกำหนดตัวเลขชี้บิต (Indexing number) ที่ใช้ได้ทั้งเลขลบ** และเลขบวก กำหนดลงใน [ ]

โดยที่ตัวเลขชี้บิตจะมีอยู่สองตัวสำหรับการกำหนดช่วงโดยจะถูกคั่นด้วยเครื่องหมาย: (Colon) สำหรับตัวเลขด้านซ้ายจะเป็น MSB และตัวเลขด้านขวาคือ LSB ตัวอย่างเช่น wire [0:5] Bus1 ; wire [5:0] Bus2; wire [5:-5] Address; เป็นต้น

2.1.6 สัญญาณภายนอก (External Signals)

2.1.6.1 สัญญาณ Internal เทียบกับ External (Internal vs External Signals)

สัญญาณที่พูดถึงก่อนหน้าที่อยู่ในประเภท Internalที่ใช้ภายในโมดูลในส่วนของ Body และไม่สามารถเข้าถึง (Accessible) ได้จากภายนอกโมดูลถ้าปราศจากสัญญาณ External ที่อยู่ในส่วนของการ Interfaceดังนั้นความแตกต่างระหว่างสัญญาณ Internal และExternal คือ สัญญาณ Internal จะใช้ภายในตัวโมดูลสำหรับการรับส่งข้อมูลต่าง ๆ ส่วนสัญญาณ External จะเป็นตัวที่ติดต่อกับอุปกรณ์ต่าง ๆ ที่อยู่ภายนอกโมดูลโดยจะกำหนดเป็นพอร์ตของโมดูล (Module port)นั่นเอง สำหรับการรับข้อมูลอินพุทเข้ามา หรือส่งข้อมูลเอาท์พุทออกไป

2.1.6.2 พอร์ตของโมดูล (Module Ports)

โมดูลสามารถสื่อสารระหว่างภายใน และภายนอกผ่านทางพอร์ต ในการกำหนดการใช้งานพอร์ตนั้นจะต้องกำหนดความกว้างของพอร์ต และทิศทางของข้อมูลที่จะผ่านพอร์ต (เข้าหรือออกเทียบกับตัวโมดูลนั้นๆ)

ในภาษา Verilog เราสามารถกำหนดทิศทางของพอร์ตได้เป็น

- **Input** : ข้อมูลจะถูกอ่านโดยโมดูลจากภายนอกผ่านทาง input ports เราไม่สามารถเขียนข้อมูลจากภายในโมดูลไปยังพอร์ตชนิดนี้ได้
- **Output** : ข้อมูลจะถูกส่งไปยังภายนอกโมดูลผ่านทาง output ports เราไม่สามารถอ่านข้อมูลเข้ามายังภายในโมดูลผ่านทางพอร์ตชนิดนี้ได้
- **Inout** : ข้อมูลสามารถถูกอ่าน และเขียนผ่านทางพอร์ตชนิดนี้ได้ ดังนั้นจึงถูกเรียกว่า bidirectional ports

2.1.6.3 การกำหนดการใช้งานพอร์ต (The Specification of Ports)

การกำหนดการใช้งานพอร์ตในภาษา Verilog ประกอบด้วยสองส่วนดังนี้

- ชื่อของพอร์ตที่ถูกเรียงอยู่ในส่วนที่เรียกว่า Port list ที่ตามหลังชื่อของโมดูล รายชื่อของพอร์ตต่าง ๆ จะอยู่ในวงเล็บ โดยถูกคั่นด้วยเครื่องหมาย , (comma)
- หลังจากส่วนของ Port list จะต้องตามด้วย Port declaration ที่เป็นตัวกำหนดทิศทางของพอร์ตรวมทั้งขนาดของพอร์ตแต่ละพอร์ต การกำหนดการใช้งานในส่วนนี้จะคล้ายกับการกำหนดการใช้งานของ Internal signals โดยมีไวยากรณ์ดังนี้

Keyword vector range identifier;

โดยที่ *Keyword* คือ *input, output* หรือ *inout*

2.1.6.4 การสร้างรีจิสเตอร์ที่เอาท์พุท (Registered Outputs)

ในการออกแบบระบบใหญ่ ๆ ที่ดีนั้น เรามักจะกำหนดให้ค่าเอาท์พุทของแต่ละโมดูลมีการเก็บค่าไว้ด้วยรีจิสเตอร์ ทั้งนี้เพื่อให้ง่ายในการเชื่อมต่อ และได้ค่าที่เสถียรก่อนที่จะนำไปใช้งานหรือเป็นอินพุทของโมดูลอื่นๆ เนื่องจากเราทราบว่าถ้าเอาท์พุทต่อโดยตรงกับสัญญาณประเภท Nets เช่น wire นั้นจะมีสภาวะเป็น High impedance เมื่อไม่มีตัวขับ หรือแหล่งจ่ายต่ออยู่ ซึ่งสภาวะนี้ไม่เป็นที่ต้องการในการใช้งานวิธีการที่จะทำให้เป็นเอาท์พุทแบบรีจีสเตอร์ สามารถทำได้ง่าย ๆ โดยกำหนด Internal signals แบบ **reg**(Register) และให้สัญญาณเป็นชื่อเดียวกันกับเอาท์พุทพอร์ต

2.1.7 โมดูลใน Verilog (Module in Verilog)

2.1.7.1 การเรียกใช้งานโมดูล (Module Instantiation)

การออกแบบแบบลำดับชั้น (Hierarchical design) ไม่จำเป็นต้องใช้แค่อุปกรณ์พื้นฐานทั่วไป (Primitives)เท่านั้น เรายังสามารถเรียกใช้โมดูลที่เราออกแบบไว้แล้วมาประกอบกันเป็นวงจร หรือระบบขนาดใหญ่ที่ซับซ้อน วิธีการนี้ทำให้เราสามารถตรวจสอบ ค้นหา และทำความเข้าใจได้ง่าย มากกว่าการออกแบบโดยให้ทุกอย่างอยู่ในระดับเดียวกัน หรืออยู่ในโมดูล ๆ เดียว (Flattened design)การเรียกใช้งานโมดูลใน Verilog มีวิธีการเหมือนกับการเรียกใช้อุปกรณ์พื้นฐานต่าง ๆ โดยที่ไม่จำเป็นต้องใช้การเรียนรู้เพิ่มเติม นอกจากนี้การใช้โมดูลย่อย ในการออกแบบ จะมีความยืดหยุ่นในการออกแบบมากกว่าอุปกรณ์พื้นฐาน เนื่องจากสามารถมีหลายอินพุท หลายเอาท์พุท เป็นต้น อีกอย่างคือ โมดูลสามารถเรียกใช้อุปกรณ์พื้นฐานได้ แต่ในทางกลับกันอุปกรณ์พื้นฐานไม่สามารถเรียกใช้โมดูลได้ ดังนั้นการออกแบบโดยใช้โมดูล ที่ประกอบด้วยโมดูลย่อย จึงสามารถทำการออกแบบเป็นลำดับชั้นได้โดยง่าย และสะดวกกว่าข้อจำกัดที่สำคัญอย่างหนึ่งในการใช้งานโมดูลคือ เราไม่สามารถออกแบบโมดูลภายในโมดูลใด ๆ ได้ดังนั้นเราจะต้องเขียนแยกไว้ก่อน หรือหลังตัวโมดูลที่จะเรียกมาใช้งาน โดยสามารถอยู่ในไฟล์เดียวกันได้ หรือแยกเขียนเป็นไฟล์สำหรับโมดูลต่างหาก แล้วใช้คำสั่ง **'include** สำหรับการนำโมดูลนั้นมาใช้งาน

2.1.7.2 หลักการต่อเชื่อมพอร์ต (Port Connection Rules)

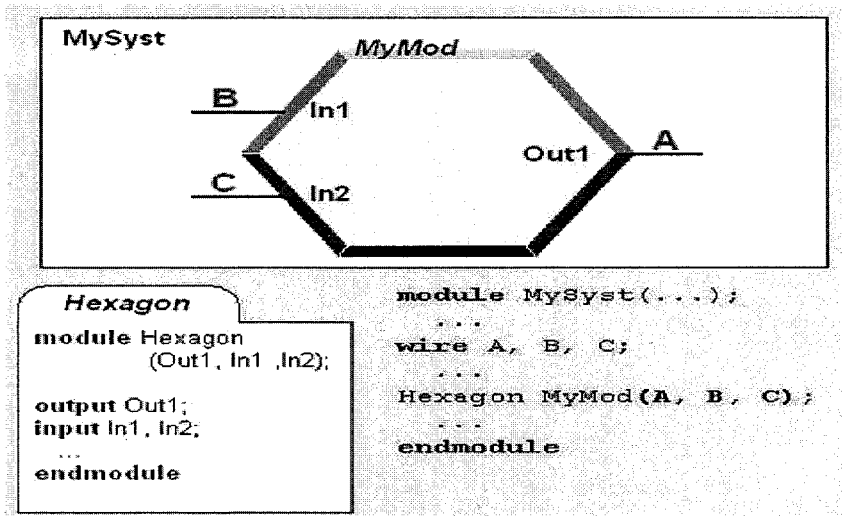
การเชื่อมต่อพอร์ตระหว่างโมดูล กับสิ่งต่าง ๆ รอบตัว แต่ละพอร์ตประกอบด้วยส่วนที่ต่อเชื่อมภายในโมดูล (Internal part) และส่วนที่อยู่ภายนอก (External part) โดยที่ชนิดของสัญญาณส่วนภายในและภายนอกโมดูลที่สามารถเป็นได้ทั้ง **net** และ**reg** มีข้อกำหนดดังแสดงในตารางที่ 2.1

	Internal part	Internal part
INPUT	net	net,reg
OUTPUT	net,reg	net
INOUT	net	net

ตารางที่ 2.1 แสดงตารางการเชื่อมต่อพอร์ตระหว่างโมดูล

2.1.7.3 การเชื่อมต่อพอร์ตโดยลำดับ (Connecting Ports by Ordered Port List)

การเชื่อมต่อสัญญาณไปยังพอร์ตในโมดูลสามารถทำได้โดยใช้ภาษาทำการแมพ (Map) ตามตำแหน่งของขาที่กำหนดไว้ในโมดูล ดังตัวอย่างในรูปที่ 2.1 จะเห็นว่าเราเรียกใช้โมดูลHexagon มาใช้งานในโมดูล MySystและกำหนดให้มีการเชื่อมต่อสัญญาณ A, B, Cเข้าไปที่พอร์ตต่าง ๆ ของโมดูล Hexagon ที่ชื่อว่า MyMod เนื่องจากโมดูล Hexagonกำหนดพอร์ตเรียงลำดับดังนี้คือ (Out1, In1, In2) ดังนั้นตอนที่นำมาใช้งานใน MySyst ในบรรทัด Hexagon MyMod (A, B, C); นั่นหมายถึงว่าตามตำแหน่งแล้ว A จะต่อเชื่อมกับ Out1, B จะต่อเชื่อมกับ In1, และ Cจะต่อเชื่อมกับ In2 ตามลำดับของพอร์ตในโมดูลนั้น ๆ



รูปที่ 2.1 แสดงการเชื่อมต่อพอร์ต

2.1.7.4 การเชื่อมต่อพอร์ตโดยชื่อ (Connecting Ports by Name)

สำหรับการเชื่อมต่อด้วยวิธีการนี้ เราสามารถสลับที่ของพอร์ตในโมดูลที่ถูกเรียกมาใช้งานได้ แต่เราจะต้องกำกับชื่อของพอร์ตไปด้วย ตัวอย่างเช่น Hexagon MyMod (.In1(B), .In2(C), .Out1(A)); โดยมีการเชื่อมต่อของสัญญาณเหมือนกันกับตัวอย่างในรูปข้างบน เพียงแต่ว่าเราสามารถสลับตำแหน่งของพอร์ตโดยมีชื่อพอร์ตกำกับไว้ได้

2.1.8 ตัวแปร และพารามิเตอร์ (Variables and Parameters)

จากที่ทราบมาแล้วว่า กรณีที่มีสัญญาณที่มีค่า ๆ หนึ่งที่ตัวส่ง (หรือที่เราเรียกว่า Source หรือ Driver)แล้วต้องการส่งผ่านค่านี้ไปยังอีกที่ ๆ หนึ่ง เราสามารถกำหนดให้เป็นการทำงานในลักษณะของ Dataflow ได้โดยใช้ Nets ซึ่งถ้าสัญญาณที่ Driver ถูกตัดขาดแล้ว สถานะที่ Nets ที่ใช้ในการรับค่าจะเป็น High impedanceแต่สำหรับ Registers แล้วถึงแม้ว่าสัญญาณที่ Driver จะถูกตัดออกไป แต่มันก็ยังสามารถคงสถานะของค่าสุดท้ายที่ได้รับอยู่ได้คุณสมบัติในการคงค่าได้ของ Registers จำเป็นต้องใช้ในการอธิบายวงจรในแบบของ Behavioral styleที่ส่วนใหญ่เป็นการเขียนอัลกอริทึมในการอธิบายพฤติกรรมการทำงาน และจำเป็นต้องมีการใช้งานของตัวแปร (Variables) ต่าง ๆ อยู่ภายใน คุณสมบัติของตัวแปรโดยทั่วไปคือ สามารถคงค่าได้ตราบเท่าที่ไม่มีการเปลี่ยนแปลงค่าใหม่เกิดขึ้น ดังนั้น Registers ในภาษา Verilog จึงเปรียบเสมือนตัวแปรที่ใช้ในการอธิบายวงจรลักษณะนี้แทนที่จะเป็น Nets

2.1.8.1 รีจีสเตอร์ (Reregisters)

ความหมายโดยทั่วไปของ Registers ในภาษา Verilog ก็คือเป็นตัวที่ใช้ในการเก็บค่าต่าง ๆ ซึ่งเปรียบได้กับ Variables ในภาษาชั้นสูงทั่วไป เช่น C หรือ Pascalวิธีการกำหนดการใช้งานของ Registers จะใช้คำสั่ง **reg** นำหน้าชื่อของสัญญาณที่จะกำหนด สำหรับการใช้งานปกติที่ใช้เก็บค่าต่าง ๆ ของสัญญาณลอจิก นอกจากนี้เรายังสามารถกำหนดการใช้งานของตัวแปรรีจิสเตอร์ประเภทอื่น ๆ สำหรับการใช้งานในภาษา Verilog เพื่อให้มีความสะดวกมากยิ่งขึ้น คือ **integer**,**time**, **real**, **real time** ดังรายละเอียดต่อไปนี้

- **reg** : ใช้ในการเก็บค่าลอจิก คล้ายกับ Flip-flop ในฮาร์ดแวร์ แต่ไม่จำเป็นต้องมีสัญญาณนาฬิกามาใช้ในการเก็บค่า ค่าเริ่มต้นของ register จะเป็น 'x' (ไม่เหมือนกับ Nets ที่เป็น 'z') เช่น
  - o **reg** A;
  - o **reg** [3:0] B, C;

- **time**: เป็นรีจิสเตอร์ประเภทพิเศษ ที่ใช้สำหรับการจำลองการทำงาน (Simulation) การแก้ไข(Debugging) รวมทั้งการรายงาน (Reporting) เป็นตัวแปรที่มีความกว้าง 64 บิต เช่น

  o **time** sim_time;

  o **time** setup_time;

- **integer**: เป็นรีจิสเตอร์ที่ใช้ในทางคณิตศาสตร์ สำหรับการเก็บค่าจำนวนเต็ม สามารถเป็นไปได้ทั้งจำนวนบวก และลบ โดยตัวแปรรีจีสเตอร์นี้มีขนาดเท่ากับ 32 บิต ต่างกับ **reg** คือถ้ามีค่าลบและใช้งานใน Expression ค่าของมันก็ยังเป็นลบอยู่ (ต่างกับ **reg** จะมีค่าเป็นบวกเสมอ) เช่น

  o **integer** loop_count;

  o **integer** counter;

- **real, realtime**: เป็นรีจิสเตอร์ที่ใช้ในทางคณิตศาสตร์ สำหรับการเก็บค่าจำนวนจริงที่มีจุด

ทศนิยม เขียนได้ทั้งเป็นเลขยกกำลังในแบบของ Scientific form หรือจุดทศนิยมธรรมดา โดยที่ **realtime** จะใช้ในการแทนค่าทางเวลา แต่ทั้งสองประเภทนี้สามารถใช้แทนกันได้ เช่น

  o **real** exact, average;

  o **realtime** exact_simtime;

2.1.8.2 เวกเตอร์ และอาร์เรย์ (Vectors and Arrays)

เราสามารถกำหนด **reg** เป็นตัวแปรประเภทเวกเตอร์ (Vector) ได้ โดยกำหนดช่วงบิตที่จะใช้งาน(Indexing) อยู่ในเครื่องหมาย [*MSB*: *LSB*] ที่อยู่ระหว่างคำว่า **reg** และ *Instance name* เช่น **reg** [7:0] Data; //เป็นเวกเตอร์ชื่อ Data ที่มีขนาดความกว้าง 8 บิตอีกประเภทหนึ่งของรีจิสเตอร์ที่สามารถทำได้ คือตัวแปรแบบอาร์เรย์ (Array) – ที่ไม่มีอยู่ในสัญญาณประเภท Nets – ความแตกต่างที่ชัดเจนระหว่างเวกเตอร์ และอาร์เรย์คือ มิติ โดยที่เวกเตอร์มีเพียงมิติเดียวนั้นคือความกว้าง (Width) แต่อาร์เรย์มีสองมิติคือทั้งความกว้าง (Width) และความลึก (Depth) หรืออาจมองอีกนัยหนึ่งคือ อาร์เรย์เป็นการนำเอาเวกเตอร์มาเรียงซ้อน ๆ กันสำหรับวิธีการกำหนดอาร์เรย์ จะมีการกำหนดช่วงความกว้างอยู่หลังจากคำว่า **reg** และความลึกหลังจาก*Instance name* เช่น

reg Data[7:0];

//เป็นการกำหนดอาร์เรย์ชื่อ Data ที่มีความกว้าง 1 บิต และความลึก 8 ตำแหน่ง

reg [7:0] MyMem [3:0];

//เป็นอาร์เรย์ชื่อ MyMem มีความกว้าง 8 บิตและความลึก 4 ตำแหน่ง

ความแตกต่างระหว่างเวกเตอร์ และอาร์เรย์อีกอย่างคือ เวกเตอร์สามารถเป็นรีจิสเตอร์ประเภท **reg** เท่านั้น แต่อาร์เรย์สามารถประกอบด้วยรีจิสเตอร์ประเภทต่าง ๆ เช่น **reg, integer** หรือ **time** แต่ไม่ใช่ **real** หรือ **realtime** สำหรับการอ้างอิงถึงในแต่ละตำแหน่งของอาร์เรย์นั้นสามารถกำหนดได้โดยตรง เช่น MyMem[2];เป็นการอ้างอิงถึงตำแหน่งที่สองของอาร์เรย์ MyMem แต่กรณีที่เราต้องการอ้างอิงลึกลงถึงระดับบิตเราไม่สามารถทำได้โดยตรง เราจำเป็นจะต้องมีตัวแปรเวกเตอร์ชั่วคราว (TempReg) มารับค่าของอาร์เรย์ในตำแหน่งที่ต้องการก่อน แล้วจึงอ้างอิงถึงบิตที่ต้องการในเวกเตอร์ชั่วคราว

2.1.8.3 ค่าคงที่ในภาษา Verilog (Constants in Verilog)

เหตุผลของการกำหนดตัวเลข ๆ หนึ่ง ให้เป็นค่าคงที่ที่มีชื่อ ๆ หนึ่งเพื่อให้ง่ายในการทำความเข้าใจ และสามารถเรียกใช้งานได้หลาย ๆ ครั้ง นอกจากนี้ยังง่ายในการแก้ไขค่า โดยที่เราสามารถเปลี่ยนแปลงค่าได้จากที่เดียว แล้วส่วนอื่นของโปรแกรมที่ใช้ค่าคงที่นั้น ๆก็จะเปลี่ยนแปลงตาม โดยที่เราไม่ต้องเสียเวลาไปไล่เปลี่ยนทีละบรรทัด ซึ่งง่ายในการผิดพลาด และช้าค่าคงที่จะไม่สามารถเปลี่ยนแปลงค่า หรือกำหนดค่าให้ใหม่ได้ในโมดูล ดังนั้นเราจึงไม่สามารถใช้ค่าคงที่ในลักษณะของตัวแปรได้ ทำให้ค่าคงที่ในภาษา Verilog จึงไม่ถูกจัดอยู่ในประเภทของทั้ง Nets และ Registers แต่มันถูกเรียกชื่อใหม่ว่า **parameter**

2.1.9 การอธิบายพฤติกรรมเบื้องต้น (Behavioral Basics)

ในการอธิบายพฤติกรรมการทำงานของวงจร จำเป็นจะต้องกระทำเป็นส่วน ๆ ที่เราเรียกว่า Block โดยมีคำสั่งต่าง ๆ ที่เป็น Procedure statements สำหรับการกำหนดการทำงานในแต่ละส่วนในโมดูลหนึ่ง อาจมีได้หลาย ๆ Blocks ซึ่งแต่ละ Block ทำงานเป็นอิสระต่อกันแบบขนาน (Concurrency) เหมือนกับ Continuous assignments ในบทที่ผ่านมา เมื่อสัญญาณใดสัญญาณหนึ่งใน Block มีการเปลี่ยนแปลง Block นั้นก็จะมีการทำงาน หรือถูกประมวลผล โดยคำสั่งภายในจะถูกแปลตามความหมาย แบบเรียงลำดับ (Sequential) ตามลักษณะของคำสั่งชนิดของ Blocks สามารถแบ่งออกได้เป็นสองลักษณะคือ

- Initial blocks
- Always blocks

เมื่อ Block ใด ๆ มีคำสั่งหลาย ๆ คำสั่ง เราสามารถรวมเป็นกลุ่มของชุดคำสั่งได้ ด้วยคำว่า **begin** แล้วจบด้วย **end** สำหรับการทำงานแบบเรียงลำดับ (Sequential statement execution) หรือคำว่า **fork** และ**joint** สำหรับการทำงานแบบขนาน (Concurrent statement execution)

2.1.9.1 Behavioral blocks

มีลักษณะการทำงานเหมือนกัน ยกเว้น **initial** block จะถูกคอมไพล์ที่ตอนเริ่มต้นของการจำลองการทำงาน (time 0) และกระทำเพียงครั้งเดียวเท่านั้น ส่วน **always** block มีการเริ่มการทำงานที่เวลาเท่ากับ 0 เหมือนกัน แต่จะวนรอบไปเรื่อย ๆ ไม่มีสิ้นสุด (Infinite loop) ตราบเท่าที่กระบวนการจำลองทำงานยังมีอยู่ โดยที่ทุก Blocks ในโมดูลจะทำงานพร้อม ๆ กันจากเวลาเริ่มต้น สำหรับค่าปริยายของรีจิสเตอร์ จะมีค่าเป็น High impedance ('z') ซึ่งในการจำลองการทำงานให้ถูกต้องจำเป็นต้องเซตค่าเริ่มต้นให้ก่อน อาจจะเป็น '0' หรือค่าที่เหมาะสม โดยใช้คำสั่ง **initial** นั่นเอง ส่วนหน้าที่ของคำสั่ง **always** เป็นการเริ่มต้นการอธิบายพฤติกรรมการทำงานของวงจรในโมดูลถึงแม้ว่าในแต่ละ Blocks จะทำงานพร้อม ๆ กัน แต่ในการเขียนเป็นโปรแกรมเราจำเป็นต้องเขียนเรียงกันลงมาทีละบรรทัด เนื่องจากเราไม่สามารถเขียนทุก
Blocks ได้พร้อม ๆ กัน

2.1.9.2 การกำหนดค่าตัวแปร (Variable Assignment)

สิ่งสำคัญที่สุดในการกระทำในการอธิบายพฤติกรรมการทำงาน คือการส่งผ่านค่าจาก Expression ไปยังตัวแปร การส่งผ่านค่านี้เราเรียกว่า Procedural assignment ซึ่งแตกต่างจาก Continuous assignmentดังต่อไปนี้

- ตัวแปรทางด้านซ้ายมือของการส่งผ่านค่า (The target of assignment) จะต้องเป็น Registers

(**reg**, **integer**, **real**, **time**), บิต หรือส่วนหนึ่งของรีจิสเตอร์เวกเตอร์ โดยที่ไม่สามารถเป็น Nets ได้

- ไม่มีคำว่า **assign** ในการส่งผ่านค่า
- ต้องกำหนดภายใน Behavioral block (**initial** or **always**)
- ต้องเป็นสัญญาณชนิดเดียวกัน
- การส่งผ่านค่าแบบ Continuous assignment จะมีการเปลี่ยนแปลงค่าใหม่เมื่อมีการเปลี่ยนแปลงค่าของสัญญาณใดสัญญาณหนึ่งใน Expression ทางด้านขวามือ แต่สำหรับ Procedural assignment ใน **always** จะมีการประมวลผลค่าใหม่เมื่อมีการเปลี่ยนแปลงค่าของสัญญาณใดสัญญาณหนึ่งใน Sensitivity list ที่อยู่ข้างหลัง **always**

2.1.10 การควบคุมพฤติกรรมของวงจรขั้นสูง (Advance Control over Behavior)

2.1.10.1 เหตุการณ์ (Events)

เราใช้เครื่องหมาย @ สำหรับการกำหนดการเกิดขึ้นของเหตุการณ์ (Event control statement) สามารถทำได้โดยใช้ @ ตามด้วยชื่อของ Registers หรือ Nets ดังตัวอย่างต่อไปนี้

@ (posedge CLK) Q = D ;

@ (negedge CLK) Q = D ;

posedge และ **negedge** เป็นคำเริ่มต้น (Prefix) ที่กำหนดเหตุการณ์ในช่วงขอบขาขึ้น หรือขอบขาลงของสัญญาณนั้น ๆ เช่น จะทำการส่งผ่านค่า (Assignment) Q = D; ที่ขอบขาขึ้นของ CLK เป็นต้น

2.1.10.2 คำสั่งรอ (Wait Statement)

การควบคุมเหตุการณ์ (Event control) เป็นการควบคุมการเปลี่ยนแปลงค่าค่าสัญญาณ หรือตัวแปร โดยอาจจะมีเงื่อนไข หรือช่วงเวลามาเกี่ยวข้อง

คำสั่ง **wait** สามารถนำมาใช้ควบคุมการกระทำอย่างมีเงื่อนไข เช่นอาจใช้สร้างตัว 3-state buffer หรือตัวLevel-sensitive event control สำหรับการใช้งานจะต้องมีเงื่อนไขการรอ (Enable) อยู่ข้างในวงเล็บหลังจากคำสั่ง **wait** แล้วตามด้วย Statements

Wait (enable) statement;

เช่น wait (EN) #5 C=A+B; สามารถแปลได้ดังนี้คือ การกระทำ C = A+B; จะเกิดขึ้น
ภายหลัง 5 หน่วยเวลา หลังจากที่สัญญาณ EN เป็น '1'
แต่ถ้าใช้ wait; โดยไม่มีเงื่อนไขใด ๆ ทั้งสิ้น จะเป็นการให้รอโดยไม่มีที่สิ้นสุด หรือเป็นการหยุดการจำลองการทำงานนั่นเอง

2.1.10.3 Sensitivity List

เมื่อไรก็ตามที่คำสั่งถูกกระทำโดยเหตุการณ์ที่เกิดขึ้นในสัญญาณหนึ่ง เรากรณีนี้ว่าคำสั่งนั้นไว(Sensitive) ต่อสัญญาณนั้น ๆ ซึ่งในภาษา Verilog เองไม่จำกัดว่าการไวต่อสัญญาณจะเป็นสัญญาณเดียวหรือหลาย ๆ สัญญาณ ตามแต่ที่จะปรากฏในคำสั่ง หรือชุดคำสั่งนั้น ๆ และเราสามารถกำหนดรายชื่อของสัญญาณต่าง ๆ เหล่านั้นได้ (โดยใช้ **or** เป็นตัวกั้น) เรียกว่า *Sensitivity list*

การกำหนด Sensitivity signals ส่วนมากจะใช้ในคำสั่ง **always** สำหรับการควบคุมการทำงานของชุดคำสั่งต่าง ๆ ที่อยู่ระหว่าง **begin** และ **end** ใน **always** block นั่นคือ ถ้าหากตัวใดตัวหนึ่งใน Sensitivity list เกิดการเปลี่ยนแปลง ตัว **always** block ก็จะมีการประมวลผล และปรับปรุงค่าเกิดขึ้น

2.1.10.4 การส่งผ่านค่าแบบ Non-blocking (Non-blocking Assignment)

การส่งผ่านค่าแบบ Non-blocking assignment เป็นวิธีการกำหนดการส่งผ่านค่าในทุก ๆ บรรทัดที่เขียนเรียงกันนั้น เกิดขึ้นพร้อม ๆ กัน การใช้งานในลักษณะนี้จะใช้เครื่องหมาย '<=' แทนที่จะเป็น '=' (Blocking assignment)

เหตุผลที่เราต้องการ Non-blocking assignment เพื่อให้ทุก ๆ การส่งผ่านค่าเกิดขึ้นพร้อม ๆ กันเป็นลักษณะของ Concurrent data transfer ซึ่งสามารถหลีกเลี่ยงการเกิด Race condition จากการใช้ Blocking assignment

ตัวอย่างการทำงานของ **always** block ที่มีการใช้ Blocking และ Non-blocking assignment จะเห็นได้ว่า Blocking assignment จะมีการ Update ค่าทันทีหลังจากจบบรรทัด แต่ Non-blocking การ Update ค่าของตัวแปรต่างๆจะเกิดขึ้นพร้อม ๆ กันหลังจากเจอคำสั่ง **end**

