เครื่องควบคุม CMOS IMAGE SENSOR โดย FPGA

CMOS IMAGE SENSOR CONTROLLER WITH FPGA

โดย

นายวันชัย ฉีเฉิศ

นายวิฑูร ศรีวิชัย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

เครื่องควบคุม CMOS IMAGE SENSOR โดย FPGA

CMOS IMAGE SENSOR CONTROLLER WITH FPGA

โดย

นายวันชัย ลีเลิศ 46015239

นายวิทูร ศรีวิชัย 46015241

อาจารย์ที่ปรึกษา

รศ.ดร.ชูชาติ ปิณฑวิรุจน์

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องควบคุม CMOS IMAGE SENSOR โดย FPGA

ผู้จัดทำ

1. นายวันชัย ลีเลิศ รหัส 46015239

2. นายวิทูร ศรีวิชัย รหัส 46015241

.....

(รศ.ดร.ชูชาติ ปิณฑวิรุจน์)

อาจารย์ที่ปรึกษา

เครื่องควบคุม CMOS IMAGE SENSOR โดย FPGA

นายวันชัย ลีเลิศ 46015239
นายวิทูร ศรีวิชัย 46015241
รศ.ดร. ชูชาติ ปิณฑวิรุจน์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2548

บทคัดย่อ

ปริญญานิพนธ์นี้เกี่ยวกับการสร้างเครื่องควบคุม CMOS image sensor โดยการประยุกต์ใช้ FPGA อุปกรณ์ที่ใช้ในการออกแบบ ประกอบด้วยสามส่วนประกอบหลักคือ CMOS image sensor, FPGA และ RAMในส่วนของ CMOS image sensor จะให้เอาท์พุทเป็นสัญญาณดิจิตอลแทนค่าในแต่ละพิกเซล ซึ่งข้อมูลดิจิตอลนี้สามารถที่จะดึงจากเซ็นเซอร์ได้โดยตรง โดยการใช้ FPGA และ RAM เราสามารถทำการโปรแกรม FPGA ให้จัดเก็บค่าข้อมูลพิกเซลไว้ใน RAM ซึ่ง CMOS image sensor จะถูกควบคุมโดยสัญญาณควบคุม และค่าข้อมูลพิกเซลใน RAM จะถูกส่งข้อมูลผ่านพอร์ต USB เพื่อที่จะแสดงผลทางคอมพิวเตอร์

CMOS IMAGE SENSOR CONTROLLER WITH FPGA

Mr. Wanchai Leelert 46015239

Mr. Withoon Sriwichai 46015241

Assoc. Prof. Dr. Chuchart Pintavirooj Advisor

Educational Year 2005

ABSTRACT

This thesis concerns about CMOS image sensor controller with FPGA. The devices used in this construction consist of three main components which are CMOS image sensor, FPGA and RAM. CMOS image sensor produces a digital output representing each pixel which digital data can be pulled straight from the sensor. By using a FPGA and RAM, we can program the FPGA to dump the pixel data straight into RAM. The CMOS image sensor is controlled by the master control signal generator component which the pixel data is buffered in RAM to be uploaded through the USB port to be display on a PC.

กิตติกรรมประกาศ

ในการจัดทำปริญญานิพนธ์นี้ ทางผู้เขียนได้รับความช่วยเหลือ และสนับสนุนจากบุคคลหลายฝ่ายด้วยกันดังนั้นทางผู้เขียนรู้สึกซาบซึ้งเป็นอย่างยิ่งจึงขอขอบพระคุณทุกท่านมา ณ.โอกาสนี้

ขอขอบพระคุณ รศ.ดร.ชูชาติ ปิณฑวิรุจน์ อาจารย์ที่ปรึกษา ที่เป็นผู้ให้ความรู้ความเข้าใจด้านต่างๆในการศึกษารวมถึง ความเอื้อเฟื้อสถานที่ เครื่องมืออุปกรณ์ ตลอดจน คอมพิวเตอร์ที่ใช้ในการทดลองและค้นคว้าเกี่ยวกับโครงงาน

ขอขอบพระคุณ รุ่นพี่ป.โท ผู้ซึ่งให้ความรู้และให้คำแนะนำ เกี่ยวกับการเขียนโปรแกรมด้วยภาษา VHDL และแนะนำอุปกรณ์ต่างๆที่น่าสนใจที่ใช้ในการทดลอง

ขอขอบคุณทุกท่านที่ทำงานในห้อง Biomedical Signal and Image Processing Laboratory ที่ให้คำปรึกษาและคำแนะนำที่มีประโยชน์ต่อการทำโครงงานเป็นอย่างยิ่ง

ขอขอบพระคุณ คุณพ่อ คุณแม่ และครอบครัวที่อบรมเลี้ยงดู ให้กำลังใจและสนับสนุนในการศึกษามาโดยตลอด

ผู้จัดทำ

นายวันชัย ลีเลิศ

นายวิทูร ศรีวิชัย

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

ในปัจจุบันความเจริญก้าวหน้าของเทคโนโลยี ได้มีการพัฒนาให้ก้าวหน้าไปอย่างรวดเร็ว ชีวิตประจำวันส่วนใหญ่ได้สัมผัสกับระบบดิจิตอลไม่ว่าโดยตรงหรือทางอ้อม ระบบดิจิตอลที่เห็นได้ชัดเจนอย่างง่ายคือ คอมพิวเตอร์ที่เราใช้งานกันอยู่ทุกวัน และการสื่อสารแบบไร้สายสมัยใหม่ไม่ว่าเป็นโทรศัพท์หรืออินเตอร์เน็ตมักจะส่งและรับสัญญาณแบบดิจิตอล โดยที่ระบบดิจิตอลในไอซีจะเป็นตัวควบคุมอุปกรณ์อิเล็กทรอนิกส์และไฟฟ้าต่างๆ ดังที่เรามักจะคุ้นกับคำว่าระบบตัวเลขหรือดิจิตอล

ระบบภาพดิจิตอลได้ถูกนำมาใช้อย่างกว้างขวางเป็นประโยชน์อย่างมากในปัจจุบัน จึงเกิดการพัฒนาระบบภาพดิจิตอลอย่างรวดเร็ว Image Sensor เป็นส่วนสำคัญอย่างมากในระบบการเกิดภาพดิจิตอลที่ได้นำมาใช้อย่างกว้างขวางซึ่ง Image Sensor ที่ใช้กันอยู่ปัจจุบันมีอยู่ 2 ชนิดคือ CCD กับ CMOS ซึ่งในการประยุกต์ใช้งาน CMOS Image Sensor จะทำได้ง่ายกว่าและประหยัดการสิ้นเปลืองพลังงานน้อยกว่าและประกอบกับราคาถูกกว่า จึงเป็นที่น่าสนใจในการประยุกต์มาใช้งานร่วมกับ FPGA (Fields Programmable Gate Array) ซึ่งเป็นคอนโทรลเลอร์ที่ใช้ต้นทุนในการออกแบบต่ำ ลดจำนวนอุปกรณ์ที่ใช้ลงได้มากและสามารถออกแบบควบคุมระบบดิจิตอลได้อย่างมีประสิทธิภาพ โดยการใช้ภาษา VHDL เขียนโปรแกรมอธิบายการทำงาน ซึ่งเป็นภาษาที่สามารถบรรยายวงจรดิจิตอลได้หลายระดับ (Level of abstraction) ตั้งแต่ระดับพฤติกรรมการทำงานของวงจร (Behavioral level) จนถึงระดับลอจิกเกต (Gate level) ซึ่งในแต่ละระดับก็จะมีรายละเอียดที่แตกต่างกันไป

1.1 วัตถุประสงค์ของโครงงาน

1. สามารถเรียนรู้หลักการทำงานของ FPGA เพื่อนำไปใช้งาน
2. สามารถประยุกต์ใช้งาน FPGA ร่วมกับ CMOS Image Sensor
3. สามารถศึกษาและเข้าใจระบบการทำงานของ CMOS Image Sensor
4. สามารถควบคุมการออกแบบ FPGA โดยใช้ภาษา VHDL ได้
5. สามารถนำข้อมูลภาพจาก CMOS Image Sensor นำมาจัดเก็บในคอมพิวเตอร์เพื่อใช้ในการประยุกต์งานทางด้านประมวลผลภาพ (Image Processing)

1.2 ขอบเขตการดำเนินการ

1. ศึกษาหลักการทำงานของ CMOS Image Sensor เพื่อที่ประยุกต์ใช้งาน
2. ศึกษาหลักการทำงานของ FPGA เพื่อประยุกต์ใช้งานร่วมกับ CMOS Image Sensor
3. ศึกษาและใช้ภาษา VHDL ในการออกแบบบรรยายเชิงพฤติกรรมใช้ร่วมกับ FPGA เพื่อควบคุม CMOS Image Sensor เพื่อดึงสัญญาณภาพออกมาและอินเตอร์เฟสกับแรม
4. ศึกษาและใช้ C++ Builder ในการประมวลผลสัญญาณภาพดิจิตอล ให้ออกเป็นรูปภาพทางจอคอมพิวเตอร์
5. ศึกษาและใช้งาน USB Module

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1. เข้าใจถึงวิธีการออกแบบวงจรดิจิตอลเบื้องต้น
2. เข้าใจในหลักการทำงานและการประยุกต์ใช้งาน CMOS Image Sensor กับการประมวลผลสัญญาณภาพ
3. เข้าใจและสามารถเขียนโปรแกรมออกแบบ FPGA โดยภาษา VHDL ที่สูงขึ้น
4. เข้าใจและสามารถเขียนโปรแกรม C++ Builder เพื่อนำสัญญาณภาพดิจิตอลประมวลผลแสดงบนจอคอมพิวเตอร์
5. เข้าใจและสามารถใช้งาน USB Module

บทที่2
โครงสร้างหลักการทำงานของ FPGA และภาษาที่ใช้ออกแบบ

ในช่วงก่อนทศวรรษ 1970 อุตสาหกรรมเซมิคอนดัคเตอร์ได้ถูกปลุกให้ตื่นตัวขึ้นหลังจากที่มีการประดิษฐ์วงจรรวมหรือไอซี ตัวแรกสำเร็จ ในยุคแรกนั้นไอซีขนาดเล็กหรือ SSI (Small-Scale Integration) ประกอบไปด้วยเกทดิจิตอลจำนวนไม่มาก นัก(ประมาณ 1 ถึง 10 ตัว) ต่อมาได้มีการเพิ่มปริมาณของเกทดิจิตอลและฟังก์ชันทางลอจิกให้มากขึ้น จนเป็น MSI (Medium - Scale Integration) การพัฒนาไอซีเป็นไปอย่างต่อเนื่องจนมาถึงยุคของ LSI (Large-Scale Integration) ซึ่งเป็นยุคที่มีการ สร้างไมโครโปรเซสเซอร์ตัวแรกขึ้นและในปัจจุบันเป็นยุคของ VLSI (Very Large-Scale Integration) ซึ่งเทคโนโลยีในการสร้างไอซีรุดหน้า จนสามารถสร้างไมโครโปรเซสเซอร์ขนาด 64 บิต ที่มีหน่วยความจำแคชกับหน่วยคำนวณทางคณิตศาสตร์ ของโฟล์ทติงพอยน์ (Floating-Point Arithmetic Units) รวมอยู่ภายในตัวมันและเนื่องจากการปรับปรุงเทคโนโลยีของกระบวนการสร้างซีมอส ที่มีมาอย่างต่อเนื่องทำให้ขนาดของทรานซิสเตอร์ที่บรรจุอยู่ภายในไอซีมีขนาดเล็กลงเรื่อยๆ จนบางคนโดยเฉพาะในญี่ปุ่นใช้คำว่า ULSI (Ultralarge Scale Integration) เพื่อใช้เรียกระดับของไอซีในปัจจุบัน แต่คนส่วนมากยังมักนิยมเรียกเพียงแค่ VLSI

จากการปรากฏตัวของ VLSI ในช่วงทศวรรษ 1980 ทำให้วิศวกรเริ่มมีการออกแบบไอซีตามความต้องการของลูกค้า ซึ่งใช้ในระบบที่เจาะจงนอกเหนือจากการใช้ไอซีมาตรฐานเพียงอย่างเดียว โดยไอซีเหล่านี้มีชื่อเรียกว่า ASIC :Application-Specific Integrated Circuit (ออกเสียงว่า เอ-ซิก) ซึ่งตัวอย่างของ ASIC ได้แก่ชิพไอซีที่ใช้สำหรับตุ๊กตาของเล่นพูดได้ ดาวเทียม และชิพที่ภายในบรรจุด้วยไมโครโปรเซสเซอร์กับอุปกรณ์ทางลอจิกอื่นๆ

2.1 Field Programmable Gate Array (FPGA)

FPGA นับว่าเป็นอุปกรณ์ตัวใหม่ในตระกูลของ ASIC ซึ่งมีการเจริญเติบโตอย่างรวดเร็วและมีบทบาทที่สำคัญในการเข้ามาแทนที่ระบบอิเล็กทรอนิกส์ที่ใช้ TTL โครงสร้างภายในของ FPGA ประกอบไปด้วยอะเรย์ (Array) ของลอจิกเกทต่างๆมากมาย ซึ่งในปัจจุบันความจุเกทภายในตัวชิพ FPGA ได้เพิ่มขึ้น จากระดับไม่กี่พันตัวจนถึงระดับล้านตัวซึ่งสามารถรองรับวงจรดิจิตอลที่มีความสลับซับซ้อนได้เป็นอย่างดี นอกจากนี้ในด้านการออกแบบพัฒนาและทดสอบก็ทำได้ง่ายซึ่งในปัจจุบัน การออกแบบวงจรโดยใช้ FPGA กำลังเป็นที่นิยมและมีแนวโน้มที่จะนำมาใช้งานมากขึ้นเรื่อย

ในปัจจุบันมี FPGA อยู่ 4 ชนิดที่วางขายอยู่ในท้องตลาดได้แก่ Symmetrical Array, Row-Based, Hierarchical PLD และ Sea-of-Gates ซึ่งแต่ละชนิดก็มีลักษณะการเชื่อมต่อภายในและการโปรแกรมที่แตกต่างกันไป

2.1.1 การแบ่งประเภทของ FPGA ตามเทคโนโลยีที่ใช้ในการโปรแกรม

ซึ่งมีอยู่ 2 แบบคือ การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพของตัวชิพ และการโปรแกรมโดยการใช้หน่วยความจำ

2.1.1.1 การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพ

- Fuse เป็นวิธีการโปรแกรมที่สามารถทำได้เพียงครั้งเดียว ซึ่งหลังจากที่โปรแกรมแล้วจุดเชื่อมต่อจะขาดจากกัน

- Anti Fuse เป็นวิธีการโปรแกรมที่คล้ายกับแบบ Fuse แต่ต่างกันที่หลังจากทำการโปรแกรม แล้วจุดเชื่อมต่อจะเชื่อมถึงกัน

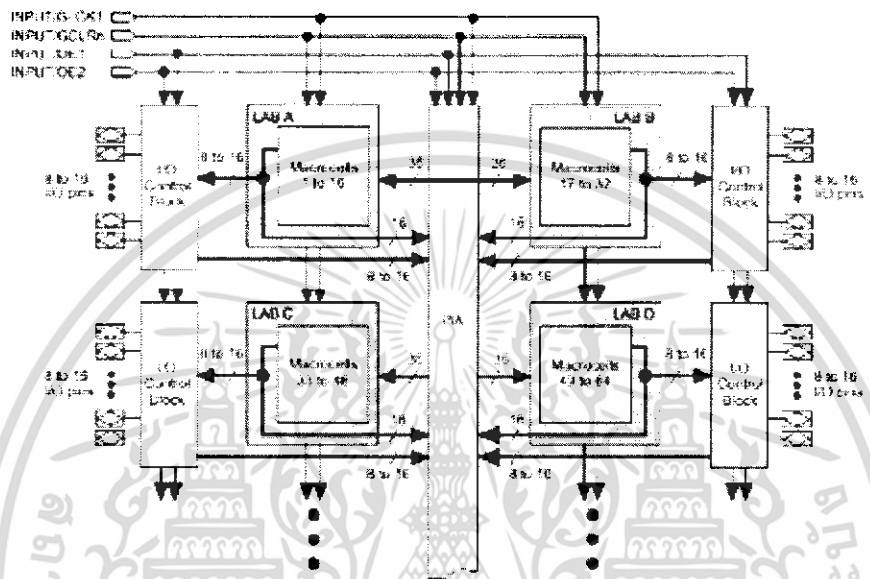
2.1.1.2 การโปรแกรมโดยใช้หน่วยความจำ

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

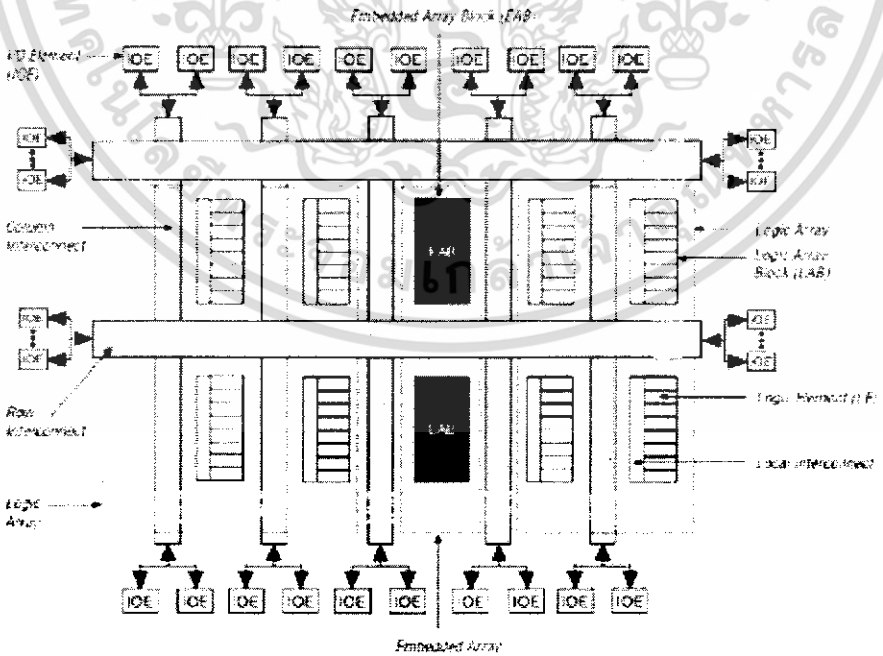
- SRAM Based FPGA แบบนี้จะใช้เทคโนโลยีในการโปรแกรมเหมือนกับ SRAM (Static-RAM) ทำให้สามารถโปรแกรมซ้ำได้โดยไม่จำกัดจำนวนครั้ง นอกจากนี้ยังมีความจุของเกทในระดับปานกลางถึงสูงมาก (ประมาณ 10,000 - 1,000,000 เกท) ซึ่งข้อดีของ SRAM Based FPGA คือใช้เวลาในการโปรแกรมน้อย (ระดับ nsec) การโปรแกรมทำได้ง่ายเทียบได้กับการเขียน SRAM ทั่วไป และเหมาะสำหรับการออกแบบวงจรที่มีความสลับซับซ้อน ส่วนข้อเสียคือไม่สามารถเก็บโปรแกรมในภาวะที่ไม่มีไฟเลี้ยงได้ดังนั้น FPGA ชนิดนี้จึงมักใช้ควบคู่กับ ROM เพื่อเก็บโปรแกรมและทำการโหลดโปรแกรมลงในตัวชิพในขณะที่เริ่มต้นใช้งาน

2.1.2 โครงสร้างภายในของ FPGA

ลักษณะโครงสร้างภายในของ FPGA จะเป็นอะเรย์ของบล็อกลอจิกที่สามารถทำการโปรแกรมดังรูปที่ 2.1 และ 2.2



รูปที่ 2.1 โครงสร้างภายในของ FPGA ตระกูล MAX7000S



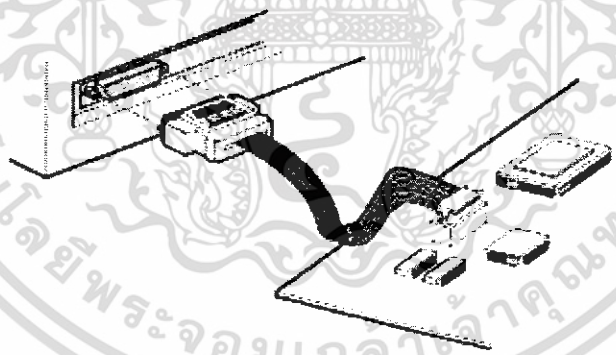
รูปที่ 2.2 โครงสร้างภายในของ FPGA ตระกูล FLEX10K

2.1.3 ปัจจัยที่ทำให้การออกแบบ FPGA ทำได้ง่ายและสะดวกรวดเร็ว

- ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพ เพียงแต่มีความรู้เกี่ยว กับขั้นตอนการออกแบบลอจิกก็เพียงพอแล้ว ต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายใน รวมถึงภาษาแอสเซมบลีของไมโครโปรเซสเซอร์ตัวนั้นด้วย

- มีการออกแบบ โดยใช้ภาษาในการอธิบายการทำงานของวงจร หรือ HDL (Hardware Description Language) เป็นเครื่องมือในการออกแบบ ซึ่งเป็นวิธีการที่มีความยืดหยุ่นสูง ทำได้รวดเร็ว และไม่จำเป็นต้องทราบถึงลักษณะของวงจรที่ต้องการว่าจะเชื่อมต่อกันอย่างไร เพียงแต่กำหนดลักษณะการทำงานให้มัน จากนั้นตัวซอฟต์แวร์จะทำ Synthesis and Optimize ให้ทั้งหมด นอกจากนี้ภาษาที่ใช้ยังเป็นมาตรฐาน เดียวกันสามารถใช้ได้กับชิพทุกตัวและทุกบริษัท

- การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นาน เพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของ คอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพขณะที่อยู่ในระบบได้โดยไม่จำเป็น ต้องถอดมาโปรแกรมข้างนอก ดังรูปที่ 2.3 และที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสีย ค่าใช้จ่ายเพิ่มแต่อย่างใด



รูปที่ 2.3 การโปรแกรมลงในชิพ

2.1.4 การออกแบบโดยใช้ภาษาอธิบายพฤติกรรมของฮาร์ดแวร์

ในการออกแบบวงจรดิจิตอลนั้นสามารถทำได้โดยการวาดวงจร (Schematic) หรือใช้ภาษาอธิบายพฤติกรรม (Hardware Description Language) ของฮาร์ดแวร์ ในกรณีของการออกแบบวงจรด้วย ASIC ชนิด Full Custom ผู้ออกแบบจะต้องเขียนวงจรด้วย Schematic จากนั้นจะนำวงจรที่ออกแบบไว้ไปทำการจำลองการทำงาน (Simulate) ซึ่งหากผลออกมาเป็นที่พอใจก็จะต้อง Layout

เป็นชั้นสาร และในการออกแบบ ASIC ชนิดนี้ผู้ออกแบบจำเป็นจะต้องทราบถึงเทคโนโลยีที่ใช้ในการสร้างด้วย หลังจากได้ Layout ที่สมบูรณ์แล้ว จึงจะส่งไปเข้ากระบวนการสร้างไอซีหรือ Fabrication เพื่อสร้างเป็นชิพไอซีออกมา แต่ในการออกแบบวงจรด้วย FPGA โดยการใช้ Schematic หรือใช้ภาษาอธิบายการทำงานของวงจรจะทำได้สะดวกกว่า เนื่องจากวิธีการนี้ผู้ออกแบบไม่จำเป็นต้องคำนึงถึงเทคโนโลยีที่จะใช้สร้างไอซีและที่สำคัญ การออกแบบโดยวิธีนี้สามารถแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่า เพราะไม่ต้องวาดวงจรใหม่นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยีสำหรับภาษาที่ใช้ สำหรับอธิบายพฤติกรรมของฮาร์ดแวร์ที่ใช้กันก็มี VHDL, AHDL และ Verilog เป็นต้น ส่วนรายละเอียดของขั้นตอนในการออกแบบสามารถอธิบายได้ดังนี้

2.1.4.1การสังเคราะห์วงจร(Logic-Synthesis)

ในขั้นตอนนี้จะใช้ซอฟต์แวร์ในการสังเคราะห์วงจร (Synthesis Tools) ทำการสังเคราะห์พฤติกรรมของวงจรที่ได้จากการออกแบบด้วย Schematic หรือ VHDL ซึ่งต้องทำการตรวจสอบด้วยว่าซอฟต์แวร์นั้นสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการหรือไม่ ตัวอย่างเช่น FPGA ของบริษัท XILINX และบริษัท ALTERA จะมีซอฟต์แวร์หลายตัวที่สามารถใช้ได้ เช่น Max Plus II ในขั้นตอนนี้ ซอฟต์แวร์สังเคราะห์วงจรจะทำการแปลงโค้ด VHDL และทำการ Optimize เพื่อให้ได้วงจรตามเทคโนโลยีที่เลือกใช้ในการสังเคราะห์วงจรนั้นวงจรระดับเกท (Gate Level) จะไม่เหมาะสมกับโครงสร้างที่มีอยู่ในอุปกรณ์ FPGA ดังนั้นในการ Optimize ซอฟต์แวร์สังเคราะห์วงจรจะต้องทำการ Optimize ให้ได้เป็นวงจรที่ประกอบด้วยกลุ่มของลอจิกที่เหมาะสมกับอุปกรณ์ FPGA นั้นๆจึงทำให้ผลที่ได้มีประสิทธิภาพ และในขั้นตอนการสังเคราะห์วงจรนี้ ผู้ออกแบบสามารถกำหนดข้อบังคับสำหรับโมเดลแต่ละตัวได้เช่น ข้อบังคับในเรื่องเวลา (Timing Constraints) หรือข้อบังคับในเรื่องของพื้นที่ (Area) หรือกำหนดชนิดและตำแหน่งของ I/O ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอน Optimize เพื่อให้วงจรที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการ Optimize คือการเทียบ (Mapping) โมเดลให้เข้ากับเทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างและสถาปัตยกรรมภายในอุปกรณ์ FPGA เมื่อทำการสังเคราะห์วงจรเสร็จแล้ว ซอฟแวร์การสังเคราะห์วงจรก็จะมีการรายงานผลว่า โมเดลที่ออกแบบไปนั้นเป็นอย่างไร เช่น มีค่าความหน่วง (Delay) เท่าใด ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้าง เมื่อมาถึงขั้นตอนนี้ ผู้ออกแบบก็จะทราบว่าโมเดลเป็นไปตามข้อบังคับหรือไม่ ถ้าไม่ก็สังเคราะห์ใหม่จนกว่าจะเป็นไปตามที่กำหนด

2.1.4.2การแบ่งวงจร(Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์ เป็นส่วนย่อยๆ สำหรับลงใน CLBs, IOBs หรือองค์ประกอบอื่นๆ ภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่จะแยกออกจากกัน มีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้ เพื่อลดความหนาแน่นในตอนทำการเชื่อมต่อสัญญาณ (routing) ในขั้นตอนนี้จะใช้ซอฟต์แวร์ทำโดยซอฟต์แวร์จะเทียบส่วนประกอบของวงจรเช่น เกท (gate), ฟลิป-ฟลอบ (flip-flop) ลงในทรัพยากรต่างๆ ที่มีอยู่ในอุปกรณ์ FPGA หลังจากทำขั้นตอนนี้เสร็จแล้วผู้ออกแบบสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGA ไปเท่าไร ส่วนข้อมูลทางเวลานั้น ผู้ออกแบบจะทราบเฉพาะความหน่วงภายในแต่ละส่วนเท่านั้น หรือที่เรียกว่าความหน่วงลอจิก (logic delay) ส่วนซอฟต์แวร์จะรวมเอาซอฟต์แวร์ย่อยอื่นๆ อีก เพื่อให้การทำ PPR (Partitioning Placement & Routing) เป็นไปอย่างต่อเนื่อง

2.1.4.3การวางอุปกรณ์(Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Partitioning) มาแล้วว่าควรจะอยู่ ณ ตำแหน่งไหนในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ดีที่สุด เช่นวงจรส่วนไหนควรอยู่ใกล้กัน เพื่อจะได้ค้นหาเส้นทางได้ (route) ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมีความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือ Router ทำการค้นหาเส้นทางสัญญาณได้ไม่หมด การวางอุปกรณ์ที่ดีควรวางส่วนต่างๆให้อยู่ใกล้กัน โดยเฉพาะส่วนที่มีการเชื่อมต่อสัญญาณด้วยกันนอกจากนั้นการกำหนดตำแหน่งขา I/O (I/O pin) ตามตำแหน่งขา I/O ของ FPGA บนแผ่น PCB ก็จะมีผลโดยตรงเลยคือซอฟต์แวร์จะวาง I/O ลงในตำแหน่งที่ผู้ออกแบบกำหนดซึ่งบางครั้งตำแหน่งที่กำหนดไปไม่เหมาะสมดังนั้นการกำหนดขา I/O ควรกำหนดตำแหน่งให้เหมาะสมหรือไม่ก็ให้ซอฟต์แวร์จัดการเอง

2.1.4.4การเชื่อมต่อสัญญาณ(Routing)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA ขั้นตอนนี้จะทำต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดี ซอฟต์แวร์ก็จะทำการเชื่อมต่อสัญญาณได้ไม่หมด (เนื่องจากจำนวนทรัพยากรสำหรับเชื่อมต่อสัญญาณนั้นมีอยู่จำกัด) หรือเกิดความหน่วงเกินค่าที่กำหนดในข้อบังคับ ผู้ออกแบบสามารถทำขั้นตอนนี้ได้โดยใช้ซอฟท์แวร์หรือผู้ออกแบบจะทำการ เชื่อมต่อสัญญาณด้วยตนเองก็ได้ แต่ทางที่ดีควรใช้ซอฟต์แวร์ทำดีกว่า นอกจากนั้นการกำหนดข้อบังคับทางเวลาจะช่วยให้ผลที่ได้จากการเชื่อมต่อสัญญาณดีขึ้นได้

