

การออกแบบวงจรจัดเรียงข้อมูลแบบขนานโดยใช้เอฟพีจีเอ



นาย ชานู เชาว์ปฏิภาณ

เลขหมู่.....
เลขทะเบียน... 43933
วัน, เดือน, ปี... 18 ต.ค. 2545

b.....
i.....

โครงการพิเศษที่เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาฟิสิกส์ประยุกต์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2544

A parallel sorter design with FPGA



Mr. Chand Chaopatipharn

A Special Project Submitted on Partial Fulfillment of the

Reruirement For the Degree of Bachelor of Science

Department of Applied Physics

Faculity of Science

King Mongkut's Institute of Technology Ladkrabang

2001

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หัวข้อโครงการพิเศษ
นักศึกษา
อาจารย์ที่ปรึกษา

การออกแบบวงจรจัดเรียงข้อมูลแบบขนาน โดยใช้ เอฟพีจีเอ
นาย ชานู เชาว์ปฏิภาณ
ผศ.อนุพงษ์ สรงประภา

ภาควิชาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
อนุมัติให้นำโครงการฉบับนี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรวิทยาศาสตรบัณฑิต

.....หัวหน้าภาควิชาฟิสิกส์ประยุกต์
(ผศ. วิชาญ เตชิตธีระ)

คณะกรรมการ โครงการพิเศษ

.....ประธานกรรมการ
(ผศ. อนุพงษ์ สรงประภา)

.....กรรมการ
(รศ.ดร. ปรีชา ยุพพิน)

.....กรรมการ
(อ.สุรชาติ กมลดีถก)

.....กรรมการ
(อ.ธนภรณ์ ตีลาวัฒนานนท์)

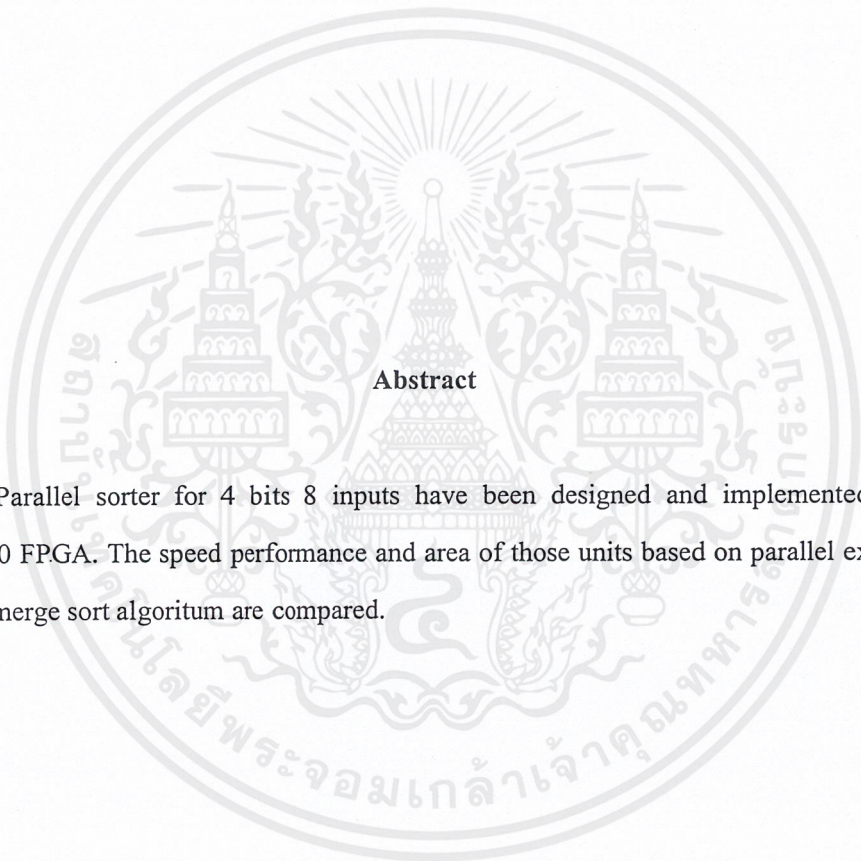
ลิขสิทธิ์ของภาควิชาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อโครงการพิเศษ	การออกแบบวงจรจัดเรียงข้อมูลแบบขนานโดยใช้เอฟพีจีเอ
นักศึกษา	นาย ชานู เชาว์ปฏิภาณ
อาจารย์ที่ปรึกษา	ผศ.อนุพงษ์ ศรีงประภา
ภาควิชา	ฟิสิกส์ประยุกต์
ปีการศึกษา	2544

บทคัดย่อ

โครงการนี้เป็นการออกแบบวงจรจัดเรียงข้อมูลแบบขนานขนาด 4 บิตและสามารถรับค่าเข้ามาครั้งละ 8 ข้อมูลซึ่งออกแบบลงบนชิป XC2S200 ของบริษัทไซลิงค์ มีการเปรียบเทียบในด้านความเร็วของการทำงานและขนาดของวงจรที่อยู่บนพื้นฐานของการจัดเรียงแบบแลกเปลี่ยนข้อมูลและการจัดเรียงแบบแบ่งส่วน

Special Project Title	A parallel sorter design with FPGA
Name	Mr. Chand Chaopatipharn
Special Project Advisor	Asst.Prof. Anupong Srongprapa
Department	Applied Physics
Academic Year	2001



Abstract

Parallel sorter for 4 bits 8 inputs have been designed and implemented on Xilinx XC2S200 FPGA. The speed performance and area of those units based on parallel exchange sort parallel merge sort algorithm are compared.

กิตติกรรมประกาศ

โครงการนี้สำเร็จลุล่วงเพราะความช่วยเหลือจากบุคคลดังต่อไปนี้
พ่อ และ แม่ ที่คอยห่วงใยและเอาใจใส่ด้วยดีเสมอมา
อาตุ๋ อาอิ ที่ช่วยเหลือในหลายสิ่งหลายอย่างจนผมสามารถเรียนจบ
อาจารย์ อนุพงษ์ ที่ให้โอกาสทำในสิ่งที่ผมสนใจ
พี่ๆจาก TMEC (IC DESIGN) ที่ให้คำชี้แนะและช่วยเหลือ
เพื่อนๆ ที่ให้ความช่วยเหลือในการตรวจการสะกดคำ



สารบัญเรื่อง

	หน้า
บทคัดย่อโครงการพิเศษภาษาไทย	ก
บทคัดย่อโครงการพิเศษภาษาอังกฤษ	ข
กิตติกรรมประกาศ	ค
สารบัญเรื่อง	ง
สารบัญรูปภาพ	จ
สารบัญตาราง	ฉ
บทที่ 1 บทนำ	
1.1 ความสำคัญและที่มา	1
1.2 วัตถุประสงค์	2
1.3 ขอบเขตของโครงการ	2
1.4 ขั้นตอนการทำงาน	2
1.5 ประโยชน์ที่คาดว่าจะได้รับ	3
บทที่ 2 หลักการใช้งานเอฟพีจีเอและการจัดเรียงข้อมูล	
2.1 ภาษาวีเอชดีแอล	4-9
2.2 เอฟพีจีเอ	10-14
2.3 การจัดเรียงข้อมูลแบบอนุกรม	15-20
2.4 การประมวลผลแบบขนาน	21-23
บทที่ 3 การดำเนินงานและการวิจัย	
3.1 ขั้นตอนการออกแบบ	24
3.2 การกำหนดการออกแบบ	26-32
3.3 กรรมวิธีทดสอบการทำงาน	33-35
บทที่ 4 ผลการทดสอบการทำงานของวงจร	36
บทที่ 5 สรุปผลการดำเนินการ	47
ภาคผนวก	
เอกสารอ้างอิง	

สารบัญรูปภาพ

	หน้า
รูปที่ 2.1 แสดงมุมมองการ interface	7
รูปที่ 2.2 สถาปัตยกรรมภายในเอฟพีจีเอของ XILINX	11
รูปที่ 2.3 โครงสร้างของซีแอลบี	12
รูปที่ 2.4 แสดงผังวงจรของไอ โอบีภายในเอฟพีจีเอ	13
รูปที่ 2.5 แสดงส่วนเชื่อมต่อโยระหว่างไอ โอบีกับซีแอลบีของเอฟพีจีเอ	14
รูปที่ 2.6 แสดงโครงสร้างของ Exchange – Sort	22
รูปที่ 2.7 แสดงโครงสร้างของ Bitonic Merge Sort	23
รูปที่ 3.1 แสดงขั้นตอนการออกแบบ	25
รูปที่ 3.2 แสดงโครงสร้างของ Parallel Exchange sort	29
รูปที่ 3.3 แสดงโครงสร้างภายในของส่วนจัดเรียง(Exchange sort)	29
รูปที่ 3.4 แสดงโครงสร้างของ Bitonic Merge Sort	32
รูปที่ 3.5 แสดงโครงสร้างภายในของส่วนจัดเรียง(Bitonic sort)	32
รูปที่ 3.6 แสดงหน้าจอโปรแกรม Modelsim	33
รูปที่ 3.7 แสดงหน้าจอโปรแกรม Leonardo Spectrum	34
รูปที่ 3.8 แสดงหน้าจอโปรแกรม Impact	35
รูปที่ 4.1 ผลที่ได้จากการจำลองการทำงานของ Parallel Exchange Sort	36
รูปที่ 4.2 ผลที่ได้จากการจำลองการทำงานของ Parallel Bitonic Merge Sort	37
รูปที่ 4.3 แสดงวงจร Parallel Exchange Sort ที่ผ่านการสังเคราะห์	39
รูปที่ 4.4 แสดงวงจรส่วนจัดเรียงแบบ Parallel Exchange Sort	39
รูปที่ 4.5 แสดงวงจร Parallel Bitonic Merge Sort ที่ผ่านการสังเคราะห์	40
รูปที่ 4.6 แสดงวงจรส่วนจัดเรียงแบบ Parallel Bitonic Merge Sort	40
รูปที่ 4.7 แสดงพื้นที่ภายในชิปที่ถูกนำไปใช้เป็นวงจร Parallel Exchange Sort	41
รูปที่ 4.8 แสดงพื้นที่ภายในชิปที่ถูกใช้เป็นวงจร Parallel Bitonic Merge Sort	41
รูปที่ 4.9 แสดงผลจากวงจร Parallel Exchange Sort	42
รูปที่ 4.8 แสดงผลจากวงจร Parallel Bitonic Merge Sort	43

สารบัญตาราง

	หน้า
ตารางที่ 2.3 เปรียบเทียบการทำงานของแต่ละอัลกอริทึม	20
ตารางที่ 4.1 แสดงผลการทำงานที่ได้จากวงจร Exchange Sort	37
ตารางที่ 4.2 แสดงผลการทำงานของวงจร Bitonic Merge Sort	38



บทที่ 1

บทนำ

เอฟพีจีเอ (FPGA , Field Programmable Gate Array) เป็นวงจรรคิติดอลอเนกประสงค์ซึ่งมีอุปกรณ์พื้นฐานคือ เกตจำนวนมาก สามารถปรับลักษณะการเชื่อมโยงเป็นวงจรด้วยการโปรแกรม เพื่อให้ตัว เอฟพีจีเอ มีฟังก์ชันการทำงานตามที่ผู้ออกแบบต้องการ ในการทำ เอฟพีจีเอ เมื่อเทียบกับการทำเอซิค (ASIC , Application Specific Integrated Circuits) นั้นก็มีทั้งข้อดีและข้อเสีย คือการทำเอฟพีจีเอจะมีข้อจำกัดในด้านขนาดของวงจรเพราะภายในอุปกรณ์เอฟพีจีเอจะมีจำนวนเกตให้ใช้จำนวนจำกัดและการทำเอฟพีจีเอก็เหมาะกับการทำผลิตภัณฑ์ต้นแบบหรือเพื่อผลิตในปริมาณต่ำ ส่วนข้อดีของการทำเอฟพีจีเอ ก็คือระยะเวลาที่ใช้ในการทำตั้งแต่เขียนโค้ดจนกระทั่ง download นั้นน้อยกว่าการทำ ASIC มาก และการตรวจสอบแก้ไขโมเดลก็ทำได้สะดวก

ในปัจจุบันเอฟพีจีเอถูกออกแบบให้มีประสิทธิภาพมากขึ้น และสะดวกขึ้น ทั้งนี้ก็เนื่องจากทางบริษัทผู้ผลิตอุปกรณ์เอฟพีจีเอได้เพิ่มความสามารถของอุปกรณ์เอฟพีจีเอ โดยเพิ่มจำนวนองค์ประกอบภายในหรือปรับปรุงโครงสร้างสถาปัตยกรรมภายใน และยังได้เพิ่มประสิทธิภาพของซอฟต์แวร์ที่ใช้ทำ พีพีอาร์ (PPR , Partitioning, Placement and Routing) สำหรับอุปกรณ์นั้นด้วย

ในการออกแบบวงจรรคิติดอลนั้นทำได้ทั้งการวาดวงจร (Schematic Drawing) หรือใช้ภาษาอธิบายฮาร์ดแวร์ ในกรณีที่ออกแบบโดยวิธีการวาดวงจรผู้ทำการออกแบบต้องคำนึงถึงเทคโนโลยีที่จะใช้ซึ่งแต่ละเทคโนโลยีก็มีความแตกต่างกันไป ส่วนการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ทำได้สะดวกกว่าเพราะวิธีนี้ผู้ออกแบบไม่ต้องคำนึงถึงเทคโนโลยีที่จะใช้ และที่สำคัญการออกแบบโดยวิธีนี้สามารถแก้ไขโมเดลหรือเปลี่ยนเทคโนโลยีได้สะดวกกว่าเพราะไม่ต้องวาดวงจรใหม่ ดังนั้นการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยี สำหรับภาษาอธิบายฮาร์ดแวร์ที่ใช้นั้นมี 2 ภาษาคือ VHDL และ Verilog ในการเขียนโค้ดนั้นสิ่งที่ต้องคำนึงคือเขียนอย่างไรจึงจะสามารถสังเคราะห์เป็นวงจรได้และได้โมเดลที่มีคุณสมบัติถูกต้องตามกำหนด เพราะ ลักษณะการเขียนโค้ดจะมีผลโดยตรงกับโมเดลที่ได้ เนื่องจากในการสังเคราะห์วงจรนั้นซอฟต์แวร์สังเคราะห์วงจรจะสังเคราะห์วงจรตามโค้ด เช่น การเขียนโค้ดอธิบายการทำงานของโมเดลวงจรอันเดียวกัน แต่ เขียนในลักษณะที่ต่างกัน เมื่อนำไปสังเคราะห์เป็นวงจร วงจรที่ได้ก็จะต่างกัน จากวงจรที่ต่างกันเมื่อนำไปทำต้นแบบด้วยเอฟพีจีเอ หรือการทำเอซิคก็จะได้ไอซีที่มีคุณสมบัติต่างกันทั้งในด้านขนาดหรือความเร็ว (Area and Time) ดังนั้นการออกแบบขั้นต้นนี้ผู้ออก

แบบต้องระมัดระวังเป็นพิเศษส่วนการที่จะเขียนโค้ดในลักษณะใดที่ให้ได้ผลลัพธ์ที่ดีที่สุดนั้นก็ขึ้นอยู่กับประสบการณ์ของผู้ออกแบบ

จะเห็นได้ว่าการออกแบบวงจรโดยใช้เอฟพีจีเอ นั้นทำได้สะดวกกว่าการทำเอซิคมากเพราะใช้เวลาน้อยกว่า ส่วนที่สำคัญในการทำเอฟพีจีเอ คือซอฟต์แวร์ที่ใช้ตั้งแต่เขียนโค้ดจนกระทั่งดาวน์โหลดลงในในตัวชิปเอฟพีจีเอ ต้องเป็นซอฟต์แวร์ที่ทำงานต่อเนื่องกันได้ กล่าวคือ ผลลัพธ์ที่ได้จากซอฟต์แวร์ที่ใช้ในการทำการสังเคราะห์วงจรต้องนำไปใช้ กับซอฟต์แวร์ที่ใช้ในการทำพีอาร์ได้

ในปัจจุบันแนวทางการออกแบบระบบทางดิจิทัลมักนำแนวความคิดการประมวลผลแบบขนานที่มีหน่วยประมวลผลซึ่งทำงานเฉพาะด้านหลายหน่วยช่วยกันทำงานมาใช้มากขึ้น เนื่องจาก การประยุกต์เฉพาะทางบางอย่างต้องการความเร็วสูงในการทำและอาจไม่เหมาะที่จะนำมาประมวลผลด้วยคอมพิวเตอร์ส่วนบุคคลซึ่งถูกออกแบบมาใช้ทำงานประยุกต์อย่างกว้างๆในงานทั่วไป ดังนั้นการนำเอฟพีจีเอมาสร้างเป็นหน่วยจัดเรียงข้อมูลแบบขนานก็จัดอยู่ในแนวทางดังกล่าว

1.1 ความสำคัญและที่มา

ปัจจุบันอุตสาหกรรมอิเล็กทรอนิกส์มีการนำเข้าสู่ส่งออกและใช้ในประเทศสูงมากแต่เนื่องจากในประเทศไทยยังไม่สามารถออกแบบวงจรรวม (integrated circuit) และผลิตขึ้นมาเองได้จึงทำให้เราต้องพึ่งพาเทคโนโลยีจากต่างประเทศเป็นหลักแต่ในตอนนี้เราสามารถออกแบบวงจรรวมโดยใช้ภาษาอธิบายฮาร์ดแวร์ กับตัวชิป เอฟพีจีเอในการออกแบบวงจรรวมขึ้นมา

1.2 วัตถุประสงค์

เพื่อศึกษาเทคโนโลยีรูปแบบหนึ่งที่ใช้ในการออกแบบวงจรรวมโดยใช้ภาษาอธิบายฮาร์ดแวร์กับ ชิพเอฟพีจีเอ โครงการนี้ตั้งใจออกแบบวงจรรวมที่ใช้กับงานเฉพาะด้าน (Custom made) ที่ใช้ในการจัดเรียงข้อมูลแบบขนาน

1.3 ขอบเขตของโครงการ

1. ศึกษาภาษาอธิบายฮาร์ดแวร์
2. ศึกษาการใช้งาน เอฟพีจีเอ
3. ศึกษาการจัดเรียงข้อมูล
4. ออกแบบวงจรจัดเรียงข้อมูลแบบขนาน (Parallel Sorter)

1.4 ขั้นตอนการดำเนินงาน

1. ศึกษาการใช้ภาษาวีเอชดีแอล และใช้งานซอฟต์แวร์ที่ใช้ในการจำลองการทำงาน
2. ศึกษาหลักของการจัดเรียงข้อมูล
3. ออกแบบภาพรวมของระบบและแบ่งออกเป็นส่วนๆ ในการสร้าง
4. ทดสอบวงจรที่ออกแบบไว้ในแต่ละส่วน และนำมาประกอบกันและทดสอบจนกระทั่งแน่ใจว่าสามารถทำงานได้ถูกต้อง
5. นำ Core ที่ได้ทำการออกแบบไปสังเคราะห์ เพื่อแปลงโค้ด ไปเป็นระดับ Gate – Level และนำไปทำการ Place & Route เพื่อบันทึกลงชิปเอฟพีจีเอ

1.5 ประโยชน์ที่ได้รับ

เข้าใจขั้นตอนการออกแบบวงจรรวมโดยการใช้งาน เอฟพีจีเอ และสามารถใช้อธิบายฮาร์ดแวร์ ในการออกแบบวงจรรวมที่ใช้กับงานเฉพาะด้านได้

บทที่ 2

หลักการใช้งานเอพฟิจีเอและการจัดเรียงข้อมูล

ในการประยุกต์ใช้งานเอพฟิจีเอในงานจัดเรียงข้อมูลจะต้องทำความเข้าใจทั้งส่วนของซอฟต์แวร์ออกแบบวงจรรวม และฮาร์ดแวร์ของเอพฟิจีเอรวมถึงหลักการจัดเรียงข้อมูล ในบทนี้จะกล่าวถึง ภาษาที่ใช้ และส่วนประกอบภายในเอพฟิจีเอ และการจัดเรียงข้อมูลในวิธีการต่างๆ

2.1 วีเอชดีแอล (VHDL) เป็นชื่อย่อมาจาก Very High Speed Integrated Circuit Hardware Description Language เป็นภาษาที่ใช้ในการออกแบบระบบดิจิทัลและใช้ในการจำลองการทำงานของวงจรมีคุณสมบัติที่สามารถเข้าใจได้ทั้งคนและเครื่อง

2.1.1 ความสามารถของภาษา วีเอชดีแอล

- ภาษาวีเอชดีแอลสามารถใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างผู้ผลิตชิปกับผู้ออกแบบ
- ใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่าง CAE และ CAD Tools เช่น โค้ดวีเอชดีแอล สามารถคอมไพล์โดยใช้ Compiler และ Simulator ได้หลายตัวแตกต่างกัน
- ภาษาวีเอชดีแอลเป็นแบบทั่วไป คือไม่อิงเทคโนโลยีอันใดอันหนึ่งแต่สามารถอิงเทคโนโลยีก็ได้ ในขณะเดียวกันก็สามารถสนับสนุนหลายๆเทคโนโลยี
- สนับสนุนการออกแบบทั้งระบบ Synchronous และ Asynchronous
- สนับสนุนการออกแบบระบบดิจิทัล ในหลายๆเทคนิค เช่น Finite State Machine Algorithmic , Boolean Equation
- ตัวภาษาวีเอชดีแอล สามารถอ่านและทำความเข้าใจได้โดยมนุษย์
- ภาษาวีเอชดีแอล เป็นมาตรฐานรับรองโดย IEEE และ ANSI ทำให้โมเดลที่ออกแบบโดยภาษาวีเอชดีแอล สามารถเคลื่อนย้ายไปยังระบบใดๆ ก็ได้และสามารถนำกลับมาใช้ใหม่ได้
- ภาษาวีเอชดีแอล สนับสนุนรูปแบบการเขียนถึง 3 รูปแบบ ได้แก่ แบบ Behavioral Style , แบบ Structural Style , แบบ Data Flow หรือสามารถเขียนได้ทั้ง 3 รูปแบบ
- สนับสนุนการออกแบบขนาดใหญ่โดยใช้ความสามารถของส่วนประกอบ (Component) เช่น Function , procedure , package
- ไม่จำเป็นต้องศึกษาซอฟต์แวร์จำลองการทำงานเพราะตัวจำลองการทำงาน สามารถเขียนได้จากภาษาวีเอชดีแอล

- สามารถเขียนโมเดลได้ไม่จำกัดขนาดไม่มีข้อกำหนดในตัวภาษาเรื่องขนาดของโมเดล
- สามารถอธิบายตัวแปรที่เกี่ยวกับฟังก์ชันทางด้านเวลา เช่น Propagation Delay , Min-Max Delay, Setup, Holding time, Spike Detection สามารถอธิบายได้โดยในตัวภาษา
- โมเดลที่สร้างด้วยภาษาวีเอชดีแอลนั้น ไม่เพียงอธิบายฟังก์ชันในการทำงานเท่านั้น แต่ยังสามารถอธิบายถึงรายละเอียดของตัวโมเดล เช่น Total Area และ Speed ของโมเดล
- ภาษาวีเอชดีแอล เป็นมาตรฐานที่ใช้โดยบริษัทและผู้ออกแบบหลายๆแห่ง จึงเป็นการง่ายที่จะทำความเข้าใจ ถึงแม้จะมาจากแหล่งต่างๆ
- โมเดลที่สร้างขึ้นสามารถจำลองการทำงานได้
- การอธิบายโมเดลแบบบีแฮฟวิเออร์สามารถสังเคราะห์ไปเป็นระดับเกต (Gate – level) ได้ถ้าทำตามกฎการสังเคราะห์

2.1.2 หลักการสร้างโมเดลโดยใช้ภาษาวีเอชดีแอล

ตัวภาษาวีเอชดีแอล นั้นสนับสนุนหลักการต่างๆ ในการเขียนและแก้ไขวงจรดิจิทัลที่มีความซับซ้อนให้มีประสิทธิภาพซึ่งมีหลักการ ดังนี้

- Top Down Design
- Modularity
- Abstraction
- Information Hiding

Top Down Design ในการพัฒนางจรรวมดิจิทัลขนาดใหญ่ที่มีความซับซ้อน ผู้ออกแบบมักมองให้อยู่ในรูปของ Block Diagram ก่อนที่จะย่อยรูปแบบให้ลึกถึงรายละเอียดซึ่งตัวภาษานั้นอนุญาตให้

- อธิบายการทำงานของแต่ละ Block
- วิเคราะห์การทำงาน (Analyze)
- จัดการแก้ไขและปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการก่อนที่จะลงรายละเอียดให้ลึกลงไป

Modularity เป็นการออกแบบโดยแบ่งฮาร์ดแวร์ออกเป็นส่วนย่อยๆซึ่งปกติการทำงานของฮาร์ดแวร์ใหญ่ๆจะประกอบด้วยฮาร์ดแวร์ย่อยๆ เมื่อเราออกแบบโดยใช้ภาษาวีเอชดีแอลหน้าที่การทำงานต่างๆสามารถอธิบายได้เป็นโมดูล (คล้าย function หรือ Procedure) ซึ่งแสดงส่วนการทำงานอย่างชัดเจน ซึ่งการแยกรูปแบบใหญ่ๆออกเป็น ส่วนย่อยๆจะทำให้ง่ายต่อการเข้าใจ

Abstraction เป็นรูปแบบที่อธิบายการทำงานของตัวแบบมากกว่าที่จะอธิบายว่าจะพัฒนารูปแบบนี้ได้อย่างไร

Information Hiding เป็นรูปแบบที่ซ่อนรายละเอียดในการสร้างโมดูลนั้นๆ โดยไม่ต้องทำให้โมดูลอื่นๆ รู้การทำงานภายในทำให้ง่ายต่อการจัดการและทำความเข้าใจ

2.1.3 รูปแบบการเขียน

การเขียนโมดูลในภาษาวีเอชดีแอล สามารถแบ่งได้ เป็นรูปแบบต่างๆดังนี้

Behavioral Model เรียกอีกอย่างว่า **algorithmic description** เป็นรูปแบบที่บรรยายพฤติกรรมของระบบดิจิทัลซึ่งส่วนนี้คล้ายกับภาษาระดับสูงอย่าง C หรือ PASCAL ในการจำลองการทำงานคำสั่งแต่ละคำสั่งจะถูกดำเนินไปอย่างมีลำดับจากบนลงล่างยกเว้นในกรณีของคำสั่งลูปหรือมีการเรียกใช้โปรแกรมย่อยรูปแบบนี้จะไม่ให้รายละเอียดเกี่ยวกับโครงสร้างของฮาร์ดแวร์แต่จะให้รายละเอียดความสัมพันธ์ระหว่างอินพุต กับ เอาท์พุต ได้ดี

Dataflow Model เรียกอีกอย่างว่า **Register Transfer Level (RTL)** เป็นรูปแบบที่ถูกเขียนขึ้นเพื่อสังเคราะห์วงจรอัตโนมัติ

Structural Model เป็นรูปแบบที่ใช้ในการเชื่อมต่อกันระหว่างอุปกรณ์ต่างๆที่ประกอบกันเป็นวงจรหรือระบบดิจิทัลเป็นการเขียนที่แสดงถึงโครงสร้างของฮาร์ดแวร์

Mixed – Level Model เป็นรูปแบบที่อ่อนตัวของภาษาที่สามารถรวมรูปแบบทั้ง 3 ในการเขียนให้เป็นระบบดิจิทัลได้ ฉะนั้นรูปแบบนี้จึงเป็นการเขียนแบบผสม

2.1.4 โครงสร้างทางภาษา

ในภาษาวีเอชดีแอลนั้นประกอบไปด้วยส่วนต่างๆที่สำคัญและเป็นพื้นฐานของการเขียนรูปแบบระบบคิิตอลที่สำคัญ 4 ส่วนคือ

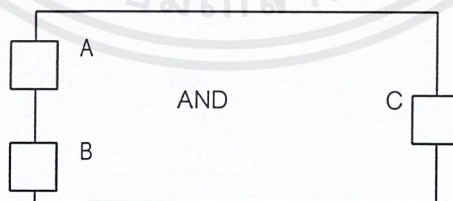
- Entity Design unit
- Architecture design unit
- Package design unit
- Configuration design unit

Entity design unit เป็นส่วนที่ใช้ในการติดต่อระหว่างโลกภายนอกกับตัวแบบมีรูปแบบการเขียนดังนี้

```
ENTITY ชื่อของ entity IS
PORT( กำหนดว่าจะรับสัญญาณเข้าหรือออกและชนิดของสัญญาณ)
END ชื่อของ entity ;
```

ตัวอย่าง

```
ENTITY AND IS
PORT( A,B : IN BIT;
      C : OUT BIT);
END AND;
```



รูปที่ 2.1 แสดงมุมมองการ interface

รูปที่ 2.1 เป็น entity ที่บรรยายอุปกรณ์ที่ชื่อ AND ในส่วนของ entity มีการกำหนดจุดต่อภายใน 3 จุด จุด A กับ B เป็นจุดที่รับสัญญาณเข้ามา และจุด C เป็นจุดที่สัญญาณไหลออก ประเภทของสัญญาณเป็นแบบ BIT คือมี 2 สถานะคือ 1 และ 0

Architecture design unit คือส่วนที่ใช้บรรยายกำหนดพฤติกรรมต่างๆต่อจาก ส่วน entity ที่ได้ออกแบบไว้ซึ่ง ในส่วนนี้จะเป็นส่วนที่แสดงการทำงานของวงจร มีรูปแบบการเขียนดังนี้

```
ARCHITECTURE ชื่อของarchitecture OF ชื่อของ entity IS
BEGIN
END ชื่อของarchitecture ;
```

ตัวอย่าง

```
ARCHITECTURE RTL OF AND IS
BEGIN
    C <= A and B;
END RTL;
```

ในส่วนนี้ได้บรรยายพฤติกรรมของวงจรแบบ Dataflow model การทำงานของวงจร เมื่อมีสัญญาณจาก A และ B มีเข้ามาให้ทำการ and กันแล้วส่งค่าออกไปที่ C ซึ่งในส่วนของ Architecture นี้จะเป็นส่วนที่บรรจุรูปแบบการทำงานของวงจร

Package design unit ข้อมูลต่างๆ ตลอดจนถึงโปรแกรมย่อยที่เป็นประโยชน์ในการ เขียนจะถูกเก็บไว้ในส่วนนี้และข้อมูลเหล่านี้จะถูกนำไปใช้โดย entity หรือ architecture โดยปกติแล้ว package จะถูกแบ่งออกเป็น 2 ส่วน

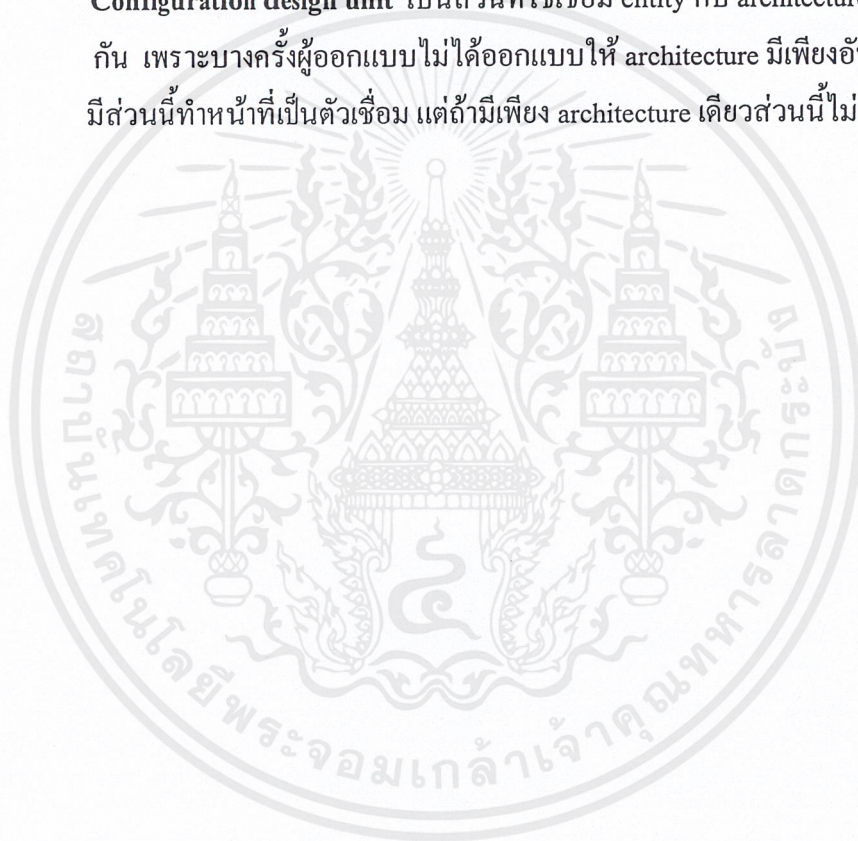
- **package declaration** เป็นส่วนที่มีความสำคัญที่สุดเพราะเป็นส่วนที่กำหนดชื่อของสิ่งที่อยู่ภายใน package เพื่อที่สามารถนำไปใช้จากภายนอกตัว package มีรูปแบบการเขียนดังนี้

```
PACKAGE ชื่อของpackage IS
PACKAGE DECLARATIVE PART
END ชื่อของpackage ;
```

สิ่งที่สามารถประกาศได้ในส่วน PACKAGE DECLARATIVE PART
ได้แก่ type , subtype , signal , constant , component

package body เป็นส่วนที่บรรจุโครงสร้างที่เป็นชุดคำสั่งที่เป็นลำดับที่ใช้ในการบรรยายพฤติกรรมของโปรแกรมย่อยทั้งหลายซึ่งชื่อของโปรแกรมย่อยเหล่านั้นถูกประกาศไว้ใน package declaration

Configuration design unit เป็นส่วนที่ใช้เชื่อม entity กับ architecture เข้าด้วยกัน เพราะบางครั้งผู้ออกแบบไม่ได้ออกแบบให้ architecture มีเพียงอันเดียวจึงมีส่วนนี้ทำหน้าที่เป็นตัวเชื่อม แต่ถ้ามีเพียง architecture เดียวส่วนนี้ไม่ต้องมีก็ได้

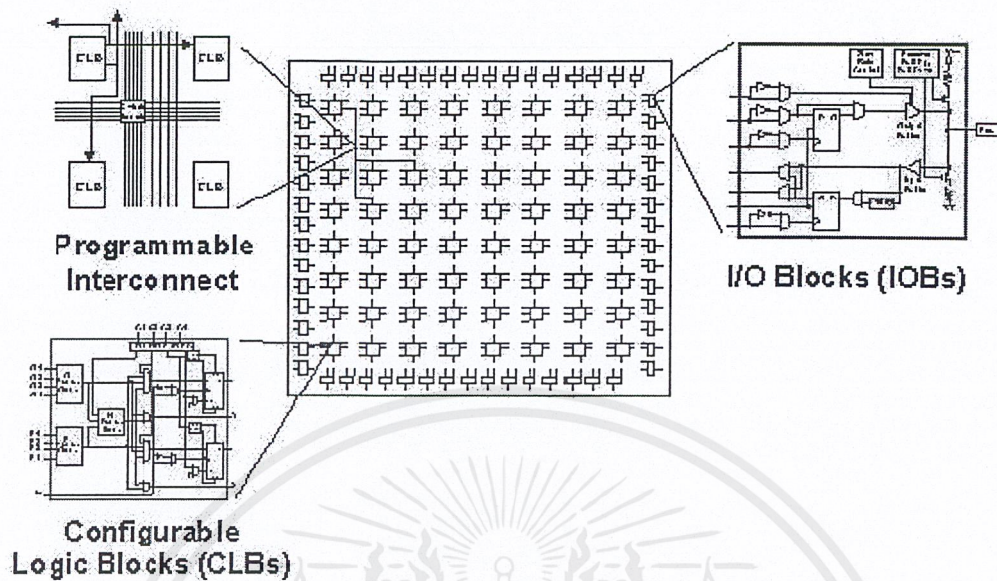


2.2 เอฟพีจีเอ (Field Programmable Gate Array)

เป็นชิปที่มีสนามวงจรถนาคใหญ่อยู่ภายในเราสามารถนำมาใช้และออกแบบวงจรต่างๆได้ตามที่เราต้องการโดยมีโปรแกรมสำเร็จรูปที่ใช้ในการออกแบบอยู่แล้ว มีความเร็วสูง มีสถาปัตยกรรมการออกแบบคล้าย ซีพีแอลดี (CPLD , Complex Programmable Logic Device) แต่มีส่วนประกอบที่ซับซ้อนและมีประสิทธิภาพมากกว่าและมีประสิทธิภาพการทำงาน ปริมาณความหนาแน่นของเกตสูง สามารถจะกำหนดฟังก์ชันการทำงานได้ความต้องการของผู้ใช้โดยผ่านการโปรแกรม เอฟพีจีเอได้รวบรวมข้อดีทั้งหมดของการทำคัสตัมวีแอลเอสไอ (Custom VLSI) มารวมไว้ทั้งหมดได้ แก่การออกแบบการผลิต , ลดเวลาที่จะส่งตัวผลิตภัณฑ์ออกสู่ตลาด ซึ่งเป็นประโยชน์ต่อการผลิตวงจรเป็นอย่างมากนักออกแบบเพียงกำหนดฟังก์ชันการทำงานของวงจร ดังนั้นการออกแบบวงจรโดยใช้ เอฟพีจีเอ สามารถออกแบบและทดสอบภายในเวลาเพียง 2-3 วันเท่านั้น ตรงกันข้ามกับการออกแบบโดยใช้เกตอาเรย์ ซึ่งใช้เวลาหลายสัปดาห์การเปลี่ยนแปลงแก้ไขแบบก็เช่นเดียวกันจากประโยชน์ของ เอฟพีจีเอ ดังกล่าวมาทำให้เกิดการประหยัดค่าใช้จ่ายเป็นอย่างมาก เพราะได้ลดความเสี่ยงในการที่จะต้องแก้ไขตัววงจร การเลื่อนเวลาการออกผลิตภัณฑ์

2.2.1 สถาปัตยกรรมภายในของ เอฟพีจีเอ แบ่งเป็น 3 ส่วน คือ

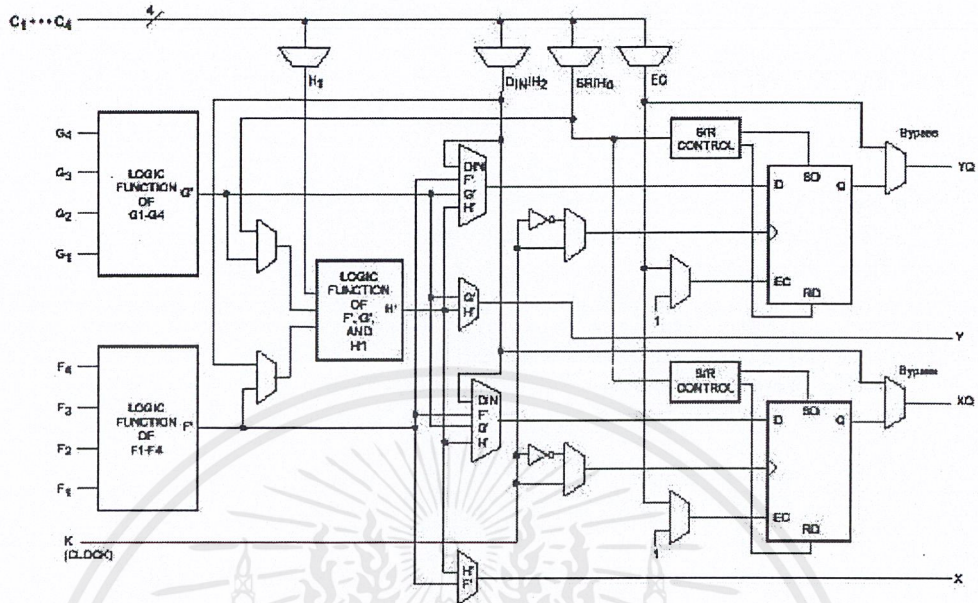
- ซีแอลบี (CLB , Configuration Logic Block)
- ไอโอบี (IOB , Input Output Block)
- ส่วนเชื่อมโยง (Interconnect)



รูปที่ 2.2 สถาปัตยกรรมภายใน เอฟพีเอ ของ XILINX

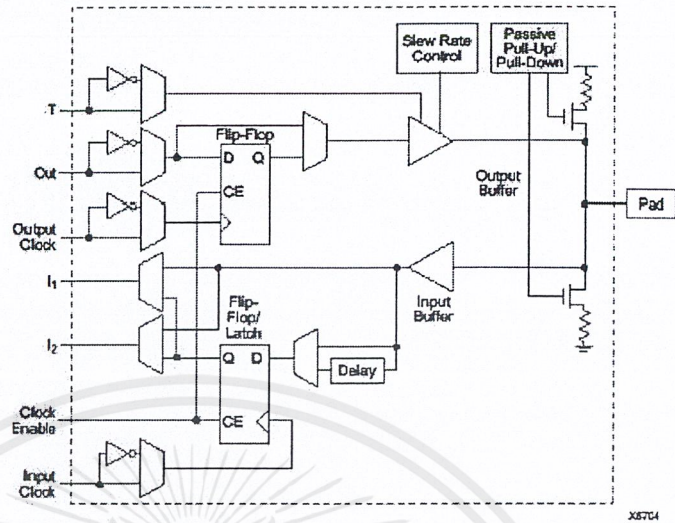
ภายในสถาปัตยกรรมของเอฟพีเอจะมีลักษณะเป็นตารางของลอจิกบล็อก (Logic block) และล้อมรอบไปด้วยบล็อกการเชื่อมต่อของไอโอ การเชื่อมต่อระหว่างซีแอลบี และไอโอบี ทำได้โดยผ่านช่องที่ว่างพาดผ่านระหว่างแถวและคอลัมน์ หน้าที่ของซีแอลบี และไอโอบีแต่ละตัวการเชื่อมต่อภายในถูกกำหนดไว้ในโปรแกรมคอนฟิกูเรชันหรือเก็บไว้ในอีพรอม (EPROM) ภายในแอลซีเอ (LCA , Logic Cell Array) หรือ ซีแอลบีที่ต่อกันเป็นอาร์เรย์ โปรแกรมจะถูกโหลดเข้าสู่แอลซีเอ เมื่อมีการจ่ายไฟ ซึ่งเป็นส่วนหนึ่งของการเริ่มต้น ระบบ (System initialization)