2.2 การใช้งาน TLC 5941

2.2.1 คุณสมบัติของ TLC 5941

- เอาท์พุทขนาด 16 ช่อง
- โหมด Grayscale ควบคุมด้วย PWM มีขนาด 12 บิต (4096 ระดับ)
- โหมด Dot Correction สามารถควบคุมกระแสได้ขนาด 6 บิต(64 ระดับ)
- สามารถจ่ายกระแสได้ตั้งแต่ 0 mA ถึง 80 mA
- สามารถจ่ายแรงดันให้หลอด LED ได้ถึง 17 โวลต์
- แรงดันไฟที่ใช้มีขนาด 3 V ถึง 5.5 V
- การติดต่อส่งข้อมูลเป็นแบบอนุกรม
- อัตราการส่งผ่านข้อมูลสูงสุดที่ความถี่ 30 MHz
- สามารถตรวจสอบความผิดพลาดที่เกิดขึ้นจากอุณหภูมิ
- สามารถตรวจสอบและป้องกันหลอด LED

2.2.2 การประยุกต์ใช้งาน

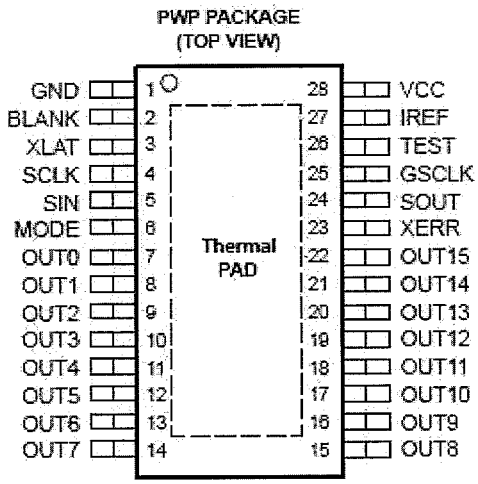
- ใช้แสดงผลแบบ Monocolor และ Multicolor
- ใช้ควบคุมการผสมสีต่างๆแสดงผลทางหลอด LED
- ใช้ในการควบคุมระดับความสว่างของหลอด LED

TLC 5941 มี 2 โหมดในการทำงานคือ

1. โหมดโหมด Grayscale จะใช้ PWM ในการควบคุมความสว่างของเอาท์พุททั้ง 16 ช่อง และมีสัญญาณขนาด 12 บิต หรือสามารถควบคุมความสว่างได้ 4096 ระดับ
2. โหมด Dot Correction สามารถควบคุมกระแสของแต่ละช่องได้โดยมีขนาด 6 บิต หรือสามารถ จ่ายกระแสได้ 64 ระดับและจะปรับความสว่างของหลอดLED โดยการจ่ายกระแส

ทั้งสองโหมดนี้จะติดต่อส่งข้อมูลแบบอนุกรม และ TLC5941 จะต้องต่อตัวต้านทานภายนอกเพื่อเป็นการกำหนดค่ากระแสต่ำสุดทั้ง 16 ช่องสัญญาณเอาท์พุทและ TLC5941 มีคุณสมบัติในการตรวจสอบความผิดพลาดของวงจรภายในโดยสามารถแสดงจุดที่หลอดLED เกิดการเสียหายที่ช่องสัญญาณใดและสามารถตรวจสอบความผิดพลาดในการแสดงผลเนื่องจากผลของอุณหภูมิ

2.2.3 หน้าที่ของขาต่างๆ



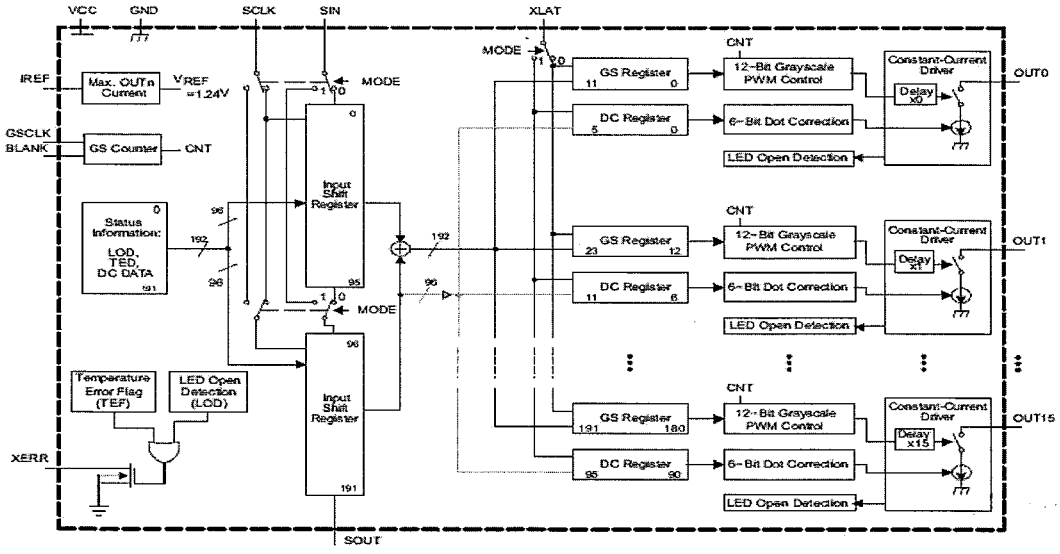
รูปที่ 2.2 แสดงขา TLC 5941

- ขา GND ทำหน้าที่ ต่อลงกราวด์
- ขา BLANK ทำหน้าที่ เมื่อขา BLANK เป็น “H” สัญญาณเอาท์พุททุกช่องจะหยุดทำงาน GS counter จะถูก Reset เมื่อขา BLANK เป็น “L” สัญญาณเอาท์พุททุกช่องจะถูกควบคุมโดย Grayscale PWM
- ขา XLAT ทำหน้าที่ เป็นสัญญาณการกระตุ้นแบบ Latch โดย เมื่อขา XLAT เป็น “H” TLC5941 จะทำการเขียนข้อมูลจากอินพุท Shift register ส่งไปยัง GS register เมื่อ

(MODE = “L”) หรือส่งไปยัง DC register เมื่อ (MODE = “H”) เมื่อขา XLAT เป็น “L” ข้อมูลใน GS register หรือ DC register จะคงเดิมไม่มีการเปลี่ยนแปลง

- ขา SCLK ทำหน้าที่ เลื่อนข้อมูลจากอินพุทแบบอนุกรมมาเก็บใน Shift register
- ขา SIN ทำหน้าที่ เป็นขาสัญญาณอินพุทติดต่อรับข้อมูลแบบอนุกรม
- ขา MODE ทำหน้าที่ เป็นขาเลือกโหมดการทำงานเมื่อ MODE = GND จะเป็นการเลือกใช้ GS mode MODE = VCC จะเป็นการเลือกใช้ DC mode
- ขา OUT0-OUT15 ทำหน้าที่ เป็นขาเอาท์พุท
- ขา XERR ทำหน้าที่ เป็นขาตรวจสอบความผิดพลาดของเอาท์พุท
- ขา SOUT ทำหน้าที่ เป็นขาที่ส่งสัญญาณเอาท์พุทแบบอนุกรม
- ขา GSCLK ทำหน้าที่ เป็นขาอ้างอิงสัญญาณนาฬิกาเพื่อควบคุมสัญญาณGrayscalePWM
- ขา TEST ทำหน้าที่ เป็นทดสอบ จะต้องต่อกับ VCC
- ขา IREF ทำหน้าที่ เป็นขาอ้าอิงของกระแสเอาท์พุท
- ขา VCC ทำหน้าที่ เป็นขาที่ต่อกับแหล่งจ่าย

2.2.4 บล็อกไดอะแกรม

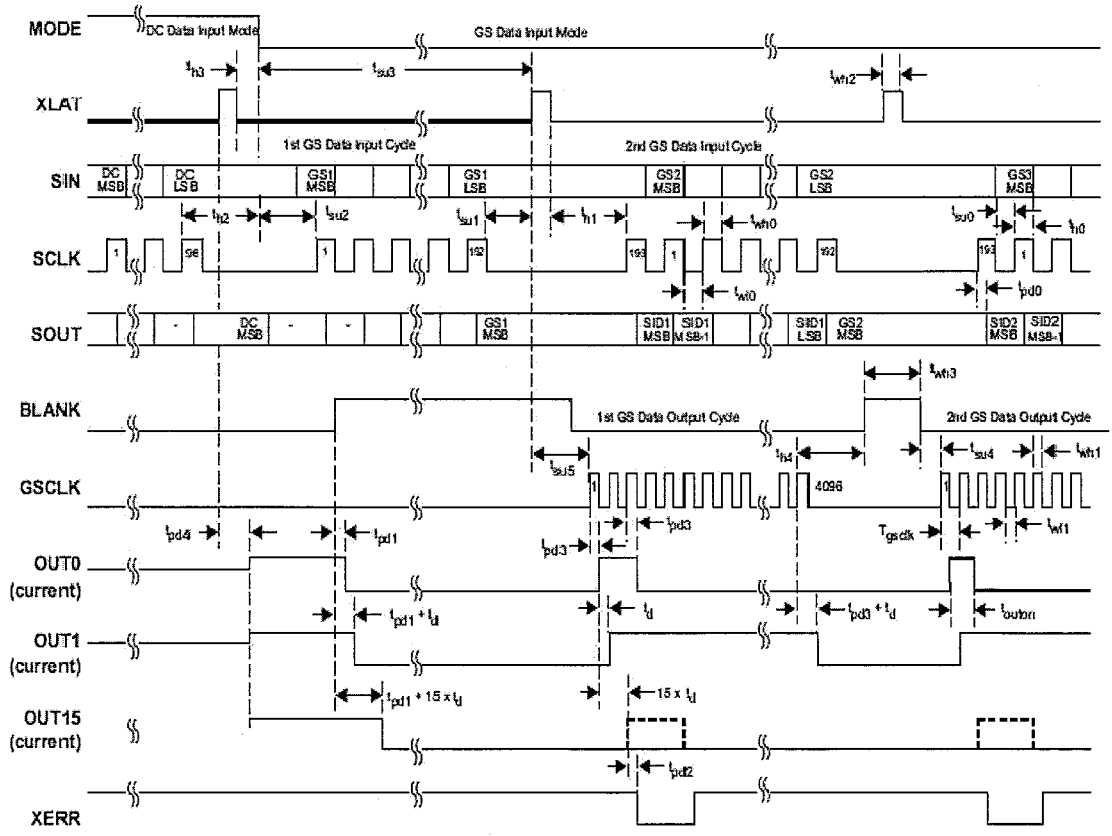


รูปที่ 2.3 แสดงบล็อกไดอะแกรม TLC 5941

จากรูป 2.3 เมื่อมีสัญญาณอินพุทส่งเข้ามาแบบอนุกรมทางขา SIN โดยสัญญาณอินพุทจะต้องรอสัญญาณSCLK เพื่อเป็นตัวเลื่อนข้อมูลมาเก็บไว้ที่ Shift register ภายในตัว TLC5941 โดยที่ MODE = “H” Shift register จะมีขนาด 96 บิต และจะรอจนกว่าสัญญาณ XLAT = “H” จะทำงานส่งข้อมูลไปเก็บที่ DC register ซึ่งมีขนาด 6 บิตต่อหนึ่งช่องสัญญาณ และค่ากระแสที่จะจ่ายให้กับหลอด LED จะขึ้นอยู่กับข้อมูลขนาด 6 บิต

MODE = “L” Shift register ภายในตัว TLC5941 จะมีขนาด 192 บิต เมื่อมีสัญญาณเข้ามาใน Shift register จนครบ 192 บิต จากนั้นจนกระทั่งสัญญาณ XLAT = “H” จะทำการนำข้อมูลส่งไปยัง GS register ซึ่งมีขนาด 12 บิตต่อหนึ่งช่องสัญญาณ ซึ่งข้อมูลขนาด 12 บิต เป็นข้อมูลค่าระดับความเข้มของสี ส่วนสัญญาณ GSCLK กับสัญญาณ BLANK จะทำหน้าที่ควบคุมโหมด Grayscale PWM โดยที่สัญญาณ BLANK =”L” และมีสัญญาณนาฬิกาจาก GSCLK เข้ามาสู่วงจรภายใน GS counter จะได้ค่าที่เป็นตัวกำหนดความสว่างของทุกช่องสัญญาณเอาท์พุท

2.2.5 การติดต่อส่งข้อมูลแบบอนุกรม



รูปที่ 2.4 แสดงไทม์มิ่งไดอะแกรมของการส่งข้อมูลแบบอนุกรม

TLC5941 มีการติดต่อส่งข้อมูลแบบอนุกรมได้หลายทางซึ่งสามารถติดต่อกับตัวประมวลผลสัญญาณในทางดิจิตอลต่างๆได้หรือติดต่อกับไมโครคอนโทรลเลอร์ได้ โดยที่ต้องการสัญญาณอินพุทเพียง 3 ขา ได้แก่สัญญาณขอบขาขึ้นของสัญญาณ SCLK เพื่อเป็นตัวเลื่อนข้อมูลจากอินพุทมาเก็บไว้ที่" Shift register ภายในตัว TLC5941 ซึ่งมีขนาด 96 บิต หรือ 192บิตจะขึ้นกับการเลือกโหมดในการเขียนโปรแกรม สัญญาณ XLAT เป็น High จะเป็นสัญญาณกระตุ้นเพื่อรับสัญญาณอินพุทชุดใหม่เข้ามาโดยที่สัญญาณที่เข้ามาก่อนจะเป็นบิตสำคัญสูงสุด ข้อมูล Grayscale และ ข้อมูล Dot correction สามารถเข้ามาระหว่างช่วงสัญญาณGSCLK ทำงานได้ถึงแม้ว่าจะมีสัญญาณข้อมูล Grayscaleชุดใหม่เข้ามาก็ตามและสัญญาณ XLAT ควรจะเป็น High ก็ต่อเมื่อสัญญาณ Gray scale สิ้นสุดซึ่งสามรถดูในรูปที่ 2.5 ประกอบ เราสามารถติดต่อ TLC5941 ได้ตั้งแต่ 2 ตัวขึ้นไปโดยการต่อจากขา SOUT ไปยัง TLC5941 อีกตัวหนึ่งได้

2.2.6 การกำหนดค่ากระแสสูงสุดของแต่ละช่องสัญญาณเอาท์พุท

ค่ากระแสสูงสุดของแต่ละช่องสัญญาณเอาท์พุทสามารถกำหนดได้โดยการใช้ตัวต้านทาน $R_{(IREF)}$ ซึ่งจะต่อระหว่างขา IREF และกราวด์ แรงดัน IREFจะถูกกำหนดมาเรียบร้อยโดยค่าเท่าไปคือ 1.24 โวลต์โดยที่กระแสสูงสุดหาได้จากกระแสที่ไหลผ่านตัวต้านทานคูณกับค่า Factor

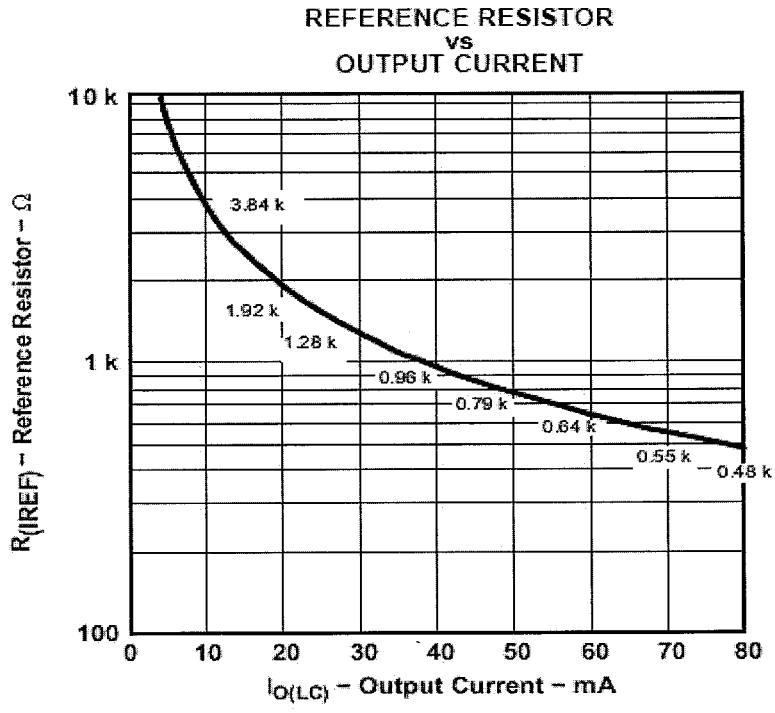
$$\text{Imax} = \frac{V(IREF)}{R(IREF)} \quad 31.5 \qquad (1)$$

เมื่อ

V(IREF) = 1.24 V

R(IREF) = ค่าความต้านทานภายนอก

Imax จะต้องมีค่าระหว่าง 5 mA ถึง 80 mA ถ้ากระแส Imax ต่ำกว่า 5 mA อาจจะทำให้กระแสที่เอาท์พุทไม่เสถียรและกระแสเอาท์พุทที่ต่ำกว่า 5 mA สามารถปรับค่าให้สูงขึ้นเมื่อเลือกใช้โหมด Dot correctionได้



รูปที่ 2.5 แสดงกราฟการอ้างอิงค่าความต้านทานต่อกระแสที่เอาท์พุท

2.2.7 การกำหนดค่าในโหมด DOT CORRECTION

TLC5941 สามารถปรับค่ากระแสของแต่ละช่องสัญญาณเอาท์พุทได้อย่างละเอียดโดยแยกอิสระจากกัน ความสามารถนี้เรียกว่า " DOT CORRECTION " ซึ่งคุณสมบัตินี้สามารถปรับความสว่างของหลอด LED ที่ต่อเข้ากับช่องสัญญาณแต่ละช่อง โดยแต่ละช่องสัญญาณเอาท์พุทซึ่งจะสามารถโปรแกรมค่าได้และจะมีขนาด 6 บิตต่อหนึ่งช่องสัญญาณ แต่ละช่องสามารถปรับระดับได้ 64 ระดับ จาก 0% ถึง 100% ของค่ากระแสสูงสุด Imax และขา TEST จะต้องต่อกับแหล่งจ่าย VCC เพื่อที่จะทำให้ DOT CORRECTION ทำงานได้อย่างถูกต้อง

การหาค่ากระแสของแต่ละช่องสัญญาณเอาท์พุท

$$IOUTn = Imax \frac{DCn}{63} \qquad (2)$$

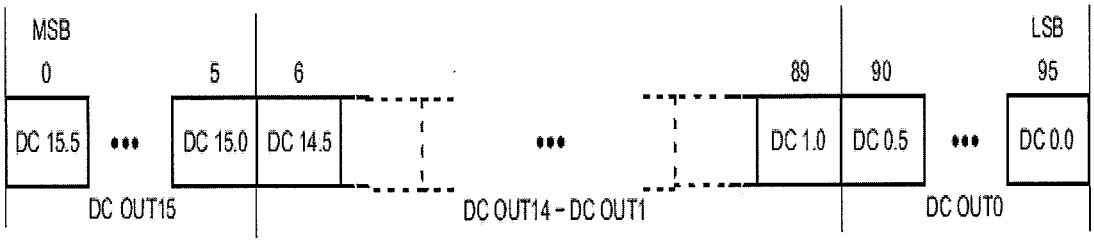
เมื่อ

Imax = ค่ากระแสที่มีค่าสูงสุด

DCn = ค่าของ DOT CORRECTION ที่โปรแกรมไว้ของแต่ละช่องสัญญาณเอาท์พุท n
(DCn = 0 ถึง 63)

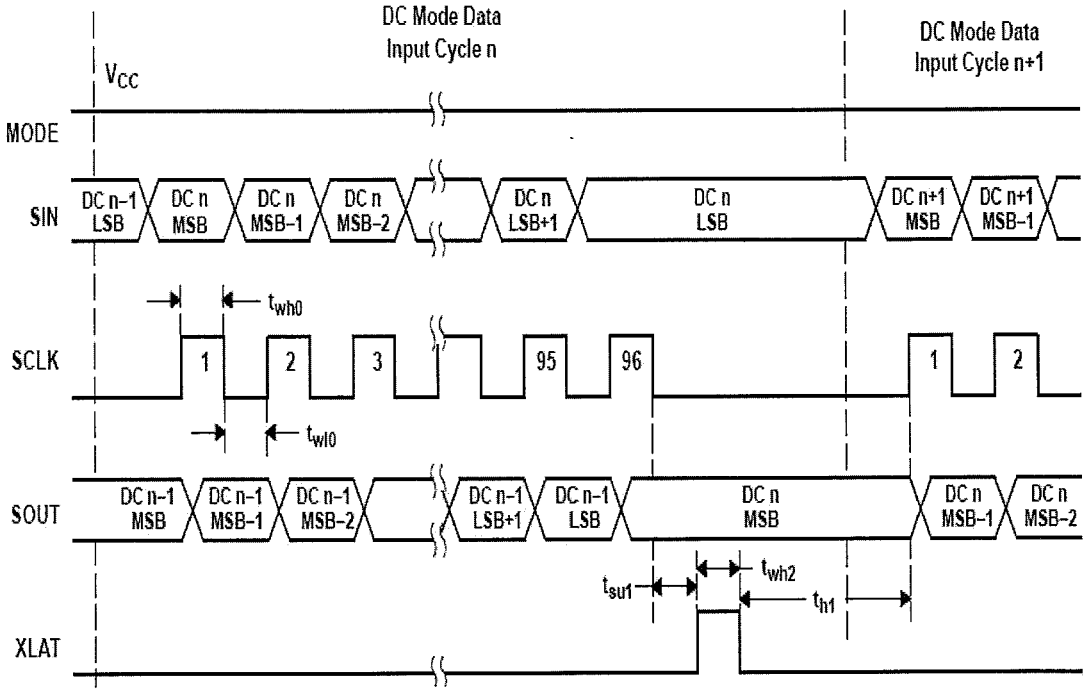
n = 0 ถึง 15

รูปที่ 2.6 เป็นการแสดงรูปแบบข้อมูลของ DOT CORRECTION ซึ่งมีขนาด 6 บิต x 16 ช่อง สัญญาณ รวมเป็น 96 บิต รูปแบบนี้คือ Big-Endian โดยวิธีนี้คือจะส่งบิตข้อมูลที่เป็น MSB เข้าไปก่อนและตามด้วย MSB-1 โดย DC15.5 ย่อมาจากบิตที่ 5 = MSB ของช่องสัญญาณเอาท์พุทที่15



รูปที่ 2.6 แสดงรูปแบบชุดข้อมูลแบบ DOT CORRECTION

เมื่อ MODE ถูกกำหนดเป็น VCC หรือ “H” TLC5941 จะเข้าสู่โหมด Dot correction ดังนั้น ขนาดอินพุทของ Shift register ภายในจะมีขนาดเป็น 96 บิต เมื่อข้อมูลเข้ามาใน Shift register จนครบ TLC5941 จะเขียนข้อมูลไปยัง DC register เมื่อสัญญาณที่ขา XLAT เป็น High เมื่อ XLAT เป็น Low TLC5941 จะทำการเก็บข้อมูลไปไว้ที่ DC register และในตอนที่สัญญาณ XLAT มีค่าเป็น High สัญญาณ SCLK และ SIN ห้ามมีการเปลี่ยนแปลง หลังจากที่สัญญาณ XLAT เป็น Low ค่าใน DC register จะไม่มีการเปลี่ยนแปลงจะคงสถานะไว้ และเมื่อ XLAT เป็น High อีกครั้งจะเป็นการนำข้อมูล Dot correction ชุดใหม่มาแสดงที่เอาท์พุทแทนข้อมูลชุดเก่าทำให้ค่ากระแสที่เอาท์พุทมีการเปลี่ยนแปลงตามข้อมูลอินพุทที่ป้อนเข้ามา ซึ่งสัญญาณที่ขา BLANK จะมีการเปลี่ยนแปลงเป็น High หรือ Low ก็ไม่มีผลต่อการรับข้อมูลหรือแสดงผลที่ทางเอาท์พุท



รูปที่ 2.7 แสดงไทม์มิ่งไดอะแกรมของข้อมูลอินพุท Dot Correction

2.2.8 การกำหนดค่า GRAYSCALE

TLC5941 สามารถปรับความสว่างของแต่ละช่องสัญญาณเอาท์พุทโดยการใช้ PWM ในการควบคุมโดยใช้ขนาด 12 บิตต่อช่องสัญญาณเอาท์พุท หรือสามารถปรับระดับความสว่างได้ 4096 ระดับ จาก 0% ถึง 100%

ความสว่างของแต่ละเอาท์พุทสามารถหาได้จาก

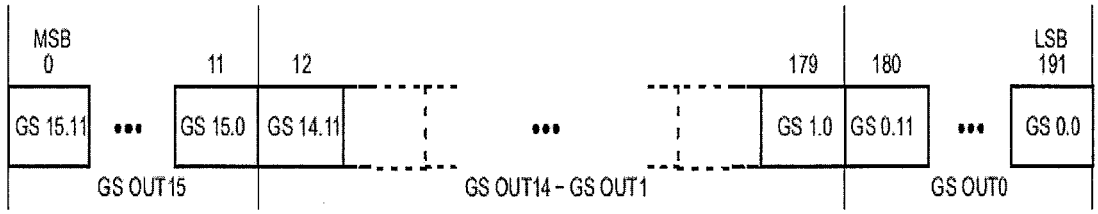
$$\text{Brigthness in \%} = \frac{GSn}{4095} \quad 100 \qquad (3)$$

เมื่อ

GSn = ค่า Grayscale ที่โปรแกรมไว้ของแต่ละเอาท์พุท n (GSn = 0 ถึง 4095)

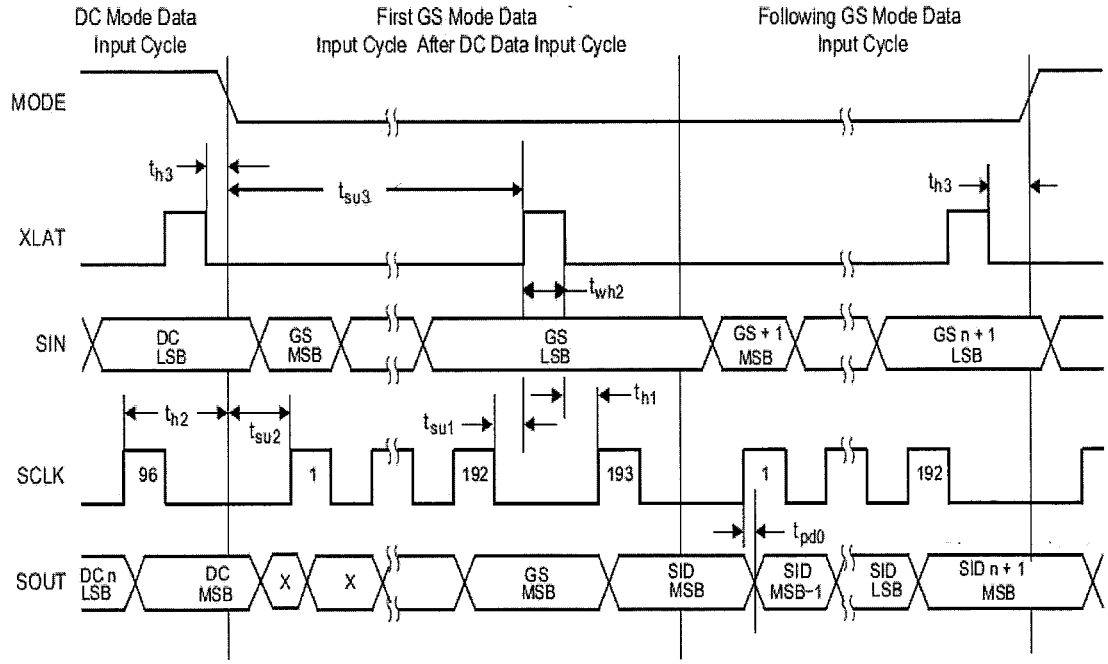
n = 0 ถึง 15

รูปที่ 2.8 เป็นการแสดงข้อมูล Grayscale ทั้งหมด ซึ่งเมื่อข้อมูลจาก Shift register ถูกส่งมายัง Grayscale register ข้อมูลทั้งหมดจะถูกส่งเข้ามาพร้อมกัน รูปแบบข้อมูล Grayscale จะประกอบด้วย 16 x 12 บิต จะเท่ากับ 192 บิตในหนึ่งชุดข้อมูลโดยที่สัญญาณที่เป็น MSB จะต้องถูกส่งเข้ามาเป็นสัญญาณแรก