2.1.4.5ความหน่วงค้านเวลา(Delay)

ในการทำ FPGA นั้นความหน่วงที่เกิดขึ้นเป็นความหน่วงที่เกิดจากการวางตำแหน่ง (layout) ของอุปกรณ์ ซึ่งผู้ออกแบบไม่สามารถเข้าไปแก้ไขได้ แต่สามารถทำให้มีความหน่วงน้อยที่สุดได้ สำหรับความหน่วงที่เกิดขึ้นนั้นแยกได้เป็นสองประเภทคือ

- ความหน่วงลอจิก (Logic delay) เป็นความหน่วงภายในองค์ประกอบของอุปกรณ์ FPGA เอง
- ความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณ (Routing delay)

เป็นความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณระหว่างองค์ประกอบภายในอุปกรณ์ FPGA โดยปกติแล้ว ค่าความหน่วงลอจิกไม่ควรเกิน 50% ของค่าความหน่วงที่ยอมรับได้ เพราะความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณมักจะมีค่ามากกว่าค่าความหน่วงลอจิก ดังนั้นในการวางอุปกรณ์ และเชื่อมต่อสัญญาณ ผู้ออกแบบควรกำหนดข้อบังคับทางเวลา เพื่อให้ซอฟต์แวร์ได้ทำงานอย่างมีประสิทธิภาพเพิ่มขึ้น และเพื่อให้ได้ผลลัพธ์ที่ดีขึ้นค่าความหน่วงที่ได้หลังจากการวางอุปกรณ์ และเชื่อมต่อสัญญาณแล้วจะมีค่าความหน่วงที่ค่อนข้างแน่นอน ซึ่งผู้ออกแบบสามารถทราบได้ว่าโมเดลที่ออกแบบนั้นเป็นไปตามข้อกำหนดหรือไม่

2.1.4.6การจำลองการทำงานของวงจร(Simulation)

ในขั้นตอนนี้เป็นขั้นตอนที่สำคัญอีกขั้นตอนหนึ่ง เพราะเป็นขั้นตอนที่ผู้ออกแบบตรวจสอบฟังก์ชันการทำงานของโมเดลว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหนเพื่อจะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้จะมีซอฟต์แวร์ที่ใช้สำหรับทำการจำลองการทำงานของวงจรที่ใช้อยู่ เช่น Model Sim ของบริษัท Model Technology หรือ Max Plus II ของบริษัท Altera ในการจำลองการทำงานของวงจร ควรทำทุกครั้งหลังจากที่มีการทำแต่ละขั้นตอนหลักเสร็จแล้ว เพื่อจะได้ทราบว่าข้อผิดพลาดของโมเดลเกิดขึ้นตอนไหน จะได้แก้ไขข้อผิดพลาดตรงขั้นตอนนั้นๆ ได้เลย ไม่ต้องมาคอยตรวจหาขั้นตอนที่ทำให้เกิดข้อผิดพลาด นั่นคือการทำการจำลองการทำงานของวงจร ต้องทำทั้งหลังการเขียนโค้ด, การสังเคราะห์วงจร และการทำ PPR การจำลองการทำงานของวงจรหลังจากที่เขียนโค้ดเสร็จแล้วนั้น ผู้ออกแบบสามารถทราบได้แค่ โมเดลทำงานถูกต้องหรือไม่เท่านั้น (functional test) ยังไม่สามารถตรวจสอบการทำงานในเชิงเวลาได้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่สังเคราะห์เป็นวงจรแล้ว เพื่อจรวจสอบว่าฟังก์ชันการทำงานยังคงถูกต้องหรือไม่ และค่าความหน่วงที่เกิดขึ้นเป็นไปตามข้อบังคับหรือไม่ มีข้อผิดพลาดเกิดขึ้นหรือไม่ถ้ามีจะแก้ไขให้ถูกต้อง

ในการจำลองการทำงานของวงจรหลังจากที่ทำการวางอุปกรณ์ การเชื่อมต่อสัญญาณ (post layout simulation) แล้วก็มีความสำคัญเช่นกันเพราะผลที่ได้จากการจำลองการทำงานของวงจรในตอนนี้ จะเป็นผลลัพธ์ของโมเดลเลย ซึ่งผู้ออกแบบนอกจากจะตรวจสอบฟังก์ชันการทำงานแล้วยังต้องตรวจสอบคุณสมบัติอื่นๆ เช่น ความหน่วงที่ได้จากการทำ PPR ในรูปแบบค่าความหน่วงมาตรฐาน (Standard Delay Format : SDF) ว่าตรงตามที่กำหนดหรือไม่ หรือตรวจสอบว่าวงจรรวมสามารถใช้งานที่ความถี่สูงสุดเท่าไรนั่นเอง ในการจำลองการทำงานของวงจรควรใช้ซอฟต์แวร์ตัวเดียวกันตลอดเพื่อจะได้เปรียบเทียบผลที่ได้จากขั้นตอนต่างๆ

2.1.4.7การโปรแกรมอุปกรณ์-FPGA(Configuration)

หลังจากที่โมเดลผ่านขั้นตอนต่างๆ จนกระทั่งผ่านการทำ PPR (Partitioning, Placement & Routing) แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด (download) ลงในอุปกรณ์ FPGA ได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้เป็นข้อมูลวงจร (configuration data) ซึ่งอยู่ในรูปของบิตสตรีม (bit stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชันการทำงานตามโมเดลที่ผู้ออกแบบต้องการ ซึ่งในขั้นตอนนี้จะใช้วิธีที่แตกต่างกันออกไป สำหรับอุปกรณ์ FPGA ของแต่ละบริษัทผู้ผลิต คือในกรณีที่เป็นอุปกรณ์ FPGA ชนิดที่ต้องโปรแกรมโดยวิธี SRAM นั้น ในการใช้งาน ผู้ออกแบบจะต้องเก็บข้อมูลวงจรไว้ในหน่วยความจำประเภท EPROM หรือ serial PROM ด้วยเพื่อจะใช้งานสะดวกขึ้น คือในการใช้งานโมเดลครั้งต่อไปไม่ต้องดาวน์โหลดข้อมูลวงจรจากเครื่องคอมพิวเตอร์อีก เพราะมีข้อมูลวงจรเก็บอยู่ในหน่วยความจำอยู่แล้ว แต่กรณีที่อุปกรณ์ FPGA เป็นชนิดที่โปรแกรมโดยใช้วิธี EPROM หรือ Anti fuse ก็ไม่จำเป็นต้องมีหน่วยความจำสำหรับเก็บข้อมูลวงจร เพราะว่าอุปกรณ์ FPGA ชนิดนี้เมื่อดาวน์โหลดข้อมูลวงจรลงไป ข้อมูลที่ดาวน์โหลดลงไปก็ยังคงอยู่ในอุปกรณ์ FPGA และครั้งต่อไปก็ใช้งานโมเดลที่ออกแบบไว้ได้เลย

2.1.5 เครื่องมือสำหรับการออกแบบ FPGA

จะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้นทำได้สะดวกกว่า ASIC มากเพราะใช้เวลาน้อยกว่ามากด้วย ส่วนสำคัญที่ใช้ในการทำ FPGA คือ ซอฟต์แวร์ที่ใช้ตั้งแต่เขียนโค้ดอธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลดลงใน อุปกรณ์ FPGA ซึ่งซอฟต์แวร์ที่ใช้ ต้องเป็นซอฟต์แวร์ที่ทำงานต่อเนื่องกันได้ สำหรับซอฟต์แวร์ที่ใช้ทำการจำลองการทำงานของวงจรนั้น ต้องสามารถใช้งานต่อเนื่องกับซอฟต์แวร์ที่ใช้ทั้งระบบ เพราะโมเดลที่ได้จากการทำขั้นตอนต่างๆ ด้วยซอฟต์แวร์ ต่างๆ ต้องเอามาจำลองการทำงานได้ และในการจำลองการทำงานของวงจรควรใช้ซอฟต์แวร์ตัวเดียวกัน

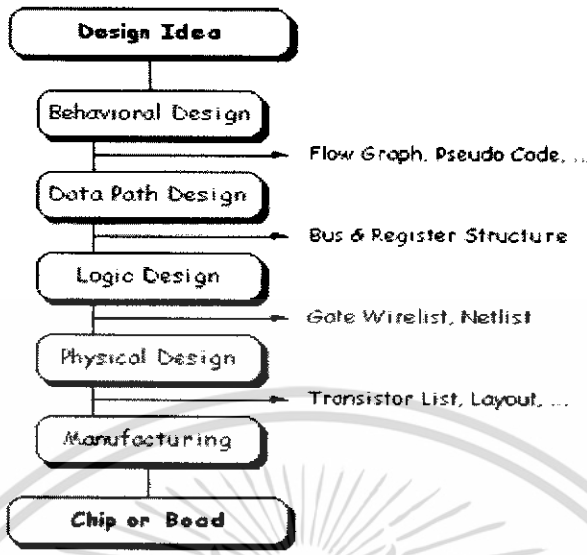
ตลอดทั้งระบบ เพื่อจะได้เปรียบเทียบผลได้ง่าย ในอดีตซอฟต์แวร์ส่วนใหญ่จะใช้งานอยู่บนคอมพิวเตอร์สมรรถนะสูงอย่างเวิร์คสเตชั่น (Workstation) ในปัจจุบันมีการพัฒนาซอฟต์แวร์ที่ใช้บนพีซี (PC) มากขึ้นซึ่งสามารถลดค่าใช้จ่ายในด้านอุปกรณ์คอมพิวเตอร์

2.2 ภาษา VHDL

ความซับซ้อนและขนาดของระบบดิจิตอลในปัจจุบันได้เพิ่มมากขึ้นทุกขณะ ส่งผลให้มีการนำคอมพิวเตอร์เพื่อช่วยในการออกแบบ หรือ CAD มาใช้ในขบวนการออกแบบฮาร์ดแวร์เพิ่มขึ้นเช่นกัน อีกทั้งอุปกรณ์และวิธีการออกแบบใหม่ๆ ก็ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้กับนักออกแบบมากขึ้นด้วยสำหรับภาษาบรรยายอุปกรณ์ฮาร์ดแวร์(HDL :Hardware Description Language) ก็เป็นเครื่องมืออย่างหนึ่งที่ได้รับการพัฒนามาอย่างต่อเนื่อง เพื่อช่วยให้การปรับปรุงขบวนการออกแบบระบบดิจิตอลเป็นไปอย่างมีประสิทธิภาพ

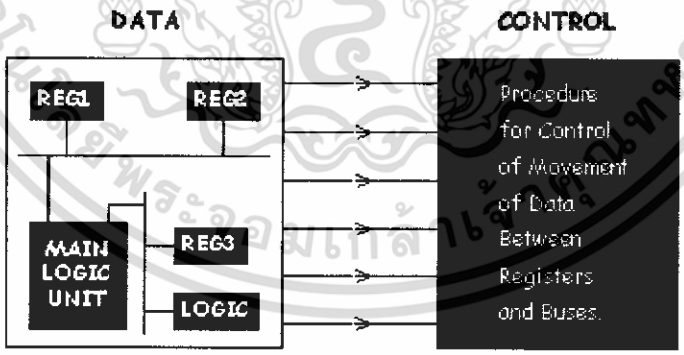
2.2.1 การออกแบบระบบดิจิตอล

ในการออกแบบระบบดิจิตอล เริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวร์ ที่ใช้งานได้จะต้องผ่านขั้นตอนต่างๆมากมาย และในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์ในแต่ละขั้นก่อนเข้าสู่กระบวนการออกแบบในขั้นต่อไป รูปที่ 2.4 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบดิจิตอลทั่วไป ขั้นแรกผู้ออกแบบจะกำหนดแนวความคิดในการออกแบบแล้วทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ซึ่งภายในขั้นตอนนี้ผู้ออกแบบจำเป็นต้องสร้างรูปแบบระบบในเชิงพฤติกรรมขึ้นมาตรวจสอบ ซึ่งอาจจะเป็นผังงานแสดงแบบหรือรหัสคำสั่งเทียม (Pseudo code) ก็ได้



รูปที่ 2.4 แสดงขั้นตอนการออกแบบระบบดิจิตอล

ขั้นตอนต่อไปเป็นการออกแบบระบบเส้นทางของข้อมูล โดยที่ผู้ที่ออกแบบจะกำหนดส่วนประกอบของรีจิสเตอร์และวงจรลอจิก ที่จำเป็นทั้งหมดเพื่อนำมาประกอบเป็นระบบที่สมบูรณ์ โดยแต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่ง หรือสองทิศทาง (Unidirectional or Bidirectional Bus) ส่วนกระบวนการในการควบคุมการเคลื่อนย้ายข้อมูล ระหว่างรีจิสเตอร์และวงจรลอจิกจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ดังรูปที่ 2.5



รูปที่ 2.5 การออกแบบระบบเส้นทางของข้อมูล

ขั้นตอนถัดมาเป็นการออกแบบวงจรลอจิก ซึ่งจะเกี่ยวข้องกับการนำเกทดิจิตอลพื้นฐานและฟลิปฟลอป (flip-flop) มาประกอบเป็นอุปกรณ์ย่อยต่างๆ เช่น รีจิสเตอร์เก็บข้อมูลบัสวงจรลอจิก และส่วนควบคุมฮาร์ดแวร์ ซึ่งผลลัพธ์ที่ได้ในขั้นตอนนี้จะเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิปฟลอปนั่นเอง การออกแบบในขั้นตอนถัดไปเป็นการเปลี่ยนเครือข่ายการโยง

ใยที่ได้จากขั้นตอนที่แล้วให้เป็นลำดับของทรานซิสเตอร์ (Transistor List) และ Layout ซึ่งขั้นตอนนี้จะเกี่ยวข้องโดยตรงกับการจัดวางทรานซิสเตอร์หรือไลบรารีเซลล์เพื่อ แทนเกทและฟลิปฟลอปต่างๆและในขั้นตอนสุดท้ายจะเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด

2.2.2 ประวัติความเป็นมาของภาษา VHDL

VHDL ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC : Very High Speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูง (High Level Language) ที่ใช้สำหรับการออกแบบฮาร์ดแวร์ในระบบดิจิตอล ตัวของภาษาสามารถบรรยายพฤติกรรมการทำงานในรูปของลำดับชั้น (Hierarchy) และสามารถเขียนได้หลายรูปแบบ ด้วยเหตุผลนี้จึงทำให้ภาษา VHDL เป็นเครื่องมือที่ใช้ออกแบบตั้งแต่ขั้นตอนบนสุด คือ แนวความคิดที่จะแก้ปัญหา ลงไปทีละขั้นจนถึงขั้นตอนของการสร้างวงจรจริง และตัวภาษาก็เปิดโอกาสให้วิศวกรได้พัฒนาและจำลองการทำงานของรูปแบบฟังก์ชันการทำงานของวงจรอย่างสังเขป โดยยังไม่ต้องคำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง นอกจากนั้น VHDL ยังเป็นภาษาที่สนับสนุนลักษณะต่างๆ ของระบบดิจิตอลที่มีความซับซ้อนได้ทั้งหมด ดังนั้น VHDL จึงเป็นภาษาที่น่าสนใจในการศึกษาและนำไปใช้งานเป็นอย่างยิ่ง วิวัฒนาการของภาษา VHDL เริ่มต้นประมาณปี ค.ศ. 1981 เมื่อกระทรวงกลาโหมสหรัฐอเมริกา หรือ DoD (Department of Defense) ได้พยายามปรับปรุงอุปกรณ์อิเล็กทรอนิกส์และคอมพิวเตอร์ที่ใช้ในกิจการทางทหาร ให้มีความทันสมัยมากขึ้น ประกอบกับเทคโนโลยีทางด้านไมโครอิเล็กทรอนิกส์มีการพัฒนาไปอย่างรวดเร็วดังจะเห็นได้จากการนำวงจรดิจิตอลหลายๆ วงจรมาทำการผลิตอยู่บนแผ่นซิลิกอนที่มีพื้นที่เพียง 1 - 2 ตารางเซนติเมตรเท่านั้น ซึ่งเป็นผลให้ประสิทธิภาพในการทำงานของวงจรสูงขึ้นตลอดจนความน่าเชื่อถือ ในการทำงานและความคงทนต่อสภาพแวดล้อมสูง แต่เนื่องจากในขณะนั้นขั้นตอนของการออกแบบการผลิต และการตรวจสอบวงจรต้นแบบ เป็นขบวนการที่ต้องใช้วิศวกร และเวลาในดำเนินการมาก ฉะนั้นทาง DoD จึงจัดตั้งโครงการขึ้นมาเพื่อศึกษาวิธีการที่ช่วยในการพัฒนาวงจรอิเล็กทรอนิกส์ โดยเฉพาะอย่างยิ่งวงจรระบบดิจิตอล ให้สามารถนำไปผลิตได้เร็วขึ้น ซึ่งโครงการดังกล่าวมีชื่อว่า "Very High Speed Integrated Circuits" หรือ VHSIC โดยในระยะแรกนั้นโครงการนี้ถือเป็นความลับทางด้านความมั่นคงของประเทศ และอยู่ภายใต้ความควบคุมดูแลของ United States International Traffic and Arms Regulations (ITAR) สำหรับมาตรฐานของภาษาที่ใช้บรรยายพฤติกรรมวงจรหรือฮาร์ดแวร์ของระบบ สำหรับโครงการ VHSIC ที่ DoDได้ให้ไว้สามารถสรุปได้ดังนี้

-ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอล และมีคุณสมบัติที่สามารถเข้าใจได้ทั้งมนุษย์และเครื่องคอมพิวเตอร์โดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก

-สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้

- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร

ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง เช่นเดียวกับภาษาปาสคาล หรือภาษาซี ซึ่งในทางวิศวกรรม ภาษาที่ใช้ในการออกแบบฮาร์ดแวร์นี้เรียกว่า "Hardware Description Language"หรือHDL

ในตอนเริ่มแรกนั้น DoD ได้มอบหมายให้บริษัทไอบีเอ็ม เท็กซัสอินสตูเมนท์ และอินเตอร์เมทริกซ์ เป็นผู้ศึกษาและพัฒนาโครงการ ซึ่งการดำเนินงานเป็นไปอย่างต่อเนื่อง จนกระทั่งในปี ค.ศ.1985 ทาง ITAR ได้ยกเลิกข้อจำกัดในการถ่ายทอดเทคโนโลยีทางทหารออกจากโครงการนี้ ดังนั้นภาษา VHDL จึงเริ่มเป็นที่รู้จักกันโดยทั่วไป และประมาณปี ค.ศ. 1987 IEEE ได้ทำการกำหนดมาตรฐานของภาษานี้เป็น IEEE 1076-1987 และมีชื่อเรียกว่า VHDL ซึ่งมาตรฐานนี้ได้รับการปรับปรุงจนเป็นมาตรฐาน IEEE 1076-1993 หรือ VHDL 1993 เนื่องจากในขณะนั้น DoD เป็นลูกค้ารายใหญ่ของอุตสาหกรรมอิเล็กทรอนิกส์และคอมพิวเตอร์ ดังนั้นจึงมีผู้รับโครงการต่างๆ จาก DoD ไปดำเนินการวิจัยและพัฒนาเป็นจำนวนมาก และเพื่อให้ทุกโครงการอยู่ในมาตรฐานเดียวกันหมด ดังนั้นทาง DoD จึงได้กำหนดว่าทุกๆ โครงการต้องเขียนอยู่ในรูปของภาษา VHDLเท่านั้น ซึ่งทำให้ DoD สามารถนำโครงการเหล่านี้ไปจำลองกับเครื่องคอมพิวเตอร์ได้หลายๆระบบ

2.2.3 องค์ประกอบพื้นฐานของ VHDL

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบของ VHDL จะประกอบไปด้วยส่วนกำหนดการเชื่อมต่อ (Interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (Architecture) โดยในการบรรยายการเชื่อมต่อจะขึ้น ต้นด้วยคำว่า ENTITY แล้วตามด้วยชื่อขององค์ประกอบจากนั้นตามด้วยคำว่า IS และถัดมาจะเป็นการบรรยายถึงพอร์ตการติดต่อ อินพุท - เอาท์พุท ขององค์ประกอบส่วนลักษณะภายนอกอื่นๆ เช่น เวลา อุณหภูมิก็สามารถรวมเข้าไปในส่วนนี้ได้เช่นกัน โดยที่ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้ บรรยายหน้าที่การทำงานขององค์ประกอบ โดยหน้าที่การทำงานนี้จะขึ้นอยู่กับสัญญาณอินพุท เอาท์พุทและพารามิเตอร์อื่นๆ ที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อดังรูปที่ 2.6 และสำหรับการบรรยายหน้าที่ขององค์ประกอบจะเริ่มต้นหลังจาก คำว่า BEGIN เป็นต้นไป

```
ENTITY component_name IS
    Input and output ports
    Physical and other parameters
END [component_name] ;

ARCHITECTURE identifier OF component_name IS
    [declartion]
BEGIN
    specification of the functionality of the component
    in terms of its input lines and as influenced
    by physical and other parameters
END [identifier];
```

รูปที่ 2.6 การกำหนดการเชื่อมต่อและสถาปัตยกรรม

2.2.3.1การกำหนดการเชื่อมต่อ

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ โดยในระดับนี้ต้องกำหนดพอร์ตสำหรับการติดต่อกับองค์ประกอบภายนอกอื่นๆ ดังตัวอย่างในรูปที่ 2.7 ซึ่งเป็นบล็อกไดอะแกรม และการบรรยายการเชื่อมต่อขององค์ประกอบสำหรับตัวจ่ายสัญญาณนาฬิกา ในบรรทัดแรกของการบรรยายการเชื่อมต่อเป็นการกำหนดชื่อขององค์ประกอบซึ่งกำหนดเป็น clock_ component ตามด้วยคำว่า PORT และชื่อของพอร์ตอยู่ภายในวงเล็บ ส่วน IN และ OUT เป็นการกำหนดโหนดของสัญญาณให้เป็นอินพุทหรือเอาท์พุท และ BIT เป็นการแสดงชนิดของข้อมูล


```
ENTITY clock_component IS
    PORT (en : IN BIT;ck : OUT BIT)
END clock_name;
```

รูปที่ 2.7 บล็อกไดอะแกรมและการบรรยายการเชื่อมต่อของ clock_ component

2.2.3.2การกำหนดรูปแบบการบรรยาย

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายภายในส่วนนี้ ซึ่งในการบรรยายสามารถกำหนดค่าของสัญญาณเอาท์พุทในเทอมของอินพุท หรือในรูปขององค์ประกอบอื่นๆ หรือทั้งสองอย่างรวมกันก็ได้ ดังตัวอย่างการบรรยายของ clock_component ในรูปที่ 2.8 ซึ่งเป็นการบรรยายในเชิงพฤติกรรมโดยมี en เป็นอินพุทและ ck เป็นเอาท์พุท PROCESS เป็นคำที่ใช้ในการเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรม และภายในโปรเซสกำหนดให้ periodic เป็นตัวแปรที่มีค่าเริ่มต้นเป็น "0" ถ้าสัญญาณ en มีค่าเป็น "1"' จะทำให้ตัวแปร periodic ถูกคอมพลีเมนท์ (complement) และส่งค่าให้กับ ck ซึ่งเป็นสัญญาณเอาท์พุท และสำหรับคำสั่ง WAIT จะเป็นการกำหนดให้สัญญาณมีคาบเวลาเท่ากับ 1 ไมโครวินาที

```
ARCHITECTURE behavioral OF clock_component IS

BEGIN
   PEOCESS
      VARIABLE periodic : BIT := '0';
   BEGIN
      IF en='1' THEN
        periodic := Not periodic;
      END IF;
       ck <= periodic;
       WAIT FOR 1 US;
  END PROCESS;
END behavioral;
```

รูปที่ 2.8 การบรรยายเชิงพฤติกรรมของ clock _ component

2.2.3.3หน่วยการออกแบบแพ็กเก็ต

ข้อมูลต่างๆ ตลอดจนโปรแกรมย่อย ที่เป็นประโยชน์ต่อการเขียนรูปแบบการบรรยายระบบดิจิตอล สามารถเก็บไว้ในส่วนของแพ็กเก็ต ซึ่งหน่วยการออกแบบต่างๆ เช่น หน่วยการออกแบบ Entity หน่วยการออกแบบสถาปัตยกรรมหรือหน่วยการออกแบบแพ็กเกจอื่นๆ สามารถเรียกข้อมูลเหล่านี้ไปใช้ได้ นอกจากนั้นสิ่งที่นิยมทำกันมากคือการนำรูปแบบมาตรฐานต่างๆ เช่น อุปกรณ์มาตรฐาน (เช่น ไอซีตระกูล 74XX เป็นต้น) มาเก็บไว้ในรูปของแพ็กเกจที่ทุกคนสามารถเข้าถึงได้ ตามปกติแล้วแพ็กเกจจะแบ่งออกเป็น 2 ส่วนคือ การประกาศแพ็กเกจ (Package declaration) และส่วนของบอดี้แพ็กเกจ (Package body) เนื่องจากแพ็กเกจถูกสร้างขึ้นเป็นส่วนแยกต่างหากออกจากรูปแบบที่กำลังเขียนอยู่ ฉะนั้นการที่นำแพ็กเกจไปใช้นั้นจะต้องมีการเชื่อมโยงหรืออ้างอิงเสียก่อน ซึ่งในภาษา VHDL สามารถกระทำได้ด้วยชุดคำสั่ง USE

-PACKAGE-DECLARATION

ส่วนที่มีความสำคัญที่สุดของแพ็กเกจ (ถ้ามองในแง่ของการนำไปใช้จากภายนอก) ได้แก่ ส่วนการประกาศแพ็กเกจ เนื่องจากเป็นส่วนที่ใช้กำหนดชื่อของสิ่งที่ประกาศอยู่ภายในแพ็กเกจ สำหรับนำไปใช้ภายนอกตัวของแพ็กเกจเอง ถ้ามีการประกาศสิ่งใดๆ ในส่วนของส่วนบอดี้แพ็กเกจ แต่ไม่ถูกประกาศในส่วนการประกาศแพ็กเกจจะทำให้ค่าและพฤติ กรรมไม่สามารถนำไปใช้งานในส่วนนอกได้ซึ่งเปรียบเทียบได้กับสิ่งที่ประกาศไว้ในส่วนของการประกาศ Entity คือ จุดเชื่อมต่อหรือพอร์ต ที่มีหน้าที่ติดต่อกับโลกภายนอก ฉะนั้นโดยทั่วไปแล้วแพ็กเกจสามารถสร้างขึ้นได้โดยไม่จำเป็นต้องมีส่วนบอดี้ และยังสามารถนำไปใช้งานจากรูปแบบภายนอกได้ เช่น ใช้สำหรับประกาศชนิด (Type) หรือสัญญาณ เช่นเดียวกับส่วนบอดี้แพ็กเกจที่ไม่จำเป็นต้องมีส่วนของการประกาศแพ็กเกจ แต่แพ็กเกจนั้นจะไม่สามารถนำไปใช้จากรูปแบบอื่นได้

```
PACKAGE package_name IS
    Package_declarative_part
END package_name;
```

รูปที่ 2.9 โครงสร้างทั่วไปของส่วนการประกาศแพ็กเกจ

-PACKAGE-BODY

โครงสร้างซึ่งประกอบด้วย ลำดับคำสั่งที่ใช้บรรยายฟังก์ชันการทำงานของโปรแกรมย่อยทั้งหลาย ซึ่งชื่อของโปรแกรมย่อยนั้นๆ ได้ถูกประกาศไปแล้วในส่วนของการประกาศแพ็กเกจจะถูกเก็บไว้ในส่วนของบอดี้แพ็กเกจ ทั้งนี้รวมถึงการกำหนดค่าคงที่ต่างๆ อันได้แก่ค่าคงที่ที่ถูกประกาศชื่อไว้ก่อนในส่วนของการประกาศแพ็กเกจ และถูกกำหนดค่าในส่วนของบอดี้แพ็กเกจ ฉะนั้นในส่วนของบอดี้แพ็กเกจจึงไม่จำเป็นต้องมี ถ้าในส่วนของการประกาศแพ็กเกจไม่มีการประกาศชื่อที่เป็นโปรแกรมย่อยหรือค่าคงที่ การเขียนบอดี้แพ็กเกจนั้นจะเป็นไปตามกฎเกณฑ์

```
PACKAGE BODY package_name IS
    declarative part
END package_name;
```

รูปที่ 2.10 โครงสร้างของบอดี้แพ็กเกจ

2.2.3.4หน่วยการออกแบบConfiguration

ดังที่ทราบกันแล้วว่าระบบดิจิตอลรูปแบบหนึ่งไม่ว่าจะเป็นอะไรก็ตาม จะสามารถมีหน่วยการออกแบบ Entity ได้ เพียงหนึ่งเดียวเท่านั้น ซึ่งในหน่วยการออกแบบ Entity หนึ่งหน่วยนี้อาจจะมีสถาปัตยกรรมที่เป็นหน่วยรองได้หลายหน่วย ดังนั้นจะต้องมีหน่วยการออกแบบ Configuration มาเพื่อกำหนดการใช้ Configuration ของการประกอบ Entity กับหน่วยการออกแบบสถาปัตยกรรมหน่วยใดๆ เข้าด้วยกัน

```
COMFIGURTRION identifier OF entiry_name IS
    Configuretion_declarative_part
END ;
```