- ซีแอลบี (CLB , Configuration Logic Block) ภายในซีแอลบีแต่ละตัวประกอบด้วยหน่วยของคอมบิเนชันลอจิกที่สามารถโปรแกรมได้ และส่วนของเรจิสเตอร์เก็บข้อมูล ส่วนของวงจรรวมบิเนชันลอจิก



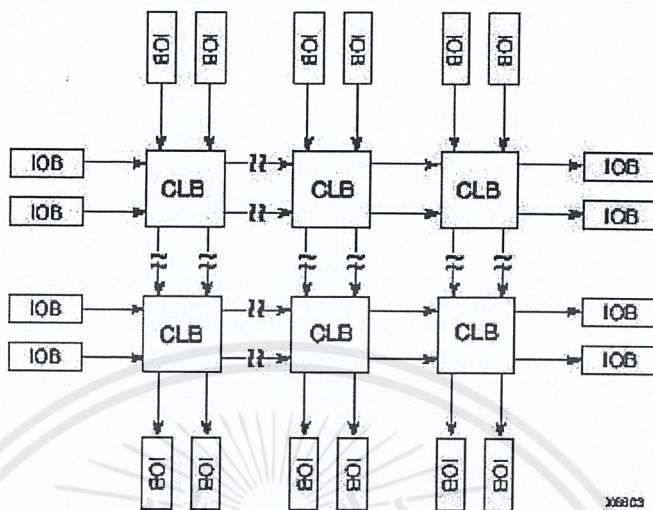
รูปที่ 2.3 โครงสร้างของซีแอลบี

- ไอโอบี (IOB , Input Output Block)เป็นส่วนติดต่อกับวงจรรภายนอกของแอลซีเอสสร้างมาจากส่วนของอุปกรณ์ อินพุต/เอาพุตที่สามารถโปรแกรมได้แต่ละตัวสามารถโปรแกรมได้อย่างอิสระโดยจะให้เป็นอินพุต เอาท์พุตแบบ 3 สถานะหรือ ไอโอแบบสองทิศทางก็ได้โดยอินพุตสามารถโปรแกรมให้รู้จักทั้งระดับสัญญาณทีทีแอลและซีมอสเทรตโฮล ไอโอบีแต่ละตัวมีฟลิปฟลอปสามารถใช้เป็นบัฟเฟอร์สำหรับอินพุตและเอาต์พุต



รูปที่ 2.4 แสดงผังวงจรของไอโอภายในเอฟพีจีเอ

- ส่วนเชื่อมต่อ (Interconnect) ความยืดหยุ่นของการใช้แอลซีเอมาทำเป็นอุปกรณ์ขึ้นอยู่กับการโปรแกรมทรัพยากรต่างๆ ที่อยู่ภายในเข้าด้วยกันการที่จะควบคุมการเชื่อมต่อระหว่างจุดสองจุดภายในชิปเหมือนกับเกตอานะรี่ทั่ว ๆ ไป การเชื่อมต่อภายในแอลซีเอประกอบด้วยเน็ตเวิร์ค 2 ทิศทางคือทางแถวและคอลัมน์ซึ่งวางอยู่ระหว่างซีแอลบี programmable switch จะทำการเชื่อมต่ออินพุตและเอาต์พุตของไอโอบี และซีแอลบีที่จุดต่อร่วมระหว่างแถวกับคอลัมน์สามารถสลับสัญญาณจากเส้นทางไปยังส่วนต่างๆ



รูปที่ 2.5 แสดงส่วนเชื่อมโยงระหว่างไอโอบีกับซีแอลบีของเอฟพีจีเอ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

(Unsorted กับ Sorted) แต่แบบ Exchange Selection ใช้การแลกเปลี่ยนค่าระหว่างค่าต่ำสุดกับค่าที่อยู่ตรงตำแหน่งต้นรายการ

ตัวอย่าง การจัดเรียงข้อมูลแบบ Exchange Selection

14 3 22 9 10 2 7 25

หลังจากผ่านรอบแรกเลข 2 มีค่าต่ำสุดจึงถูกดันมาแทนที่เลข 14 เลข 14 จะไปอยู่แทนตำแหน่งของเลข 2

2 3 22 9 10 14 7 25

ในรอบที่ 2 เลข 3 จะมีค่าต่ำสุดในรายการที่เหลือ

2 3 22 9 10 14 7 25

ในรอบสุดท้ายผลลัพธ์จะเป็น

2 3 7 9 10 14 22 25

2.3.3 การจัดเรียงแบบ Insertion Sort

เป็นการจัดเรียงข้อมูลที่พัฒนามาจาก Selection Sort แทนที่เราจะอ่านข้อมูลทั้งหมดแล้วค่อยจัดเรียง การจัดเรียงแบบ Insertion Sort นั้นจะเรียงลำดับข้อมูลไปเรื่อยๆ ตามที่แต่ละข้อมูลที่ถูกรับเข้ามา

ตัวอย่าง การจัดเรียงข้อมูลแบบ Insertion Sort

14 3 22 9 10 2 7 25

Unsorted List

ในรอบแรกข้อมูลที่อ่านเข้ามาคือ 14 นำไปเก็บไว้ในส่วนที่จัดเรียงข้อมูล

3 22 9 10 2 7 25 14

Unsorted List

Sorted List

ในรอบที่ 2 อ่านค่าเลข 3 เข้ามา แล้วนำไปจัดเรียงเก็บไว้ในส่วนที่จัดเรียงข้อมูล

22 9 10 2 7 25 14 3

Unsorted List

Sorted List

ดำเนินการอ่านข้อมูลจาก Unsorted List ไปแทรกใน Sorted List ต่อไปเรื่อยๆ จะได้ผลลัพธ์ในรอบสุดท้ายเป็น

2 3 7 9 10 14 22 25

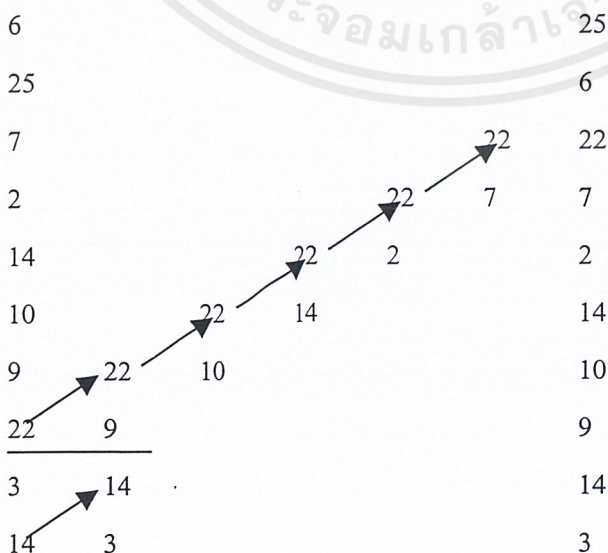
2.3.4 การจัดเรียงข้อมูลแบบ Bubble Sort

การจัดเรียงข้อมูลแบบนี้จะคล้ายๆกับ การจัดเรียงแบบ Exchange Selection Sort คือ เป็นการเปรียบเทียบค่าข้อมูลที่ละคู่ และแลกเปลี่ยนค่าข้อมูลแต่ละคู่ตามตำแหน่งที่เหมาะสมเทคนิคการจัดเรียงแบบ Bubble Sort จะเป็นวิธีการที่ทำย้อนหลังจากท้ายสุดของข้อมูลย้อนกลับมายังต้นข้อมูล โดยจะเปรียบเทียบคู่ของข้อมูลระหว่างตำแหน่งท้ายสุดกับตัวที่อยู่ถัดขึ้นมาข้างหน้าทีละตัวตามลำดับ หากข้อมูลใดมีค่ามากกว่าก็จะลอย (float) เหมือนกับฟองสบู่ (bubble) ออกมาอยู่ข้างหน้า และจะทำวนเช่นนี้เป็นลูป จนกว่าค่าของข้อมูลทุกตัวจะถูกเรียงลำดับ

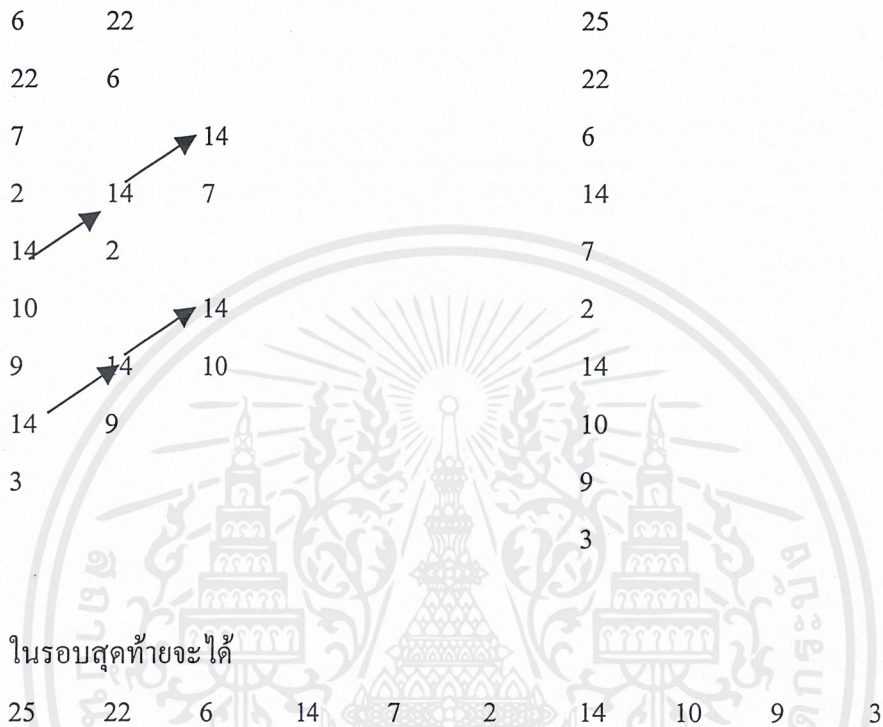
ตัวอย่าง การจัดเรียงข้อมูลแบบ Bubble Sort

6 25 7 2 14 10 9 22 3 14

เพื่อที่จะแสดงการลอยตัวให้เห็นเด่นชัดจะจัดเรียงข้อมูลเป็นแนวตั้งสังเกตว่าในรอบแรกค่าที่จะลอยตัวขึ้นมาหลังจากการเปรียบเทียบคือ 25 เพราะจากตำแหน่งล่างสุดเปรียบเทียบ 14 กับ 3 ได้ค่า 14 ลอยตัวขึ้นมาเมื่อเอา 14 เปรียบเทียบกับ 22 ค่า 22 จะลอยขึ้นมาและค่า 22 จะลอยขึ้นมาตามลำดับจนถึงการเปรียบเทียบกับค่า 25 จะได้ค่า 25 ลอยตัวขึ้นมา



ในรอบที่ 2 เปรียบเทียบค่าที่เหลือ (ตัดค่า 25 ออกไปเพราะวางถูกตำแหน่งแล้ว)
จัดการเปรียบเทียบจากล่างสุดขึ้นไปบนจะได้ 22 ลอยตัวขึ้นมา



2.3.5 การจัดเรียงข้อมูลแบบ Quick Sort

แนวคิดพื้นฐานของการจัดเรียงแบบนี้ คือการใช้ผลของการเปรียบเทียบในหนึ่งรอบเป็นตัวกำหนดเงื่อนไขในการเปรียบเทียบครั้งต่อไปตามลำดับ ในกระบวนการเปรียบเทียบจะแบ่งข้อมูลออกเป็น 2 ส่วนโดยย้ายค่าของชุดข้อมูลทำให้ด้านหนึ่งมีค่าสูงกว่าค่าของชุดข้อมูลอีกด้าน

ตัวอย่าง การเรียงลำดับแบบ Quick Sort

6 25 7 2 14 10 9 22 3 14

จุดมุ่งหมายของการเปรียบเทียบจะแบ่งข้อมูลออกเป็น 2 ส่วนเอาค่าที่สูงกว่าไว้ด้านล่างและต่ำกว่าไว้ด้านบน เริ่มด้วยการเปรียบเทียบ 6 กับ 14 ผลลัพธ์คือไม่ต้องแลกเปลี่ยน เพราะ 6 น้อยกว่าการเปรียบเทียบถัดไประหว่าง 6 กับ 3 ผลลัพธ์คือ 3 ย้ายไปด้านบน 6 ย้ายมาด้านล่างถัดไปเปรียบเทียบ 25 กับ 6 ผลลัพธ์ 6 ถูกย้ายไปด้านบน 25 ถูกย้ายไปด้านล่าง ต่อมา 6 จะถูกเปรียบเทียบกับ 22,9,10,14 ตามลำดับ ผลลัพธ์คือทุกตัวคงเดิม จนมาถึง 6 กับ

2 จะได้ว่า 2 ถูกย้ายขึ้น 6 ลงล่าง การเปรียบเทียบครั้งสุดท้ายคือ 6 กับ 7 ผลลัพธ์สลับค่ากัน ตรงจุดนี้ถือว่าการสิ้นสุดการย้ายค่าในรอบแรกสรุปได้ผลลัพธ์เป็นครึ่งบนเลขตั้งต้นในรอบนี้มี 3 ค่าคือ 3,2,6 ส่วนครึ่งล่างมี 7 ค่า คือ 7,14,10, 9,22,25,14 เมื่อจบรอบแรกแล้ว จะเริ่มต้นทำการเปรียบเทียบสลับค่าไปมาโดยใช้ตรรกแบบเดิมของส่วนครึ่งบน และส่วนครึ่งล่างไปเรื่อยๆจากอัลกอริทึมการจัดเรียงข้อมูลที่ได้แสดงไว้แต่ละอัลกอริทึมก็มีข้อดีและข้อเสียที่แตกต่างกันดังได้แสดงไว้ในตารางข้างล่าง



ตารางที่ 2.3 เปรียบเทียบการทำงานของแต่ละอัลกอริทึม

อัลกอริทึม	ข้อดี	ข้อเสีย
Insertion Sort	<ul style="list-style-type: none"> - เข้าใจง่าย - ทำงานได้เร็วสำหรับชุดข้อมูลขนาดเล็ก 	<ul style="list-style-type: none"> - ทำงานช้ามากสำหรับชุดข้อมูลขนาดใหญ่
Selection Sort	<ul style="list-style-type: none"> - เข้าใจง่าย - ทำงานได้เร็วสำหรับชุดข้อมูลขนาดเล็ก 	<ul style="list-style-type: none"> - ทำงานช้ามากสำหรับชุดข้อมูลขนาดใหญ่
Exchange Sort	<ul style="list-style-type: none"> - เข้าใจง่าย - ทำงานได้เร็วสำหรับชุดข้อมูลขนาดเล็ก 	<ul style="list-style-type: none"> - ทำงานช้ามากสำหรับชุดข้อมูลขนาดใหญ่
Bubble Sort	<ul style="list-style-type: none"> - ทำงานได้เร็วมากสำหรับชุดข้อมูลที่มีการเรียงลำดับมาเกือบทั้งหมด 	<ul style="list-style-type: none"> - ทำงานได้ช้ามากสำหรับชุดข้อมูลที่ไม่มีการเรียงลำดับเลย
Quick Sort	<ul style="list-style-type: none"> - ทำงานได้เร็วกับข้อมูลขนาดใหญ่ 	<ul style="list-style-type: none"> - จะมีความยุ่งยากถ้ามีข้อมูลซ้ำกันมากๆ

2.4 การประมวลผลแบบขนาน (Parallel Processing)

การประมวลผลแบบขนานเริ่มต้นจากการพัฒนาเทคนิคประมวลผลแบบเรียงลำดับ (Sequential Computing) เช่น Divide et impera , Vectorization , Vector iteration เป็นต้น มีวัตถุประสงค์เพื่อเพิ่มประสิทธิภาพและความเร็วให้สูงขึ้นโดยใช้หน่วยประมวลผลจำนวนมาก

2.4.1 การจัดเรียงข้อมูลแบบขนาน (Parallel Sort)

การจัดเรียงข้อมูลแบบขนานมีความสำคัญมากสำหรับผู้ออกแบบระบบแบบขนาน อัลกอริทึมที่ใช้ในการจัดเรียงเป็นแบบ Internal Sort ในโครงการนี้ทำการศึกษาอัลกอริทึม 2 แบบคือ

- Parallel Exchange Sort
- Parallel Bitonic Merge Sort

- Parallel Exchange Sort

อัลกอริทึมนี้ใช้โครงสร้างการเชื่อมต่อ (Interconnection structures) แบบ Mesh 2 มิติ (Unit แต่ละส่วนสามารถนำมาต่อกับ Unit ข้างเคียงได้ 2 Unit) มีอัลกอริทึมที่ใช้จัดเรียงและโครงสร้างดังรูปที่ 2.6

for k = 1 to n/2 do ; n คือ จำนวนข้อมูล , I คือ ค่าเริ่มต้นครั้งแรก

for I = 1 to n/2 do in parallel

if $a_i > b_i$ then ; a_i และ b_i คือ register ของ unit ที่ i

INTERCHANGE[a_i, b_i]

end if

SHIFTR[1,1:n/2]

For I = 1 to n/2 do in parallel

if $a_i > b_i$ then

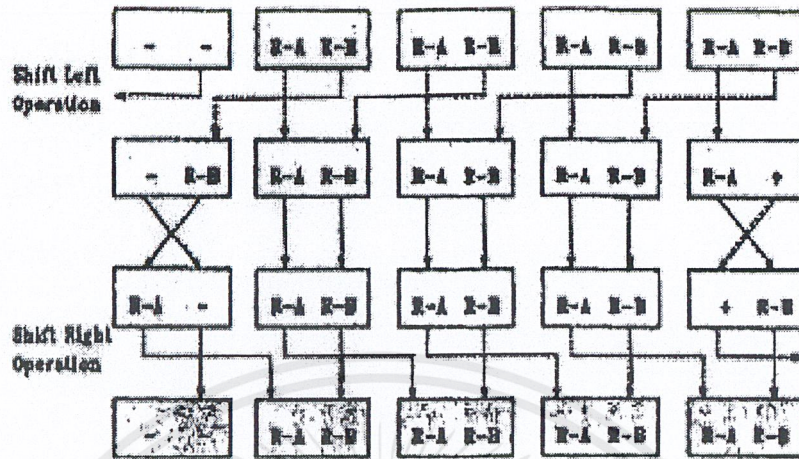
INTERCHANGE[a_i, b_i]

end if

end for

SHIFTL[1,1:n/2]

end for



รูปที่ 2.6 แสดง โครงสร้างของ Exchange – Sort

- Parallel Bitonic Merge Sort

อัลกอริทึมนี้ใช้โครงสร้างการเชื่อมต่อ(Interconnection structures) แบบ Shuffle – Exchange มีอัลกอริทึมที่ใช้จัดเรียงและ โครงสร้างดังรูปที่ 2.7

For all P_i where $0 < i < n - 1$ do

If even(i) then

$T \leftarrow \text{exchange}(A)$ ‘ A คือ จำนวนข้อมูลที่จะจัดเรียง

If $m = 0$ then ‘ sort low to high

$\text{Exchange}(A) \leftarrow \max(A, T)$ ‘ T คือ ค่าที่ได้จากยูนิตที่ทำงานเสร็จ

$A \leftarrow \min(A, T)$

Else

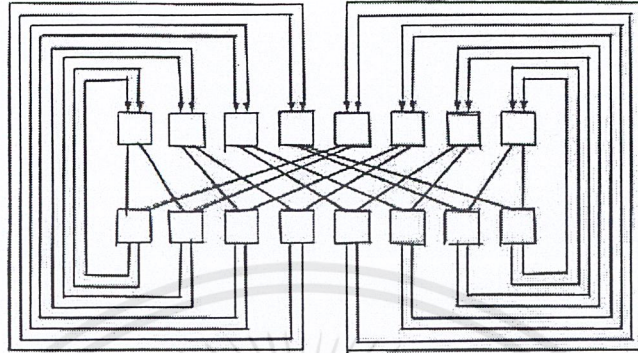
$\text{Exchange}(A) \leftarrow \min(A, T)$

$A \leftarrow \max(A, T)$

End if

End if

End for



รูปที่ 2.7 แสดงโครงสร้างของ Bitonic Merge Sort

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

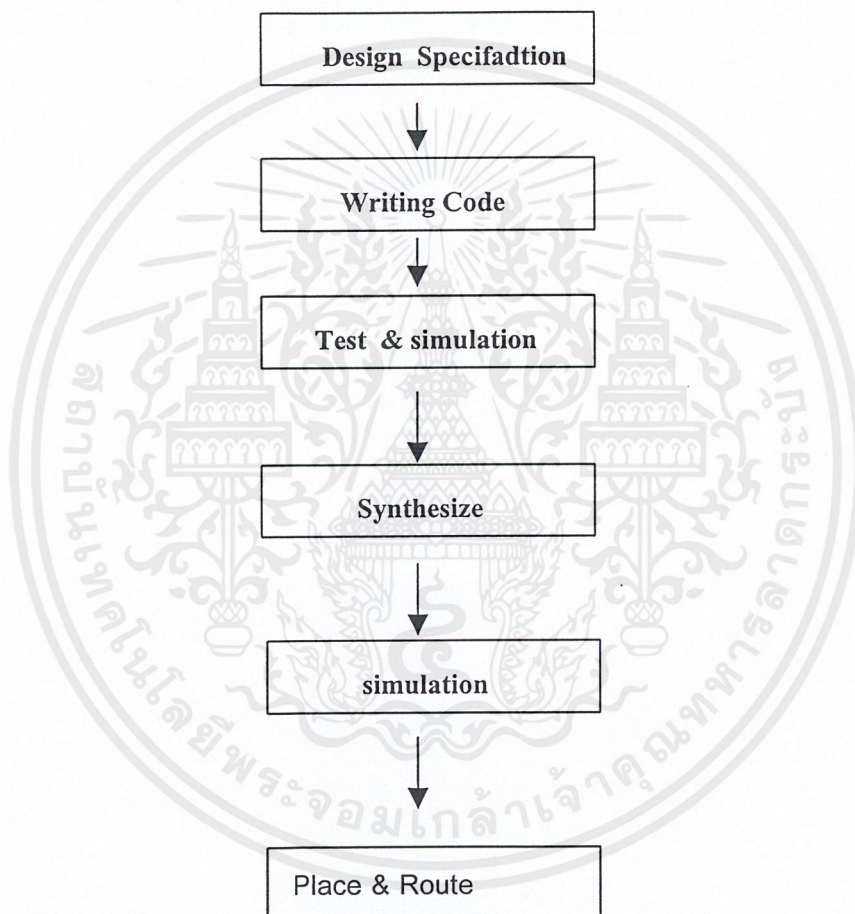
บทที่ 3

การดำเนินงานและการวิจัย

ในบทนี้จะกล่าวถึงขั้นตอนการทำงานใน โครงการตั้งแต่ขั้นตอนการออกแบบตลอดจนรายละเอียดของแต่ละชิ้นส่วน (component) ที่ออกแบบรวมถึงการทดสอบ

3.1 ขั้นตอนการออกแบบ

ในโครงการนี้ทำการศึกษาและออกแบบวงจรจัดเรียงข้อมูลแบบขนานโดยใช้ชิปของ XILINX ในตระกูล SpartanII ชิปในตระกูลนี้มีความจุตั้งแต่ 15,000 ถึง 200,000 เกต โครงการนี้เลือกใช้ชิปเบอร์ XC2S200 ซึ่งมีความจุ 200,000 เกตมีไอโอให้ใช้งานถึง 168 ขา และเลือก อัลกอริทึม 2 แบบในการศึกษาคือ Exchange sort และ Bitonic Merge sort ดังนั้นสิ่งที่ป็นข้อกำหนดในการออกแบบคือ ศึกษาอัลกอริทึมของการจัดเรียงแล้วแปลงออกมาให้อยู่ในรูปวงจร เมื่อสามารถแปลงให้อยู่ในรูปของวงจรได้แล้วจึงทำการแยกออกเป็นส่วนประกอบย่อยๆ สิ่งเหล่านี้คือข้อกำหนดในการสร้าง เมื่อเราแยกออกเป็นชิ้นส่วน (component) ย่อยๆแล้วจึงนำชิ้นส่วนเหล่านี้มาเข้าสู่ขั้นตอนการเขียนโค้ดให้อยู่ในรูปแบบภาษาวีเอชดีแอล เมื่อเขียนโค้ดแต่ละชิ้นส่วนเสร็จจึงนำโค้ดเหล่านี้เข้าสู่ขบวนการทดสอบเพื่อดูว่าโค้ดนั้นสามารถทำงานตามที่ออกแบบไว้ได้อย่างถูกต้องหรือไม่ ถ้าไม่ถูกต้อง ต้องนำโค้ดกลับไปแก้ไขจากนั้นนำมาทดสอบอีกจนแน่ใจว่าสามารถทำงานได้อย่างสมบูรณ์ เมื่อผ่านขั้นตอนการทดสอบโค้ดแล้วจากนั้นนำโค้ดเข้าสู่ขบวนการสังเคราะห์ (synthesis) เพื่อให้ออกมาอยู่ในรูปของวงจรจากนั้นนำโค้ดไปทดสอบอีกครั้ง ว่าสามารถทำงานได้อย่างถูกต้องหรือไม่ ถ้าทำงานได้ถูกต้องจึงเข้าสู่ขั้นตอนการ Place&Route เพื่อบันทึกลงในชิปเอฟพีจีเอ รูปที่ 3.1 แสดง Flow chat ขั้นตอนการทำงาน



รูปที่ 3.1 แสดงขั้นตอนการออกแบบ

3.2 การกำหนดการออกแบบ(Design Specification)

รายละเอียดในส่วนนี้แบ่งออกเป็น 2 ส่วนคือ

- Parallel exchange sort
- Parallel bionic merge sort

- Parallel exchange sort

จากอัลกอริทึมนี้แบ่งการทำงานออกเป็นส่วนต่างๆดังนี้

Top_Exchange เป็นยูนิิตที่ประกอบด้วยส่วนคอนโทรลและส่วนจัดเรียง ดังนั้นในการเขียน โค้ดจะสร้าง Entity ขึ้นมาตัวหนึ่งและใช้คำสั่ง PORT MAP เชื่อมทั้ง 2 ส่วนเข้าด้วยกัน

entity top_exchange is

port(

clock: in std_logic;

reset: in std_logic;

ctrl: in std_logic;

unit1a,unit1b,unit2a,unit2b,unit3a,unit3b,unit4a,unit4b : in

std_logic_vector

(3 downto 0);

out1a,out1b,out2a,out2b,out3a,out3b,out4a,out4b : out std_logic_vector(3

downto 0)

);

end top_exchange;

Control Unit ยูนิตตัวนี้ทำงานที่ขอบขาขึ้นของสัญญาณนาฬิกาและเมื่อสัญญาณควบคุมมีลอจิกเป็น 1 จะสั่งให้ตัวจัดเรียงอ่านข้อมูลเข้ามาและทำการจัดเรียง

```
entity ctrl_load is
port(
    clock:      in std_logic;
    reset:      in std_logic;
    ctrl:       in std_logic;
    x,y :       out std_logic
);
end ctrl_load;
```

Load ยูนิตตัวนี้ทำงานที่ขอบขาขึ้นของสัญญาณนาฬิกาและเมื่อสัญญาณควบคุมมีลอจิกเป็น 1 ประกอบด้วยส่วนเปรียบเทียบข้อมูลจำนวนมากซึ่งทำหน้าที่จัดเรียงข้อมูล ในการเขียนโค้ดจะสร้าง Entity ขึ้นมาตัวหนึ่งและใช้คำสั่ง PORT MAP เชื่อมส่วนเปรียบเทียบข้อมูลทั้งหมดเข้าด้วยกัน การเชื่อมสัญญาณในส่วนนี้จะทำให้เกิดการแลกเปลี่ยนข้อมูล การเลื่อนข้อมูลทางซ้ายและทางขวา

```
entity load is
port(
    clk,x,y : in std_logic; -- control
    ctrl    : in std_logic;
    dat     : in std_logic_vector(31 downto 0);
    d_out   : out std_logic_vector(31 downto 0)
);
end load;
```

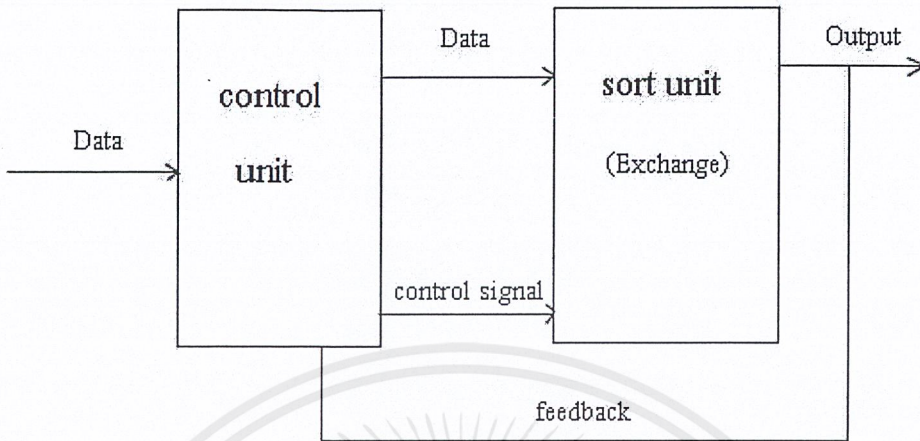
Comparator Unit ยูนิต์ตัวนี้ทำงานที่ขอบขาขึ้นของสัญญาณนาฬิกาและเมื่อสัญญาณควบคุมมีลอจิกเป็น 1 จะรับข้อมูลเข้ามาที่ตะกุ่มแล้วทำการเปรียบเทียบและส่งผลออกมาภายใน 1 สัญญาณนาฬิกา

```
entity compare is
port(
    dat1,dat2 : in std_logic_vector(3 downto 0);
    start : in std_logic;
    max,min : out std_logic_vector(3 downto 0)
);
end compare;
```

Temp register รีจิสเตอร์ตัวนี้เป็นรีจิสเตอร์ขนาด 4 บิตทำงานที่ขอบขาขึ้นของสัญญาณนาฬิกาตั้งนั้นในการเขียนโค้ด

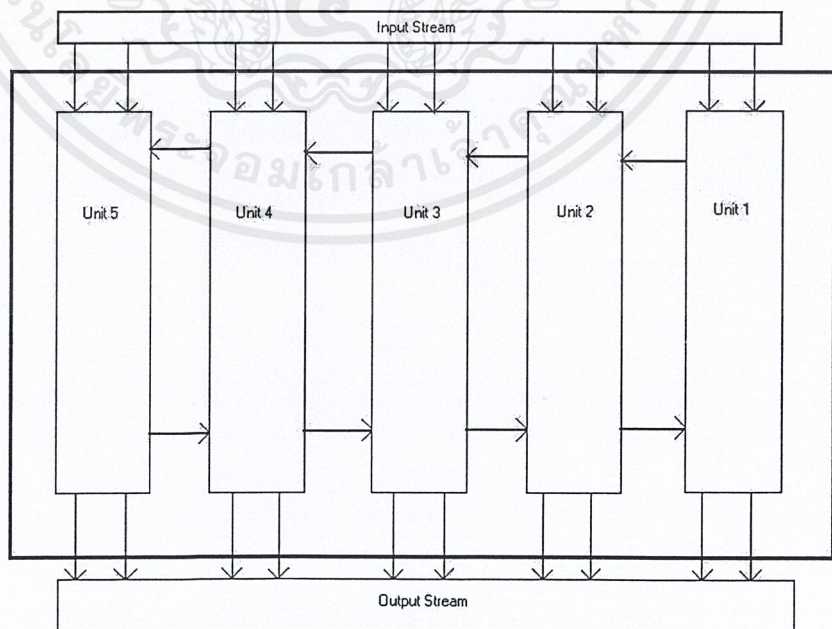
```
entity temp is
port(dat10 : in std_logic_vector(3 downto 0);
    control: in std_logic;
    reg : out std_logic_vector(3 downto 0)
);
end temp;
```

เมื่อทำการออกแบบแต่ละส่วนเสร็จจากนั้นนำมาประกอบกันดังรูป 3.2



รูปที่ 3.2 แสดงโครงสร้างของ Parallel Exchange sort

จากรูปที่ 3.2 ภายในประกอบด้วย หน่วยหลัก 2 ส่วน คือ ส่วนควบคุม และ ส่วนจัดเรียง ในส่วนจัดเรียงนั้นประกอบด้วย unit ย่อยทั้งหมด 5 unit และสามารถรับข้อมูลได้ครั้งละ 8 ค่า โดยที่ unit ที่ 5 จะเป็น unit ที่ใช้พักข้อมูล และ unit ที่ 1 ก็มีส่วนพักข้อมูลอยู่บางส่วน ส่วน unit ที่ 2 ถึง 4 จะไม่มีส่วนพักข้อมูลอยู่เลยดังรูปที่ 3.3



รูปที่ 3.3 แสดงโครงสร้างภายในของส่วนจัดเรียง (Exchange sort)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- **Parallel bitonic merge sort**

จากอัลกอริทึมนี้แบ่งการทำงานออกเป็นส่วนต่างๆดังนี้

Top_Bitonic เป็นยูนิตที่ประกอบด้วยส่วนคอนโทรลและส่วนจัดเรียงดังนั้นในการเขียนโค้ดจะสร้าง Entity ขึ้นมาตัวหนึ่งและใช้คำสั่ง PORT MAP เชื่อมทั้ง 2 ส่วนเข้าด้วยกัน

```
entity top_bitonic is
port (
clock:   in std_logic;
reset:   in std_logic;
ctrl:    in std_logic;
unit1a,unit1b,unit2a,unit2b,unit3a,unit3b,unit4a,unit4b : in
std_logic_vector(3 downto 0);
out1a,out1b,out2a,out2b,out3a,out3b,out4a,out4b : out
std_logic_vector(3 downto 0)
);
end top_bitonic;
```

Control Unit ยูนิตตัวนี้ทำงานที่ขอบขาขึ้นของสัญญาณนาฬิกาและเมื่อสัญญาณควบคุมมีลอจิกเป็น 1 จะสั่งให้ตัวจัดเรียงอ่านข้อมูลเข้ามาและทำการจัดเรียง

```
entity ctrl_load_bi is
port(
clock:      in std_logic;
reset:      in std_logic;
ctrl:       in std_logic;
x,y : out std_logic
);
end ctrl_load_bi;
```

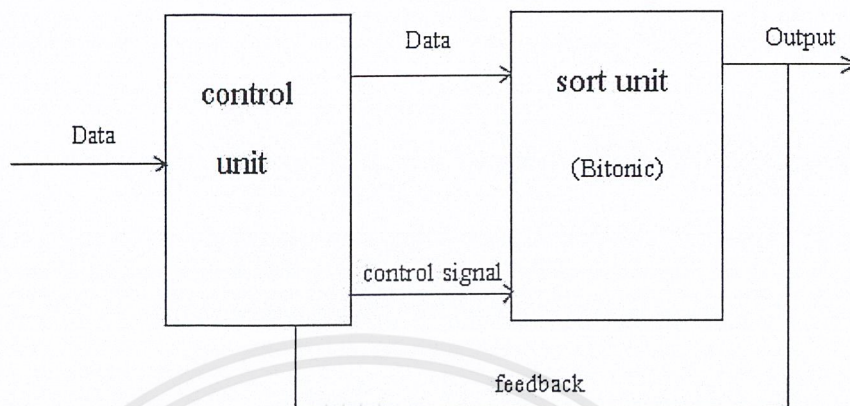
Load ยูนิตตัวนี้ทำงานที่ขอบขาขึ้นของสัญญาณนาฬิกาและเมื่อสัญญาณควบคุมมีลอจิกเป็น 1 ประกอบด้วยส่วนเปรียบเทียบข้อมูลจำนวนมากซึ่งทำหน้าที่จัดเรียงข้อมูล ในการเขียนโค้ดจะสร้าง Entity ขึ้นมาตัวหนึ่งและใช้คำสั่ง PORT MAP เชื่อมส่วนเปรียบเทียบข้อมูลทั้งหมดเข้าด้วยกัน การเชื่อมสัญญาณในส่วนนี้จะทำให้เกิดการแลกเปลี่ยนข้อมูล และการป้อนข้อมูลกลับ

```
entity load_bi is
port(
  clk,x,y : in std_logic; -- control
  ctrl_u : in std_logic;
  dat : in std_logic_vector(31 downto 0);
  d_out : out std_logic_vector(31 downto 0)
);
end load_bi;
```

Comparator Unit ยูนิตตัวนี้ทำงานที่ขอบขาขึ้นของสัญญาณนาฬิกาและเมื่อสัญญาณควบคุมมีลอจิกเป็น 1 จะรับข้อมูลเข้ามาที่ละคู่แล้วทำการเปรียบเทียบและส่งผลออกมาภายใน 1 สัญญาณนาฬิกา

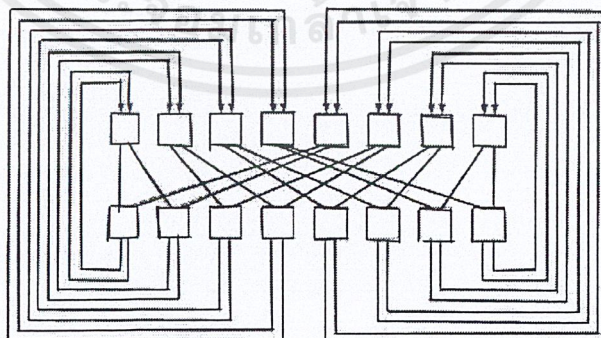
```
entity compare is
port(
  dat1,dat2 : in std_logic_vector(3 downto 0);
  start : in std_logic;
  min,max : out std_logic_vector(3 downto 0)
);
end compare;
```

เมื่อออกแบบแต่ละ ส่วนเสร็จนำมาประกอบกันดังรูปที่ 3.4



รูปที่ 3.4 แสดง โครงสร้างของ Bitonic Merge Sort

จากรูปที่ 3.4 ภายในประกอบด้วย หน่วยหลัก 2 ส่วน คือ ส่วนควบคุม และ ส่วนจัดเรียง ในส่วนจัดเรียงนั้นภายในประกอบด้วย Comparator unit จำนวน 8 unit และสามารถรับข้อมูลได้ครั้งละ 8 ชุดดังรูปที่ 3.5



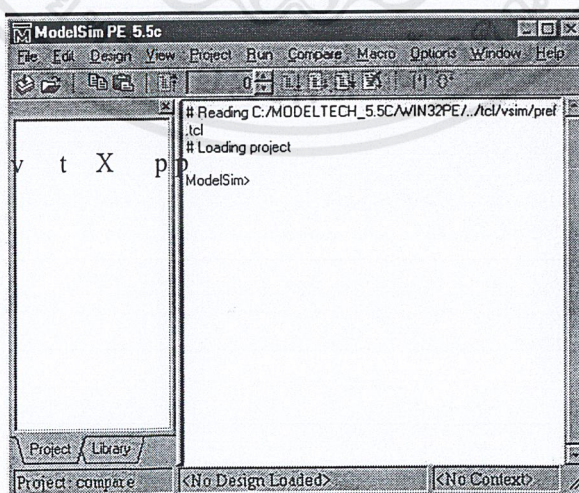
รูปที่ 3.5 แสดง โครงสร้างภายในของส่วนจัดเรียง (Bitonic sort)

3.3 กรรมวิธีทดสอบการทำงาน

ในการทดสอบการทำงานของวงจรที่ได้ออกแบบจะใช้ซอฟต์แวร์ในการจำลองการทำงานของโค้ดเมื่อได้ผลตรงตามที่ต้องการจึงนำโค้ดไปสังเคราะห์จากนั้นนำไปจำลองการทำงานอีกครั้งถ้าได้ผลตรงตามที่ต้องการจึงนำไปโปรแกรมลงชิปซอฟต์แวร์ที่ใช้ในโครงงานนี้ประกอบด้วย

- โมเดลซิม (Model Sim)
- ลีโอนาดอสเปกตรัม (Leonardo Spectrum)
- อิมแพค (Impact)

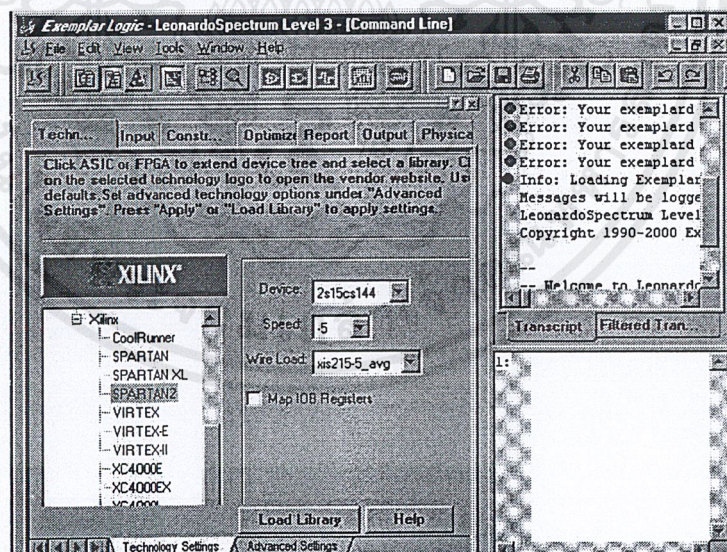
โมเดลซิม (model sim) เป็นโปรแกรมที่ใช้ในการคอมไพล์โค้ดที่เขียนขึ้นตรงตามหลักไวยากรณ์ของภาษาหรือไม่และยังสามารถจำลองการทำงานของโค้ดที่เขียนขึ้นสามารถทำงานได้ตรงตามที่ต้องการหรือไม่ โดยจะแสดงผลให้ผู้ออกแบบดูในรูปของ Wave form เพื่อตรวจสอบความถูกต้องของโค้ด และยังให้ผู้ใช้สามารถตั้ง Force input signal เพื่อกำหนดสัญญาณอินพุตให้กับตัววงจรได้ หรือสร้างเป็น Macro file เพื่อเก็บ Test pattern เอาไว้สำหรับจำลองการทำงาน ในการจำลองการทำงานนั้นผู้ใช้สามารถเลือกที่จะทดสอบได้ในแบบ Single-Step นั่นคือผู้ใช้สามารถทดสอบการทำงานของโค้ดได้ที่ละบรรทัดว่าหลังจากทำคำสั่งในบรรทัดนั้นไปแล้วจะมีผลกระทบอย่างไรต่อตัวแปร (Variable) หรือสัญญาณในวงจร (signal)