รูปที่ 2.8 แสดงรูปแบบชุดข้อมูลแบบ GRAYSCALE

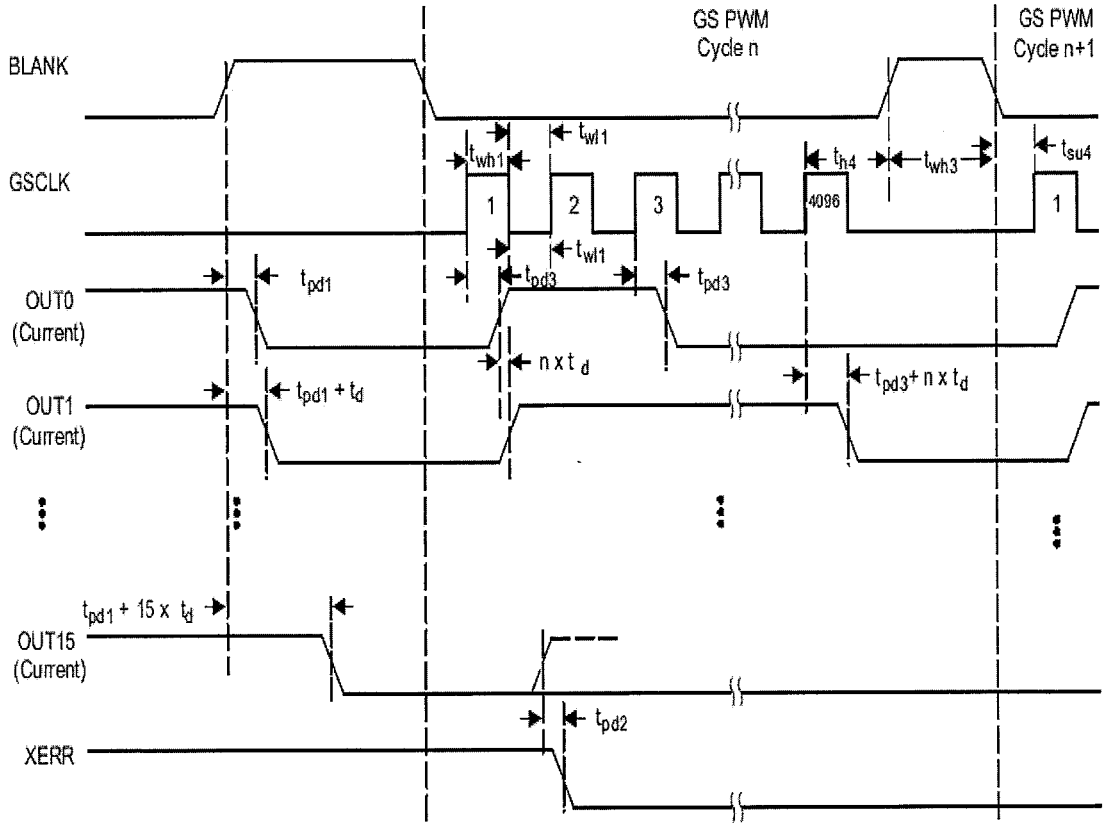
เมื่อขา MODE ถูกกำหนด เป็น Low หรือ GND TLC5941 จะเข้าสู่โหมด Grayscale ดังนั้น ขนาดอินพุทของ Shift register ภายในจะมีขนาดเป็น 192 บิต เมื่อข้อมูลเข้ามาใน Shift register จนครบTLC5941 จะเขียนข้อมูลไปยัง GS register เมื่อสัญญาณที่ขา XLAT เป็น High และเมื่อมีข้อมูล Grayscale ชุดใหม่เข้ามาและ XLAT เป็น High ข้อมูลชุดนี้จะเป็นข้อมูลที่ถูกต้องเมื่อดูการไทม์มิ่งไดอะแกรมประกอบและสัญญาณ XLAT ควรที่จะเป็น High เมื่อสัญญาณส่งครบ 192 บิต และสัญญาณ BLANK ควรเป็น High เมื่อข้อมูล Grayscale ครบ 192 บิตเช่นกัน เมื่อสัญญาณแรกของข้อมูล Grayscale หลังจากข้อมูล Dot Correction จะต้องการสัญญาณ SCLK เพิ่มขึ้นหลังจากสัญญาณ XLAT ในตอนที่ MODE = High ซึ่งจะทำให่มข้อมูลมีความสมบูรณ์ และข้อมูล Grayscale จะถูกแทนที่ด้วยข้อมูลใหม่จาก Shift register เมื่อมีข้อมูลชุดใหม่เข้ามา



รูปที่ 2.9 แสดงไทม์มิ่งไดอะแกรมของข้อมูลอินพุท Grayscale

2.2.9 การทำงานของ GRAYSCALE PWM

Grayscale PWM จะเริ่มทำงานที่ขอบขาลงของสัญญาณ BLANK เมื่อสัญญาณ BLANK เป็น Low แล้ว จะเกิดสัญญาณ GSCLK ลูกแรกและจะเพิ่มขึ้นโดยวงจร Grayscale counter โดยการเพิ่มขึ้นหนึ่งครั้งจะเป็นการสวิตช์ทุกช่องสัญญาณให้ติดค่า Grayscale จะไม่เท่ากับศูนย์และทุกๆ ขอบขาขึ้นของ GSCLK จะทำการเพิ่มค่าขึ้นทีละหนึ่ง TLC5941 จะทำการเปรียบเทียบค่า Grayscale ของแต่ละช่องสัญญาณเอาท์พุทกับค่า Grayscale counter ค่า Grayscale ของแต่ละช่องสัญญาณเอาท์พุทจะมีค่าเท่ากันและจะหยุดทำการนับเมื่อสัญญาณ BLANK เป็น High หรือหลังจากสัญญาณ GSCLK มีค่าเท่ากับ 4096 จะทำการรีเซตค่าของวงจร Grayscale counter เป็นศูนย์ จากรูป 2.10เมื่อค่า Grayscale counter นับถึงค่า FFFh จะหยุดนับและสัญญาณเอาท์พุทจะดับลงเมื่อสัญญาณ BLANK เป็น High ก่อนที่ Grayscale counter จะนับถึงค่า FFFh จะถูกรีเซทค่าให้เป็นศูนย์



รูปที่ 2.10 แสดงไทม์มิ่งไดอะแกรมของข้อมูลอินพุท Grayscale PWM

2.2.10 ช่วงเวลาที่เอาท์พุททำงาน

จำนวนช่วงเวลาแต่ละเอาท์พุททำงานในฟังก์ชั่นสัญญาณนาฬิกา Grayscale และทำการโปรแกรมค่า Grayscale PWM ช่วงเวลาในการทำงานของแต่ละเอาท์พุทสามารถคำนวณได้จาก

$$T\_on_n = \frac{GSn}{F(gsclk)} \quad ton\_err \qquad (4)$$

เมื่อ

T_on_n = เวลาที่ OUTn ทำงาน

GSn = OUTn ที่โปรแกรมค่า Grayscale PWM มีค่าระหว่าง 0 ถึง 4095

ton_err = ค่าเวลาที่เอาท์พุทเกิดการผิดพลาด ดูได้จากตารางคุณสมบัติการสวิตช์

สมการที่ 4 ใช้เมื่อความถี่ GSCLK สูงมากๆ และค่า Grayscale PWM ต่ำมากๆ ซึ่งค่าเวลา T_on_n จะติดลบซึ่งถ้าค่าT_on_n ติดลบจะทำให้เอาท์พุทไม่ทำงาน

ยกตัวอย่าง ถ้า GSCLK = 30 MHz, GSn = 1 และค่า ton_err ทั่วไปเท่ากับ 50 nS เมื่อคำนวณออกมาจะได้ค่าเวลา T_on_n = -16.6 nS ดังนั้นสัญญาณที่เอาท์พุทจะไม่ทำงานภายในสภาวะนี้ ควรเพิ่มค่าGrayscale PWM หรือลดค่า GSCLK ลงเพื่อให้เอาท์พุททำงาน

2.2.11 อัตราการส่งผ่านข้อมูลแบบอนุกรม

จากรูปที่ 2.11 เป็นการแสดงการต่อแบบ Cascade โดยต่อเป็นจำนวน n TLC59419 ต่อเข้ากับไมโครคอนโทรลเลอร์โดยสร้างเป็นวงจรพื้นฐานแสดงผลทาง LED จำนวนสูงสุดในการต่อ TLC5941 คือ 40 ตัว การคำนวณความถี่ต่ำสุดที่วงจรต้องการ คือ

$$F_{(GSCLK)} = 4096 \times F_{(Update)} \qquad (5)$$

$$F_{(SCLK)} = 193 \times F_{(Update)} \times n \qquad (6)$$

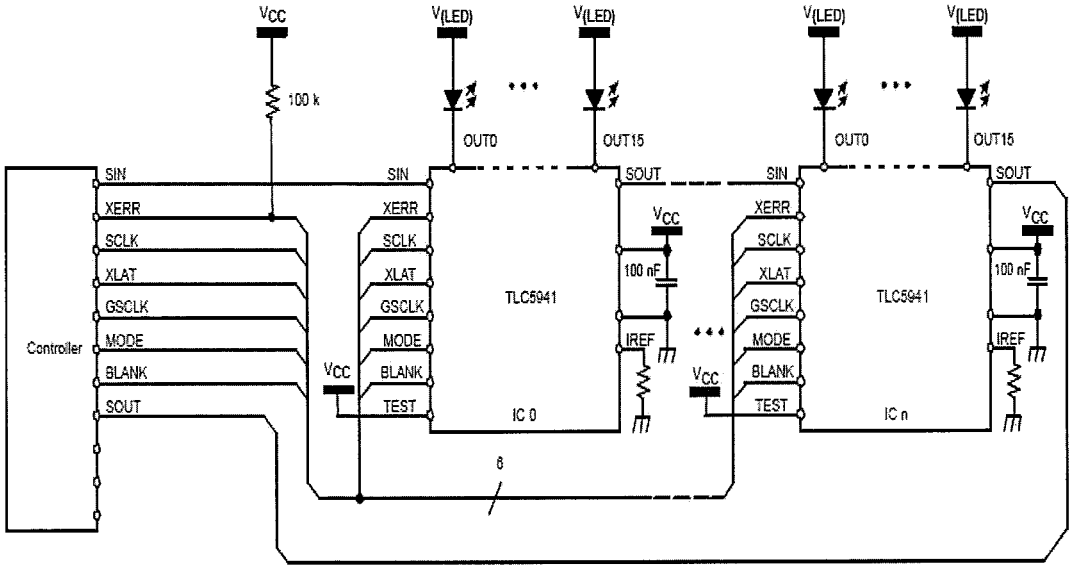
เมื่อ

$F_{(GSCLK)}$ = ความถี่ต่ำสุดที่ GSCLK ต้องการ

$F_{(SCLK)}$ = ความถี่ต่ำสุดที่ SCLK และ SIN ต้องการ

$F_{(Update)}$ = อัตราการ Update ของระบบ Cascade

n = จำนวนของ TLC5941 ที่ใช้ต่อแบบ Cascade

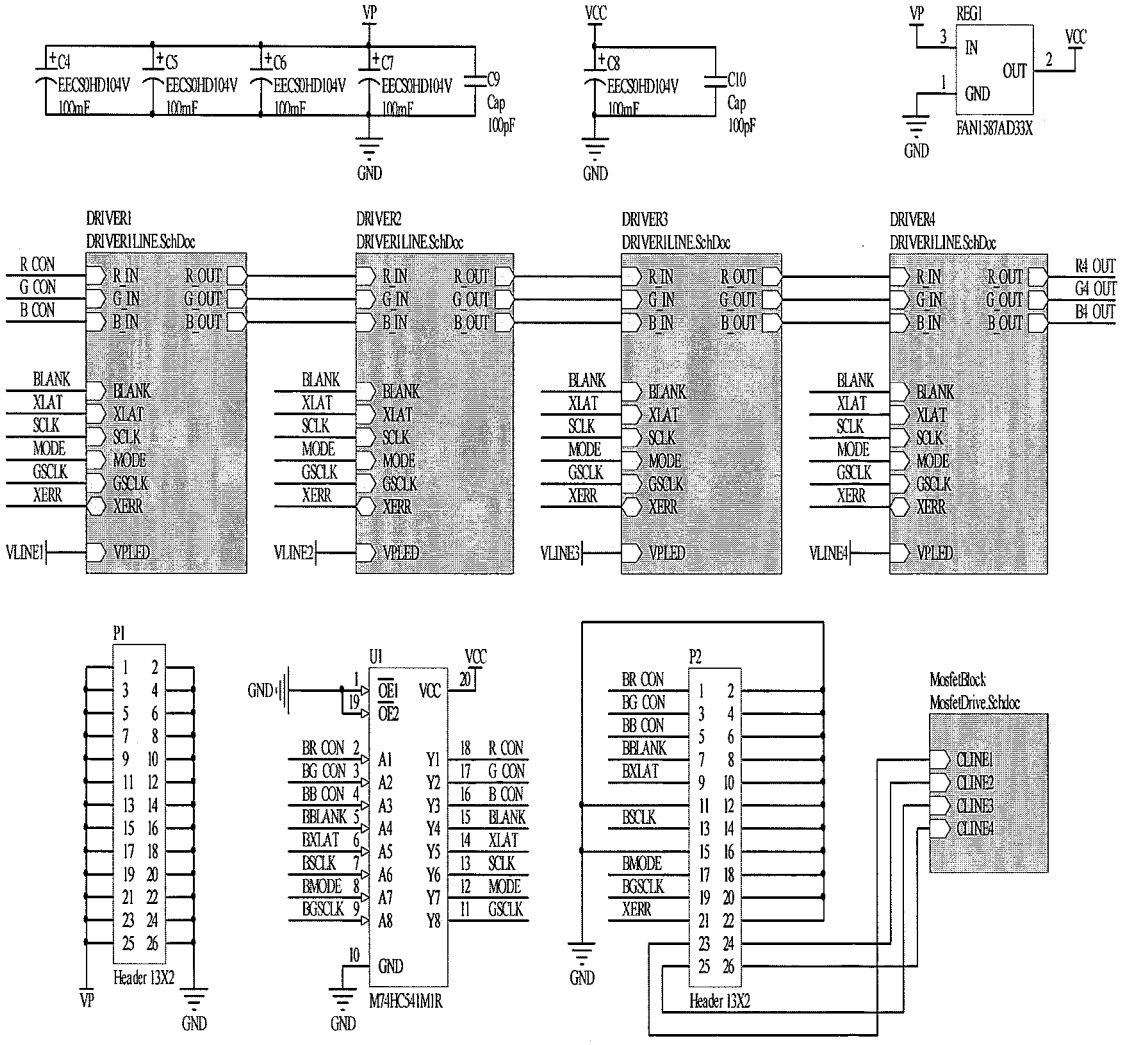


รูปที่ 2.11 แสดงการต่อวงจรแบบ Cascade

บทที่3

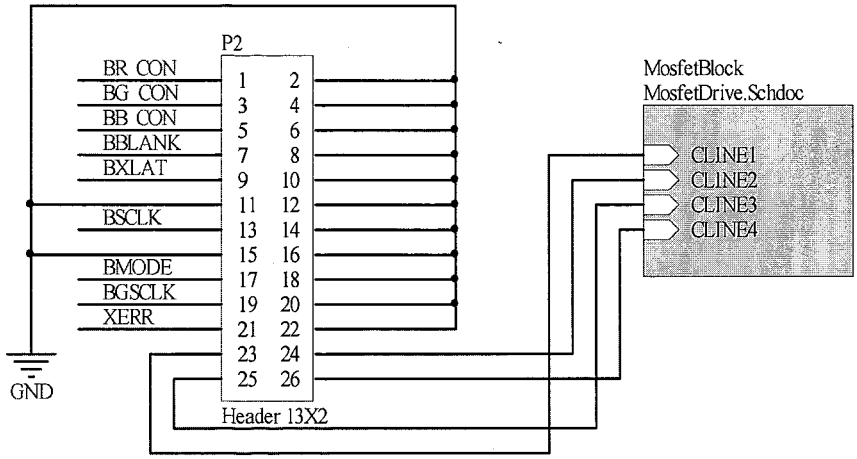
การออกแบบ

3.1 การออกแบบจอแสดงผลLED 3สี



รูปที่3.1 แสดงการออกแบบวงจรทั้งหมดของจอแสดงผลLED 3สี

3.1.1 กำหนดพอร์ตอินพุทของจอแสดงผล LED 3สี

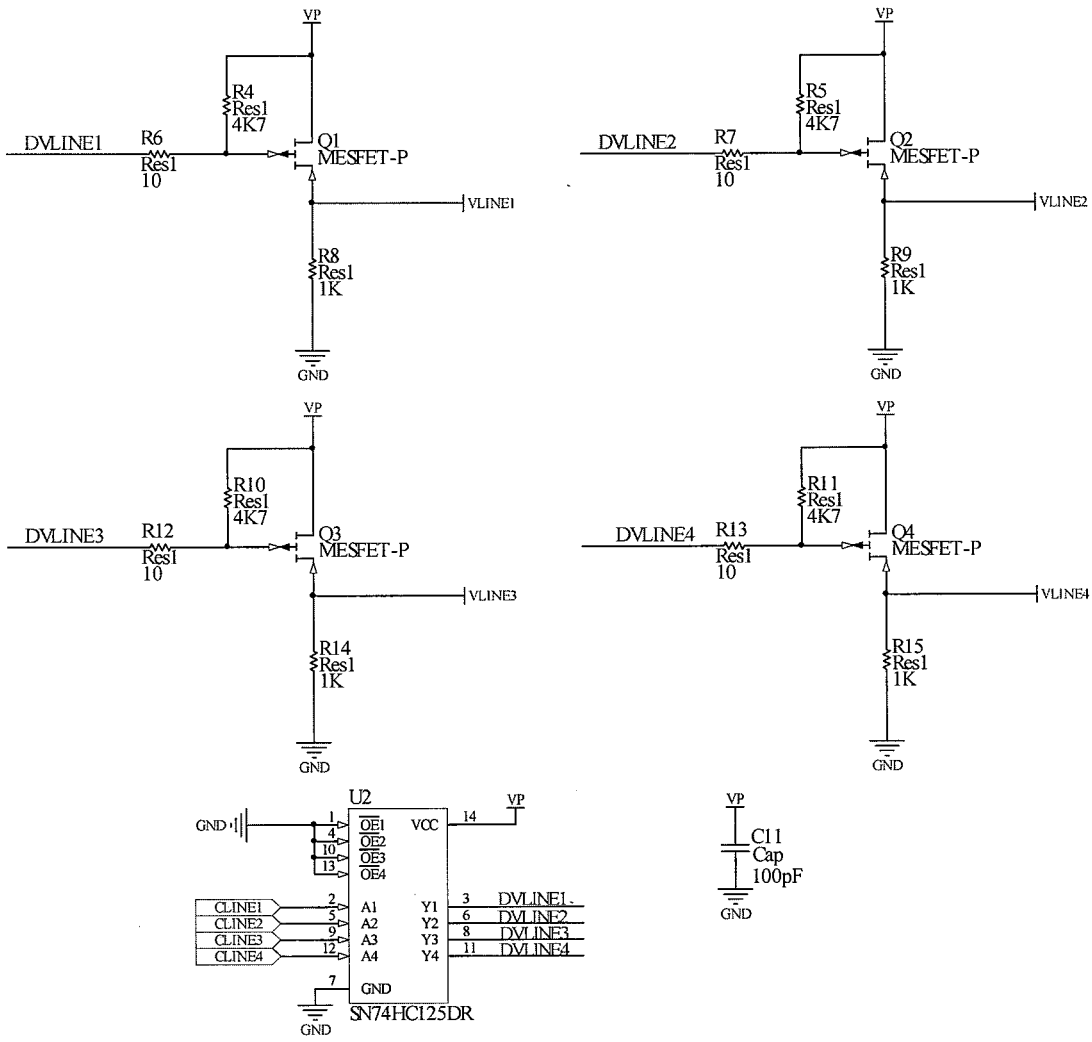


รูปที่3.2 แสดงการกำหนดพอร์ตอินพุทของจอแสดงผล LED 3สี

ตารางที่3.1 แสดงหน้าที่ของขาอินพุทต่างๆ

ขาที่	ชื่อ	ชนิดของพอร์ต	หน้าที่	หมายเหตุ
1	BR_CON	อินพุท	ควบคุมความสว่างของสีแดง	-
3	BG_CON	อินพุท	ควบคุมความสว่างของสีเขียว	-
5	BB_CON	อินพุท	ควบคุมความสว่างของสีน้ำเงิน	-
7	BBLANK	อินพุท	ควบคุมการสว่างของสีทั้งหมด	-
9	BXLAT	อินพุท	นำข้อมูลจากรีจีสเตอร์ภายออกไปยังเอาท์พุท	-
13	BSCLK	อินพุท	นำข้อมูลสีเก็บไว้ในรีจีสเตอร์ภายในไอซีTLC 5941	-
17	BMODE	อินพุท	เลือกลักษณะโหมดการทำงาน	-
19	BGSCLK	อินพุท	อ้างอิงสัญญาณนาฬิกาโดยใช้หลักการPWM	-
21	XERR	เอาท์พุท	ตรวจสอบความผิดพลาดของไอซีTLC 5941	-
23	CLINE1	อินพุท	ควบคุมการทำงานของMOSFET ชุดที่1	-
24	CLINE2	อินพุท	ควบคุมการทำงานของMOSFET ชุดที่2	-
25	CLINE3	อินพุท	ควบคุมการทำงานของMOSFET ชุดที่3	-
26	CLINE4	อินพุท	ควบคุมการทำงานของMOSFET ชุดที่4	-

3.1.2 การออกแบบภาคMOSFETDRIVER

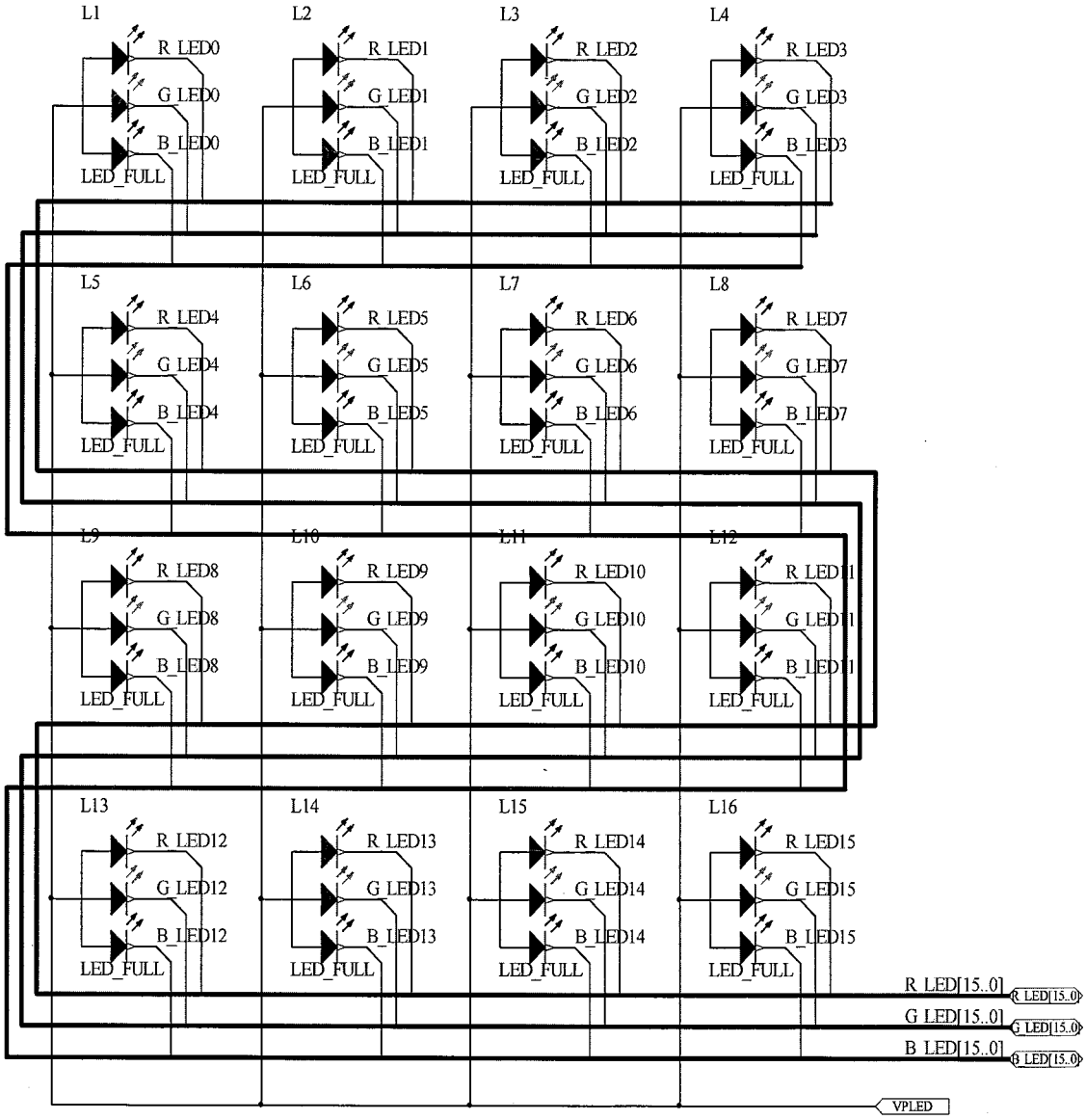


รูปที่3.3 แสดงการออกแบบภาคMOSFETDRIVER

ในการไบอัสให้กับMOSFETแต่ละตัวเราจะใช้IC Buffer เป็นตัวกันชนเพื่อป้องกันกระแสไหลย้อนกลับไปที่ Chip FPGA เราจะกำหนดให้ IC Buffer Enable ตลอดและMOSFETจะได้รับการไบอัสจากFPGA ซึ่งจะเป็นตัวควบคุมการไบอัสให้แก่MOSFETและMOSFETจะทำหน้าที่ขยายกระแสให้แก่หลอดLED3สีโดยจะต่อเข้าที่ขาAnode ซึ่งกำหนดให้เป็นขาร่วมของ LED3 สีโดยการสแกนแถว ROW เราจะใช้วงจร 2 – 4 Decoder เป็นตัวเลือกแถวและต้องให้สัญญาณจากการถอดรหัส Decoder มีการ Delay ก่อนการเปลี่ยนสถานะ นั่นคือเพื่อป้องการการจ่ายกระแสพร้อมกัน

ของ MOSFET โดยที่การเลือกแถว Row เมื่อมีการเปลี่ยนแปลงของอินพุท Row จากการถอดรหัส จะoff MOSFET ทุกตัวก่อน จากนั้นจะหน่วงเวลา และจะเริ่มเปลี่ยนเอาท์พุท ให้เป็นค่าเดียวกับอินพุท

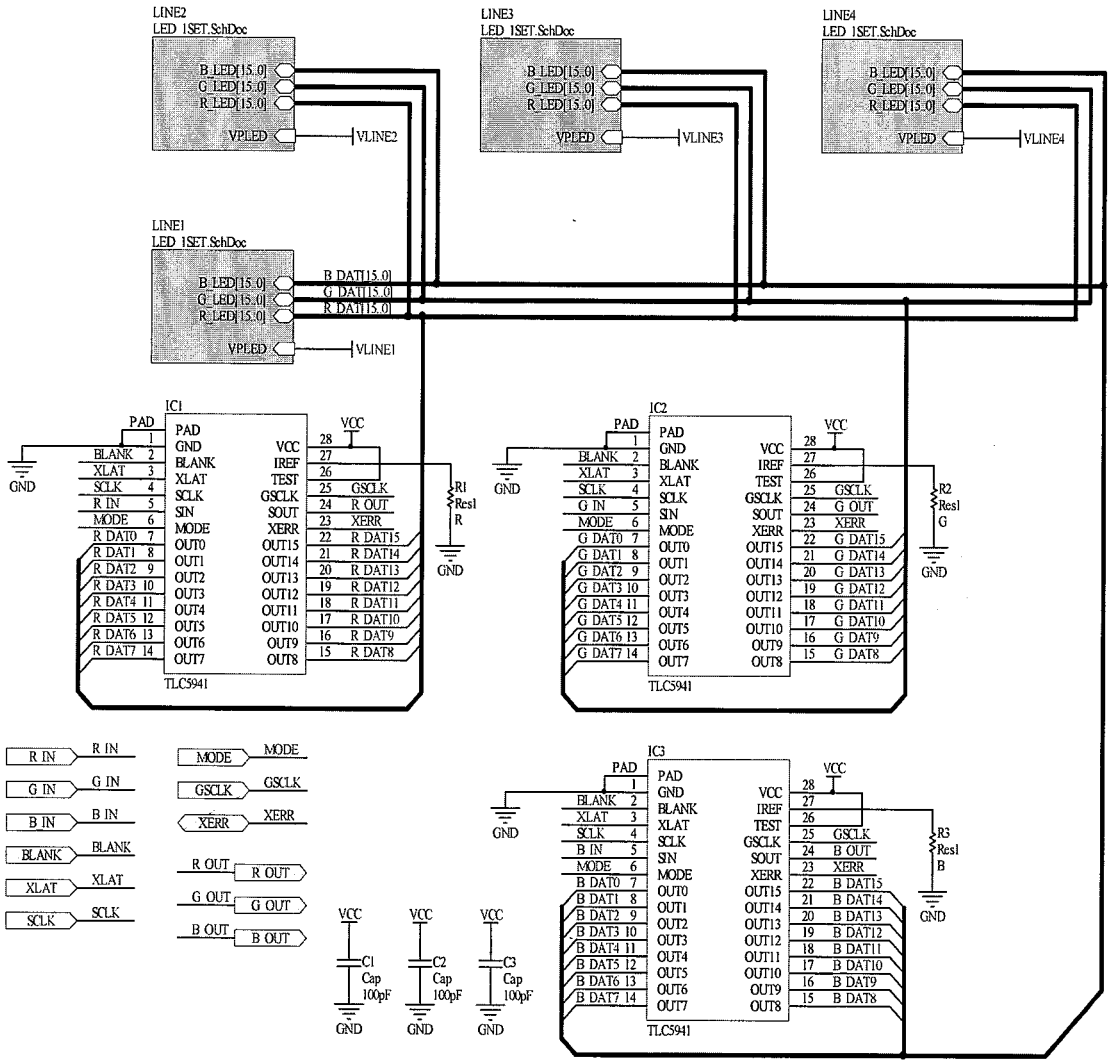
3.1.3 การออกแบบการต่อLED 3สี



รูปที่3.4 แสดงการต่อLED 3สี

ในการควบคุมการทำงานของหลอดLED 3สีจะใช้เทคนิคการสแกนซึ่งเหมือนกับการทำงานของจอ LCD โดยที่ขา Anode จะต่อเข้ากับ MOSFET ส่วนของขา Cathode จะต่อกับขาเอาท์พุทของIC TLC5941