รูปที่ 2.11 โครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ

2.2.3.5โปรแกรมย่อย

การใช้ฟังก์ชันและโพรซีเจอร์ใน VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาชั้นสูงทั่วๆไป ค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่นถ้าใช้ฟังก์ชันแทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรลอจิกจริงๆ ในขณะที่ถ้าใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูล หรือในการคำนวณค่าการหน่วงเวลาแล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์ รูปที่2.12 แสดงการใช้โพรซีเจอร์เพื่อเปลี่ยนข้อมูลชนิด 8 บิตเป็นค่าจำนวนเต็ม และรูปที่ 2.13 แสดงการใช้ฟังก์ชันโดยกำหนดให้ X เป็นตัวแปรชนิดบิตแทนการกระทำในสมการบูลีน

```
TYPE byte IS ARRAY (7 DOWNTO 0) OF BIT;
 ...
PROCEDURE byte_to_integer (ib : IN byte; oi : OUT INTEGER) IS
        VARIABLE result: INTEGER := 0;
BEGIN
        FOR i IN 0 TO 7 LOOP
                IF ib(i) = '1' THEN
                result  := result + 2**i;
                END IF;
        END LOOP;
        oi := result;
END byte_to_integer
```

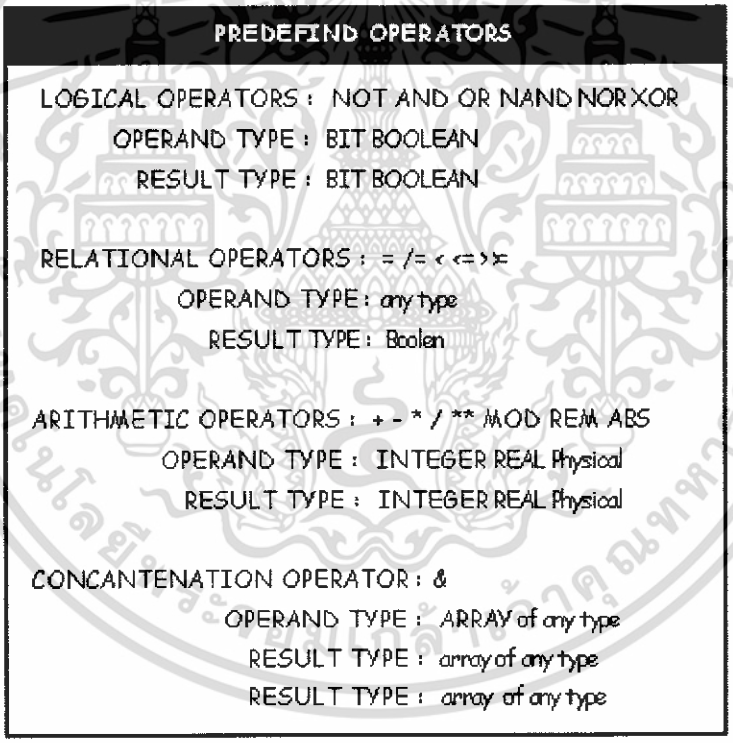
รูปที่ 2.12 การใช้โพรซีเจอร์

```
FUNCTION f (a, b, c:BIT) RETURN BIT  IS
        VARIABLE x:BIT;
BEGIN
        x := ((NOT a) AND (NOT b) AND c);
        RETURN x;
END f;
```

รูปที่ 2.13 การใช้ฟังก์ชัน

2.2.3.6โอเปอร์เรเตอร์

การบรรยายเชิงพฤติกรรมในภาษา VHDL มีตัวดำเนินการหรือโอเปอร์เรเตอร์ทางลอจิกและคณิตศาสตร์เช่นเดียว กับภาษาซอฟต์แวร์ทั่วไปดังรูปที่ 2.14



รูปที่ 2.14 ตัวดำเนินการใน VHDL

2.2.3.7เวลาและความพร้อมเพรียง

ในวงจรอิเล็กทรอนิกส์อุปกรณ์ทุกๆ ตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (Always Active) และจะมีเรื่องของเวลาเข้ามาเกี่ยวข้องในทุกๆเหตุการณ์ที่เกิดขึ้นเสมอ VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อให้สามารถบรรยายรูปแบบ และการพ้องกันของเวลา สำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายในส่วนของการบรรยายสถาปัตยกรรม จะมี

การทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โปรเซสซึ่งมีการทำงานภายในเป็นแบบลำดับคำสั่งก็ตาม ซึ่งหากมีหลายๆโปรเซสอยู่ภายในโครงสร้างเดียวกัน ทุกๆโปรเซสก็จะทำงานไปพร้อมๆกันด้วย

2.2.3.8สัญญาณและตัวแปร

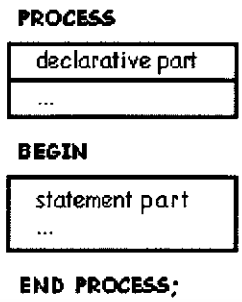
สัญญาณมีลักษณะเป็นเสมือนตัวกลางฮาร์ดแวร์ ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับสัญญาณจะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการส่งผ่านค่าของสัญญาณ เช่น w <= a AFTER 12 NS หมายถึงการกำหนดค่าสัญญาณ a ให้กับ w หลังจากเวลา ผ่านไป 12 นาโนวินาที ในทางตรงข้ามตัวแปรมีลักษณะเป็นเสมือนตัวกลางที่ใช้ในการส่งผ่านข้อมูล และไม่มีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย ซึ่งตัวแปรจะถูกใช้ในส่วนที่มีการทำงานเป็นแบบลำดับคำสั่งเช่นใน ฟังก์ชันโพรซีเจอร์ และโปรเซสสำหรับการกำหนดค่าให้กับตัวแปรจะใช้สัญลักษณ์ :=

2.2.3.9การบรรยายเชิงพฤติกรรม

การบรรยายลักษณะการทำงานของอุปกรณ์ฮาร์ดแวร์ในเชิงพฤติกรรม เป็นการบรรยายลักษณะการเปลี่ยนแปลงของข้อมูลในรูปแบบของอัลกอริธึม สำหรับการคำนวณผลลัพธ์ที่เกิดขึ้นซึ่งสืบเนื่องมาจากการเปลี่ยนแปลงสภาวะของข้อมูลที่เข้ามา โดยไม่คำนึงถึงลักษณะโครงสร้างหรือความสัมพันธ์ของอุปกรณ์ที่อยู่ภายใน ว่าจะเป็นอย่างไร ในหัวข้อนี้จะแสดงถึงการบรรยายเชิงพฤติกรรมแทนการใช้โมดูลฮาร์ดแวร์รวมถึงข้อกำหนดต่างๆ ที่ควรรู้

2.2.3.10โปรเซส

โปรเซสเป็นรูปแบบพื้นฐานอย่างหนึ่งที่ใช้ในการกำหนดให้กับสัญญาณ โปรเซสจะอยู่ในสถานะที่เตรียมพร้อมอยู่เสมอและจะปฏิบัติคำสั่งพร้อมๆกันกับโปรเซสอื่นๆที่อยู่ในสถาปัตยกรรมบรรยายเดียวกัน โดยโปรเซสจะปฏิบัติงานตามคำสั่งทันทีที่มีเหตุการณ์เกิดขึ้นกับสัญญาณที่อยู่ทางด้านขวามือของสัญลักษณ์กำหนดค่าให้กับสัญญาณ (<=) การบรรยายโปรเซสจะเริ่มต้นด้วยคำสั่ง PROCESS และจบด้วยคำสั่ง END PROCESS ในรูปที่ 2.15 เป็นการแสดงส่วนประกอบของการบรรยายแบบโปรเซส ซึ่งประกอบด้วยส่วนของการประกาศตัวแปร ที่ต้องใช้และส่วนของการปฏิบัติคำสั่งเพื่อให้ได้ผลลัพธ์ที่ต้องการ



รูปที่ 2.15 รูปแบบของการบรรยายแบบโปรเซส

2.2.3.11การกำหนดตัวดำเนินการภายในโปรเซส

ตัวดำเนินการภายในโปรเซสมี 3 ชนิดคือ ตัวแปร (Variable) ไฟล์ (File) และตัวคงที่ (Constant) ซึ่งตัวดำเนินการทั้งสามชนิดนี้ หากมีการประกาศไว้ในโปรเซสใดก็จะใช้ได้เฉพาะภายในโปรเซสนั้นเท่านั้น สำหรับการติดต่อกับภายนอกหรือระหว่างโปรเซสสามารถทำได้โดยใช้สัญญาณ (Signal) หรือตัวคงที่ที่ได้ประกาศไว้ในส่วนของ ARCHITECTURE ในรูปที่ 2.16 แสดงตัวอย่างการประกาศตัวกระทำภายในโปรเซส ซึ่งจะอยู่ระหว่างคำสั่ง PROCESS และ BEGIN และค่าเริ่มต้นที่ถูกกำหนดให้กับตัวดำเนินการภายในโปรเซส จะถูกนำมาใช้ในตอนเริ่มต้นของการปฏิบัติเพียงครั้งเดียวเท่านั้น ต่างกับค่าเริ่มต้นที่อยู่ภายในโปรแกรมย่อยจะถูกนำมาใช้ทุกครั้งที่มีการเรียกใช้โปรแกรมย่อยนั้น ๆ

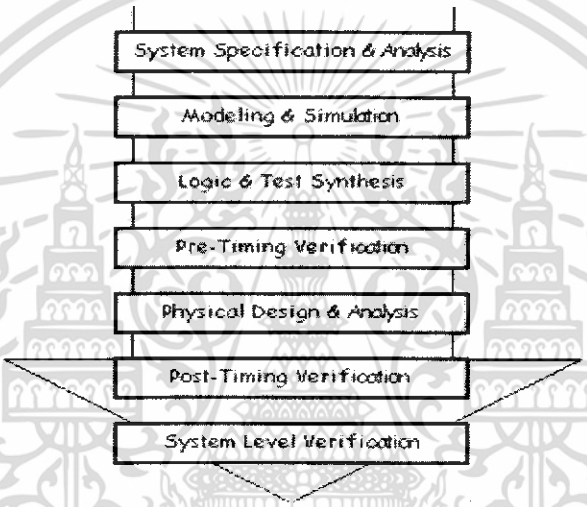
```
PROCESS
  FILE flush : TEXT IS IN "filename.dat";
  VARIABLE var : BIT;
  CONSTANT n : INTEGER := 0;
BEGIN
  ...
END PROCESS;
```

รูปที่ 2.16 ตัวอย่างการประกาศตัวดำเนินการภายในโปรเซส

2.2.3.12การออกแบบจากบนลงล่าง

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน วิศวกรหรือผู้ออกแบบมักจะมองการออกแบบให้อยู่ในรูปของ บล็อกไดอะแกรมก่อนที่ทำวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษา VHDL นั้นอนุญาตให้อธิบายและวิเคราะห์การทำงานของแต่ละบล็อก รวมถึงการปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามต้องการ นอกจากนี้ยังสามารถเพิ่มเติมในราย

ละเอียดในแต่ละขั้นตอนได้ ซึ่งหลักการนี้สอดคล้องกับหลักการออกแบบจากบนลงล่าง (Top - Down Design) นั่นเอง ถ้าทดลองเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Bottom - Up Design) จะเห็นได้ว่า การออกแบบจากล่างขึ้นบนจะใช้เวลาการออกแบบมากกว่า 90% เนื่องจากเป็นการวาดวงจรด้วยอุปกรณ์ต่างๆ (Schematic capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบก่อนแล้วจึงทำการจำลองการทำงานและตรวจสอบความถูกต้อง ดังนั้นการใช้ภาษา VHDL กับหลักการออกแบบจากบนลงล่าง จึงเป็นทางออกให้กับวิศวกรให้สามารถออกแบบและพัฒนาวงจรที่มีความซับซ้อนได้มากขึ้น ทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบด้วย



รูปที่ 2.17 ขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 2.17 แสดงถึงขั้นตอนของการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจมีข้อ=แตกต่างไปจากนี้บ้างเล็กน้อยเนื่องจากขั้นตอนของการผลิต (Implementation) สามารถกระทำได้หลายเทคโนโลยี สำหรับรายละเอียดของขั้นตอนการออกแบบจากบนลงล่างในแต่ละขั้นตอนมีดังนี้

-สร้างข้อกำหนดของความต้องการ และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ในการแก้ปัญหา

-เขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษา VHDL หรือ ภาษา HDL อื่น ๆ สำหรับบรรยายพฤติกรรมการทำงาน พร้อมทั้งจำลองการทำงาน เพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

-หลังจากที่ได้หลักการขั้นต้นพร้อมแนวความคิดที่ผ่านการตรวจสอบแล้ว หลักการนี้จะถูกเพิ่มเติมในรายละเอียดลงมา เป็นลำดับขั้นที่สอง จนกระทั่งอยู่ในระดับที่จะนำไปผลิตวงจริง หรือสังเคราะห์ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับวงจรออกแบบจะถูกกำหนดขึ้น และระบบช่วยการออกแบบจะสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้นให้อยู่ในรูปของวงจร ที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระดับเกท และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้นหรือไม่ก็อยู่ในรูปของNetlistที่สามารถนำไปผลิตในอุปกรณ์อื่นได้

-หลังจากการสังเคราะห์วงจรให้อยู่ในระดับเกทหรือ Netlist แล้ว ข้อมูลนี้จะถูกใช้สำหรับจำลองการทำงานในเรื่องความถูกต้องของฟังก์ชัน พร้อมกับนำข้อมูลที่เกี่ยวข้องกับเวลาเข้ามาประกอบการพิจารณาด้วย ซึ่งตามปกติแล้วอุปกรณ์ทางอิเล็กทรอนิกส์ทุกชิ้น จะมีเวลาหน่วงของการแพร่กระจาย (Propagation Delay Time) เสมอ ถึงแม้ว่าจะเป็นเวลาที่น้อยมากในระดับนาโนวินาทีก็ตาม แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชันต่างๆ จำนวน 10,000 เกทขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนต้องอาจทำให้การทำงานของวงจรรวมทั้งหมดผิดพลาดไปหรือไม่สามารถทำงานในย่านความถี่สัญญาณนาฬิกาที่สูงได้

-ผลิตเป็นวงจรจริง (Technology and device mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจจะอยู่ในรูปของแผงวงจรไฟฟ้าที่ประกอบด้วยอุปกรณ์หลายๆ ชิ้นหรืออยู่ในรูปของวงจรรวมASIC

-ทำการตรวจสอบการทำงานและตัวแปรทางด้านเวลาทั้งหมด เพื่อความถูกต้องของวงจรเป็นครั้งสุดท้าย ก่อนนำไปรวมเข้ากับอุปกรณ์อื่นๆให้เป็นระบบดิจิตอล เนื่องจากในขั้นตอนนี้วงจรที่ออกแบบ จะประกอบด้วยจุดต่อทางอินพุทและเอาท์พุท ซึ่งเป็นจุดต่อสำหรับการรับและส่งสัญญาณกับภายนอก

-นำวงจรที่ออกแบบไว้ประกอบเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบที่สมบูรณ์ แล้วทำการทดสอบการทำงานทั้งระบบร่วมกับอุปกรณ์อื่นๆ อีกครั้งเพื่อควบคุมคุณภาพของผลิตภัณฑ์

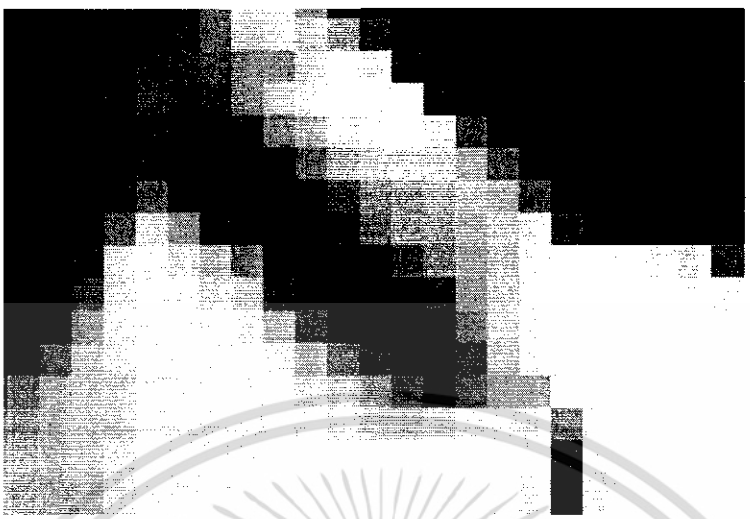
บทที่ 3

การส่งผ่านข้อมูลภาพที่ได้จาก CMOS Image Sensor และการแสดงผล

3.1 ภาพดิจิตอล

กล้องดิจิตอลใช้ตัวรับภาพในการบันทึกภาพแทนฟิล์ม ซึ่งบนตัวรับภาพจะประกอบด้วยตารางสี่เหลี่ยมเล็กๆ หลายแสนหลายล้านชิ้นเรียกว่า พิกเซล (Pixel) หนึ่งตารางสี่เหลี่ยมจะมีสีเพียงสีเดียวเท่านั้น ตารางสี่เหลี่ยมที่ประกอบรวมเป็นภาพดิจิตอลภาพหนึ่งนั้นมีขนาดเล็กมากๆ เราจึงดูไม่ออกว่าภาพเหล่านั้นที่จริงแล้ว คือการเรียงต่อกันของตารางสีดังรูปที่ 3.1 ซึ่งเป็นภาพที่เราเห็นปกติ หากเราเปิดภาพดิจิตอลในเครื่องคอมพิวเตอร์แล้วใช้แว่นขยายส่องดูจะเห็นตารางสีสี่เหลี่ยมได้อย่างชัดเจนดังรูปที่ 3.2 การเรียงตัวกันอย่างเป็นระเบียบของตารางสีเหล่านี้เป็นเสมือนแผนที่ของตารางสี ซึ่งเป็นที่มาของชื่อภาพประเภทนี้ซึ่งเรียกว่าเป็นภาพบิทแม็พ (Bit Map)

รูปที่ 3.1 ภาพปกติ



รูปที่ 3.2 ภาพที่เกิดจากการขยายแล้ว

คุณภาพของภาพดิจิตอล ไม่ว่าจะเป็นภาพที่นำไปอัด หรือแสดงบนจอมอนิเตอร์ จะขึ้นอยู่กับข้อมูลของภาพที่มีอยู่ หรืออีกนัยหนึ่งก็คือ ขึ้นอยู่กับจำนวนของพิกเซลที่มีอยู่บนตัวรับภาพ จำนวนพิกเซลที่มากขึ้นหมายถึงรายละเอียดของข้อมูลภาพที่สูงขึ้น ซึ่งจะมีส่วนช่วยให้ได้ภาพถ่ายที่มีคุณภาพดีขึ้นไปด้วย (เมื่อนำไปอัด – ขยาย) เราวัดจำนวนของพิกเซลในภาพดิจิตอลโดยการนำเอาจำนวนของพิกเซลแนวตั้งคูณกับจำนวนของพิกเซลแนวนอน ซึ่งค่าที่ได้ก็คือค่าความละเอียดของกล้องดิจิตอลที่เรารู้จักกันว่ากล้องตัวนี้ 3 ล้าน กล้องตัวนี้ 5 ล้านหรือกล้องตัวนี้ 6 ล้านเป็นต้น

ภาพขนาด 1600 x 1200 = 1,920,000 พิกเซล หรือเป็นความละเอียดสูงสุดที่สุดที่กล้องดิจิตอล 2 ล้านพิกเซลสามารถบันทึกได้ ความละเอียดของกล้องดิจิตอลที่ระบุไว้จะดูที่ความละเอียดสูงสุดที่กล้องสามารถบันทึกได้เท่านั้น (นอกจากกล้องบางรุ่นที่ระบุความละเอียดโดยใช้หลักการของการประมวลผลและเพิ่มข้อมูลให้กับภาพ)

3.1.1 ความรู้เกี่ยวกับ Image Sensor

ตัว Sensor ในการรับภาพถือเป็นปัจจัยสำคัญที่กำหนดประเภทของกล้องดิจิตอลในปัจจุบัน นอกจากจะเป็นตัวกำหนดประเภทแล้ว ยังเป็นตัวบ่งบอกถึงคุณภาพของตัวกล้องอีกด้วย นั่นเพราะว่ากล้องที่ใช้เทคโนโลยี CMOS นั้นเป็นกล้องที่จัดอยู่ในระดับ Low end หรือกล้องราคาต่ำสำหรับมือใหม่ อีกทั้งยังพบมากในกล้องเก่าๆ ที่ตกรุ่นไปแล้ว ส่วนกล้องที่ใช้เทคโนโลยี CCD นั้นจะเป็นกล้องที่ออกมาสู่ตลาดในช่วงปัจจุบัน และเป็นกล้องที่มีคุณภาพของภาพถ่ายที่ดีกว่า แต่ต้องไม่ลืมด้วยว่าคุณก็จะต้องแลกมาด้วยจำนวนเงินที่เพิ่มสูงขึ้นเช่นเดียวกัน

ทั้ง CCD และ CMOS นั้นเริ่มต้นจากจุดเดียวกัน นั่นคืออาศัยหลักการทำงานในการอาศัย Photosite ให้เปลี่ยนแสงที่ตกกระทบให้กลายเป็นอิเล็กตรอน เพื่อบ่งบอกค่าของแสงสีนั้นๆ ภายในตัว Sensor ทั้งสองชนิดนี้จึงประกอบไปด้วย Photosite ขนาดเล็กๆนับล้านๆชิ้น เพื่อทำหน้าที่ในการรับแสง ข้อแตกต่างของทั้งสองเทคโนโลยีเกิดขึ้นเมื่อเข้าสู่ขั้นตอนของการคำนวณค่าของแสงนั้นๆ จากแต่ละ Photosite ในอุปกรณ์ที่ใช้ CCD นั้นจะทำการประจุค่านั้นๆโดยตรงในแต่ละ Photosite จากนั้นจะแปลงค่าอนาล็อกของแสงที่ตกกระทบให้กลายไปเป็นค่าดิจิตอล กระบวนการทั้งหมดนี้สามารถเกิดขึ้นได้อย่างรวดเร็ว แตกต่างจาก CMOS ที่ถึงแม้ว่าแต่ละ Photosite จะสามารถประจุค่าได้โดยตรงเช่นกัน แต่กระบวนการส่งผ่านข้อมูลเหล่านั้นยังต้องอาศัยสายขนาดเล็กมากๆ เพื่อทำการส่งผ่านข้อมูลเหล่านั้นเพื่อประมวลผลอีกต่อหนึ่ง

ในกระบวนการผลิตนั้น CCD จะใช้กรรมวิธีพิเศษในการสร้าง ความสามารถในการส่งผ่านประจุโดยตรงไปยังตัว Chip โดยไม่มีปัญหาในการตัดทอนสัญญาณ ซึ่งด้วยวิธีการนี้จึงต้องอาศัยขบวนการผลิตที่มีคุณภาพสูงมากเป็นพิเศษ เพื่อให้ได้ตัว Sensor ที่มีคุณภาพและมีความไวต่อแสงอย่างยิ่งยวด ในขณะที่ CMOS นั้นยังอาศัยเทคโนโลยีการผลิตในรูปแบบเดิมๆ ที่ผ่านมา และแทบไม่มีการเปลี่ยนแปลงอะไรมากนัก ซึ่งอาศัยเทคนิค เช่นเดียวกับการผลิต Microprocessor ดังนั้นด้วยความแตกต่างในเทคโนโลยีและกระบวนการผลิตนี้เอง ทำให้ทั้ง CCD และ CMOS นี้มีต้นทุนในการผลิตและมีคุณภาพของการประมวลผลที่แตกต่างกันไปด้วย

3.1.2 เปรียบเทียบ CCD กับ CMOS

- CCD Sensors ถูกสร้างด้วยเทคโนโลยีและกรรมวิธีการผลิตขั้นสูง ทำให้ไม่มีปัญหาในการรบกวนของสัญญาณภาพ ในขณะที่ CMOS นั้นอาศัยกระบวนการเก่าๆในการผลิต อีกทั้งยังเต็มไปด้วยสัญญาณรบกวนที่จะลดทอนคุณภาพของภาพที่ได้

- แต่ละ Photosite ของ CMOS นั้นจะประกอบไปด้วย ทรานซิสเตอร์จำนวนมากที่วางเรียงติดกันอยู่ ทำให้ปัญหาในการส่งผ่านข้อมูลมีความผิดพลาดสูง อีกทั้งที่ Cell รับแสงนั้นยังมีความไวต่อแสงน้อยกว่าเทคโนโลยี CCD

- อุปกรณ์ CCD นั้นมีอัตราบริโภคพลังงานสูงกว่า CMOS นั่นเพราะว่า มันต้องอาศัยการประจุอิเล็กตรอนโดยตรง เพื่อทำให้เกิดสัญญาณและค่าดิจิตอลถ่ายทอดออกไปได้ ซึ่งเมื่อเปรียบเทียบแล้ว มีอัตราสูงกว่า 100 เท่าเลยทีเดียว เมื่อเทียบกับ CMOS

- การผลิต CMOS sensor นั้น สามารถพิมพ์ลายวงจรลงบนแผ่นเวเฟอร์ ซิลิคอน มาตรฐานแทบทุกชนิดได้ ในขณะที่ CCD นั้นจะต้องอาศัยแผ่นซิลิคอนที่ถูกผลิตมาเป็นพิเศษ เพื่อทำการผลิต และนี่คืออีกสาเหตุหนึ่ง ที่ทำไมต้นทุนการผลิตถึงได้แตกต่างกัน

ด้วยความแตกต่างกันในหลายๆ ด้านนี้เอง ทำให้คุณจะเห็น CCD sensor นี้อยู่ในเฉพาะกล้องดิจิตอล รุ่นใหม่ๆระดับมืออาชีพ ซึ่งให้ความสมจริงของภาพ และมีความละเอียดสูง ในขณะที่ CMOS sensor นั้นส่วนใหญ่มักพบในกล้องดิจิตอลมือสมัครเล่นหรือในระดับ Low end ซึ่งมีราคาไม่แพงนัก แต่ก็ต้องไม่ลืมด้วยว่าไม่อาจคาดหวังถึงความละเอียดและคุณภาพของภาพถ่ายในระดับมืออาชีพได้ อย่างไรก็ตามถึงตอนนี้ผู้ผลิต CMOS sensor รายใหญ่ก็กำลังพัฒนาและปรับปรุงเพื่อให้คุณภาพของ CMOS ก้าวขึ้นมาเทียบเคียงกับ CCD ได้

3.2 ระบบบัส USB

ระบบบัส USB เป็นระบบบัสที่ถูกออกแบบมาให้มีความง่ายในการเชื่อมต่อ และทรงประสิทธิภาพในการสื่อสารข้อมูลกับอุปกรณ์รอบข้างหลายๆชนิด โดยปราศจากข้อจำกัด และการขัดขวางของการอินเตอร์เฟสทางด้านฮาร์ดแวร์ ในโลกของ USB จะไม่มีการใช้ดิพสวิทช์ หรือแม้แต่จั๊มเปอร์ เพื่อกำหนดค่าทางด้านฮาร์ดแวร์แต่อย่างใด ซึ่งการตั้งค่าการทำงานต่างๆ จะถูกกระทำโดยระบบปฏิบัติการอย่างอัตโนมัติ นอกจากนี้การเชื่อมต่ออุปกรณ์ USB เข้ากับระบบยังสามารถทำในขณะที่เครื่องคอมพิวเตอร์ยังคงทำงานอยู่ได้ ซึ่งการใช้งานในลักษณะนี้คือรูปแบบของระบบปลั๊กแอนด์เพลย์ (Plug and Play) อย่างแท้จริง และในส่วนของการเพิ่มจำนวนพอร์ทการสื่อสารข้อมูลก็สามารถทำได้อย่างง่ายดายด้วยการใช้ USB ฮับ (USB Hub) มาต่อพ่วงเข้ากับระบบ จุดเด่นที่น่าสนใจอีกประการหนึ่งของระบบบัสนี้คือ ความเร็วในการส่งถ่ายข้อมูลซึ่งสูงถึง 480 เมกะบิทต่อวินาทีสำหรับ USB ในเวอร์ชั่น 2.0 ซึ่งสูงกว่าการเชื่อมต่อแบบขนานและอนุกรมเป็นอย่างมาก

3.2.1 จุดเด่นหลักๆของระบบบัส USB

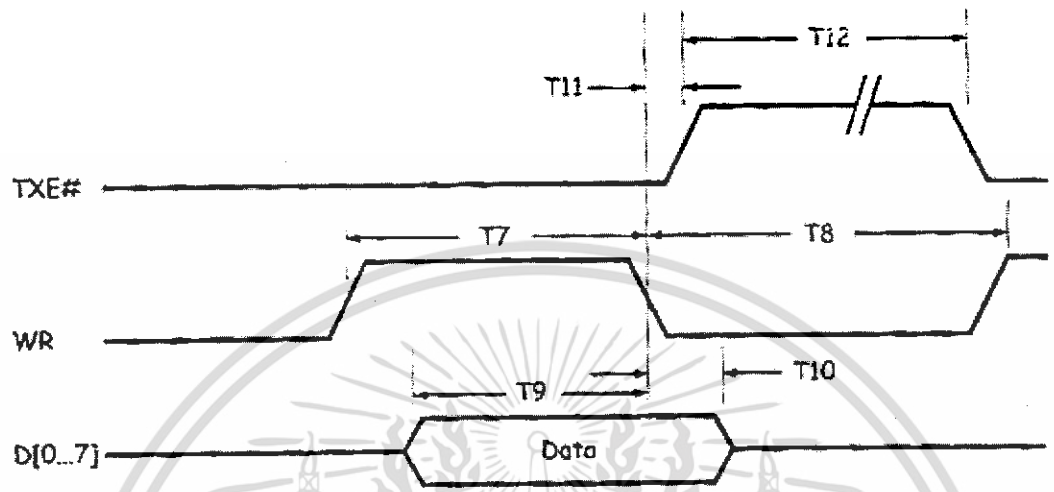
- ผู้ใช้สามารถนำอุปกรณ์ I/O มาต่อเข้ากับพอร์ท USB ในขณะที่เครื่องคอมพิวเตอร์กำลังทำงานอยู่ได้ (Hot-Pluggable)