รูปที่ 3.6 แสดงหน้าจอโปรแกรม Modelsim

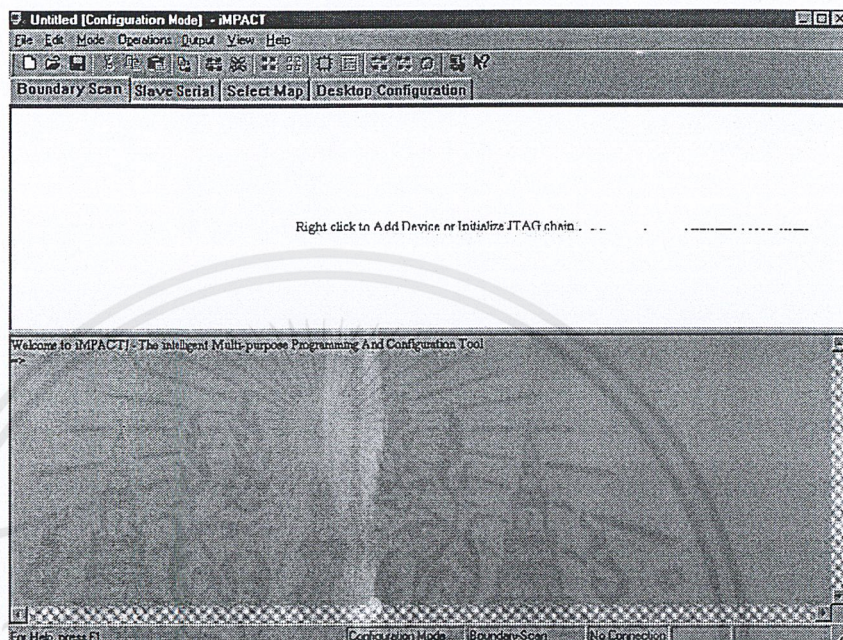
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ลีโอนาโดสเปกตรัม (Leonardo Spectrum) เป็นโปรแกรมที่ใช้ในการสังเคราะห์วงจร (การ สังเคราะห์วงจร คือการแปลงโค้ดที่ผ่านการตรวจสอบความถูกต้องของการทำงานให้เป็นวงจรในระดับ Gate-level) ซึ่งกระบวนการสังเคราะห์นั้นผู้ใช้จะต้องกำหนดว่าจะ สังเคราะห์ลงฮาร์ดแวร์ตัวใด และยังสามารถกำหนดรูปแบบของการสังเคราะห์ ได้ว่าต้องการวงจรแบบมีจำนวนเกตน้อยที่สุดหรือต้องการวงจรแบบมีความเร็วสูงสุด ผู้ใช้สามารถเลือกเอาท์พุทของการสังเคราะห์ ได้หลายแบบเช่นสร้าง ไฟล์นามสกุล. VHDL , ไฟล์นามสกุล.V , ไฟล์นามสกุล.XNF หากเลือกเอาท์พุทเป็นไฟล์. VHDL แล้วจะได้โค้ดในระดับ Gate-level ซึ่งผู้ใช้สามารถนำไปจำลองการทำงานได้ ไฟล์ที่ได้จากการสังเคราะห์นี้ยังทำงานเหมือนกับโค้ดในระดับ Behavior หรือไม่ โปรแกรม Leonardo อนุญาตให้ผู้ใช้สามารถกำหนดค่า Timing Constrain ของวงจรได้ หากวงจรที่สร้างไม่สามารถบรรจลงในฮาร์ดแวร์ที่กำหนดไว้โปรแกรมจะแจ้งให้ทราบและพยายามแมปกับอุปกรณ์ตัวที่มีความจุสูงกว่าซึ่งอยู่ในตระกูลเดียวกันให้



รูปที่ 4.4 แสดงหน้าจอ โปรแกรม Leonardo Spectrum

อิมแพค (Impact) เป็นโปรแกรมที่ใช้ในการโปรแกรมข้อมูลที่อยู่ในรูปของ Bit file ลงสู่ชิป โดยใช้สาย Parallel cable ผ่านทาง Parallel Port



รูปที่ 4.8 แสดงหน้าจอโปรแกรม Impact

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

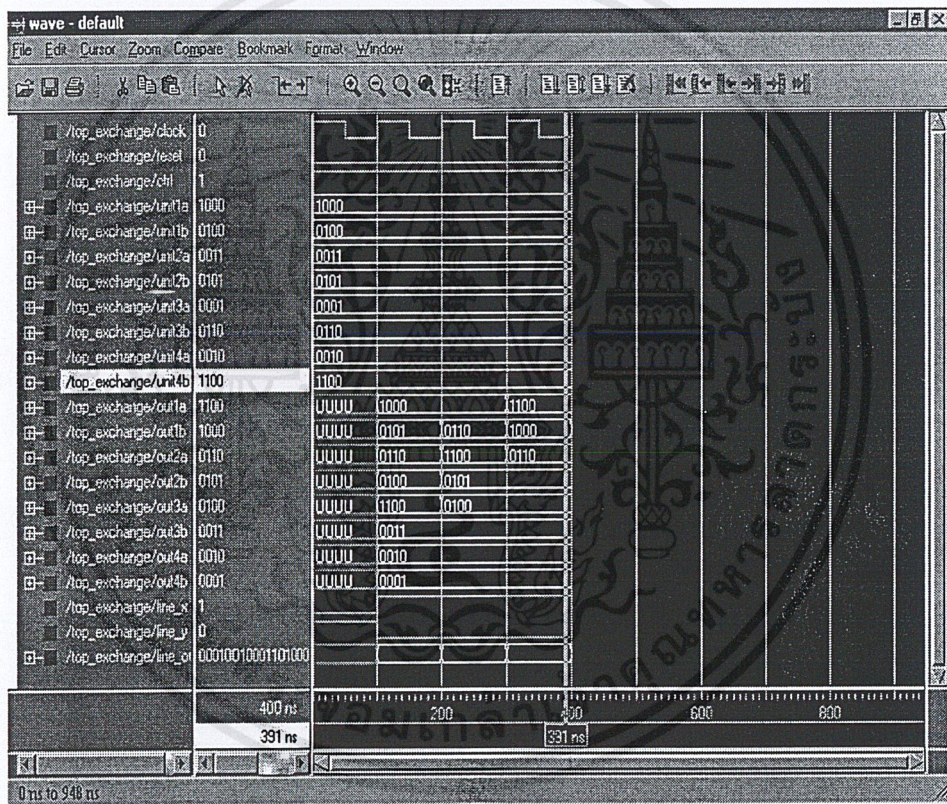
บทที่ 4

ผลการทดสอบการทำงานของวงจร

4.1 ผลการจำลองการทำงานจากโค้ด

ในการทดสอบนี้ใช้การป้อนค่าเข้าทาง input และดูผลจาก output ที่ออกมาเปรียบเทียบกับ input ที่ป้อนเข้าไปว่าการจัดเรียงถูกต้องหรือไม่

- ผลจาก Parallel Exchange sort

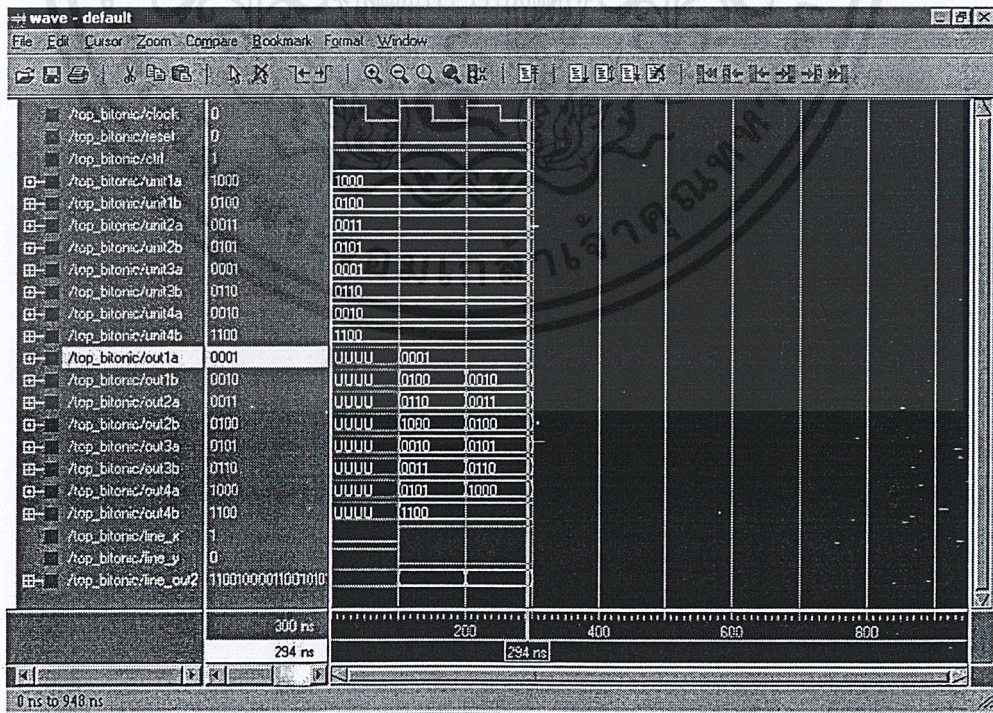


รูปที่ 4.1 ผลที่ได้จากการจำลองการทำงานของ Parallel Exchange Sort

ตารางที่ 4.1 แสดงผลการทำงานที่ได้จากวงจร Exchange Sort

Input	Output ครั้งที่1	Output ครั้งที่2	Output ครั้งที่3
1000	1000	1000	1100
0100	0101	0110	1000
0011	0110	1100	0110
0101	0100	0101	0101
0001	1100	0100	0100
0110	0011	0011	0011
0010	0010	0010	0010
1100	0001	0001	0001

- ผลจาก Parallel Bitonic Merge Sort



รูปที่ 4.2 ผลที่ได้จากการจำลองการทำงาน Parallel Bitonic Merge Sort

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

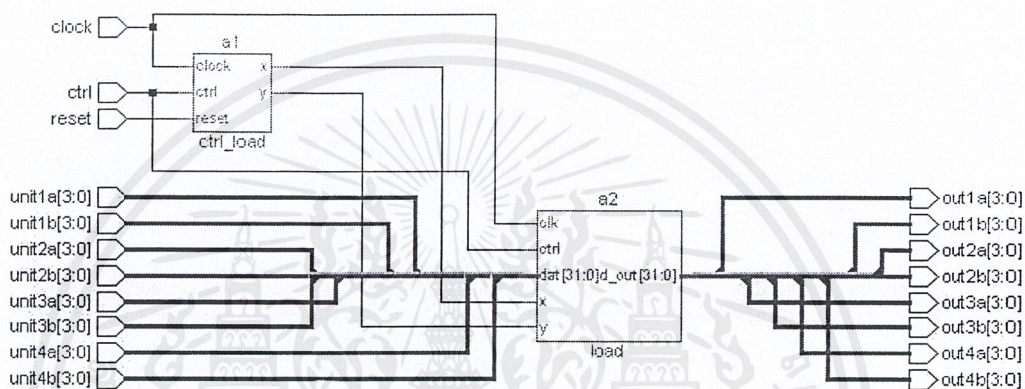
ตารางที่ 4.2 แสดงผลการทำงานของวงจร Parallel Bitonic Merge Sort

Input	Output ครั้งที่1	Output ครั้งที่2
1000	0001	00001
0100	0100	0010
0011	0110	0011
0101	1000	0100
0001	0010	0101
0110	0011	0110
0010	0101	1000
1100	1100	1100

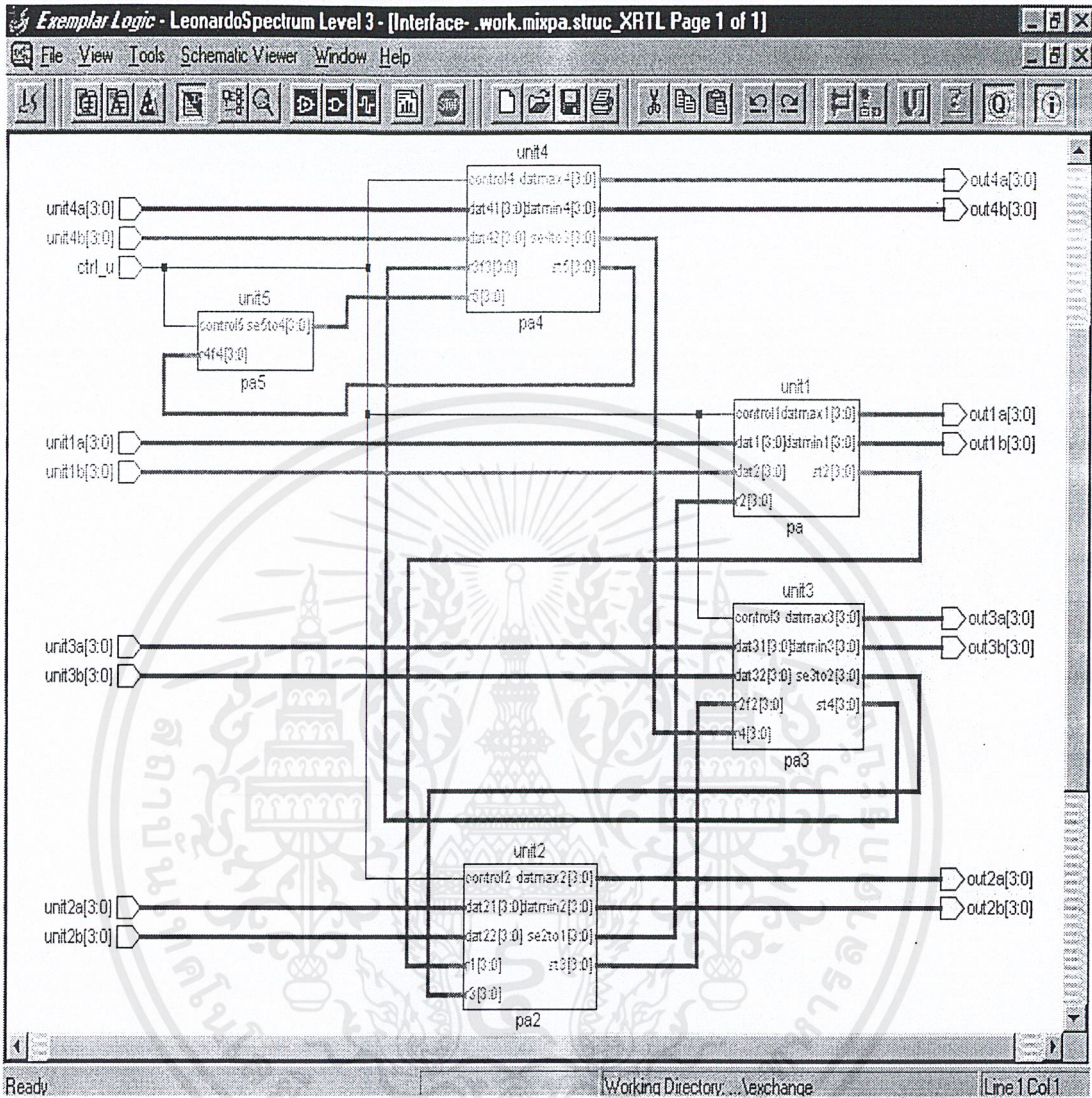
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.2 ผลจากการสังเคราะห์วงจร

- Parallel Exchange Sort



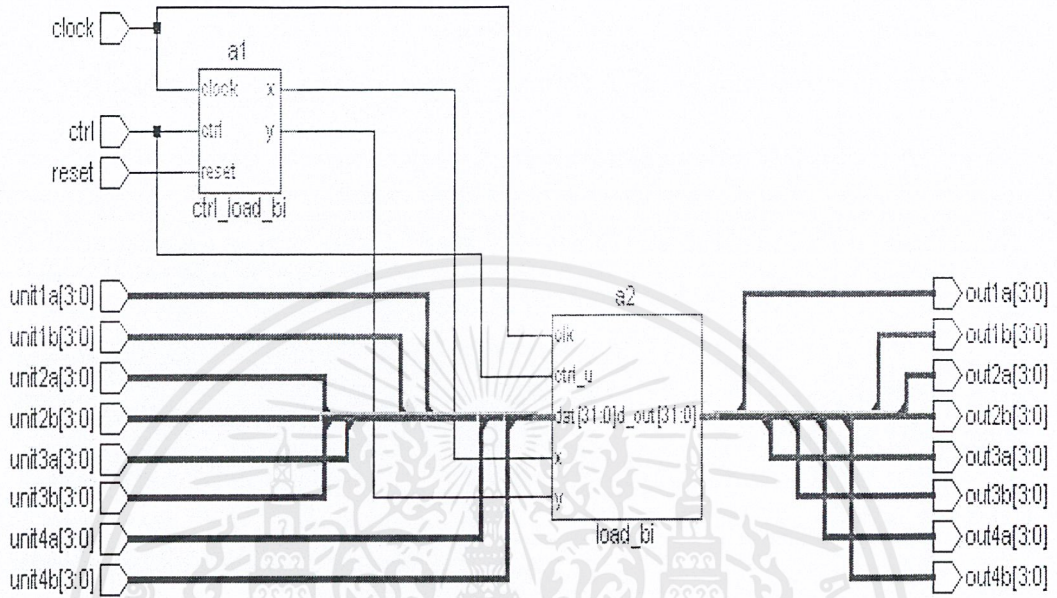
รูปที่ 4.3 แสดงวงจร Parallel Exchange Sort ที่ผ่านการสังเคราะห์



รูปที่ 4.4 แสดงวงจรส่วนจัดเรียงแบบ Parallel Exchange Sort

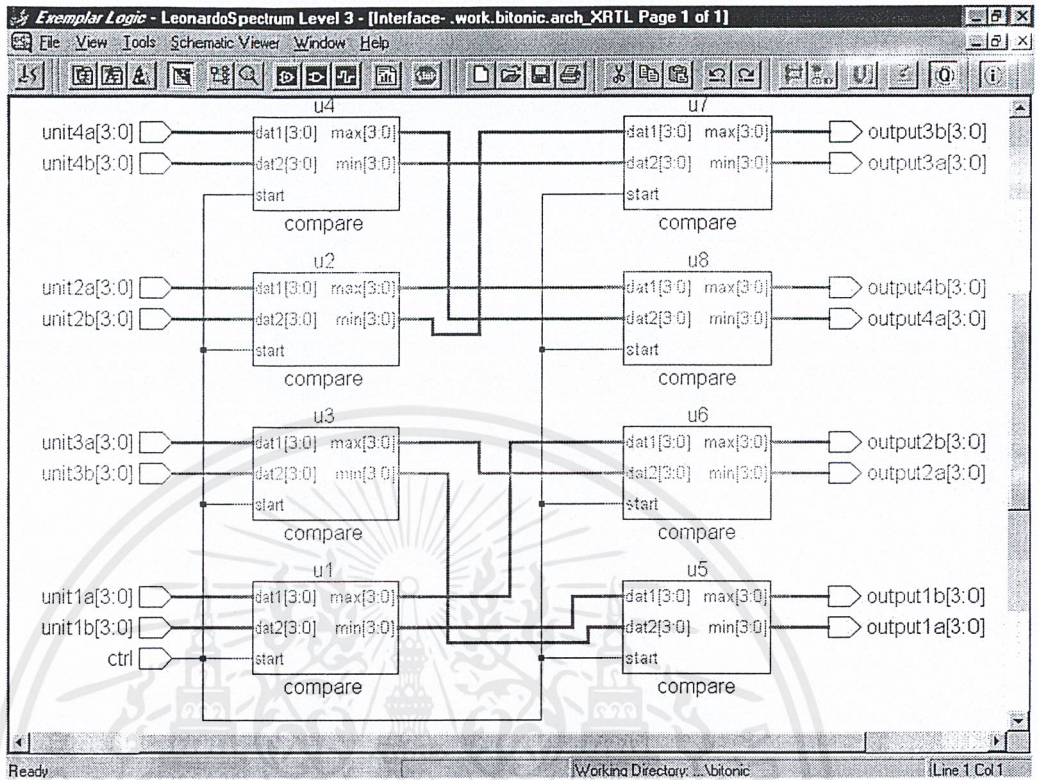
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- Parallel Bitonic Merge Sort



รูปที่ 4.5 แสดงวงจร Parallel Bitonic Merge Sort ที่ผ่านการสังเคราะห์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

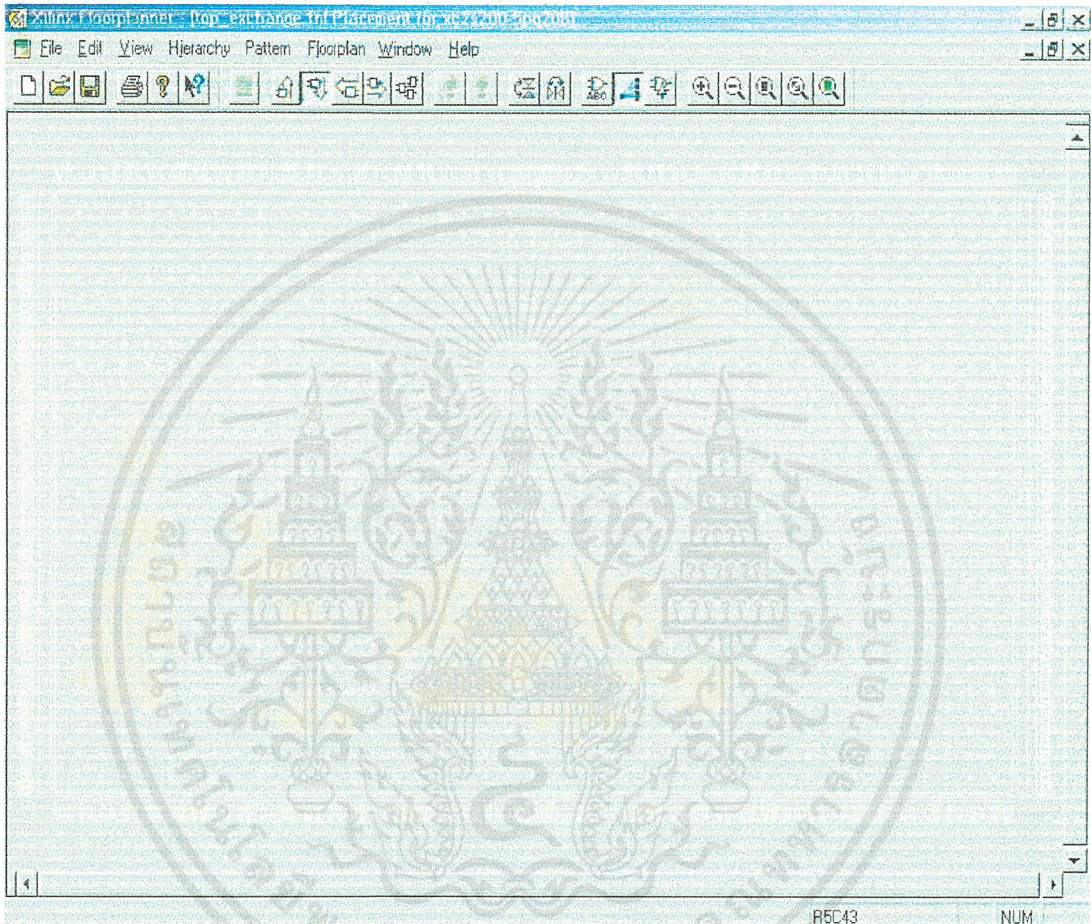


รูปที่ 4.6 แสดงวงจรส่วนจัดเรียงแบบ Parallel Bitonic Merge Sort

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.3 พื้นที่ที่ถูกรใช้งานภายในชิป

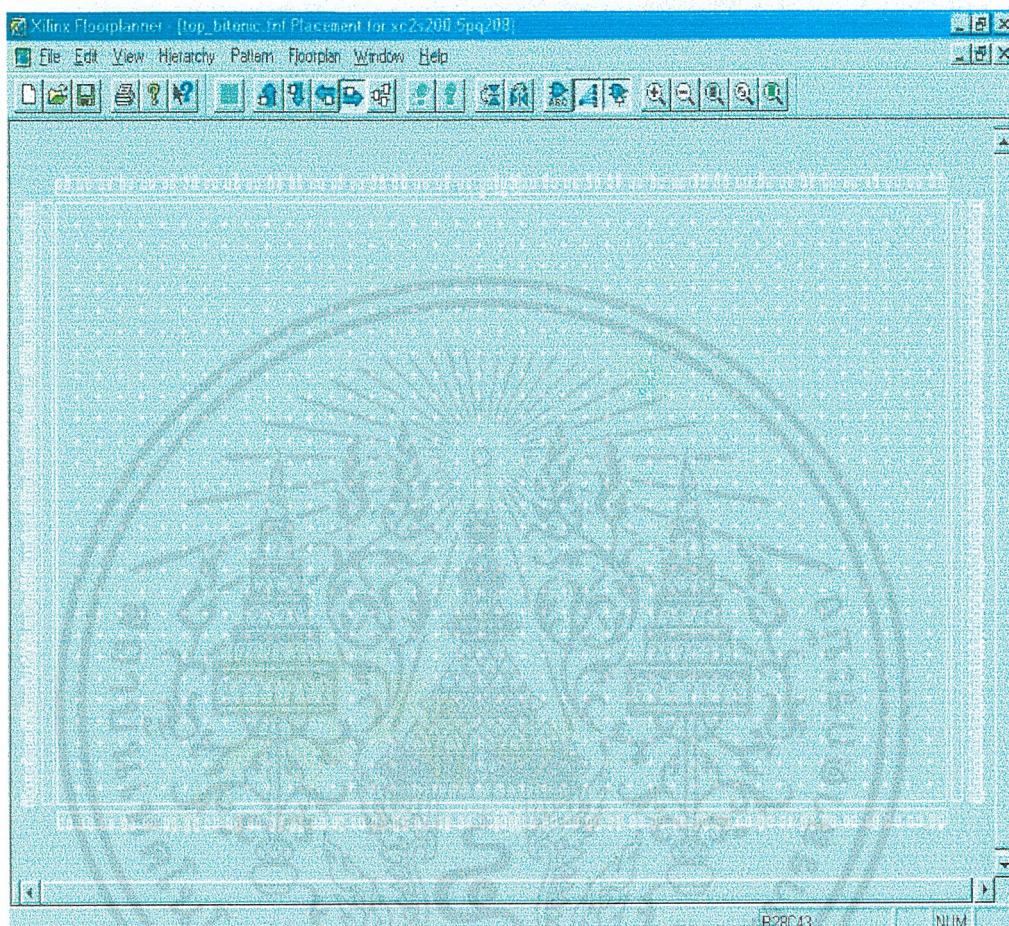
- วงจร Parallel Exchange Sort



รูปที่ 4.5 แสดงพื้นที่ภายในชิปที่ถูกรนำไปใช้เป็นวงจร Parallel Exchange Sort

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

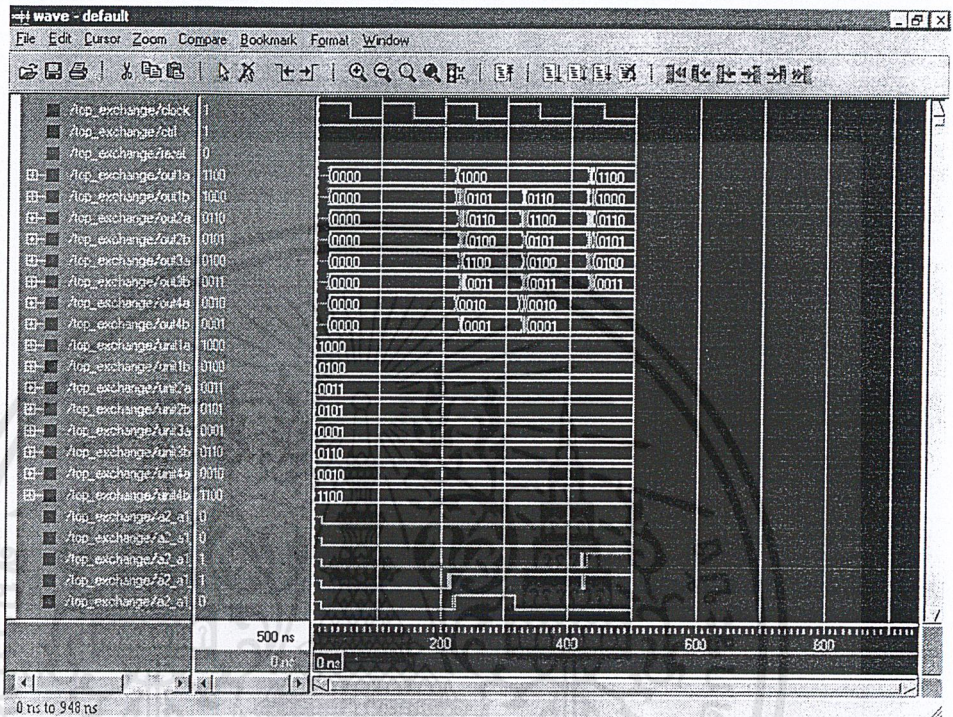
- วงจร Parallel Bitonic Sort



รูปที่ 4.6 แสดงพื้นที่ภายในชิปที่ถูกใช้เป็นวงจร Parallel Bitonic Merge Sort

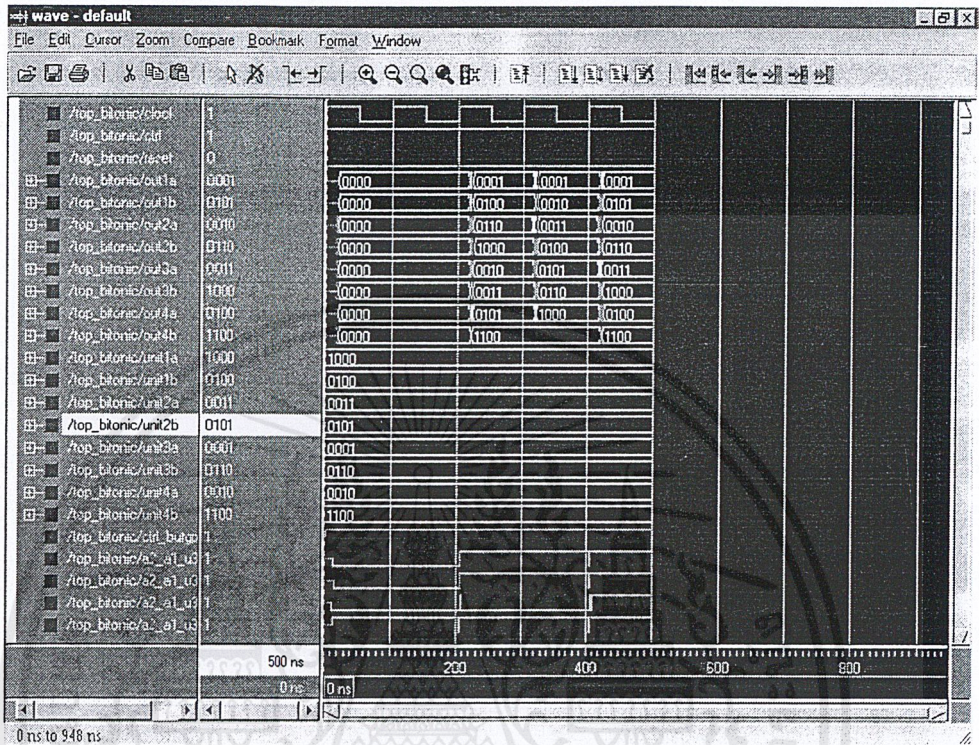
4.4 ผลจากการทำงานจริง

- วงจร Parallel Exchange Sort



รูปที่ 4.8 แสดงผลจากวงจร Parallel Exchange Sort

- วงจร Parallel Bitonic Merge Sort



รูปที่ 4.9 แสดงผลจากวงจร Parallel Bitonic Merge Sort

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5

สรุปผลการดำเนินการ

5.1 สรุปผลการดำเนินการ

จากการทดสอบวงจรจัดเรียงข้อมูลแบบขนาน โดยใช้อัลกอริทึมในการจัดเรียงแบบ Exchange Sort และ Bitonic Merge Sort จากทดสอบอัลกอริทึมทั้ง 2 แบบพบว่า การจัดเรียงข้อมูลแบบ Bitonic Merge Sort ใช้จำนวนเกตเท่ากับ 1,736 เกตในทางทฤษฎีการจัดเรียงแบบนี้ทำงานเสร็จภายใน 3 สัญญาณนาฬิกา แต่ในตอนที่ทำงานจริงใช้ 5 สัญญาณนาฬิกา ส่วน Exchange Sort ใช้จำนวนเกตเท่ากับ 2,914 เกต ในทางทฤษฎีการจัดเรียงแบบนี้ทำงานเสร็จภายใน 3 สัญญาณนาฬิกา แต่ในตอนที่ทำงานจริงใช้ 5 สัญญาณนาฬิกา แสดงว่าทั้ง 2 อัลกอริทึมที่ได้ออกแบบไว้มีข้อผิดพลาดเกิดขึ้น สาเหตุที่เป็นเช่นนี้เพราะเกิดส่วนที่เป็น Critical Path ขึ้นในวงจร เมื่อมีส่วนนี้เกิดขึ้นจะทำให้วงจรทำงานคลาดเคลื่อน สำหรับวงจร Parallel Exchange Sort ที่ออกแบบทำงานผิดพลาดเนื่องจากใช้สัญญาณนาฬิกาเกินมา 1 ลูก เพราะส่วนที่ทำหน้าที่ควบคุมการทำงาน (Control Unit) ให้สัญญาณอ่านข้อมูลเกินมา 1 ครั้งแต่ถ้าปล่อยให้ทำงานต่อเนื่องจะทำงานได้ตรงตามทฤษฎี ส่วนวงจร Bitonic Merge Sort ทำงานผิดพลาดไม่ตรงกับทางทฤษฎี

5.2 ปัญหาที่พบ

- ไม่สามารถทำการ Download ข้อมูลลงบนตัวชิปจากสายที่มากับตัว Board
- ยังไม่สามารถแก้ปัญหาเมื่อเกิด Critical Path ได้เนื่องจากโปรแกรมที่ใช้งานไม่สามารถบอกได้ว่าส่วนไหนของโค้ดที่ทำให้เกิด Critical Path

5.3 ข้อเสนอแนะ

โครงการนี้เป็นเพียงการศึกษาอัลกอริทึมในการจัดเรียงที่เป็นฮาร์ดแวร์ถ้าต้องพัฒนาตัว Sorter ต่อควรที่จะเปลี่ยนการรับข้อมูลให้เป็นขนาด 8 bit และรับข้อมูลเข้ามามากกว่า 8 ชุด เนื่องจากข้อจำกัดของขา IC ควรสร้างหน่วยความจำข้อมูลขึ้นมาเพื่อเก็บข้อมูลที่อ่านเข้ามาพักไว้ก่อนเพื่อรอการส่งต่อไปยังตัว Sorter หลังจากจัดเรียงเสร็จก็นำไปเก็บไว้ที่หน่วยความจำเพื่อรอการส่งออก



ภาคผนวก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

----- TOP_EXCHANGE -----

library ieee ;

use ieee.std_logic_1164.all;

entity top_exchange is

port(

clock: in std_logic;

reset: in std_logic;

ctrl: in std_logic;

unit1a,unit1b,unit2a,unit2b,unit3a,unit3b,unit4a,unit4b : in std_logic_vector(3 downto 0);

out1a,out1b,out2a,out2b,out3a,out3b,out4a,out4b : out std_logic_vector(3 downto 0)

);

end top_exchange;

architecture mix of top_exchange is

component ctrl_load

port(

clock: in std_logic;

reset: in std_logic;

ctrl: in std_logic;

x,y : out std_logic

);

end component;

component load

port(

clk,x,y : in std_logic; -- control

ctrl : in std_logic;

dat : in std_logic_vector(31 downto 0);

d_out : out std_logic_vector(31 downto 0)

);

end component;

```
signal line_x,line_y : std_logic;
signal line_out2 : std_logic_vector(31 downto 0);
```

```
begin
```

```
a1:ctrl_load port map(
```

```
    clock => clock,
    reset => reset,
    ctrl => ctrl,
    x    => line_x,
    y    => line_y
);
```

```
a2:load port map(
```

```
    clk => clock,
    x   => line_x,
    y   => line_y,
    ctrl => ctrl,
    dat(3 downto 0) => unit1a ,
    dat(7 downto 4) => unit1b ,
    dat(11 downto 8) => unit2a ,
    dat(15 downto 12) => unit2b ,
    dat(19 downto 16) => unit3a ,
    dat(23 downto 20) => unit3b ,
    dat(27 downto 24) => unit4a ,
    dat(31 downto 28) => unit4b ,
    d_out( 3 downto 0) =>line_out2(3 downto 0) ,
    d_out( 7 downto 4) => line_out2(7 downto 4) ,
    d_out( 11 downto 8) => line_out2(11 downto 8),
    d_out( 15 downto 12) => line_out2(15 downto 12),
    d_out( 19 downto 16) => line_out2(19 downto 16),
    d_out( 23 downto 20) => line_out2(23 downto 20),
    d_out( 27 downto 24) => line_out2(27 downto 24),
    d_out( 31 downto 28) => line_out2(31 downto 28)
);
```

```

out1a <= line_out2( 3 downto 0);
out1b <= line_out2( 7 downto 4);
out2a <= line_out2( 11 downto 8);
out2b <= line_out2( 15 downto 12);
out3a <= line_out2( 19 downto 16);
out3b <= line_out2( 23 downto 20);
out4a <= line_out2( 27 downto 24);
out4b <= line_out2( 31 downto 28);
end mix;

```

----- CTRL_LOAD-----

```

library ieee ;
use ieee.std_logic_1164.all;

```

```

entity ctrl_load is

```

```

port(

```

```

    clock:    in std_logic;

```

```

    reset:    in std_logic;

```

```

    ctrl:     in std_logic;

```

```

    x,y : out std_logic

```

```

);

```

```

end ctrl_load;

```

```

architecture behav of ctrl_load is

```

```

-- define the states of FSM model

```

```

type state_type is (S0, S1, S2, S3);

```

```

signal current_state,next_state : state_type;

```

```

-- signal x,y : std_logic;

```

```

begin

```

```

-- cocurrent process#1: state registers

```

```

state_reg: process(clock, reset,next_state)

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

begin

    if (reset='1') then
    current_state <= S0;
    elsif (clock'event and clock='1') then
        current_state <= next_state;
    end if;

end process;

```

```

-- cocurrent process#2: combinational logic
comb_logic: process(current_state, ctrl)
begin

```

```

    -- use case statement to show the
    -- state transistion
    x <= '0';
    y <= '0';
    case current_state is

-- load data
        when S0 => y <= '1';
            if ctrl='0' then
                next_state <= S0;
            elsif ctrl ='1' then
                next_state <= S1;
            end if;

```

```

        when S1 => y <= '0';
            x <= '1';
            if ctrl ='0' then
                next_state <= S1;
            elsif ctrl ='1' then
                next_state <= S2;
            end if;

```

```

when S2 => y <= '0';
x <= '1';

if ctrl='0' then
    next_state <= S2;
elsif ctrl='1' then
    next_state <= S3;
end if;

-- load data
when S3 => x <= '1';
y <= '0';

if ctrl='0' then
    next_state <= S3;
elsif ctrl='1' then
    next_state <= S0;
end if;

when others =>
x <= '0';
y <= '0';
next_state <= S0;

end case;

end process;

end behavior;

-----LOAD-----
library ieee;
use ieee.std_logic_1164.all;

entity load is
port(

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

        clk,x,y : in std_logic; -- control
    ctrl : in std_logic;
    dat : in std_logic_vector(31 downto 0);
    d_out : out std_logic_vector(31 downto 0)
);
end load;

```

architecture behav of load is

component mixpa

```

port (
    unit1a,unit1b,unit2a,unit2b,unit3a,unit3b,unit4a,unit4b : in std_logic_vector(3 downto 0);
    -- control unit signal
    ctrl_u : in std_logic;
    out1a,out1b,out2a,out2b,out3a,out3b,out4a,out4b : out std_logic_vector(3 downto 0)
);
end component;

```

```

signal line_in,line_out : std_logic_vector(31 downto 0);

```

```

begin

```

```

a1:mixpa port map(
    unit1a => line_in(3 downto 0),
    unit1b => line_in(7 downto 4),
    unit2a => line_in(11 downto 8),
    unit2b => line_in(15 downto 12),
    unit3a => line_in(19 downto 16),
    unit3b => line_in(23 downto 20),
    unit4a => line_in(27 downto 24),
    unit4b => line_in(31 downto 28),
    ctrl_u => ctrl,
    out1a => line_out( 3 downto 0),
    out1b => line_out( 7 downto 4),

```

```

out2a => line_out( 11 downto 8),
out2b => line_out( 15 downto 12),
out3a => line_out( 19 downto 16),
out3b => line_out( 23 downto 20),
out4a => line_out( 27 downto 24),
out4b => line_out( 31 downto 28)
);

-- feed back
process(clk,y,x,dat,line_out)
variable temp : std_logic_vector(31 downto 0);
begin
if clk'event and clk = '1' then
  if y = '1' and x = '0' then
    temp := dat;
    line_in <= temp;
  elsif y = '0' and x = '1' then
--    line_in <= line_out;
    temp := line_out;
    line_in <= temp;
  end if;
end if;
d_out <= line_out;
end process;
--d_out <= line_out;
end behav;

-----MIXPA-----

library ieee;
use ieee.std_logic_1164.all;

entity mixpa is
--
port (
  unit1a,unit1b,unit2a,unit2b,unit3a,unit3b,unit4a,unit4b   : in  std_logic_vector(3 downto 0);
  -- control unit signal

```

```

ctrl_u          : in std_logic;
out1a,out1b,out2a,out2b,out3a,out3b,out4a,out4b : out std_logic_vector(3 downto 0)
);
end mixpa;