3.1.4 การออกแบบภาคDRIVERLINE

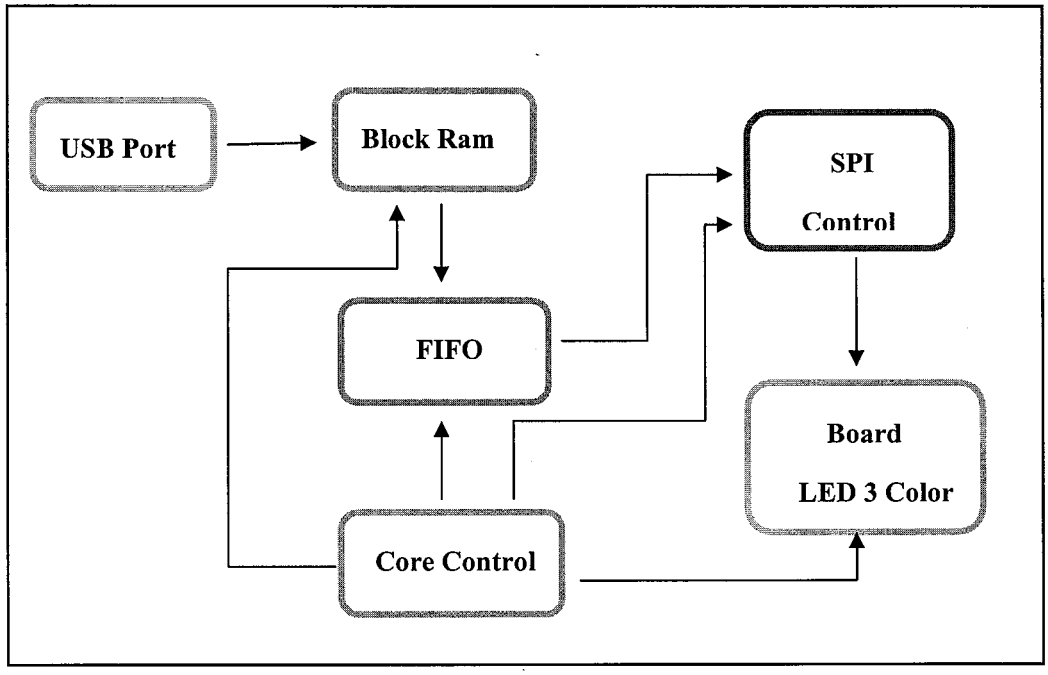


รูปที่3.5 แสดงการออกแบบภาคDRIVERLINE

จากรูป3.4 เราจะใช้IC TLC5941 เป็นตัวควบคุมการแสดงความสว่างของสีต่างๆโดยใช้หลักการของ PWM เป็นสัญญาณอ้างอิงเปรียบเทียบกับกับสัญญาณข้อมูลอินพุทที่รับมาจาก FPGA

โดยเราใช้ FPGA เป็นตัวสร้างสัญญาณควบคุมควบการทำงานทั้งหมดของ IC TLC5941 และในส่วนของเอาท์พุทจะต่อเข้ากับขา Cathode ของLED ซึ่งไอซีแต่ละตัวจะควบคุมได้เพียงหนึ่งสีและมีเอาท์พุท 16 Chanel

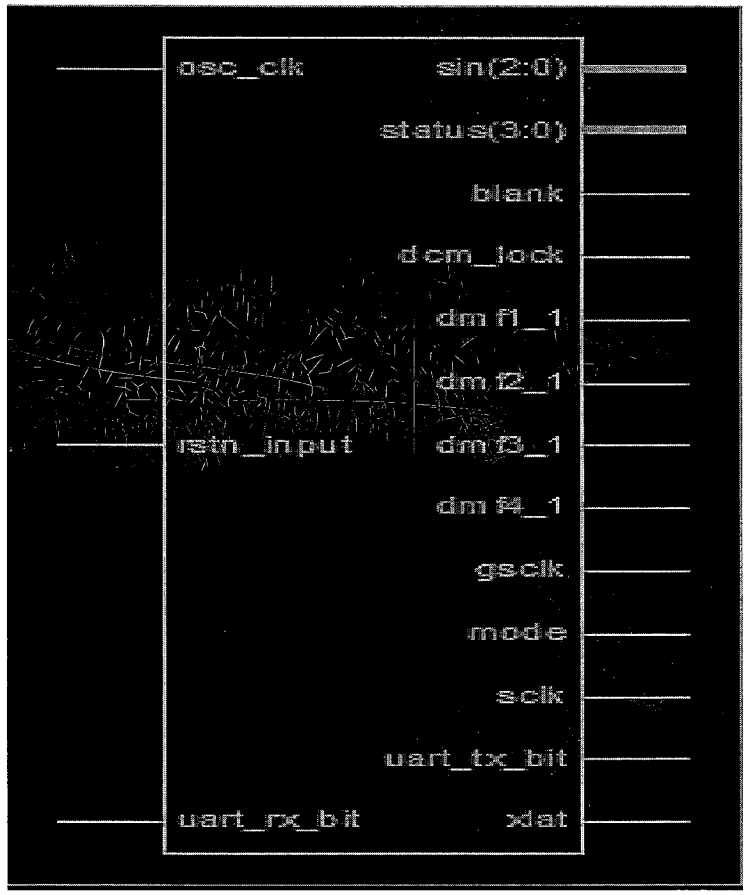
3.2 การออกแบบในส่วนของ FGPA



รูปที่3.6 แสดงบล็อกไดอะแกรมการทำงานของFGPA

จากรูปที่3.6 ข้อมูลจะถูกส่งจากคอมพิวเตอร์เข้ามายังบอร์ดFPGA (ใช้บอร์ด Panton ของ Ideaonchip) โดยผ่านทางUSB Port จากนั้นข้อมูลแต่ละPixelจะถูกนำมาเก็บไว้ที่Block Ramส่งค่าแต่ละ Pixel ไปเขียนลงใน FIFO อีกทีหนึ่ง โดยหน้าที่หลักของ FIFO จะเป็นตัวที่พักข้อมูลก่อนที่จะถูกส่งออกไปให้SPI Control จากนั้น SPI Control จะทำการอ่านข้อมูลจาก FIFO ซึ่งเป็นรูปแบบ Parallel นำมาแปลงให้เป็น Serial แล้วส่งออกไปยังBoard LED 3 Color โดยหน้าที่ของ Block SPI Control จะคอยควบคุม Timing ต่าง ๆ ในการควบคุม Block 12 Bit Shift Register With PWM Control เมื่อ Block Ram ทำอ่านข้อมูลที่จะไปแสดงบน LED ครบ 1 หน้าแล้วจะหยุดอ่านข้อมูลแต่ในส่วนของ SPI Control จะยังคงทำงานต่อไปเพราะยังมีข้อมูลค้างอยู่ใน FIFO และเมื่อ SPI Control ทำการส่งข้อมูลเสร็จ จะส่งสัญญาณให้แก่ Board LED 3 Color เพื่อให้ Board LED 3

Color เปลี่ยนแถว Row ที่จะทำการแสดงบนจอ LED และจากนั้นจะสั่งให้ Block Ram ทำงานอีกครั้งจากนั้นการทำงานก็จะเวียนซ้ำกันไปอย่างนี้เรื่อย ๆ



รูปที่3.7 แสดงบล็อกไดอะแกรม RTL Schematic OUTSIDE

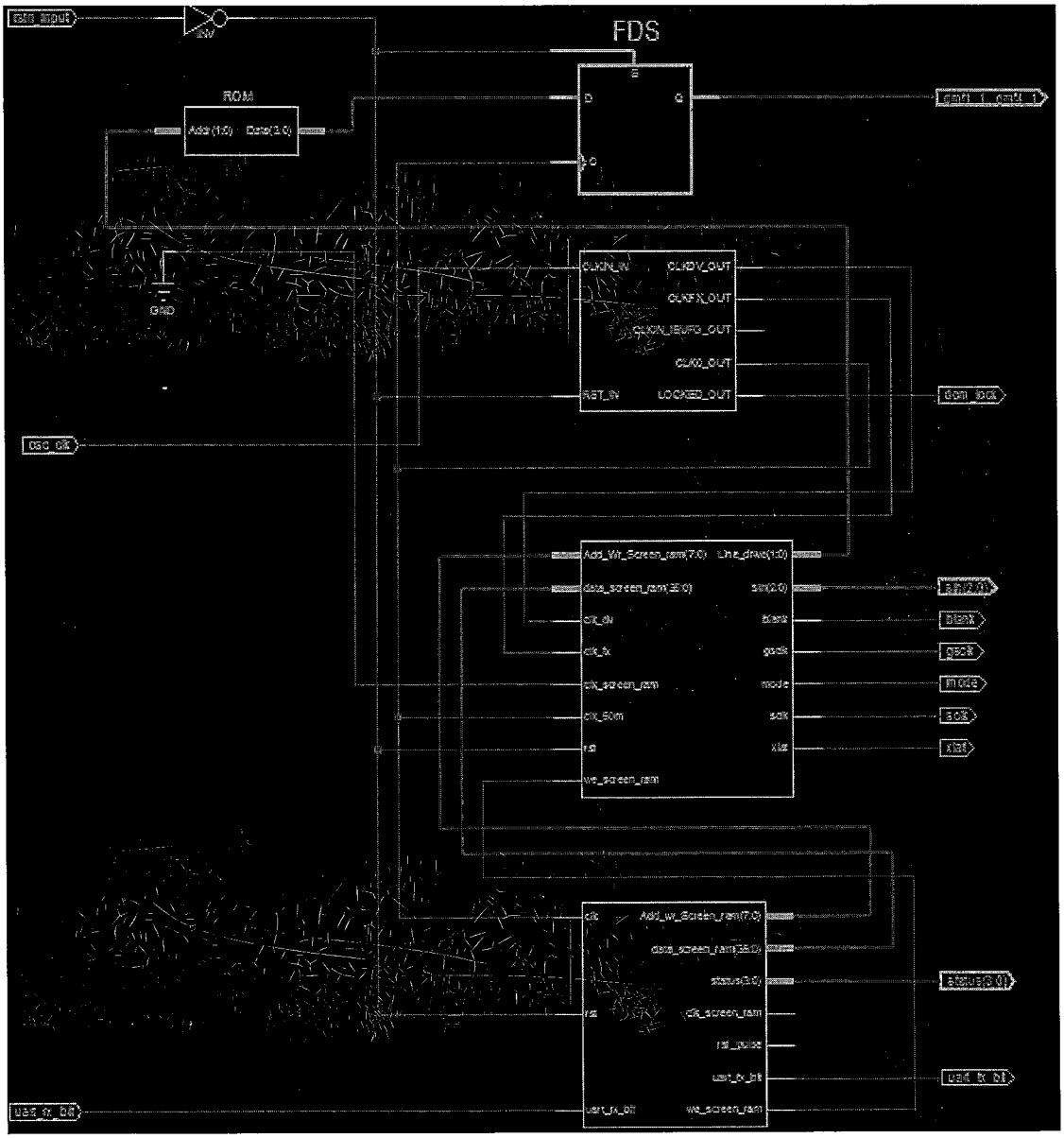
จากรูปที่3.7 มีอินพุท3อินพุทคือosc_clk ทำหน้าที่ผลิตความถี่โดยอาศัยจากOscillatorภายนอก และในการรับข้อมูลทั้งหมดจากคอมพิวเตอร์เข้ามาทางport input ชื่อ uart_rx_bit และ rstn_input ทำหน้าที่รีเซ็ตข้อมูลทั้งหมดภายในFPGA ส่วนสัญญาณทางด้านเอาท์พุทประกอบด้วยsin(2:0)เป็นสัญญาณข้อมูลของสีแดง เขียว และน้ำเงินซึ่งจะส่งไปยังจอแสดงผลLED3สี สัญญาณstatus(3:0)จะเป็นสัญญาณที่คอยติดต่อกับSPIสัญญาณBlank เป็นสัญญาณควบคุมการทำงานของIC TLC5941 และสัญญาณgsclkเป็นสัญญาณอ้างอิงมีหน้าที่เปรียบเทียบกับสัญญาณอินพุทของแต่ละสีซึ่งจะเป็นตัวกำหนดความสว่าง Mode ทำหน้าที่เป็นตัวเลือกลักษณะการทำงานของ IC TLC5941ว่าต้องการให้เป็นแบบ GS mode หรือDC mode ส่วนsclkจะเป็นสัญญาณที่จะนำ ข้อมูลแบบอนุกรมของแต่ละ

สีไปเก็บไว้ในรีจีสเตอร์ภายในของIC TLC5941 ส่วนxlat จะเป็นสัญญาณที่นำข้อมูลในรีจีสเตอร์ภายในไปยังเอาท์พุทของแต่ละช่องทางออกของ IC TLC5941และสัญญาณdm f1_1 เป็นสัญญาณควบคุมการทำงานของMOSFETตัวที่1หรือควบคุมแถวROW1

สัญญาณdm f2_1 เป็นสัญญาณควบคุมการทำงานของMOSFETตัวที่2หรือควบคุมแถวROW2

สัญญาณdm f3_1 เป็นสัญญาณควบคุมการทำงานของMOSFETตัวที่3หรือควบคุมแถวROW3

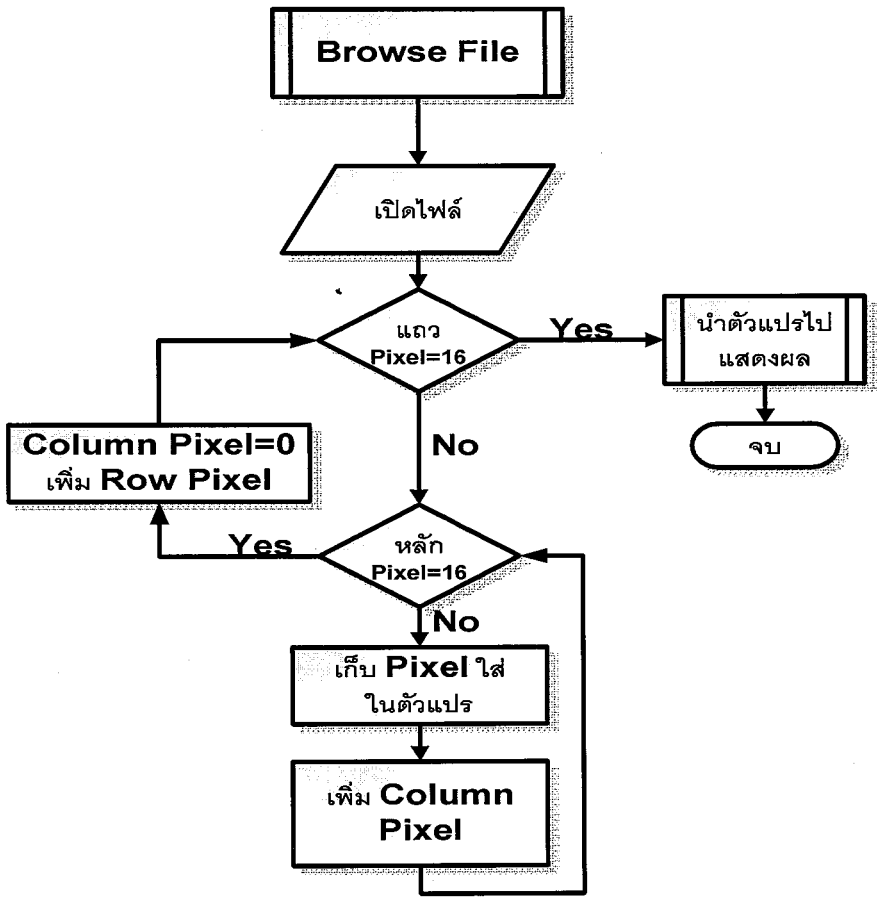
สัญญาณdm f4_1 เป็นสัญญาณควบคุมการทำงานของMOSFETตัวที่4หรือควบคุมแถวROW4



รูปที่3.8 แสดงบล็อกไดอะแกรม RTL Schematic INSIDE

3.3 การออกแบบโปรแกรมควบคุม (Visual C#)

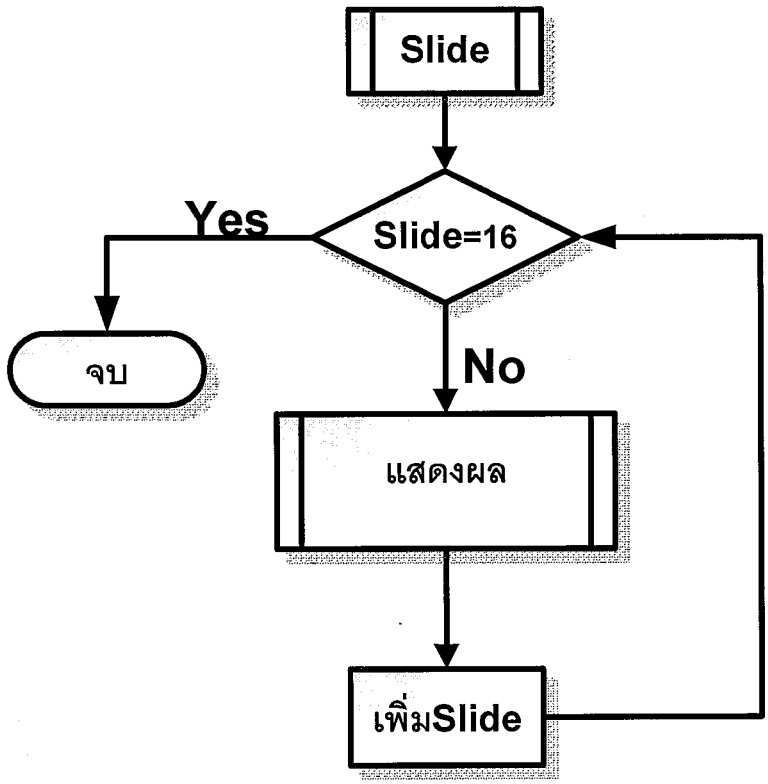
3.3.1 การเขียนโปรแกรมควบคุมการ Browse file รูปภาพ



รูปที่3.9 Flowchart ของโปรแกรมควบคุมการ Browse file รูปภาพ

ในส่วนของโปรแกรม Browse files รูปภาพ ในการทำงานจะทำการเปิดไฟล์รูปภาพจากแฟ้มข้อมูลในคอมพิวเตอร์แล้วจะส่งข้อมูลมาทำการสแกนโดยทำการตรวจสอบเงื่อนไขถ้าแถว ROWมีข้อมูลเท่ากับ16แถวแล้วจะนำข้อมูลไปแสดงที่จอแสดงผลแต่ถ้าไม่เท่ากับ16จะไปตรวจสอบเงื่อนไขต่อไปคือจะไปตรวจสอบที่แถว COLUMNว่ามีข้อมูลPIXEL เท่ากับ 16 ถ้าไม่จะนำค่าPIXEL ไปเก็บแล้วจะทำการเพิ่มCOLUMN PIXEL ไปเรื่อยๆจนมีค่าเท่ากับ16 จะส่งค่าไปให้แถวทำการตรวจสอบเงื่อนไขจนมีค่าเท่ากับ16แล้วจึงส่งค่าไปให้จอแสดงผล

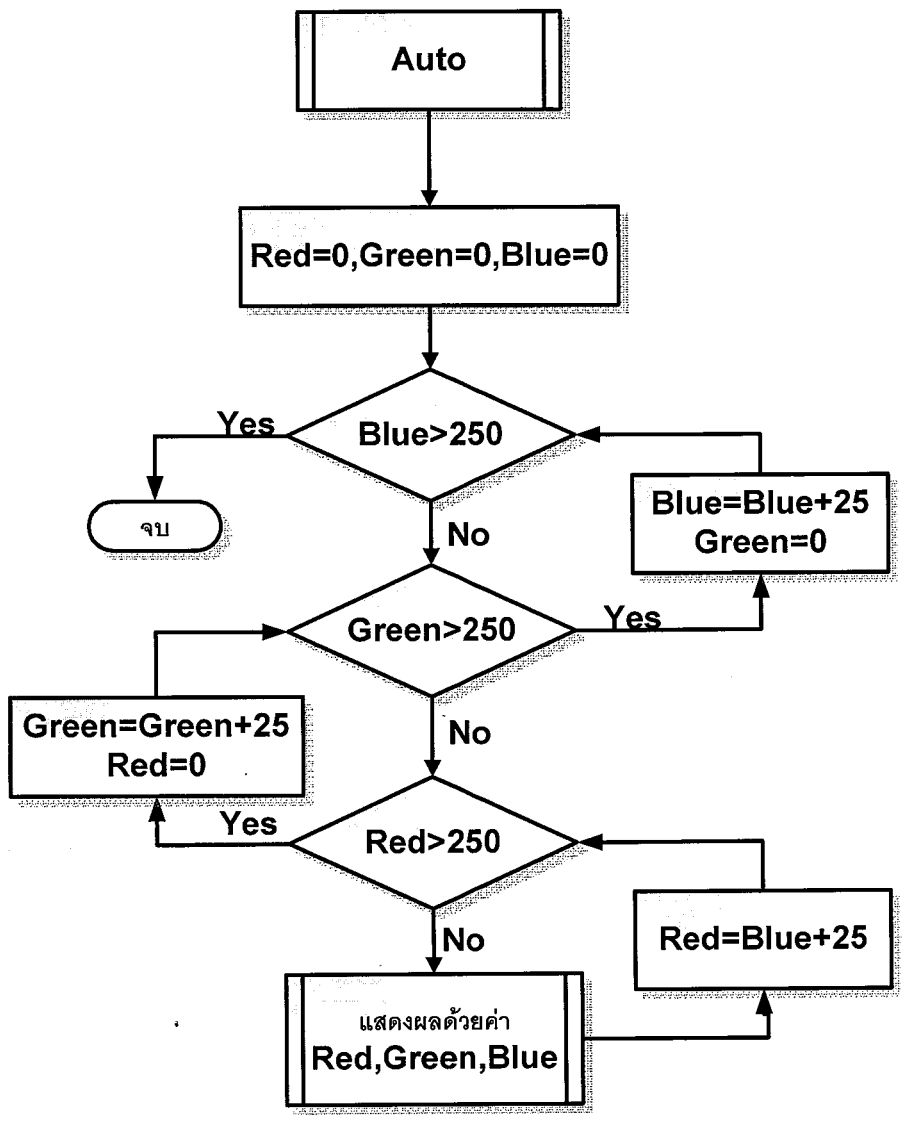
3.3.2 การเขียนโปรแกรมควบคุมการปรับระดับสีทั้งสามสีโดยการปรับทีละแถว



รูปที่3.10 Flowchart ของโปรแกรมควบคุมการปรับระดับสีทั้งสามสีโดยการปรับทีละแถว

ในส่วนของโปรแกรมควบคุมการปรับระดับสีทั้งสามสีโดยการปรับทีละแถวจะทำการตรวจสอบว่าSlide มีขนาดเท่ากับ16หรือไม่ถ้าSlide เท่ากับ16 จะไม่ส่งค่าไปแสดง แต่ถ้ามีขนาดไม่เท่ากับ16จะส่งผลไปแสดงที่บอร์ดแสดงผล RGB แล้วทำการเพิ่ม Slide ไปเรื่อยๆและจะไปตรวจสอบเงื่อนไขว่าเท่ากับ16หรือไม่จะทำไปเรื่อยๆจนเท่ากับ16แล้วจึงจะหยุดทำงาน

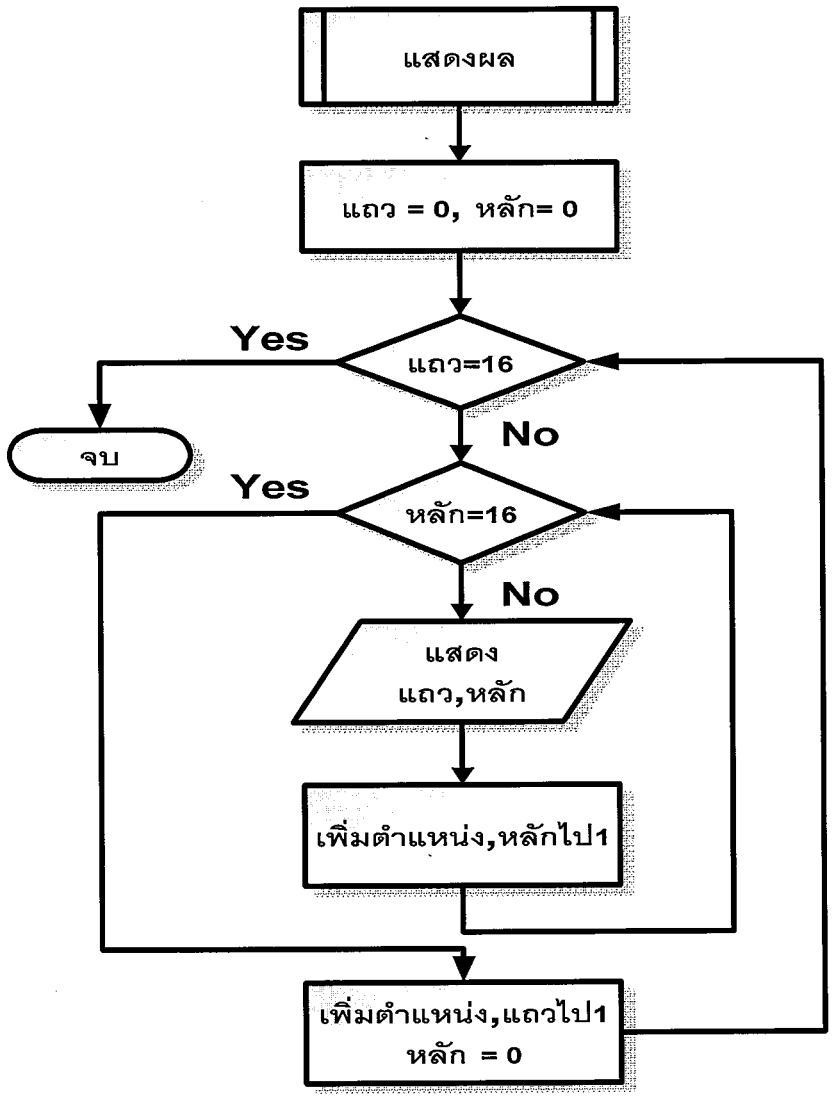
3.3.3 การเขียนโปรแกรมควบคุมการปรับระดับสีแบบอัตโนมัติ (Auto)



รูปที่3.11 โปรแกรมควบคุมการปรับระดับสีทั้งสามสีโดยการปรับทีละแถว

ในการควบคุมการปรับระดับสีอัตโนมัติจะทำการเพิ่มค่าสีแดงทีละ25 เมื่อสีแดงมีค่า250จะทำให้สีเขียวมีค่าเพิ่ม25และจะวนอย่างนี้ไปเรื่อยๆจน สีเขียวมีค่า250และจะเพิ่มค่าสีน้ำเงินขึ้น25 จากนั้นจะทำอย่างนี้ไปเรื่อยจนสีน้ำเงินมีค่า250และจะทำการรีเซ็ตค่าทุกสีเป็น0และหยุดการทำงาน

3.3.4 โปรแกรมการทำงานของบอร์ดแสดงผล RGB 16 x 16



รูปที่3.12 โปรแกรมการทำงานของบอร์ดแสดงผล RGB 16 x 16

ในส่วนของโปรแกรมการทำงานของบอร์ดแสดงผล RGB 16 x 16 จะทำการตรวจสอบว่าแถวเท่ากับศูนย์และตรวจสอบว่าหลักเท่ากับศูนย์แล้วจะตรวจสอบว่าแถวมีขนาดเท่ากับ16หรือไม่ถ้าใช่จะจบการทำงานแต่ถ้ายังไม่เท่ากับ16 จะไปตรวจสอบว่าหลักเท่ากับ16หรือไม่ถ้าใช่จะไปเพิ่มตำแหน่งแถวไปอีกหนึ่งแถวแต่ถ้ายังไม่เท่ากับ16จะส่งค่าไปแสดงที่บอร์ดแสดงผลและจะเพิ่มตำแหล่งหลักไปเรื่อยๆจนครบ16หลักแล้วจะไปเพิ่มแถวให้ครบ16จะทำงานจนแถวและหลักมีขนาดเท่ากับ16จึงจะหยุดทำงาน

บทที่ 4

วิธีการทดลองและผลการทดลอง

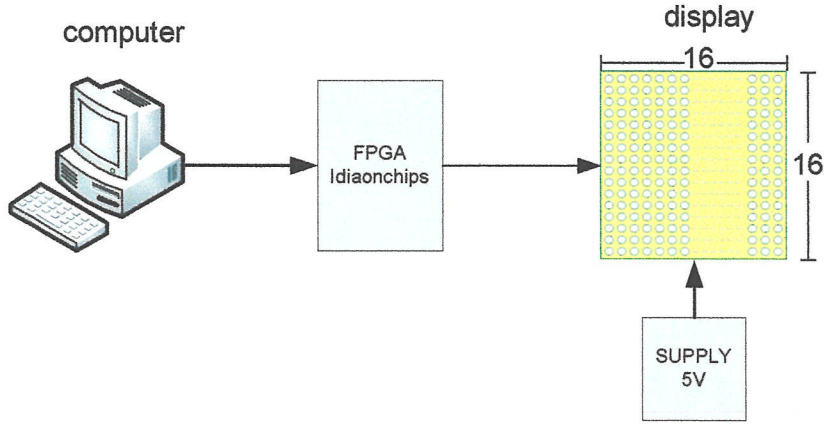
4.1 อุปกรณ์ที่ใช้ในการทดลอง

- บอร์ดแสดงผล RGB ขนาด 16 x 16	1 บอร์ด
-SUPPLY 5 V	1 ตัว
- คอมพิวเตอร์	1 เครื่อง
- บอร์ด FPGA Idiaonchips	1 เครื่อง