- ง่ายต่อการใช้งาน ซึ่งเครื่องคอมพิวเตอร์จะรู้จัก และจดจำอุปกรณ์ I/O ที่ถูกนำมาต่อเข้าไปในระบบโดยผ่านทางดีไวซ์ไดร์ฟเวอร์ที่เหมาะสม อีกทั้งการตั้งค่าต่างๆ จะเป็นไปอย่างอัตโนมัติ
- ลดความสับสนของการเชื่อมต่อด้วยการใช้คอนเน็คเตอร์เพียงชนิดเดียว
- ประสิทธิภาพการส่งถ่ายข้อมูลสูง ด้วยการส่งถ่ายข้อมูลที่มีความเร็วถึง 480 เมกะบิทต่อวินาทีสำหรับอุปกรณ์แบบไฮสปีด ซึ่งมีค่าสูงกว่าพอร์ทขนานและอนุกรมหลายเท่า
- สามารถต่ออุปกรณ์ได้ถึง 127 ตัว
- สายเคเบิลที่ใช้เชื่อมต่อจะมีสายของแหล่งจ่ายไฟรวมอยู่ภายในตัวมันด้วย
- มีการจัดการพลังงานที่ชาญฉลาด ซึ่งอุปกรณ์ต่างๆจะมีการลดการใช้กำลังงานของตัวมันเองลงเมื่อไม่มีการใช้งาน
- มีการตรวจจับและแก้ไขความผิดพลาดของข้อมูลอย่างอัตโนมัติ

3.2.2 ลักษณะการทำงานของ Ezy USB-M01

การส่งข้อมูลจากวงจรที่เชื่อมต่อกับโมดูล เช่น ไมโครคอนโทรลเลอร์ ไปยังโฮสคอมพิวเตอร์ สามารถทำได้โดยการเขียนข้อมูลขนาด 1 ไบท์ ไปยังโมดูลในขณะที่ขา TXE# มีสภาวะเป็นลอจิกต่ำ ถ้าบัฟเฟอร์สำหรับส่ง (384 ไบท์) ถูกเขียนข้อมูลลงไปแล้ว หรือยังอยู่ในสภาวะที่ไม่ว่าง (Busy) เนื่องจากการเขียนข้อมูลในครั้งก่อนหน้านี้ ตัวโมดูลจะทำให้สภาวะที่ขา TXE# นี้เป็นลอจิกสูงเพื่อหยุดการเขียนข้อมูลจนกระทั่งข้อมูลของ FIFO ถูกส่งถ่ายผ่านสาย USB ไปยังโฮสแล้ว ซึ่งจะเป็นไปตามรูปที่ 3.3 และตารางที่ 3.1

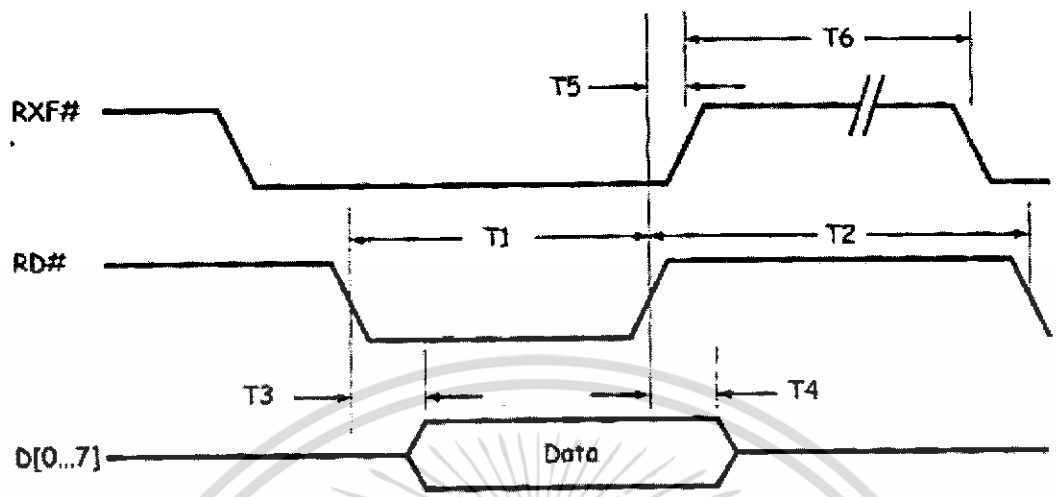
เมื่อโฮสส่งข้อมูลไปยังอุปกรณ์รอบข้างโดยผ่านสาย USB แล้ว ตัวโมดูลจะทำให้ขา RXF# อยู่ในสภาวะลอจิกต่ำ เพื่อเป็นการบอกวงจรเชื่อมต่อว่าในขณะนั้นมีข้อมูลอยู่อย่างน้อย 1 ไบท์ พร้อมให้วงจรเชื่อมต่ออ่านออกไปได้ ข้อมูลจะถูกอ่านโดยวงจรเชื่อมต่อจนกระทั่งขา RXF# ของโมดูลมีสภาวะเป็นลอจิกสูง ซึ่งเป็นการบ่งบอกว่าข้อมูลที่อยู่ในบัฟเฟอร์หมดลงแล้ว ซึ่งจะเป็นไปตามรูปที่ 3.4 และตารางที่ 3.2



รูปที่ 3.3 ไดอะแกรมเวลาสำหรับการเขียนข้อมูล Ezy USB-M01

เวลา	คำอธิบาย	ค่าต่ำสุด	ค่าสูงสุด	หน่วย
T7	ความกว้างของพัลส์ในช่วงเวลาที่ WR ทำงาน	50	-	ns
T8	ค่าเวลาพรีชาร์ทระหว่างที่ WR หยุดทำงานถึง WR เริ่มทำงานอีกครั้ง	50	-	ns
T9	ช่วงเวลาที่ WR ทำงานเพื่อรับข้อมูล	-	20	ns
T10	ช่วงเวลาข้อมูลยังคงอยู่หลังจากที่ WR หยุดทำงาน	10	-	ns
T11	ช่วงเวลาที่ WR หยุดทำงานเมื่อเทียบกับ TXE#	5	25	ns
T12	ช่วงเวลาที่TXE# หยุดทำงานหลังจากกระบวนการเขียน	80	-	ns

ตารางที่ 3.1 ค่าเวลาต่างๆสำหรับการเขียนข้อมูล Ezy USB-M01

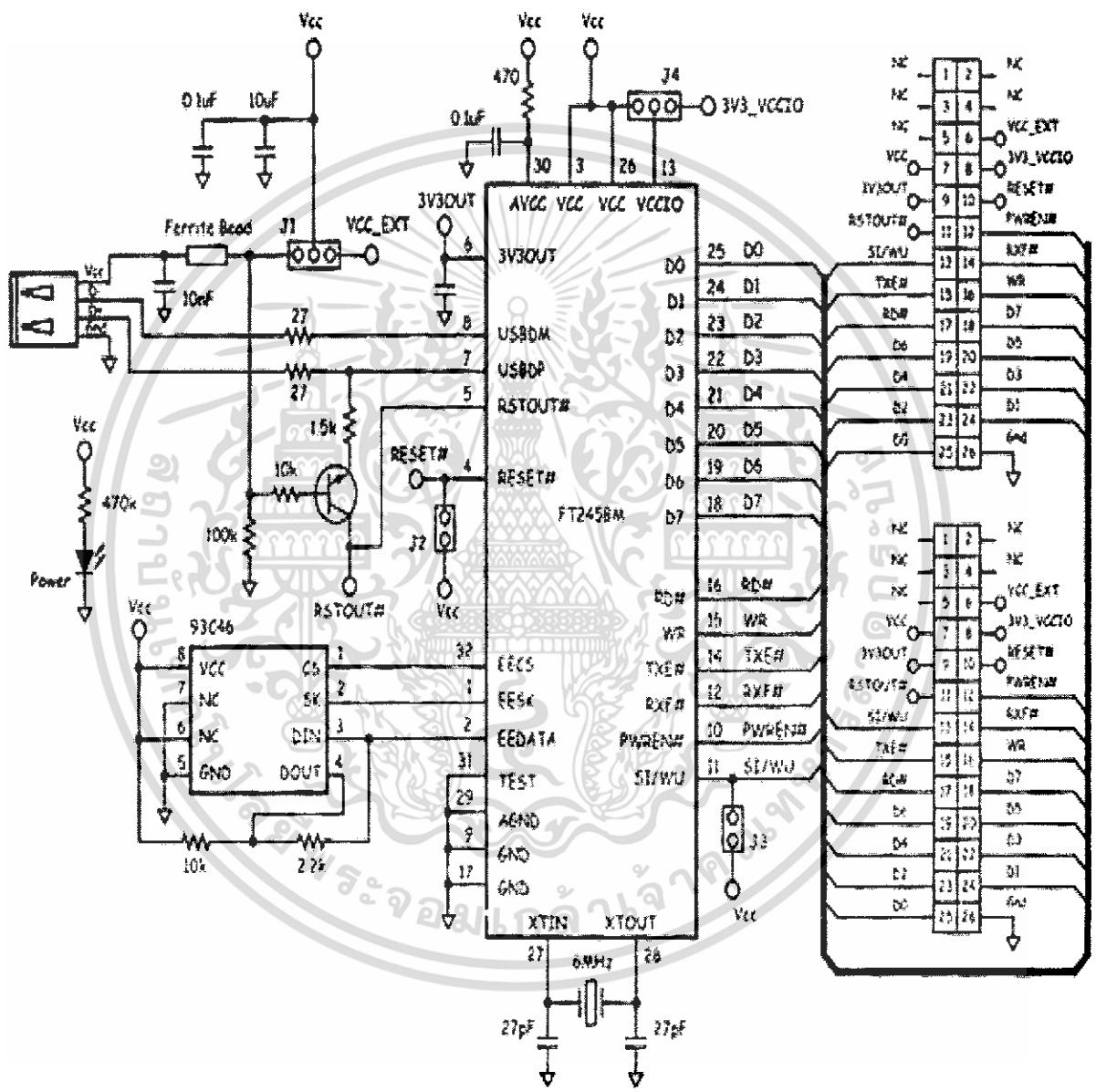


รูปที่ 3.4 ไดอะแกรมเวลาสำหรับการอ่านข้อมูล Ezy USB-M01

เวลา	คำอธิบาย	ค่าต่ำสุด	ค่าสูงสุด	หน่วย
T1	ความกว้างของพัลส์ในช่วงเวลาที่ RD ทำงาน	50	-	ns
T2	ค่าเวลาพรีชาร์ทระหว่างที่ RD หยุดทำงานถึง RD เริ่มทำงานอีกครั้ง	50	-	ns
T3	ช่วงเวลาที่ RD ทำงานเพื่อรับข้อมูล	-	30	ns
T4	ช่วงเวลาข้อมูลยังคงอยู่หลังจากที่ RD หยุดทำงาน	10	-	ns
T5	ช่วงเวลาที่ RD หยุดทำงานเมื่อเทียบกับ RXF#	5	25	ns
T6	ช่วงเวลาที่RXF# หยุดทำงานหลังจากกระบวนการอ่าน	80	-	ns

ตารางที่ 3.2 ค่าเวลาต่างๆสำหรับการอ่านข้อมูล Ezy USB-M01

ในการนำไปใช้งาน Ezy USB-M01 จะมีลักษณะของวงจรภายในดังรูปที่ 3.5 ซึ่งจะบอกถึงชื่อและตำแหน่งของขาต่างๆ



รูปที่ 3.5 วงจรภายในของ Ezy USB-M01

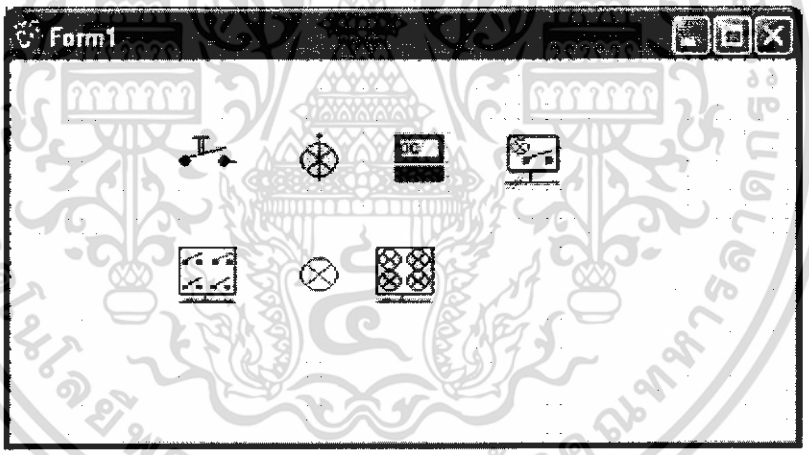
3.3 การเขียนโปรแกรมแบบวิชวล C++ Builder

3.3.1 การเขียนโปรแกรมแบบวิชวล

วิชวล (Visual) หมายถึงภาพหรือสิ่งที่เรามองเห็น การเขียนโปรแกรมแบบวิชวลหรือวิชวลโปรแกรมมิ่ง (Visual Programming) จึงหมายถึงการเขียนโปรแกรมด้วยภาพ หรือการเขียนโปรแกรมด้วยสิ่งที่เรามองเห็น

ส่วนประกอบเบื้องต้นสำหรับใช้เขียนโปรแกรมแบบวิชวลมี 4 รายการดังต่อไปนี้

1. ฟอร์ม (Form) เป็นวินโดว์ (Window – บริเวณสี่เหลี่ยมผืนผ้าบนจอภาพ) ว่างใช้สำหรับออกแบบโปรแกรมดังรูปที่ 3.6 การทำงานต่างๆของโปรแกรมจะปรากฏอยู่บนฟอร์ม



รูปที่ 3.6 แสดงฟอร์มซึ่งเป็นวินโดว์ของโปรแกรม

2. คอมโพเนนท์ (Component) เป็นส่วนประกอบต่างๆที่จะต้องใช้ในโปรแกรม เช่น เมนู (Menu) ปุ่มหรือบัททอน (Button – ปุ่มสำหรับกดให้โปรแกรมทำงานอย่างหนึ่ง) เมสเสจบ็อกซ์ (Message Box – กรอบแสดงข้อความ) ไดอะล็อกบ็อกซ์ (Dialog Box – กรอบสำหรับโต้ตอบกับโปรแกรม)

3. คำสั่งสำหรับจัดลักษณะและการทำงานของคอมโพเนนท์

4. คำสั่งสำหรับควบคุมการทำงานของโปรแกรม

3.3.2 C++ และ C++ Builder

C++ และ C++ Builder ต่างก็เป็นภาษาคอมพิวเตอร์ ซึ่งใช้สำหรับเขียนโปรแกรม เพื่อสั่งให้คอมพิวเตอร์ทำงานต่างๆ ตามที่เราต้องการ C++ และ C++ Builder มีคำและกฎเกณฑ์ในการเขียนเหมือนกันเกือบทุกประการ แต่มีวิธีการเขียนโปรแกรมต่างกัน กล่าวคือ C++ ใช้วิธีเขียนโปรแกรมด้วยอักษรและเครื่องหมาย แต่ C++ Builder ใช้วิธีเขียนโปรแกรมแบบวิชวล ดังนั้นผู้ที่เขียนโปรแกรมด้วย C++ จึงสามารถเปลี่ยนมาเขียนโปรแกรมด้วย C++ Builder ได้อย่างรวดเร็ว เพราะเปลี่ยนแปลงเฉพาะวิธีการเท่านั้น ซึ่งวิธีการของ C++ Builder จะทำให้เราสามารถเขียนโปรแกรมได้ง่ายกว่าวิธีการของ C++

C++ Builder เป็น ANSI C++ ซึ่งหมายความว่า C++ Builder เป็นภาษา C++ ที่เป็นไปตามมาตรฐาน ANSI (American National Standards Institute - สถาบันมาตรฐานแห่งชาติของสหรัฐอเมริกา ทำหน้าที่กำหนดมาตรฐานต่างๆรวมทั้งภาษา C++ ด้วย) ดังนั้นผู้ที่เคยเขียนโปรแกรมด้วย ANSI C++ มาก่อนไม่ว่าจะเป็นผลิตภัณฑ์ของบริษัทใดก็ตาม จะสามารถเขียนโปรแกรมด้วย C++ Builder ได้ทันที

วิธีการเขียนโปรแกรมด้วย C++Builder ทำได้โดยการนำคอมโพเนนท์ที่เรามองเห็นมาวางในฟอร์ม แล้วกำหนดลักษณะและการทำงานให้กับคอมโพเนนท์นั้น ในบางโปรแกรมอาจจะต้องเขียนคำสั่งควบคุมการทำงานของโปรแกรมเพิ่มเติม ต่อจากนั้นจึงคอมไพล์ (Compile) และรัน (Run) โปรแกรมนั้นได้ ผลจากการคอมไพล์โปรแกรมจะได้ไฟล์ชนิด .EXE ซึ่งสามารถนำไปรันในคอมพิวเตอร์เครื่องอื่นได้อย่างอิสระ โดยไม่ต้องเกี่ยวข้องกับ C++Builder อีกเลย

C++Builder เป็นภาษา C++ ประเภทที่สามารถสร้างโปรแกรมใช้งานได้อย่างรวดเร็ว ภาษาคอมพิวเตอร์ประเภทนี้มีชื่อย่อว่า (RAD – Rapid Application Development) ซึ่งหมายความว่าด้วย C++Builder เราจะสามารถเขียนโปรแกรมภาษา C++ เพื่อรันในวินโดวส์ได้เร็วกว่าและง่ายกว่าการเขียนโปรแกรมแบบธรรมดา

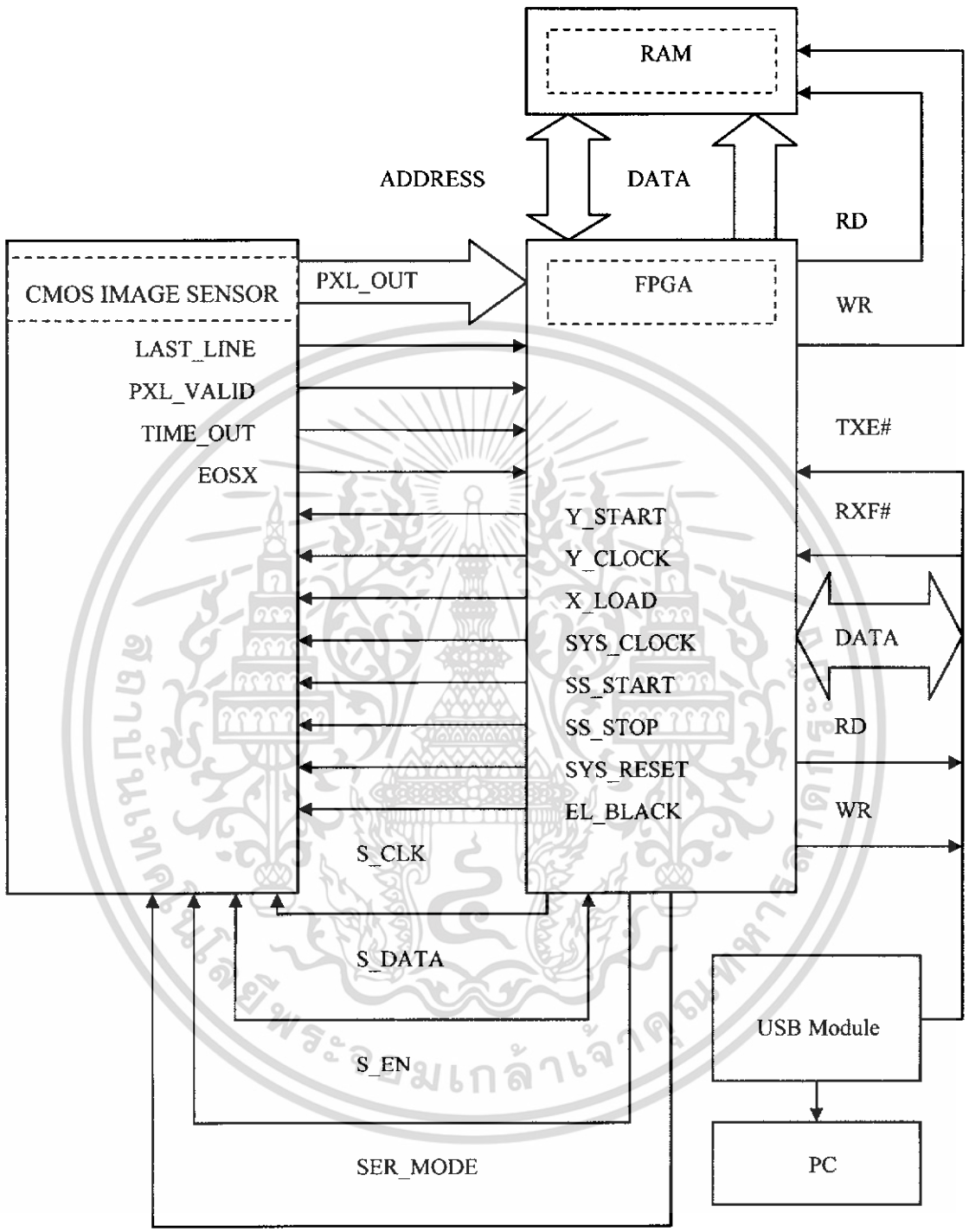
บทที่ 4
หลักการและการออกแบบโครงสร้างทางฮาร์ดแวร์

4.1 หลักการทำงานของระบบ

โดยการประยุกต์ใช้ FPGA นี้ อุปกรณ์ที่ใช้ในการออกแบบจะประกอบด้วยสามส่วนประกอบหลักคือ CMOS Image Sensor, FPGA และ RAM โดย FPGA จะเป็นตัวควบคุมการทำงานเกือบทั้งหมด ไม่ว่าจะเป็นการสร้างสัญญาณเพื่อนำไปเซ็ตค่ารีจิสเตอร์และค่าต่างๆให้กับ CMOS Image Sensor การรับค่าข้อมูลภาพมาจาก CMOS Image Sensor เพื่อทำการจัดเก็บ ประมวลผล และส่งผ่านข้อมูลเพื่อนำไปแสดงผลต่อไป

ในส่วนของ CMOS Image Sensor จะให้เอาท์พุทเป็นสัญญาณดิจิตอลแทนค่าในแต่ละพิกเซล เราสามารถทำการโปรแกรมด้วย FPGA ให้รับข้อมูลออกมาและทำการจัดเก็บค่าข้อมูลพิกเซลไว้ใน RAM แล้วยังทำการส่งค่าข้อมูลพิกเซลใน RAM ผ่านพอร์ต USB เพื่อที่จะแสดงผลทางคอมพิวเตอร์ต่อไป

ดังนั้นการทำงานของระบบจะมีลักษณะดังรูปที่ 4.1 ซึ่ง FPGA จะต้องติดต่อกับอุปกรณ์รอบข้าง คือ CMOS Image Sensor , RAM และ USB Module ซึ่งลักษณะการติดต่อในแต่ละอุปกรณ์ก็จะมีความแตกต่างกันออกไป มีทั้งการรับอินพุทเข้ามาและการส่งเอาท์พุทออกไป เพื่อควบคุมการทำงานในแต่ละส่วน



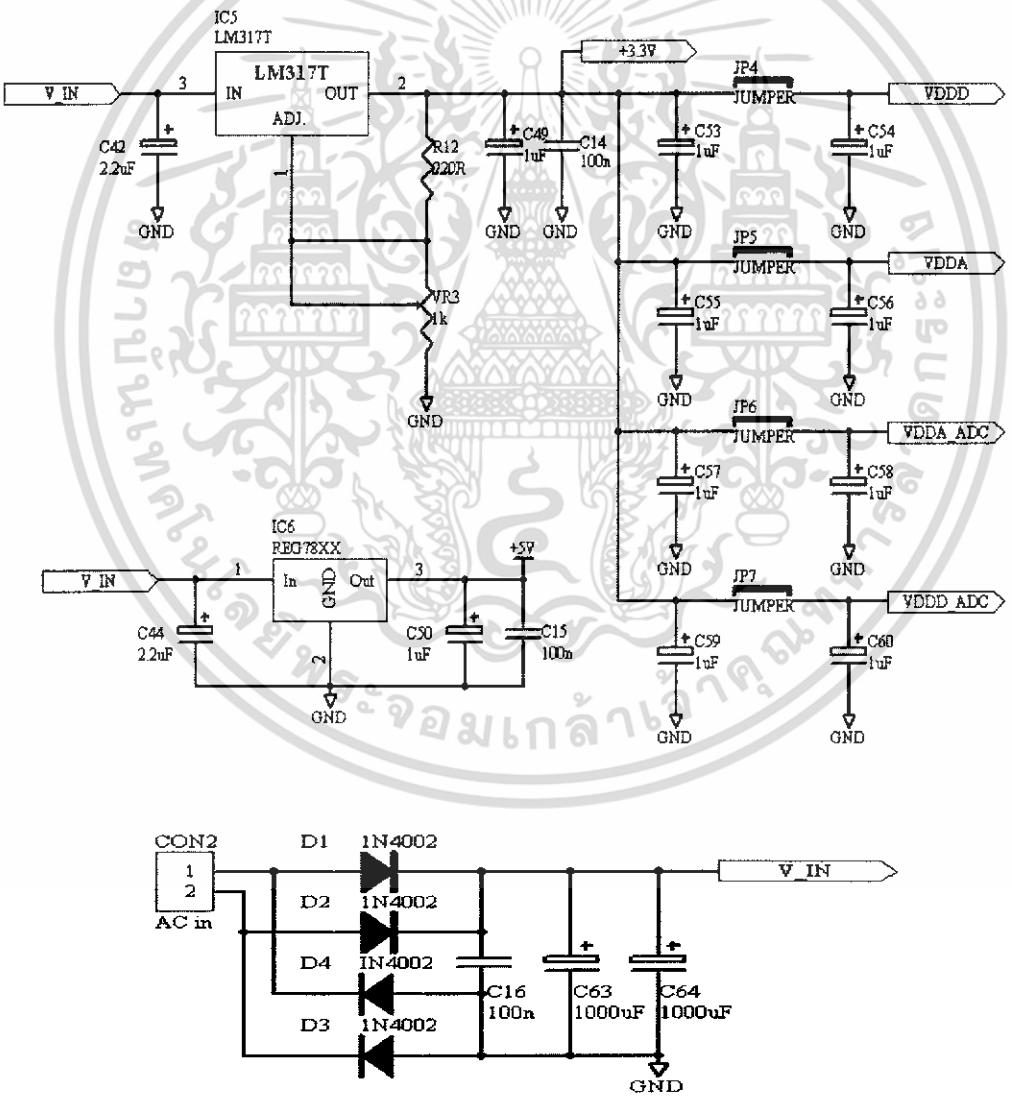
รูปที่ 4.1 แสดงบล็อกไดอะแกรมการทำงานของระบบ

4.2 การออกแบบโครงสร้างทางฮาร์ดแวร์

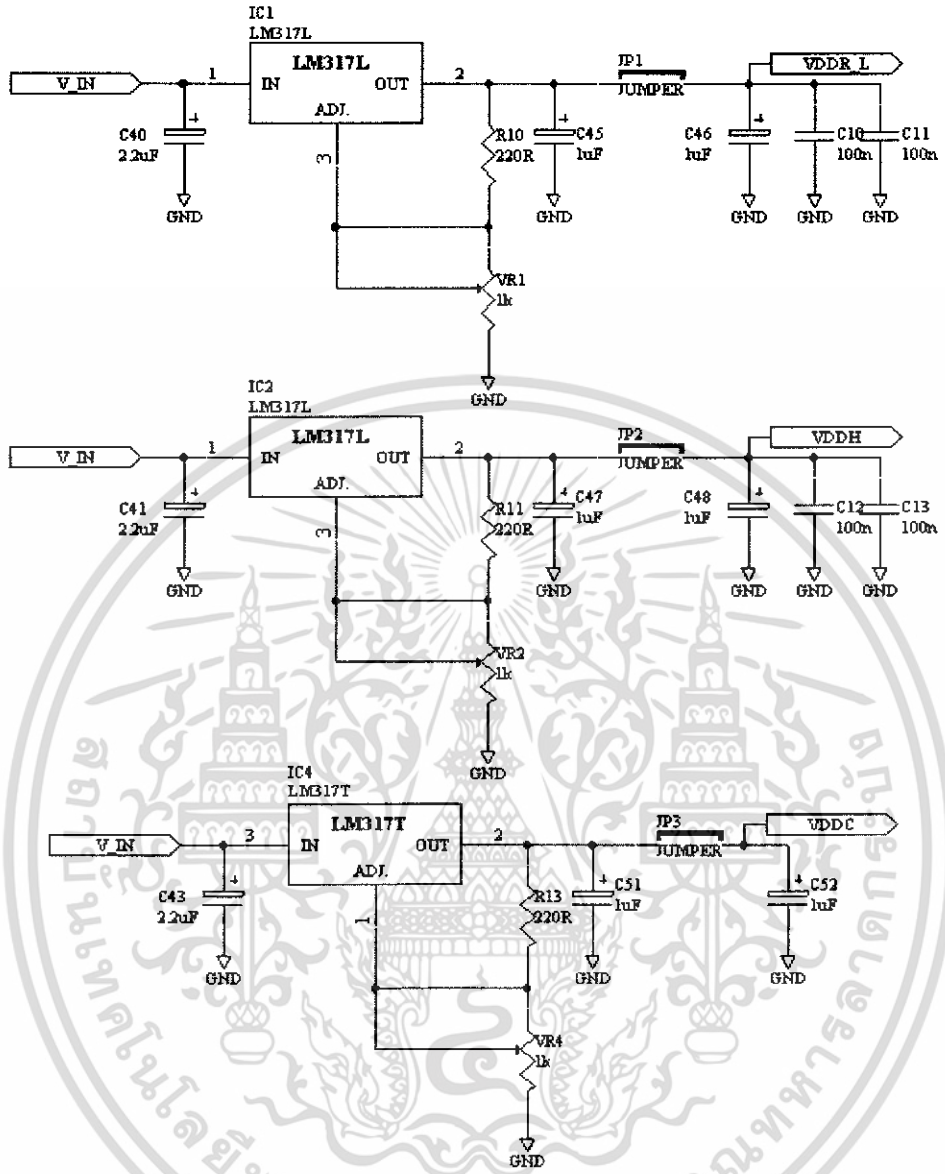
4.2.1 วงจรในส่วนของแหล่งจ่ายไฟให้กับ CMOS Image Sensor

