อุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์

Single chip VGA Controller

โดย

นายปริญญา โพธิ์คำ รหัส 47010435

นายพลวัฒน์ กลับคง รหัส 47010493

อาจารย์ที่ปรึกษา

ดร.กสิน วิเชียรชม

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

อุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์

Single chip VGA Controller

โดย

นายปริญญา โพธิ์คำ

นายพลวัฒน์ กลับคง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ ปีการศึกษา 2550

ภาควิชา อิเล็กทรอนิกส์

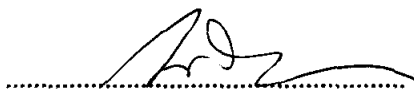
คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง อุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์

(Single chip VGA Controller)

ผู้จัดทำ 1.นายปริญญา โพธิ์คำ รหัส 47010435 ชั้นปีที่ 4

2 .นายพลวัฒน์ กลับคง รหัส 47010493 ชั้นปีที่ 4



.....อาจารย์ที่ปรึกษา

(ดร.กสิน วิเชียรชม)

อุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์

นายปริญญา โพธิ์คำ รหัส 47010435
นายพลวัฒน์ กลับคง รหัส 47010493
ดร.กสิน วิเชียรชม อาจารย์ที่ปรึกษา
ปีการศึกษา 2550

บทคัดย่อ

เครื่องแสดงภาพบนจอมอนิเตอร์ผ่านทางพอร์ต D-Sub ทำหน้าที่แสดงไฟล์ภาพที่มีอยู่ใน SD Card ออกทางจอมอนิเตอร์ ซึ่งในอุปกรณ์ชิ้นนี้มีส่วนการทำงาน 2 ส่วน คือ ส่วนเก็บข้อมูล โดยจะนำข้อมูลจาก SD Card ซึ่งจะมีการเก็บข้อมูลภาพในรูปแบบของ bitmap โดยใช้ระบบไฟล์แบบ FAT32 และส่วนควบคุมการทำงานซึ่งใช้ไมโครคอนโทรเลอร์อ่านไฟล์จาก SD Card นำมาประมวลผล ส่งเป็นสัญญาณภาพผ่านพอร์ต D-Sub เพื่อแสดงผลบนจอภาพภายนอก การติดต่อระหว่างไมโครคอนโทลเลอร์กับ SD Card ใช้การเชื่อมต่อแบบ SPI โดยอุปกรณ์ชิ้นนี้สามารถที่จะควบคุมการเลือกดูไฟล์ภาพได้

Single chip VGA Controller

Mr. Parinya Phokham ID.47010435
Mr. Phonlawat Klubkong ID.47010493
Dr. Kasin Vichienchom Advisor
Education Year 2007

Abstract

This report describes a design of the device that generates VGA signal to display image data from the SD Card. It consists of two parts, a data storage unit and a VGA signal generator. The SD Card stores the image file in Bitmap using FAT32 file system and interfaces with the generator unit, the ARM7 microcontroller, via SPI Bus. The ARM7 generates VGA signal to drive the external monitor through D-Sub port.

กิตติกรรมประกาศ

โครงงานอุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์ ซึ่งประกอบด้วยด้วยชิ้นงานและเอกสารประกอบโครงงานนี้ จะสำเร็จลุล่วงมาด้วยดีมิได้หากขาด อาจารย์กสิน วิเชียรชม อาจารย์ที่ปรึกษาผู้คอยให้คำแนะนำ และดูแลอย่างใกล้ชิดมาโดยตลอด พร้อมทั้งการให้ความช่วยเหลือจากเพื่อนในภาควิชา สุดท้ายบุคคลที่จะลืมมิได้เลยคือ ผู้ปกครองซึ่งคอยสนับสนุนและคอยให้กำลังใจเสมอมา

นายปริญญา โพธิ์คำ
นายพลวัฒน์ กลับคง
ผู้จัดทำ

สารบัญ

สารบัญรูป

หน้า

สารบัญรูป

หน้า

สารบัญตาราง

หน้า

บทที่ 1

บทนำ

1.1 ความเป็นมาของโครงงาน

ปัจจุบันเทคโนโลยีในการเก็บรูปภาพเป็นไฟล์ดิจิตอล มีการใช้งานแพร่หลายมากขึ้นโดยวิธีการนำภาพออกมาแสดงนั้นมีหลายรูปแบบ ทั้งเปิดดูในคอมพิวเตอร์ เครื่องเล่นไฟล์ต่าง ๆ รวมทั้งพิมพ์ออกมาเป็นรูปภาพ โดยผู้จัดทำต้องการนำมาแสดงบนจอมอนิเตอร์ โดยไม่ต้องทำงานผ่านเครื่องคอมพิวเตอร์ เพื่อการประหยัดพลังงาน