4.2 การต่อใช้งานบอร์ดแสดงผลกับบอร์ดFPGA

4.2.1 วิธีการทดลอง

1) ต่ออุปกรณ์ต่างๆดังรูป 4.1



รูปที่ 4.1 แสดงการต่อวงจรใช้งาน

2) ต่อคอมพิวเตอร์เข้ากับบอร์ด FPGA โดยผ่านทาง USB Port
3) ต่อ SUPPLY 5 V เข้ากับบอร์ดแสดงผล RGB ขนาด 16 x 16
4) ต่อบอร์ด FPGA เข้ากับบอร์ดแสดงผล RGB ขนาด 16 x 16 โดยขณะที่ต่อควรปิด SUPPLY 5 V ก่อนเสมอ

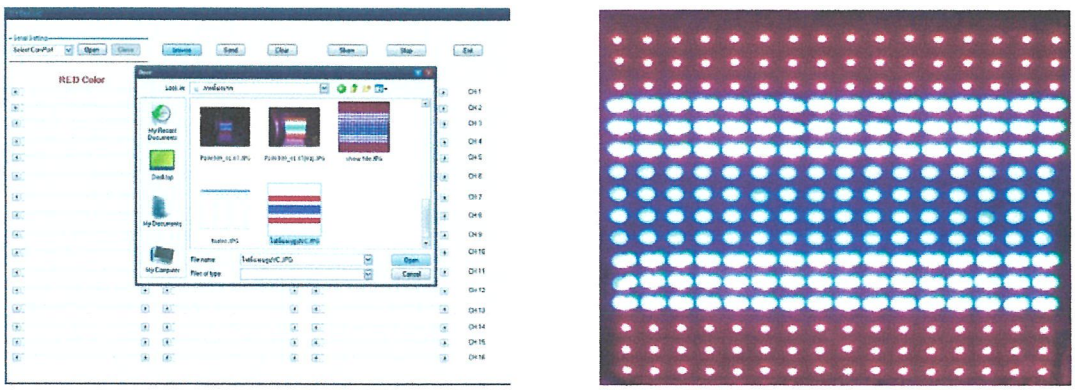
4.3 การแสดงภาพโดยการ Browse file รูปภาพ

4.3.1 วิธีการทดลอง

1) เปิดคอมพิวเตอร์ และ บอร์ด FPGA Idiaonchips แล้วทำการเชื่อมต่อเข้ากับบอร์ดแสดงผล RGB ขนาด 16 x 16 และเปิดโปรแกรมควบคุมการปรับระดับสีที่ได้ออกแบบไว้ (โปรแกรมที่เขียนขึ้นด้วย Visual C#)
2) กดเลือก Port เพื่อทำการเชื่อมต่อระหว่างคอมพิวเตอร์กับบอร์ด FPGA เมื่อทำการเลือก Port แล้วกดปุ่ม open
3) กดปุ่ม Browse บนหน้าจอควบคุมการปรับระดับสี
4) เลือก files รูปภาพที่ต้องการนำเสนอสังเกตบนจอแสดงผล

4.3.2 ผลการทดลอง

เมื่อกำหนด Port ในการเชื่อมต่อระหว่างคอมพิวเตอร์กับบอร์ด FPGA แล้วกด open แล้วทำการกดปุ่ม Browse file รูปจะได้หน้าต่างควบคุมแสดงในรูปที่ 4.2 (ก) แล้วทำการกด open จะได้ผลการทดลองดังแสดงในรูปที่ 4.2 (ข)



(ก) (ข)

รูปที่ 4.2 แสดงผลการเลือกไฟล์รูปภาพ

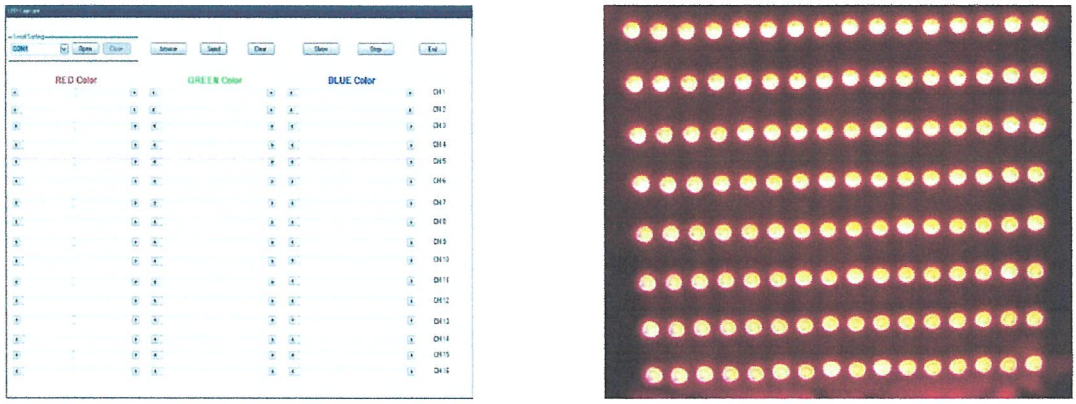
4.4 การควบคุมสีโดยการปรับระดับสีทั้งสามสีโดยการปรับทีละแถว

4.4.1 วิธีการทดลอง

1) จากการทดลองในหัวข้อที่ 4.3 เมื่อทำการกดปุ่ม Clear จะทำให้บอร์ดแสดงค่าเหมือนกับตอนเริ่มต้น

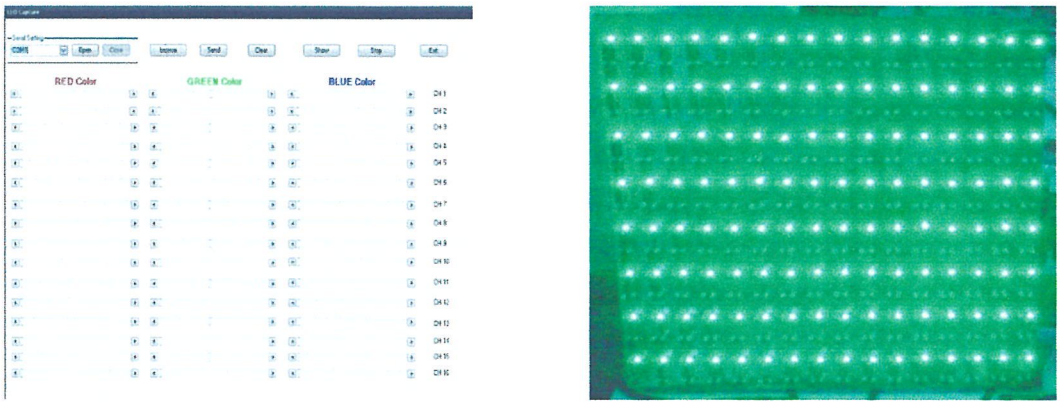
2) ทำการปรับ Score bar ทีละสีโดยทำการปรับสีแดง (RED Color) โดยปรับให้อยู่ตรงกลางทีละแถวเว้นแถวดังรูปที่ 4.3 (ก)
3) กดปุ่ม Send จะได้ภาพสีแดงโดยการปรับทีละแถวแสดงทางบอร์ดแสดงผล RGB ขนาด 16 x 16 ดังรูปที่ 4.3 (ข)
4) เมื่อได้สีแดงตามที่ได้ปรับ Score bar แล้วทำการกดปุ่ม Clear
5) ทำการปรับ Score bar ทีละสีโดยทำการปรับสีเขียว (GREEN Color) โดยปรับให้อยู่ตรงกลางทีละแถวเว้นแถวดังรูปที่ 4.4 (ก)
6) กดปุ่ม Send จะได้ภาพสีเขียวโดยการปรับทีละแถวแสดงทางบอร์ดแสดงผล RGB ขนาด 16 x 16 ดังรูปที่ 4.4 (ข)
7) เมื่อได้สีเขียวตามที่ได้ปรับ Score bar แล้วทำการกดปุ่ม Clear
8) ทำการปรับ Score bar ทีละสีโดยทำการปรับสีน้ำเงิน (BLUE Color) โดยปรับให้อยู่ตรงกลางทีละแถวเว้นแถวดังรูปที่ 4.5 (ก)
กดปุ่ม Send จะได้ภาพสีน้ำเงินโดยการปรับทีละแถวแสดงทางบอร์ดแสดงผล RGB ขนาด 16 x 16 ดังรูปที่ 4.5 (ข)

4.4.2 ผลการทดลอง



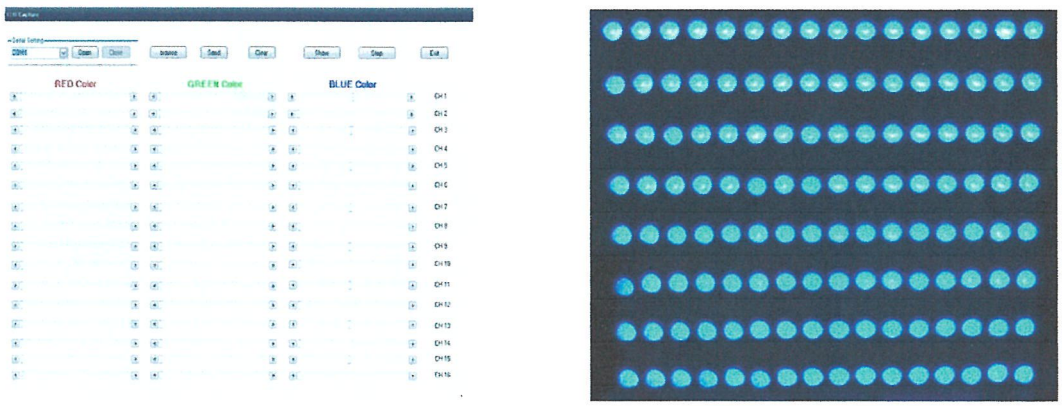
(ก) (ข)

รูปที่ 4.3 แสดงการปรับระดับสีแดง



(ก) (ข)

รูปที่ 4.4 แสดงการปรับระดับสีเขียว



(ก) (ข)

รูปที่ 4.5 แสดงการปรับระดับสีน้ำเงิน

จากการทดลอง เมื่อเรากดปุ่มเลือก Port ในการติดต่อระหว่างคอมพิวเตอร์กับบอร์ดFPGA โดยผ่านทาง USB Port จะได้ผลการปรับค่าสีแดงทีละแถวโดยแสดงดังรูปที่แสดงโดยทั้งหมดเป็นการปรับระดับสีทีละสีโดยปรับสีแดง, สีเขียว และน้ำเงิน ตามลำดับ

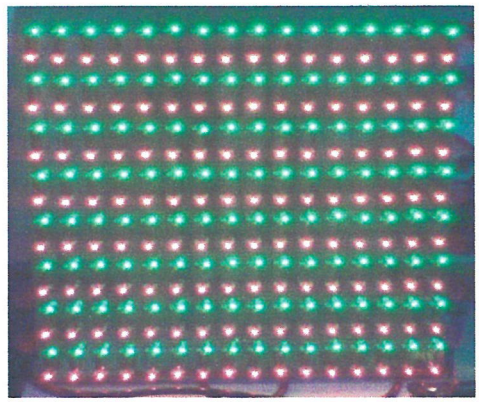
4.5 การควบคุมการปรับระดับสีแบบอัตโนมัติ (Auto)

4.5.1 วิธีการทดลอง

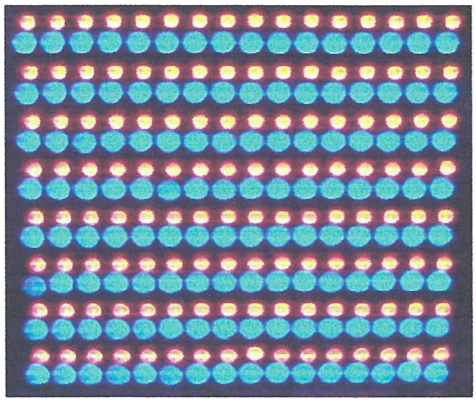
1) จากการทดลองในหัวข้อที่ 4.3 เมื่อทำการกดปุ่ม Clear จะทำให้บอร์ดแสดงค่าเหมือนกับตอนเริ่มต้น
2) กดปุ่ม Show บนหน้าต่างการควบคุม
3) สังเกตผลการทดลองที่บอร์ดแสดงผล RGB ขนาด 16x 16

4.5.2 ผลการทดลอง

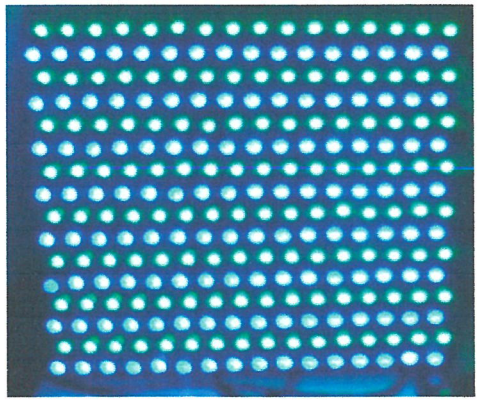
(ก)



(ข)



(ค)



(ง)

รูปที่ 4.6 แสดงการปรับสีแบบอัตโนมัติ

จากการทดลองเมื่อเราทำการกดปุ่ม Show บอร์ดแสดงผล RGB ขนาด 16x 16 จะทำการปรับระดับสีเริ่มจากสีแดงจะสว่างขึ้นเรื่อยๆแล้วสีเขียวจะค่อยๆสว่างตามโดยสีแดงจะวนลูปก่อน 10ครั้ง แล้วสีเขียวจะค่อยๆสว่างตามแล้วสีน้ำเงินก็จะสว่างจะทำงานแบบนี้ไปเรื่อยๆจนถึงค่าสูงสุดคือ 250 จะทำให้หลอด RGB ทั้งสามสีดับลง

บทที่ 5

สรุปผลการทดลอง

5.1 สรุปผลการทดลอง

จากการทดลองสามารถแสดงข้อมูลทั้งหมดจากคอมพิวเตอร์ไปแสดงที่จอแสดงผลLED 3 สีได้3รูปแบบคือ

1.ปรับระดับสีต่างๆได้ด้วยตัวเอง เราสามารถปรับระดับสีแดง สีเขียว สีน้ำเงิน ได้อย่างอิสระตามความต้องการสามารถปรับระดับความสว่างได้ตั้งแต่0-255

2.ปรับระดับสีแบบอัตโนมัติ เป็นการแสดงการผสมสีแบบอัตโนมัติโดยการทำงานคือจะเริ่มจากสีแดงโดยค่าความสว่างของสีแดงจะเพิ่มขึ้นที่ละ25จนครบ250แล้วจะทำการเพิ่มสีเขียวขึ้นจาก0ไปเป็น25จากนั้นสีแดงจะทำงานซ้ำไปเรื่อยจนเพิ่มค่าสีเขียวถึง250เมื่อสีแดงและสีเขียวมีค่า250แล้วจะทำการเพิ่มสีน้ำเงินขึ้นจาก0ไปเป็น25จากนั้นสีแดงกับสีเขียวจะทำงานวนซ้ำอย่างนี้ไปเรื่อยๆจนสีน้ำเงินมีค่าเท่ากับ250และสีแดงกับสีเขียวเท่ากับ250จะเป็นการแสดงค่าสูงสุดจากนั้นจะทำการรีเซ็ตค่ากับมาเป็น0ทั้ง3สีและหยุดารทำงาน

3.นำไฟล์รูปภาพจากคอมพิวเตอร์ไปแสดงที่จอแสดงผลLED 3สี จะแสดงไฟล์รูปภาพที่ต้องการซึ่งขนาด16x16 PixelไปแสดงยังจอแสดงผลLED 3สี

5.2 ปัญหาและวิธีแก้ไข

จากการทดลองจะพบปัญหาคือเมื่อทำการปรับค่าความสว่างมีค่ามากๆจะเกิดความผิดเพี้ยนของหลอดLEDซึ่งเกิดจากแหล่งจ่ายไฟมีกระที่ไม่เพียงพอและในการแสดงภาพที่มีความไวในการเปลี่ยนแปลงจะเห็นได้ว่ามีการสแกนเกิดขึ้นทำให้ได้ภาพที่ไม่ต่อเนื่องเกิดจากพอร์ตอินพุทของบอร์ดFPGAเป็นแบบRS232วิธีแก้คือเปลี่ยนพอร์ตอินพุทให้เป็นการรับข้อมูลแบบUSB และเมื่อนำไฟล์รูปที่มีความละเอียดมากกว่า16x16 PixelไปแสดงยังจอแสดงผลLED 3สีภาพที่ได้จะมีความหยาบและระยะในการมองภาพจะใช้ระยะที่ไกลเกิดจากระยะห่างของหลอดLED วิธีการแก้ไขคือลดระยะห่างระหว่างหลอดให้น้อยลง

5.3 แนวทางในการประยุกต์ใช้งาน

1. สามารถนำไปประยุกต์ใช้เป็นจอแสดงผลภาพขนาดใหญ่

2. สามารถนำไปประยุกต์ใช้งานกลางแจ้งเนื่องจากหลอดLEDมีความสว่างมากกว่าโทรทัศน์

บรรณานุกรม

ธีรยศ เวียงทอง."เรียนรู้การออกแบบระบบดิจิตอลด้วยภาษ Verilog เบื้องต้น." [Online]. Available : http://www.aldec.com/products/stutorials. 2005

Michael D. Ciletti. **Advand Digital Design with the VERILOG HDL** :PEARSON Prentice Hall.

Stuart Sutherland, Simon Davidmann, Peter Flake. **System Verilog For Design** : Kluwer Academic Publishers.

Pong P. Chu. **FPGA PROTOTYPING BY VERILOG EXAMPLES**. United States of America: A JOHN WILEY & SONS, INC., PUBLICATIO.

ภาคผนวก

Source Code Visual C#

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;

namespace ControlLED
{

  public partial class ControlPanel : Form
  {

    private Bitmap bm;
    Thread MyThread;

    private Form m_InstanceRef = null;
    public Form InstanceRef
    {
      get
      {
        return m_InstanceRef;
      }
      set
      {
        m_InstanceRef = value;
      }
    }

    public ControlPanel()
```

```
{
    InitializeComponent();
}

private void bttCaptureArea_Click(object sender, EventArgs e)
{
    //this.Hide();
    //Form1 form1 = new Form1();
    //form1.InstanceRef = this;
    //form1.Show();
    timer1.Enabled = false;
}

private void btnStop_Click(object sender, EventArgs e)
{
    timer1.Enabled = false;
}
private void bttExit_Click(object sender, EventArgs e)
{
    Application.Exit();
}



private void ControlPanel_Load(object sender, EventArgs e)
{

    cbxComport.Items.Add("COM1");
    cbxComport.Items.Add("COM2");
    cbxComport.Items.Add("COM3");
```

```
    cbxComport.Items.Add("COM4");
    cbxComport.Items.Add("COM5");
    btnCloseSerial.Enabled = false;
    timer1.Interval = 300;
    timer1.Enabled = false;

}



private void btnExit_Click(object sender, EventArgs e)
{
    Application.Exit();
}

private void btnOpenSerial_Click(object sender, EventArgs e)
{

    btnOpenSerial.Enabled = false;
    btnCloseSerial.Enabled = true;

    serialPort1.PortName = cbxComport.Text;
    serialPort1.BaudRate = 230400;
    serialPort1.Open();

}

private void btnCloseSerial_Click(object sender, EventArgs e)
{
    btnOpenSerial.Enabled = true;
```

```
    btnCloseSerial.Enabled = false;
    timer1.Enabled = false;

    btnClear_Click(sender, e);
    cbxComport.Text = "Select ComPort";
    serialPort1.Close();

}

private void SendDat_serial()
{

    Color[,] PixelColor = new Color[16,16];
    //byte[] dat = new byte[3];
    int x_size, y_size;

    send_header();

    for (y_size = 0; y_size < 16; y_size++)
    {
        for (x_size = 0; x_size < 16; x_size++)
        {
            PixelColor[y_size, x_size] = bm.GetPixel(x_size, y_size);
        }

    }

    serial_out(PixelColor);

    bm.Dispose();
```

```
}



private void send_header()
{
   byte[] dat = new byte[3];

   dat[0] = 0xff;

   serialPort1.Write(dat, 0, 1);
}

private void serial_out(Color[,] Pixelcolor)
{
   byte[] dat = new byte[768];
   int i, j;

   for (j = 0; j < 16; j++)
   {
      for (i = 0; i < 16; i++)
      {
         if (Pixelcolor[j, i].R == 0xff)
         {
            dat[(i * 16 * 3) + (j * 3)] = 0xfe;
         }
         else
         {
            dat[(i * 16 * 3) + (j * 3)] = Convert.ToByte(Pixelcolor[j, i].R);
         }
```

```
            if (Pixelcolor[j, i].G == 0xff)
            {
               dat[(j * 16 * 3) + (i * 3) + 1] = 0xfe;
            }
            else
            {
               dat[(j * 16 * 3) + (i * 3) + 1] = Convert.ToByte(Pixelcolor[j, i].G);
            }

            if (Pixelcolor[j, i].B == 0xff)
            {
               dat[(j * 16 * 3) + (i * 3) + 2] = 0xfe;
            }
            else
            {
               dat[(j * 16 * 3) + (i * 3) + 2] = Convert.ToByte(Pixelcolor[j, i].B);
            }

        }
    }

    serialPort1.Write(dat, 0, 768);
}


private void btnSendManual_Click(object sender, EventArgs e)
{
    SendDat_serial();
```

```
    }

    private void hsbR_Scroll(object sender, ScrollEventArgs e)
    {
//      send_scrollbar();
    }

    private void send_scrollbar()
    {
      byte[] dat = new byte[768];
      byte[] r = new byte[16];
      byte[] g = new byte[16];
      byte[] b = new byte[16];


      r[0] = Convert.ToByte(hsbR1.Value);
      if (r[0] == 0xff)
      {
        r[0] = 0xfe;
      }

      g[0] = Convert.ToByte(hsbG1.Value);
      if (g[0] == 0xff)
      {
        g[0] = 0xfe;
      }

      b[0] = Convert.ToByte(hsbB1.Value);
      if (b[0] == 0xff)
      {
```

```
    b[0] = 0xfe;
}

r[1] = Convert.ToByte(hsbR2.Value);
if (r[1] == 0xff)
{
    r[1] = 0xfe;
}

g[1] = Convert.ToByte(hsbG2.Value);
if (g[1] == 0xff)
{
    g[1] = 0xfe;
}

b[1] = Convert.ToByte(hsbB2.Value);
if (b[1] == 0xff)
{
    b[1] = 0xfe;
}


r[2] = Convert.ToByte(hsbR3.Value);
if (r[2] == 0xff)
{
    r[2] = 0xfe;
}

g[2] = Convert.ToByte(hsbG3.Value);
if (g[2] == 0xff)
```

```
{
   g[2] = 0xfe;
}

b[2] = Convert.ToByte(hsbB3.Value);
if (b[2] == 0xff)
{
   b[2] = 0xfe;
}

r[3] = Convert.ToByte(hsbR4.Value);
if (r[3] == 0xff)
{
   r[3] = 0xfe;
}

g[3] = Convert.ToByte(hsbG4.Value);
if (g[3] == 0xff)
{
   g[3] = 0xfe;
}

b[3] = Convert.ToByte(hsbB4.Value);
if (b[3] == 0xff)
{
   b[3] = 0xfe;
}

r[4] = Convert.ToByte(hsbR5.Value);
if (r[4] == 0xff)
```

```
{
   r[4] = 0xfe;
}

g[4] = Convert.ToByte(hsbG5.Value);
if (g[4] == 0xff)
{
   g[4] = 0xfe;
}

b[4] = Convert.ToByte(hsbB5.Value);
if (b[4] == 0xff)
{
   b[4] = 0xfe;
}

r[5] = Convert.ToByte(hsbR6.Value);
if (r[5] == 0xff)
{
   r[5] = 0xfe;
}

g[5] = Convert.ToByte(hsbG6.Value);
if (g[5] == 0xff)
{
   g[5] = 0xfe;
}

b[5] = Convert.ToByte(hsbB6.Value);
if (b[5] == 0xff)
```

```
{
   b[5] = 0xfe;
}


r[6] = Convert.ToByte(hsbR7.Value);
if (r[6] == 0xff)
{
   r[6] = 0xfe;
}

g[6] = Convert.ToByte(hsbG7.Value);
if (g[6] == 0xff)
{
   g[6] = 0xfe;
}

b[6] = Convert.ToByte(hsbB7.Value);
if (b[6] == 0xff)
{
   b[6] = 0xfe;
}


r[7] = Convert.ToByte(hsbR8.Value);
if (r[7] == 0xff)
{
   r[7] = 0xfe;
}
```

```
g[7] = Convert.ToByte(hsbG8.Value);
if (g[7] == 0xff)
{
   g[7] = 0xfe;
}

b[7] = Convert.ToByte(hsbB8.Value);
if (b[7] == 0xff)
{
   b[7] = 0xfe;
}


r[8] = Convert.ToByte(hsbR9.Value);
if (r[8] == 0xff)
{
   r[8] = 0xfe;
}

g[8] = Convert.ToByte(hsbG9.Value);
if (g[8] == 0xff)
{
   g[8] = 0xfe;
}

b[8] = Convert.ToByte(hsbB9.Value);
if (b[8] == 0xff)
{
   b[8] = 0xfe;
}
```

```
r[9] = Convert.ToByte(hsbR10.Value);
if (r[9] == 0xff)
{
   r[9] = 0xfe;
}

g[9] = Convert.ToByte(hsbG10.Value);
if (g[9] == 0xff)
{
   g[9] = 0xfe;
}

b[9] = Convert.ToByte(hsbB10.Value);
if (b[9] == 0xff)
{
   b[9] = 0xfe;
}

r[10] = Convert.ToByte(hsbR11.Value);
if (r[10] == 0xff)
{
   r[10] = 0xfe;
}

g[10] = Convert.ToByte(hsbG11.Value);
if (g[10] == 0xff)
{
   g[10] = 0xfe;
}
```

```
b[10] = Convert.ToByte(hsbB11.Value);
if (b[10] == 0xff)
{
   b[10] = 0xfe;
}

r[11] = Convert.ToByte(hsbR12.Value);
if (r[11] == 0xff)
{
   r[11] = 0xfe;
}

g[11] = Convert.ToByte(hsbG12.Value);
if (g[11] == 0xff)
{
   g[11] = 0xfe;
}

b[11] = Convert.ToByte(hsbB12.Value);
if (b[11] == 0xff)
{
   b[11] = 0xfe;
}

r[12] = Convert.ToByte(hsbR13.Value);
if (r[12] == 0xff)
{
   r[12] = 0xfe;
}
```

```
g[12] = Convert.ToByte(hsbG13.Value);
if (g[12] == 0xff)
{
   g[12] = 0xfe;
}

b[12] = Convert.ToByte(hsbB13.Value);
if (b[12] == 0xff)
{
   b[12] = 0xfe;
}


r[13] = Convert.ToByte(hsbR14.Value);
if (r[13] == 0xff)
{
   r[13] = 0xfe;
}

g[13] = Convert.ToByte(hsbG14.Value);
if (g[13] == 0xff)
{
   g[13] = 0xfe;
}

b[13] = Convert.ToByte(hsbB14.Value);
if (b[13] == 0xff)
{
   b[13] = 0xfe;
```

```
}

r[14] = Convert.ToByte(hsbR15.Value);
if (r[14] == 0xff)
{
   r[14] = 0xfe;
}

g[14] = Convert.ToByte(hsbG15.Value);
if (g[14] == 0xff)
{
   g[14] = 0xfe;
}

b[14] = Convert.ToByte(hsbB15.Value);
if (b[14] == 0xff)
{
   b[14] = 0xfe;
}

r[15] = Convert.ToByte(hsbR16.Value);
if (r[15] == 0xff)
{
   r[15] = 0xfe;
}

g[15] = Convert.ToByte(hsbG16.Value);
if (g[15] == 0xff)
{
   g[15] = 0xfe;
```

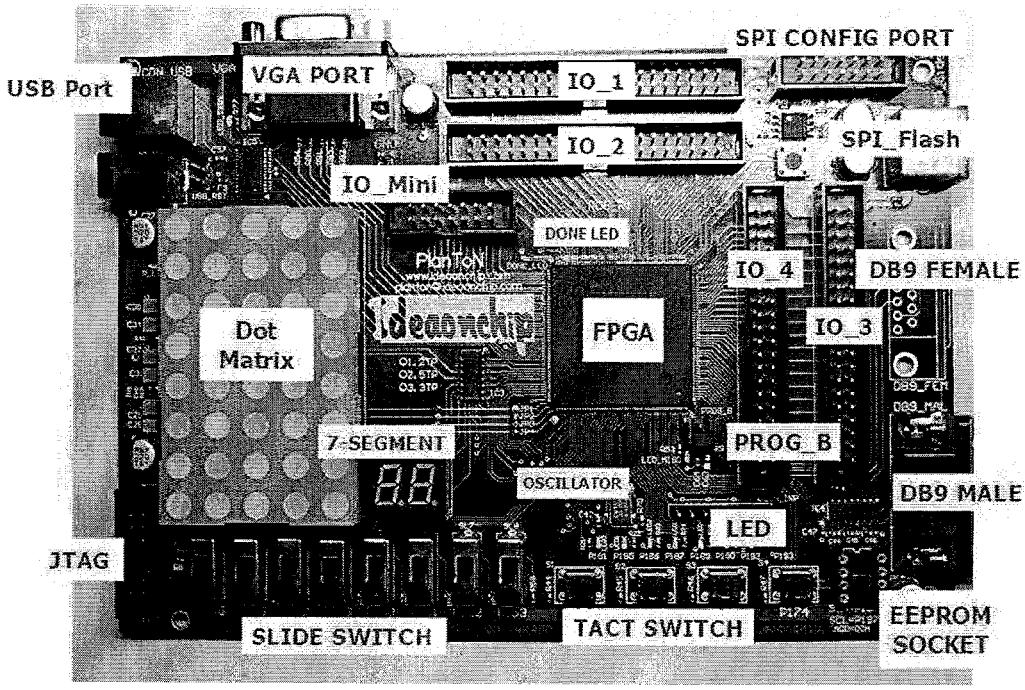
```
    }

    b[15] = Convert.ToByte(hsbB16.Value);
    if (b[15] == 0xff)
    {
      b[15] = 0xfe;
    }

    send_header();
    int x,y;

      for (y = 0; y < 16; y++)
      {
        for (x = 0; x < 16; x++)
        {
          dat[(x * 3)] = r[y];
          dat[(x * 3) + 1] = g[y];
          dat[(x * 3) + 2] = b[y];
          serialPort1.Write(dat, 0, 3);
        }
      }