ในส่วนวงจรของแหล่งจ่ายไฟดังรูปที่ 4.2 เป็นชุดแหล่งจ่ายที่ปรับค่าได้ ซึ่งให้เอาท์พุทออกไป 4 เอาท์พุท คือ VDDD, VDDA, VDDA_ADC และ VDDD_ADC ซึ่งทั้งหมดจะมีค่าเอาท์พุทเท่ากันและปรับใช้งานที่ +3.3 V แต่จะจ่ายให้กับ CMOS Image Sensor แต่ละจุดต่างกัน

ส่วนวงจรของแหล่งจ่ายไฟดังรูปที่ 4.3 จะให้เอาท์พุทเดียวในแต่ละวงจร คือสามารถแยกปรับค่าเอาท์พุทแต่ละตัวได้ซึ่งค่าที่ใช้ประกอบไปด้วย VDDR_L (+4.5V), VDDH (+4.5V) และ VDDC (+3.3V)



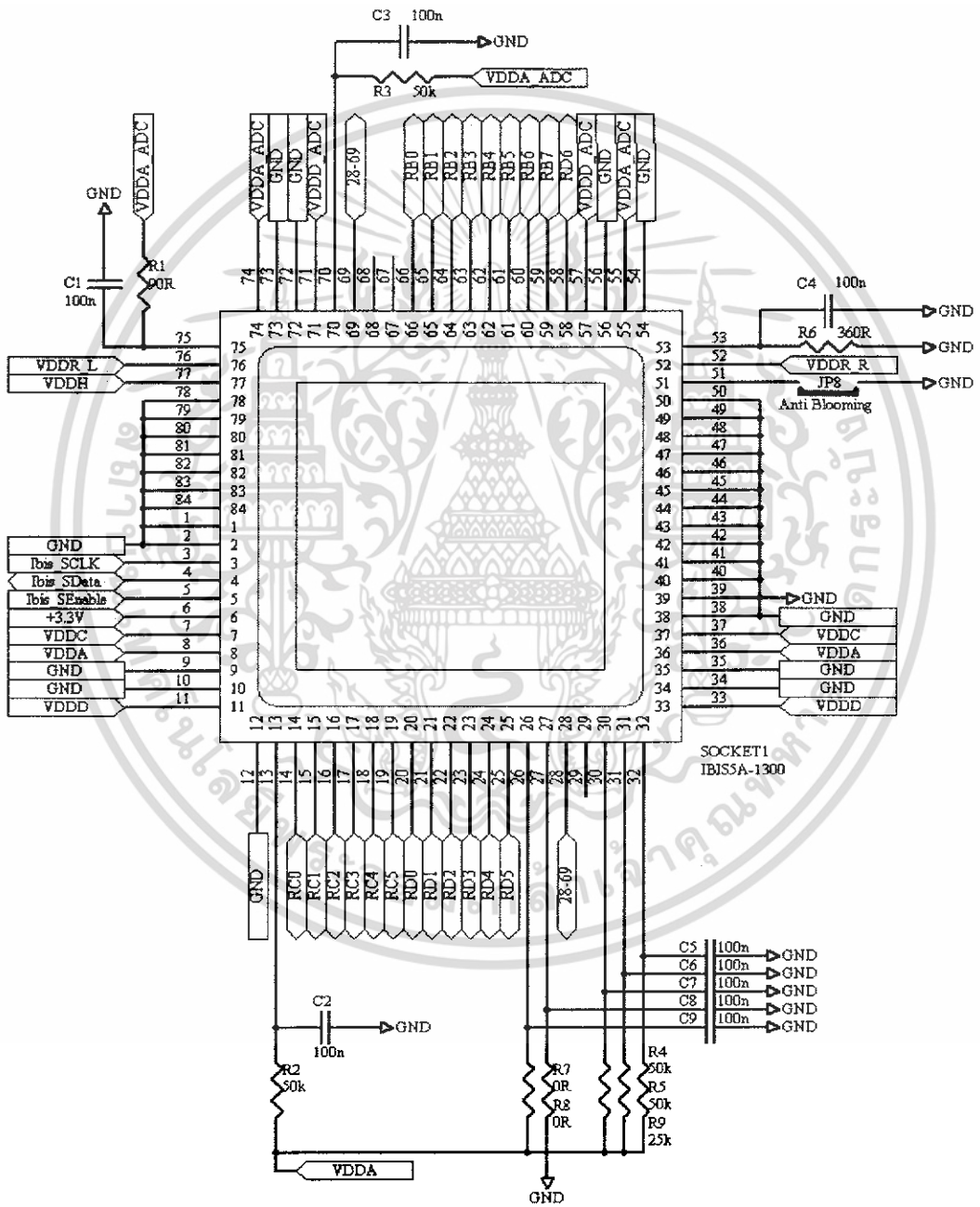
รูปที่ 4.2 ชุดแหล่งจ่ายไฟให้กับ CMOS Image Sensor ชุด 1



รูปที่ 4.3 ชุดแหล่งจ่ายไฟให้กับ CMOS Image Sensor ชุด 2

4.2.2 การจัดวงจรในส่วนของ CMOS Image Sensor

การจัดวงจรในส่วนของ CMOS Image Sensor ซึ่งมีทั้งหมด 84 ขา แสดงดังรูปที่ 4.4 ซึ่งจะมีทั้งส่วนที่เป็นอินพุท ส่วนที่เป็นเอาท์พุท ส่วนที่เป็นไฟเลี้ยงและส่วนที่ไม่ได้ใช้งาน ซึ่งรายละเอียดในการจัดขาแต่ละขาของ CMOS Image Sensor จะแสดงดังตารางที่ 4.1



รูปที่ 4.4 การจัดวงจรในส่วนของ CMOS Image Sensor

ลำดับขา	ชื่อขา	รายละเอียด
1	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
2	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
3	Ibis_SCLK	ต่อกับขา s_clk ในรูปที่ 5.1
4	Ibis_SData	ต่อกับขา s_data ในรูปที่ 5.1
5	Ibis_SEnable	ต่อกับขา s_en ในรูปที่ 5.1
6	+3.3V	ต่อกับขา +3.3V ในรูปที่ 4.2
7	VDDC	ต่อกับขา VDDC ในรูปที่ 4.3
8	VDDA	ต่อกับขา VDDA ในรูปที่ 4.2
9	GND	ต่อลงกราวด์
10	GND	ต่อลงกราวด์
11	VDDD	ต่อกับขา VDDD ในรูปที่ 4.2
12	GND	ต่อลงกราวด์เพื่อเลือกโหมดการอินเตอร์เฟสแบบ SERIAL
13	VDDA	ต่อกับขา VDDA ในรูปที่ 4.2 เพื่อไบอัส Decoder Stage
14	RC0	ต่อกับขา y_start ในรูปที่ 5.1
15	RC1	ต่อกับขา y_clock ในรูปที่ 5.1
16	RC2	เป็นขาที่แสดงสัญญาณเมื่อจบ line สุดท้าย (last_line)
17	RC3	ต่อกับขา x_loadในรูปที่ 5.1
18	RC4	ต่อกับขา sys_clock ในรูปที่ 5.1
19	RC5	ต่อกับขา pixel_valid ในรูปที่ 5.1
20	RD0	ต่อกับขา ss_start ในรูปที่ 5.1
21	RD1	ต่อกับขา ss_stop ในรูปที่ 5.1
22	RD2	ต่อกับขา time_out ในรูปที่ 5.1
23	RD3	ต่อกับขา reset (SYS_RESET) ในรูปที่ 5.1
24	RD4	เป็นขา Enables electrical black ของ output amplifier
25	RD5	เป็นขา Diagnostic end of scan ของ X register
26	VDDA	ต่อ VDDA ของรูปที่ 4.2 เพื่อเป็นแรงดันอ้างอิงสูงของ DAC

ตารางที่ 4.1 แสดงรายละเอียดการจัดวงจรในส่วนของ CMOS Image Sensor

ลำดับขา	ชื่อขา	รายละเอียด
27	GND	ต่อลงกราวด์เพื่อเป็นแรงดันอ้างอิงต่ำของ DAC
28	28-69	เป็นขา PXL_OUT1 ต่อเป็นอินพุทให้กับ DAC ขา 69
29	-	เป็นขา PXL_OUT2 ซึ่งไม่ใช้งาน
30	VDDA	ต่อ VDDA ของรูปที่ 4.2 เพื่อไบอัส Output Amplifier
31	VDDA	ต่อ VDDA ของรูปที่ 4.2 เพื่อไบอัส Column Amplifiers
32	VDDA	ต่อ VDDA ของรูปที่ 4.2 เพื่อเป็น Pre-charge Bias
33	VDDD	ต่อ VDDD ของรูปที่ 4.2
34	GND	ต่อลงกราวด์
35	GND	ต่อลงกราวด์
36	VDDA	ต่อ VDDA ของรูปที่ 4.2
37	VDDC	ต่อ VDDC ของรูปที่ 4.3
38	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
39	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
40	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
41	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
42	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
43	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
44	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
45	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
46	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
47	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
48	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
49	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
50	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
51	GND	ไม่ได้ใช้งานปล่อยลอยไว้
52	VDDR_R	ต่อกับ C 100 nF ลงกราวด์

ตารางที่ 4.1 (ต่อ) แสดงรายละเอียดการจัดวงจรในส่วนของ CMOS Image Sensor

ลำดับขา	ชื่อขา	รายละเอียด
53	GND	ต่อ R 360 โอห์มลงกราวด์เพื่อเป็นแรงดันอ้างอิงต่ำของADC
54	GND	ต่อลงกราวด์
55	VDDA_ADC	ต่อ VDDA_ADC ของรูปที่ 4.2
56	GND	ต่อลงกราวด์
57	VDDD_ADC	ต่อ VDDD_ADC ของรูปที่ 4.2
58	RD6	เป็นขา ACD clock (40 MHz)
59	RB7	ต่อกับขา DIN_IMAGE7 รูปที่ 5.1
60	RB6	ต่อกับขา DIN_IMAGE6 รูปที่ 5.1
61	RB5	ต่อกับขา DIN_IMAGE5 รูปที่ 5.1
62	RV4	ต่อกับขา DIN_IMAGE4 รูปที่ 5.1
63	RB3	ต่อกับขา DIN_IMAGE3 รูปที่ 5.1
64	RB2	ต่อกับขา DIN_IMAGE2 รูปที่ 5.1
65	RB1	ต่อกับขา DIN_IMAGE1 รูปที่ 5.1
66	RB0	ต่อกับขา DIN_IMAGE0 รูปที่ 5.1
67	-	เป็นขา ACD_OUT<1> ซึ่งไม่ได้ใช้งาน
68	-	เป็นขา ACD_OUT<0> ซึ่งไม่ได้ใช้งาน
69	28-69	เป็นขา ADC Analog input
70	VDDA_ADC	ต่อ VDDA_ADC ของรูปที่ 4.2 เพื่อไบอัส Input Stage ADC
71	VDDD_ADC	ต่อ VDDD_ADC ของรูปที่ 4.2
72	GND	ต่อลงกราวด์
73	GND	ต่อลงกราวด์
74	VDDA_ADC	ต่อ VDDA_ADC ของรูปที่ 4.2
75	VDDA_ADC	ต่อVDDD_ADC ของรูปที่ 4.2 เพื่อเป็นแรงดันอ้างอิงสูงADC
76	VDDR_L	ต่อ VDDR_L ของรูปที่ 4.3
77	VDDH	ต่อ VDDH ของรูปที่ 4.3
78	GND	ไม่ได้ใช้งาน โดยต่อลงกราวด์

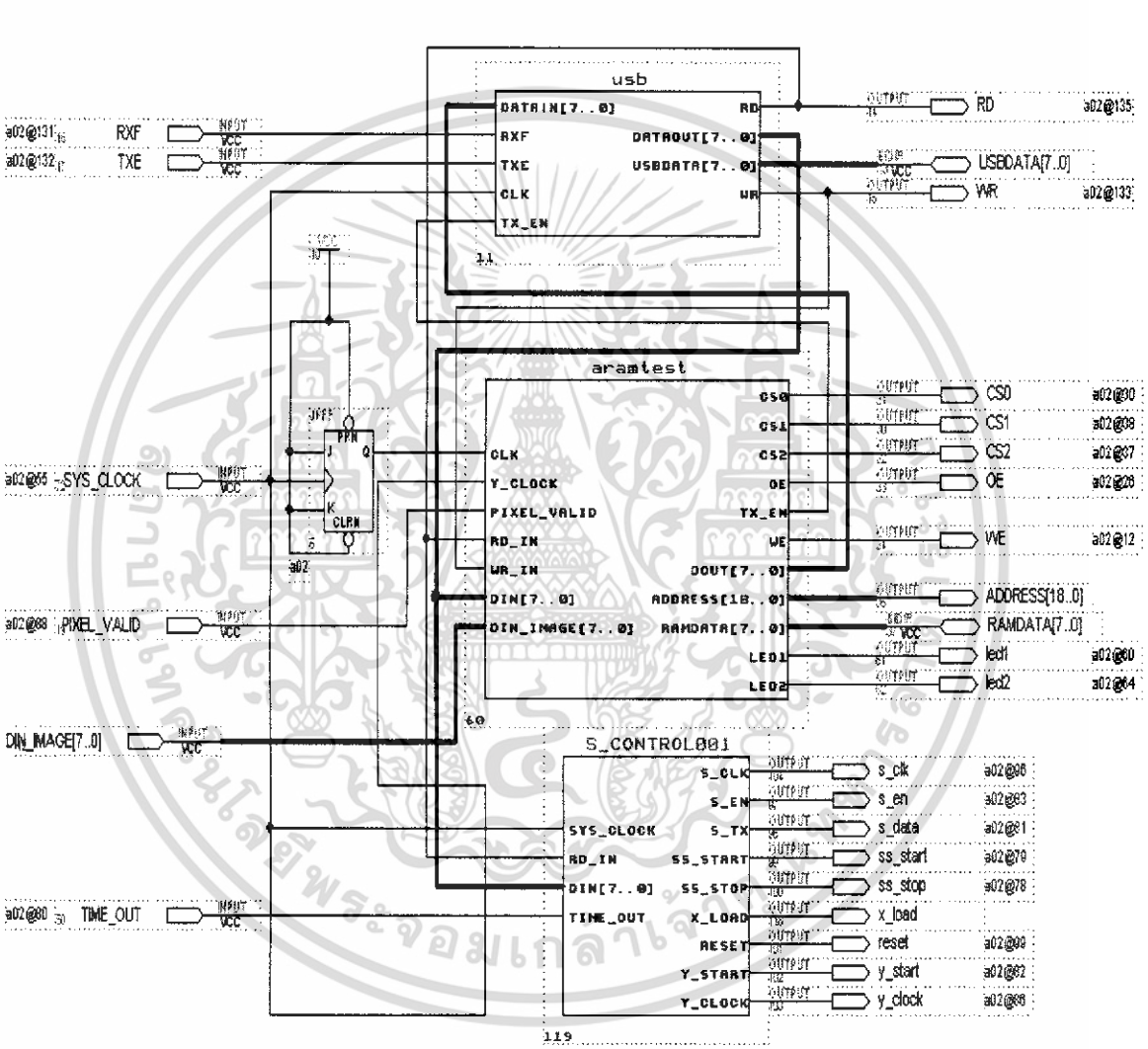
ตารางที่ 4.1 (ต่อ) แสดงรายละเอียดการจัดวงจรในส่วนของ CMOS Image Sensor

ลำดับขา	ชื่อขา	รายละเอียด
79	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
80	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
81	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
82	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
83	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์
84	GND	ไม่ได้ใช้งานโดยต่อลงกราวด์

ตารางที่ 4.1 (ต่อ) แสดงรายละเอียดการจัดวงจรในส่วนของ CMOS Image Sensor

บทที่ 5
การออกแบบโปรแกรมและการทดลอง

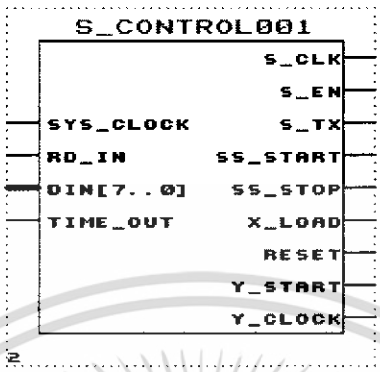
5.1 การออกแบบโปรแกรมในส่วนของ FPGA



รูปที่ 5.1 บล็อกไดอะแกรมแสดงการควบคุมการทำงานในส่วนของ FPGA

จากรูปที่ 5.1 แสดงการทำงานของบล็อกไดอะแกรมซึ่งสามารถเขียนได้โดยโปรแกรม MAX+PLUS II โดยจะมีส่วนหลักๆที่สำคัญอยู่3ส่วนคือ ส่วนส่งสัญญาณควบคุม CMOS Image Sensor ส่วนเก็บข้อมูลลงใน RAM และส่วนส่งข้อมูลผ่าน USB Module เพื่อที่จะใช้โปรแกรม C++ builder ดึงข้อมูลจาก USB Module เพื่อประมวลผลภาพแสดงออกหน้าจอคอมพิวเตอร์

5.1.1 ส่วนของการส่งสัญญาณควบคุมและเซ็ตรีจีสเตอร์ของ CMOS Image Sensor



รูปที่ 5.2 บล็อกไดอะแกรมส่วนของการส่งสัญญาณควบคุมและเซ็ตรีจีสเตอร์

จากรูปที่ 5.2 แสดงการทำงานของการส่งสัญญาณเพื่อเซ็ตรีจีสเตอร์และส่งสัญญาณควบคุม CMOS Image Sensor ซึ่งการเซ็ตรีจีสเตอร์ของ CMOS Image Sensor เป็นการส่งข้อมูลแบบอนุกรมหรือที่เรียกกันว่า Serial 3 wire interface ซึ่งเป็นการส่งสัญญาณ 3 สาย ให้กับ CMOS Image Sensor โดยมีสัญญาณดังกล่าวคือ S_CLK , S_EN และ S_TX (S_DATA)

- S_CLK เป็นสัญญาณนาฬิกาที่ใช้ควบคุมจังหวะเวลาในการส่งข้อมูล
- S_EN เป็นสัญญาณที่แสดงสถานะของการส่งข้อมูล ซึ่งปกติจะเป็นลอกจิกหนึ่ง เมื่อต้องการจะส่งข้อมูล S_EN จะเป็นลอจิกศูนย์เพื่อทำการส่งข้อมูล
- S_DATA เป็นสัญญาณที่ใช้ส่งข้อมูลทีละบิท เพื่อนำข้อมูลที่ส่งไปนี้ไปเซ็ตรีจีสเตอร์ของ CMOS Image Sensor ตามต้องการ

ส่วนด้านสัญญาณควบคุม CMOS Image Sensor นั้นจะต้องส่งสัญญาณควบคุมจากภายนอก 6 สัญญาณ ซึ่งมีดังนี้

-SS_START เป็นพัลส์เดี่ยวเพื่อทริกเริ่มการทำงานของระบบ

-SS_STOP เป็นพัลส์เดี่ยว ทริกต่อเมื่อ TIME_OUT ออกมา เพื่อแสดงว่าจบช่วงเวลา Integration time ที่ได้เซ็ตเอาไว้

-Y_START เป็นพัลส์เดี่ยวทริกเพื่อเริ่มการทำงานส่วนเอาท์พุทเพื่อให้ PIXEL ออกมา

-Y_CLOCK เป็นสัญญาณนาฬิกาที่ใช้ควบคุมจังหวะการปรากฏของ PIXEL

-X_LOAD เป็นสัญญาณในการเลื่อนตำแหน่งของ PIXEL

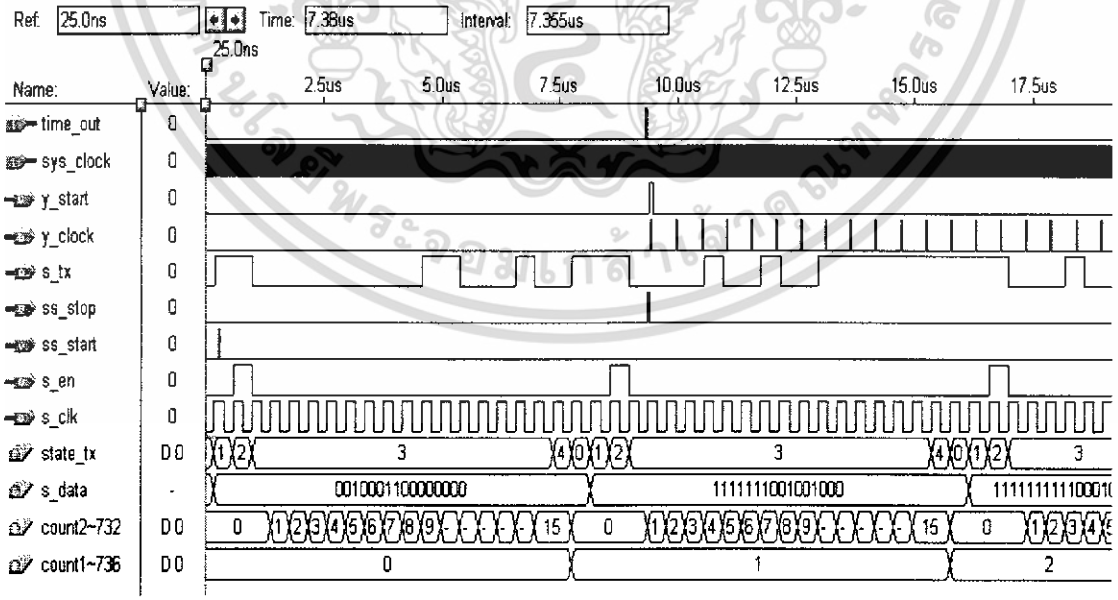
-SYS_CLOCK เป็นสัญญาณนาฬิกาของระบบที่จ่ายให้กับ CMOS Image Sensor

จากการทดลองในส่วนนี้เป็นเหมือนส่วนในการถ่ายภาพ ซึ่งในการเซ็ตรีจีสเตอร์ได้เซ็ตเป็นแบบ Syschonous shutter ซึ่งเป็นโหมดในการถ่ายภาพแบบภาพนิ่ง ซึ่งเวลาในการถ่ายภาพนั้นสามารถเซ็ตได้จากรีจีสเตอร์ 3 (0011) ซึ่งเวลาที่ถ่ายนั้นสามารถคำนวณได้จาก

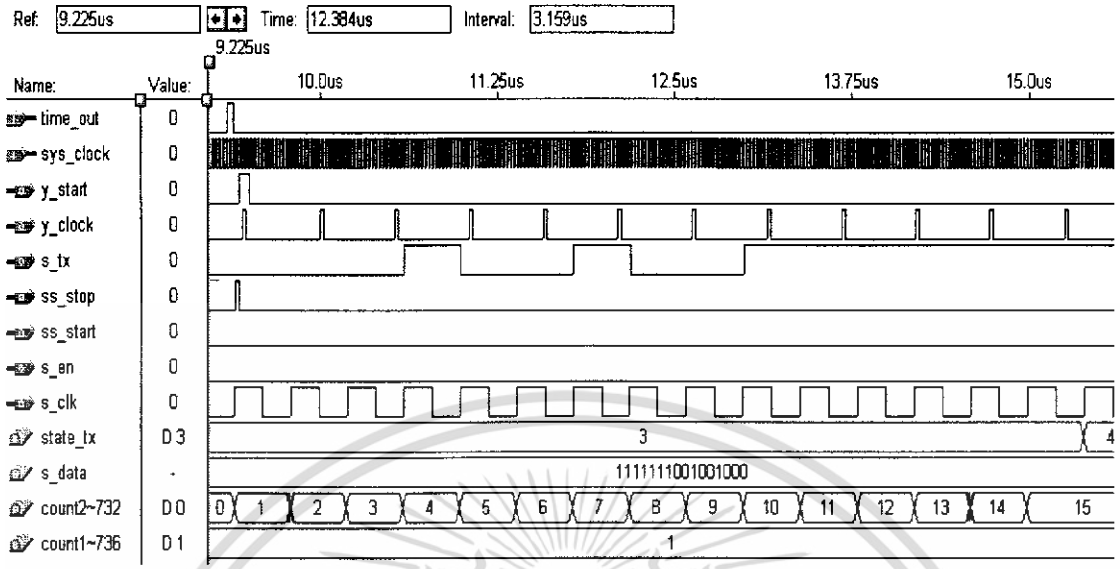
$$T_{INT}\ MAX = 2^{12} * 256\ (\text{maximum granularity}) * (40\ MHZ)^{-1} = 26.2\ ms$$

จากรูปที่ 5.3 แสดงรูปสัญญาณ SS_START เมื่อเริ่มทริกจะเข้าสู่ช่วงเวลา Integration time จนกว่า TIME_OUT จะออกและ SS_STOP ทริกเพื่อหยุดในการถ่ายภาพ โดยรูปที่ 5.4 แสดงถึง Y_START เริ่มทริกเพื่อให้ PIXEL_ OUT ออก โดยสัญญาณจะออกตามจังหวะของ Y_CLOCK ซึ่งสัญญาณที่วัดได้ต่างๆ มีดังนี้

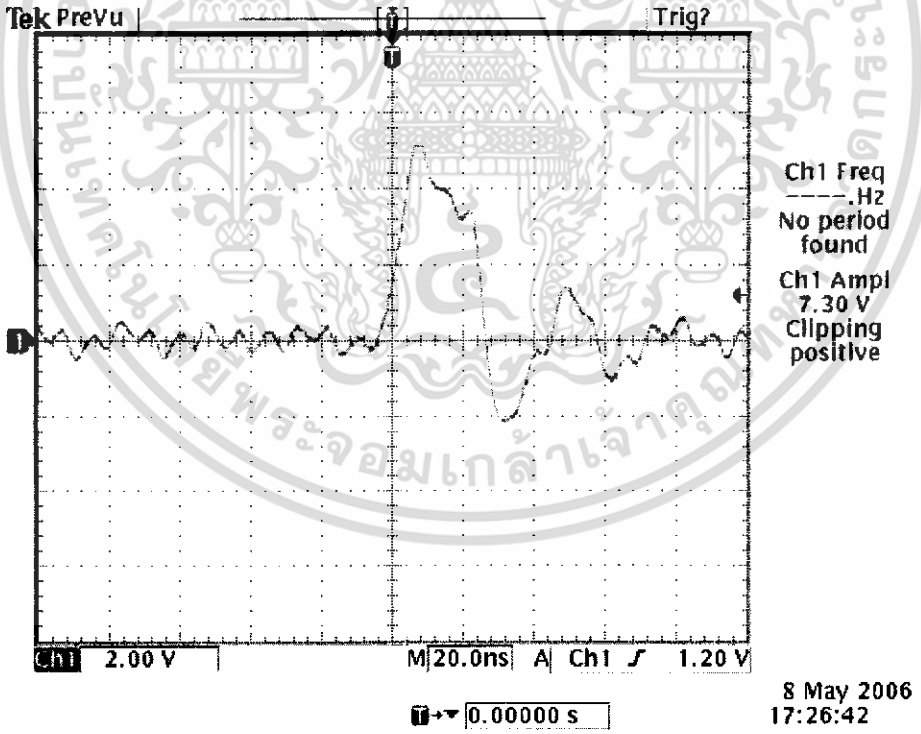
- สัญญาณที่วัดได้ของ SS_START แสดงดังรูปที่ 5.5
- สัญญาณที่วัดได้ของ TIME_OUT แสดงดังรูปที่ 5.6
- สัญญาณที่วัดได้ของ SS_STOP แสดงดังรูปที่ 5.7
- สัญญาณที่วัดได้ของ Y_START แสดงดังรูปที่ 5.8
- สัญญาณที่วัดได้ของ Y_CLOCK แสดงดังรูปที่ 5.9



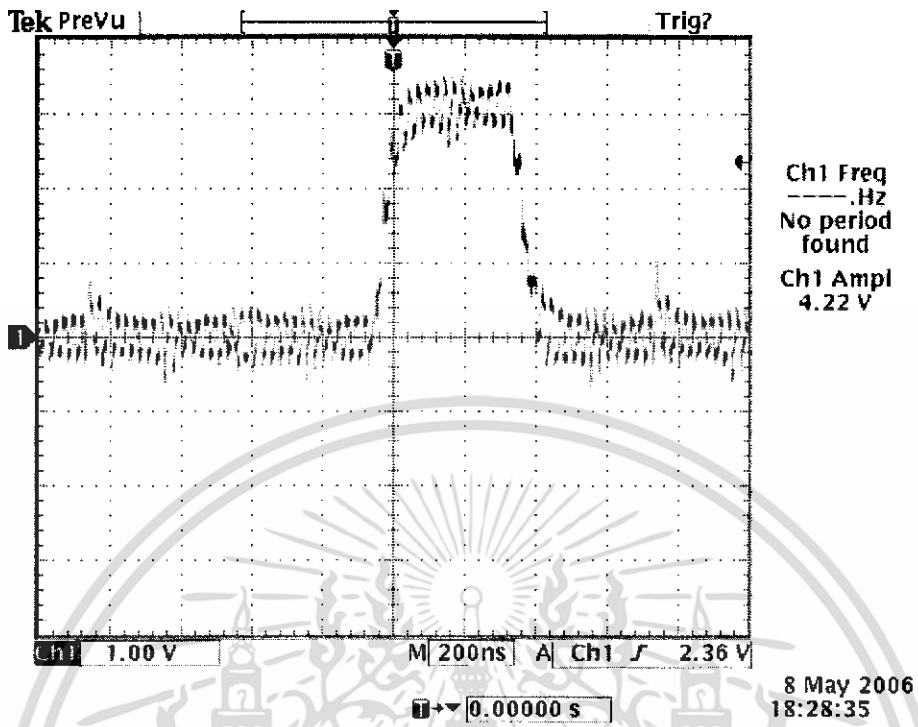
รูปที่ 5.3 สัญญาณทั้งหมดที่ Simulate ได้จากโปรแกรม MAX+PLUS II