```

architecture struc of mixpa is

```
-- component in mixpa
```

```
component pa
```

```
port(dat1,dat2,r2 : in std_logic_vector(3 downto 0);
```

```
-- r2 for receive form unit 2
```

```
control1 : in std_logic;
```

```
datmax1,datmin1,st2 : out std_logic_vector(3 downto 0)
```

```
-- st2 : send to unit2
```

```
);
```

```
end component;
```

```
component pa2
```

```
port (dat21,dat22,r1,r3 : in std_logic_vector(3 downto 0);
```

```
-- r1 : receive from 1
```

```
-- r3 : receive form 3
```

```
control2 : in std_logic;
```

```
datmax2,datmin2,st3,se2to1 : out std_logic_vector(3 downto 0)
```

```
--st3 : send to unit 3
```

```
);
```

```
end component;
```

```
component pa3
```

```
port (dat31,dat32,r2f2,r4 : in std_logic_vector(3 downto 0);
```

```
control3 : in std_logic;
```

```
datmax3,datmin3,st4,se3to2 : out std_logic_vector(3 downto 0)
```

```
);
```

```
end component;
```

```
component pa4
```

```

port (dat41,dat42,r3f3,r5 : in std_logic_vector(3 downto 0);
      control4 : in std_logic;
      datmax4,datmin4,st5,se4to3 : out std_logic_vector(3 downto 0)
      );

```

```

end component;

```

```

component pa5

```

```

port (r4f4 : in std_logic_vector(3 downto 0);
      control5 : in std_logic;
      se5to4 : out std_logic_vector(3 downto 0)
      );

```

```

end component;

```

```

--signal in mixpa

```

```

signal s12,s21,s23,s32,s34,s43,s45,s54 : std_logic_vector(3 downto 0);

```

```

begin

```

```

unit1:pa

```

```

port map (
  dat1 => unit1a,
  dat2 => unit1b,
  r2   => s21,
  control1 => ctrl_u,
  datmax1 => out1a,
  datmin1 => out1b,
  st2   => s12
);

```

```

unit2:pa2

```

```

port map (
  dat21 => unit2a,
  dat22 => unit2b,
  r1   => s12,
  r3   => s32,

```

```
control2 => ctrl_u,  
datmax2 => out2a,  
datmin2 => out2b,  
st3    => s23,  
se2to1 => s21  
);
```

unit3:pa3

```
port map (  
    dat31 => unit3a,  
    dat32 => unit3b,  
    r2f2  => s23,  
    r4    => s43,  
    control3 => ctrl_u,  
    datmax3 => out3a,  
    datmin3 => out3b,  
    st4    => s34,  
    se3to2 => s32  
);
```

unit4:pa4

```
port map (  
    dat41 => unit4a,  
    dat42 => unit4b,  
    r3f3  => s34,  
    r5    => s54,  
    control4 => ctrl_u,  
    datmax4 => out4a,  
    datmin4 => out4b,  
    st5    => s45,  
    se4to3 => s43  
);
```

unit5:pa5

```
port map (  
    r4f4  => s45,
```

```

        control5 => ctrl_u,
        se5to4  => s54
    );
end struc;

```

-----PA-----

```

library ieee;
use ieee.std_logic_1164.all;

```

```
entity pa is
```

```
port(dat1,dat2,r2 : in std_logic_vector(3 downto 0);
```

```
-- r2 for receive form unit 2
```

```
control1 : in std_logic;
```

```
datmax1,datmin1,si2 : out std_logic_vector(3 downto 0)
```

```
-- si2 : send to unit2
```

```
);
```

```
end pa;
```

```
architecture struc of pa is
```

```
component compare
```

```
port(dat1,dat2 : in std_logic_vector(3 downto 0);
```

```
start : in std_logic;
```

```
max,min : out std_logic_vector(3 downto 0)
```

```
);
```

```
end component;
```

```
component temp
```

```
port(dat10 : in std_logic_vector(3 downto 0);
```

```
control: in std_logic;
```

```
reg : out std_logic_vector(3 downto 0)
```

```
);
```

```
end component;
```

```
signal temp1,temp2,temp3,temp4 : std_logic_vector(3 downto 0);
```

```

-- temp1 connect temp
-- temp2 connect datout2
-- temp3 connect t1(reg)
-- temp4 connect

begin
u11:compare --compare
  port map (
    dat1 => dat1,
    dat2 => dat2,
    start => control1,
    max => temp1,
    min => temp2
  );
ta:temp
  port map (
    dat10 => temp1,
    control => control1,
    reg => temp3--datout1
  );
tb:temp
  port map(
    dat10 => temp3,
    control => control1,
    reg => temp4
  );
u12:compare
  port map( dat1 => r2,
    dat2 => temp4,
    start => control1,
    max => datmax1,
    min => datmin1
  );
  st2 <= temp2; -- for send to unit two
end struc;

```

```

-----PA2-----
library ieee;
use ieee.std_logic_1164.all;

entity pa2 is
port (dat21,dat22,r1,r3 : in std_logic_vector(3 downto 0);
-- r1 : receive from 1
-- r3 : receive form 3
    control2 : in std_logic;
    datmax2,datmin2,st3,se2to1 : out std_logic_vector(3 downto 0)
--st3 : send to unit 3
    );
end pa2;

architecture struc2 of pa2 is
-- component declaretion
-- start
component compare
port(dat1,dat2 : in std_logic_vector(3 downto 0);
    start : in std_logic;
    max,min : out std_logic_vector(3 downto 0)
);
end component;

-- end
signal down1,down_max,down_min,down2 : std_logic_vector(3 downto 0); --right,left,
begin

u21:compare
    port map(
        dat1 => dat21,
        dat2 => dat22,
        start => control2,
        max => down1,
        min => st3--left

```

```

);

--st3 <= left;

u22:compare
  port map(
    dat1 => down1,
    -- dat 2connect unit 1
    dat2 => r1,
    start => control2,
    max => down_max,
    min => down_min
  );

```

```

u23:compare
  port map(
    dat1 => down_max,
    dat2 => down_min,
    start => control2,
    max => se2to1,--right,
    min => down2
  );

```

```

u24:compare
  port map(
    dat1 => r3,
    dat2 => down2,
    start => control2,
    max => datmax2,
    min => datmin2
  );

```

```
end struc2;
```

```
-----PA3-----
```

```
library ieee;
```

```

use ieee.std_logic_1164.all;

entity pa3 is
port (dat31,dat32,r2f2,r4 : in std_logic_vector(3 downto 0);
      control3 : in std_logic;
      datmax3,datmin3,st4,se3to2 : out std_logic_vector(3 downto 0)
      );
end pa3;
architecture struc3 of pa3 is

component compare
port(dat1,dat2 : in std_logic_vector(3 downto 0);
      start : in std_logic;
      max,min : out std_logic_vector(3 downto 0)
);
end component;

signal down1,down_max,down_min,down2 : std_logic_vector(3 downto 0); --right,left,

begin

u31:compare
port map(
dat1 => dat31,
dat2 => dat32,
start => control3,
max => down1,
min => st4--left
);

--st4 <= left;

u32:compare
port map(
dat1 => down1,

```

```

-- dat2connect unit 2

dat2 => r2f2,

start => control3,

max => down_max,

min => down_min

);

u33:compare
port map(
dat1 => down_max,
dat2 => down_min,
start => control3,
max => se3to2,--right,
min => down2
);

u34:compare
port map(
dat1 => r4,
dat2 => down2,
start => control3,
max => datmax3,
min => datmin3
);

end struc3;

-----PA4-----

library ieee;
use ieee.std_logic_1164.all;

entity pa4 is
port (dat41,dat42,r3f3,r5 : in std_logic_vector(3 downto 0);
control4 : in std_logic;
datmax4,datmin4,st5,se4to3 : out std_logic_vector(3 downto 0)
);

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

end pa4;
architecture struc4 of pa4 is

component compare
port(dat1,dat2 : in std_logic_vector(3 downto 0);
      start   : in std_logic;
      max,min  : out std_logic_vector(3 downto 0)
);
end component;

signal down1,down_max,down_min,down2 : std_logic_vector(3 downto 0);--right,left,

begin

u41:compare
port map(
  dat1 => dat41,
  dat2 => dat42,
  start => control4,
  max => down1,
  min => st5--left
);

--st5 <= left;

u42:compare
port map(
  dat1 => down1,
  -- dat 2connect unit 3
  dat2 => r3f3,
  start => control4,
  max => down_max,
  min => down_min
);

```

u43:compare

```
port map(  
    dat1 => down_max,  
    dat2 => down_min,  
    start => control4,  
    max  => se4to3,--right,  
    min  => down2  
);
```

u44:compare

```
port map(  
    dat1 => r5,  
    dat2 => down2,  
    start => control4,  
    max  => datmax4,  
    min  => datmin4  
);
```

end struc4;

-----PA5-----

library ieee;

use ieee.std_logic_1164.all;

entity pa5 is

```
port (r4f4 : in std_logic_vector(3 downto 0);  
      control5 : in std_logic;  
      se5to4 : out std_logic_vector(3 downto 0)  
);
```

end pa5;

architecture struc5 of pa5 is

component temp

```
port(dat10 : in std_logic_vector(3 downto 0);  
      control: in std_logic;
```

```

    reg : out std_logic_vector(3 downto 0)
  );
end component;

```

```

signal temp1: std_logic_vector(3 downto 0);

```

```

begin

```

```

temp_B:temp

```

```

  port map(
    dat10 => r4f4,
    control => control5,
    reg => temp1
  );

```

```

temp_a:temp

```

```

  port map(
    dat10 => temp1,
    control => control5,
    reg => se5to4
  );

```

```

end struc5;

```

```

-----COMPARE-----

```

```

library ieee;

```

```

use ieee.std_logic_1164.all;

```

```

entity compare is

```

```

  port(dat1,dat2 : in std_logic_vector(3 downto 0);

```

```

        start : in std_logic;

```

```

        max,min : out std_logic_vector(3 downto 0)

```

```

  );

```

```

end compare;

```

```

architecture arc_of_com of compare is

```

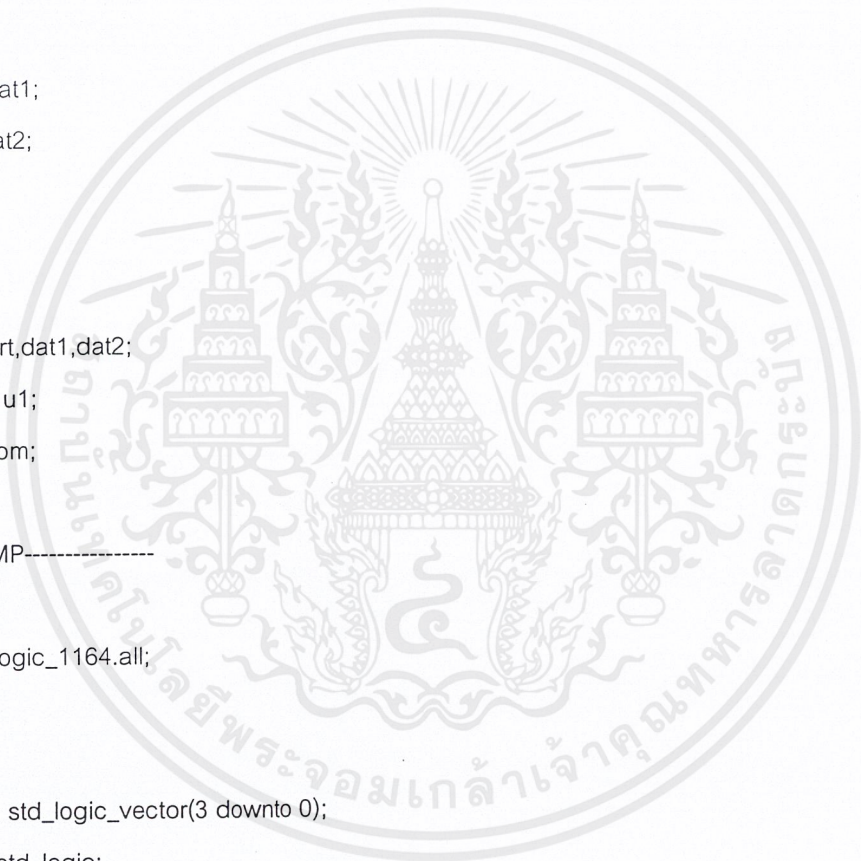
```

begin
u1:process(start,dat1,dat2)
begin
if start = '1' then
if(dat1 > dat2) then
max <= dat1;
min <= dat2;
elsif(dat1 < dat2) then
max <= dat2;
min <= dat1;
else
max <= dat1;
min <= dat2;
end if;
end if;
-- wait on start,dat1,dat2;
end process u1;
end arc_of_com;

-----TEMP-----
library ieee;
use ieee.std_logic_1164.all;
--
entity temp is
port(dat10 : in std_logic_vector(3 downto 0);
control: in std_logic;
reg : out std_logic_vector(3 downto 0)
);
end temp;

architecture behav of temp is
--signal temp : bit_vector(3 downto 0);
begin
process(control,dat10)

```



```
begin
  if control = '1' then
    reg <= dat10;
  else
    reg <= "0000";
  end if;
-- wait on control,dat10;
  end process;
end behav;
```



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Introduction

The Spartan™-II 2.5V Field-Programmable Gate Array family gives users high performance, abundant logic resources, and a rich feature set, all at an exceptionally low price. The six-member family offers densities ranging from 15,000 to 200,000 system gates, as shown in Table 1. System performance is supported up to 200 MHz.

Spartan-II devices deliver more gates, I/Os, and features per dollar than other FPGAs by combining advanced process technology with a streamlined Virtex-based architecture. Features include block RAM (to 56K bits), distributed RAM (to 75,264 bits), 16 selectable I/O standards, and four DLLs. Fast, predictable interconnect means that successive design iterations continue to meet timing requirements.

The Spartan-II family is a superior alternative to mask-programmed ASICs. The FPGA avoids the initial cost, lengthy development cycles, and inherent risk of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary (impossible with ASICs).

Features

- Second generation ASIC replacement technology
 - Densities as high as 5,292 logic cells with up to 200,000 system gates
 - Streamlined features based on Virtex architecture
 - Unlimited reprogrammability
 - Very low cost

- System level features
 - SelectRAM+™ hierarchical memory:
 - 16 bits/LUT distributed RAM
 - Configurable 4K bit block RAM
 - Fast interfaces to external RAM
 - Fully PCI compliant
 - Low-power segmented routing architecture
 - Full readback ability for verification/observability
 - Dedicated carry logic for high-speed arithmetic
 - Dedicated multiplier support
 - Cascade chain for wide-input functions
 - Abundant registers/latches with enable, set, reset
 - Four dedicated DLLs for advanced clock control
 - Four primary low-skew global clock distribution nets
 - IEEE 1149.1 compatible boundary scan logic
- Versatile I/O and packaging
 - Low cost packages available in all densities
 - Family footprint compatibility in common packages
 - 16 high-performance interface standards
 - Hot swap Compact PCI friendly
 - Zero hold time simplifies system timing
- Fully supported by powerful Xilinx development system
 - Foundation ISE Series: Fully integrated software
 - Alliance Series: For use with third-party tools
 - Fully automatic mapping, placement, and routing

Table 1: Spartan-II FPGA Family Members

Device	Logic Cells	System Gates (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O ⁽¹⁾	Total Distributed RAM Bits	Total Block RAM Bits
XC2S15	432	15,000	8 x 12	96	86	6,144	16K
XC2S30	972	30,000	12 x 18	216	132	13,824	24K
XC2S50	1,728	50,000	16 x 24	384	176	24,576	32K
XC2S100	2,700	100,000	20 x 30	600	196	38,400	40K
XC2S150	3,888	150,000	24 x 36	864	260	55,296	48K
XC2S200	5,292	200,000	28 x 42	1,176	284	75,264	56K

Notes:

1. All user I/O counts do not include the four global clock/user input pins. See details in Table 3, page 3.

General Overview

The Spartan-II family of FPGAs have a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), surrounded by a perimeter of programmable Input/Output Blocks (IOBs). There are four Delay-Locked Loops (DLLs), one at each corner of the die. Two columns of block RAM lie on opposite sides of the die, between the CLBs and the IOB columns. These functional elements are interconnected by a powerful hierarchy of versatile routing channels (see Figure 1).

Spartan-II FPGAs are customized by loading configuration data into internal static memory cells. Unlimited reprogramming cycles are possible with this approach. Stored values in these cells determine logic functions and interconnections implemented in the FPGA. Configuration data can be read from an external serial PROM (master serial mode), or written into the FPGA in slave serial, slave parallel, or Boundary Scan modes.

Spartan-II FPGAs are typically used in high-volume applications where the versatility of a fast programmable solution adds benefits. Spartan-II FPGAs are ideal for shortening product development cycles while offering a cost-effective solution for high volume production.

Spartan-II FPGAs achieve high-performance, low-cost operation through advanced architecture and semiconductor technology. Spartan-II devices provide system clock rates up to 200 MHz. Spartan-II FPGAs offer the most cost-effective solution while maintaining leading edge performance. In addition to the conventional benefits of high-volume programmable logic solutions, Spartan-II FPGAs also offer on-chip synchronous single-port and dual-port RAM (block and distributed form), DLL clock drivers, programmable set and reset on all flip-flops, fast carry logic, and many other features.

The Xilinx XC17S00A PROM family is recommended for serial configuration of Spartan-II FPGAs. The In-System Programmable (ISP) XC18V00 PROM family is recommended for parallel or serial configuration.

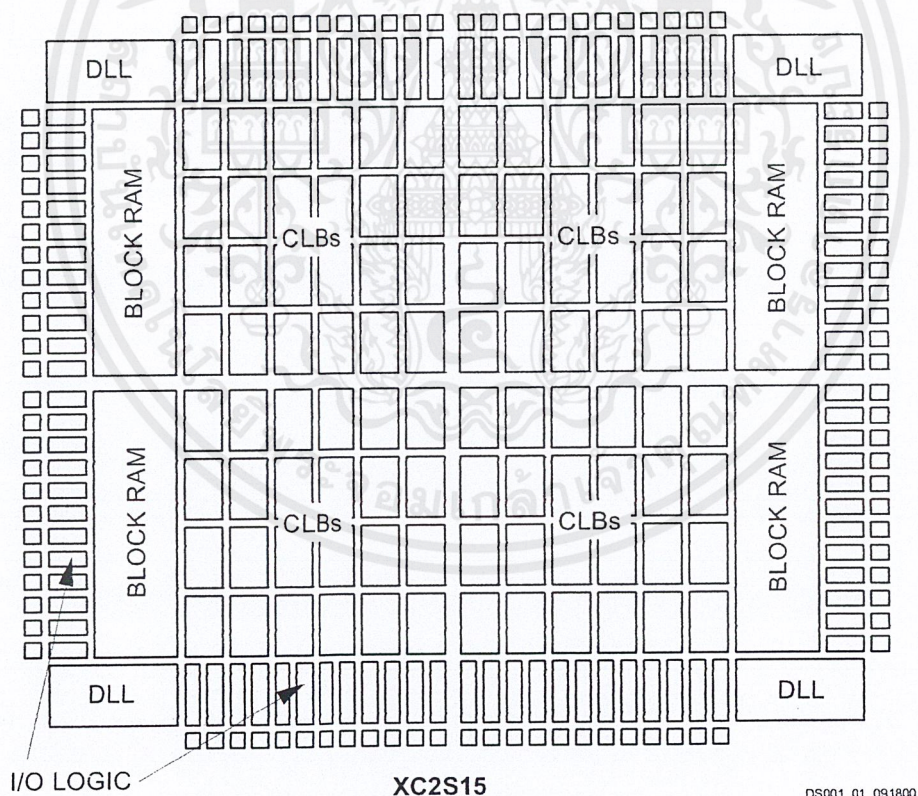


Figure 1: Basic Spartan-II Family FPGA Block Diagram

Spartan-II Product Availability

Table 2 shows the package and speed grades available for Spartan-II family devices. Table 3 shows the maximum user I/Os available on the device and the number of user I/Os available for each device/package combination. The four

global clock pins are usable as additional user I/Os when not used as a global clock pin. These pins are not included in user I/O counts.

Table 2: Spartan-II Package and Speed Grade Availability

Device	Pins	100	144	144	208	256	456
	Type	Plastic VQFP	Plastic TQFP	Chip Scale BGA	Plastic PQFP	Fine Pitch BGA	Fine Pitch BGA
	Code	VQ100	TQ144	CS144	PQ208	FG256	FG456
XC2S15	-5	C, I	C, I	C, I	-	-	-
	-6	C	C	C	-	-	-
XC2S30	-5	C, I	C, I	C, I	C, I	-	-
	-6	C	C	C	C	-	-
XC2S50	-5	-	C, I	-	C, I	C, I	-
	-6	-	C	-	C	C	-
XC2S100	-5	-	C, I	-	C, I	C, I	C, I
	-6	-	C	-	C	C	C
XC2S150	-5	-	-	-	C, I	C, I	C, I
	-6	-	-	-	C	C	C
XC2S200	-5	-	-	-	C, I	C, I	C, I
	-6	-	-	-	C	C	C

Notes:

1. C = Commercial, $T_J = 0^\circ$ to $+85^\circ\text{C}$; I = Industrial, $T_J = -40^\circ\text{C}$ to $+100^\circ\text{C}$.

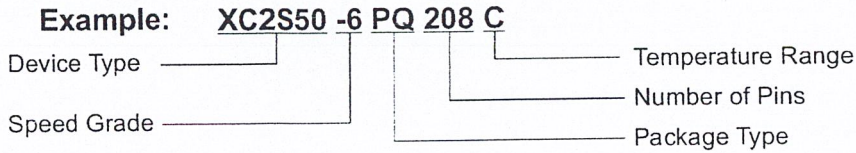
Table 3: Spartan-II User I/O Chart⁽¹⁾

Device	Maximum User I/O	Available User I/O According to Package Type					
		VQ100	TQ144	CS144	PQ208	FG256	FG456
XC2S15	86	60	86	86	-	-	-
XC2S30	132	60	92	92	132	-	-
XC2S50	176	-	92	-	140	176	-
XC2S100	196	-	92	-	140	176	196
XC2S150	260	-	-	-	140	176	260
XC2S200	284	-	-	-	140	176	284

Notes:

1. All user I/O counts do not include the four global clock/user input pins.

Ordering Information



Device Ordering Options

Device	Speed Grade		Number of Pins / Package Type		Temperature Range (T _J)	
XC2S15	-5	Standard Performance	VQ100	100-pin Plastic Very Thin QFP	C = Commercial	0°C to +85°C
XC2S30	-6	Higher Performance	CS144	144-ball Chip-Scale BGA	I = Industrial	-40°C to +100°C
XC2S50			TQ144	144-pin Plastic Thin QFP		
XC2S100			PQ208	208-pin Plastic QFP		
XC2S150			FG256	256-ball Fine Pitch BGA		
XC2S200			FG456	456-ball Fine Pitch BGA		

Revision History

Version No.	Date	Description
2.0	09/18/00	Sectioned the Spartan-II Family data sheet into four modules. Added industrial temperature range information.
2.1	10/31/00	Removed Power down feature.
2.2	03/05/01	Added statement on PROMs.
2.3	11/01/01	Update Product Availability chart. Minor text edits.

The Spartan-II Family Data Sheet

DS001-1, *Spartan-II 2.5V FPGA Family: Introduction and Ordering Information* (Module 1)

DS001-2, *Spartan-II 2.5V FPGA Family: Functional Description* (Module 2)

DS001-3, *Spartan-II 2.5V FPGA Family: DC and Switching Characteristics* (Module 3)

DS001-4, *Spartan-II 2.5V FPGA Family: Pinout Tables* (Module 4)

PN 011311

Architectural Description

Spartan-II Array

The Spartan-II user-programmable gate array, shown in Figure 1, is composed of five major configurable elements:

- IOBs provide the interface between the package pins and the internal logic
- CLBs provide the functional elements for constructing most logic
- Dedicated block RAM memories of 4096 bits each
- Clock DLLs for clock-distribution delay compensation and clock domain control
- Versatile multi-level interconnect structure

As can be seen in Figure 1, the CLBs form the central logic structure with easy access to all support and routing structures. The IOBs are located around all the logic and memory elements for easy and quick routing of signals on and off the chip.

Values stored in static memory cells control all the configurable logic elements and interconnect resources. These values load into the memory cells on power-up, and can reload if necessary to change the function of the device.

Each of these elements will be discussed in detail in the following sections.

Input/Output Block

The Spartan-II IOB, as seen in Figure 1, features inputs and outputs that support a wide variety of I/O signaling standards. These high-speed inputs and outputs are capable of supporting various state of the art memory and bus interfaces. Table 1 lists several of the standards which are supported along with the required reference, output and termination voltages needed to meet the standard.

The three IOB registers function either as edge-triggered D-type flip-flops or as level-sensitive latches. Each IOB has a clock signal (CLK) shared by the three registers and independent Clock Enable (CE) signals for each register.

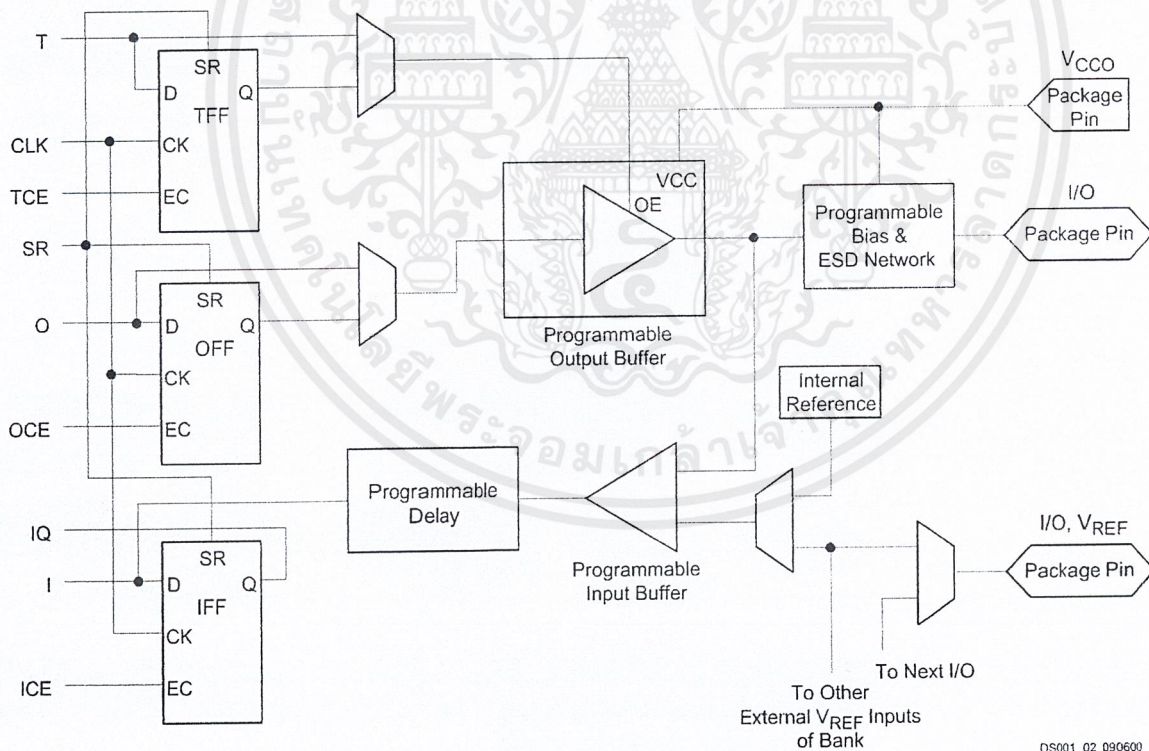


Figure 1: Spartan-II Input/Output Block (IOB)

DS001_02_090600

In addition to the CLK and CE control signals, the three registers share a Set/Reset (SR). For each register, this signal can be independently configured as a synchronous Set, a synchronous Reset, an asynchronous Preset, or an asynchronous Clear.

A feature not shown in the block diagram, but controlled by the software, is polarity control. The input and output buffers and all of the IOB control signals have independent polarity controls.

Optional pull-up and pull-down resistors and an optional weak-keeper circuit are attached to each pad. Prior to configuration all outputs not involved in configuration are forced into their high-impedance state. The pull-down resistors and the weak-keeper circuits are inactive, but inputs may optionally be pulled up.

Table 1: Standards Supported by I/O (Typical Values)

I/O Standard	Input Reference Voltage (V_{REF})	Output Source Voltage (V_{CCO})	Board Termination Voltage (V_{TT})
LVTTTL (2-24 mA)	N/A	3.3	N/A
LVC MOS2	N/A	2.5	N/A
PCI (3V/5V, 33 MHz/66 MHz)	N/A	3.3	N/A
GTL	0.8	N/A	1.2
GTL+	1.0	N/A	1.5
HSTL Class I	0.75	1.5	0.75
HSTL Class III	0.9	1.5	1.5
HSTL Class IV	0.9	1.5	1.5
SSTL3 Class I and II	1.5	3.3	1.5
SSTL2 Class I and II	1.25	2.5	1.25
CTT	1.5	3.3	1.5
AGP-2X	1.32	3.3	N/A

The activation of pull-up resistors prior to configuration is controlled on a global basis by the configuration mode pins. If the pull-up resistors are not activated, all the pins will float. Consequently, external pull-up or pull-down resistors must be provided on pins required to be at a well-defined logic level prior to configuration.

All pads are protected against damage from electrostatic discharge (ESD) and from over-voltage transients. Two forms of over-voltage protection are provided, one that permits 5V compliance, and one that does not. For 5V compliance, a zener-like structure connected to ground turns on when the output rises to approximately 6.5V. When 5V com-

pliance is not required, a conventional clamp diode may be connected to the output supply voltage, V_{CCO} . The type of over-voltage protection can be selected independently for each pad.

All Spartan-II IOBs support IEEE 1149.1-compatible boundary scan testing.

Input Path

A buffer in the Spartan-II IOB input path routes the input signal either directly to internal logic or through an optional input flip-flop.

An optional delay element at the D-input of this flip-flop eliminates pad-to-pad hold time. The delay is matched to the internal clock-distribution delay of the FPGA, and when used, assures that the pad-to-pad hold time is zero.

Each input buffer can be configured to conform to any of the low-voltage signaling standards supported. In some of these standards the input buffer utilizes a user-supplied threshold voltage, V_{REF} . The need to supply V_{REF} imposes constraints on which standards can be used in close proximity to each other. See *I/O Banking*, page 3.

There are optional pull-up and pull-down resistors at each input for use after configuration.

Output Path

The output path includes a 3-state output buffer that drives the output signal onto the pad. The output signal can be routed to the buffer directly from the internal logic or through an optional IOB output flip-flop.

The 3-state control of the output can also be routed directly from the internal logic or through a flip-flop that provides synchronous enable and disable.

Each output driver can be individually programmed for a wide range of low-voltage signaling standards. Each output buffer can source up to 24 mA and sink up to 48 mA. Drive strength and slew rate controls minimize bus transients.

In most signaling standards, the output high voltage depends on an externally supplied V_{CCO} voltage. The need to supply V_{CCO} imposes constraints on which standards can be used in close proximity to each other. See *I/O Banking*.

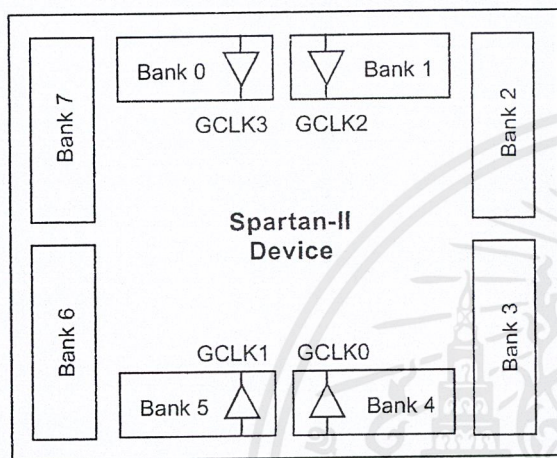
An optional weak-keeper circuit is connected to each output. When selected, the circuit monitors the voltage on the pad and weakly drives the pin High or Low to match the input signal. If the pin is connected to a multiple-source signal, the weak keeper holds the signal in its last state if all drivers are disabled. Maintaining a valid logic level in this way helps eliminate bus chatter.

Because the weak-keeper circuit uses the IOB input buffer to monitor the input level, an appropriate V_{REF} voltage must be provided if the signaling standard requires one. The provision of this voltage must comply with the I/O banking rules.

I/O Banking

Some of the I/O standards described above require V_{CC0} and/or V_{REF} voltages. These voltages are externally connected to device pins that serve groups of IOBs, called banks. Consequently, restrictions exist about which I/O standards can be combined within a given bank.

Eight I/O banks result from separating each edge of the FPGA into two banks (see Figure 2). Each bank has multiple V_{CC0} pins which must be connected to the same voltage. Voltage is determined by the output standards in use.



DS001_03_060100

Figure 2: Spartan-II I/O Banks

Within a bank, output standards may be mixed only if they use the same V_{CC0} . Compatible standards are shown in Table 2. GTL and GTL+ appear under all voltages because their open-drain outputs do not depend on V_{CC0} .

Table 2: Compatible Output Standards

V_{CC0}	Compatible Standards
3.3V	PCI, LVTTTL, SSTL3 I, SSTL3 II, CTT, AGP, GTL, GTL+
2.5V	SSTL2 I, SSTL2 II, LVCMOS2, GTL, GTL+
1.5V	HSTL I, HSTL III, HSTL IV, GTL, GTL+

Some input standards require a user-supplied threshold voltage, V_{REF} . In this case, certain user-I/O pins are automatically configured as inputs for the V_{REF} voltage. About one in six of the I/O pins in the bank assume this role.

V_{REF} pins within a bank are interconnected internally and consequently only one V_{REF} voltage can be used within

each bank. All V_{REF} pins in the bank, however, must be connected to the external voltage source for correct operation.

In a bank, inputs requiring V_{REF} can be mixed with those that do not but only one V_{REF} voltage may be used within a bank. Input buffers that use V_{REF} are not 5V tolerant. LVTTTL, LVCMOS2, and PCI are 5V tolerant. The V_{CC0} and V_{REF} pins for each bank appear in the device pinout tables.

Within a given package, the number of V_{REF} and V_{CC0} pins can vary depending on the size of device. In larger devices, more I/O pins convert to V_{REF} pins. Since these are always a superset of the V_{REF} pins used for smaller devices, it is possible to design a PCB that permits migration to a larger device. All V_{REF} pins for the largest device anticipated must be connected to the V_{REF} voltage, and not used for I/O.

Table 3: Independent Banks Available

Package	VQ100 PQ208	CS144 TQ144	FG256 FG456
Independent Banks	1	4	8

Configurable Logic Block

The basic building block of the Spartan-II CLB is the logic cell (LC). An LC includes a 4-input function generator, carry logic, and storage element. Output from the function generator in each LC drives the CLB output and the D input of the flip-flop. Each Spartan-II CLB contains four LCs, organized in two similar slices; a single slice is shown in Figure 3.

In addition to the four basic LCs, the Spartan-II CLB contains logic that combines function generators to provide functions of five or six inputs.

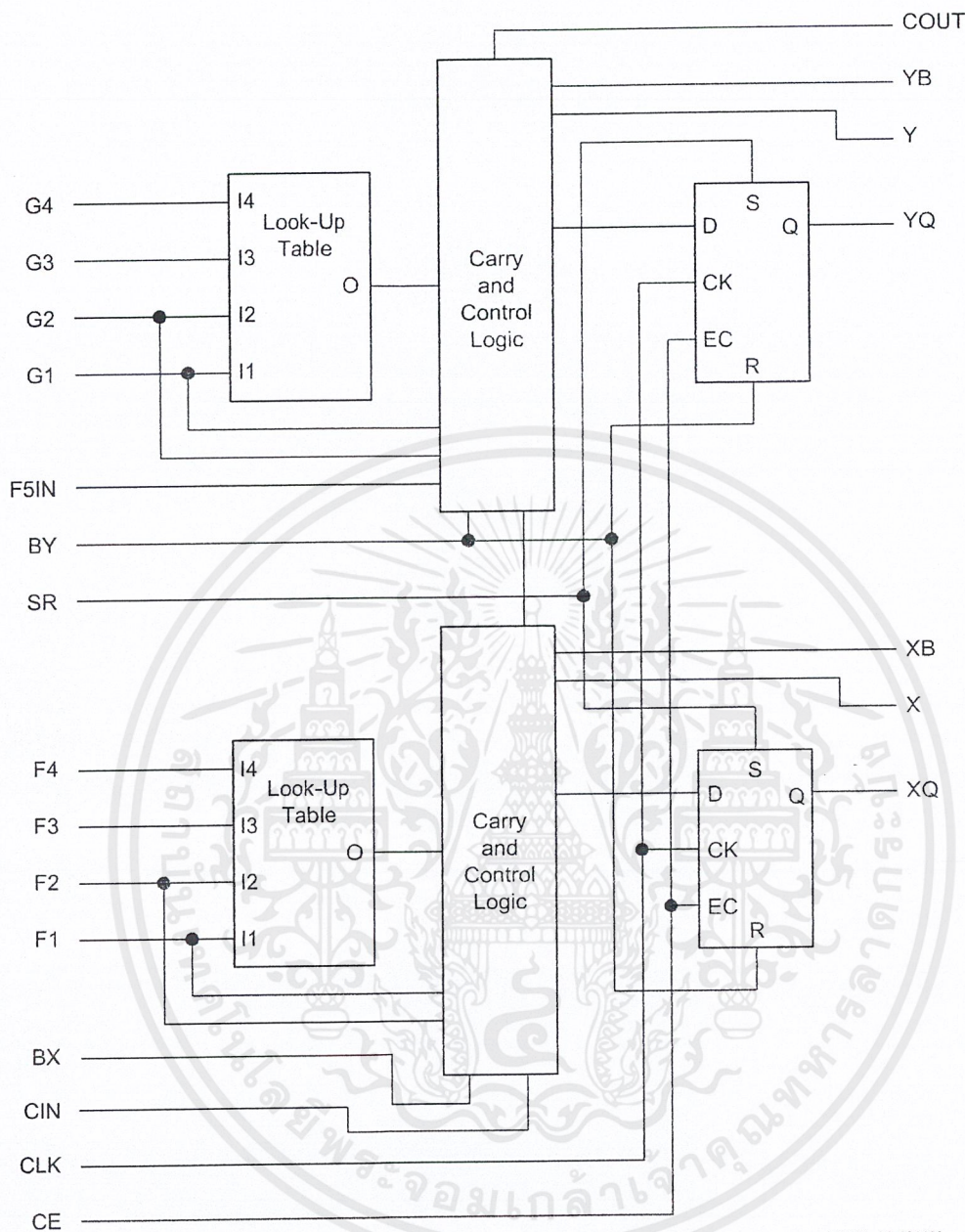
Look-Up Tables

Spartan-II function generators are implemented as 4-input look-up tables (LUTs). In addition to operating as a function generator, each LUT can provide a 16 x 1-bit synchronous RAM. Furthermore, the two LUTs within a slice can be combined to create a 16 x 2-bit or 32 x 1-bit synchronous RAM, or a 16 x 1-bit dual-port synchronous RAM.

The Spartan-II LUT can also provide a 16-bit shift register that is ideal for capturing high-speed or burst-mode data. This mode can also be used to store data in applications such as Digital Signal Processing.

Storage Elements

Storage elements in the Spartan-II slice can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches. The D inputs can be driven either by function generators within the slice or directly from slice inputs, bypassing the function generators.



DS001_04_091400

Figure 3: Spartan-II CLB Slice (two identical slices in each CLB)

In addition to Clock and Clock Enable signals, each slice has synchronous set and reset signals (SR and BY). SR forces a storage element into the initialization state specified for it in the configuration. BY forces it into the opposite state. Alternatively, these signals may be configured to operate asynchronously.

All control signals are independently invertible, and are shared by the two flip-flops within the slice.

Additional Logic

The F5 multiplexer in each slice combines the function generator outputs. This combination provides either a function generator that can implement any 5-input function, a 4:1 multiplexer, or selected functions of up to nine inputs.

Similarly, the F6 multiplexer combines the outputs of all four function generators in the CLB by selecting one of the F5-multiplexer outputs. This permits the implementation of any 6-input function, an 8:1 multiplexer, or selected functions of up to 19 inputs.

Each CLB has four direct feedthrough paths, one per LC. These paths provide extra data input lines or additional local routing that does not consume logic resources.

Arithmetic Logic

Dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions. The Spartan-II CLB supports two separate carry chains, one per slice. The height of the carry chains is two bits per CLB.

The arithmetic logic includes an XOR gate that allows a 1-bit full adder to be implemented within an LC. In addition, a dedicated AND gate improves the efficiency of multiplier implementation.

The dedicated carry path can also be used to cascade function generators for implementing wide logic functions.

BUFTs

Each Spartan-II CLB contains two 3-state drivers (BUFTs) that can drive on-chip busses. See *Dedicated Routing*, page 6. Each Spartan-II BUFT has an independent 3-state control pin and an independent input pin.

Block RAM

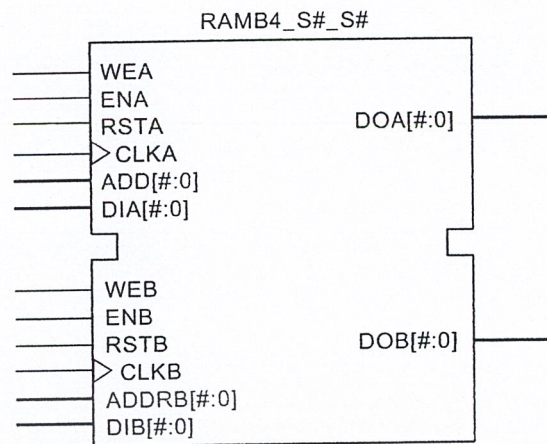
Spartan-II FPGAs incorporate several large block RAM memories. These complement the distributed RAM Look-Up Tables (LUTs) that provide shallow memory structures implemented in CLBs.

Block RAM memory blocks are organized in columns. All Spartan-II devices contain two such columns, one along each vertical edge. These columns extend the full height of the chip. Each memory block is four CLBs high, and consequently, a Spartan-II device eight CLBs high will contain two memory blocks per column, and a total of four blocks.