}

private void hsbG_Scroll(object sender, ScrollEventArgs e)
{
  //send_scrollbar();
}

private void hsbB_Scroll(object sender, ScrollEventArgs e)
```

```
{
    //send_scrollbar();
}

private void btnClear_Click(object sender, EventArgs e)
{
    send_header();
    int x;
    byte[] dat = new byte[768];

    for (x = 0; x < 256; x++)
    {
        dat[x * 3] = 0;
        dat[(x * 3) + 1] = 0;
        dat[(x * 3) + 2] = 0;
    }

    serialPort1.Write(dat, 0, 768);
}

private void btnScSend_Click(object sender, EventArgs e)
{
    send_scrollbar();
}

private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog ofd = new OpenFileDialog();
    if (ofd.ShowDialog() == DialogResult.Cancel)
        return;
```

```
    bm = (Bitmap)Bitmap.FromFile(ofd.FileName);

    SendDat_serial();
}



private void btnShow_Click(object sender, EventArgs e)
{

    MyThread = new Thread(new ThreadStart(show));
    MyThread.Start();

}

private void show()
{
    int x, y;
    byte[] dat = new byte[768];


    for (int blue = 0; blue < 0xf0; blue += 25)
    {
        for (int green = 0; green < 0xf0; green += 25)
        {
            for (int red = 0; red < 0xf0; red += 25)
            {
                send_header();
                for (y = 0; y < 16; y++)
                {
```

```
                for (x = 0; x < 16; x++)
                {
                    dat[(x * 3)] = (byte)red;
                    dat[(x * 3) + 1] = (byte)green;
                    dat[(x * 3) + 2] = (byte)blue;
                    serialPort1.Write(dat, 0, 3);
                }
            }
            Thread.Sleep(50);
        }
    }
}
send_header();
dat = new byte[768];