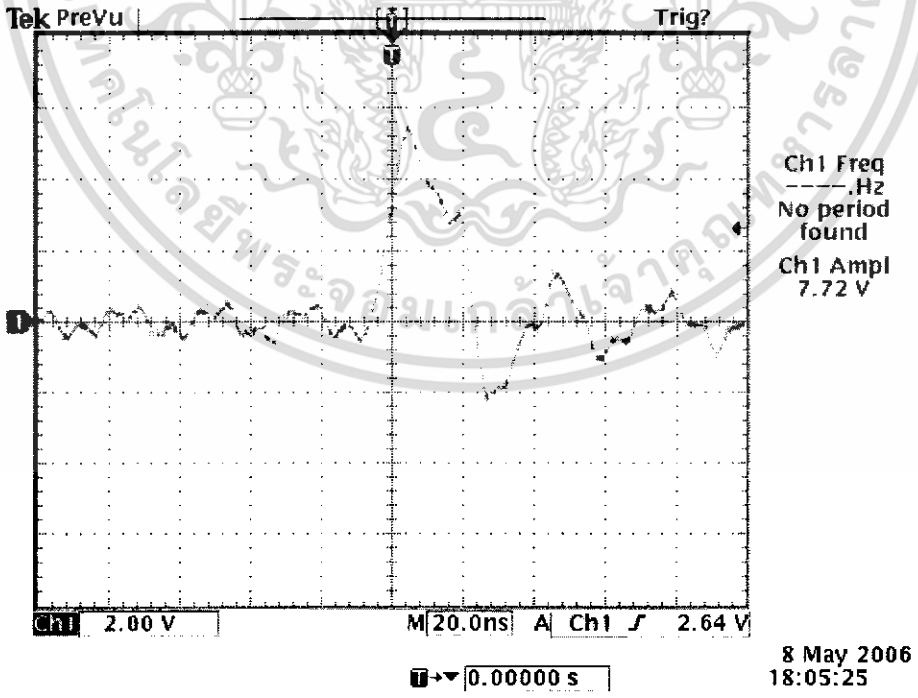
รูปที่ 5.4 สัญญาณหลังจากที่ TIME OUT ปรากฏ



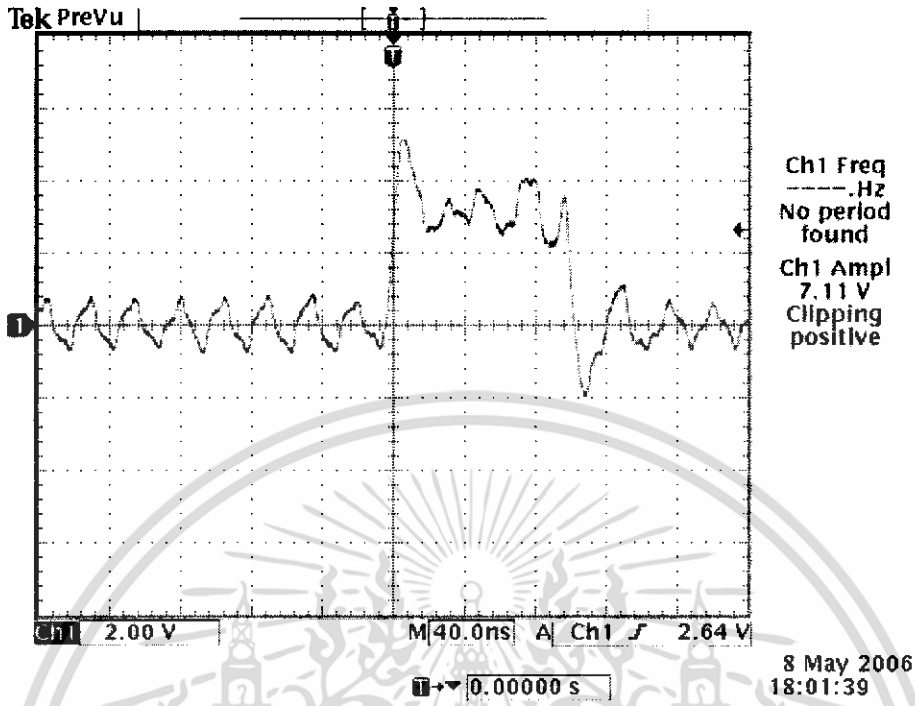
รูปที่ 5.5 สัญญาณ SS_START ที่ได้จากการวัด



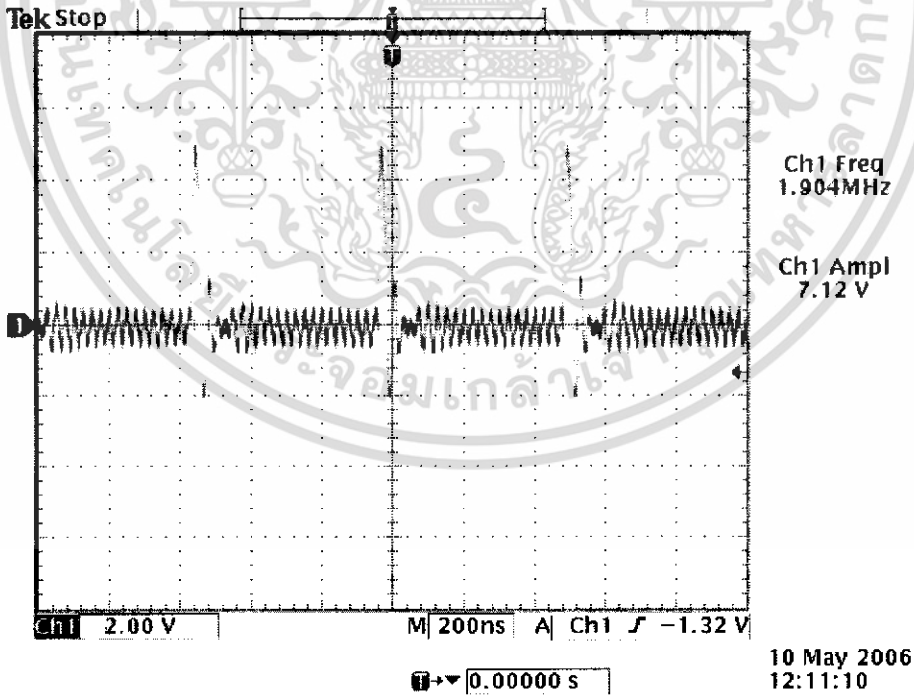
รูปที่ 5.6 สัญญาณ TIME_OUT ที่ได้จากการวัด



รูปที่ 5.7 สัญญาณ SS_STOP ที่ได้จากการวัด

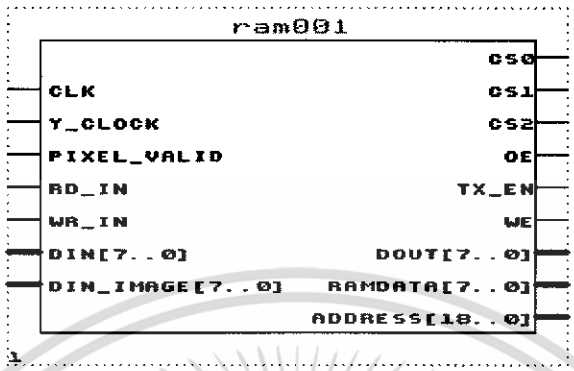


รูปที่ 5.8 สัญญาณ Y_START ที่ได้จากการวัด



รูปที่ 5.9 สัญญาณ Y_CLOCK ที่ได้จากการวัด

5.1.2 ส่วนควบคุมการเก็บค่าข้อมูล PIXEL เข้าไปไว้ใน RAM



รูปที่ 5.10 บล็อกไดอะแกรมส่วนการควบคุมการเก็บค่าข้อมูล PIXEL เข้าไปไว้ใน RAM

จากรูปที่ 5.10 แสดงการทำงานของการเก็บข้อมูลภาพ (PIXEL) ไว้ใน RAM โดยส่วนนี้เป็นส่วนที่เขียนติดต่อการเก็บร่วมกับการส่งด้วยคือต้องส่งไปไว้ที่ USB MODULE ซึ่งจะต้องเขียนโปรแกรมร่วมกับส่วนของ USB MODULE โดยที่จะใช้โปรแกรม C++ builder เป็นตัวควบคุมการเขียนและการอ่านซึ่งจะส่งสัญญาณผ่าน DIN ซึ่งเป็นข้อมูลขนาด 8 บิต เพื่อให้มาเปรียบเทียบว่าจะเขียนลงใน RAM หรืออ่านไปเก็บไว้ที่ USB MODULE โดยการทำงานในแต่ละขาของบล็อกมีดังนี้

-CLK เป็นสัญญาณนาฬิกาของระบบโดยจะทำการเขียนหรืออ่านข้อมูลตามจังหวะสัญญาณนี้

-Y_CLOCK เป็นสัญญาณที่ควบคุมการออกมาของ PIXEL

-PIXEL_VALID เป็นสัญญาณลอจิก 1 เมื่อมี PIXEL ออกมา

-RD_IN เป็นสัญญาณเอาท์พุทขนาด 1 บิท เพื่อกระตุ้นบัฟเฟอร์ของ USB MODULE ในเวลาอ่านข้อมูล

-WR_IN เป็นสัญญาณเอาท์พุทขนาด 1 บิท เพื่อกระตุ้นบัฟเฟอร์ของ USB MODULE ในเวลาเขียนข้อมูล

-DIN เป็นสัญญาณอินพุตขนาด 8 บิท ส่งมาจาก USB MODULE เพื่อควบคุมการอ่านเขียนของ RAM

-DIN_IMAGE เป็นสัญญาณอินพุตขนาด 8 บิท ที่ส่งสัญญาณภาพ (PIXEL) เพื่อจะนำมาเก็บไว้ใน RAM

-CS เป็นขา Chip Select ของ RAM ไว้สำหรับเลือกว่าจะเก็บไว้ใน RAM ตัวไหน โดยมี 3 ตัวให้เลือก

-OE เป็นขา OUTPUT ENABLE ทำงานที่ลอจิก 0

-TX_EN เป็นขาแสดงสภาวะการทำงานของ RAM

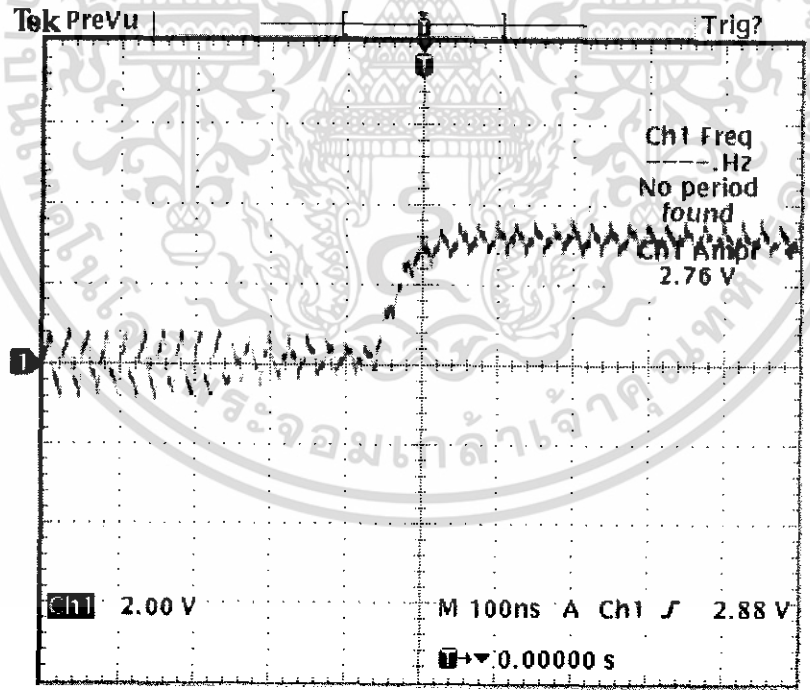
-WE เป็นขา WRITE ENABLE เมื่อจะทำการเขียนข้อมูลลง RAM ขานี้จะต้องเซ็ตเป็นลอกจิก 0

-DOUT เป็นเอาท์พุทขนาด 8 บิท เพื่อส่งไปที่ USB MODULE เพื่อส่งเข้า COMPUTER ต่อไป

-RAMDATA เป็นข้อมูลที่ RAM สามารถเก็บได้จาก PIXEL

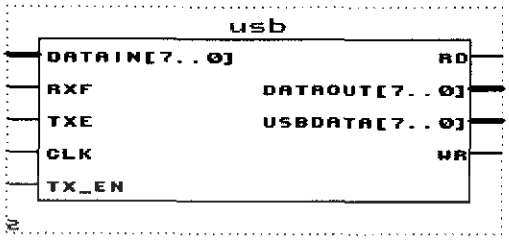
-ADDRESS เป็นตำแหน่งที่ RAM เก็บข้อมูล

จำนวน PIXEL ของภาพที่สามารถเก็บได้สามารถเซ็ตได้จากรีจิสเตอร์เพื่อได้ขนาดภาพที่เราต้องการ ซึ่ง RAM จะเก็บภาพจนหมด ซึ่งเมื่อจำนวน PIXEL ไหลออกมาจนหมด LAST_LINE ก็จะปรากฏออกมา ซึ่งจากการวัดสามารถแสดงได้ดังรูปที่ 5.11



รูปที่ 5.11 สัญญาณ LAST_LINE ที่ได้จากการวัด

5.1.3 ส่วนการนำค่าข้อมูล PIXEL จาก RAM ส่งผ่าน USB MODULE



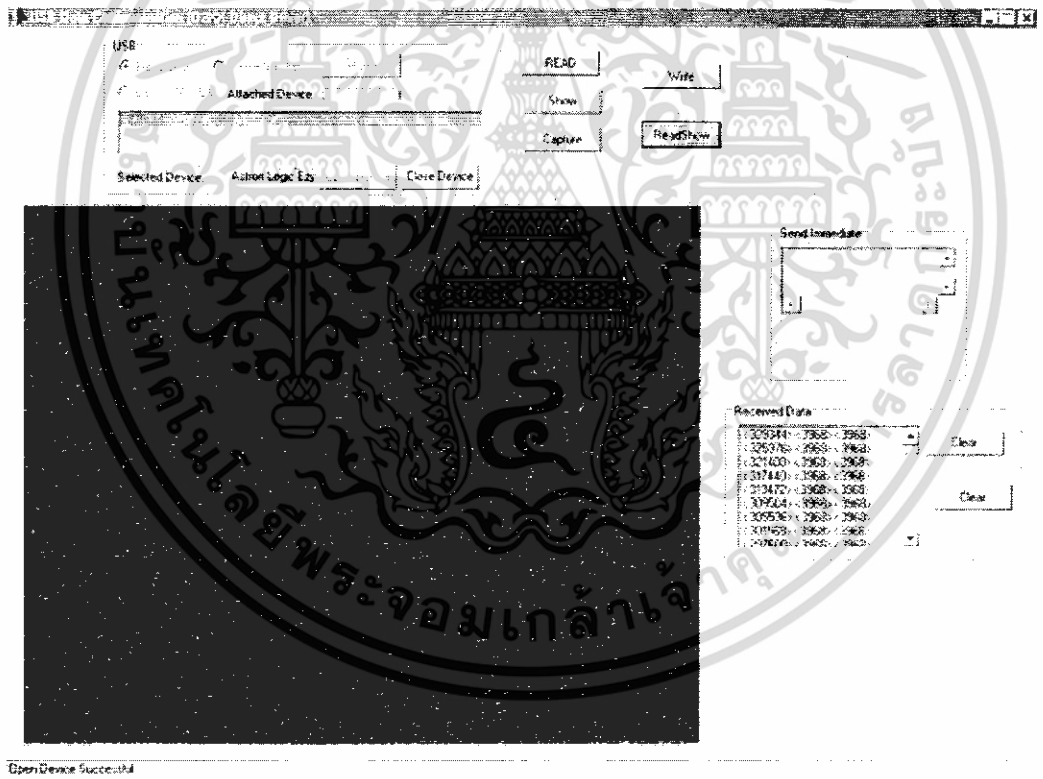
รูปที่ 5.12 บล็อกไดอะแกรมส่วน USB MODULE

จากรูปที่ 5.12 แสดงการทำงานของบล็อกไดอะแกรมส่วนนี้สามารถที่จะอ่านและเขียนเนื่องจากเขียนโปรแกรมแบบควบคุม USB MODULE แบบ 2 ทิศทาง (Bi-Directional) โดยการทำงานในแต่ละขาของบล็อกมีดังนี้

-CLK	เป็นสัญญาณนาฬิกาเพื่อกระตุ้นการทำงานของวงจรทั้งหมด
-DATAIN	เป็นข้อมูลของ PIXEL ที่ได้รับจาก RAM
-RXF#	เป็นอินพุทขนาด 1 บิท จะได้รับสัญญาณจาก USB MODULE เมื่อ RXF# เป็นลอจิก 0 แสดงว่า USB MODULE พร้อมจะให้อ่านข้อมูลจาก FIFO แล้ว
-TXE#	เป็นอินพุทขนาด 1 บิท จะได้รับสัญญาณจาก USB MODULE เมื่อ RXF เป็นลอจิก 0 แสดงว่า USB MODULE พร้อมที่จะรับข้อมูลแล้ว
-TX_EN	เป็นขาแสดงสภาวะการทำงานของ RAM
-RD	เป็นสัญญาณเอาท์พุทขนาด 1 บิท เพื่อกระตุ้นบัฟเฟอร์ของ USB MODULE ในเวลาอ่านข้อมูล
-WR	เป็นสัญญาณเอาท์พุทขนาด 1 บิท เพื่อกระตุ้นบัฟเฟอร์ของ USB MODULE ในเวลาเขียนข้อมูล
-DATAOUT	เป็นข้อมูลขนาด 8 บิทที่ส่งมาจาก COMPUTER เพื่อส่งไปใน RAM เพื่อใช้ควบคุมการอ่านเขียนของ RAM
-USBDATA	เป็นข้อมูลภาพ PIXEL ที่อ่านออกมาจาก RAM เพื่อส่งให้กับ COMPUTER เพื่อนำไปประมวลภาพต่อไป

5.2 โปรแกรมควบคุมในส่วนของ C++ builder

การออกแบบในส่วนของโปรแกรม C++ Builder โดยทำการสร้างแอปพลิเคชั่นขึ้นมา ซึ่งจะใช้ควบคุมการทำงานของระบบ และจะควบคุมการทำงานของ FPGA ผ่าน USB Module โดยทำการส่งค่าแต่ละค่าเข้าไปให้ FPGA เพื่อกำหนดการทำงานของ FPGA ในลักษณะต่างๆ ที่แต่ละค่านั้นๆส่งไป คือ เซ็ตรีจิสเตอร์ ควบคุมการกำหนดภาพแล้วทำการเก็บค่าข้อมูลภาพไว้ใน RAM และส่งผ่านค่าข้อมูลภาพจาก RAM มายังคอมพิวเตอร์ โดยที่การแสดงผลภาพก็จะใช้โปรแกรม C++ Builder ด้วยเช่นกัน ซึ่งแสดงดังรูปที่ 5.13 โดยการส่งค่าไปควบคุม FPGA และการแสดงผลภาพนั้นจะทำได้โดยการคลิกปุ่มต่างๆ ที่ได้ทำการออกแบบไว้



รูปที่ 5.13 แสดงแอปพลิเคชั่นที่สร้างขึ้นโดยโปรแกรม C++ Builder

บทที่ 6
สรุปและวิจารณ์ผลการทดลอง

6.1 สรุปผลการทดลอง

จากการทดลอง เมื่อทำการป้อนสัญญาณเซ็ตรีจีสเตอร์และสัญญาณควบคุมให้ CMOS Image Sensor แล้วทำการวัดสัญญาณควบคุมที่จุดต่างๆ โดยจากสัญญาณควบคุมที่วัดได้นั้นจะมีลักษณะไม่ค่อยเป็นไปตามที่เราได้ทำการออกแบบไว้ในโปรแกรมซักเท่าไหร่ คือจะมีสัญญาณรบกวน (Noise) ค่อนข้างมากเมื่อเทียบกับการ Simulate ในโปรแกรม MAX+PLUS II ซึ่งได้รูปแบบของสัญญาณที่ถูกต้องและเป็นไปตามที่เราออกแบบไว้

ในการทดลองหลังจากทำการเซ็ตรีจิสเตอร์ให้กับ CMOS Image Sensor แล้ว ในขั้นตอนของการป้อนสัญญาณควบคุมเมื่อทำการป้อนสัญญาณ SS_START ให้กับ CMOS Image Sensor จะทำให้ได้สัญญาณ TIME_OUT ออกมาที่เอาท์พุทหนึ่งของ CMOS Image Sensor เป็นลักษณะพัลส์เดียวในช่วงเวลาหนึ่ง ซึ่งในการเซ็ตให้ CMOS Image Sensor ทำงานในโหมดของ Synchronous Shutter นั้นจะต้องป้อนสัญญาณ SS_STOP หลังจากสัญญาณ TIME_OUT เกิดขึ้น นั่นคือสัญญาณ SS_STOP จะปรากฏก็ต่อเมื่อมีสัญญาณ TIME_OUT จาก CMOS Image Sensor จ่ายออกมา

ซึ่งเมื่อ TIME_OUT ถูกจ่ายออกมาและทำการป้อนสัญญาณ SS_STOP เพื่อกำหนดโหมด Synchronous Shutter แล้ว ต่อไปเป็นการป้อนสัญญาณ Y_START และ Y_CLOCK เพื่อเป็นการเริ่มการอ่านค่าข้อมูลพิกเซลออกมาและการเริ่มของค่าข้อมูลภาพในแต่ละเส้นตามลำดับ เมื่อทำการวัด PIXEL_OUT ซึ่งเป็นสัญญาณอนาล็อกเอาท์พุท ปรากฏว่าจะเกิดขึ้นอย่างรวดเร็วโดยไม่สามารถใช้ออสซิลโลสโคปจับได้ นั่นอาจเป็นเพราะมีสัญญาณปรากฏมาชั่วขณะหนึ่งแล้วก็หายไป เมื่อลองทำการการตรวจสอบวัดสัญญาณ LAST_LINE ซึ่งเป็นสัญญาณดิจิตอลเอาท์พุทและจะปรากฏเมื่อค่าข้อมูลภาพเส้นสุดท้ายถูกส่งออกมา ซึ่งผลการวัดก็ตรวจพบสัญญาณ LAST_LINE เช่นกัน แต่เมื่อทำการวัด PIXEL_VALID ซึ่งเป็นสัญญาณดิจิตอลเอาท์พุทและจะถูกจ่ายออกมาเป็นลอจิก “1” ระหว่างการส่งค่าข้อมูลภาพออกมา แต่ปรากฏว่าไม่สามารถใช้ออสซิลโลสโคปจับได้ ซึ่งในการทำงานของ FPGA ในส่วนของการรับค่าข้อมูลภาพและนำไปจัดเก็บไว้ใน RAM จะต้องอาศัยสัญญาณ PIXEL_VALID เป็นตัวอ้างอิงในการเริ่มการจัดเก็บ เมื่อนำค่าข้อมูลภาพที่เก็บไว้ใน RAM มาแสดงผล ปรากฏว่าภาพที่ได้มีลักษณะเป็นลายจุดขาวดำ ไม่เป็นรูปร่างดังรูปที่ 5.13

6.2 ปัญหาที่เกิดขึ้น

- เกิดสัญญาณรบกวนในระบบค่อนข้างมาก
- การตอบสนองของ CMOS Image Sensor ในบางเอาท์พุทไม่เป็นไปตามปกติและไม่สามารถตรวจเช็คได้
- การเซ็ตค่ารีจิสเตอร์ภายในให้ CMOS Image Sensor ไม่สามารถตรวจสอบได้ในทางปฏิบัติ

6.3 แนวทางแก้ไข

- ออกแบบระบบแหล่งจ่ายไฟเลี้ยงให้มีความเสถียรภาพที่ดีขึ้นเพื่อลดสัญญาณรบกวน
- ออกแบบการจัดวางอุปกรณ์ให้ดีขึ้นเพื่อลดสัญญาณรบกวน
- ตรวจเช็ค CMOS Image Sensor ว่ายังใช้งานได้ปกติอยู่หรือเปล่า

บรรณานุกรม

1.การประยุกต์ใช้ FPGA,“ปฏิบัติการอิเล็กทรอนิกส์3”,ภาควิชาอิเล็กทรอนิกส์คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, กรุงเทพมหานคร, 2546

2. นุกูล กระจาย, “ การเขียนโปรแกรมแบบวิชวลด้วย C++Builder5 ”, สุวีริยาสาสน์, 392 หน้า, 2544

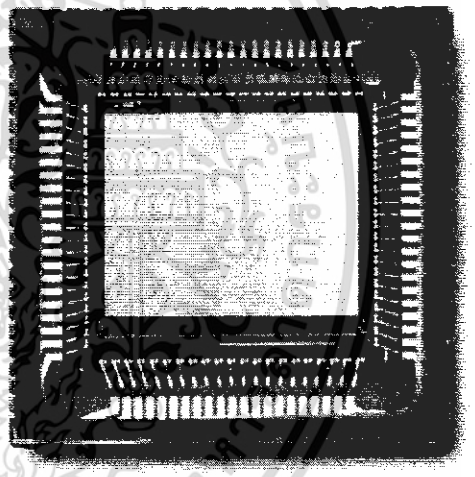
3. ชำนาญ ปัญญาใส, “ ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล ”, ซีเอ็ดยูเคชั่น, 432 หน้า, 2547

4. วรเทพ ไพบูลย์รัตนากร, “ สัมผัสโลก USB ด้วย Ezy USB Module ”, บริษัท แอสตรอน ลอจิก รีเสิร์ชแอนด์ดิวิลอปเมนต์ จำกัด, 440 หน้า, 2546

ภาคผนวก

IBIS5-A-1300

1.3M Pixel
Dual Shutter Mode
CMOS Image Sensor



Datasheet

1 Introduction

Overview

The IBIS5-A-1300 is a solid state CMOS image sensor that integrates the functionality of complete analog image acquisition, digitizer and digital signal processing system on a single chip. This 1.3-mega pixel (1280 x 1024) CMOS active pixel sensor dedicated to industrial vision features both rolling and snapshot shutter. Full frame readout time is 36 ms (max. 27.5 fps), and readout speed can be boosted by windowed region of interest (ROI) readout. High dynamic range scenes can be captured using the double and multiples slope functionality. The sensor is available in a Monochrome version or Bayer (RGB) patterned color filter array.

User programmable row and column start/stop positions allow windowing down to 2x1 pixel window for digital zoom. Sub sampling reduces resolution while maintaining the constant field of view and an increased frame rate. The analog video output of the pixel array is processed by an on-chip analog signal pipeline. Double Sampling (DS) eliminates the fixed pattern noise. The programmable gain and offset amplifier maps the signal swing to the ADC input range. A 10-bit ADC converts the analog data to a 10-bit digital word stream. The sensor uses a 3-wire Serial-Parallel (SPI) interface or a 16-bit parallel interface. It operates with a 3.3V power supply and requires only one master clock for operation up to 40 MHz. It is housed in an 84-pin ceramic LCC package.

The IBIS5-A-1300 is designed taking into consideration interfacing requirements to standard video encoders. In addition to the 10-bit pixel data stream, the sensor outputs the valid frame, line and pixel sync signals needed for encoding.

This datasheet allows the user to develop a camera system based on the described timing and interfacing.

Key features

The main features of the image sensor are identified as:

- SXGA resolution: 1280 by 1024 pixels.
- 6.7 µm high fill factor square pixels (based on the high-fill factor active pixel sensor technology of FillFactory (US patent No. 6,225,670 and others)).
- Peak QE x FF of 30%.
- Optical format: 2/3" (8.6mm x 6.9mm).
- Pixel rate of 40 MHz using a 40 MHz system clock.
- Optical dynamic range: 64dB (1600:1) in single slope operation and 80…100dB in multiple slope operation.
- On-chip 10 bit, 40Msamples/s ADC.
- Programmable gain and offset output amplifier.
- Both rolling curtain shutter and synchronous shutter.
- Random programmable windowing and sub-sampling modes.
- Low fixed pattern noise (<0.5% RMS).
- On-chip timing and control logic sequencer.
- Processing is done in a CMOS 0.35 µm triple metal process.

Part number

Part number	Package	Monochrome/ color die	Glass lid
IBIS5-A-1300-M-1 CYII5SM1300AA-HBC (Preliminary)	84 pins JLCC package*	Monochrome	Monochrome**
IBIS5-A-1300-M-2 CYII5SM1300AA-QBC (Preliminary)	84 pins LCC package	Monochrome	Monochrome
IBIS5-A-1300-C-1 CYII5SC1300AA-HAC (Preliminary)	84 pins JLCC package	Color	Color***
IBIS5-A-1300-C-2 CYII5SC1300AA-QAC (Preliminary)	84 pins LCC package	Color	Color

* JLCC package for use in evaluation kits only.
** D263 is used as monochrome glass lid (see Figure 36 for spectral transmittance).
*** S8612 is used as color glass lid (see Figure 37 for spectral transmittance).
Other packaging combinations are available upon special request.