Table 4: Spartan-II Block RAM Amounts

Spartan-II Device	# of Blocks	Total Block RAM Bits
XC2S15	4	16K
XC2S30	6	24K
XC2S50	8	32K
XC2S100	10	40K
XC2S150	12	48K
XC2S200	14	56K

Each block RAM cell, as illustrated in Figure 4, is a fully synchronous dual-ported 4096-bit RAM with independent control signals for each port. The data widths of the two ports can be configured independently, providing built-in bus-width conversion.



DS001_05_060100

Figure 4: Dual-Port Block RAM

Table 5 shows the depth and width aspect ratios for the block RAM.

Table 5: Block RAM Port Aspect Ratios

Width	Depth	ADDR Bus	Data Bus
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

The Spartan-II block RAM also includes dedicated routing to provide an efficient interface with both CLBs and other block RAMs.

Programmable Routing Matrix

It is the longest delay path that limits the speed of any worst-case design. Consequently, the Spartan-II routing architecture and its place-and-route software were defined in a single optimization process. This joint optimization minimizes long-path delays, and consequently, yields the best system performance.

The joint optimization also reduces design compilation times because the architecture is software-friendly. Design cycles are correspondingly reduced due to shorter design iteration times.

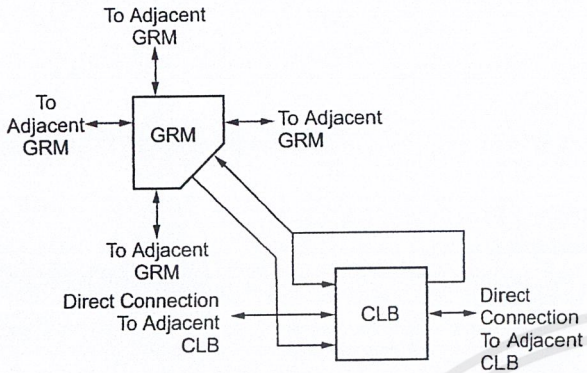
Local Routing

The local routing resources, as shown in Figure 5, provide the following three types of connections:

- Interconnections among the LUTs, flip-flops, and General Routing Matrix (GRM)
- Internal CLB feedback paths that provide high-speed connections to LUTs within the same CLB, chaining

them together with minimal routing delay

- Direct paths that provide high-speed connections between horizontally adjacent CLBs, eliminating the delay of the GRM



DS001_06_032300

Figure 5: Spartan-II Local Routing

General Purpose Routing

Most Spartan-II signals are routed on the general purpose routing, and consequently, the majority of interconnect resources are associated with this level of the routing hierarchy. The general routing resources are located in horizontal and vertical routing channels associated with the rows and columns CLBs. The general-purpose routing resources are listed below.

- Adjacent to each CLB is a General Routing Matrix (GRM). The GRM is the switch matrix through which horizontal and vertical routing resources connect, and is also the means by which the CLB gains access to the general purpose routing.
- 24 single-length lines route GRM signals to adjacent

GRMs in each of the four directions.

- 96 buffered Hex lines route GRM signals to other GRMs six blocks away in each one of the four directions. Organized in a staggered pattern, Hex lines may be driven only at their endpoints. Hex-line signals can be accessed either at the endpoints or at the midpoint (three blocks from the source). One third of the Hex lines are bidirectional, while the remaining ones are unidirectional.
- 12 Longlines are buffered, bidirectional wires that distribute signals across the device quickly and efficiently. Vertical Longlines span the full height of the device, and horizontal ones span the full width of the device.

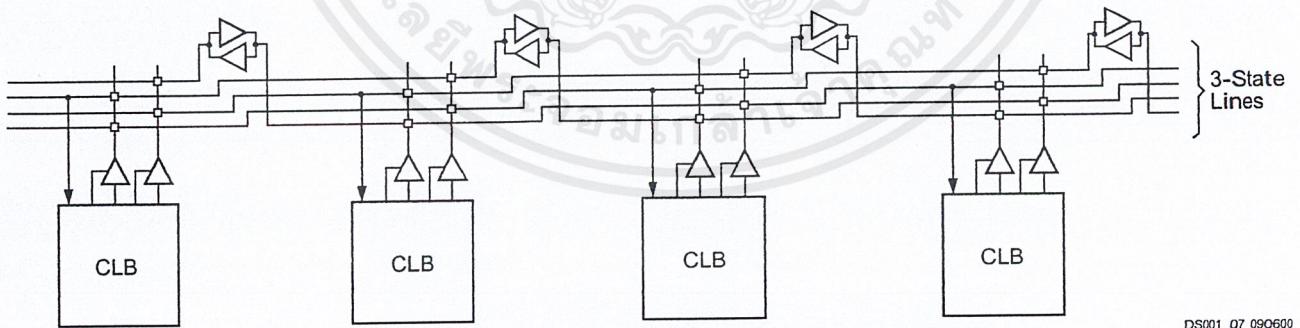
I/O Routing

Spartan-II devices have additional routing resources around their periphery that form an interface between the CLB array and the IOBs. This additional routing, called the VersaRing, facilitates pin-swapping and pin-locking, such that logic redesigns can adapt to existing PCB layouts. Time-to-market is reduced, since PCBs and other system components can be manufactured while the logic design is still in progress.

Dedicated Routing

Some classes of signal require dedicated routing resources to maximize performance. In the Spartan-II architecture, dedicated routing resources are provided for two classes of signal.

- Horizontal routing resources are provided for on-chip 3-state busses. Four partitionable bus lines are provided per CLB row, permitting multiple busses within a row, as shown in Figure 6.
- Two dedicated nets per CLB propagate carry signals vertically to the adjacent CLB.



DS001_07_090600

Figure 6: BUFT Connections to Dedicated Horizontal Bus Lines

Global Routing

Global Routing resources distribute clocks and other signals with very high fanout throughout the device. Spartan-II devices include two tiers of global routing resources referred to as primary and secondary global routing resources.

- The primary global routing resources are four dedicated global nets with dedicated input pins that are designed to distribute high-fanout clock signals with minimal skew. Each global clock net can drive all CLB, IOB, and block RAM clock pins. The primary global nets may only be driven by global buffers. There are four global buffers, one for each global net.
- The secondary global routing resources consist of 24 backbone lines, 12 across the top of the chip and 12 across bottom. From these lines, up to 12 unique signals per column can be distributed via the 12 longlines in the column. These secondary resources are more flexible than the primary resources since they are not restricted to routing only to clock pins.

Clock Distribution

The Spartan-II family provides high-speed, low-skew clock distribution through the primary global routing resources described above. A typical clock distribution net is shown in Figure 7.

Four global buffers are provided, two at the top center of the device and two at the bottom center. These drive the four primary global nets that in turn drive any clock pin.

Four dedicated clock pads are provided, one adjacent to each of the global buffers. The input to the global buffer is selected either from these pads or from signals in the general purpose routing.

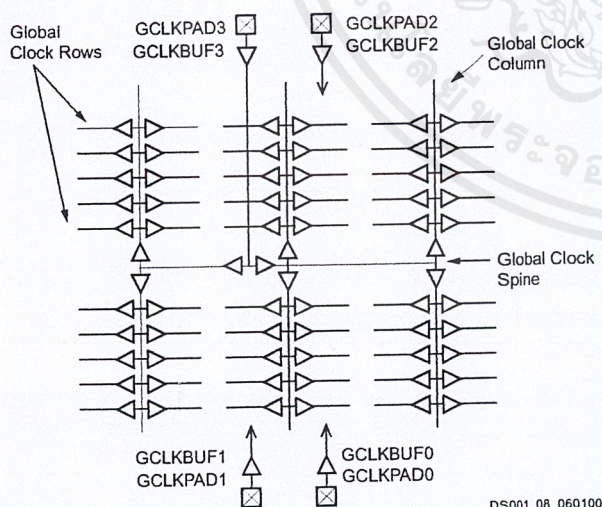


Figure 7: Global Clock Distribution Network

Delay-Locked Loop (DLL)

Associated with each global clock input buffer is a fully digital Delay-Locked Loop (DLL) that can eliminate skew between the clock input pad and internal clock-input pins throughout the device. Each DLL can drive two global clock networks. The DLL monitors the input clock and the distributed clock, and automatically adjusts a clock delay element. Additional delay is introduced such that clock edges reach internal flip-flops exactly one clock period after they arrive at the input. This closed-loop system effectively eliminates clock-distribution delay by ensuring that clock edges arrive at internal flip-flops in synchronism with clock edges arriving at the input.

In addition to eliminating clock-distribution delay, the DLL provides advanced control of multiple clock domains. The DLL provides four quadrature phases of the source clock, can double the clock, or divide the clock by 1.5, 2, 2.5, 3, 4, 5, 8, or 16. It has six outputs.

The DLL also operates as a clock mirror. By driving the output from a DLL off-chip and then back on again, the DLL can be used to deskew a board level clock among multiple Spartan-II devices.

In order to guarantee that the system clock is operating correctly prior to the FPGA starting up after configuration, the DLL can delay the completion of the configuration process until after it has achieved lock.

Boundary Scan

Spartan-II devices support all the mandatory boundary-scan instructions specified in the IEEE standard 1149.1. A Test Access Port (TAP) and registers are provided that implement the EXTEST, SAMPLE/PRELOAD, and BYPASS instructions. The TAP also supports two USERCODE instructions and internal scan chains.

The TAP uses dedicated package pins that always operate using LVTTTL. For TDO to operate using LVTTTL, the V_{CC0} for Bank 2 must be 3.3V. Otherwise, TDO switches rail-to-rail between ground and V_{CC0} .

Boundary-scan operation is independent of individual IOB configurations, and unaffected by package type. All IOBs, including unbonded ones, are treated as independent 3-state bidirectional pins in a single scan chain. Retention of the bidirectional test capability after configuration facilitates the testing of external interconnections.

Table 6 lists the boundary-scan instructions supported in Spartan-II FPGAs. Internal signals can be captured during EXTEST by connecting them to unbonded or unused IOBs. They may also be connected to the unused outputs of IOBs defined as unidirectional input pins. This technique partially compensates for the absence of INTEST support.

Table 6: Boundary-Scan Instructions

Boundary-Scan Command	Binary Code[4:0]	Description
EXTEST	00000	Enables boundary-scan EXTEST operation
SAMPLE	00001	Enables boundary-scan SAMPLE operation
USR1	00010	Access user-defined register 1
USR2	00011	Access user-defined register 2
CFG_OUT	00100	Access the configuration bus for Readback
CFG_IN	00101	Access the configuration bus for Configuration
INTEST	00111	Enables boundary-scan INTEST operation
USRCODE	01000	Enables shifting out USER code
IDCODE	01001	Enables shifting out of ID Code
HIZ	01010	Disables output pins while enabling the Bypass Register
JSTART	01100	Clock the start-up sequence when StartupClk is TCK
BYPASS	11111	Enables BYPASS
RESERVED	All other codes	Xilinx reserved instructions

The public boundary-scan instructions are available prior to configuration. After configuration, the public instructions remain available together with any USERCODE instructions installed during the configuration. While the SAMPLE and BYPASS instructions are available during configuration, it is recommended that boundary-scan operations not be performed during this transitional period.

In addition to the test instructions outlined above, the boundary-scan circuitry can be used to configure the FPGA, and also to read back the configuration data.

To facilitate internal scan chains, the User Register provides three outputs (Reset, Update, and Shift) that represent the corresponding states in the boundary-scan internal state machine.

Figure 8 is a diagram of the Spartan-II family boundary scan logic. It includes three bits of Data Register per IOB, the IEEE 1149.1 Test Access Port controller, and the Instruction Register with decodes.

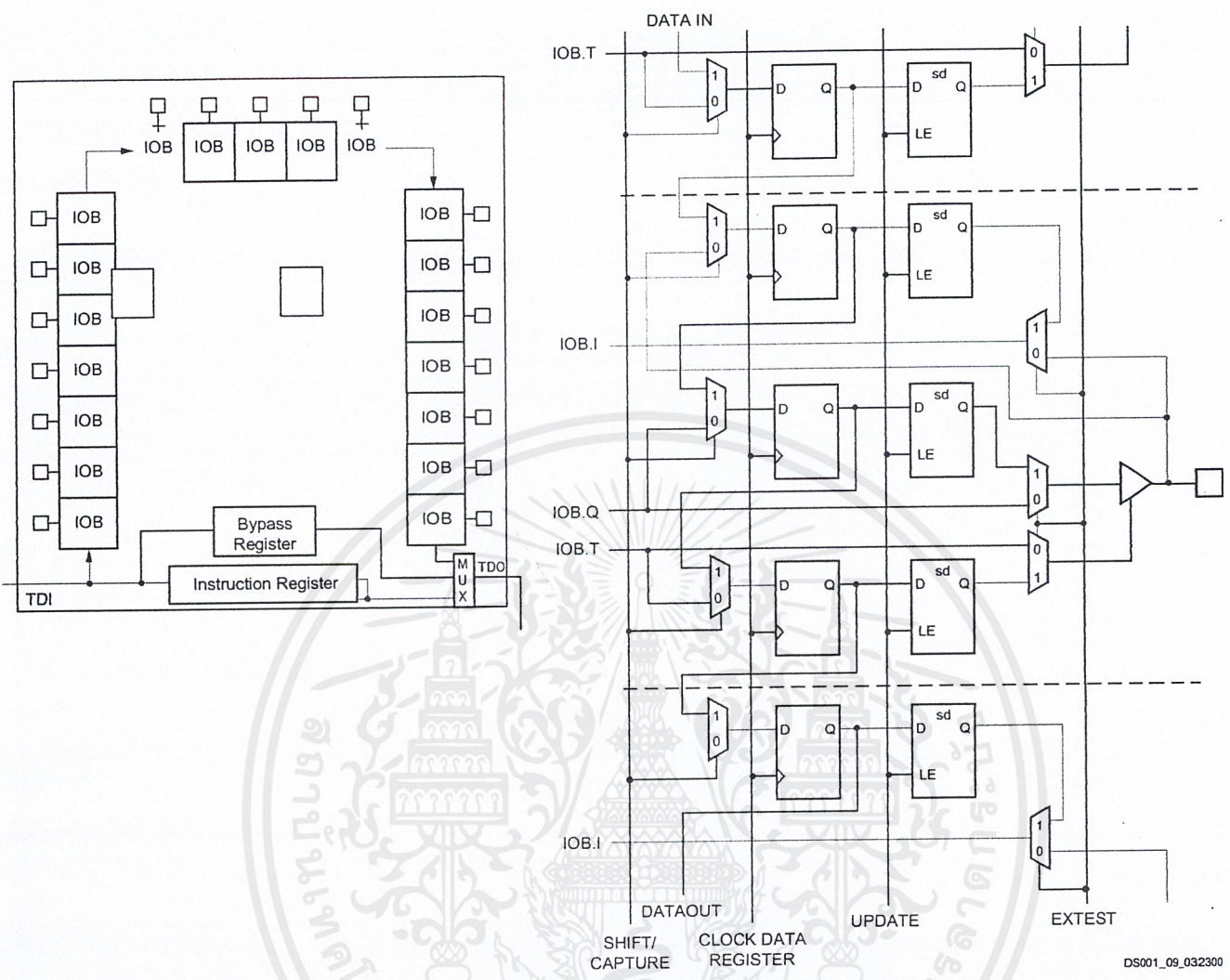


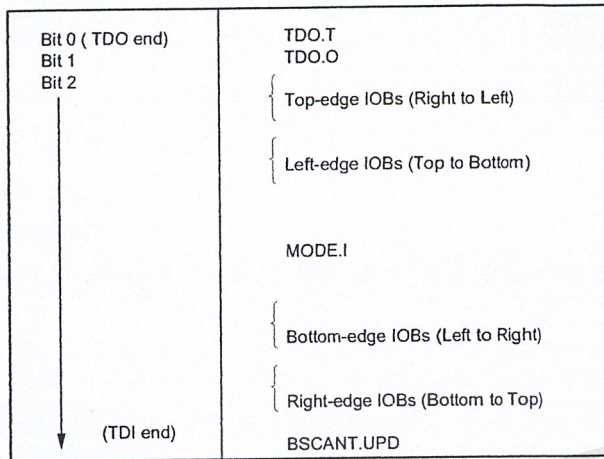
Figure 8: Spartan-II Family Boundary Scan Logic

Bit Sequence

The bit sequence within each IOB is: In, Out, 3-State. The input-only pins contribute only the In bit to the boundary scan I/O data register, while the output-only pins contributes all three bits.

From a cavity-up view of the chip (as shown in the FPGA Editor), starting in the upper right chip corner, the boundary scan data-register bits are ordered as shown in Figure 9.

BSDL (Boundary Scan Description Language) files for Spartan-II family devices are available on the Xilinx website, in the File Download area.



DS001_10_032300

Figure 9: Boundary Scan Bit Sequence

Development System

Spartan-II FPGAs are supported by the Xilinx Foundation and Alliance CAE tools. The basic methodology for Spartan-II design consists of three interrelated steps: design entry, implementation, and verification. Industry-standard tools are used for design entry and simulation (for example, Synopsys FPGA Express), while Xilinx provides proprietary architecture-specific tools for implementation.

The Xilinx development system is integrated under the Xilinx Design Manager software, providing designers with a common user interface regardless of their choice of entry and verification tools. The software simplifies the selection of implementation options with pull-down menus and on-line help.

Application programs ranging from schematic capture to Placement and Routing (PAR) can be accessed through the software. The program command sequence is generated prior to execution, and stored for documentation.

Several advanced software features facilitate Spartan-II design. Relationally-Placed Macros (RPMs), for example, are schematic-based macros with relative location constraints to guide their placement. They help ensure optimal implementation of common functions.

For HDL design entry, the Xilinx FPGA development system provides interfaces to several synthesis design environments.

A standard interface-file specification, Electronic Design Interchange Format (EDIF), simplifies file transfers into and out of the development system.

Spartan-II FPGAs supported by a unified library of standard functions. This library contains over 400 primitives and macros, ranging from 2-input AND gates to 16-bit accumulators, and includes arithmetic functions, comparators, counters, data registers, decoders, encoders, I/O functions, latches,

Boolean functions, multiplexers, shift registers, and barrel shifters.

The "soft macro" portion of the library contains detailed descriptions of common logic functions, but does not contain any partitioning or placement information. The performance of these macros depends, therefore, on the partitioning and placement obtained during implementation.

RPMs, on the other hand, do contain predetermined partitioning and placement information that permits optimal implementation of these functions. Users can create their own library of soft macros or RPMs based on the macros and primitives in the standard library.

The design environment supports hierarchical design entry, with high-level schematics that comprise major functional blocks, while lower-level schematics define the logic in these blocks. These hierarchical design elements are automatically combined by the implementation tools. Different design entry tools can be combined within a hierarchical design, thus allowing the most convenient entry method to be used for each portion of the design.

Design Implementation

The place-and-route tools (PAR) automatically provide the implementation flow described in this section. The partitioner takes the EDIF netlist for the design and maps the logic into the architectural resources of the FPGA (CLBs and IOBs, for example). The placer then determines the best locations for these blocks based on their interconnections and the desired performance. Finally, the router interconnects the blocks.

The PAR algorithms support fully automatic implementation of most designs. For demanding applications, however, the user can exercise various degrees of control over the process. User partitioning, placement, and routing information is optionally specified during the design-entry process. The implementation of highly structured designs can benefit greatly from basic floorplanning.

The implementation software incorporates Timing Wizard[®] timing-driven placement and routing. Designers specify timing requirements along entire paths during design entry. The timing path analysis routines in PAR then recognize these user-specified requirements and accommodate them.

Timing requirements are entered on a schematic in a form directly relating to the system requirements, such as the targeted clock frequency, or the maximum allowable delay between two registers. In this way, the overall performance of the system along entire signal paths is automatically tailored to user-generated specifications. Specific timing information for individual nets is unnecessary.

Design Verification

In addition to conventional software simulation, FPGA users can use in-circuit debugging techniques. Because Xilinx devices are infinitely reprogrammable, designs can be veri-

fied in real time without the need for extensive sets of software simulation vectors.

The development system supports both software simulation and in-circuit debugging techniques. For simulation, the system extracts the post-layout timing information from the design database, and back-annotates this information into the netlist for use by the simulator. Alternatively, the user can verify timing-critical portions of the design using the static timing analyzer.

For in-circuit debugging, the development system includes a download and readback cable, which connects the FPGA in the target system to a PC or workstation. After downloading the design into the FPGA, the designer can single-step the logic, readback the contents of the flip-flops, and so observe the internal logic state. Simple modifications can be downloaded into the system in a matter of minutes.

Configuration

Configuration is the process by which the bitstream of a design, as generated by the Xilinx development software, is loaded into the internal configuration memory of the FPGA. Spartan-II devices support both serial configuration, using the master/slave serial and JTAG modes, as well as byte-wide configuration employing the Slave Parallel mode.

Configuration File

Spartan-II devices are configured by sequentially loading frames of data that have been concatenated into a configuration file. Table 7 shows how much nonvolatile storage space is needed for Spartan-II devices.

It is important to note that, while a PROM is commonly used to store configuration data before loading them into the FPGA, it is by no means required. Any of a number of different kinds of under populated nonvolatile storage already available either on or off the board (i.e., hard drives, FLASH

cards, etc.) can be used. For more information on configuration without a PROM, refer to **XAPP098, The Low-Cost, Efficient Serial Configuration of Spartan FPGAs.**

Table 7: Spartan-II Configuration File Size

Device	Configuration File Size (Bits)
XC2S15	197,696
XC2S30	336,768
XC2S50	559,200
XC2S100	781,216
XC2S150	1,040,096
XC2S200	1,335,840

Modes

Spartan-II devices support the following four configuration modes:

- Slave Serial mode
- Master Serial mode
- Slave Parallel mode
- Boundary-scan mode

The Configuration mode pins (M2, M1, M0) select among these configuration modes with the option in each case of having the IOB pins either pulled up or left floating prior to configuration. The selection codes are listed in Table 8.

Configuration through the boundary-scan port is always available, independent of the mode selection. Selecting the boundary-scan mode simply turns off the other modes. The three mode pins have internal pull-up resistors, and default to a logic High if left unconnected.

Table 8: Configuration Modes

Configuration Mode	Preconfiguration Pull-ups	M0	M1	M2	CCLK Direction	Data Width	Serial D _{OUT}
Master Serial mode	No	0	0	0	Out	1	Yes
	Yes	0	0	1			
Slave Parallel mode	Yes	0	1	0	In	8	No
	No	0	1	1			
Boundary-Scan mode	Yes	1	0	0	N/A	1	No
	No	1	0	1			
Slave Serial mode	Yes	1	1	0	In	1	Yes
	No	1	1	1			

Signals

There are two kinds of pins that are used to configure Spartan-II devices: Dedicated pins perform only specific configuration-related functions; the other pins can serve as general purpose I/Os once user operation has begun.

The dedicated pins comprise the mode pins (M2, M1, M0), the configuration clock pin (CCLK), the PROGRAM pin, the DONE pin and the boundary-scan pins (TDI, TDO, TMS, TCK). Depending on the selected configuration mode, CCLK may be an output generated by the FPGA, or may be generated externally, and provided to the FPGA as an input.

Note that some configuration pins can act as outputs. For correct operation, these pins require a V_{CC0} of 3.3V to drive an LVTTTL signal. All the relevant pins fall in banks 2 or 3.

For a more detailed description than that given below, see DS001-4, *Spartan-II 2.5V FPGA Family: Pinout Tables* and XAPP176, *Spartan-II FPGA Series Configuration and Readback*.

The Process

The sequence of steps necessary to configure Spartan-II devices are shown in Figure 10. The overall flow can be divided into three different phases.

- Initiating Configuration
- Configuration memory clear
- Loading data frames
- Start-up

The memory clearing and start-up phases are the same for all configuration modes; however, the steps for the loading of data frames are different. Thus, the details for data frame loading are described separately in the sections devoted to each mode.

Initiating Configuration

There are two different ways to initiate the configuration process: applying power to the device or asserting the PROGRAM input.

Configuration on power-up occurs automatically unless it is delayed by the user, as described in a separate section below. The waveform for configuration on power-up is shown in Figure 11, page 13. Before configuration can begin, V_{CC0} Bank 2 must be greater than 1.0V. Furthermore, all V_{CCINT} power pins must be connected to a 2.5V supply. For more information on delaying configuration, see *Clearing Configuration Memory*, page 13.

Once in user operation, the device can be re-configured simply by pulling the PROGRAM pin Low. The device acknowledges the beginning of the configuration process by driving DONE Low, then enters the memory-clearing phase.

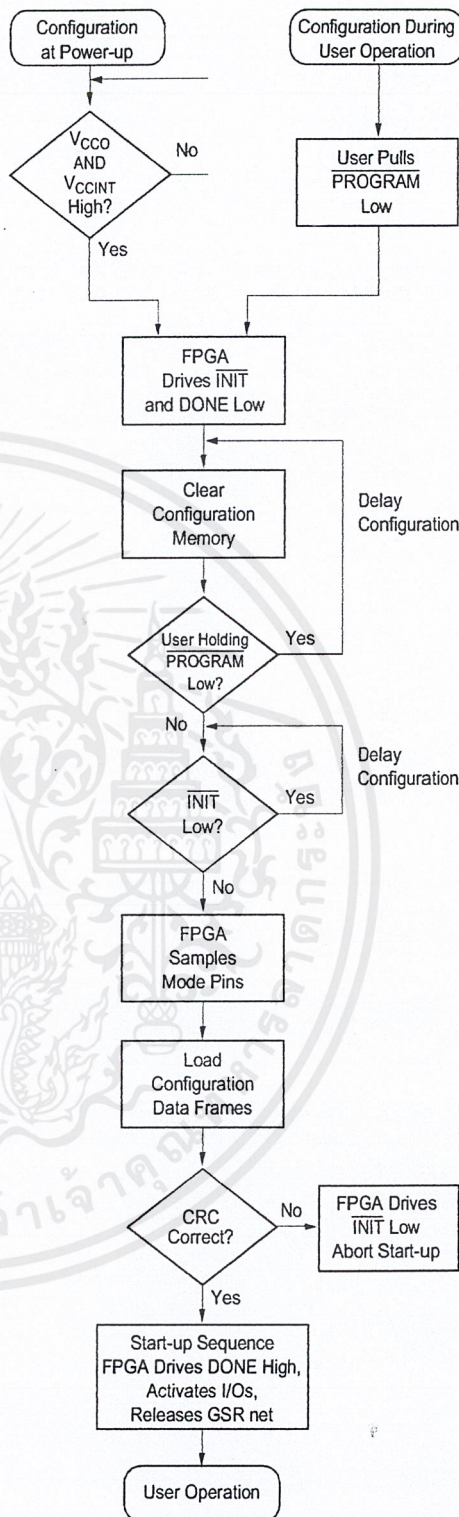
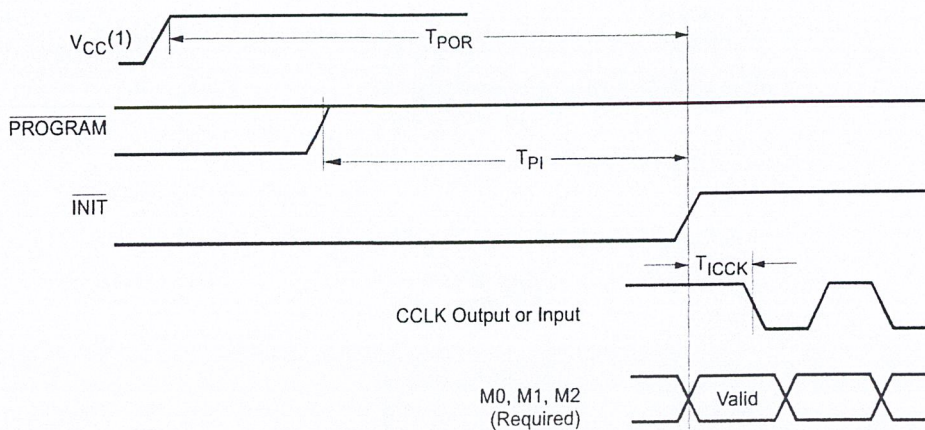


Figure 10: Configuration Flow Diagram

DS001_11_090600



DS001_12_032300

Symbol	Description		Units
T_{POR}	Power-on reset	2	ms, max
T_{PL}	Program latency	100	μ s, max
T_{ICCK}	CCLK output delay (Master Serial mode only)	0.5	μ s, min
		4	μ s, max
$T_{PROGRAM}$	Program pulse width	300	ns, min

Notes: referring to waveform above:

- Before configuration can begin, V_{CCINT} must be greater than 1.6V and V_{CCO} Bank 2 must be greater than 1.0V.

Figure 11: Configuration Timing on Power-Up

Clearing Configuration Memory

The device indicates that clearing the configuration memory is in progress by driving \overline{INIT} Low. At this time, the user can delay configuration by holding either $\overline{PROGRAM}$ or \overline{INIT} Low, which causes the device to remain in the memory clearing phase. Note that the bidirectional \overline{INIT} line is driving a Low logic level during memory clearing. Thus, to avoid contention, use an open-drain driver to keep \overline{INIT} Low.

With no delay in force, the device indicates that the memory is completely clear by driving \overline{INIT} High. The FPGA samples its mode pins on this Low-to-High transition.

Loading Configuration Data

Once \overline{INIT} is High, the user can begin loading configuration data frames into the device. The details of loading the configuration data are discussed in the sections treating the configuration modes individually. The sequence of operations necessary to load configuration data using the serial modes is shown in Figure 13. Loading data using the Slave Parallel mode is shown in Figure 18, page 18.

CRC Error Checking

During the loading of configuration data, a CRC value embedded in the configuration file is checked against a CRC value calculated within the FPGA. If the CRC values

do not match, the FPGA drives \overline{INIT} Low to indicate that a frame error has occurred and configuration is aborted.

To reconfigure the device, the $\overline{PROGRAM}$ pin should be asserted to reset the configuration logic. Recycling power also resets the FPGA for configuration. See Clearing Configuration Memory.

Start-up

The start-up sequence oversees the transition of the FPGA from the configuration state to full user operation. A match of CRC values, indicating a successful loading of the configuration data, initiates the sequence.

During start-up, the device performs four operations:

1. The assertion of DONE. The failure of DONE to go High may indicate the unsuccessful loading of configuration data.
2. The release of the Global Three State. This activates all the I/Os.
3. Negates Global Set Reset (GSR). This allows all flip-flops to change state.
4. The assertion of Global Write Enable (GWE). This allows all RAMs and flip-flops to change state.

By default, these operations are synchronized to CCLK. The entire start-up sequence lasts eight cycles, called C0-C7, after which the loaded design is fully functional. The default timing for start-up is shown in the top half of Figure 12. The four operations can be selected to switch on any CCLK cycle C1-C6 through settings in the Xilinx Development Software. Heavy lines show default settings.

The bottom half of Figure 12 shows another commonly used version of the start-up timing known as Sync-to-DONE. This version makes the GTS, GSR, and GWE events conditional upon the DONE pin going High. This timing is important for a daisy chain of multiple FPGAs in serial mode, since it ensures that all FPGAs go through start-up together, after all their DONE pins have gone High.

Sync-to-DONE timing is selected by setting the GTS, GSR, and GWE cycles to a value of DONE in the configuration options. This causes these signals to transition one clock cycle after DONE externally transitions High.

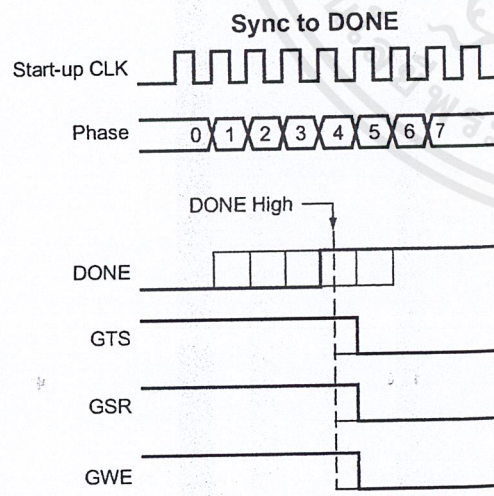
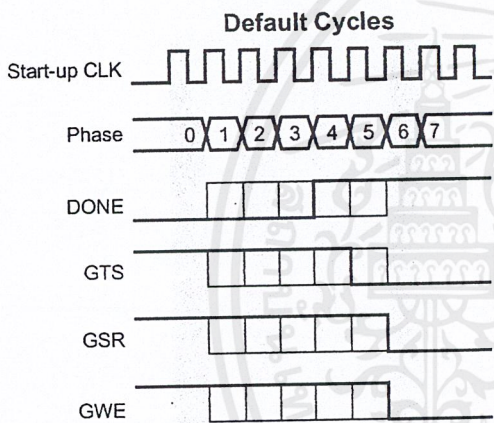


Figure 12: Start-Up Waveforms

Serial Modes

There are two serial configuration modes: In Master Serial mode, the FPGA controls the configuration process by driving CCLK as an output. In Slave Serial mode, the FPGA passively receives CCLK as an input from an external agent (e.g., a microprocessor, CPLD, or second FPGA in master mode) that is controlling the configuration process. In both modes, the FPGA is configured by loading one bit per CCLK cycle. The MSB of each configuration data byte is always written to the DIN pin first.

See Figure 13 for the sequence for loading data into the Spartan-II FPGA serially. This is an expansion of the "Load Configuration Data Frames" block in Figure 10, page 12.

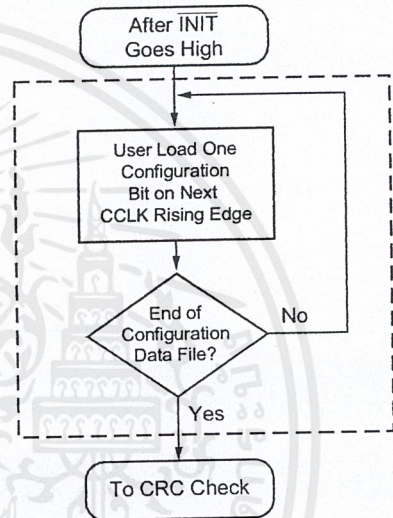
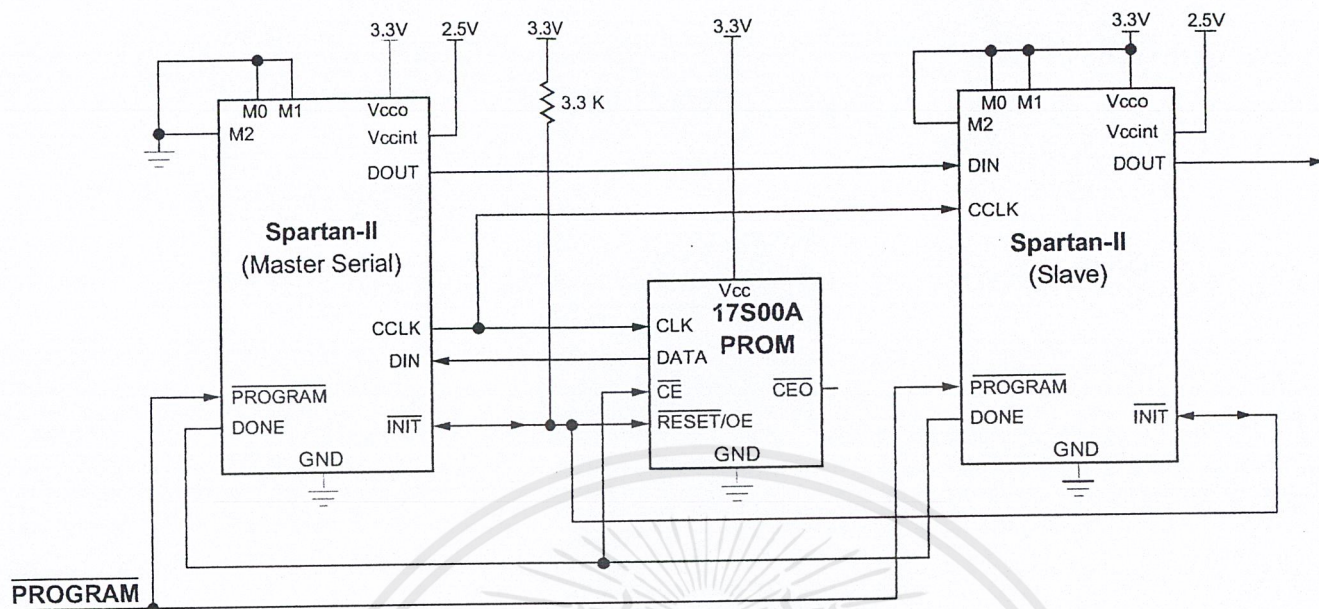


Figure 13: Loading Serial Mode Configuration Data

Slave Serial Mode

In Slave Serial mode, the FPGAs CCLK pin is driven by an external source, allowing FPGAs to be configured from other logic devices such as microprocessors or in a daisy-chain configuration. Figure 14 shows connections for a Master Serial FPGA configuring a Slave Serial FPGA from a PROM. A Spartan-II device in slave serial mode should be connected as shown for the third device from the left. Slave Serial mode is selected by a <11x> on the mode pins (M0, M1, M2).

Figure 15 shows the timing for Slave Serial configuration. The serial bitstream must be setup at the DIN input pin a short time before each rising edge of an externally generated CCLK. Multiple FPGAs in Slave Serial mode can be daisy-chained for configuration from a single source. After an FPGA is configured, data for the next device is routed to the DOUT pin. Data on the DOUT pin changes on the rising edge of CCLK. Configuration must be delayed until INIT pins of all daisy-chained FPGAs are High. For more information, see Start-up, page 13.

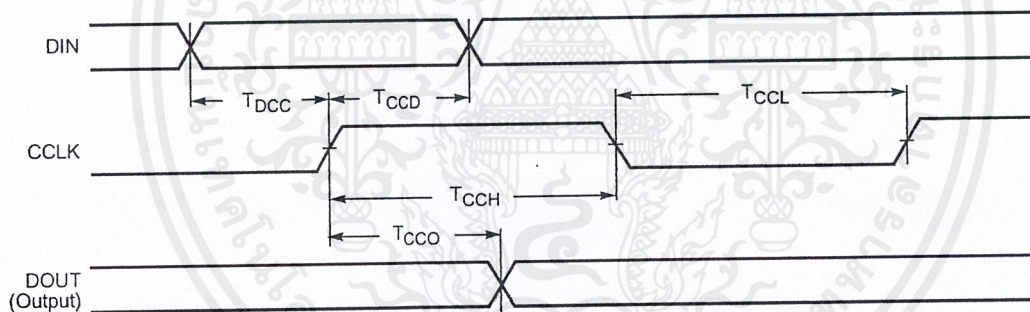


DS001_15_022601

Notes:

1. If the DriveDone configuration option is not active for any of the FPGAs, pull up DONE with a 3.3K Ω resistor.

Figure 14: Master/Slave Serial Configuration Circuit Diagram



DS001_16_032300

Symbol		Description		Units
T_{DCC}	CCLK	DIN setup	5	ns, min
T_{CCD}		DIN hold	0	ns, min
T_{CCO}		DOUT	12	ns, max
T_{CCH}		High time	5	ns, min
T_{CCL}		Low time	5	ns, min
F_C		Maximum frequency	66	MHz, max

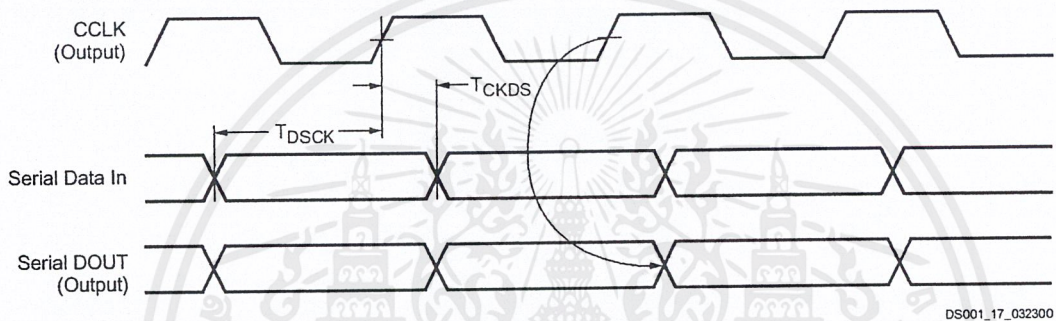
Figure 15: Slave Serial Mode Timing

Master Serial Mode

In Master Serial mode, the CCLK output of the FPGA drives a Xilinx PROM which feeds a serial stream of configuration data to the FPGA's DIN input. Figure 14 shows a Master Serial FPGA configuring a Slave Serial FPGA from a PROM. A Spartan-II device in Master Serial mode should be connected as shown for the device on the left side. Master Serial mode is selected by a <00x> on the mode pins (M0, M1, M2). The PROM RESET pin is driven by INIT, and CE input is driven by DONE. The interface is identical to the slave serial mode except that an oscillator internal to the FPGA is used to generate the configuration clock (CCLK). Any of a number of different frequencies ranging from 4 to 60 MHz can be set using the ConfigRate option in the Xilinx development software. On power-up, while the first 60 bytes

of the configuration data are being loaded, the CCLK frequency is always 2.5 MHz. This frequency is used until the ConfigRate bits, part of the configuration file, have been loaded into the FPGA, at which point, the frequency changes to the selected ConfigRate. Unless a different frequency is specified in the design, the default ConfigRate is 4 MHz. The period of the CCLK signal created by the internal oscillator has a variance of +45%, -30% from the specified value.

Figure 16 shows the timing for Master Serial configuration. The FPGA accepts one bit of configuration data on each rising CCLK edge. After the FPGA has been loaded, the data for the next device in a daisy-chain is presented on the DOUT pin after the rising CCLK edge.