for (x = 0; x < 256; x++)
{
    dat[x * 3] = 0;
    dat[(x * 3) + 1] = 0;
    dat[(x * 3) + 2] = 0;
}
serialPort1.Write(dat, 0, 768);
}
private void btnStop_Click_1(object sender, EventArgs e)
{
    MyThread.Abort();
    btnClear_Click(sender, e);
}
}
}
```


Digital Design
Ideaonchip



PlanTon Board
FPGA Development Kit

Feature

- FPGA : Xilinx Spartan 3E XC3S250E-4PQ208
- Internal memory 221Kbit
- Male and Female DB9 RS232 Port
- 0.30 INCH 7- Segment 2 Digit Green color
- Dot Matrix LED 5x8 Size 5mm Row Anode, Green color
- USB 2.0 Port
- VGA Port (8 color)
- Atmel 4Mbit SPI Flash

Other Description

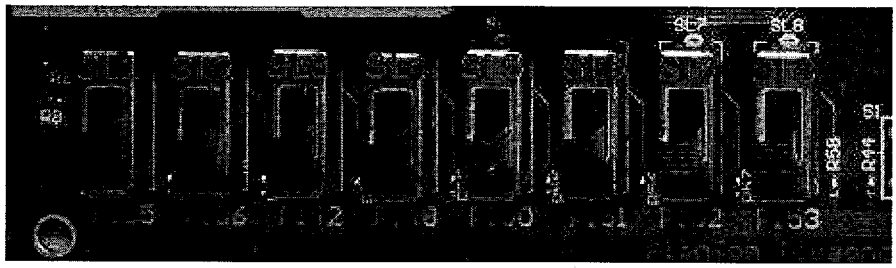
- Gold Plate PCB
- High Quality IC and almost SMD Type on Board
- JTAG Download Cable
- 5 Tact Switches and 8 Slide Switches
- 9 LED SMD Type
- 4 High Speed IO Port (66 IO)
- 1 Low Speed IO Port (12 IO)
- I2C EEPROM (Only Socket EEPROM is option)
- Done LED
- High Quality 50 MHz Oscillator SMD Type
- Ground probe holder
- Power Supply Test point (1.25V , 2.5V , 3.3V)
- Schematic , Example Code , User Guide , Installation Guide include in CD
- ISE Webpack and Modelsim XE include CD

สำหรับนักศึกษาในระดับปริญญาตรีขึ้นไป ถ้ามีเอกสารรับรองจากอาจารย์ที่ปรึกษา จะสามารถนำมาอ้างอิงเพื่อขอซื้อบอร์ดในราคาพิเศษได้ครับ

รายละเอียดอุปกรณ์บนบอร์ดทดลอง

Switch

Slide Switch



บนบอร์ด PlanTon Kit จะมี Slide Switch อยู่ทั้งหมด 8 Switch ประกอบไปด้วย SL1 ถึง SL8 โดยที่ Slide Switch นี้ จะต่อกับ Port ของ FPGA โดยตรง

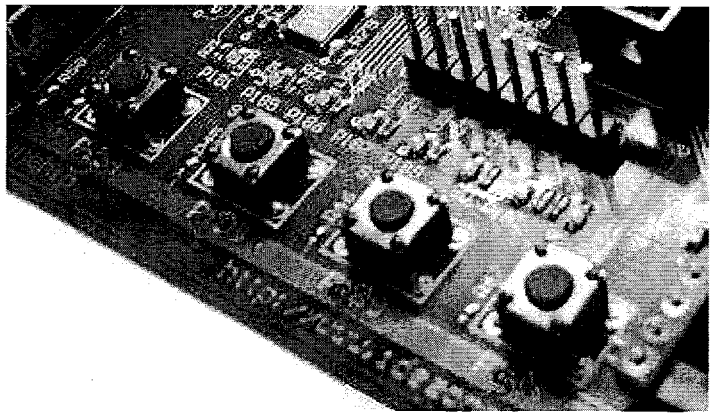
ตำแหน่งตามสภาวะ Logic ของ Slide Switch

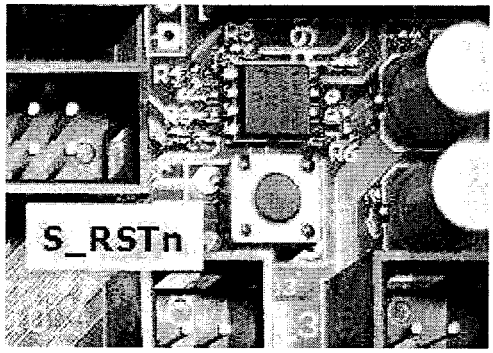
เมื่อดัน Slide Switch ขึ้นไปด้านบนสภาวะ Logic ที่ขา FPGA จะเป็น '1' และในทางกลับกันถ้าดัน Slide Switch ลงสภาวะ Logic ที่ขาของ FPGA จะเป็น '0'

ตำแหน่งของ Slide Switch ที่เชื่อมต่อกับขาของ FPGA

ตำแหน่ง Slide SW	FPGA Port
SL1	P135
SL2	P136
SL3	P142
SL4	P148
SL5	P150
SL6	P151
SL7	P152
SL8	P153

Tact Switch



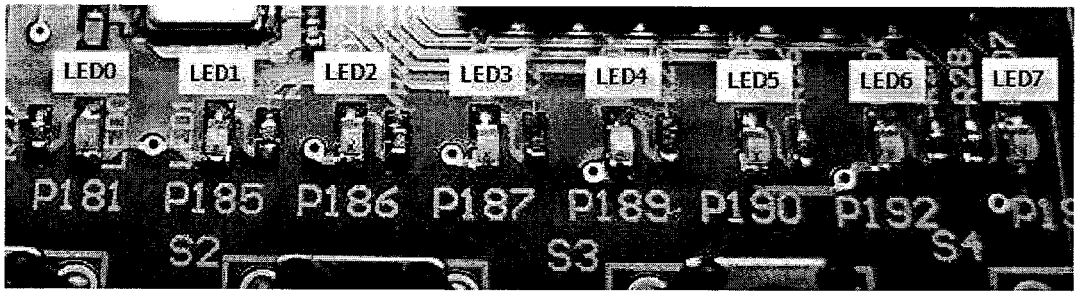


Tact Switch เป็นสวิทช์ที่เป็นลักษณะ “กดติด ปล่อยดับ” โดยสภาวะ Logic ในขณะกดคือ ‘0’ และในขณะปล่อยคือ ‘1’ Port ของ FPGA ที่เชื่อมต่อกับ Tact Switch แสดงดังตารางต่อไปนี้

ตำแหน่ง Tact Switch	FPGA Port
S1	P154
S2	P159
S3	P169
S4	P174
S_RSTn	P54

LED

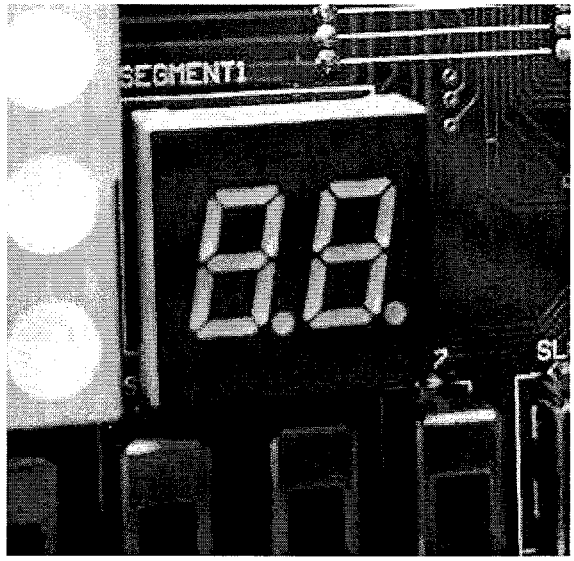
8 user LED



PlanTon มี LED ชนิด SMT 8 ดวง ซึ่ง LED เหล่านี้จะต่ออยู่กับ Port ของ FPGA โดยตรง โดยมีรายละเอียดดังนี้

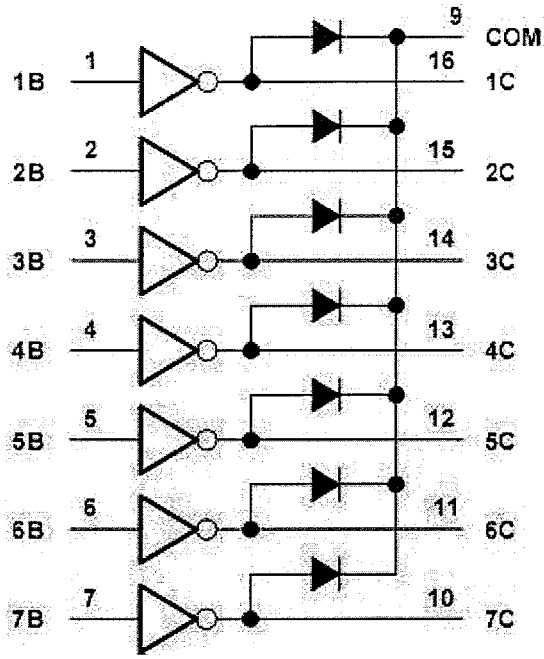
ตำแหน่ง LED	FPGA Port
LED0	P181
LED1	P185
LED2	P186
LED3	P187
LED4	P189
LED5	P190
LED4	P192
LED7	P193

7 SEGMENT

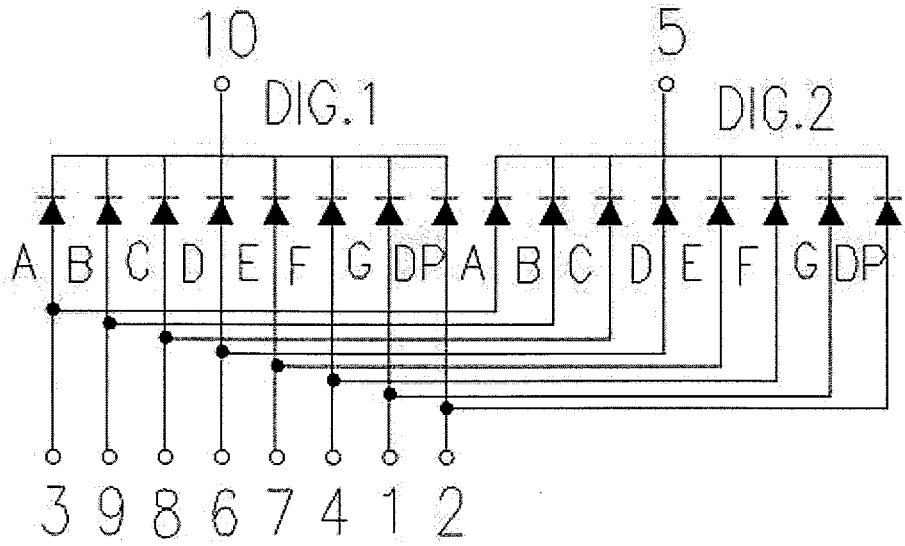


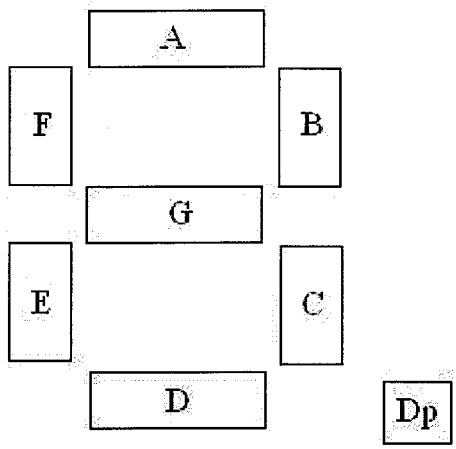
7 SEGMENT 2 DIGIT,0.30 INCH,COMMON CATHODE,
COLOR GREEN บนบอร์ด PlanTon จะเชื่อมต่อกับ Port ของ FPGA โดยมี Buffer คั่นเพื่อป้องกันไม่ให้ FPGA จ่ายกระแสมากจนเกินไป และในส่วนขา Common ของ 7 segment นั้น ต่อผ่าน IC Transistor Array เบอร์ ULN2003 เพื่อลดภาระของการรับกระแส sink อีกเช่นกัน ดังนั้น ถ้าท่านต้องการให้ Column ใด ๆ ของ 7 segment ติด ท่านต้องส่ง สภาวะ Logic '1' มาที่ Column นั้น ๆด้วย เพราะ ULN2003 มีลักษณะ logic เป็นแบบ Inverting ลองดูภาพ Logic Diagram ของ ULN2003 ประกอบนะครับ

logic diagram



วงจรของ 7-Segment ที่ใช้กับบอร์ด PlanTon

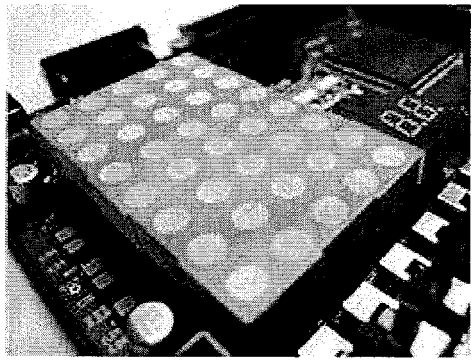




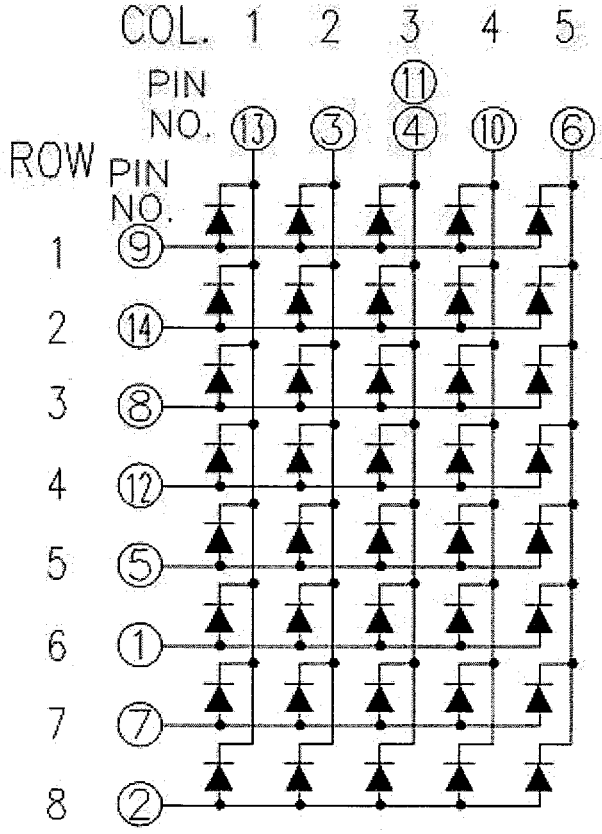
ตารางการเชื่อมต่อ Port ของ 7 segment กับ Port ของ FPGA

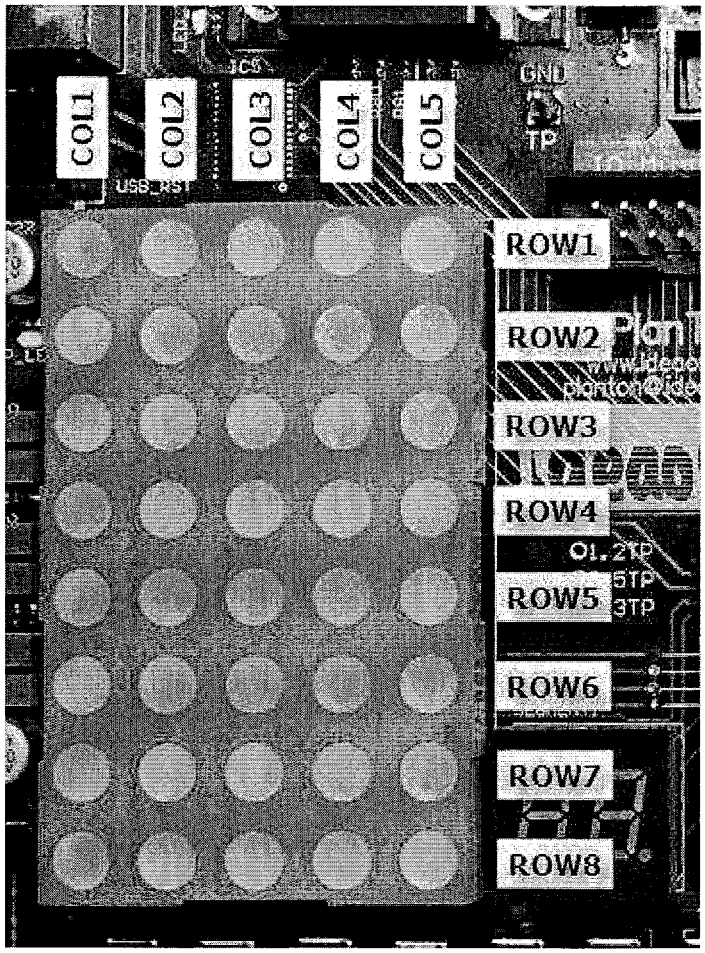
ตำแหน่ง 7-Segment	FPGA Port
A	P178
B	P172
C	P171
D	P167
E	P168
F	P177
G	P180
Dp	P179
Column 1	P144
Column 2	P140

Dot Matrix LED



Dot Matrix LED นั้นเป็นการนำ LED มาวางกันในลักษณะ Matrix ดังแสดงในรูปด้านล่าง





ดังนั้น ถ้าเราต้องการให้ Dot Matrix LED ติดเป็นลักษณะที่เราต้องการเช่น ตัวเลขหรือข้อความต่าง ๆ เราต้องใช้วิธีการ Scan เพื่อให้เห็นเป็นภาพที่เราต้องการ

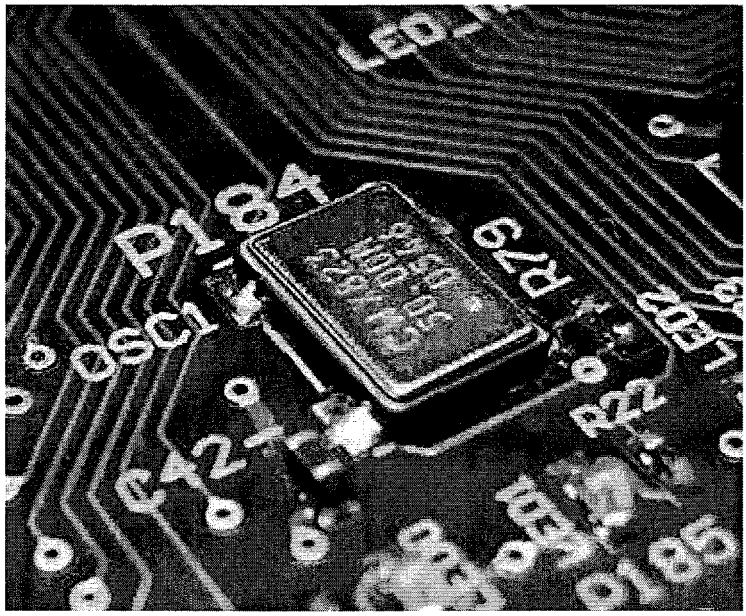
บนบอร์ด PlanTon ได้ต่อ IO ต่าง ๆ ที่เชื่อมต่อกับ Dot Matrix LED ผ่านทาง ULN2003 และ Transister ดังนั้น การใช้งาน Dot Matrix LED จึงไม่จำเป็นต้องใช้ Port ของ FPGA จ่ายกระแสโดยตรง โดยที่ Row ของ Dot Matrix LED จะใช้ Transistor เป็นตัวขยายกระแส และในส่วน Col จะใช้ ULN2003 เพื่อรองรับกระแส Sink จาก Dot Matrix LED ดังนั้นเวลาใช้งาน หากต้องการให้ Column ใด ทำงาน ต้องส่งสัญญาณ '1' มาที่ Column นั้น เพราะวงจรภายในของ ULN2003 มีลักษณะเป็น Inverting

ตารางการเชื่อมต่อขา ของ Dot Matrix LED กับ Port ของ FPGA

ตำแหน่ง Dot Matrix	FPGA Port
ROW1	P164
ROW2	P162
ROW3	P165
ROW4	P163
ROW5	P137
ROW6	P139
ROW7	P134
ROW8	P138
COL1	P161
COL2	P145
COL3	P160
COL4	P147
COL5	P146

OSCILLATOR

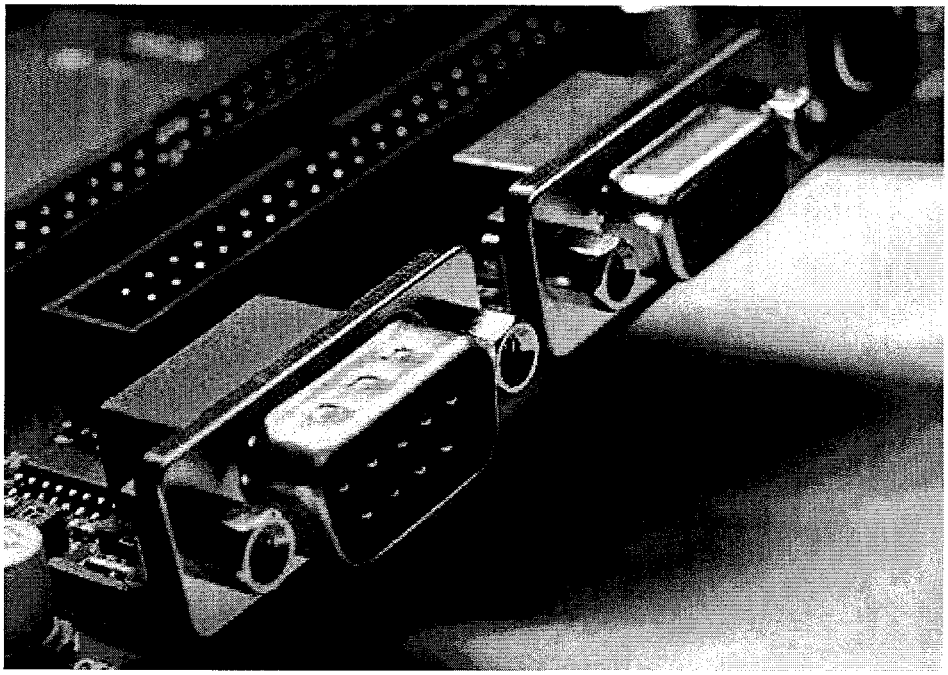
50 MHz SMD OSCILLATOR



Clock Source บนบอร์ด PlanTon นั้นจะเป็น Clock ที่มาจาก Oscillator OSC1 นะครับ ซึ่งมีค่าความถี่ที่ 50MHz เชื่อมต่อกับ Port GCLK ของ FPGA โดยตรง

ตำแหน่ง OSC	FPGA Port
OSC1	P194

RS232



Port RS232 ของ บอร์ดทดลอง PlanTon มีทั้งแบบ Male และ Female โดย IC ที่ใช้สำหรับ ช่วยในการสื่อสารแบบ RS 232 เป็น IC เบอร์ MAX3232 ซึ่งเป็น IC ที่ใช้ในการสื่อสารแบบ RS232 โดยใช้แรงดันไฟเลี้ยง IC ที่ 3.3 V เท่านั้น PIN ที่ต่อกับ ขาของ FPGA แสดงดังตารางด้านล่าง

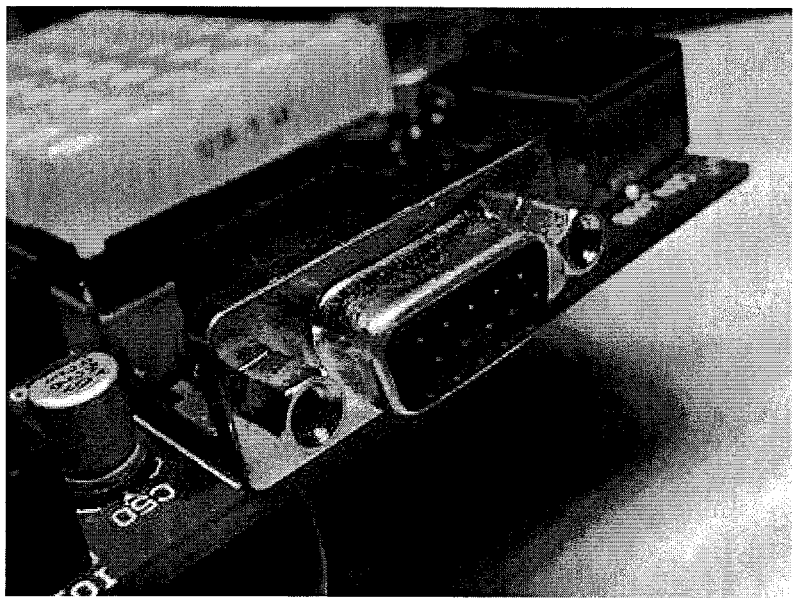
DB9 - Female Connector

ขา RS232	FPGA Port
TXD	P202
RXD	P203

DB9 - male Connector

ขา RS232	FPGA Port
TXD	P200
RXD	P199

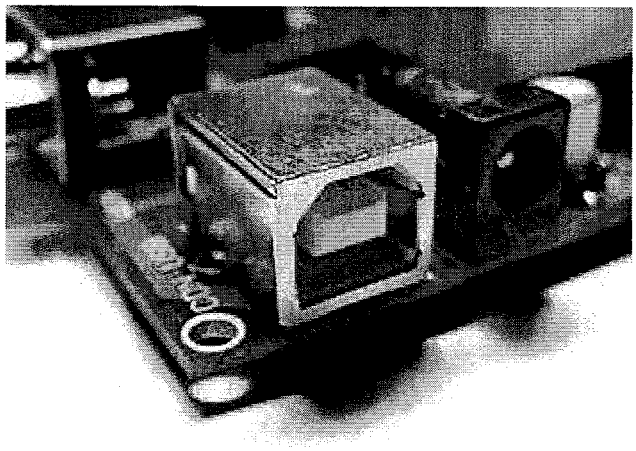
VGA Port



8 color VGA Port ใช้งานเพื่อต่อกับจอ Monitor โดยที่สามารถควบคุมการแสดงสีบนจอแสดงได้สูงสุด 8 สี การใช้งานนั้นต้องมีความเข้าใจกับ Timing ของ VGA ก่อนครับ จึงจะสามารถควบคุมการใช้งานได้ Port ของ VGA นั้นจะมีด้วยกัน 5 Port คือ HS , VS , RED , GREEN , BLUE โดยมีการต่อกับ Port ของ FPGA ดังต่อไปนี้

ตำแหน่ง VGA	FPGA Port
HS	P122
VS	P123
R	P128
G	P127
B	P126

USB To Serial Port



Port USB บน PlanTon เป็นการนำ IC FT232RL มาใช้งาน เพื่อให้เพิ่มความสะดวกในการติดต่อกับ Computer มากขึ้น โดยที่ฝั่ง Computer นั้นเราจะเชื่อมต่อกับ Port USB และทางฝั่ง FPGA เราจะใช้งานเหมือนกับว่าเราต่อกับ Serial Port อยู่ ดังนั้นจำทำให้ง่ายต่อการใช้งาน USB มากขึ้น

ก่อนการใช้งานนั้น จะต้องมีการ Setup Driver ของ USB Port บนคอมพิวเตอร์ก่อน โดยที่ท่านสามารถ Download Driver เวอร์ชั่น ล่าสุดได้ที่เว็บไซต์ http://www.ideaonchip.com หรือ เปิดใน Folder USBDriver กับ DVD ที่แถมมากับบอร์ด

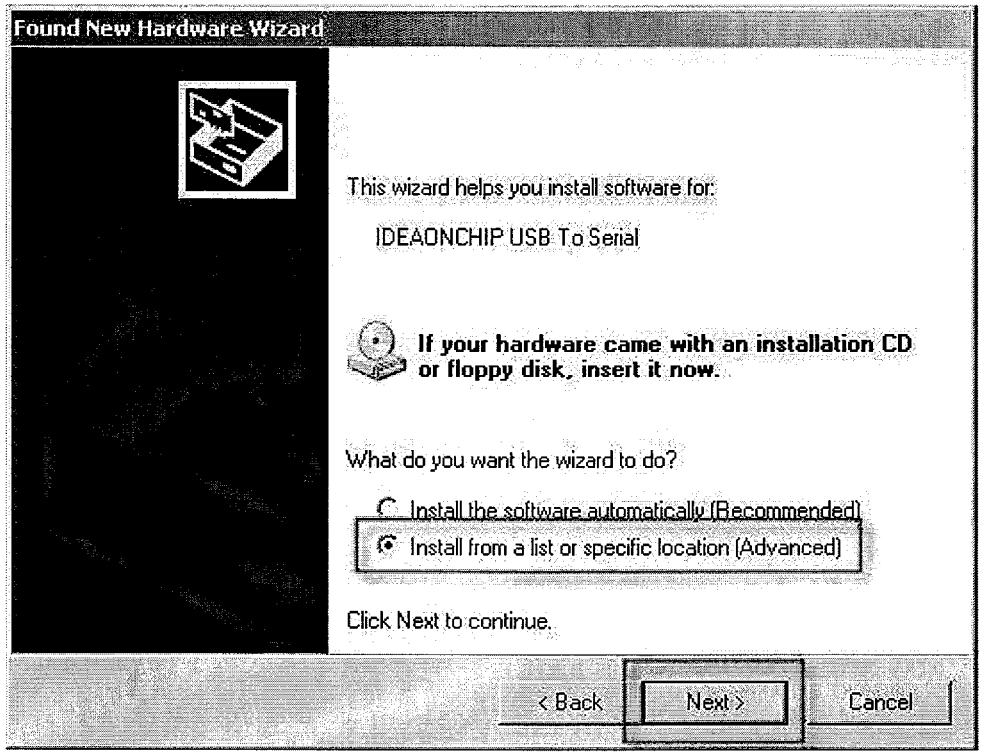
ขั้นตอนการ Setup Driver บน Computer

ให้ทำการจ่ายไฟให้กับบอร์ดทดลอง PlanTon และเชื่อมต่อสาย USB เข้ากับบอร์ด และอีกด้านหนึ่งของสายเชื่อมต่อกับ Port USB ของ Computer

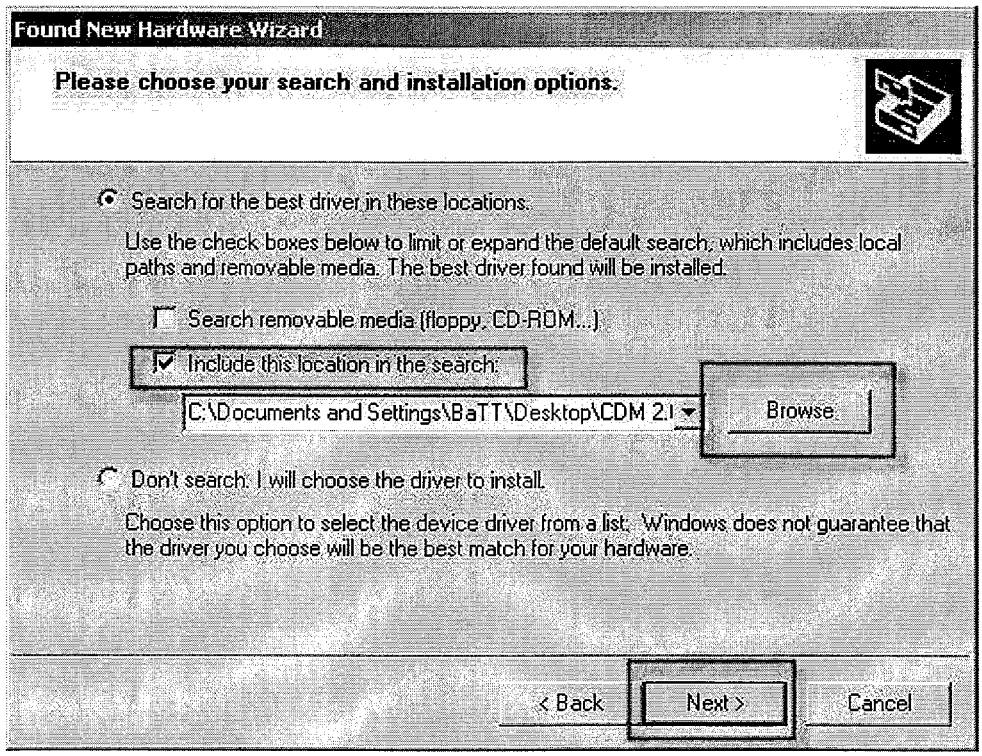
เมื่อเสียบสาย USB เข้าด้วยกันแล้ว บน Computer จะพบว่ามีอุปกรณ์ใหม่ โดยมีข้อความแจ้งดังรูปด้านล่าง