2 Specifications

2.1 Pixel characteristics

Table 1: Pixel characteristics

Parameter	Specification	Remarks
Pixel architecture	*4-transistor active pixel sensor*	Allows for both rolling and synchronous (snapshot) shutter.
Pixel size	*6.7 µm x 6.7 µm*	The resolution and pixel size results in a 2/3" optical format.
Resolution	*1280 x 1024*	
Pixel rate	*40 MHz*	Using a 40 MHz system clock.
Shutter types	*- Rolling* *- Snapshot*	- Continuous imaging. - Triggered synchronous shutter with integration and readout separate in time.
Full frame rate	*27.5 fps*	Depending on shutter type and integration time.

2.2 Electro-optical specifications

2.2.1 Overview

Table 2: Electro-optical specifications

Parameter	Specification	Remarks
FPN (on chip corrected)	*< 0.2% RMS*	Synchronous (snapshot) shutter. Rolling curtain shutter.
PRNU	*<10% p-p*	*2% RMS.*
Conversion gain	*17.6 uV/electron*	@ output.
Output signal amplitude	*1.1V* *1.8V*	Unity gain. Maximum output signal amplitude.
Saturation charge	*62.500 e-*	
Sensitivity	*592 V.m2/W.s*	Average white light.
	3.29 V/lux.s	Visible band only (180 lx = 1 W/m2).
	8.46 V/lux.s	Visible + NIR (70 lx = 1 W/m2).
Peak QE * FF Peak SR * FF	*>30%* *0.16 A/W*	Average QE*FF = 25-30%. Average SR*FF = 0.12 A/W. See spectral response curve.
Dark current *(@ RT)*	*7.22 mV/s* *or ~ 410 e-/s*	Auto saturation time in the order of 150s.
Noise electrons	*< 40 e-*	Synchronous shutter. Rolling curtain shutter.
S/N ratio	*1600:1 (64 dB)*	Synchronous shutter. Rolling curtain shutter.

Parameter	Specification	Remarks
Spectral sensitivity range	*400 – 1000 nm*	
Parasitic sensitivity	*< 1%*	I.e. sensitivity of the storage node during read out (after integration).
Optical cross talk	*8%*	Cross talk to the nearest neighbor.
Power dissipation	*175 mWatt*	Typical.

2.2.2 Spectral response curve

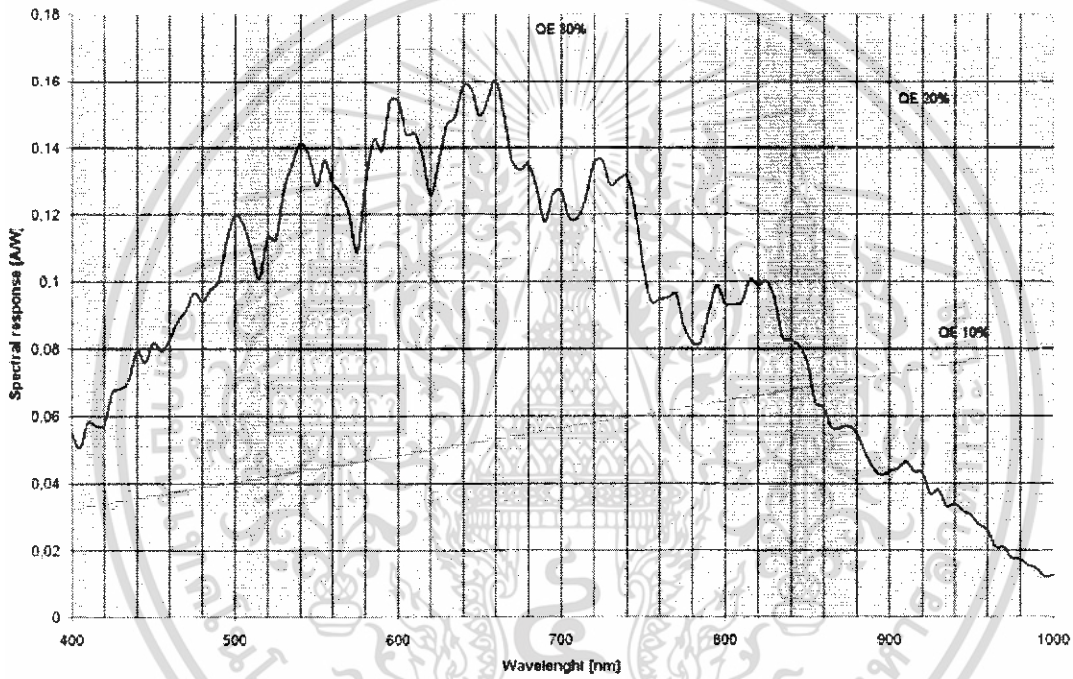


Figure 1: Spectral response curve

Figure 1 shows the spectral response characteristic. The curve is measured directly on the pixels. It includes effects of non-sensitive areas in the pixel, e.g. interconnection lines. The sensor is light sensitive between 400 and 1000 nm. The peak QE * FF is approximately 30% between 500 and 700 nm. In view of a fill factor of 50%, the QE is thus larger than 60% between 500 and 700 nm.

2.2.3 Photo-voltaic response curve

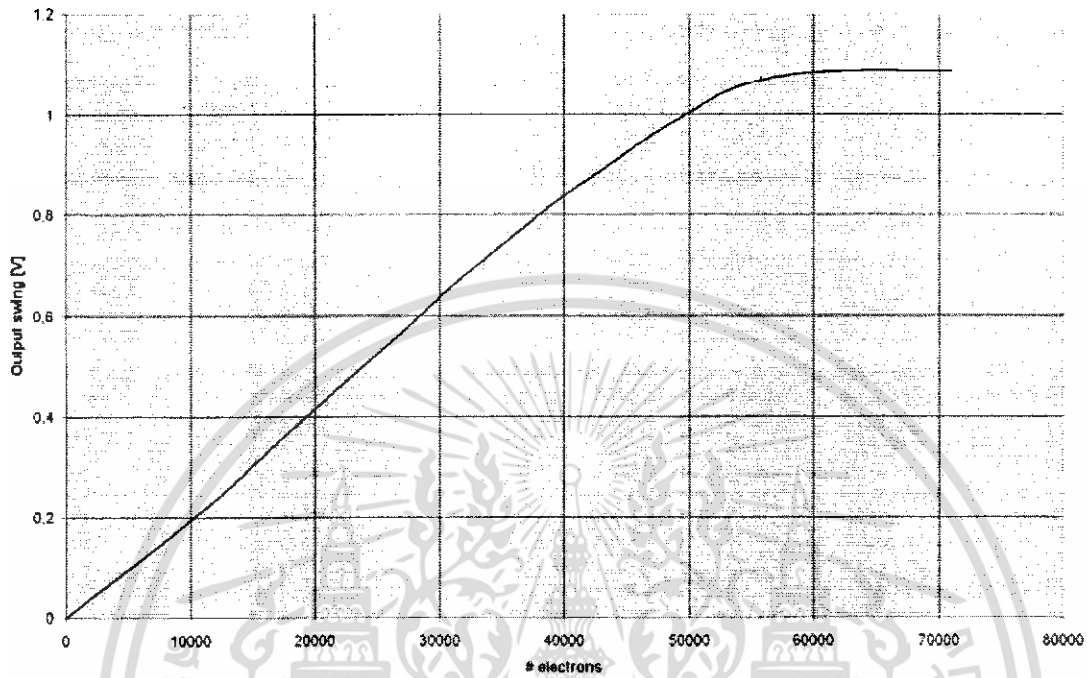


Figure 2: photo-voltaic response curve

Figure 2 shows the pixel response curve in linear response mode. This curve is the relation between the electrons detected in the pixel and the output signal. The voltage to electrons conversion gain of the pixel is 17.6 µV/electron.

2.3 Features and general specifications

Table 3: Features and general specifications

Feature	Specification/Description
Electronic shutter types	1. Rolling curtain shutter. 2. Synchronous (snapshot) shutter.
Windowing (ROI)	Implemented as scanning of lines/columns from an uploaded position.
Sub-sampling modes: • X-direction • Y-direction	1:2 sub-sampling. Sub-sampling patterns: a. XXOOXXOO (for Bayer pattern color filter) b. OOXXOOXX (for Bayer pattern color filter) c. XOXOXOXO d. OXOXOXOX Identical sub-sample patterns as X-direction.

Feature	Specification/Description
Extended dynamic range	In rolling shutter: Normal (1) or double (2) slope. In Synchronous shutter: 1, 2, 3 or 4 slopes.
Digital output	10 bit ADC @ 40 MSamples/s.
Programmable gain range	x1 to x12, in 16 steps of approx. 1.5 dB using 4-bits programming.
Programmable offset	128 steps (7 bit).
Supply voltage VDD	Image core supply: Range from 3.3 V to 4.5 V Analog supply: Nominal 3.3 V Digital: Nominal 3.3 V
Logic levels	3.3 V (Digital supply).
Operational temperature range	0°C to 60°C, with degradation of dark current.
Die size (with scribe lines)	10.1 mm by 9.3 mm (x by y).
Package	84 pins LCC.

2.4 Electrical specifications

2.4.1 Absolute maximum ratings

Table 4: Absolute maximum ratings

Symbol	Parameter	Value	Unit
VDD	DC supply voltage	-0.5 to 4.5	V
V_{IN}	DC input voltage	-0.5 to 3.8	V
V_{OUT}	DC output voltage	-0.5 to 3.8	V
I_{IO}	DC current drain per pin; any single input or output.	± 50	mA
T_L	Lead temperature (5 seconds soldering).	350	°C

- Absolute Ratings are those values beyond which damage to the device may occur.
- VDD = VDDD = VDDA (VDDD is supply to digital circuit, VDDA to analog circuit).

2.4.2 Recommended operating conditions

Table 5: Recommended operation conditions

Symbol	Parameter	Min	Typ	Max	Unit
VDDH	Voltage on HOLD switches.	+3.3	+4.5	+4.5	V
VDDR_LEFT	Highest reset voltage.	+3.3	+4.5	+4.5	V
VDDC	Pixel core voltage.	+2.5	+3.0	+3.3	V
VDDA	Analog supply voltage of the image core.	+3.0	+3.3	+3.6	V
VDDD	Digital supply voltage of the image core.	+3.0	+3.3	+3.6	V

Symbol	Parameter	Min	Typ	Max	Unit
GNDA	Analog ground	-0.5	0	+0.5	V
GNDD	Digital ground	-0.5	0	+0.5	V
GND_AB	Anti-blooming ground.	-0.5	0	+0.5	V
T_A	Commercial operating temperature.	0	30	60	°C

- All parameters are characterized for DC conditions after thermal equilibrium has been established.
- Unused inputs must always be tied to an appropriate logic level, e.g. either VDD or GND.
- This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however it is recommended that normal precautions be taken to avoid application of any voltages higher than the maximum rated voltages to this high impedance circuit.

2.4.3 DC electrical characteristics

Table 6: DC electrical characteristics

Symbol	Characteristic	Condition	Min	Max	Unit
V_{IH}	Input high voltage		2.1		V
V_{IL}	Input low voltage			0.6	V
I_{IN}	Input leakage current	V_{IN} = VDD or GND	-10	+10	µA
V_{OH}	Output high voltage	VDD=min; I_{OH}= -100mA	2.2		V
V_{OL}	Output low voltage	VDD=min; I_{OH}= 100mA		0.5	V
I_{DD}	Maximum operating current	System clock <= 40MHz	40	60	mA

stop the synchronous shutter.

3.10 Sequencer

Figure 7 showed a number of control signals that are needed to operate the sensor in a particular sub-sampling mode, with a certain integration time, output amplifier gain, etc. Most of these signals are generated on chip by the sequencer that uses only a few control signals. These control signals should be generated by the external system:

- SYS_CLOCK (X-clock), which will define the pixel rate,
- Y_START pulse, which will indicate the start of a new frame,
- Y_CLOCK, which will select a new row and will start the row blanking sequence, including the synchronization and loading of the X-register.
- SS_START and SS_STOP to control the integration period in snapshot shutter mode.

The relative position of the pulses will be determined by a number of data bits that are uploaded in internal registers through the serial or parallel interface.

3.10.1 Internal registers

Table 18 shows a list of the internal registers with a short description. In the next section, the registers are explained in more detail.

Table 18: Internal registers

Register	Bit	Name	Description
0 (0000)	11:0	SEQUENCER register	*Default value <11:0>:"000011000100"*
	0	SHUTTER_TYPE	1 = rolling shutter 0 = synchronous shutter
	1	FRAME_CAL_MODE	0 = fast 1 = slow
	2	LINE_CAL_MODE	0 = fast 1 = slow
	3	CONT_CHARGE	1 = "Continuous" precharge enabled.
	4	GRAN_X_SEQ_LSB	Granularity of the X sequencer clock
	5	GRAN_X_SEQ_MSB	
	6	GRAN_SS_SEQ_LSB	Granularity of the SS sequencer clock
	7	GRAN_SS_SEQ_MSB	
	8	KNEEPOINT_LSB	Sets reset voltage for multiple slope operation.
	9	KNEEPOINT_MSB	
	10	KNEEPOINT_ENABLE	1 = Enables multiple slope operation in synchronous shutter mode
	11	VDDR_RIGHT_EXT	1 = Disables circuit that generates VDDR_RIGHT voltage so external voltage can be applied.
1 (0001)	11:0	NROF_PIXELS	Number of pixels to count (maximum 1280/2).

Register	Bit	Name	Description
			Default value <11:0>:"001001111111"
2 (0010)	11:0	NROF_LINES	Number of lines to count. *Default value <11:0>:"001111111111"*
3 (0011)	11:0	INT_TIME	Integration time. *Default value <11:0>:"111111111111"*
4 (0100)	10:0	X_REG	X start position (maximum 1280/2). *Default value <10:0>:"00000000000"*
5 (0101)	10:0	YL_REG	Y-left start position. *Default value <10:0>:"00000000000"*
6 (0110)	10:0	YR_REG	Y-right start position. *Default value <10:0>:"00000000000"*
7 (0111)	7:0	IMAGE CORE register	*Default value <7:0>:"00000000"*
	0	TEST_EVEN	Test even columns.
	1	TEST_ODD	Test odd columns.
	2	X_SUBSAMPLE	Enable sub-sampling in X-direction.
	3	X_SWAP12	Swap columns 1-2, 5-6, …
	4	X_SWAP30	Swap columns 3-4, 7-8, …
	5	Y_SUBSAMPLE	Enable sub-sampling in Y-direction.
	6	Y_SWAP12	Swap rows 1-2, 5-6, …
	7	Y_SWAP30	Swap rows 3-4, 7-8, …
8 (1000)	6:0	AMPLIFIER register	*Default value <6:0>:"1010000"*
	0	GAIN<0>	Output amplifier gain setting.
	1	GAIN<1>	
	2	GAIN<2>	
	3	GAIN<3>	
	4	UNITY	1 = Amplifier in unity gain mode.
	5	DUAL_OUT	1 = Activates second output.
	6	STANDBY	0 = Amplifier in standby mode.
9 (1001)	6:0	DACRAW_REG	Amplifier DAC raw offset. *Default value <6:0>:"1000000"*
10 (1010)	6:0	DACFINE_REG	Amplifier DAC fine offset. *Default value <6:0>:"1000000"*
11 (1011)	2:0	ADC register	*Default value <2:0>:"011"*
	0	TRISTATE_OUT	0 = output bus in tri-state.
	1	GAMMA	0 = Gamma-correction on.
	2	BIT_INV	1 = Bit inversion on output bus.
12 (1100)		Reserved.	
13 (1101)		Reserved.	
14 (1110)		Reserved.	
15 (1111)		Reserved.	

3.10.2 Detailed description of the internal registers

3.10.2.1 Sequencer register (7:0)

3.10.2.1.1 Shutter type (bit 0)

The IBIS5-A-1300 image sensor has 2 shutter types:

0 = synchronous shutter.
1 = rolling shutter.

3.10.2.1.2 Output amplifier calibration (bits 1 and 2)

Bits FRAME_CAL_MODE and LINE_CAL_MODE define the calibration mode of the output amplifier.

During every row-blanking period, a calibration is done of the output amplifier. There are 2 calibration modes. The FAST mode (= 0) can force a calibration in one cycle but is not so accurate and suffers from KTC noise, while the SLOW mode (= 1) can only make incremental adjustments and is noise free.
Approximately 200 or more "slow" calibrations will have the same effect as 1 "fast" calibration.

Different calibration modes can be set at the beginning of the frame (FRAME_CAL_MODE bit) and for every subsequent line that is read (LINE_CAL_MODE bit). The beginning of a frame is defined by the Y_START input (see lower), Y_CLOCK defines the beginning of a new row.

3.10.2.1.3 Continuous charge (bit 3)

For some applications it might be necessary to use continuous charging of the pixel columns instead of a pre-charge on every line sample operation.
Setting bit CONT_CHARGE to "1" will activate this function. The resistor connected to pin PC_CMD is used to control the current level on every pixel column.

3.10.2.1.4 Internal clock granularities (bits 4, 5, 6 and 7)

The system clock is divided several times on chip.

The X-shift-register that controls the column/pixel readout, is clocked by half the system clock rate. Odd and even pixel columns are switched to 2 separate buses. In the output amplifier the pixel signals on the 2 buses are combined into one pixel stream at the same frequency as SYS_CLOCK.

The clock, that drives the "snapshot" or synchronous shutter sequencer, can be programmed using the bits GRAN_SS_SEQ_MSB (bit 7) and GRAN_SS_SEQ_LSB (bit 6).
This way the integration time in synchronous shutter mode can be a multiple of 32, 64, 128 or 256 times the system clock period. To overcome global reset issues it is advised

that the longest SS granularity is used (bits 6&7 set to '1').

Table 19: SS sequencer clock granularities

GRAN_SS_SEQ_MSB/LSB	SS sequencer clock	Integration time step*
00	*32 x SYS_CLOCK*	*800 ns*
01	*64 x SYS_CLOCK*	*1.6 us*
10	*128 x SYS_CLOCK*	*3.2 us*
11	*256 x SYS_CLOCK*	*6.4 us*

* using a SYS_CLOCK of 40 MHz (25 ns period)

The clock that drives the X-sequencer can be a multiple of 4, 8, 16 or 32 times the system clock. Clocking the X-sequencer at a slower rate (longer row blanking time; pixel read out speed is always equal to the SYSTEM_CLOCK) can result in more signal swing for the same light conditions.

Table 20: X sequencer clock granularities

GRAN_X_SEQ_MSB/LSB	X-sequencer clock	Row Blanking Time*
00	*4 x SYS_CLOCK*	*3.5 us*
01	*8 x SYS_CLOCK*	*7 us*
10	*16 x SYS_CLOCK*	*14 us*
11	*32 x SYS_CLOCK*	*28 us*

* using a SYS_CLOCK of 40 MHz (25 ns period)

3.10.2.1.5 Pixel reset knee-point for multiple slope operation (bits 8, 9 and 10)

In normal (single slope) mode the pixel reset is controlled from the left side of the image core using the voltage applied on pin VDDR_LEFT as pixel reset voltage.
In multiple slope operation one or more variable pixel reset voltages have to be applied.
Bits KNEE_POINT_MSB and KNEE_POINT_LSB select the on chip-generated pixel reset voltage.
Bit KNEE_POINT_ENABLE set to "1" switches control to the right side of the image core so the pixel reset voltage (VDDR_RIGHT), selected by bits KNEE_POINT_MSB/LSB, is used.
Bit KNEE_POINT_ENABLE should only be used for multiple slope operation in synchronous shutter mode. In rolling shutter mode, only the bits KNEE_POINT_MSB/LSB must be used to select the second knee-point in dual slope operation.

Table 21: Multiple slope register settings

KNEE_POINT		Pixel reset voltage (V)	Knee-point (V)
MSB/LSB	ENABLE (1)	VDDR_RIGHT	
00	*0 or 1*	*VDDR_LEFT*	*0*
01	*1*	*VDDR_LEFT – 0.76*	*+ 0.76*
10	*1*	*VDDR_LEFT – 1.52*	*+ 1.52*
11	*1*	*VDDR_LEFT – 2.28*	*+ 2.28*

The actual knee-point depends on VDDH, VDDR_LEFT and VDDC applied to the sensor.

3.10.2.1.6 External pixel reset voltage for multiple slope (bit 11)

When bit VDDR_RIGHT_EXT is set to "1", the circuit that generates the variable pixel reset voltage is disabled and the voltage externally applied to pin VDDR_RIGHT is used as the double/multiple slope reset voltage.
When bit VDDR_RIGHT_EXT is set to "0" the variable pixel reset voltage (used for multiple slope operation) can be monitored on pin VDDR_RIGHT.

3.10.2.2 NROF_PIXELS register (11:0)

After the internal x_sync is generated (start of the pixel readout of a particular row), the PIXEL_VALID signal goes high. The PIXEL_VALID signal goes low when the pixel counter reaches the value loaded in the NROF_PIXEL register. Due to the fact that 2 pixels are read at the same clock cycle this number have to be divided by 2 (NROF_PIXELS = (width of ROI / 2) - 1).

3.10.2.3 NROF_LINES register (11:0)

After the internal yl_sync is generated (start of the frame readout with Y_START), the line counter increases with each Y_CLOCK pulse until it reaches the value loaded in the NROF_LINES register and an LAST_LINE pulse is generated.

3.10.2.4 INT_TIME register (11:0)

The INT_TIME register is used to set the integration time of the electronic shutter. The interpretation of the INT_TIME depends on the chosen shutter type (rolling or synchronous).

Synchronous shutter

After the SS_START pulse is applied an internal counter counts the number of SS granulated clock cycles until it reaches the value loaded in the INT_TIME register and a TIME_OUT pulse is generated. This TIME_OUT pulse can be used to generate the SS_STOP pulse to stop the integration. When the INT_TIME register is used the maximum integration time is:

$T_{INT\_MAX} = 2^{12} * 256$ (maximum granularity) * $(40\ MHZ)^{-1} = 26.2$ ms.

This maximum time can be increased if an external counter is used to trigger SS_STOP. The minimal value that should be loaded into the INT_TIME register is 10 (see also 3.10.2.1.4).

Rolling shutter

When the Y_START pulse is applied (start of the frame readout), the sequencer will generate the yl_sync pulse for the left Y-shift register (read out Y-shift register). This loads the left Y-shift register with the pointer loaded in YL_REG register. At each Y_CLOCK pulse, the pointer shifts to the next row and the integration time counter increases until it reaches the value loaded in the INT_TIME register. At that moment, the

yr_sync pulse for the right Y-shift register is generated which loads the right Y-shift register (reset Y-shift register) with the pointer loaded in YR_REG register (*Figure 17*). The integration time counter is reset when the sync for the left Y-shift register is asserted. Both shift registers keep moving until the next sync is asserted (Y_START for the left Y-shift register and the sync for the right Y-shift register is generated when the integration time counter reaches the INT_TIME value).

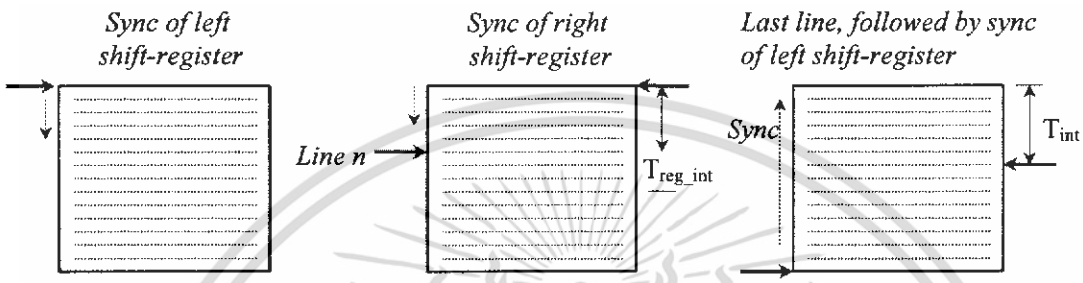


Figure 17: Synchronization of the shift registers in rolling shutter mode.

$\mathbf{T_{reg\_int}}$ Difference between left and right pointer = value set in the INT_TIME register (number of lines).

The actual integration time is given by:

$\mathbf{T_{int}}$ Integration time [# lines] = NROF_LINES register - INT_TIME register

3.10.2.5 X_REG register (10:0)

The X_REG register determines the start position of the window in the X-direction. In this direction, there are 640 possible starting positions (2 pixels are addressed at the same time in one clock cycle). If sub sampling is enabled only the even pixels can be set as starting position (for instance: 0, 2, 4, 6, 8… 638).

3.10.2.6 YL_REG (10:0) and YR_REG (10:0)

The YL_REG and YR_REG registers determine the start position of the window in the Y-direction. In this direction, there are 1024 possible starting positions. In rolling shutter mode the YL_REG register sets the start position of the read (left) pointer and the YR_REG sets the start position of the reset (right) pointer. For both shutter types YL_REG should always be equal to YR_REG.

3.10.2.7 Image core register (7:0)

Bits 1:0 of the IMAGE_CORE register define the test mode of the image core. Setting 00 is the default and normal operation mode. In case the bit is set to 1, the odd (bit 1) or even (bit 0) columns are tight to the reset level. If the internal ADC is used bits 0 and 1 can be used to create test pattern to test the sample moment of the ADC. If the ADC sample moment is not chosen correctly the created test pattern will not be black-white-black-etc.

(IMAGE_CORE register set at 1 or 2) or black-black-white-white-black-black (IMAGE_CORE register set at 9) but grey shadings if the sensor is saturated. See also paragraph 3.8.10 for detailed ADC timing.
Bits 7:2 of the IMAGE_CORE register define the sub-sampling mode in the X-direction (bits 4:2) and in the Y-direction (bits 7:5). The sub-sampling modes and corresponding bit setting are given in Table 12 (section 3.5) and Table 13 (section 3.6).

3.10.2.8 Amplifier register (6:0)

3.10.2.8.1 GAIN (bits 3:0)

The gain bits determine the gain setting of the output amplifier. They are only effective if UNITY = 0. The gains and corresponding bit setting are given in Table 14 section 3.7.2).

3.10.2.8.2 UNITY (bit 4)

In case UNITY = 1, the gain setting of GAIN is bypassed and the gain amplifier is put in unity feedback.

3.10.2.8.3 DUAL_OUT (bit 5)

If DUAL_OUT = 1, the two output amplifiers are active. If DUAL_OUT = 0, the signals from the two busses are multiplexed to output PXL_OUT1 which should be connected to ADC_IN. The gain amplifier and output driver of the second path are put in standby.

3.10.2.8.4 STANDBY

If STANDBY = 0, the complete output amplifier is put in standby. For normal use STANDBY should be set to 1.

3.10.2.9 DAC_RAW register (6:0) and DAC_FINE (6:0) register

These registers determine the black reference level at the output of the output amplifier. Bit setting 1111111 for DAC_RAW register gives the highest offset voltage, bit setting 0000000 for DAC_RAW register gives the lowest offset voltage. Ideally, if the two output paths have no offset mismatch, the DAC_FINE register must be set to 1000000. Deviation from this value can be used to compensate the internal mismatch (see 3.7).

3.10.2.10 ADC register (2:0)

3.10.2.10.1 TRISTATE_OUT (bit 0)

In case TRISTATE = 0, the ADC_D<9:0> outputs are in tri-state mode. TRISTATE = 1 for normal operation mode.

3.10.2.10.2 GAMMA (bit 1)

If GAMMA is set to 1, the ADC input to output conversion is linear; otherwise the

conversion follows a 'gamma' law (more contrast in dark parts of the window, lower contrast in the bright parts). See section 3.8.3 for more details.

3.10.2.10.3 BIT_INV (bit 2)

If BIT_INV = 1, 0000000000 is the conversion of the lowest possible input voltage, otherwise the bits are inverted.

3.10.3 Data interfaces

2 different data interfaces are implemented. They can be selected using pins IF_MODE (pin 12) and SER_MODE (pin 6).

Table 22: Serial and parallel interface selection

IF_MODE	SER_MODE	Selected Interface
1	X	*Parallel*
0	1	*Serial 3 Wire*
0	0	*No mode selected.*

3.10.3.1 Parallel interface

The parallel interface uses a 16-bit parallel input (P_DATA <15:0>) to upload new register values. Asserting P_WRITE will load the parallel data into the internal register of the IBIS5-A-1300 where it is decoded.

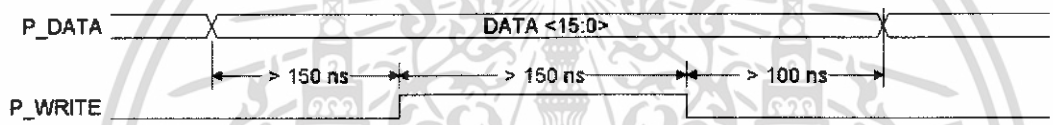


Figure 18: Parallel interface timing

P_DATA (15:12)	Address bits	REG_ADDR (3:0)
P_DATA (11:0)	Data bits	REG_DATA (11:0)

3.10.3.2 Serial-3-wire interface

The serial-3-wire interface (or Serial-to-Parallel Interface) uses a serial input to shift the data in the register buffer. When the complete data word is shifted into the register buffer the data word is loaded into the internal register where it is decoded.