DS001_17_032300

Symbol		Description		Units
T_{DSCK}	CCLK	DIN setup	5.0	ns, min
T_{CKDS}		DIN hold	0.0	ns, min
		Frequency tolerance with respect to nominal	+45%, -30%	-

Figure 16: Master Serial Mode Timing

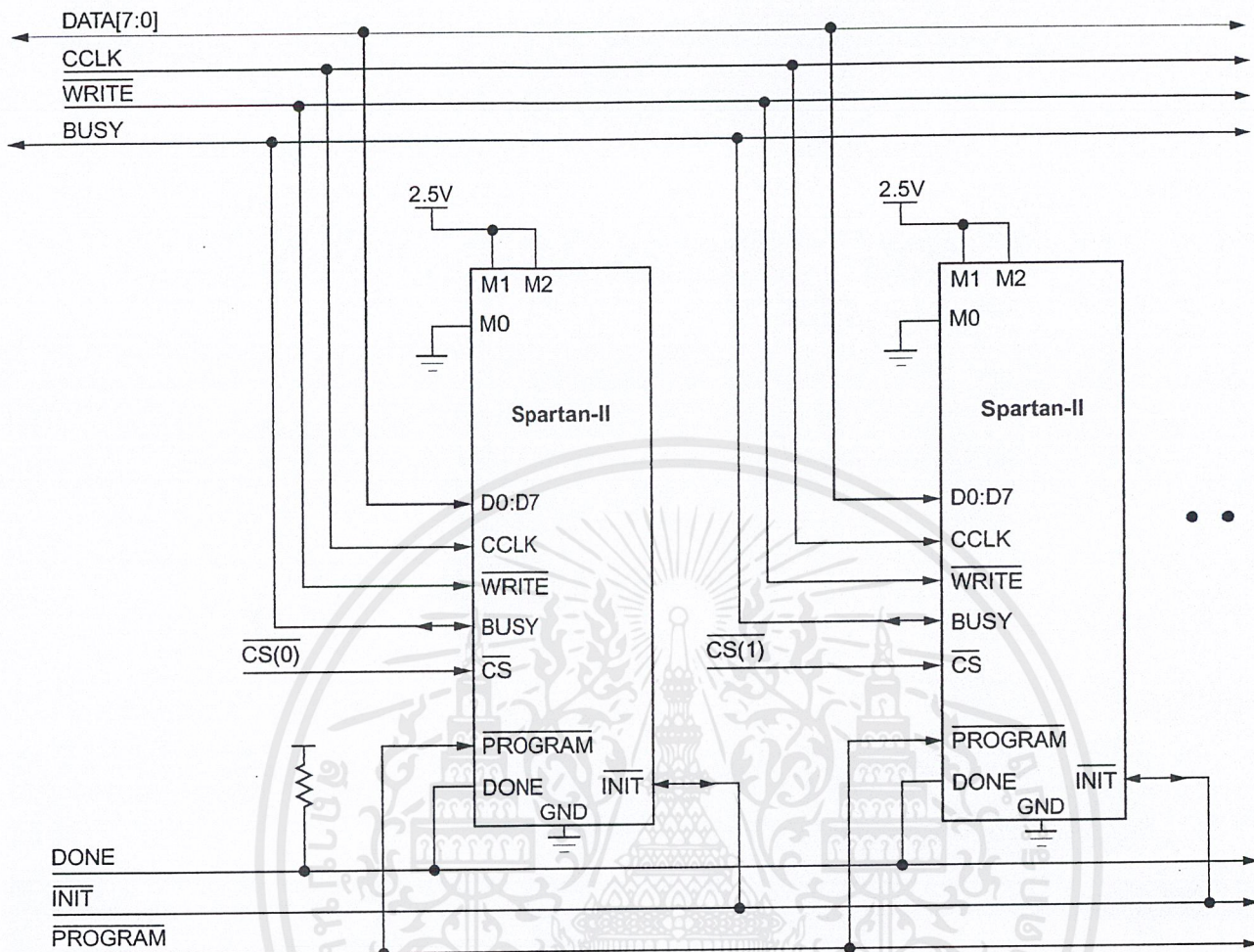
Slave Parallel Mode

The Slave Parallel mode is the fastest configuration option. Byte-wide data is written into the FPGA. A BUSY flag is provided for controlling the flow of data at a clock frequency F_{CCNH} above 50 MHz.

Figure 17, page 17 shows the connections for two Spartan-II devices using the Slave Parallel mode. Slave Parallel mode is selected by a <011> on the mode pins (M0, M1, M2).

The agent controlling configuration is not shown. Typically, a processor, a microcontroller, or CPLD controls the Slave Parallel interface. The controlling agent provides byte-wide configuration data, CCLK, a Chip Select (CS) signal and a Write signal (WRITE). If BUSY is asserted (High) by the FPGA, the data must be held until BUSY goes Low.

After configuration, the pins of the Slave Parallel port (D0-D7) can be used as additional user I/O. Alternatively, the port may be retained to permit high-speed 8-bit read-back. Then data can be read by de-asserting WRITE. See Readback, page 18.



DS001_18_032300

Figure 17: Slave Parallel Configuration Circuit Diagram

Multiple Spartan-II FPGAs can be configured using the Slave Parallel mode, and be made to start-up simultaneously. To configure multiple devices in this way, wire the individual CCLK, Data, WRITE, and BUSY pins of all the devices in parallel. The individual devices are loaded separately by asserting the CS pin of each device in turn and writing the appropriate data. Sync-to-DONE start-up timing is used to ensure that the start-up sequence does not begin until all the FPGAs have been loaded. See Start-up, page 13.

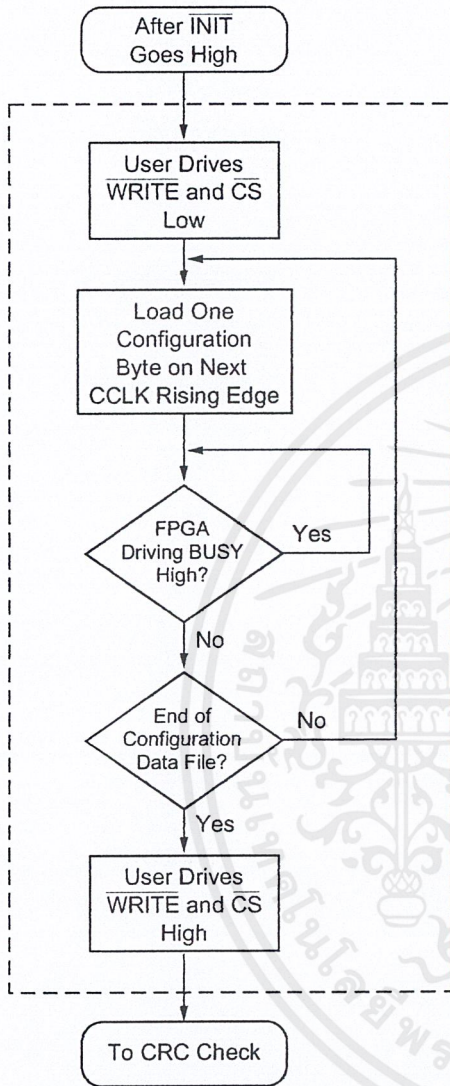
Write

When using the Slave Parallel Mode, write operations send packets of byte-wide configuration data into the FPGA. Figure 18, page 18 shows a flowchart of the write sequence used to load data into the Spartan-II FPGA. This is an expansion of the "Load Configuration Data Frames" block in Figure 10, page 12. The timing for write operations is shown in Figure 19, page 19.

For the present example, the user holds WRITE and CS Low throughout the sequence of write operations. Note that when CS is asserted on successive CCLKs, WRITE must remain either asserted or de-asserted. Otherwise an abort will be initiated, as in the next section.

1. Drive data onto D0-D7. Note that to avoid contention, the data source should not be enabled while CS is Low and WRITE is High. Similarly, while WRITE is High, no more than one device's CS should be asserted.
2. On the rising edge of CCLK: If BUSY is Low, the data is accepted on this clock. If BUSY is High (from a previous write), the data is not accepted. Acceptance will instead occur on the first clock after BUSY goes Low, and the data must be held until this happens.
3. Repeat steps 2 and 3 until all the data has been sent.
4. De-assert CS and WRITE.

If CCLK is slower than F_{CCNH} , the FPGA will never assert BUSY. In this case, the above handshake is unnecessary, and data can simply be entered into the FPGA every CCLK cycle.



DS001_19_032300

Figure 18: Loading Configuration Data for the Slave Parallel Mode

A configuration packet does not have to be written in one continuous stretch, rather it can be split into many write sequences. Each sequence would involve assertion of \overline{CS} .

In applications where multiple clock cycles may be required to access the configuration data before each byte can be loaded into the Slave Parallel interface, a new byte of data may not be ready for each consecutive CCLK edge. In such

a case the \overline{CS} signal may be de-asserted until the next byte is valid on D0-D7. While \overline{CS} is High, the Slave Parallel interface does not expect any data and ignores all CCLK transitions. However, to avoid aborting configuration, \overline{WRITE} must continue to be asserted while \overline{CS} is asserted.

Abort

To abort configuration during a write sequence, de-assert \overline{WRITE} while holding \overline{CS} Low. The abort operation is initiated at the rising edge of CCLK, as shown in Figure 20, page 19. The device will remain BUSY until the aborted operation is complete. After aborting configuration, data is assumed to be unaligned to word boundaries and the FPGA requires a new synchronization word prior to accepting any new packets.

Boundary-Scan Mode

In the boundary-scan mode, no nondedicated pins are required, configuration being done entirely through the IEEE 1149.1 Test Access Port.

Configuration through the TAP uses the special CFG_IN instruction. This instruction allows data input on TDI to be converted into data packets for the internal configuration bus.

The following steps are required to configure the FPGA through the boundary-scan port.

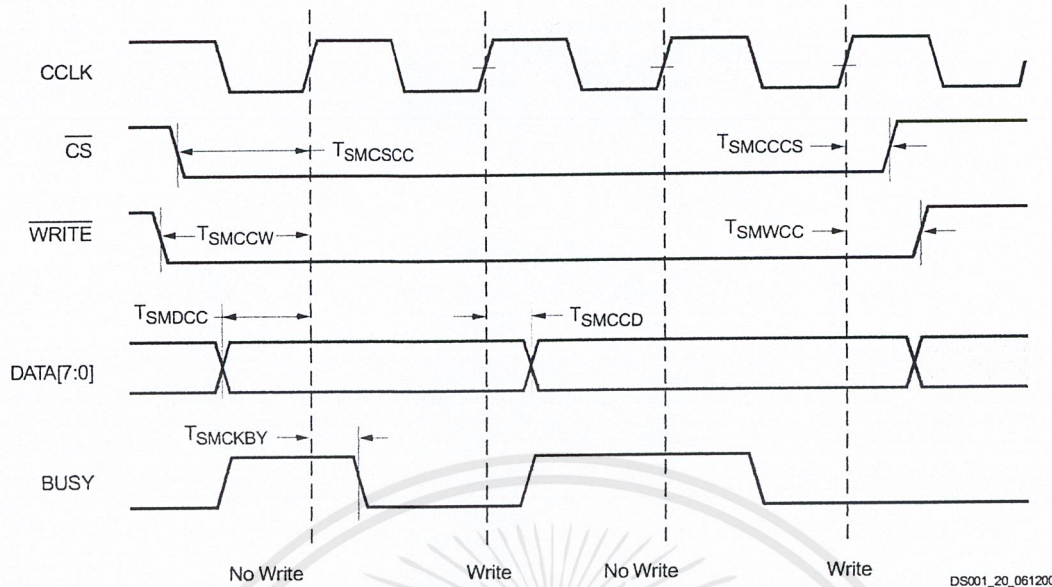
1. Load the CFG_IN instruction into the boundary-scan instruction register (IR)
2. Enter the Shift-DR (SDR) state
3. Shift a standard configuration bitstream into TDI
4. Return to Run-Test-Idle (RTI)
5. Load the JSTART instruction into IR
6. Enter the SDR state
7. Clock TCK through the sequence (the length is programmable)
8. Return to RTI

Configuration and readback via the TAP is always available. The boundary-scan mode simply locks out the other modes. The boundary-scan mode is selected by a <10x> on the mode pins (M0, M1, M2).

Readback

The configuration data stored in the Spartan-II configuration memory can be readback for verification. Along with the configuration data it is possible to readback the contents of all flip-flops/latches, LUT RAMs, and block RAMs. This capability is used for real-time debugging.

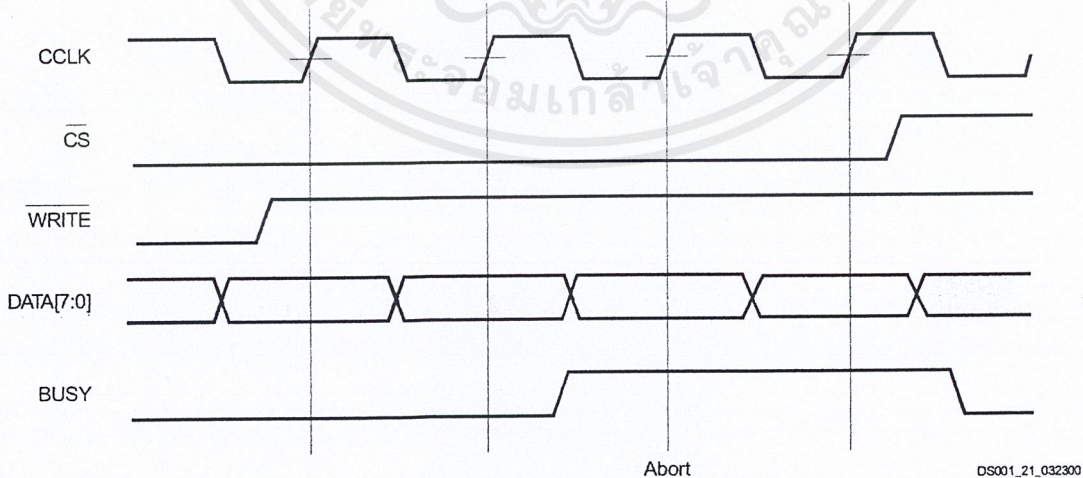
For more detailed information see XAPP176, *Spartan-II FPGA Family Configuration and Readback*.



DS001_20_061200

Symbol		Description		Units
T_{SMDC}	CCLK	D0-D7 setup/hold	5	ns, min
T_{SMCCD}		D0-D7 hold	0	ns, min
T_{SMCSCC}		\overline{CS} setup	7	ns, min
T_{SMCCCS}		\overline{CS} hold	0	ns, min
T_{SMCCW}		WRITE setup	7	ns, min
T_{SMWCC}		WRITE hold	0	ns, min
T_{SMCKBY}		BUSY propagation delay	12	ns, max
F_{CC}		Maximum frequency	66	MHz, max
F_{CCNH}		Maximum frequency with no handshake	50	MHz, max

Figure 19: Slave Parallel Write Timing



DS001_21_032300

Figure 20: Slave Parallel Write Abort Waveforms

Design Considerations

This section contains more detailed design information on the following features:

- Delay-Locked Loop . . . see page 20
- Block RAM . . . see page 24
- Versatile I/O . . . see page 29

Using Delay-Locked Loops

The Spartan-II FPGA family provides up to four fully digital dedicated on-chip Delay-Locked Loop (DLL) circuits which provide zero propagation delay, low clock skew between output clock signals distributed throughout the device, and advanced clock domain control. These dedicated DLLs can be used to implement several circuits which improve and simplify system level design.

Introduction

As FPGAs grow in size, quality on-chip clock distribution becomes increasingly important. Clock skew and clock delay impact device performance and the task of managing clock skew and clock delay with conventional clock trees becomes more difficult in large devices. The Spartan-II family of devices resolve this potential problem by providing up to four fully digital dedicated on-chip Delay-Locked Loop (DLL) circuits which provide zero propagation delay and low clock skew between output clock signals distributed throughout the device.

Each DLL can drive up to two global clock routing networks within the device. The global clock distribution network minimizes clock skews due to loading differences. By monitoring a sample of the DLL output clock, the DLL can compensate for the delay on the routing network, effectively eliminating the delay from the external input port to the individual clock loads within the device.

In addition to providing zero delay with respect to a user source clock, the DLL can provide multiple phases of the source clock. The DLL can also act as a clock doubler or it can divide the user source clock by up to 16.

Clock multiplication gives the designer a number of design alternatives. For instance, a 50 MHz source clock doubled by the DLL can drive an FPGA design operating at 100 MHz. This technique can simplify board design because the clock path on the board no longer distributes such a high-speed signal. A multiplied clock also provides designers the option of time-domain-multiplexing, using one circuit twice per clock cycle, consuming less area than two copies of the same circuit. Two DLLs in can be connected in series to increase the effective clock multiplication factor to four.

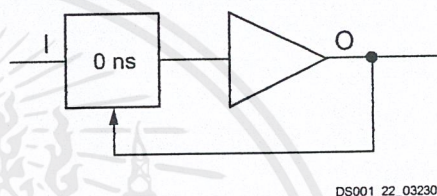
The DLL can also act as a clock mirror. By driving the DLL output off-chip and then back in again, the DLL can be used to de-skew a board level clock between multiple devices.

In order to guarantee the system clock establishes prior to the device "waking up," the DLL can delay the completion of the device configuration process until after the DLL achieves lock.

By taking advantage of the DLL to remove on-chip clock delay, the designer can greatly simplify and improve system level design involving high-fanout, high-performance clocks.

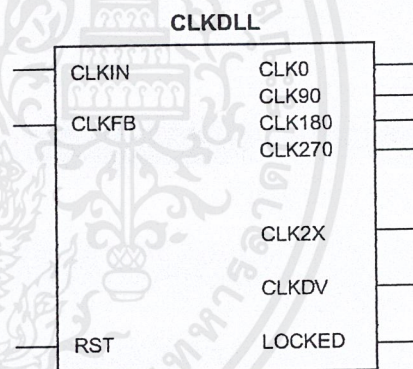
Library DLL Symbols

Figure 21 shows the simplified Xilinx library DLL macro symbol, BUFGDLL. This macro delivers a quick and efficient way to provide a system clock with zero propagation delay throughout the device. Figure 22 and Figure 23 show the two library DLL primitives. These symbols provide access to the complete set of DLL features when implementing more complex applications.



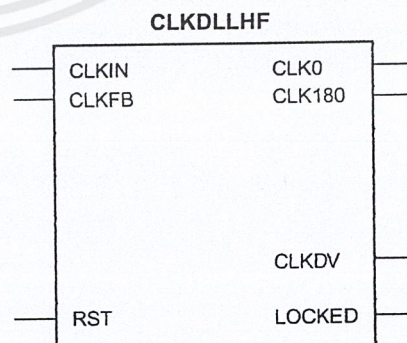
DS001_22_032300

Figure 21: Simplified DLL Macro Symbol BUFGDLL



DS001_23_032300

Figure 22: Standard DLL Symbol CLKDLL



DS001_24_032300

Figure 23: High-Frequency DLL Symbol CLKDLLHF

BUFGDLL Pin Descriptions

Use the BUFGDLL macro as the simplest way to provide zero propagation delay for a high-fanout on-chip clock from an external input. This macro uses the IBUFG, CLKDLL and BUFG primitives to implement the most basic DLL application as shown in Figure 24.

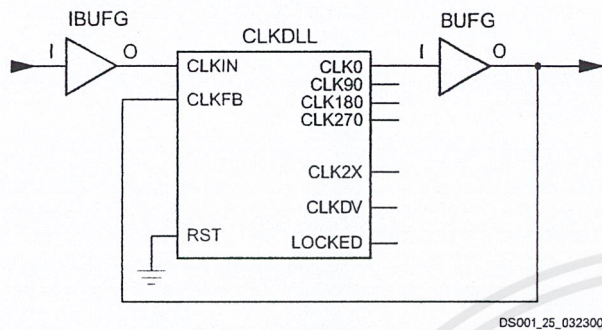


Figure 24: BUFGDLL Schematic

This symbol does not provide access to the advanced clock domain controls or to the clock multiplication or clock division features of the DLL. This symbol also does not provide access to the RST, or LOCKED pins of the DLL. For access to these features, a designer must use the library DLL primitives described in the following sections.

Source Clock Input — I

The I pin provides the user source clock, the clock signal on which the DLL operates, to the BUFGDLL. For the BUFGDLL macro the source clock frequency must fall in the low frequency range as specified in the data sheet. The BUFGDLL requires an external signal source clock. Therefore, only an external input port can source the signal that drives the BUFGDLL I pin.

Clock Output — O

The clock output pin O represents a delay-compensated version of the source clock (I) signal. This signal, sourced by a global clock buffer BUFG symbol, takes advantage of the dedicated global clock routing resources of the device.

The output clock has a 50/50 duty cycle unless you deactivate the duty cycle correction property.

CLKDLL Primitive Pin Descriptions

The library CLKDLL primitives provide access to the complete set of DLL features needed when implementing more complex applications with the DLL.

Source Clock Input — CLKIN

The CLKIN pin provides the user source clock (the clock signal on which the DLL operates) to the DLL. The CLKIN frequency must fall in the ranges specified in the data sheet. A global clock buffer (BUFG) driven from another CLKDLL

or one of the global clock input buffers (IBUFG) must source this clock signal.

Feedback Clock Input — CLKFB

The DLL requires a reference or feedback signal to provide the delay-compensated output. Connect only the CLK0 or CLK2X DLL outputs to the feedback clock input (CLKFB) pin to provide the necessary feedback to the DLL. The feedback clock input can also be provided through one of the following pin.

IBUFG - Global Clock Input Pad

If an IBUFG sources the CLKFB pin, the following special rules apply.

1. An external input port must source the signal that drives the IBUFG I pin.
2. The CLK2X output must feed back to the device if both the CLK0 and CLK2X outputs are driving off chip devices.
3. That signal must directly drive only OBUFs and nothing else.

These rules enable the software determine which DLL clock output sources the CLKFB pin.

Reset Input — RST

When the reset pin RST activates the LOCKED signal deactivates within four source clock cycles. The RST pin, active High, must either connect to a dynamic signal or tied to ground. As the DLL delay taps reset to zero, glitches can occur on the DLL clock output pins. Activation of the RST pin can also severely affect the duty cycle of the clock output pins. Furthermore, the DLL output clocks no longer deskew with respect to one another. For these reasons, rarely use the reset pin unless re-configuring the device or changing the input frequency.

2x Clock Output — CLK2X

The output pin CLK2X provides a frequency-doubled clock with an automatic 50/50 duty-cycle correction. Until the CLKDLL has achieved lock, the CLK2X output appears as a 1x version of the input clock with a 25/75 duty cycle. This behavior allows the DLL to lock on the correct edge with respect to source clock. This pin is not available on the CLKDLLHF primitive.

Clock Divide Output — CLKDV

The clock divide output pin CLKDV provides a lower frequency version of the source clock. The CLKDV_DIVIDE property controls CLKDV such that the source clock is divided by N where N is either 1.5, 2, 2.5, 3, 4, 5, 8, or 16.

This feature provides automatic duty cycle correction such that the CLKDV output pin always has a 50/50 duty cycle.

1x Clock Outputs — CLK[0|90|180|270]

The 1x clock output pin CLK0 represents a delay-compensated version of the source clock (CLKIN) signal. The

CLKDLL primitive provides three phase-shifted versions of the CLK0 signal while CLKDLLHF provides only the 180 phase-shifted version. The relationship between phase shift and the corresponding period shift appears in Table 9.

The timing diagrams in Figure 25 illustrate the DLL clock output characteristics.

Table 9: Relationship of Phase-Shifted Output Clock to Period Shift

Phase (degrees)	Period Shift (percent)
0	0%
90	25%
180	50%
270	75%

The DLL provides duty cycle correction on all 1x clock outputs such that all 1x clock outputs by default have a 50/50 duty cycle. The DUTY_CYCLE_CORRECTION property (TRUE by default), controls this feature. In order to deactivate the DLL duty cycle correction, attach the DUTY_CYCLE_CORRECTION=FALSE property to the DLL symbol. When duty cycle correction deactivates, the output clock has the same duty cycle as the source clock.

The DLL clock outputs can drive an OBUF, a BUFG, or they can route directly to destination clock pins. The DLL clock outputs can only drive the BUFGs that reside on the same edge (top or bottom).

Locked Output — LOCKED

In order to achieve lock, the DLL may need to sample several thousand clock cycles. After the DLL achieves lock the LOCKED signal activates. The DLL timing parameter section of the data sheet provides estimates for locking times.

In order to guarantee that the system clock is established prior to the device "waking up," the DLL can delay the completion of the device configuration process until after the DLL locks. The STARTUP_WAIT property activates this feature.

Until the LOCKED signal activates, the DLL output clocks are not valid and can exhibit glitches, spikes, or other spurious movement. In particular the CLK2X output will appear as a 1x clock with a 25/75 duty cycle.

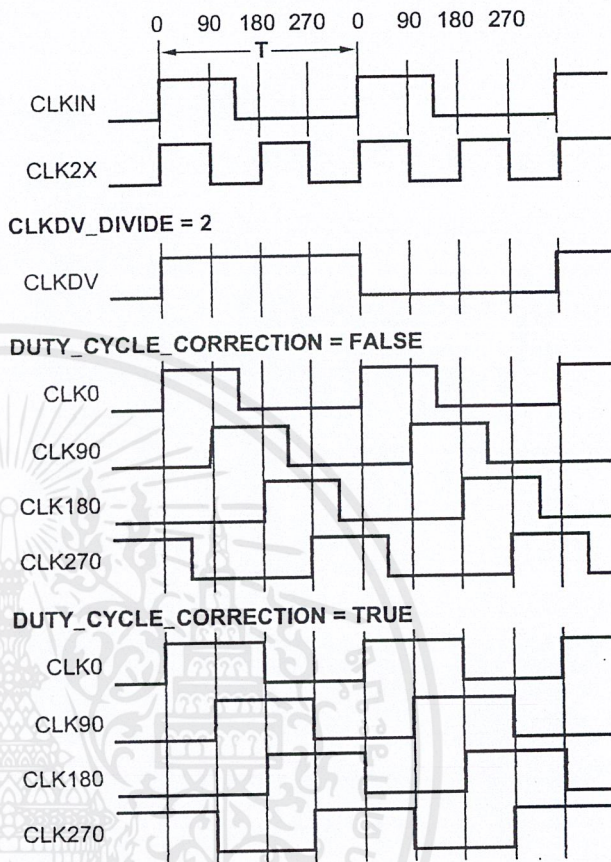
DLL Properties

Properties provide access to some of the Spartan-II family DLL features, (for example, clock division and duty cycle correction).

Duty Cycle Correction Property

The 1x clock outputs, CLK0, CLK90, CLK180, and CLK270, use the duty-cycle corrected default, exhibiting a 50/50 duty cycle. The DUTY_CYCLE_CORRECTION property (by default TRUE) controls this feature. To deactivate the DLL

duty-cycle correction for the 1x clock outputs, attach the DUTY_CYCLE_CORRECTION=FALSE property to the DLL symbol. When duty-cycle correction deactivates, the output clock has the same duty cycle as the source clock.



DS001_26_032300

Figure 25: DLL Output Characteristics

Clock Divide Property

The CLKDV_DIVIDE property specifies how the signal on the CLKDV pin is frequency divided with respect to the CLK0 pin. The values allowed for this property are 1.5, 2, 2.5, 3, 4, 5, 8, or 16; the default value is 2.

Startup Delay Property

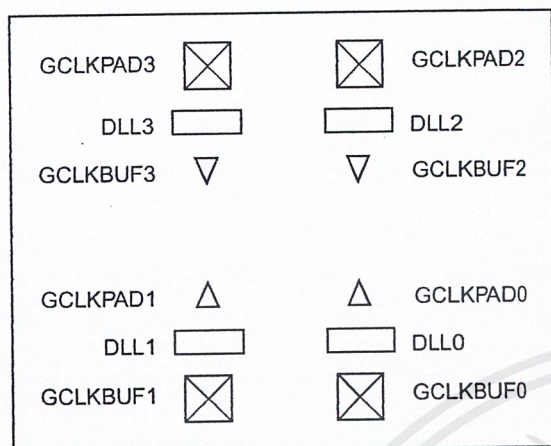
This property, STARTUP_WAIT, takes on a value of TRUE or FALSE (the default value). When TRUE the device configuration DONE signal waits until the DLL locks before going to High.

DLL Location Constraints

The DLLs are distributed such that there is one DLL in each corner of the device. The location constraint LOC, attached to the DLL symbol with the numeric identifier 0, 1, 2, or 3, controls DLL location. The orientation of the four DLLs and their corresponding clock resources appears in Figure 26.

The LOC property uses the following form.

$$LOC = DLL2$$



DS001_27_032300

Figure 26: Orientation of DLLs

Design Factors

Use the following design considerations to avoid pitfalls and improve success designing with Xilinx devices.

Input Clock

The output clock signal of a DLL, essentially a delayed version of the input clock signal, reflects any instability on the input clock in the output waveform. For this reason the quality of the DLL input clock relates directly to the quality of the output clock waveforms generated by the DLL. The DLL input clock requirements are specified in the data sheet.

In most systems a crystal oscillator generates the system clock. The DLL can be used with any commercially available quartz crystal oscillator. For example, most crystal oscillators produce an output waveform with a frequency tolerance of 100 PPM, meaning 0.01 percent change in the clock period. The DLL operates reliably on an input waveform with a frequency drift of up to 1 ns — orders of magnitude in excess of that needed to support any crystal oscillator in the industry. However, the cycle-to-cycle jitter must be kept to less than 300 ps in the low frequencies and 150 ps for the high frequencies.

Input Clock Changes

Changing the period of the input clock beyond the maximum drift amount requires a manual reset of the CLKDLL. Failure to reset the DLL will produce an unreliable lock signal and output clock.

It is possible to stop the input clock with little impact to the DLL. Stopping the clock should be limited to less than 100 μs to keep device cooling to a minimum. The clock should be stopped during a Low phase, and when restored

the full High period should be seen. During this time LOCKED will stay High and remain High when the clock is restored.

When the clock is stopped, one to four more clocks will still be observed as the delay line is flushed. When the clock is restarted, the output clocks will not be observed for one to four clocks as the delay line is filled. The most common case will be two or three clocks.

In a similar manner, a phase shift of the input clock is also possible. The phase shift will propagate to the output one to four clocks after the original shift, with no disruption to the CLKDLL control.

Output Clocks

As mentioned earlier in the DLL pin descriptions, some restrictions apply regarding the connectivity of the output pins. The DLL clock outputs can drive an OBUF, a global clock buffer BUFG, or they can route directly to destination clock pins. The only BUFGs that the DLL clock outputs can drive are the two on the same edge of the device (top or bottom).

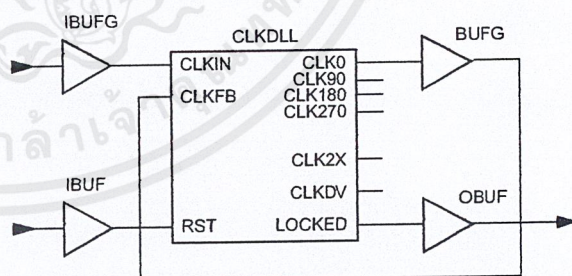
Do not use the DLL output clock signals until after activation of the LOCKED signal. Prior to the activation of the LOCKED signal, the DLL output clocks are not valid and can exhibit glitches, spikes, or other spurious movement.

Useful Application Examples

The Spartan-II DLL can be used in a variety of creative and useful applications. The following examples show some of the more common applications.

Standard Usage

The circuit shown in Figure 27 resembles the BUFGDLL macro implemented to provide access to the RST and LOCKED pins of the CLKDLL.



DS001_28_061200

Figure 27: Standard DLL Implementation

Deskew of Clock and Its 2x Multiple

The circuit shown in Figure 28 implements a 2x clock multiplier and also uses the CLK0 clock output with zero ns skew between registers on the same chip. A clock divider circuit

could alternatively be implemented using similar connections.

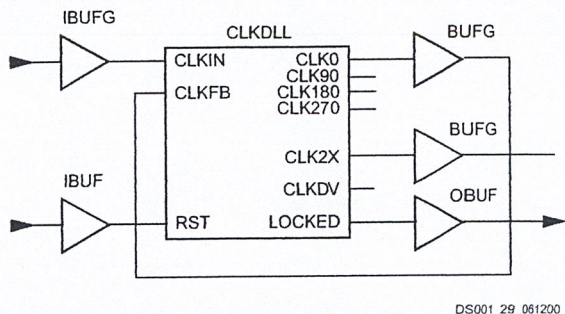


Figure 28: DLL Deskew of Clock and 2x Multiple

Because any single DLL can only access at most two BUFGs, any additional output clock signals must be routed from the DLL in this example on the high speed backbone routing.

Generating a 4x Clock

By connecting two DLL circuits each implementing a 2x clock multiplier in series as shown in Figure 29, a 4x clock multiply can be implemented with zero ns skew between registers in the same device.

If other clock output is needed, the clock could access a BUFG only if the DLLs are constrained to exist on opposite edges (Top or Bottom) of the device.

When using this circuit it is vital to use the SRL16 cell to reset the second DLL after the initial chip reset. If this is not done, the second DLL may not recognize the change of frequencies from when the input changes from a 1x (25/75) waveform to a 2x (50/50) waveform.

setup time. This is generally considered to be an inferior solution since it changes the read operation to an asynchronous function with the possibility of missing an address/control line transition during the generation of the read pulse clock.

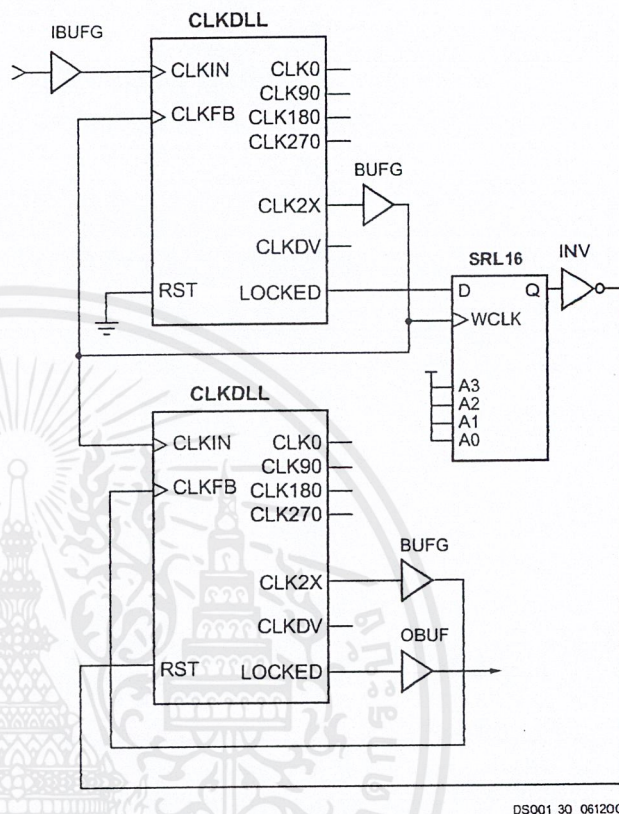


Figure 29: DLL Generation of 4x Clock

Using Block RAM Features

The Spartan-II FPGA family provides dedicated blocks of on-chip, true dual-read/write port synchronous RAM, with 4096 memory cells. Each port of the block RAM memory can be independently configured as a read/write port, a read port, a write port, and can be configured to a specific data width. The block RAM memory offers new capabilities allowing the FPGA designer to simplify designs.

Operating Modes

Block RAM memory supports two operating modes.

- Read Through
- Write Back

Read Through (One Clock Edge)

The read address is registered on the read port clock edge and data appears on the output after the RAM access time. Some memories may place the latch/register at the outputs depending on the desire to have a faster clock-to-out versus

Write Back (One Clock Edge)

The write address is registered on the write port clock edge and the data input is written to the memory and mirrored on the write port input.

Block RAM Characteristics

1. All inputs are registered with the port clock and have a setup to clock timing specification.
2. All outputs have a read through or write back function depending on the state of the port WE pin. The outputs relative to the port clock are available after the clock-to-out timing specification.
3. The block RAM are true SRAM memories and do not have a combinatorial path from the address to the output. The LUT cells in the CLBs are still available with this function.
4. The ports are completely independent from each other (*i.e.*, clocking, control, address, read/write function, and data width) without arbitration.

- 5. A write operation requires only one clock edge.
 - 6. A read operation requires only one clock edge.
- The output ports are latched with a self timed circuit to guarantee a glitch free read. The state of the output port will not change until the port executes another read or write operation.

Library Primitives

Figure 30 and Figure 31 show the two generic library block RAM primitives. Table 10 describes all of the available primitives for synthesis and simulation.

Table 10: Available Library Primitives

Primitive	Port A Width	Port B Width
RAMB4_S1	1	N/A
RAMB4_S1_S1		1
RAMB4_S1_S2		2
RAMB4_S1_S4		4
RAMB4_S1_S8		8
RAMB4_S1_S16		16
RAMB4_S2	2	N/A
RAMB4_S2_S2		2
RAMB4_S2_S4		4
RAMB4_S2_S8		8
RAMB4_S2_S16		16
RAMB4_S4	4	N/A
RAMB4_S4_S4		4
RAMB4_S4_S8		8
RAMB4_S4_S16		16
RAMB4_S8	8	N/A
RAMB4_S8_S8		8
RAMB4_S8_S16		16
RAMB4_S16	16	N/A
RAMB4_S16_S16		16

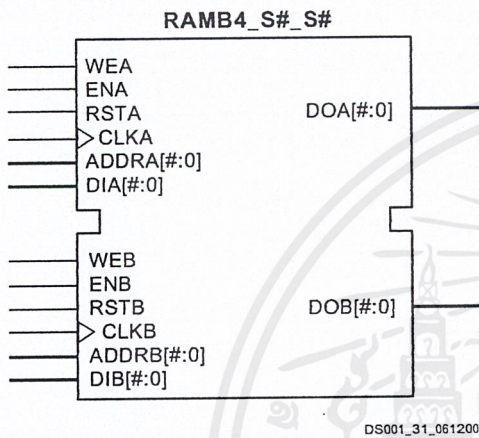


Figure 30: Dual-Port Block RAM Memory

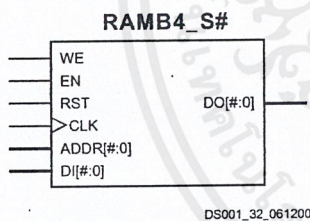


Figure 31: Single-Port Block RAM Memory

Port Signals

Each block RAM port operates independently of the others while accessing the same set of 4096 memory cells.

Table 11 describes the depth and width aspect ratios for the block RAM memory.

Table 11: Block RAM Port Aspect Ratios

Width	Depth	ADDR Bus	Data Bus
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

Clock—CLK[A|B]

Each port is fully synchronous with independent clock pins. All port input pins have setup time referenced to the port CLK pin. The data output bus has a clock-to-out time referenced to the CLK pin.

Enable—EN[A|B]

The enable pin affects the read, write and reset functionality of the port. Ports with an inactive enable pin keep the output pins in the previous state and do not write data to the memory cells.

Write Enable—WE[A|B]

Activating the write enable pin allows the port to write to the memory cells. When active, the contents of the data input bus are written to the RAM at the address pointed to by the address bus, and the new data also reflects on the data out bus. When inactive, a read operation occurs and the contents of the memory cells referenced by the address bus reflect on the data out bus.

Reset—RST[A|B]

The reset pin forces the data output bus latches to zero synchronously. This does not affect the memory cells of the RAM and does not disturb a write operation on the other port.

Address Bus—ADDR[A|B]<#:0>

The address bus selects the memory cells for read or write. The width of the port determines the required width of this bus as shown in Table 11.

Data In Bus—DI[A|B]<#:0>

The data in bus provides the new data value to be written into the RAM. This bus and the port have the same width, as shown in Table 11.

Data Output Bus—DO[A|B]<#:0>

The data out bus reflects the contents of the memory cells referenced by the address bus at the last active clock edge. During a write operation, the data out bus reflects the data in bus. The width of this bus equals the width of the port. The allowed widths appear in Table 11.

Inverting Control Pins

The four control pins (CLK, EN, WE and RST) for each port have independent inversion control as a configuration option.

Address Mapping

Each port accesses the same set of 4096 memory cells using an addressing scheme dependent on the width of the port. The physical RAM location addressed for a particular width are described in the following formula (of interest only when the two ports use different aspect ratios).

$$\text{Start} = ([\text{ADDR}_{\text{port}} + 1] * \text{Width}_{\text{port}}) - 1$$

$$\text{End} = \text{ADDR}_{\text{port}} * \text{Width}_{\text{port}}$$

Table 12 shows low order address mapping for each port width.

Table 12: Port Address Mapping

Port Width	Port Addresses																
	4095...	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	4095...	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2	2047...	07	06	05	04	03	02	01	00								
4	1023...	03		02		01		00									
8	511...	01				00											
16	255...	00															

Creating Larger RAM Structures

The block RAM columns have specialized routing to allow cascading blocks together with minimal routing delays. This achieves wider or deeper RAM structures with a smaller timing penalty than when using normal routing channels.

Location Constraints

Block RAM instances can have LOC properties attached to them to constrain the placement. The block RAM placement locations are separate from the CLB location naming convention, allowing the LOC properties to transfer easily from array to array.

The LOC properties use the following form:

$$\text{LOC} = \text{RAMB4_R\#C\#}$$

RAMB4_R0C0 is the upper left RAMB4 location on the device.

Conflict Resolution

The block RAM memory is a true dual-read/write port RAM that allows simultaneous access of the same memory cell from both ports. When one port writes to a given memory cell, the other port must not address that memory cell (for a write or a read) within the clock-to-clock setup window. The following lists specifics of port and memory cell write conflict resolution.

- If both ports write to the same memory cell simultaneously, violating the clock-to-clock setup requirement, consider the data stored as invalid.
- If one port attempts a read of the same memory cell the other simultaneously writes, violating the clock-to-clock setup requirement, the following occurs.
 - The write succeeds
 - The data out on the writing port accurately reflects the data written.
 - The data out on the reading port is invalid.

Conflicts do not cause any physical damage.

Single Port Timing

Figure 32 shows a timing diagram for a single port of a block RAM memory. The block RAM AC switching characteristics are specified in the data sheet. The block RAM memory is initially disabled.

At the first rising edge of the CLK pin, the ADDR, DI, EN, WE, and RST pins are sampled. The EN pin is High and the WE pin is Low indicating a read operation. The DO bus contains the contents of the memory location, 0x00, as indicated by the ADDR bus.

At the second rising edge of the CLK pin, the ADDR, DI, EN, WR, and RST pins are sampled again. The EN and WE pins are High indicating a write operation. The DO bus mirrors the DI bus. The DI bus is written to the memory location 0x0F.

At the third rising edge of the CLK pin, the ADDR, DI, EN, WR, and RST pins are sampled again. The EN pin is High and the WE pin is Low indicating a read operation. The DO bus contains the contents of the memory location 0x7E as indicated by the ADDR bus.