จากนั้นจะพบกับหน้าต่าง Found New Hardware Wizard ให้เลือกที่ No, Not this time และคลิกที่ Next

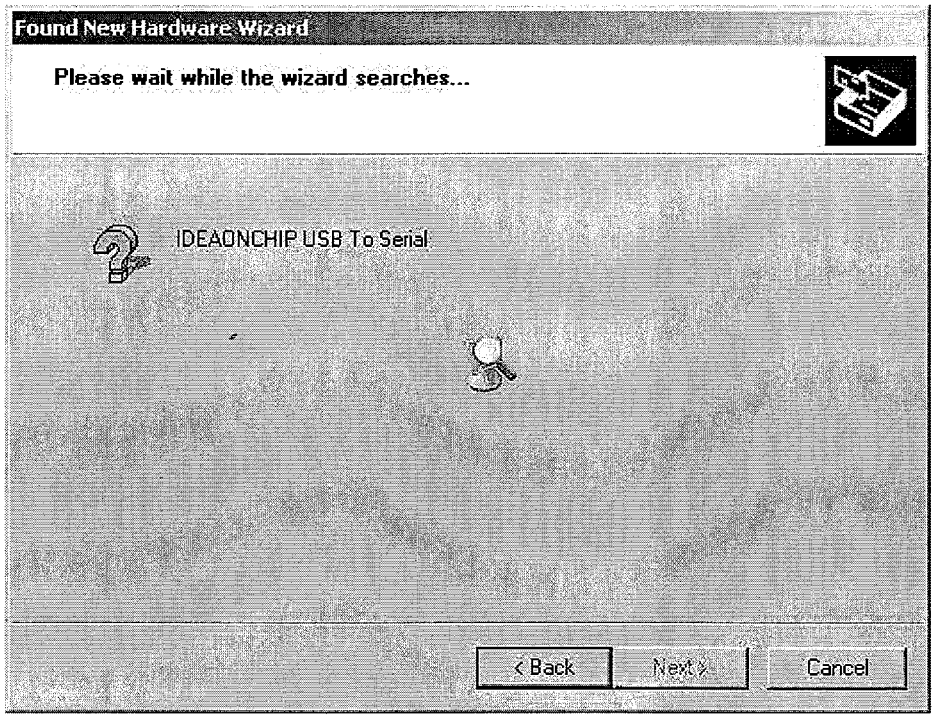
ต่อไปให้เราเลือก Install from a list or specific location (Advance) และคลิกเลือกที่ Next



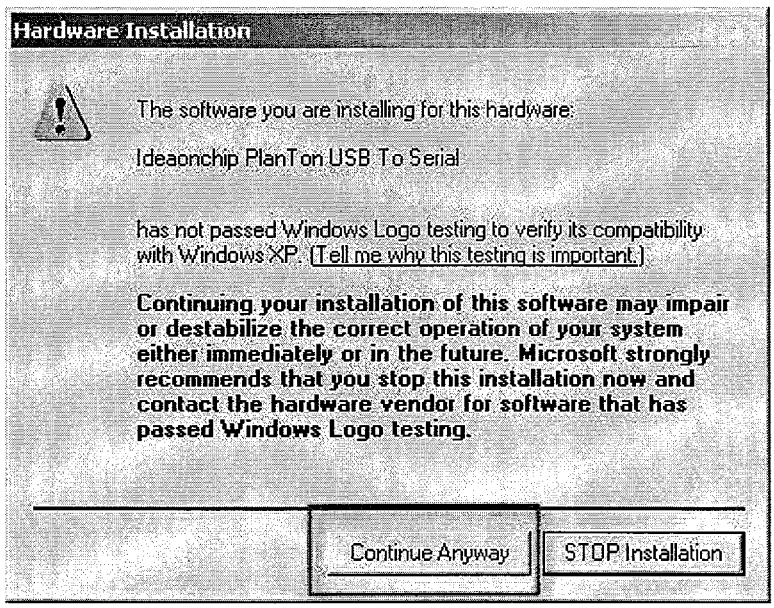
ต่อไปให้เลือกที่ Include this location in the search และจากนั้นให้คลิกที่ Browse เพื่อเลือก Folder ที่มี Driver USB อยู่ เมื่อเลือกเสร็จแล้วให้กดที่ Next ครับ



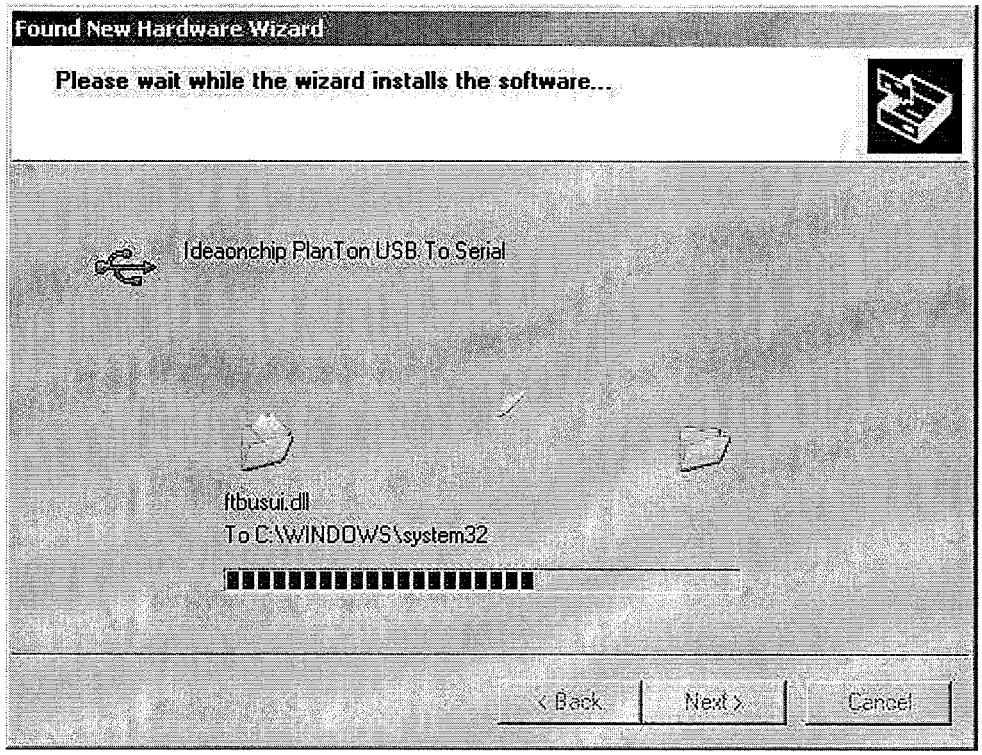
จากนั้น windows จะเริ่มทำการค้นหาและติดตั้ง Driver โดยหน้าจอจะแสดงดังภาพด้านล่าง



จากนั้นจะมีข้อความแสดงขึ้นมา เกี่ยวกับ Windows Logo Testing ให้เราเลือก Continue Anyway เพื่อดำเนินการต่อ



จากนั้น Windows จะเริ่มทำการติดตั้ง Driver สำหรับ USB Port



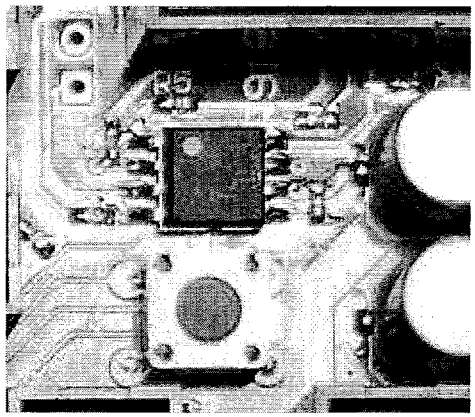
เสร็จแล้วจะขึ้นหน้าจอ Complete ให้เราคลิกเลือกที่ Finish ได้เลยครับ

เมื่อเราทำการ Install เรียบร้อยแล้ว 1 ครั้ง ที่มุมล่างขวาก็จะแสดงว่าพบ USB Serial Port อีกตัวนึงนะครับ ก็ให้เราทำการติดตั้งอีกรอบ โดยวิธีการติดตั้งก็ให้ย้อนกลับไปทำที่ข้อ 3. ใหม่อีกรอบนึงครับ และเมื่อติดตั้งครบ 2 รอบแล้ว ก็เป็นอันเสร็จสิ้นครับ

ตารางการเชื่อมต่อ Port ของ USB Port กับ Port ของ FPGA

ตำแหน่ง USB	FPGA Port
TXD	P133
RXD	P132
RTS#	P130
CTS#	P129

SPI Flash

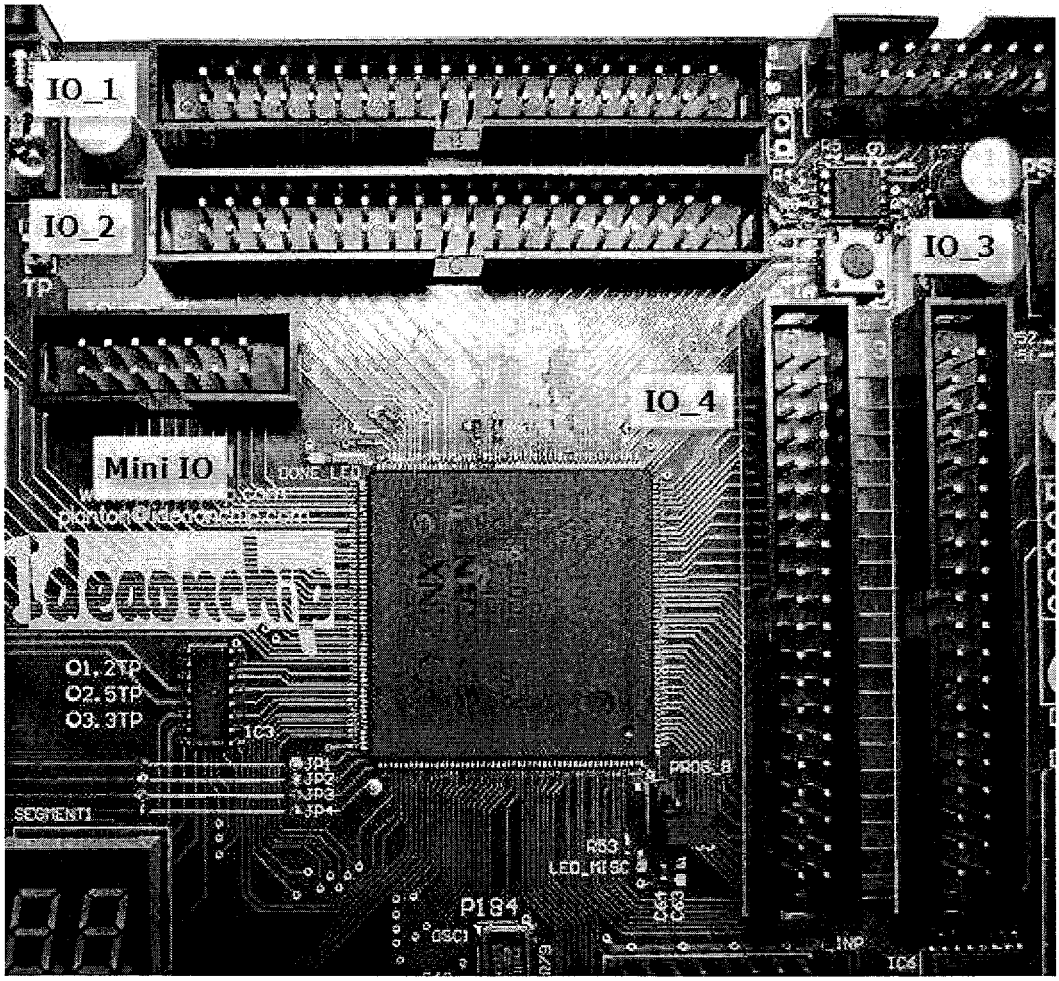


SPI Flash บน PlanTon มีไว้สำหรับเก็บ Program ของ FPGA ไว้ โดยที่เมื่อไฟดับ โปรแกรมที่เราได้ Download ลง FPGA จะหายไป และเมื่อจ่ายไฟให้บอร์ดอีกครั้ง Program จะถูกโหลดจาก SPI Flash ลงไปที่ FPGA โดยอัตโนมัติ แต่ว่าเมื่อ FPGA ถูกโปรแกรมโดย SPI Flash เรียบร้อยแล้ว เราสามารถ ใช้งาน SPI Flash นี้ได้อีก

ตารางการเชื่อมต่อ Port ของ SPI Flash กับ Port ของ FPGA

ตำแหน่งขา SPI Flash	FPGA Port
CS	P55
SI	P61
SO	P87
SCLK	P103

IO Port



Port ที่สำหรับใช้งานเพื่อต่อกับอุปกรณ์ภายนอกของ PlanTon นั้น จะมีอยู่ทั้งหมด 5 IO Port โดยจะเป็น ลักษณะ 40 Pin Box Header 4 ชุด และ 14 Pin Box Header อีก 1 ชุด

Port IO ของ Spartan3E นั้น จะมีลักษณะพิเศษ ต่างจาก FPGA ทั่วไปคือ บาง Port นั้น จะมีความสามารถ เป็นแค่ Input ได้อย่างเดียวเท่านั้น ดังนั้น ในการต่อนำไปใช้งานภายนอก ขอให้ระวังเรื่องนี้ด้วยครับ

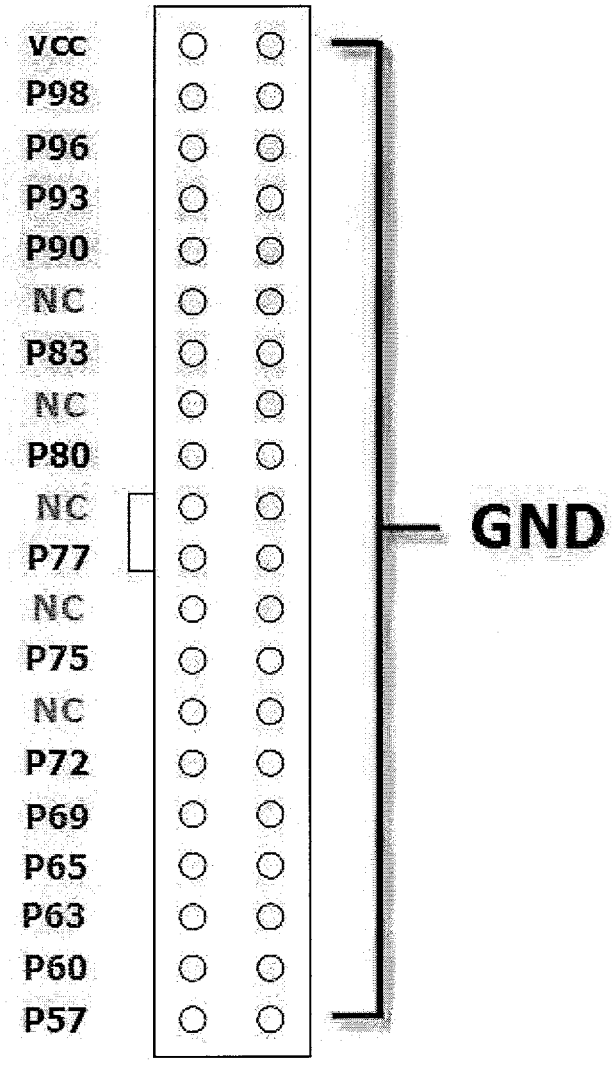
หมายเหตุ :

ตำแหน่ง IO ที่แสดง สีจะแตกต่างกันไป โดย สีแต่ละสีมีความหมายดังนี้ครับ

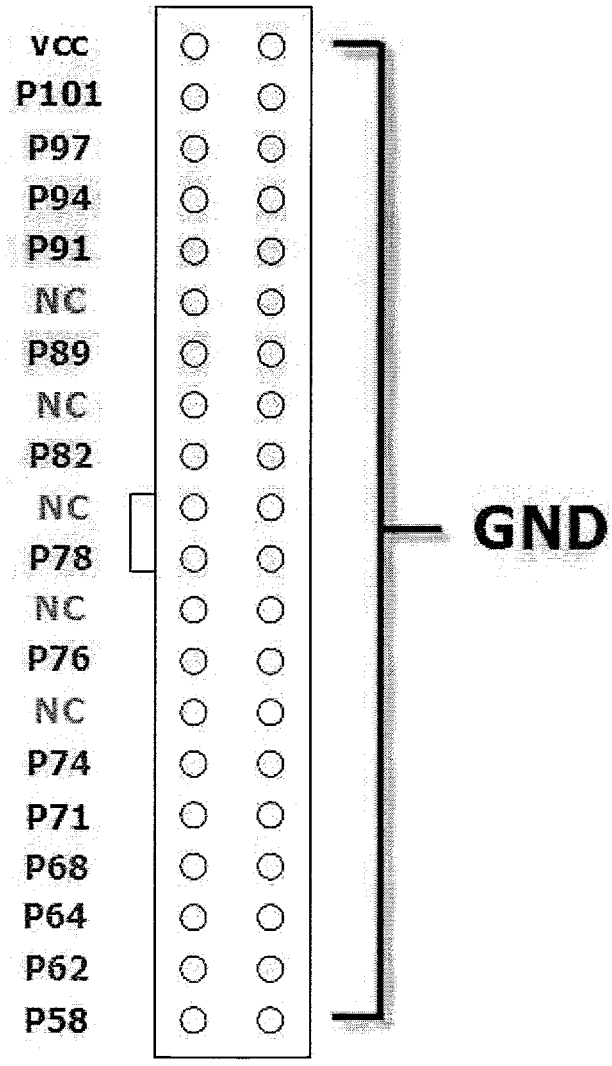
- Input อย่างเดียว
- Input และ Output
- No Connect
- VCC และ GND

ตารางการเชื่อมต่อกับ PORT ต่าง ๆ ของ IO_PORT ต่าง ๆ

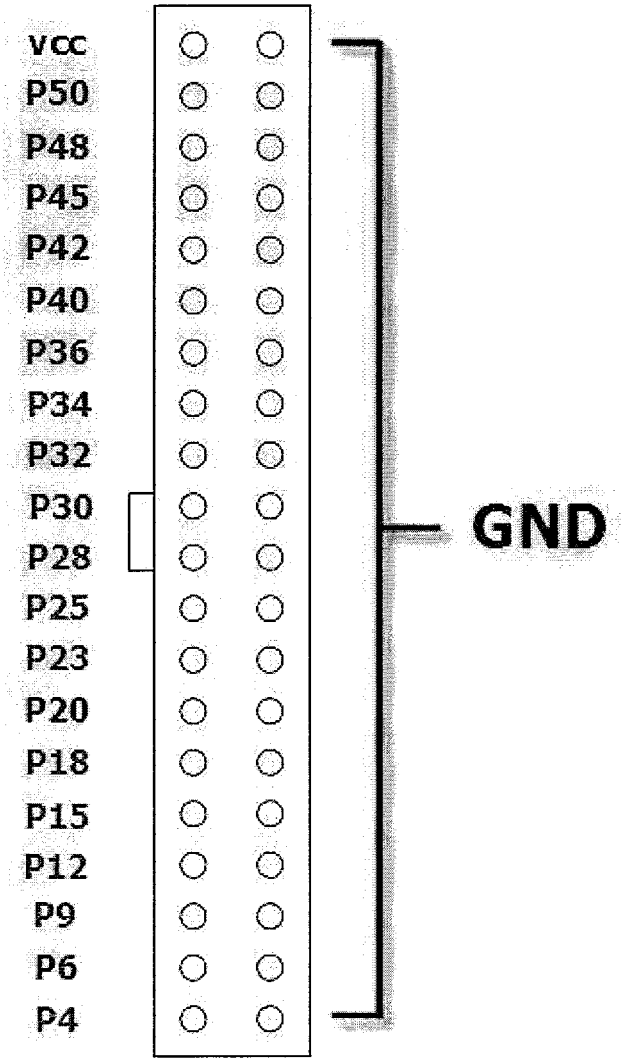
IO_1



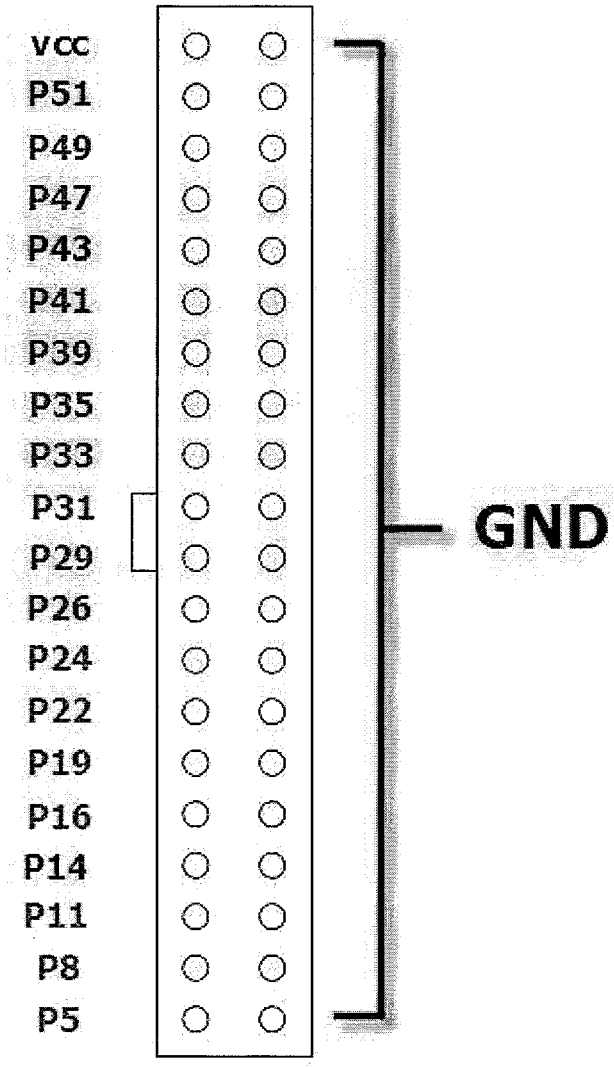
IO_2


VCC
P101
P97
P94
P91
NC
P89
NC
P82
NC
P78
NC
P76
NC
P74
P71
P68
P64
P62
P58
GND

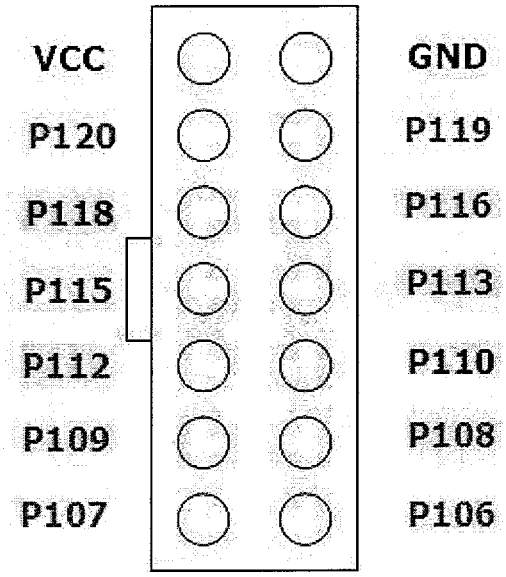
IO_3


VCC
P50
P48
P45
P42
P40
P36
P34
P32
P30
P28
P25
P23
P20
P18
P15
P12
P9
P6
P4
GND

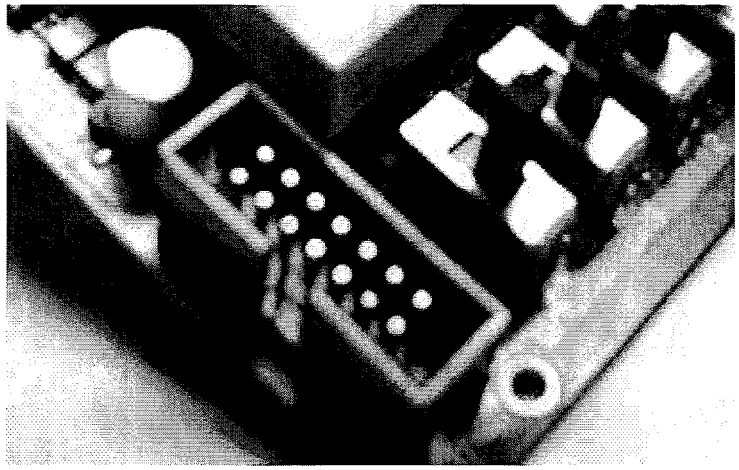
IO_4

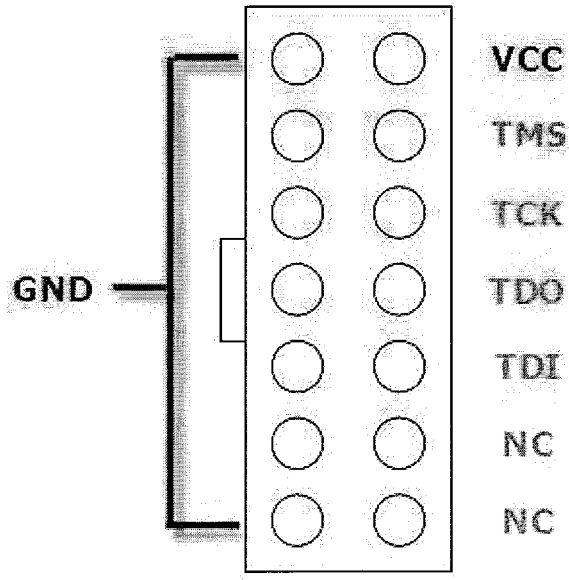

VCC
P51
P49
P47
P43
P41
P39
P35
P33
P31
P29
P26
P24
P22
P19
P16
P14
P11
P8
P5
GND

IO_Mini


VCC
GND
P120
P119
P118
P116
P115
P113
P112
P110
P109
P108
P107
P106

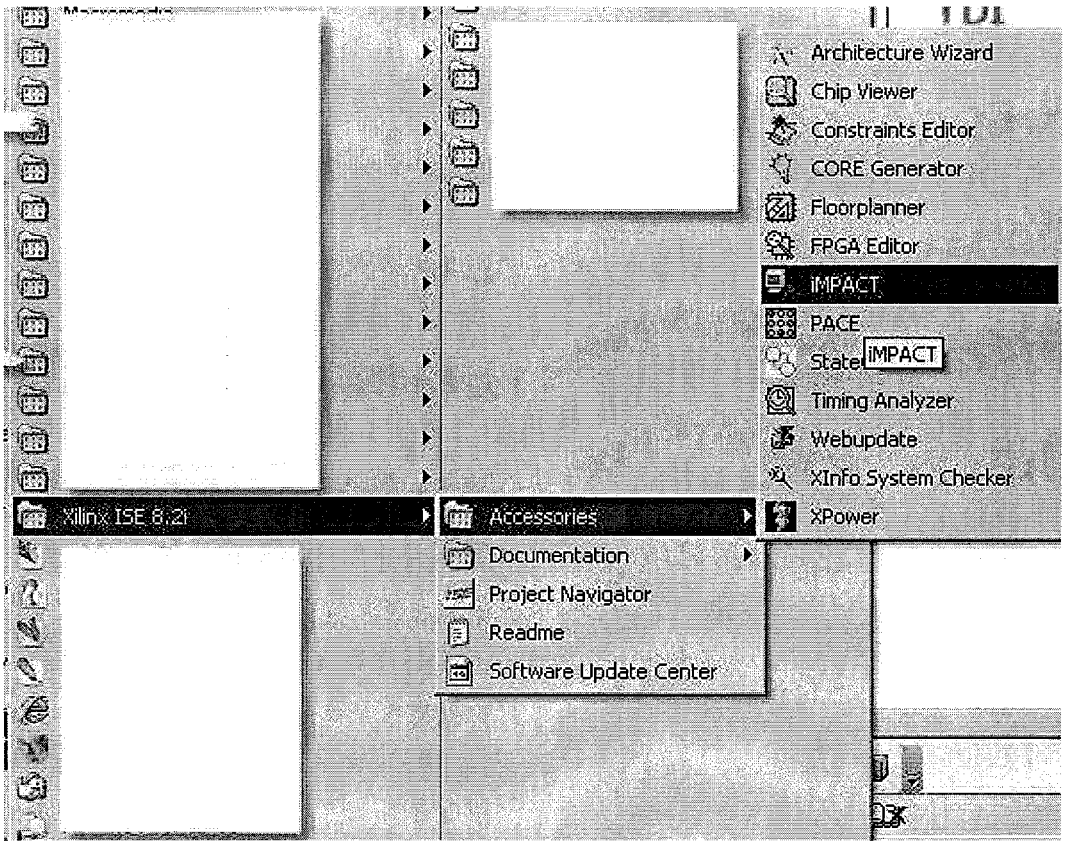
JTAG PORT



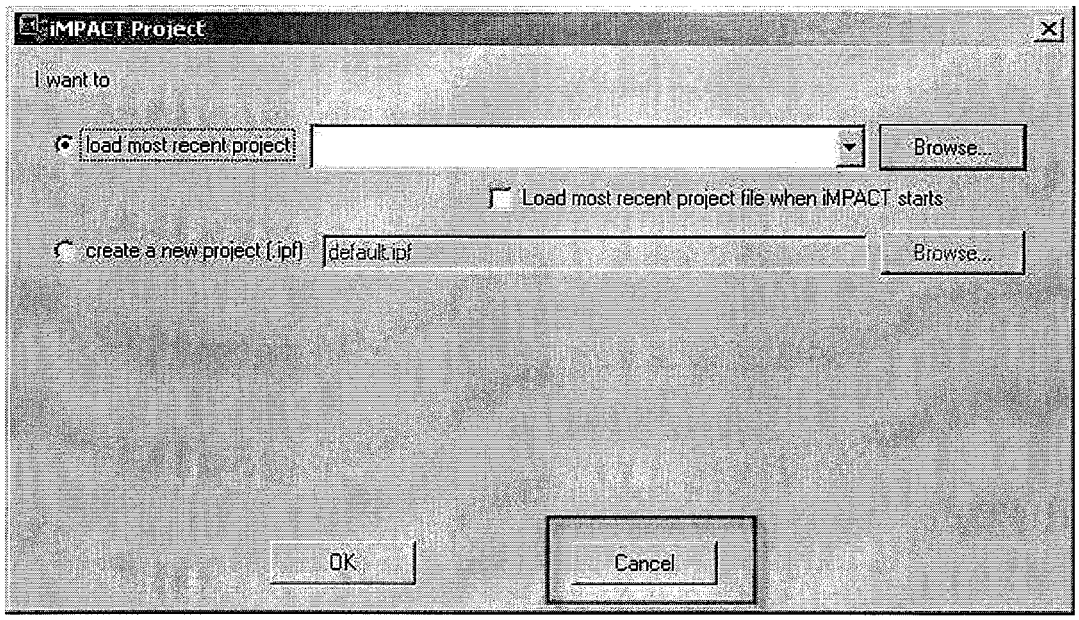


ในการโปรแกรม FPGA นั้น เราจะทำการโปรแกรมผ่านทาง JTAG Port โดยขั้นตอนการโปรแกรมลง FPGA นั้นในขั้นตอนแรกให้เราเสียบสาย JTAG ที่ Port JTAG ก่อน จากนั้นให้เริ่มจ่ายไฟให้แก่บอร์ดทดลอง

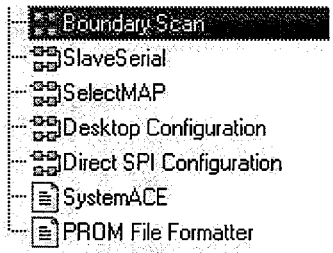
ให้เราเริ่มเรียกโปรแกรม Impact ขึ้นมา ดังรูปด้านล่างครับ



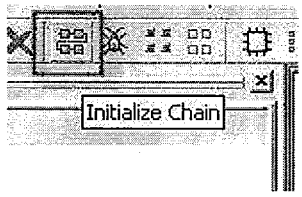
จากนั้นให้เราคลิกที่ cancel ก่อนครับ ตามรูปด้านล่างครับ



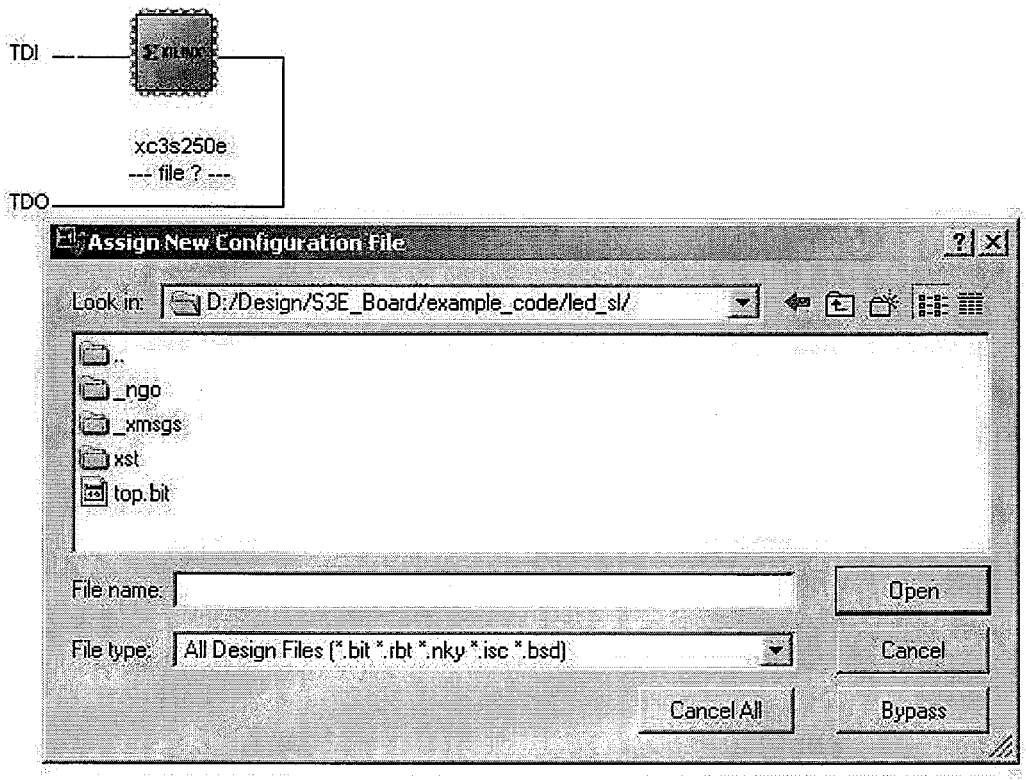
และให้เราเลือก Mode Boundary Scan โดยการ Double Click ตามรูปด้านล่างครับ



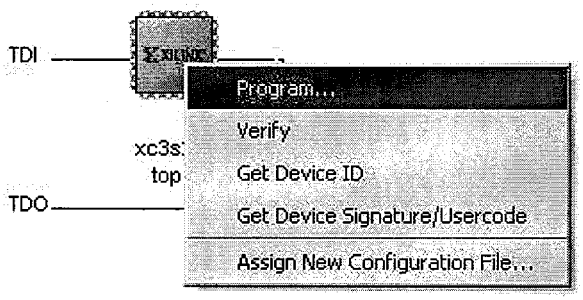
ให้เราคลิกที่ Initialize Chain เพื่อทำการหา อุปกรณ์ที่เชื่อมต่อกับ JTAG Port นะครับ



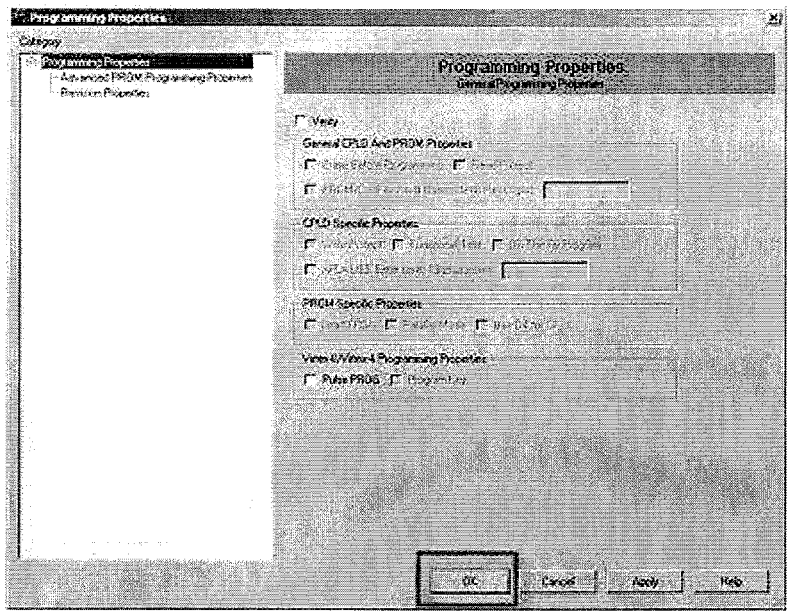
เมื่อโปรแกรมพบอุปกรณ์ที่เชื่อมต่ออยู่บน JTAG แล้ว จะแสดงหน้าต่างให้เราเลือกไฟล์ .bit เพื่อทำการโปรแกรมให้กับ FPGA ดังนั้นให้เราเลือกไฟล์ .bit ที่เราได้สร้างไว้แล้ว



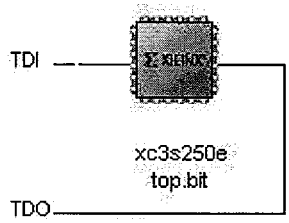
เมื่อเราได้ทำการเลือกไฟล์เสร็จแล้ว ขั้นตอนต่อไปคือการโปรแกรมลงไปที่ FPGA ให้เราคลิกขวาไปที่ FPGA แล้วเลือกที่ Program



ในหน้าต่าง Program ให้เราเลือก OK



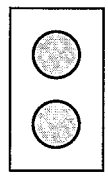
จากนั้นจะเริ่มการโปรแกรม FPGA เมื่อโปรแกรมเสร็จแล้วจะแสดงข้อความว่า Program succeeded



SPI Port

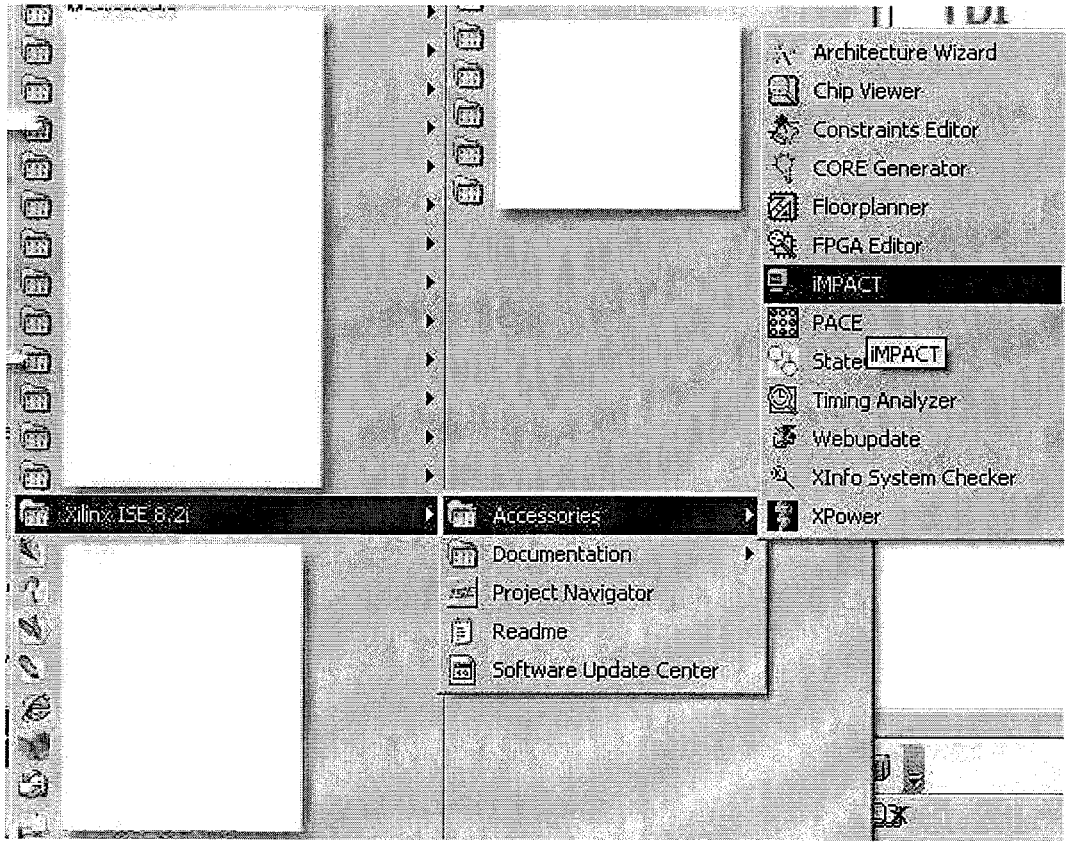
การโปรแกรมลง SPI Flash นั้น เพื่อที่จะทำให้ เวลาที่เราตัดแหล่งจ่ายของ Board ทดลองแล้วและจ่ายไฟเข้าไปใหม่ FPGA ยังสามารถทำงานได้ตามปรกติ

ในการทดลองโดยทั่วๆ ไป เวลาเราทดลองเขียน Code ไม่ว่าจะเป็นภาษาอะไรก็ตาม เราจะทำการสังเคราะห์ วงจรให้ออกมาเป็นไฟล์ .bit ไฟล์ .bit นี้เราจะนำไปโปรแกรมผ่านทาง JTAG Port เพื่อทดสอบการทำงานของ Program ที่เราได้สร้างขึ้น ถ้าหากว่าโปรแกรมที่เราสร้างขึ้นนั้นสมบูรณ์เรียบร้อยแล้ว เราจึงทำการโปรแกรมลง SPI Flash อีกครั้งนึง โดยให้นำสาย Download มาเสียบที่ SPI Port ที่อยู่บริเวณมุมบนขวาของ PlanTon Board และจากนั้น ให้เรา Set Jumper PROG_B ให้เป็น Close นะครับ พูดง่ายๆ คือเสียบ Jumper ลงไปให้ทั้ง 2 Pin ชื่อตกัน ลองดูตามรูปนะครับ

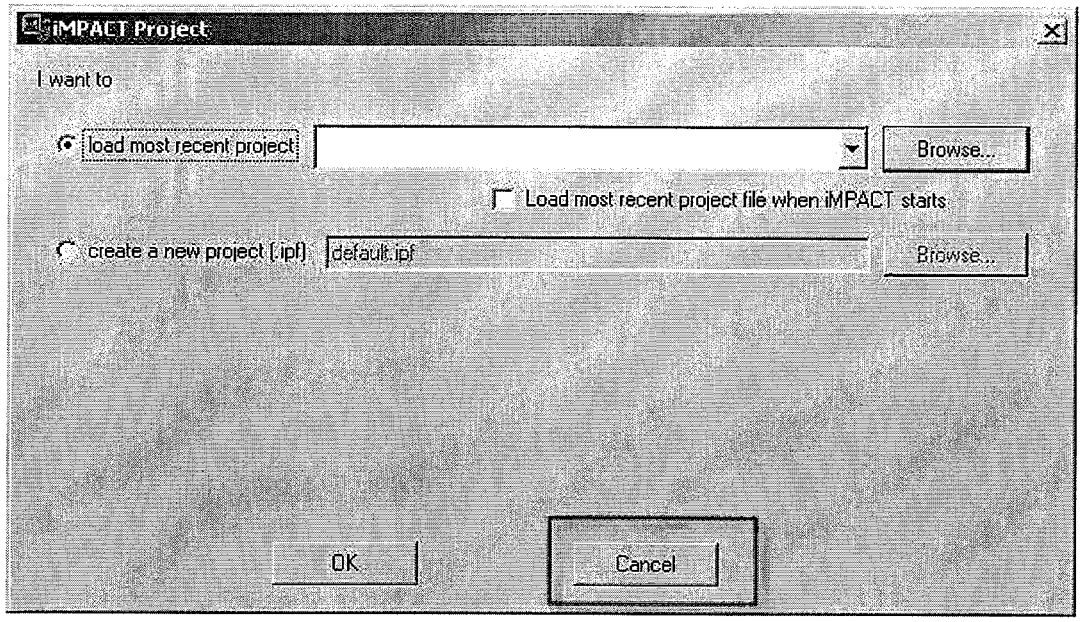


ขั้นตอนต่อไป ให้ทำตามขั้นตอนดังนี้

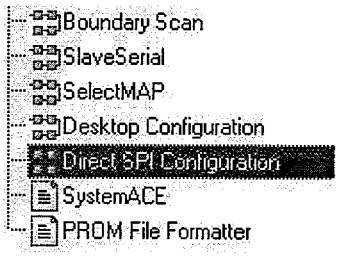
ให้เรียกโปรแกรม Impact ขึ้นมา



จะแสดงหน้าต่าง Impact Project ขึ้นมา ให้เรากด cancel ดังรูปด้านล่าง



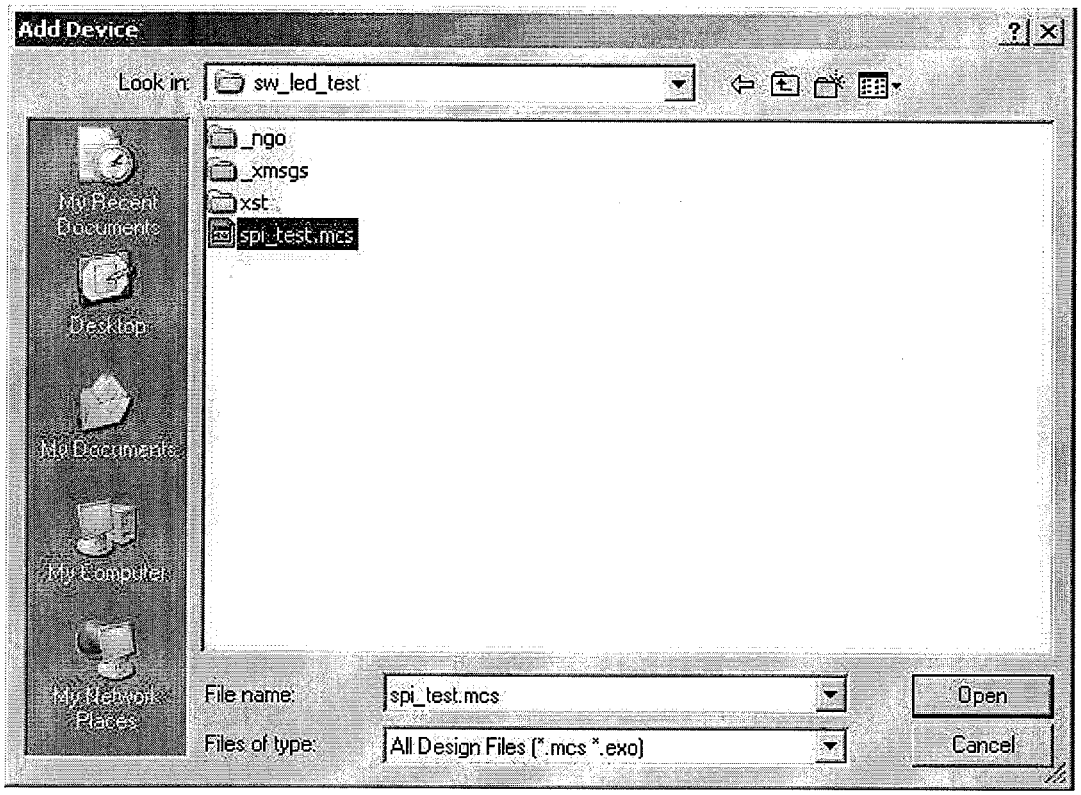
จากนั้นให้เรา Double Click ที่ Direct SPI Configuration ดังรูปด้านล่าง



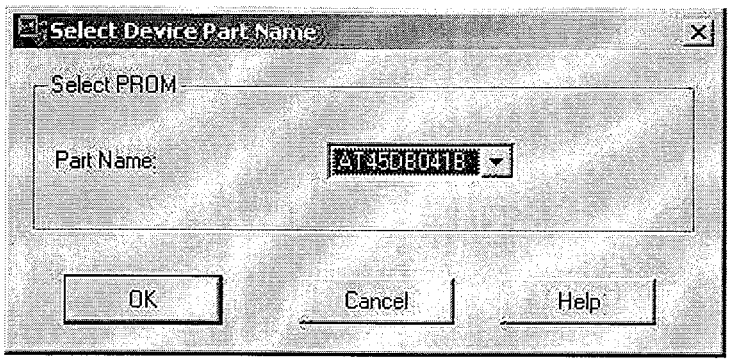
ขั้นตอนต่อไป ให้เราคลิกขวาบริเวณ หน้าต่างใหญ่ และให้คลิกเลือกที่ Add SPI Device

Right click to Add Device or Identify Device

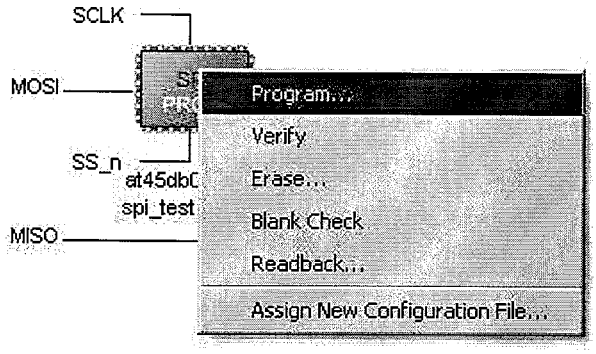
และเราจะเห็นว่า เราต้องทำการเลือกไฟล์ ที่จะโปรแกรมลง SPI Flash นะครับ ดังนั้นให้เราทำการเลือกไฟล์ได้เลยครับ



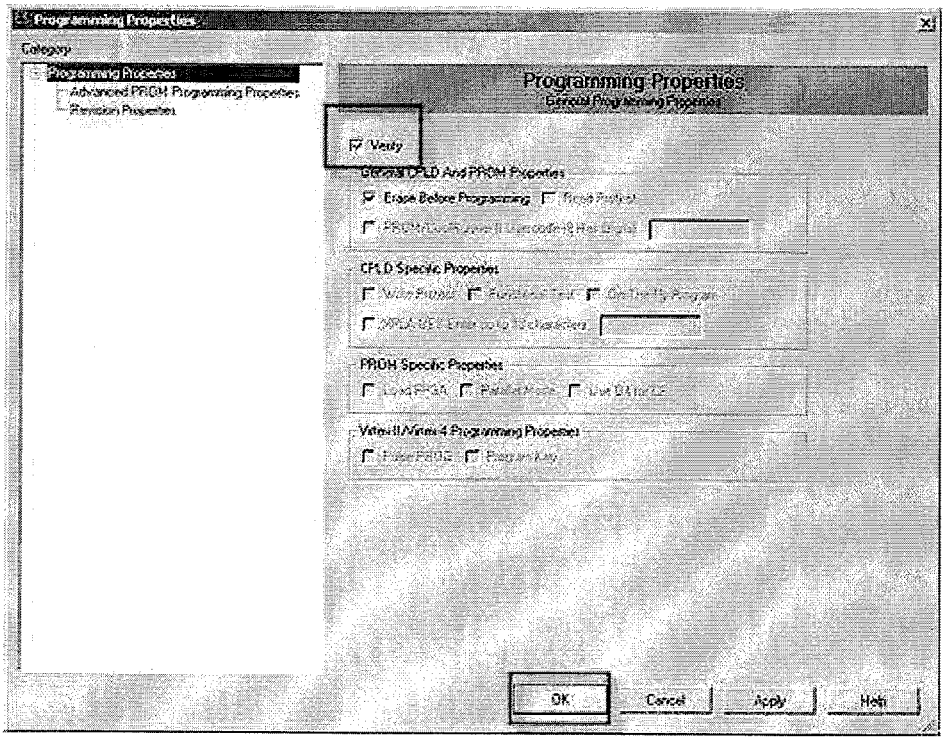
ต่อไปให้เราทำการเลือก ชนิดของ SPI Flash ที่อยู่บน PlanTon นะครับ ในที่นี้ให้เราเลือกเบอร์ AT45DB041B ครับ จากนั้นก็กด OK



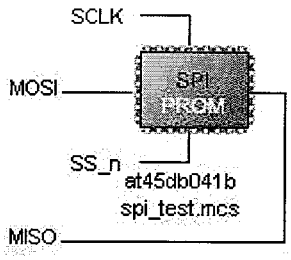
ต่อไปให้เราทำการ โปรแกรมลงสู่ SPI โดยการคลิกขวาที่ตัว SPI Flash และเลือกที่โปรแกรมครับ

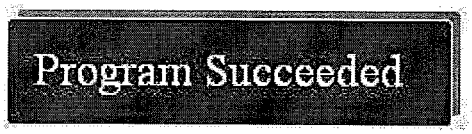


ในขั้นตอนต่อไปนี้ให้เราเลือกที่ verify ด้วยนะครับ (โดยปรกติจะถูกเลือกไว้อยู่แล้ว) เพื่อความแน่นอนของข้อมูลที่เราทำการโปรแกรมลงไปครับ จากนั้นก็ให้เราคลิกที่ OK ครับ



ถ้าการ Program ลง SPI Flash เรียบร้อย จะแสดงดังนี้ครับ





ขั้นตอนสุดท้ายให้เราถอด Jumper PROG_B ออกครับ และเราจะเห็นว่าไฟ LED Done จะติดสว่าง ดังนั้นต่อจากนี้ไปเมื่อเราปิดแหล่งจ่ายไฟ และเปิดขึ้นใหม่ FPGA จะถูก Program ด้วยข้อมูลที่อยู่ใน SPI Flash ครับ