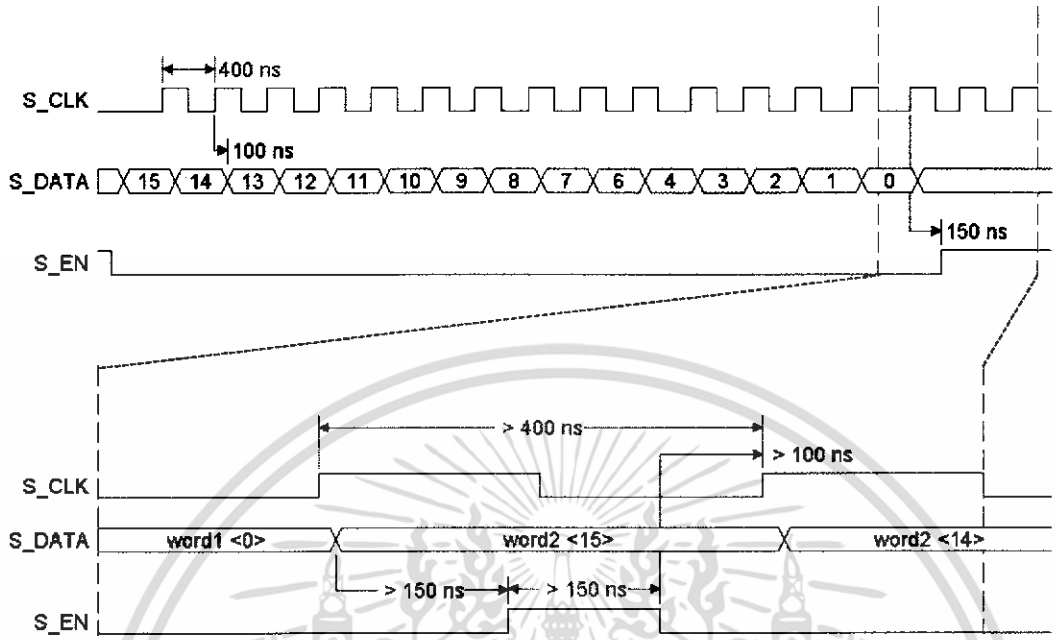


Figure 19: Serial-3 –wire interface timing

S_DATA (15:12) Address bits REG_ADDR (3:0)
S_DATA (11:0) Data bits REG_DATA (11:0)
When S_EN is asserted the parallel data is loaded into the internal registers of the IBIS5-A-1300. The maximum tested frequency of S_DATA is 2.5 MHz.

3.10.3.3 Pins involved in the interface circuitry

Table 23: Pins involved in the interface circuitry

Name	No.	Function
Digital controls		
P_DATA<0>...<7>	*38-45*	*Data parallel interface. <0> : LSB*
P_DATA<15>...<9>	*78-84*	*Data parallel interface. <15> : MSB*
P_DATA<8>	*1*	*Date parallel interface.*
SI2_ADDR<0>...<4>	*46-50*	*2 wire serial address bits (7 bits). <0>:LSB <4>:MSB Bits 4,5 and 6 are tied together.*
P_WR	*2*	*Parallel write.*
S_CLK	*3*	*Clock serial interface.*
S_DATA	*4*	*Data serial interface.*
S_EN	*5*	*Enable serial interface.*
SER_MODE	*6*	*Serial mode: 0 = Disable; 1 = Serial-3-wire enabled.*

4 Timing diagrams

4.1 Timing requirements

There are 6 control signals that operate the image sensor:

- SS_START
- SS_STOP
- Y_CLOCK
- Y_START
- X_LOAD
- SYS_CLOCK

These control signals should be generated by the external system with following time constraints to SYS_CLOCK (rising edge = active edge):

- T_{SETUP} >7.5 ns.
- T_{HOLD} > 7.5 ns.

It is important that these signals are free of any glitches.

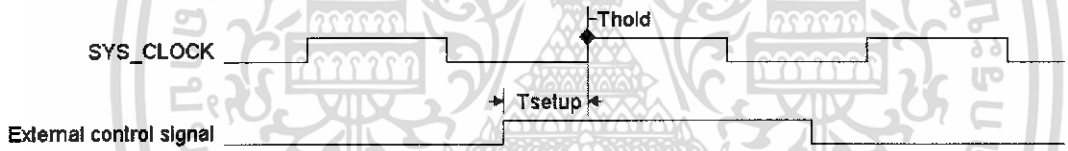


Figure 20: Relative timing of the 5 sequencer control signals

Figure 22 shows a recommended schematic for generating the basic signals and to avoid any timing problems.

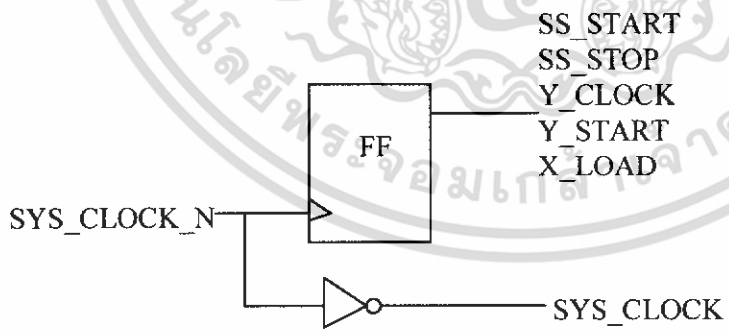


Figure 21: Recommended schematic for generating basic signals

4.2 Synchronous shutter: single slope integration

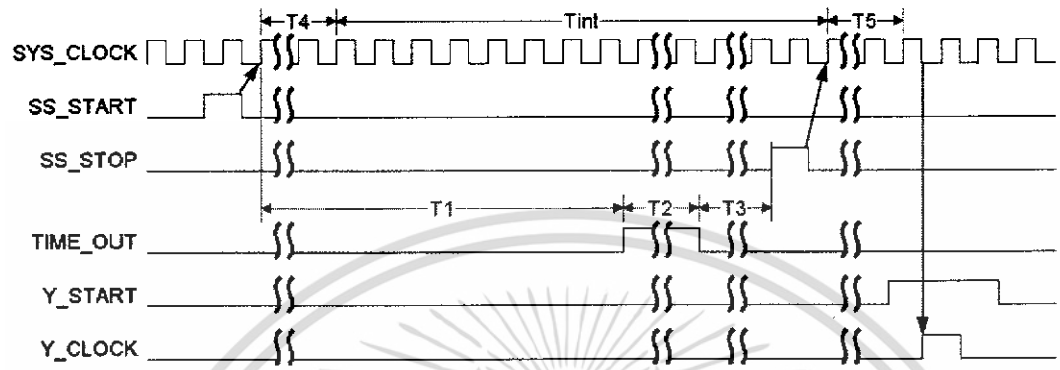


Figure 22: Synchronous shutter: single slope integration

SS_START and SS_STOP should change on the falling edge of the SYS_CLOCK (Tsetup and Thold > 7.5 ns). The pulse width of both signals should be minimum 1 SYS_CLOCK cycle. As long as SS_START or SS_STOP are asserted, the sequencer stays in a suspended state.

T_1 Time counted by the integration timer until the value of INT_TIME register is reached. The integration timer is clocked by the granulated SS-sequencer clock.

T_2 TIME_OUT signal stays high for 1 granulated SS-sequencer clock period.

T_3 There are no constraints for this time. The user can use the TIME_OUT signal to trigger the SS_STOP pin (or use an external counter to trigger SS_STOP) although that both signal can't be tied together.

T_4 During this time, the SS-sequencer applies the control signals to reset the image core and start integration. This takes 4 granulated SS-sequencer clock periods. The integration time counter starts counting at the first rising edge after the falling edge of SS_START.

T_5 The SS-sequencer puts the image core in a readable state. It takes 2 granulated SS-sequencer clock periods.

T_{int} The "real" integration or exposure time.

4.3 Synchronous shutter: pixel readout

4.3.1 Basic operation

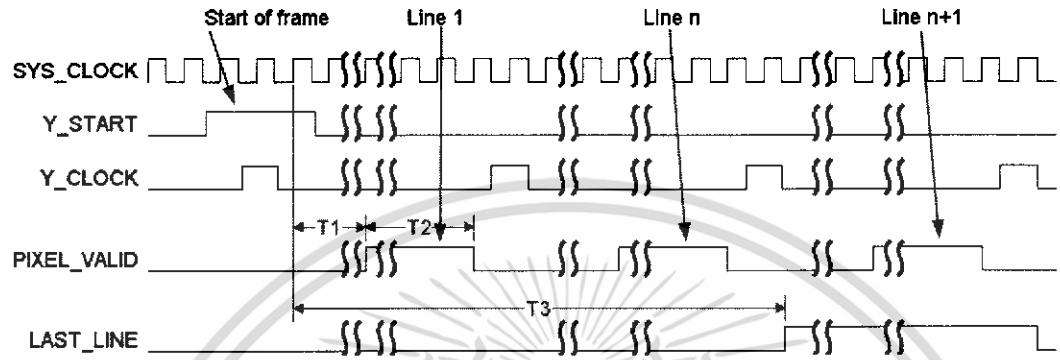


Figure 23: Synchronous shutter: pixel read out

Y_START and Y_CLOCK should change on the falling edge of the SYS_CLOCK (Tsetup and Thold > 7.5 ns). The pulse width should be minimum 1 clock cycle for Y_CLOCK and 3 clock cycles for Y_START. As long as Y_CLOCK is applied, the sequencer stays in a suspended state.

T_1 Row blanking time: During this period, the X-sequencer generates the control signals to sample the pixel signal and pixel reset levels (double sampling fpn-correction), and start the readout of one line. The row blanking time depends on the granularity of the X-sequencer clock (see below).

Table 24: Row blanking time as function of X-sequencer granularity

Granularity N_{GRAN}	T_1(µs) $= 35 \times N_{GRAN} \times T_{SYS\ CLOCK}$	GRAN_X_SEQ MSB/LSB
x 4	$140 \times T_{SYS\ CLOCK} = 3.5$	00
x 8	$280 \times T_{SYS\ CLOCK} = 7.0$	01
x 16	$560 \times T_{SYS\ CLOCK} = 14.0$	10
x 32	$1120 \times T_{SYS\ CLOCK} = 28.0$	11

T_2 Pixels counted by pixel counter until the value of NROF_PIXELS register is reached. PIXEL_VALID goes high when the internal X_SYNC signal is generated, in other words when the readout of the pixels is started. PIXEL_VALID goes low when the pixel counter reaches the value loaded in the NROF_PIXELS register (after a complete row read out).

T_3 LAST_LINE goes high when the line counter reaches the value loaded in the NROF_LINES register and stays high for 1 line period (until the next falling edge of Y-CLOCK).

On Y_START the left Y-shift-register of the image core is loaded with the YL-pointer that

is loaded in to register YL_REG.

4.3.2 Pixel output

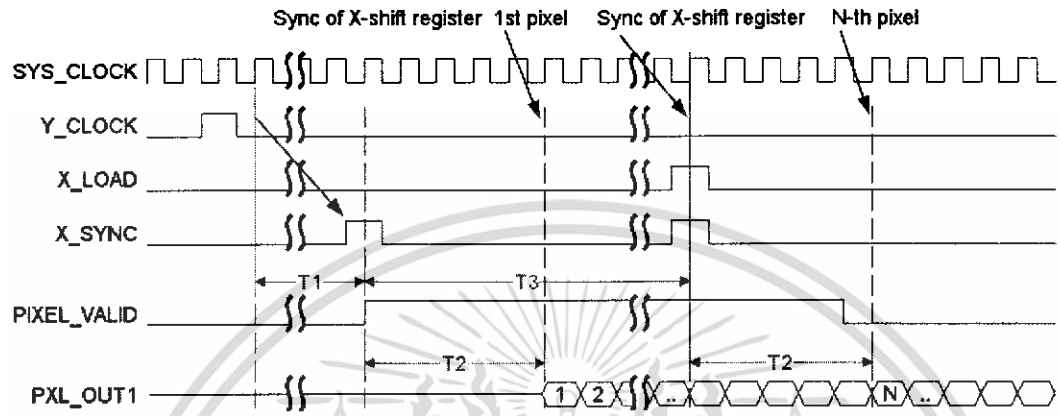


Figure 24: Pixel output

The pixel signal at the PXL_OUT1 output becomes valid after 5 SYS_CLOCK cycles when the internal X_SYNC (= start of PIXEL_VALID output or external X_LOAD pulse) pulse is asserted.

T_1 Row blanking time (see Table 24).
T_2 5 SYS_CLOCK cycles.
T_3 Time for new X-pointer position upload in X_REG register (see 4.6 for more details).

4.4 Synchronous shutter: multiple slope integration

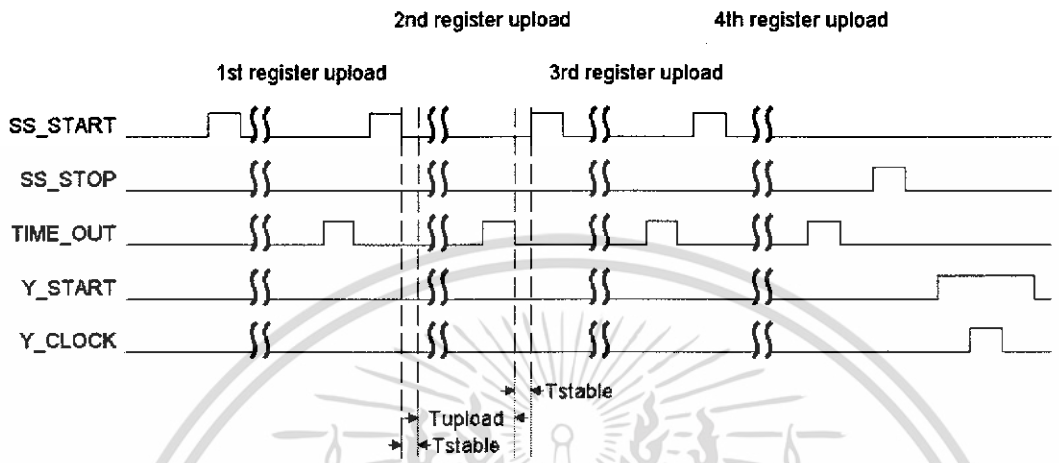


Figure 25: Multiple slope integration

Up to 4 different pixel reset voltages can be used during multiple slope operation in synchronous shutter mode. This is done by uploading new values to register bits KNEEPOINT_MSB/LSB/ENABLE before a new SS_START pulse is applied.
Bit KNEEPOINT_ENABLE should be set high to do a pixel reset with a lower voltage.
Bits KNEEPOINT_MSB/LSB/ENABLE should be set back to "0" before the SS_STOP pulse is applied. Every time an SS_START pulse is applied, the integration time counter is reset.

Table 25: Multiple slope register settings

	KNEEPOINT	
	MSB/LSB	**ENABLE**
Initial setup	*00*	*0*
1st register upload	*01*	*1*
2nd register upload	*10*	*1*
3th register upload	*11*	*1*
4th register upload	*00*	*0*

The register upload should be uploaded after time T_{stable}, otherwise the change will affect the SS-sequencer resulting in a bad pixel reset. T_{stable} depends on the granularity of the SS-sequencer clock (see Table 26).

Table 26: T_{stable} for different granularity settings

Granularity N_{pixels}	T_{stable} (µs) $= 5 \times N_{GRAN} \times T_{SYS\ CLOCK}$	GRAN_SS_SEQ MSB/LSB
x 32	$160 \times T_{SYS\ CLOCK} = 4$	00
x 64	$320 \times T_{SYS\ CLOCK} = 8$	01
x 128	$640 \times T_{SYS\ CLOCK} = 16$	10

Granularity N_{GRAN}	T_{xsize} (µs) $= 5 \times N_{GRAN} \times T_{SYS\ CLOCK}$	GRAN_SS_SEQ MSB/LSB
x 256	*1280 x $T_{SYS\ CLOCK}$ = 32*	11

T_{upload} depends on the interface mode used to upload the registers

Table 27: T_{upload} for different interface modes

Interface mode	T_{upload} (µs)
Parallel	*1*
Serial 3 Wire	*8 (2.5 MHz clock rate)*

4.5 Rolling shutter operation

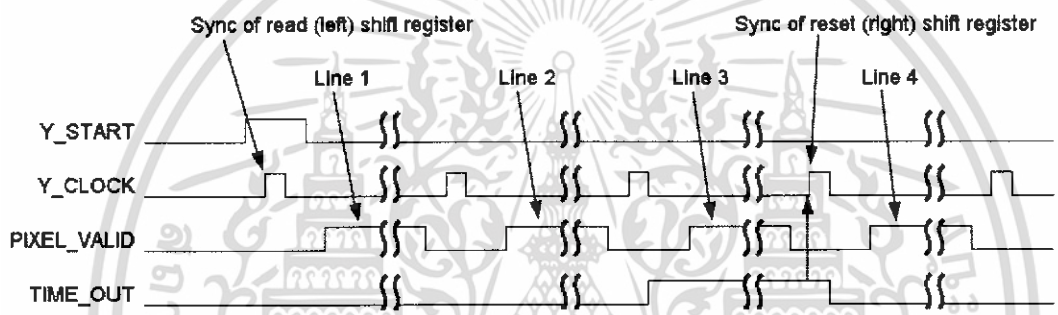


Figure 26: Rolling shutter operation

The integration of the light in the image sensor is done during readout of the other lines. The only difference with synchronous shutter is that the TIME_OUT pin is used to indicate when the Y_SYNC pulse for the right Y-shift-register (reset Y-shift register) is generated. This loads the right Y-shift-register with the pointer loaded in register YR_REG. The Y_SYNC pulse for the left Y-shift register (read Y-shift register) is generated with Y_START.
The INT_TIME register defines how many lines have to be counted before the Y_SYNC of the right Y-shift-register is generated, hence defining the integration time. See also chapter's 3.10.2 and 3.10.2.4 for a detailed description of the rolling shutter operation.

T_{int} Integration time [# lines] = register(NROF_LINES) - register(INT_TIME)

Note: For normal operation the values of the YL_REG and YR_REG registers are equal.

4.6 Windowing in X-direction

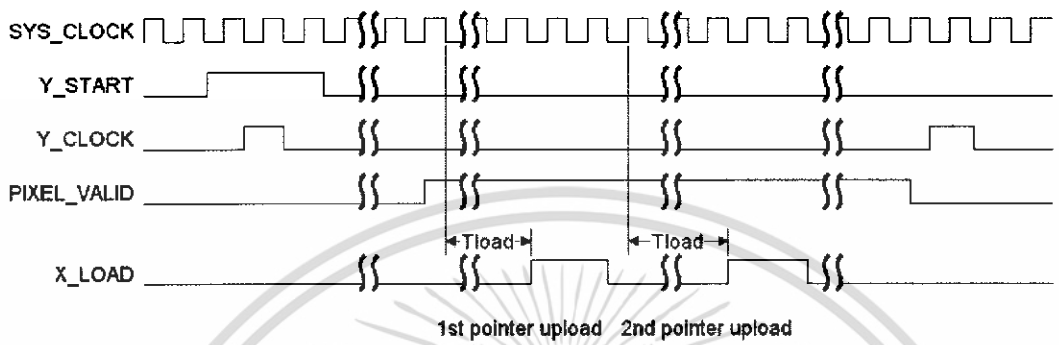


Figure 27: Windowing in the X-direction

An X_LOAD pulse overrides the internal X_SYNC signal, loading a new X-pointer (stored in the X_REG register) into the X-shift-register.

The X_LOAD pulse has to appear on the falling edge of SYS_CLOCK and has to remain 2 SYS_CLOCK cycles high overlapping 2 rising edges of SYS_CLOCK. On one of the 2 rising edges of SYS_CLOCK the new X-pointer is loaded.

T_{load} is the available time to upload the register and is defined from the previous register load to the rising edge of X_LOAD. It depends on the settling time of the register and the X-decoder.

The actual time to load the register is self depends on the interface mode that is used. The parallel interface is the fastest.

Table 28: T_{load} for different interfaces

Interface mode	T_{load} (µs)
Parallel interface	*1 (about 40 SYS_CLOCK cycles)*
Serial 3 Wire	*16 (at 2.5 MHz data rate)*

4.7 Windowing in Y-direction

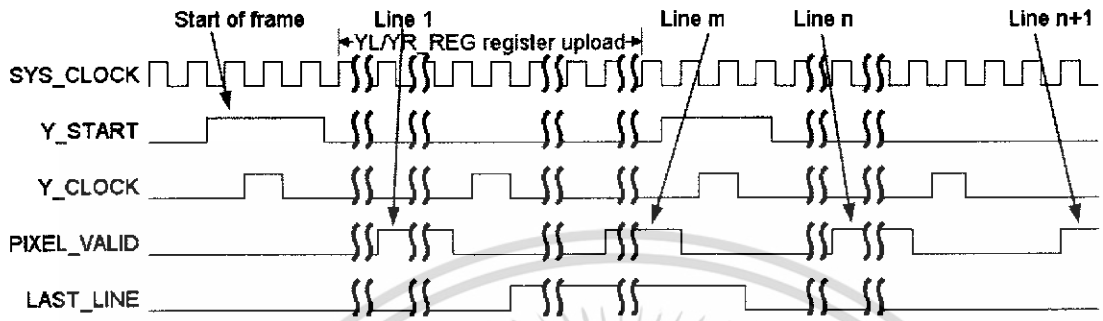


Figure 28: Windowing in the Y-direction

A new Y-pointer can be loaded into the Y-shift-register, by reapplying the Y_START pulse after loading a new Y-pointer value into the YL_REG and YR_REG registers.

Every time a Y_START pulse appears, a frame calibration of the output amplifier is done.

4.8 Initialization (start up behaviour)

To avoid any high current consumption at start up it is recommended to apply the SYS_CLOCK signal as soon as possible after or even before power on of the image sensor.

After power on of the image sensor it is recommended to apply SYS_RESET for minimal 5 SYS_CLOCK periods to assure a proper reset of the on-chip sequencer and timing circuitry. All internal register will be set to 0 after SYS_RESET is applied.

As all the IBIS5-A-1300 control signal are active high it is also recommended to apply a low level (before SYS_RESET occurs) to these pins at start up to avoid latch up.

5 Pin list

The IBIS5-A-1300 image sensor is packaged in a leadless ceramic carrier (LCC package). Table 29 is a list of all the pins and their function. In total, there are 84 pins.

Table 29: Pin list

Pin	Pin name	Pin type	Pin description
1	**P_DATA<8>**	Input	Digital input. Data parallel interface.
2	**P_WR**	Input	Digital input (active high). Parallel write.
3	**S_CLK**	Input	Digital input. Clock signal of serial interface.
4	**S_DATA**	Input	Digital input/output. Data of serial interface.
5	**S_EN**	Input	Digital input (active low). Enable of Serial-3-wire interface.
6	**SER_MODE**	Input	Digital input. Serial mode enable (1=Enable serial-3-wire, 0=disable).
7	**VDDC**	Supply	Analog supply voltage. Supply voltage of the pixel core [3.3V].
8	**VDDA**	Supply	Analog supply voltage. Analog supply voltage of the image sensor [3.3V].
9	**GNDA**	Ground	Analog ground. Analog ground of the image sensor.
10	**GNDD**	Ground	Digital ground. Digital ground of the image sensor.
11	**VDDD**	Supply	Digital supply voltage. Digital supply voltage of the image sensor [3.3V].
12	**IF_MODE**	Input	Digital input. Interface mode (1=parallel; 0=serial).
13	**DEC_CMD**	Input	Analog input. Biasing of decoder stage. Connect to VDDA with R = 50 kΩ and decouple with C=100nF to GNDA.
14	**Y_START**	Input	Digital input (active high). Start frame read out.
15	**Y_CLOCK**	Input	Digital input (active high). Line clock.
16	**LAST_LINE**	Output	Digital output. Generates a high level when the last line is read out.
17	**X_LOAD**	Input	Digital input (active high). Loads new X-position during read out.
18	**SYS_CLOCK**	Input	Digital input. System (pixel) clock (40 MHz).
19	**PXL_VALID**	Output	Digital output. Generates high level during pixel read out.
20	**SS_START**	Input	Digital input (active high). Start synchronous shutter operation.
21	**SS_STOP**	Input	Digital input (active high). Stop synchronous shutter operation.
22	**TIME_OUT**	Output	Digital output. Synchronous shutter: pulse when time-out reached. Can be used to trigger SS_STOP although both signals can't be tied together. Rolling shutter: pulse when second Y-sync appears.
23	**SYS_RESET**	Input	Digital input (active high). Global system reset.
24	**EL_BLACK**	Input	Digital input (active high). Enables electrical black in output amplifier.
25	**EOSX**	Output	Digital output. Diagnostic end-of-scan of X-register.

Pin	Pin name	Pin type	Pin description
26	**DAC_VHIGH**	Input	Analog reference input. Biasing of DAC for output dark level. Can be used to set output range of DAC. Default: Connect to VDDA with R = 0 Ω.
27	**DAC_VLOW**	Input	Analog reference input. Biasing of DAC for output dark level. Can be used to set output range of DAC. Default: Connect to GND A with R = 0 Ω.
28	**PXL_OUT1**	Output	Analog output. Analog pixel output 1.
29	**PXL_OUT2**	Output	Analog output. Analog pixel output 2. Leave not connected if not used.
30	**AMP_CMD**	Input	Analog input. Biasing of the output amplifier. Connect to VDDA with R = 50 kΩ and decouple with C=100nF to GNDA.
31	**COL_CMD**	Input	Analog input. Biasing of the column amplifiers. Connect to VDDA with R = 50 kΩ and decouple with C=100nF to GNDA.
32	**PC_CMD**	Input	Analog input. Pre-charge bias. Connect to VDDA with R = 25 kΩ and decouple with C=100nF to GNDA.
33	**VDDD**	Supply	Digital supply. Digital supply voltage of the image sensor [3.3V].
34	**GNDD**	Ground	Digital ground. Digital ground of the image sensor.
35	**GNDA**	Ground	Analog ground. Analog ground of the image sensor.
36	**VDDA**	Supply	Analog supply voltage. Analog supply voltage of the image sensor [3.3V].
37	**VDDC**	Supply	Analog supply voltage. Supply voltage of the pixel core [3.3V].
38	**P_DATA<0>**	Input	Digital input. Data parallel interface (LSB).
39	**P_DATA<1>**	Input	Digital input. Data parallel interface.
40	**P_DATA<2>**	Input	Digital input. Data parallel interface.
41	**P_DATA<3>**	Input	Digital input. Data parallel interface.
42	**P_DATA<4>**	Input	Digital input. Data parallel interface.
43	**P_DATA<5>**	Input	Digital input. Data parallel interface.
44	**P_DATA<6>**	Input	Digital input. Data parallel interface.
45	**P_DATA<7>**	Input	Digital input. Data parallel interface.
46	**SI2_ADDR<0>**	Input	Unused interface inputs. Tie to GNDD.
47	**SI2_ADDR<1>**	Input	
48	**SI2_ADDR<2>**	Input	
49	**SI2_ADDR<3>**	Input	
50	**SI2_ADDR<4>**	Input	
51	**GNDAB**	Supply	Analog supply voltage. Anti-blooming ground.
52	**VDDR_RIGHT**	Supply	Analog supply voltage. Variable reset voltage (multiple slope operation). Decouple with 1uF to GNDA.
53	**ADC_VLOW**	Input	Analog reference input. ADC low reference voltage. Default: Connect to GNDA with R = 360 Ω and decouple with C=100nF to GNDA.
54	**ADC_GNDA**	Ground	Analog ground. ADC analog ground.
55	**ADC_VDDA**	Supply	Analog supply voltage. ADC analog supply voltage [3.3V].
56	**ADC_GNDD**	Ground	Digital ground. ADC digital ground.
57	**ADC_VDDD**	Supply	Digital supply voltage. ADC digital supply voltage [3.3V].

Pin	Pin name	Pin type	Pin description
58	**ADC_CLOCK**	Input	Digital input. ADC clock (40 MHz).
59	**ADC_OUT<9>**	Output	Digital output. ADC data output (MSB).
60	**ADC_OUT<8>**	Output	Digital output. ADC data output.
61	**ADC_OUT<7>**	Output	Digital output. ADC data output.
62	**ADC_OUT<6>**	Output	Digital output. ADC data output.
63	**ADC_OUT<5>**	Output	Digital output. ADC data output.
64	**ADC_OUT<4>**	Output	Digital output. ADC data output.
65	**ADC_OUT<3>**	Output	Digital output. ADC data output.
66	**ADC_OUT<2>**	Output	Digital output. ADC data output.
67	**ADC_OUT<1>**	Output	Digital output. ADC data output.
68	**ADC_OUT<0>**	Output	Digital output. ADC data output (LSB).
69	**ADC_IN**	Input	Analog input. ADC analog input.
70	**ADC_CMD**	Input	Analog input. Biasing of the input stage of the ADC. Connect to ADC_VDDA with R = 50 kΩ and decouple with C=100nF to ADC_GNDA.
71	**ADC_VDDD**	Supply	Digital supply voltage. ADC digital supply voltage [3.3V].
72	**ADC_GNDA**	Ground	Analog ground. ADC analog ground.
73	**ADC_GNDD**	Ground	Digital ground. ADC digital ground.
74	**ADC_VDDA**	Supply	Analog supply voltage. ADC analog supply voltage [3.3V].
75	**ADC_VHIGH**	Input	Analog reference input. ADC high reference voltage. Default: Connect to VDDA with R = 90 Ω and decouple with C=100nF to GNDA.
76	**VDDR_LEFT**	Supply	Analog supply voltage. High reset level [4.5V].
77	**VDDH**	Supply	Analog supply voltage. High supply voltage for HOLD switches in the image core [4.5V]
78	**P_DATA<15>**	Input	Digital input. Data parallel interface (MSB).
79	**P_DATA<14>**	Input	Digital input. Data parallel interface.
80	**P_DATA<13>**	Input	Digital input. Data parallel interface.
81	**P_DATA<12>**	Input	Digital input. Data parallel interface.
82	**P_DATA<11>**	Input	Digital input. Data parallel interface.
83	**P_DATA<10>**	Input	Digital input. Data parallel interface.
84	**P_DATA<9>**	Input	Digital input. Data parallel interface.

REMARKS:

1. All pins with the same name can be connected together.
2. All digital input are active high (unless mentioned otherwise).
3. Digital inputs that are not used should be tied to GND.