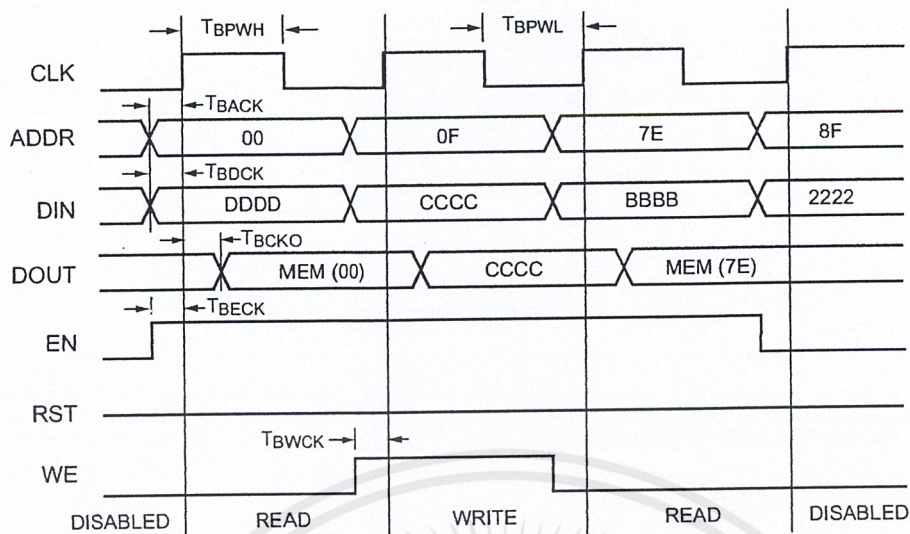
At the fourth rising edge of the CLK pin, the ADDR, DI, EN, WR, and RST pins are sampled again. The EN pin is Low indicating that the block RAM memory is now disabled. The DO bus retains the last value.

Dual Port Timing

Figure 33 shows a timing diagram for a true dual-port read/write block RAM memory. The clock on port A has a longer period than the clock on Port B. The timing parameter T_{BCCS} , (clock-to-clock setup) is shown on this diagram. The parameter, T_{BCCS} is violated once in the diagram. All other timing parameters are identical to the single port version shown in Figure 32.

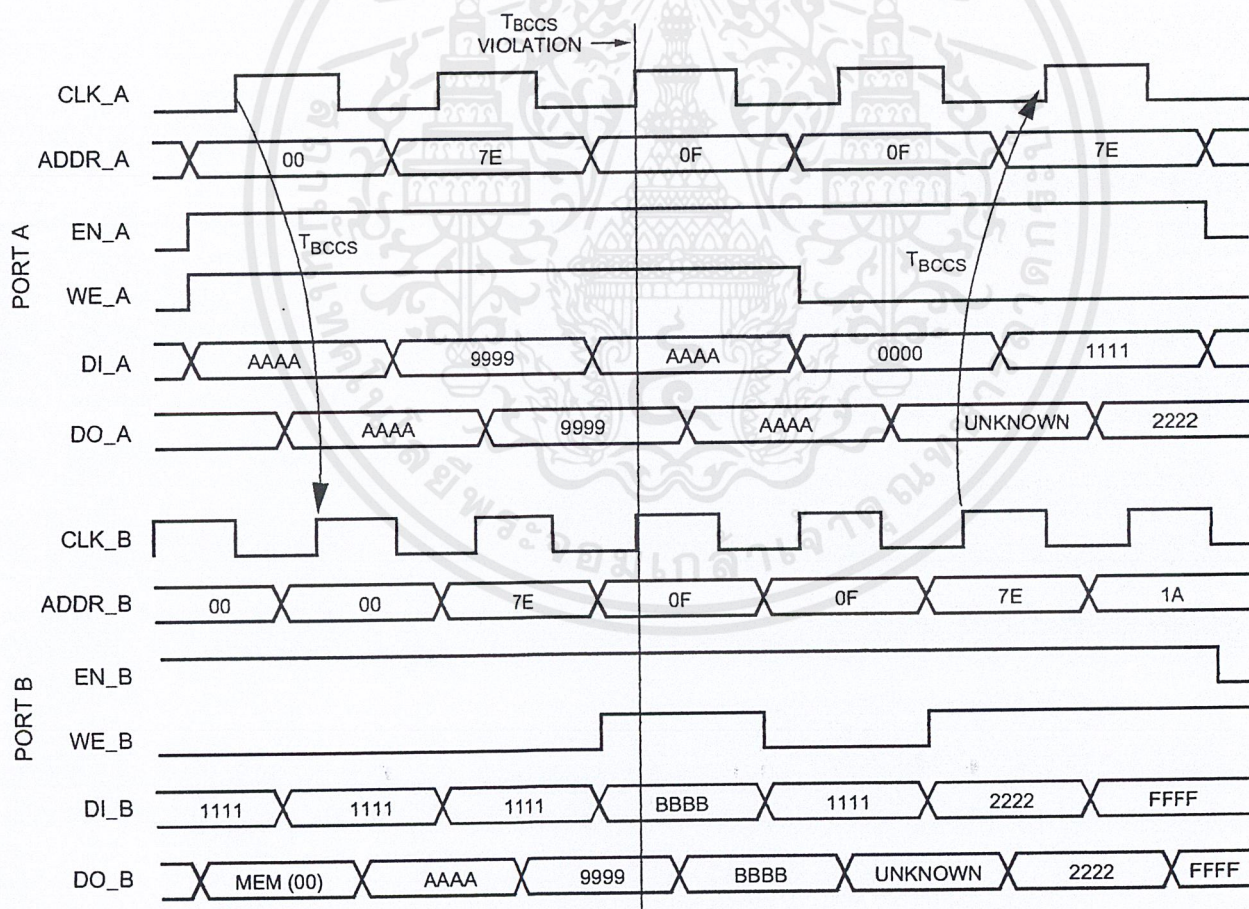
T_{BCCS} is only of importance when the address of both ports are the same and at least one port is performing a write operation. When the clock-to-clock set-up parameter is violated for a WRITE-WRITE condition, the contents of the memory at that location will be invalid. When the clock-to-clock set-up parameter is violated for a WRITE-READ condition, the contents of the memory will be correct, but the read port will have invalid data. At the first rising edge of the CLKA, memory location 0x00 is to be written with the value 0xAAAA and is mirrored on the DOA bus. The last operation of Port B was a read to the same memory location 0x00. The DOB bus of Port B does not change with the new value on Port A, and retains the last read value. A short time later, Port B executes another read to memory location 0x00, and the DOB bus now reflects the new memory value written by Port A.

At the second rising edge of CLKA, memory location 0x7E is written with the value 0x9999 and is mirrored on the DOA bus. Port B then executes a read operation to the same memory location without violating the T_{BCCS} parameter and the DOB reflects the new memory values written by Port A.



DS001_33_061200

Figure 32: Timing Diagram for Single-Port Block RAM Memory



DS001_34_061200

Figure 33: Timing Diagram for a True Dual-Port Read/Write Block RAM Memory

At the third rising edge of CLKA, the T_{BCCS} parameter is violated with two writes to memory location 0x0F. The DOA and DOB busses reflect the contents of the DIA and DIB busses, but the stored value at 0x7E is invalid.

At the fourth rising edge of CLKA, a read operation is performed at memory location 0x0F and invalid data is present on the DOA bus. Port B also executes a read operation to memory location 0x0F and also reads invalid data.

At the fifth rising edge of CLKA a read operation is performed that does not violate the T_{BCCS} parameter to the previous write of 0x7E by Port B. The DOA bus reflects the recently written value by Port B.

Initialization

The block RAM memory can initialize during the device configuration sequence. The 16 initialization properties of 64 hex values each (a total of 4096 bits) set the initialization of each RAM. These properties appear in Table 13. Any initialization properties not explicitly set configure as zeros. Partial initialization strings pad with zeros. Initialization strings greater than 64 hex values generate an error. The RAMs can be simulated with the initialization values using generics in VHDL simulators and parameters in Verilog simulators.

Initialization in VHDL and Synopsys

The block RAM structures may be initialized in VHDL for both simulation and synthesis for inclusion in the EDIF output file. The simulation of the VHDL code uses a generic to pass the initialization. Synopsys FPGA compiler does not presently support generics. The initialization values instead attach as attributes to the RAM by a built-in Synopsys `dc_script`. The `translate_off` statement stops synthesis translation of the generic statements. The following code illustrates a module that employs these techniques.

Initialization in Verilog and Synopsys

The block RAM structures may be initialized in Verilog for both simulation and synthesis for inclusion in the EDIF output file. The simulation of the Verilog code uses a `defparam` to pass the initialization. The Synopsys FPGA compiler does not presently support `defparam`. The initialization values instead attach as attributes to the RAM by a built-in Synopsys `dc_script`. The `translate_off` statement stops synthesis translation of the `defparam` statements. The following code illustrates a module that employs these techniques.

Block Memory Generation

The CoreGen program generates memory structures using the block RAM features. This program outputs VHDL or Verilog simulation code templates and an EDIF file for inclusion in a design.

Table 13: RAM Initialization Properties

Property	Memory Cells
INIT_00	255 to 0
INIT_01	511 to 256
INIT_02	767 to 512
INIT_03	1023 to 768
INIT_04	1279 to 1024
INIT_05	1535 to 1280
INIT_06	1791 to 1536
INIT_07	2047 to 1792
INIT_08	2303 to 2048
INIT_09	2559 to 2304
INIT_0a	2815 to 2560
INIT_0b	3071 to 2816
INIT_0c	3327 to 3072
INIT_0d	3583 to 3328
INIT_0e	3839 to 3584
INIT_0f	4095 to 3840

Using Versatile I/O

The Spartan-II FPGA family includes a highly configurable, high-performance I/O resource called Versatile I/O to provide support for a wide variety of I/O standards. The Versatile I/O resource is a robust set of features including programmable control of output drive strength, slew rate, and input delay and hold time. Taking advantage of the flexibility and Versatile I/O features and the design considerations described in this document can improve and simplify system level design.

Introduction

As FPGAs continue to grow in size and capacity, the larger and more complex systems designed for them demand an increased variety of I/O standards. Furthermore, as system clock speeds continue to increase, the need for high-performance I/O becomes more important. While chip-to-chip delays have an increasingly substantial impact on overall system speed, the task of achieving the desired system performance becomes more difficult with the proliferation of low-voltage I/O standards. Versatile I/O, the revolutionary input/output resources of Spartan-II devices, has resolved this potential problem by providing a highly configurable, high-performance alternative to the I/O resources of more conventional programmable devices. The Spartan-II Versatile I/O features combine the flexibility and time-to-market advantages of programmable logic with the high perfor-

mance previously available only with ASICs and custom ICs.

Each Versatile I/O block can support up to 16 I/O standards. Supporting such a variety of I/O standards allows the support of a wide variety of applications, from general purpose standard applications to high-speed low-voltage memory busses.

Versatile I/O blocks also provide selectable output drive strengths and programmable slew rates for the LVTTTL output buffers, as well as an optional, programmable weak pull-up, weak pull-down, or weak "keeper" circuit idea for use in external bussing applications.

Each Input/Output Block (IOB) includes three registers, one each for the input, output, and 3-state signals within the IOB. These registers are optionally configurable as either a D-type flip-flop or as a level sensitive latch.

The input buffer has an optional delay element used to guarantee a zero hold time requirement for input signals registered within the IOB.

The Versatile I/O features also provide dedicated resources for input reference voltage (V_{REF}) and output source voltage (V_{CCO}), along with a convenient banking system that simplifies board design.

By taking advantage of the built-in features and wide variety of I/O standards supported by the Versatile I/O features, system-level design and board design can be greatly simplified and improved.

Fundamentals

Modern bus applications, pioneered by the largest and most influential companies in the digital electronics industry, are commonly introduced with a new I/O standard tailored specifically to the needs of that application. The bus I/O standards provide specifications to other vendors who create products designed to interface with these applications. Each standard often has its own specifications for current, voltage, I/O buffering, and termination techniques.

The ability to provide the flexibility and time-to-market advantages of programmable logic is increasingly dependent on the capability of the programmable logic device to support an ever increasing variety of I/O standards

The Versatile I/O resources feature highly configurable input and output buffers which provide support for a wide variety of I/O standards. As shown in Table 14, each buffer type can support a variety of voltage requirements.

Table 14: Versatile I/O Supported Standards (Typical Values)

I/O Standard	Input Reference Voltage (V_{REF})	Output Source Voltage (V_{CCO})	Board Termination Voltage (V_{TT})
LVTTTL (2-24 mA)	N/A	3.3	N/A
LVC MOS2	N/A	2.5	N/A
PCI (3V/5V, 33 MHz/66 MHz)	N/A	3.3	N/A
GTL	0.8	N/A	1.2
GTL+	1.0	N/A	1.5
HSTL Class I	0.75	1.5	0.75
HSTL Class III	0.9	1.5	1.5
HSTL Class IV	0.9	1.5	1.5
SSTL3 Class I and II	1.5	3.3	1.5
SSTL2 Class I and II	1.25	2.5	1.25
CTT	1.5	3.3	1.5
AGP-2X	1.32	3.3	N/A

Overview of Supported I/O Standards

This section provides a brief overview of the I/O standards supported by all Spartan-II devices.

While most I/O standards specify a range of allowed voltages, this document records typical voltage values only. Detailed information on each specification may be found on the Electronic Industry Alliance Jedec website at <http://www.jedec.org>

LVTTTL — Low-Voltage TTL

The Low-Voltage TTL (LVTTTL) standard is a general purpose EIA/JESDSA standard for 3.3V applications that uses an LVTTTL input buffer and a Push-Pull output buffer. This standard requires a 3.3V output source voltage (V_{CCO}), but does not require the use of a reference voltage (V_{REF}) or a termination voltage (V_{TT}).

LVC MOS2 — Low-Voltage CMOS for 2.5V

The Low-Voltage CMOS for 2.5V or lower (LVC MOS2) standard is an extension of the LVC MOS standard (JESD 8.5) used for general purpose 2.5V applications. This standard requires a 2.5V output source voltage (V_{CCO}), but does not require the use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}).

PCI — Peripheral Component Interface

The Peripheral Component Interface (PCI) standard specifies support for both 33 MHz and 66 MHz PCI bus applications. It uses a LVTTTL input buffer and a push-pull output buffer. This standard does not require the use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}), however, it does require a 3.3V output source voltage (V_{CCO}). I/Os configured for the PCI, 33 MHz, 5V standard are also 5V-tolerant.

GTL — Gunning Transceiver Logic Terminated

The Gunning Transceiver Logic (GTL) standard is a high-speed bus standard (JESD8.3) invented by Xerox. Xilinx has implemented the terminated variation of this standard. This standard requires a differential amplifier input buffer and an open-drain output buffer.

GTL+ — Gunning Transceiver Logic Plus

The Gunning Transceiver Logic Plus (GTL+) standard is a high-speed bus standard (JESD8.3) first used by the Pentium Pro processor.

HSTL — High-Speed Transceiver Logic

The High-Speed Transceiver Logic (HSTL) standard is a general purpose high-speed, 1.5V bus standard sponsored by IBM (EIA/JESD 8-6). This standard has four variations or classes. Versatile I/O devices support Class I, III, and IV. This standard requires a Differential Amplifier input buffer and a Push-Pull output buffer.

SSTL3 — Stub Series Terminated Logic for 3.3V

The Stub Series Terminated Logic for 3.3V (SSTL3) standard is a general purpose 3.3V memory bus standard also sponsored by Hitachi and IBM (JESD8-8). This standard has two classes, I and II. Versatile I/O devices support both classes for the SSTL3 standard. This standard requires a Differential Amplifier input buffer and an Push-Pull output buffer.

SSTL2 — Stub Series Terminated Logic for 2.5V

The Stub Series Terminated Logic for 2.5V (SSTL2) standard is a general purpose 2.5V memory bus standard sponsored by Hitachi and IBM (JESD8-9). This standard has two classes, I and II. Versatile I/O devices support both classes for the SSTL2 standard. This standard requires a Differential Amplifier input buffer and an Push-Pull output buffer.

CTT — Center Tap Terminated

The Center Tap Terminated (CTT) standard is a 3.3V memory bus standard sponsored by Fujitsu (JESD8-4). This standard requires a Differential Amplifier input buffer and a Push-Pull output buffer.

AGP-2X — Advanced Graphics Port

The Intel AGP standard is a 3.3V Advanced Graphics Port-2X bus standard used with the Pentium II processor for

graphics applications. This standard requires a Push-Pull output buffer and a Differential Amplifier input buffer.

Library Symbols

The Xilinx library includes an extensive list of symbols designed to provide support for the variety of Versatile I/O features. Most of these symbols represent variations of the five generic Versatile I/O symbols:

- IBUF (input buffer)
- IBUG (global clock input buffer)
- OBUF (output buffer)
- OBUFT (3-state output buffer)
- IOBUF (input/output buffer)

IBUF

Signals used as inputs to the Spartan-II device must source an input buffer (IBUF) via an external input port. The generic IBUF symbol appears in Figure 34. The extension to the base name defines which I/O standard the IBUF uses. The assumed standard is LVTTTL when the generic IBUF has no specified extension.

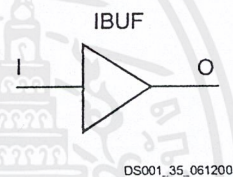


Figure 34: Input Buffer (IBUF) Symbols

The following list details the variations of the IBUF symbol:

- IBUF
- IBUF_LVCMOS2
- IBUF_PCI33_3
- IBUF_PCI33_5
- IBUF_PCI66_3
- IBUF_GTL
- IBUF_GTLP
- IBUF_HSTL_I
- IBUF_HSTL_III
- IBUF_HSTL_IV
- IBUF_SSTL3_I
- IBUF_SSTL3_II
- IBUF_SSTL2_I
- IBUF_SSTL2_II
- IBUF_CTT
- IBUF_AGP

When the IBUF symbol supports an I/O standard such as LVTTTL, LVCMOS, or PCI33_5, the IBUF automatically configures as a 5V tolerant input buffer unless the V_{CCO} for the bank is less than 2V. If the single-ended IBUF is placed in a

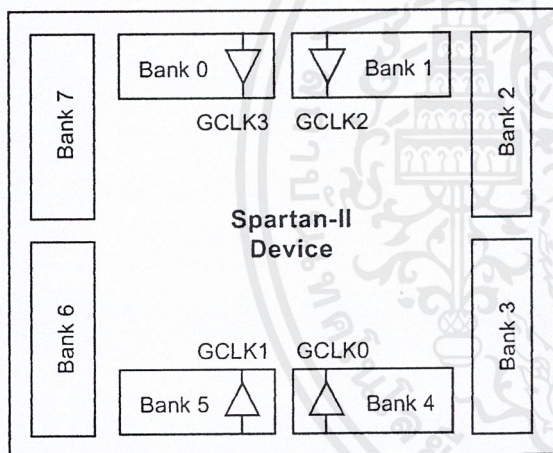
bank with an HSTL standard ($V_{CC0} < 2V$), the input buffer is not 5V tolerant.

The voltage reference signal is "banked" within the Spartan-II device on a half-edge basis such that for all packages there are eight independent V_{REF} banks internally. See Figure 35 for a representation of the I/O banks. Within each bank approximately one of every six I/O pins is automatically configured as a V_{REF} input.

IBUF placement restrictions require that any differential amplifier input signals within a bank be of the same standard. How to specify a specific location for the IBUF via the LOC property is described below. Table 15 summarizes the input standards compatibility requirements.

An optional delay element is associated with each IBUF. When the IBUF drives a flip-flop within the IOB, the delay element by default activates to ensure a zero hold-time requirement. The NODELAY=TRUE property overrides this default.

When the IBUF does not drive a flip-flop within the IOB, the delay element de-activates by default to provide higher performance. To delay the input signal, activate the delay element with the DELAY=TRUE property.



DS001_03_060100

Figure 35: I/O Banks

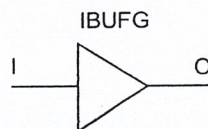
Table 15: Xilinx Input Standards Compatibility Requirements

Rule 1	All differential amplifier input signals within a bank are required to be of the same standard.
Rule 2	There are no placement restrictions for inputs with standards that require a single-ended input buffer.

IBUFG

Signals used as high fanout clock inputs to the Spartan-II device should drive a global clock input buffer (IBUFG) via an external input port in order to take advantage of one of the four dedicated global clock distribution networks.

The output of the IBUFG symbol can only drive a CLKDLL, CLKDLLHF, or a BUFG symbol. The generic IBUFG symbol appears in Figure 36.



DS001_37_061200

Figure 36: Global Clock Input Buffer (IBUFG) Symbol

The extension to the base name determines which I/O standard is used by the IBUFG. With no extension specified for the generic IBUFG symbol, the assumed standard is LVTTTL.

The following list details variations of the IBUFG symbol.

- IBUFG
- IBUFG_LVCMOS2
- IBUFG_PCI33_3
- IBUFG_PCI33_5
- IBUFG_PCI66_3
- IBUFG_GTL
- IBUFG_GTLP
- IBUFG_HSTL_I
- IBUFG_HSTL_III
- IBUFG_HSTL_IV
- IBUFG_SSTL3_I
- IBUFG_SSTL3_II
- IBUFG_SSTL2_I
- IBUFG_SSTL2_II
- IBUFG_CTT
- IBUFG_AGP

The voltage reference signal is "banked" within the Spartan-II device on a half-edge basis such that for all packages there are eight independent V_{REF} banks internally. See Figure 35 for a representation of the I/O banks. Within each bank approximately one of every six I/O pins is automatically configured as a V_{REF} input.

IBUFG placement restrictions require any differential amplifier input signals within a bank be of the same standard. The LOC property can specify a location for the IBUFG.

As an added convenience, the BUFGP can be used to instantiate a high fanout clock input. The BUFGP symbol represents a combination of the LVTTTL IBUFG and BUFG symbols, such that the output of the BUFGP can connect directly to the clock pins throughout the design.

The Spartan-II BUFGP symbol can only be placed in a global clock pad location. The LOC property can specify a location for the BUFGP.

OBUF

An OBUF must drive outputs through an external output port. The generic output buffer (OBUF) symbol appears in Figure 37.

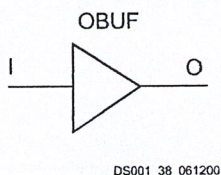


Figure 37: Output Buffer (OBUF) Symbol

The extension to the base name defines which I/O standard the OBUF uses. With no extension specified for the generic OBUF symbol, the assumed standard is slew rate limited LVTTTL with 12 mA drive strength.

The LVTTTL OBUF additionally can support one of two slew rate modes to minimize bus transients. By default, the slew rate for each output buffer is reduced to minimize power bus transients when switching non-critical signals.

LVTTTL output buffers have selectable drive strengths.

The format for LVTTTL OBUF symbol names is as follows.

OBUF_<slew_rate>_<drive_strength>

<slew_rate> is either F (Fast), or S (Slow) and <drive_strength> is specified in milliamps (2, 4, 6, 8, 12, 16, or 24).

The following list details variations of the OBUF symbol.

- OBUF
- OBUF_S_2
- OBUF_S_4
- OBUF_S_6
- OBUF_S_8
- OBUF_S_12
- OBUF_S_16
- OBUF_S_24
- OBUF_F_2
- OBUF_F_4
- OBUF_F_6
- OBUF_F_8
- OBUF_F_12
- OBUF_F_16
- OBUF_F_24
- OBUF_LVCMOS2
- OBUF_PCI33_3
- OBUF_PCI33_5
- OBUF_PCI66_3

- OBUF_GTL
- OBUF_GTLP
- OBUF_HSTL_I
- OBUF_HSTL_III
- OBUF_HSTL_IV
- OBUF_SSTL3_I
- OBUF_SSTL3_II
- OBUF_SSTL2_I
- OBUF_SSTL2_II
- OBUF_CTT
- OBUF_AGP

OBUF placement restrictions require that within a given V_{CCO} bank each OBUF share the same output source drive voltage. Input buffers of any type and output buffers that do not require V_{CCO} can be placed within any V_{CCO} bank. Table 16 summarizes the output compatibility requirements. The LOC property can specify a location for the OBUF.

Table 16: Output Standards Compatibility Requirements

Rule 1	Only outputs with standards which share compatible V _{CCO} may be used within the same bank.
Rule 2	There are no placement restrictions for outputs with standards that do not require a V _{CCO} .
V _{CCO}	Compatible Standards
3.3	LVTTTL, SSTL3_I, SSTL3_II, CTT, AGP, GTL, GTL+, PCI33_3, PCI66_3
2.5	SSTL2_I, SSTL2_II, LVCMOS2, GTL, GTL+
1.5	HSTL_I, HSTL_III, HSTL_IV, GTL, GTL+

OBUFT

The generic 3-state output buffer OBUFT, shown in Figure 38, typically implements 3-state outputs or bidirectional I/O.

The extension to the base name defines which I/O standard OBUFT uses. With no extension specified for the generic OBUFT symbol, the assumed standard is slew rate limited LVTTTL with 12 mA drive strength.

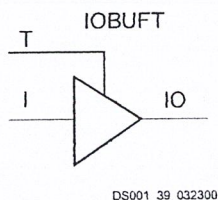
The LVTTTL OBUFT additionally can support one of two slew rate modes to minimize bus transients. By default, the slew rate for each output buffer is reduced to minimize power bus transients when switching non-critical signals.

LVTTTL 3-state output buffers have selectable drive strengths.

The format for LVTTTL OBUFT symbol names is as follows.

OBUFT_<slew_rate>_<drive_strength>

<slew_rate> can be either F (Fast), or S (Slow) and <drive_strength> is specified in milliamps (2, 4, 6, 8, 12, 16, or 24).



DS001_39_032300

Figure 38: 3-State Output Buffer Symbol (OBUFT)

The following list details variations of the OBUFT symbol.

- OBUFT
- OBUFT_S_2
- OBUFT_S_4
- OBUFT_S_6
- OBUFT_S_8
- OBUFT_S_12
- OBUFT_S_16
- OBUFT_S_24
- OBUFT_F_2
- OBUFT_F_4
- OBUFT_F_6
- OBUFT_F_8
- OBUFT_F_12
- OBUFT_F_16
- OBUFT_F_24
- OBUFT_LVCMOS2
- OBUFT_PCI33_3
- OBUFT_PCI33_5
- OBUFT_PCI66_3
- OBUFT_GTL
- OBUFT_GTLP
- OBUFT_HSTL_I
- OBUFT_HSTL_III
- OBUFT_HSTL_IV
- OBUFT_SSTL3_I
- OBUFT_SSTL3_II
- OBUFT_SSTL2_I
- OBUFT_SSTL2_II
- OBUFT_CTT
- OBUFT_AGP

The Versatile I/O OBUFT placement restrictions require that within a given V_{CC0} bank each OBUFT share the same output source drive voltage. Input buffers of any type and output buffers that do not require V_{CC0} can be placed within the same V_{CC0} bank.

The LOC property can specify a location for the OBUFT.

3-state output buffers and bidirectional buffers can have either a weak pull-up resistor, a weak pull-down resistor, or a weak "keeper" circuit. Control this feature by adding the appropriate symbol to the output net of the OBUFT (PULLUP, PULLDOWN, or KEEPER).

The weak "keeper" circuit requires the input buffer within the IOB to sample the I/O signal. So, OBUFTs programmed for an I/O standard that requires a V_{REF} have automatic placement of a V_{REF} in the bank with an OBUFT configured with a weak "keeper" circuit. This restriction does not affect most circuit design as applications using an OBUFT configured with a weak "keeper" typically implement a bidirectional I/O. In this case the IBUF (and the corresponding V_{REF}) are explicitly placed.

The LOC property can specify a location for the OBUFT.

IOBUF

Use the IOBUF symbol for bidirectional signals that require both an input buffer and a 3-state output buffer with an active high 3-state pin. The generic input/output buffer IOBUF appears in Figure 39.

The extension to the base name defines which I/O standard the IOBUF uses. With no extension specified for the generic IOBUF symbol, the assumed standard is LVTTTL input buffer and slew rate limited LVTTTL with 12 mA drive strength for the output buffer.

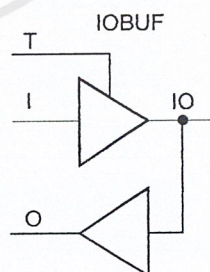
The LVTTTL IOBUF additionally can support one of two slew rate modes to minimize bus transients. By default, the slew rate for each output buffer is reduced to minimize power bus transients when switching non-critical signals.

LVTTTL bidirectional buffers have selectable output drive strengths.

The format for LVTTTL IOBUF symbol names is as follows:

IOBUF_<slew_rate>_<drive_strength>

<slew_rate> can be either F (Fast), or S (Slow) and <drive_strength> is specified in milliamps (2, 4, 6, 8, 12, 16, or 24).



DS001_40_061200

Figure 39: Input/Output Buffer Symbol (IOBUF)

The following list details variations of the IOBUF symbol:

- IOBUF
- IOBUF_S_2
- IOBUF_S_4
- IOBUF_S_6
- IOBUF_S_8
- IOBUF_S_12
- IOBUF_S_16
- IOBUF_S_24
- IOBUF_F_2
- IOBUF_F_4
- IOBUF_F_6
- IOBUF_F_8
- IOBUF_F_12
- IOBUF_F_16
- IOBUF_F_24
- IOBUF_LVCMOS2
- IOBUF_PCI33_3
- IOBUF_PCI33_5
- IOBUF_PCI66_3
- IOBUF_GTL
- IOBUF_GTLP
- IOBUF_HSTL_I
- IOBUF_HSTL_III
- IOBUF_HSTL_IV
- IOBUF_SSTL3_I
- IOBUF_SSTL3_II
- IOBUF_SSTL2_I
- IOBUF_SSTL2_II
- IOBUF_CTT
- IOBUF_AGP

When the IOBUF symbol supports an I/O standard such as LVTTTL, LVCMOS, or PCI33_5, the IBUF automatically configures as a 5V tolerant input buffer unless the V_{CC0} for the bank is less than 2V. If the single-ended IBUF is placed in a bank with an HSTL standard ($V_{CC0} < 2V$), the input buffer is not 5V tolerant.

The voltage reference signal is "banked" within the Spartan-II device on a half-edge basis such that for all packages there are eight independent V_{REF} banks internally. See Figure 35, page 32 for a representation of the Spartan-II I/O banks. Within each bank approximately one of every six I/O pins is automatically configured as a V_{REF} input.

Additional restrictions on the Versatile I/O IOBUF placement require that within a given V_{CC0} bank each IOBUF must share the same output source drive voltage. Input buffers of any type and output buffers that do not require V_{CC0} can be placed within the same V_{CC0} bank. The LOC property can specify a location for the IOBUF.

An optional delay element is associated with the input path in each IOBUF. When the IOBUF drives an input flip-flop within the IOB, the delay element activates by default to ensure a zero hold-time requirement. Override this default with the NODELAY=TRUE property.

In the case when the IOBUF does not drive an input flip-flop within the IOB, the delay element de-activates by default to provide higher performance. To delay the input signal, activate the delay element with the DELAY=TRUE property.

3-state output buffers and bidirectional buffers can have either a weak pull-up resistor, a weak pull-down resistor, or a weak "keeper" circuit. Control this feature by adding the appropriate symbol to the output net of the IOBUF (PULLUP, PULLDOWN, or KEEPER).

Versatile I/O Properties

Access to some of the Versatile I/O features (for example, location constraints, input delay, output drive strength, and slew rate) is available through properties associated with these features.

Input Delay Properties

An optional delay element is associated with each IBUF. When the IBUF drives a flip-flop within the IOB, the delay element activates by default to ensure a zero hold-time requirement. Use the NODELAY=TRUE property to override this default.

In the case when the IBUF does not drive a flip-flop within the IOB, the delay element by default de-activates to provide higher performance. To delay the input signal, activate the delay element with the DELAY=TRUE property.

IOB Flip-Flop/Latch Property

The I/O Block (IOB) includes an optional register on the input path, an optional register on the output path, and an optional register on the 3-state control pin. The design implementation software automatically takes advantage of these registers when the following option for the Map program is specified:

```
map -pr b <filename>
```

Alternatively, the IOB = TRUE property can be placed on a register to force the mapper to place the register in an IOB.

Location Constraints

Specify the location of each Versatile I/O symbol with the location constraint LOC attached to the Versatile I/O symbol. The external port identifier indicates the value of the location constrain. The format of the port identifier depends on the package chosen for the specific design.

The LOC properties use the following form:

```
LOC=A42
```

```
LOC=P37
```

Output Slew Rate Property

As mentioned above, a variety of symbol names provide the option of choosing the desired slew rate for the output buffers. In the case of the LVTTTL output buffers (OBUF, OBUFT, and IOBUF), slew rate control can be alternatively programmed with the SLEW= property. By default, the slew rate for each output buffer is reduced to minimize power bus transients when switching non-critical signals. The SLEW= property has one of the two following values.

SLEW=SLOW

SLEW=FAST

Output Drive Strength Property

The desired output drive strength can be additionally specified by choosing the appropriate library symbol. The Xilinx library also provides an alternative method for specifying this feature. For the LVTTTL output buffers (OBUF, OBUFT, and IOBUF), the desired drive strength can be specified with the DRIVE= property. This property could have one of the following seven values.

DRIVE=2

DRIVE=4

DRIVE=6

DRIVE=8

DRIVE=12 (Default)

DRIVE=16

DRIVE=24

Design Considerations

Reference Voltage (V_{REF}) Pins

Low-voltage I/O standards with a differential amplifier input buffer require an input reference voltage (V_{REF}). Provide the V_{REF} as an external signal to the device.

The voltage reference signal is "banked" within the device on a half-edge basis such that for all packages there are eight independent V_{REF} banks internally. See Figure 35, page 32 for a representation of the I/O banks. Within each bank approximately one of every six I/O pins is automatically configured as a V_{REF} input.

Within each V_{REF} bank, any input buffers that require a V_{REF} signal must be of the same type. Output buffers of any

type and input buffers can be placed without requiring a reference voltage within the same V_{REF} bank.

Output Drive Source Voltage (V_{CCO}) Pins

Many of the low voltage I/O standards supported by Versatile I/Os require a different output drive source voltage (V_{CCO}). As a result each device can often have to support multiple output drive source voltages.

The V_{CCO} supplies are internally tied together for some packages. The VQ100 and the PQ208 provide one combined V_{CCO} supply. The TQ144 and the CS144 packages provide four independent V_{CCO} supplies. The FG256 and the FG456 provide eight independent V_{CCO} supplies.

Output buffers within a given V_{CCO} bank must share the same output drive source voltage. Input buffers for LVTTTL, LVCMOS2, PCI33_3, and PCI 66_3 use the V_{CCO} voltage for Input V_{CCO} voltage.

Transmission Line Effects

The delay of an electrical signal along a wire is dominated by the rise and fall times when the signal travels a short distance. Transmission line delays vary with inductance and capacitance, but a well-designed board can experience delays of approximately 180 ps per inch.

Transmission line effects, or reflections, typically start at 1.5" for fast (1.5 ns) rise and fall times. Poor (or non-existent) termination or changes in the transmission line impedance cause these reflections and can cause additional delay in longer traces. As system speeds continue to increase, the effect of I/O delays can become a limiting factor and therefore transmission line termination becomes increasingly more important.

Termination Techniques

A variety of termination techniques reduce the impact of transmission line effects.

The following lists output termination techniques:

- None
- Series
- Parallel (Shunt)
- Series and Parallel (Series-Shunt)

Input termination techniques include the following:

- None
- Parallel (Shunt)

These termination techniques can be applied in any combination. A generic example of each combination of termination methods appears in Figure 40.

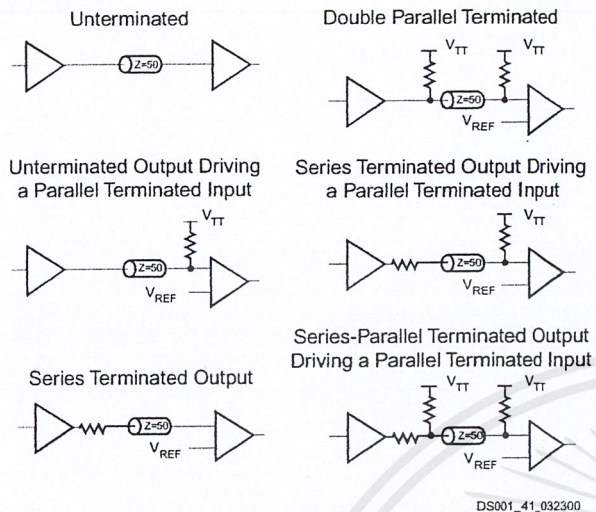


Figure 40: Overview of Standard Input and Output Termination Methods

Simultaneous Switching Guidelines

Ground bounce can occur with high-speed digital ICs when multiple outputs change states simultaneously, causing undesired transient behavior on an output, or in the internal logic. This problem is also referred to as the Simultaneous Switching Output (SSO) problem.

Ground bounce is primarily due to current changes in the combined inductance of ground pins, bond wires, and ground metallization. The IC internal ground level deviates from the external system ground level for a short duration (a few nanoseconds) after multiple outputs change state simultaneously.

Ground bounce affects stable Low outputs and all inputs because they interpret the incoming signal by comparing it to the internal ground. If the ground bounce amplitude exceeds the actual instantaneous noise margin, then a non-changing input can be interpreted as a short pulse with a polarity opposite to the ground bounce.

Table 17 provides the guidelines for the maximum number of simultaneously switching outputs allowed per output power/ground pair to avoid the effects of ground bounce. Refer to Table 16 for the number of effective output

power/ground pairs for each Spartan-II device and package combination..

Table 17: Maximum Number of Simultaneously Switching Outputs per Power/Ground Pair

Standard	Package	
	CS, FG	PQ, TQ, VQ
LVTTTL Slow Slew Rate, 2 mA drive	68	36
LVTTTL Slow Slew Rate, 4 mA drive	41	20
LVTTTL Slow Slew Rate, 6 mA drive	29	15
LVTTTL Slow Slew Rate, 8 mA drive	22	12
LVTTTL Slow Slew Rate, 12 mA drive	17	9
LVTTTL Slow Slew Rate, 16 mA drive	14	7
LVTTTL Slow Slew Rate, 24 mA drive	9	5
LVTTTL Fast Slew Rate, 2 mA drive	40	21
LVTTTL Fast Slew Rate, 4 mA drive	24	12
LVTTTL Fast Slew Rate, 6 mA drive	17	9
LVTTTL Fast Slew Rate, 8 mA drive	13	7
LVTTTL Fast Slew Rate, 12 mA drive	10	5
LVTTTL Fast Slew Rate, 16 mA drive	8	4
LVTTTL Fast Slew Rate, 24 mA drive	5	3
LVC MOS2	10	5
PCI	8	4
GTL	4	4
GTL+	4	4
HSTL Class I	18	9
HSTL Class III	9	5
HSTL Class IV	5	3
SSTL2 Class I	15	8
SSTL2 Class II	10	5
SSTL3 Class I	11	6
SSTL3 Class II	7	4
CTT	14	7
AGP	9	5

Notes:

1. This analysis assumes a 35 pF load for each output.

Table 18: Effective Output Power/Ground Pairs for Spartan-II Devices

Pkg.	Spartan-II Devices					
	XC2S 15	XC2S 30	XC2S 50	XC2S 100	XC2S 150	XC2S 200
VQ100	8	8	-	-	-	-
CS144	12	12	-	-	-	-
TQ144	12	12	12	12	-	-
PQ208	-	16	16	16	16	16
FG256	-	-	16	16	16	16
FG456	-	-	-	48	48	48

Termination Examples

Creating a design with the Versatile I/O features requires the instantiation of the desired library symbol within the design code. At the board level, designers need to know the termination techniques required for each I/O standard.

This section describes some common application examples illustrating the termination techniques recommended by each of the standards supported by the Versatile I/O features. For a full range of accepted values for the DC voltage specifications for each standard, refer to the table associated with each figure.

The resistors used in each termination technique example and the transmission lines depicted represent board level components and are not meant to represent components on the device.

GTL

A sample circuit illustrating a valid termination technique for GTL is shown in Figure 41. Table 19 lists DC voltage specifications.

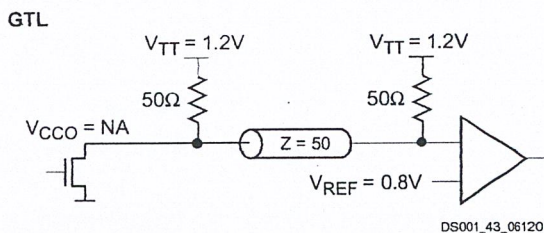


Figure 41: Terminated GTL

Table 19: GTL Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	-	N/A	-
$V_{REF} = N \times V_{TT}^{(1)}$	0.74	0.8	0.86
V_{TT}	1.14	1.2	1.26
$V_{IH} \geq V_{REF} + 0.05$	0.79	0.85	-
$V_{IL} \leq V_{REF} - 0.05$	-	0.75	0.81
V_{OH}	-	-	-
V_{OL}	-	0.2	0.4
I_{OH} at V_{OH} (mA)	-	-	-
I_{OL} at V_{OL} (mA) at 0.4V	32	-	-
I_{OL} at V_{OL} (mA) at 0.2V	-	-	40

Notes:

1. N must be greater than or equal to 0.653 and less than or equal to 0.68.

GTL+

A sample circuit illustrating a valid termination technique for GTL+ appears in Figure 42. DC voltage specifications appear in Table 20 .

HSTL Class I

A sample circuit illustrating a valid termination technique for HSTL_I appears in Figure 43. DC voltage specifications appear in Table 21.

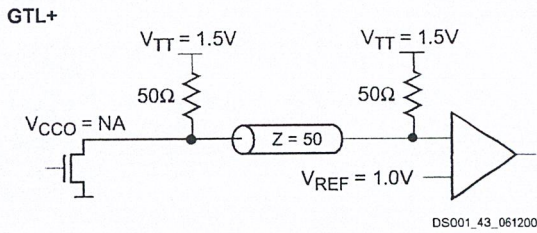


Figure 42: Terminated GTL+

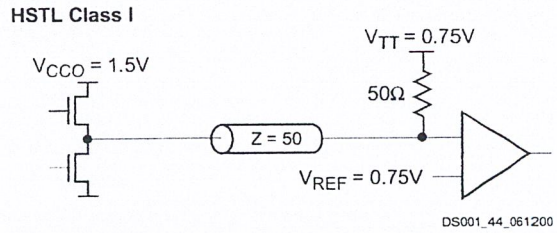


Figure 43: Terminated HSTL Class I

Table 20: GTL+ Voltage Specifications

Parameter	Min	Typ	Max
V _{CCO}	-	-	-
V _{REF} = N × V _{TT} (1)	0.88	1.0	1.12
V _{TT}	1.35	1.5	1.65
V _{IH} ≥ V _{REF} + 0.1	0.98	1.1	-
V _{IL} ≤ V _{REF} - 0.1	-	0.9	1.02
V _{OH}	-	-	-
V _{OL}	0.3	0.45	0.6
I _{OH} at V _{OH} (mA)	-	-	-
I _{OL} at V _{OL} (mA) at 0.6V	36	-	-
I _{OL} at V _{OL} (mA) at 0.3V	-	-	48

Table 21: HSTL Class I Voltage Specification

Parameter	Min	Typ	Max
V _{CCO}	1.40	1.50	1.60
V _{REF}	0.68	0.75	0.90
V _{TT}	-	V _{CCO} × 0.5	-
V _{IH}	V _{REF} + 0.1	-	-
V _{IL}	-	-	V _{REF} - 0.1
V _{OH}	V _{CCO} - 0.4	-	-
V _{OL}	-	-	0.4
I _{OH} at V _{OH} (mA)	-8	-	-
I _{OL} at V _{OL} (mA)	8	-	-

Notes:

1. N must be greater than or equal to 0.653 and less than or equal to 0.68.

HSTL Class III

A sample circuit illustrating a valid termination technique for HSTL_III appears in Figure 44. DC voltage specifications appear in Table 22.

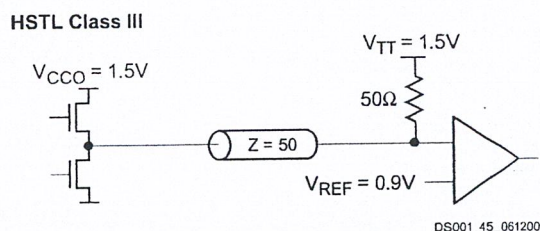


Figure 44: Terminated HSTL Class III

HSTL Class IV

A sample circuit illustrating a valid termination technique for HSTL_IV appears in Figure 45. DC voltage specifications appear in Table 23.

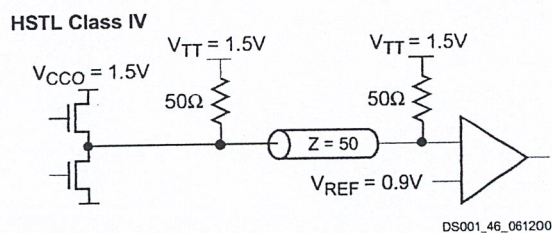


Figure 45: Terminated HSTL Class IV

Table 22: HSTL Class III Voltage Specification

Parameter	Min	Typ	Max
V_{CCO}	1.40	1.50	1.60
$V_{REF}^{(1)}$	-	0.90	-
V_{TT}	-	V_{CCO}	-
V_{IH}	$V_{REF} + 0.1$	-	-
V_{IL}	-	-	$V_{REF} - 0.1$
V_{OH}	$V_{CCO} - 0.4$	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	-8	-	-
I_{OL} at V_{OL} (mA)	24	-	-

Notes:

- Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

Table 23: HSTL Class IV Voltage Specification

Parameter	Min	Typ	Max
V_{CCO}	1.40	1.50	1.60
V_{REF}	-	0.90	-
V_{TT}	-	V_{CCO}	-
V_{IH}	$V_{REF} + 0.1$	-	-
V_{IL}	-	-	$V_{REF} - 0.1$
V_{OH}	$V_{CCO} - 0.4$	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	-8	-	-
I_{OL} at V_{OL} (mA)	48	-	-

Notes:

- Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

SSTL3 Class I

A sample circuit illustrating a valid termination technique for SSTL3_I appears in Figure 46. DC voltage specifications appear in Table 24.

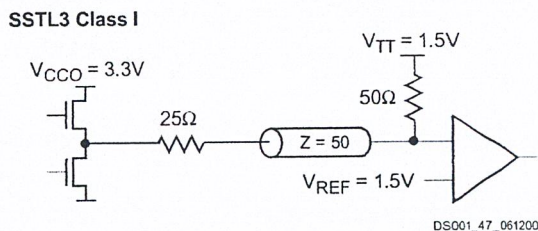


Figure 46: Terminated SSTL3 Class I

SSTL3 Class II

A sample circuit illustrating a valid termination technique for SSTL3_II appears in Figure 47. DC voltage specifications appear in Table 25.

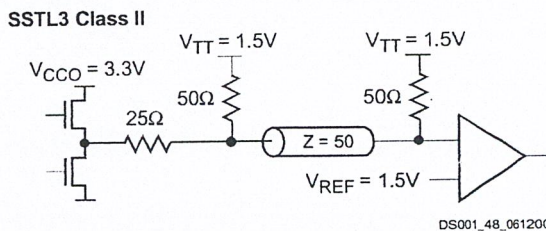


Figure 47: Terminated SSTL3 Class II

Table 24: SSTL3_I Voltage Specifications

Parameter	Min	Typ	Max
V _{CCO}	3.0	3.3	3.6
V _{REF} = 0.45 × V _{CCO}	1.3	1.5	1.7
V _{TT} = V _{REF}	1.3	1.5	1.7
V _{IH} ≥ V _{REF} + 0.2	1.5	1.7	3.9 ⁽¹⁾
V _{IL} ≤ V _{REF} - 0.2	-0.3 ⁽²⁾	1.3	1.5
V _{OH} ≥ V _{REF} + 0.6	1.9	-	-
V _{OL} ≤ V _{REF} - 0.6	-	-	1.1
I _{OH} at V _{OH} (mA)	-8	-	-
I _{OL} at V _{OL} (mA)	8	-	-

Notes:

- V_{IH} maximum is V_{CCO} + 0.3.
- V_{IL} minimum does not conform to the formula.

Table 25: SSTL3_II Voltage Specifications

Parameter	Min	Typ	Max
V _{CCO}	3.0	3.3	3.6
V _{REF} = 0.45 × V _{CCO}	1.3	1.5	1.7
V _{TT} = V _{REF}	1.3	1.5	1.7
V _{IH} ≥ V _{REF} + 0.2	1.5	1.7	3.9 ⁽¹⁾
V _{IL} ≤ V _{REF} - 0.2	-0.3 ⁽²⁾	1.3	1.5
V _{OH} ≥ V _{REF} + 0.8	2.1	-	-
V _{OL} ≤ V _{REF} - 0.8	-	-	0.9
I _{OH} at V _{OH} (mA)	-16	-	-
I _{OL} at V _{OL} (mA)	16	-	-

Notes:

- V_{IH} maximum is V_{CCO} + 0.3
- V_{IL} minimum does not conform to the formula

SSTL2_I

A sample circuit illustrating a valid termination technique for SSTL2_I appears in Figure 48. DC voltage specifications appear in Table 26.

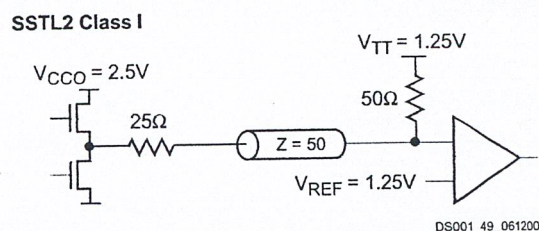


Figure 48: Terminated SSTL2 Class I

SSTL2 Class II

A sample circuit illustrating a valid termination technique for SSTL2_II appears in Figure 49. DC voltage specifications appear in Table 27.

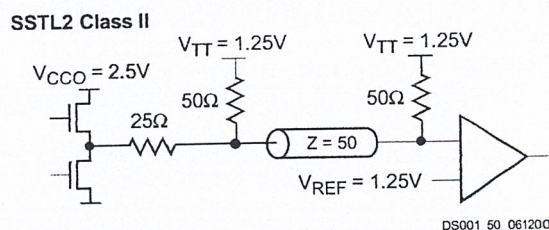


Figure 49: Terminated SSTL2 Class II

Table 26: SSTL2_I Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
$V_{REF} = 0.5 \times V_{CCO}$	1.15	1.25	1.35
$V_{TT} = V_{REF} + N^{(1)}$	1.11	1.25	1.39
$V_{IH} \geq V_{REF} + 0.18$	1.33	1.43	3.0 ⁽²⁾
$V_{IL} \leq V_{REF} - 0.18$	-0.3 ⁽³⁾	1.07	1.17
$V_{OH} \geq V_{REF} + 0.61$	1.76	-	-
$V_{OL} \leq V_{REF} - 0.61$	-	-	0.74
I_{OH} at V_{OH} (mA)	-7.6	-	-
I_{OL} at V_{OL} (mA)	7.6	-	-

Notes:

1. N must be greater than or equal to -0.04 and less than or equal to 0.04.
2. V_{IH} maximum is $V_{CCO} + 0.3$.
3. V_{IL} minimum does not conform to the formula.

Table 27: SSTL2_II Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
$V_{REF} = 0.5 \times V_{CCO}$	1.15	1.25	1.35
$V_{TT} = V_{REF} + N^{(1)}$	1.11	1.25	1.39
$V_{IH} \geq V_{REF} + 0.18$	1.33	1.43	3.0 ⁽²⁾
$V_{IL} \leq V_{REF} - 0.18$	-0.3 ⁽³⁾	1.07	1.17
$V_{OH} \geq V_{REF} + 0.8$	1.95	-	-
$V_{OL} \leq V_{REF} - 0.8$	-	-	0.55
I_{OH} at V_{OH} (mA)	-15.2	-	-
I_{OL} at V_{OL} (mA)	15.2	-	-

Notes:

1. N must be greater than or equal to -0.04 and less than or equal to 0.04.
2. V_{IH} maximum is $V_{CCO} + 0.3$.
3. V_{IL} minimum does not conform to the formula.

CTT

A sample circuit illustrating a valid termination technique for CTT appear in Figure 50. DC voltage specifications appear in Table 28.

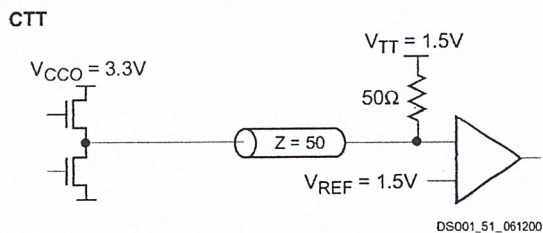


Figure 50: Terminated CTT

Table 28: CTT Voltage Specifications

Parameter	Min	Typ	Max
V _{CCO}	2.05 ⁽¹⁾	3.3	3.6
V _{REF}	1.35	1.5	1.65
V _{TT}	1.35	1.5	1.65
V _{IH} ≥ V _{REF} + 0.2	1.55	1.7	-
V _{IL} ≤ V _{REF} - 0.2	-	1.3	1.45
V _{OH} ≥ V _{REF} + 0.4	1.75	1.9	-
V _{OL} ≤ V _{REF} - 0.4	-	1.1	1.25
I _{OH} at V _{OH} (mA)	-8	-	-
I _{OL} at V _{OL} (mA)	8	-	-

Notes:
1. Timing delays are calculated based on V_{CCO} min of 3.0V.

PCI33_3 and PCI66_3

PCI33_3 or PCI66_3 require no termination. DC voltage specifications appear in Table 29.

Table 29: PCI33_3 and PCI66_3 Voltage Specifications

Parameter	Min	Typ	Max
V _{CCO}	3.0	3.3	3.6
V _{REF}	-	-	-
V _{TT}	-	-	-
V _{IH} = 0.5 × V _{CCO}	1.5	1.65	V _{CCO} + 0.5
V _{IL} = 0.3 × V _{CCO}	-0.5	0.99	1.08
V _{OH} = 0.9 × V _{CCO}	2.7	-	-
V _{OL} = 0.1 × V _{CCO}	-	-	0.36
I _{OH} at V _{OH} (mA)	Note 1	-	-
I _{OL} at V _{OL} (mA)	Note 1	-	-

Notes:
1. Tested according to the relevant specification.

PCI33_5

PCI33_5 requires no termination. DC voltage specifications appear in Table 30.

Table 30: PCI33_5 Voltage Specifications

Parameter	Min	Typ	Max
V _{CCO}	3.0	3.3	3.6
V _{REF}	-	-	-
V _{TT}	-	-	-
V _{IH}	1.425	1.5	5.5
V _{IL}	-0.5	1.0	1.05
V _{OH}	2.4	-	-
V _{OL}	-	-	0.55
I _{OH} at V _{OH} (mA)	Note 1	-	-
I _{OL} at V _{OL} (mA)	Note 1	-	-

Notes:
1. Tested according to the relevant specification.

LVTTL

LVTTL requires no termination. DC voltage specifications appears in Table 31.

Table 31: LVTTL Voltage Specifications

Parameter	Min	Typ	Max
V _{CCO}	3.0	3.3	3.6
V _{REF}	-	-	-
V _{TT}	-	-	-
V _{IH}	2.0	-	3.6
V _{IL}	-0.5	-	0.8
V _{OH}	2.4	-	-
V _{OL}	-	-	0.4
I _{OH} at V _{OH} (mA)	-24	-	-
I _{OL} at V _{OL} (mA)	24	-	-

Notes:

1. V_{OL} and V_{OH} for lower drive currents sample tested.

LVC MOS2

LVC MOS2 requires no termination. DC voltage specifications appear in Table 32.

Table 32: LVC MOS2 Voltage Specifications

Parameter	Min	Typ	Max
V _{CCO}	2.3	2.5	2.7
V _{REF}	-	-	-
V _{TT}	-	-	-
V _{IH}	1.7	-	3.6
V _{IL}	-0.5	-	0.7
V _{OH}	1.9	-	-
V _{OL}	-	-	0.4
I _{OH} at V _{OH} (mA)	-12	-	-
I _{OL} at V _{OL} (mA)	12	-	-

AGP-2X

The specification for the AGP-2X standard does not document a recommended termination technique. DC voltage specifications appear in Table 33.

Table 33: AGP-2X Voltage Specifications

Parameter	Min	Typ	Max
V _{CCO}	3.0	3.3	3.6
V _{REF} = N × V _{CCO} ⁽¹⁾	1.17	1.32	1.48
V _{TT}	-	-	-
V _{IH} ≥ V _{REF} + 0.2	1.37	1.52	-
V _{IL} ≤ V _{REF} - 0.2	-	1.12	1.28
V _{OH} ≥ 0.9 × V _{CCO}	2.7	3.0	-
V _{OL} ≤ 0.1 × V _{CCO}	-	0.33	0.36
I _{OH} at V _{OH} (mA)	Note 2	-	-
I _{OL} at V _{OL} (mA)	Note 2	-	-

Notes:

1. N must be greater than or equal to 0.39 and less than or equal to 0.41.
2. Tested according to the relevant specification.

Revision History

Version No.	Date	Description
2.0	09/18/00	Sectioned the Spartan-II Family data sheet into four modules. Corrected banking description.
2.1	03/05/01	Clarified guidelines for applying power to V_{CCINT} and V_{CCO}

The Spartan-II Family Data Sheet

DS001-1, *Spartan-II 2.5V FPGA Family: Introduction and Ordering Information* (Module 1)

DS001-2, *Spartan-II 2.5V FPGA Family: Functional Description* (Module 2)

DS001-3, *Spartan-II 2.5V FPGA Family: DC and Switching Characteristics* (Module 3)

DS001-4, *Spartan-II 2.5V FPGA Family: Pinout Tables* (Module 4)





Spartan-II 2.5V FPGA Family: Pinout Tables

DS001-4 (v2.4) April 30, 2001

Preliminary Product Specification

Pin Definitions

Pin Name	Dedicated Pin	Direction	Description
GCK0, GCK1, GCK2, GCK3	No	Input	Clock input pins that connect to Global Clock Buffers. These pins become user inputs when not needed for clocks.
M0, M1, M2	Yes	Input	Mode pins are used to specify the configuration mode.
CCLK	Yes	Input or Output	The configuration Clock I/O pin. It is an input for slave-parallel and slave-serial modes, and output in master-serial mode.
PROGRAM	Yes	Input	Initiates a configuration sequence when asserted Low.
DONE	Yes	Bidirectional	Indicates that configuration loading is complete, and that the start-up sequence is in progress. The output may be open drain.
INIT	No	Bidirectional (Open-drain)	When Low, indicates that the configuration memory is being cleared. This pin becomes a user I/O after configuration.
BUSY/DOUT	No	Output	In Slave Parallel mode, BUSY controls the rate at which configuration data is loaded. This pin becomes a user I/O after configuration unless the Slave Parallel port is retained. In serial modes, DOUT provides configuration data to downstream devices in a daisy-chain. This pin becomes a user I/O after configuration.
D0/DIN, D1, D2, D3, D4, D5, D6, D7	No	Input or Output	In Slave Parallel mode, D0-D7 are configuration data input pins. During readback, D0-D7 are output pins. These pins become user I/Os after configuration unless the Slave Parallel port is retained. In serial modes, DIN is the single data input. This pin becomes a user I/O after configuration.
WRITE	No	Input	In Slave Parallel mode, the active-low Write Enable signal. This pin becomes a user I/O after configuration unless the Slave Parallel port is retained.
CS	No	Input	In Slave Parallel mode, the active-low Chip Select signal. This pin becomes a user I/O after configuration unless the Slave Parallel port is retained.
TDI, TDO, TMS, TCK	Yes	Mixed	Boundary Scan Test Access Port pins (IEEE 1149.1).
V _{CCINT}	Yes	Input	Power supply pins for the internal core logic.
V _{CCO}	Yes	Input	Power supply pins for output drivers (subject to banking rules)
V _{REF}	No	Input	Input threshold voltage pins. Become user I/Os when an external threshold voltage is not needed (subject to banking rules).
GND	Yes	Input	Ground.
IRDY, TRDY	No	See PCI core documentation	These signals can only be accessed when using Xilinx PCI cores. If the cores are not used, these pins are available as user I/Os.

© 2000 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

XC2S200 Device Pinouts

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
GND	-	P1	GND*	GND*	-
TMS	-	P2	D3	D3	-
I/O	7	P3	C2	B1	257
I/O	7	-	-	E4	263
I/O	7	-	-	C1	266
I/O	7	-	A2	F5	269
GND	-	-	GND*	GND*	-
I/O, V _{REF}	7	P4	B1	D2	272
I/O	7	-	-	E3	275
I/O	7	-	-	F4	281
GND	-	-	GND*	GND*	-
I/O	7	-	E3	G5	284
I/O	7	P5	D2	F3	287
GND	-	-	GND*	GND*	-
V _{CCO}	7	-	V _{CCO} Bank 7*	V _{CCO} Bank 7*	-
I/O, V _{REF}	7	P6	C1	E2	290
I/O	7	P7	F3	E1	293
I/O	7	-	-	G4	296
I/O	7	-	-	G3	299
I/O	7	-	E2	H5	302
GND	-	-	GND*	GND*	-
I/O	7	P8	E4	F2	305
I/O	7	-	-	F1	308
I/O, V _{REF}	7	P9	D1	H4	314
I/O	7	P10	E1	G1	317
GND	-	P11	GND*	GND*	-
V _{CCO}	7	P12	V _{CCO} Bank 7*	V _{CCO} Bank 7*	-
V _{CCINT}	-	P13	V _{CCINT} *	V _{CCINT} *	-
I/O	7	P14	F2	H3	320
I/O	7	P15	G3	H2	323
I/O	7	-	-	J4	326
I/O	7	-	-	H1	329
I/O	7	-	F1	J5	332
GND	-	-	GND*	GND*	-
I/O	7	P16	F4	J2	335
I/O	7	-	-	J3	338
I/O	7	-	-	J1	341
I/O	7	P17	F5	K5	344
I/O	7	P18	G2	K1	347
GND	-	P19	GND*	GND*	-

XC2S200 Device Pinouts (Continued)

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
V _{CCO}	7	-	V _{CCO} Bank 7*	V _{CCO} Bank 7*	-
I/O, V _{REF}	7	P20	H3	K3	350
I/O	7	P21	G4	K4	353
I/O	7	-	-	K2	359
I/O	7	-	H2	L6	362
I/O	7	P22	G5	L1	365
I/O	7	-	-	L5	368
I/O	7	P23	H4	L4	374
I/O, IRDY ⁽¹⁾	7	P24	G1	L3	377
GND	-	P25	GND*	GND*	-
V _{CCO}	7	P26	V _{CCO} Bank 7*	V _{CCO} Bank 7*	-
V _{CCO}	6	P26	V _{CCO} Bank 6*	V _{CCO} Bank 6*	-
I/O, TRDY ⁽¹⁾	6	P27	J2	M1	380
V _{CCINT}	-	P28	V _{CCINT} *	V _{CCINT} *	-
I/O	6	-	-	M6	389
I/O	6	P29	H1	M3	392
I/O	6	-	J4	M4	395
I/O	6	-	-	N1	398
I/O	6	P30	J1	M5	404
I/O, V _{REF}	6	P31	J3	N2	407
V _{CCO}	6	-	V _{CCO} Bank 6*	V _{CCO} Bank 6*	-
GND	-	P32	GND*	GND*	-
I/O	6	P33	K5	N3	410
I/O	6	P34	K2	N4	413
I/O	6	-	-	P1	416
I/O	6	-	-	N5	419
I/O	6	P35	K1	P2	422
GND	-	-	GND*	GND*	-
I/O	6	-	K3	P4	425
I/O	6	-	-	R1	428
I/O	6	-	-	P5	431
I/O	6	P36	L1	P3	434
I/O	6	P37	L2	R2	437
V _{CCINT}	-	P38	V _{CCINT} *	V _{CCINT} *	-
V _{CCO}	6	P39	V _{CCO} Bank 6*	V _{CCO} Bank 6*	-
GND	-	P40	GND*	GND*	-
I/O	6	P41	K4	T1	440
I/O, V _{REF}	6	P42	M1	R4	443

XC2S200 Device Pinouts (Continued)

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
I/O	6	-	-	T2	449
I/O	6	P43	L4	U1	452
GND	-	-	GND*	GND*	-
I/O	6	-	M2	R5	455
I/O	6	-	-	V1	458
I/O	6	-	-	T5	461
I/O	6	P44	L3	U2	464
I/O, V _{REF}	6	P45	N1	T3	467
V _{CCO}	6	-	V _{CCO} Bank 6*	V _{CCO} Bank 6*	-
GND	-	-	GND*	GND*	-
I/O	6	P46	P1	T4	470
I/O	6	-	L5	W1	473
GND	-	-	GND*	GND*	-
I/O	6	-	-	V2	476
I/O	6	-	-	U4	482
I/O, V _{REF}	6	P47	N2	Y1	485
GND	-	-	GND*	GND*	-
I/O	6	-	M4	W2	488
I/O	6	-	-	V3	491
I/O	6	-	-	V4	494
I/O	6	P48	R1	Y2	500
I/O	6	P49	M3	W3	503
M1	-	P50	P2	U5	506
GND	-	P51	GND*	GND*	-
M0	-	P52	N3	AB2	507
V _{CCO}	6	P53	V _{CCO} Bank 6*	V _{CCO} Bank 6*	-
V _{CCO}	5	P53	V _{CCO} Bank 5*	V _{CCO} Bank 5*	-
M2	-	P54	R3	Y4	508
I/O	5	-	-	W5	518
I/O	5	-	-	AB3	521
I/O	5	-	N5	V7	524
GND	-	-	GND*	GND*	-
I/O, V _{REF}	5	P57	T2	Y6	527
I/O	5	-	-	AA4	530
I/O	5	-	-	AB4	536
I/O	5	-	P5	W6	539
I/O	5	P58	T3	Y7	542
GND	-	-	GND*	GND*	-

XC2S200 Device Pinouts (Continued)

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
V _{CCO}	5	-	V _{CCO} Bank 5*	V _{CCO} Bank 5*	-
I/O, V _{REF}	5	P59	T4	AA5	545
I/O	5	P60	M6	AB5	548
I/O	5	-	-	V8	551
I/O	5	-	-	AA6	554
I/O	5	-	T5	AB6	557
GND	-	-	GND*	GND*	-
I/O	5	P61	N6	AA7	560
I/O	5	-	-	W7	563
I/O, V _{REF}	5	P62	R5	W8	569
I/O	5	P63	P6	Y8	572
GND	-	P64	GND*	GND*	-
V _{CCO}	5	P65	V _{CCO} Bank 5*	V _{CCO} Bank 5*	-
V _{CCINT}	-	P66	V _{CCINT} *	V _{CCINT} *	-
I/O	5	P67	R6	AA8	575
I/O	5	P68	M7	V9	578
I/O	5	-	-	AB8	581
I/O	5	-	-	W9	584
I/O	5	-	-	AB9	587
GND	-	-	GND*	GND*	-
I/O	5	P69	N7	Y9	590
I/O	5	-	-	V10	593
I/O	5	-	-	AA9	596
I/O	5	P70	T6	W10	599
I/O	5	P71	P7	AB10	602
GND	-	P72	GND*	GND*	-
V _{CCO}	5	-	V _{CCO} Bank 5*	V _{CCO} Bank 5*	-
I/O, V _{REF}	5	P73	P8	Y10	605
I/O	5	P74	R7	V11	608
I/O	5	-	-	AA10	614
I/O	5	-	T7	W11	617
I/O	5	P75	T8	AB11	620
I/O	5	-	-	U11	623
V _{CCINT}	-	P76	V _{CCINT} *	V _{CCINT} *	-
I, GCK1	5	P77	R8	Y11	635
V _{CCO}	5	P78	V _{CCO} Bank 5*	V _{CCO} Bank 5*	-
V _{CCO}	4	P78	V _{CCO} Bank 4*	V _{CCO} Bank 4*	-
GND	-	P79	GND*	GND*	-

XC2S200 Device Pinouts (Continued)

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
I, GCK0	4	P80	N8	W12	636
I/O	4	P81	N9	U12	640
I/O	4	-	-	V12	646
I/O	4	P82	R9	Y12	649
I/O	4	-	N10	AA12	652
I/O	4	-	-	W13	655
I/O	4	P83	T9	AB13	661
I/O, V _{REF}	4	P84	P9	AA13	664
V _{CCO}	4	-	V _{CCO} Bank 4*	V _{CCO} Bank 4*	-
GND	-	P85	GND*	GND*	-
I/O	4	P86	M10	Y13	667
I/O	4	P87	R10	V13	670
I/O	4	-	-	AB14	673
I/O	4	-	-	W14	676
I/O	4	P88	P10	AA14	679
GND	-	-	GND*	GND*	-
I/O	4	-	-	V14	682
I/O	4	-	-	Y14	685
I/O	4	-	-	W15	688
I/O	4	P89	T10	AB15	691
I/O	4	P90	R11	AA15	694
V _{CCINT}	-	P91	V _{CCINT} *	V _{CCINT} *	-
V _{CCO}	4	P92	V _{CCO} Bank 4*	V _{CCO} Bank 4*	-
GND	-	P93	GND*	GND*	-
I/O	4	P94	M11	Y15	697
I/O, V _{REF}	4	P95	T11	AB16	700
I/O	4	-	-	AB17	706
I/O	4	P96	N11	V15	709
GND	-	-	GND*	GND*	-
I/O	4	-	R12	Y16	712
I/O	4	-	-	AA17	715
I/O	4	-	-	W16	718
I/O	4	P97	P11	AB18	721
I/O, V _{REF}	4	P98	T12	AB19	724
V _{CCO}	4	-	V _{CCO} Bank 4*	V _{CCO} Bank 4*	-
GND	-	-	GND*	GND*	-
I/O	4	P99	T13	Y17	727
I/O	4	-	N12	V16	730
I/O	4	-	-	AA18	733

XC2S200 Device Pinouts (Continued)

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
I/O	4	-	-	W17	739
I/O, V _{REF}	4	P100	R13	AB20	742
GND	-	-	GND*	GND*	-
I/O	4	-	P12	AA19	745
I/O	4	-	-	V17	748
I/O	4	-	-	Y18	751
I/O	4	P101	P13	AA20	757
I/O	4	P102	T14	W18	760
GND	-	P103	GND*	GND*	-
DONE	3	P104	R14	Y19	763
V _{CCO}	4	P105	V _{CCO} Bank 4*	V _{CCO} Bank 4*	-
V _{CCO}	3	P105	V _{CCO} Bank 3*	V _{CCO} Bank 3*	-
PROGRAM	-	P106	P15	W20	766
I/O (INIT)	3	P107	N15	V19	767
I/O (D7)	3	P108	N14	Y21	770
I/O	3	-	-	V20	776
I/O	3	-	-	AA22	779
I/O	3	-	T15	W21	782
GND	-	-	GND*	GND*	-
I/O, V _{REF}	3	P109	M13	U20	785
I/O	3	-	-	U19	788
I/O	3	-	-	V21	794
GND	-	-	GND*	GND*	-
I/O	3	-	R16	T18	797
I/O	3	P110	M14	W22	800
GND	-	-	GND*	GND*	-
V _{CCO}	3	-	V _{CCO} Bank 3*	V _{CCO} Bank 3*	-
I/O, V _{REF}	3	P111	L14	U21	803
I/O	3	P112	M15	T20	806
I/O	3	-	-	T19	809
I/O	3	-	-	V22	812
I/O	3	-	L12	T21	815
GND	-	-	GND*	GND*	-
I/O	3	P113	P16	R18	818
I/O	3	-	-	U22	821
I/O, V _{REF}	3	P114	L13	R19	827
I/O (D6)	3	P115	N16	T22	830
GND	-	P116	GND*	GND*	-

XC2S200 Device Pinouts (Continued)

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
V _{CCO}	3	P117	V _{CCO} Bank 3*	V _{CCO} Bank 3*	-
V _{CCINT}	-	P118	V _{CCINT} *	V _{CCINT} *	-
I/O (D5)	3	P119	M16	R21	833
I/O	3	P120	K14	P18	836
I/O	3	-	-	R22	839
I/O	3	-	-	P19	842
I/O	3	-	L16	P20	845
GND	-	-	GND*	GND*	-
I/O	3	P121	K13	P21	848
I/O	3	-	-	N19	851
I/O	3	-	-	P22	854
I/O	3	P122	L15	N18	857
I/O	3	P123	K12	N20	860
GND	-	P124	GND*	GND*	-
V _{CCO}	3	-	V _{CCO} Bank 3*	V _{CCO} Bank 3*	-
I/O, V _{REF}	3	P125	K16	N21	863
I/O (D4)	3	P126	J16	N22	866
I/O	3	-	-	M17	872
I/O	3	-	J14	M19	875
I/O	3	P127	K15	M20	878
I/O	3	-	-	M18	881
V _{CCINT}	-	P128	V _{CCINT} *	V _{CCINT} *	-
I/O, TRDY ⁽¹⁾	3	P129	J15	M22	890
V _{CCO}	3	P130	V _{CCO} Bank 3*	V _{CCO} Bank 3*	-
V _{CCO}	2	P130	V _{CCO} Bank 2*	V _{CCO} Bank 2*	-
GND	-	P131	GND*	GND*	-
I/O, IRDY ⁽¹⁾	2	P132	H16	L20	893
I/O	2	P133	H14	L17	896
I/O	2	-	-	L18	902
I/O	2	P134	H15	L21	905
I/O	2	-	J13	L22	908
I/O	2	-	-	K19	911
I/O (D3)	2	P135	G16	K20	917
I/O, V _{REF}	2	P136	H13	K21	920
V _{CCO}	2	-	V _{CCO} Bank 2*	V _{CCO} Bank 2*	-
GND	-	P137	GND*	GND*	-
I/O	2	P138	G14	K22	923
I/O	2	P139	G15	J21	926

XC2S200 Device Pinouts (Continued)

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
I/O	2	-	-	K18	929
I/O	2	-	-	J20	932
I/O	2	P140	G12	J18	935
GND	-	-	GND*	GND*	-
I/O	2	-	F16	J22	938
I/O	2	-	-	J19	941
I/O	2	-	-	H21	944
I/O	2	P141	G13	H19	947
I/O (D2)	2	P142	F15	H20	950
V _{CCINT}	-	P143	V _{CCINT} *	V _{CCINT} *	-
V _{CCO}	2	P144	V _{CCO} Bank 2*	V _{CCO} Bank 2*	-
GND	-	P145	GND*	GND*	-
I/O (D1)	2	P146	E16	H22	953
I/O, V _{REF}	2	P147	F14	H18	956
I/O	2	-	-	G21	962
I/O	2	P148	D16	G18	965
GND	-	-	GND*	GND*	-
I/O	2	-	F12	G20	968
I/O	2	-	-	G19	971
I/O	2	-	-	F22	974
I/O	2	P149	E15	F19	977
I/O, V _{REF}	2	P150	F13	F21	980
V _{CCO}	2	-	V _{CCO} Bank 2*	V _{CCO} Bank 2*	-
GND	-	-	GND*	GND*	-
I/O	2	P151	E14	F20	983
I/O	2	-	C16	F18	986
GND	-	-	GND*	GND*	-
I/O	2	-	-	E22	989
I/O	2	-	-	E21	995
I/O, V _{REF}	2	P152	E13	D22	998
GND	-	-	GND*	GND*	-
I/O	2	-	B16	E20	1001
I/O	2	-	-	D21	1004
I/O	2	-	-	C22	1007
I/O (DIN, D0)	2	P153	D14	D20	1013
I/O (DOUT, BUSY)	2	P154	C15	C21	1016
CCLK	2	P155	D15	B22	1019
V _{CCO}	2	P156	V _{CCO} Bank 2*	V _{CCO} Bank 2*	-

XC2S200 Device Pinouts (Continued)

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
V _{CCO}	1	P156	V _{CCO} Bank 1*	V _{CCO} Bank 1*	-
TDO	2	P157	B14	A21	-
GND	-	P158	GND*	GND*	-
TDI	-	P159	A15	B20	-
I/O (CS)	1	P160	B13	C19	0
I/O (WRITE)	1	P161	C13	A20	3
I/O	1	-	-	B19	9
I/O	1	-	-	C18	12
I/O	1	-	C12	D17	15
GND	-	-	GND*	GND*	-
I/O, V _{REF}	1	P162	A14	A19	18
I/O	1	-	-	B18	21
I/O	1	-	-	E16	27
I/O	1	-	D12	C17	30
I/O	1	P163	B12	D16	33
GND	-	-	GND*	GND*	-
V _{CCO}	1	-	V _{CCO} Bank 1*	V _{CCO} Bank 1*	-
I/O, V _{REF}	1	P164	C11	A18	36
I/O	1	P165	A13	B17	39
I/O	1	-	-	E15	42
I/O	1	-	-	A17	45
I/O	1	-	D11	D15	48
GND	-	-	GND*	GND*	-
I/O	1	P166	A12	C16	51
I/O	1	-	-	D14	54
I/O, V _{REF}	1	P167	E11	E14	60
I/O	1	P168	B11	A16	63
GND	-	P169	GND*	GND*	-
V _{CCO}	1	P170	V _{CCO} Bank 1*	V _{CCO} Bank 1*	-
V _{CCINT}	-	P171	V _{CCINT} *	V _{CCINT} *	-
I/O	1	P172	A11	C15	66
I/O	1	P173	C10	B15	69
I/O	1	-	-	E13	72
I/O	1	-	-	A15	75
I/O	1	-	-	F12	78
GND	-	-	GND*	GND*	-
I/O	1	P174	B10	C14	81
I/O	1	-	-	B14	84
I/O	1	-	-	A14	87

XC2S200 Device Pinouts (Continued)

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
I/O	1	P175	D10	D13	90
I/O	1	P176	A10	C13	93
GND	-	P177	GND*	GND*	-
V _{CCO}	1	-	V _{CCO} Bank 1*	V _{CCO} Bank 1*	-
I/O, V _{REF}	1	P178	B9	B13	96
I/O	1	P179	E10	E12	99
I/O	1	-	-	A13	105
I/O	1	-	A9	B12	108
I/O	1	P180	D9	D12	111
I/O	1	-	-	C12	114
I/O	1	P181	A8	D11	120
I, GCK2	1	P182	C9	A11	126
GND	-	P183	GND*	GND*	-
V _{CCO}	1	P184	V _{CCO} Bank 1*	V _{CCO} Bank 1*	-
V _{CCO}	0	P184	V _{CCO} Bank 0*	V _{CCO} Bank 0*	-
I, GCK3	0	P185	B8	C11	127
V _{CCINT}	-	P186	V _{CCINT} *	V _{CCINT} *	-
I/O	0	-	-	E11	137
I/O	0	P187	A7	A10	140
I/O	0	-	D8	B10	143
I/O	0	-	-	F11	146
I/O	0	P188	A6	C10	152
I/O, V _{REF}	0	P189	B7	A9	155
V _{CCO}	0	-	V _{CCO} Bank 0*	V _{CCO} Bank 0*	-
GND	-	P190	GND*	GND*	-
I/O	0	P191	C8	B9	158
I/O	0	P192	D7	E10	161
I/O	0	-	-	C9	164
I/O	0	-	-	D10	167
I/O	0	P193	E7	A8	170
GND	-	-	GND*	GND*	-
I/O	0	-	-	D9	173
I/O	0	-	-	B8	176
I/O	0	-	-	C8	179
I/O	0	P194	C7	E9	182
I/O	0	P195	B6	A7	185
V _{CCINT}	-	P196	V _{CCINT} *	V _{CCINT} *	-
V _{CCO}	0	P197	V _{CCO} Bank 0*	V _{CCO} Bank 0*	-

XC2S200 Device Pinouts (Continued)

XC2S200 Pad Name		PQ208	FG256	FG456	Bdry Scan
Function	Bank				
GND	-	P198	GND*	GND*	-
I/O	0	P199	A5	B7	188
I/O, V _{REF}	0	P200	C6	E8	191
I/O	0	-	-	D8	197
I/O	0	P201	B5	C7	200
GND	-	-	GND*	GND*	-
I/O	0	-	D6	D7	203
I/O	0	-	-	B6	206
I/O	0	-	-	A5	209
I/O	0	P202	A4	D6	212
I/O, V _{REF}	0	P203	B4	C6	215
V _{CCO}	0	-	V _{CCO} Bank 0*	V _{CCO} Bank 0*	-
GND	-	-	GND*	GND*	-
I/O	0	P204	E6	B5	218
I/O	0	-	D5	E7	221
I/O	0	-	-	A4	224
I/O	0	-	-	E6	230
I/O, V _{REF}	0	P205	A3	B4	233
GND	-	-	GND*	GND*	-
I/O	0	-	C5	A3	236
I/O	0	-	-	B3	239
I/O	0	-	-	D5	242
I/O	0	P206	B3	C5	248
TCK	-	P207	C4	C4	-
V _{CCO}	0	P208	V _{CCO} Bank 0*	V _{CCO} Bank 0*	-
V _{CCO}	7	P208	V _{CCO} Bank 7*	V _{CCO} Bank 7*	-

04/18/01

Notes:

1. IRDY and TRDY can only be accessed when using Xilinx PCI cores.
2. Pads labelled GND*, V_{CCINT}*, V_{CCO} Bank 0*, V_{CCO} Bank 1*, V_{CCO} Bank 2*, V_{CCO} Bank 3*, V_{CCO} Bank 4*, V_{CCO} Bank 5*, V_{CCO} Bank 6*, V_{CCO} Bank 7* are internally bonded to independent ground or power planes within the package.

Additional XC2S200 Package Pins
PQ208

Not Connected Pins					
P55	P56	-	-	-	-

11/02/00

FG256

V _{CCINT} Pins					
C3	C14	D4	D13	E5	E12
M5	M12	N4	N13	P3	P14
V _{CCO} Bank 0 Pins					
E8	F8	-	-	-	-
V _{CCO} Bank 1 Pins					
E9	F9	-	-	-	-
V _{CCO} Bank 2 Pins					
H11	H12	-	-	-	-
V _{CCO} Bank 3 Pins					
J11	J12	-	-	-	-
V _{CCO} Bank 4 Pins					
L9	M9	-	-	-	-
V _{CCO} Bank 5 Pins					
L8	M8	-	-	-	-
V _{CCO} Bank 6 Pins					
J5	J6	-	-	-	-
V _{CCO} Bank 7 Pins					
H5	H6	-	-	-	-
GND Pins					
A1	A16	B2	B15	F6	F7
F10	F11	G6	G7	G8	G9
G10	G11	H7	H8	H9	H10
J7	J8	J9	J10	K6	K7
K8	K9	K10	K11	L6	L7
L10	L11	R2	R15	T1	T16
Not Connected Pins					
P4	R4	-	-	-	-

11/02/00

เอกสารอ้างอิง

- [1]. จรณิต แก้วกั้งวาล **โครงสร้างไฟล์ข้อมูลอัลกอริทึมและการจัดการไฟล์ข้อมูล.**
กรุงเทพมหานคร : สำนักพิมพ์ซีเอ็ดยูเคชั่น . 2537
- [2]. ทรงลักษณ์ พิริยะไพโรจน์, สุมนา เกษมสวัสดิ์ **เรียนลัด Data Structure ด้วย Visual Basic .** กรุงเทพมหานคร : สำนักพิมพ์ซีเอ็ดยูเคชั่น . 2544
- [3]. นอ.ชาติชาย ดิษฐกุล **เอกสารประกอบการสอนวิชา VHDL**
ภาควิชาวิศวกรรมคอมพิวเตอร์
- [4]. วัชรกร หนูทอง **VHDL Application Workshop : ฝายออกแบบวงจรรวม ศูนย์วิจัย**
และพัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์แห่งชาติ (TMEC)
- [5]. วัชรกร หนูทอง **FPGA Design Workshop : ฝายออกแบบวงจรรวม ศูนย์วิจัยและ**
พัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์แห่งชาติ (TMEC)
- [6]. จันจิรา เจือกโ้ว้น , ชำนาญ ปัญญาใส **การออกแบบวงจรด้วย FPGA : ฝายออกแบบวง**
จรรวมศูนย์วิจัยและพัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์แห่งชาติ (TMEC)
- [7]. Peter J. Ashenden **The Designer 's Guide to VHDL .**
San Francisco, California , Morgan Kaufmann . 1995
- [8]. Mano, M. Morris **Logic Computer Design Fundamentals 2 Edition.** New Jersey :
Prentice Hall . 1997
- [9]. Zainalabedin Navabi **Vhdl Analysis And modeling Of Digital System 2 Edition.**
McGraw-Hill 1998
- [11]. Maek Zwolinski **Digital System Design with VHDL .** Prentice Hall 2000