เทคโนโลยีเกี่ยวกับการ์ดหน่วยความจำ (Memory Card) ได้มีการพัฒนาออกสู่ตลาดมากมายหลายรูปแบบ เช่น Memory Stick, Smart media, CF, SD/MMC Card เป็นต้น การใช้งานก็จะมีลักษณะแตกต่างกัน ในส่วนของผู้จัดทำได้เลือกใช้ SD Card ในการจัดเก็บไฟล์ภาพ เนื่องจากเห็นว่ามีความเหมาะสมในเรื่องของราคาที่ไม่แพงมากและหาซื้อได้ตามท้องตลาดทั่วไป

สุดท้ายในส่วนของไมโครคอนโทรลเลอร์ เนื่องจากผู้จัดทำต้องการไมโครคอนโทรลเลอร์ที่มีการประมวลผลทางสัญญาณที่รวดเร็ว และกินกำลังไฟต่ำ จึงได้เลือกใช้ไมโครคอนโทรลเลอร์ตระกูล ARM7 (LPC2119) ในการควบคุมการทำงานทั้งหมด

1.2 วัตถุประสงค์

เพื่อศึกษาการอ่าน-เขียน ข้อมูลจากการ์ดหน่วยความจำแบบ SD Card ศึกษาการเข้ารหัสไฟล์ภาพแบบ Bitmap ศึกษาการติดต่อสื่อสารข้อมูลในลักษณะ SPI Bus ศึกษาเกี่ยวกับการสร้างสัญญาณภาพบนจอมอนิเตอร์ และศึกษาการใช้งานไมโครคอนโทรลเลอร์ตระกูล ARM7

1.3 ขอบเขตของโครงงาน

โครงงานอุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์นี้ ได้ออกแบบและสร้างวงจรในการอ่าน-เขียนการ์ดหน่วยความจำแบบ SD Card ส่วนการถอดรหัสไฟล์ Bitmap จะใช้ไมโครคอนโทรลเลอร์ในการถอดรหัส และมีการสร้างสัญญาณภาพแบบอนาลอกผ่าน Port D-Sub เพื่อนำไปแสดงผลบนจอภายนอก ในการควบคุมการทำงานทั้งหมดจะใช้ไมโครคอนโทรลเลอร์ ARM7 (LPC2119) ในการควบคุมการทำงานทั้งหมด

1.4 โครงสร้างของรายงาน

รายงานฉบับนี้ได้อธิบายขั้นตอน วิธีในการออกแบบ รวมทั้งวงจรและผลการทดลอง ทดสอบคุณสมบัติต่างๆของอุปกรณ์เครื่องนี้ โดยเนื้อหาแบ่งเป็นบทต่างๆดังนี้

บทที่ 2 ทฤษฎี กล่าวถึงทฤษฎีและหลักการพื้นฐานต่างๆ ที่เกี่ยวข้องกับการออกแบบและสร้างอุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์

บทที่ 3 การออกแบบ กล่าวถึงขั้นตอนในการออกแบบและการติดต่อกับการ์ดหน่วยความจำแบบ SD Card โดยใช้ไมโครคอนโทรลเลอร์ ARM7

บทที่ 4 การทดลองและผลการทดลอง กล่าวถึงการทดลองและผลการทดลองเมื่อทำการทดสอบสั่งงานอุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์ด้วย Input ต่างๆ

บทที่ 5 สรุปผลการทดลอง และวิจารณ์ผลการทดลอง

บทที่ 2

ทฤษฎี

2.1 ลักษณะเฉพาะของ SD Card

ลักษณะภายนอกของ SD Card นั้นจะมีขนาดที่มีการกำหนดไว้เป็นมาตรฐาน คือ มีความกว้าง 24 mm ความยาว 32 mm และความหนา 2.1 mm มีน้ำหนักโดยประมาณ 2 กรัม

การ์ด SD จะมีขาสัมผัสทั้งหมด 9 ขา ประกอบด้วยขาที่ใช้ต่อกับแหล่งจ่ายไฟ VCC VSS และขาสัญญาณเพื่อใช้ติดต่อกับตัวอุปกรณ์ โดยตัวการ์ด SD จะมีการทำงาน 2 โหมด คือ SD Bus และ SPI Bus ทำให้ขาที่ใช้ในการติดต่อ จะมีหน้าที่การทำงานได้ 2 หน้าที่

2.1.1 วิธีการส่งข้อมูลแบบ SD Bus

การส่งแบบนี้จะใช้สายติดต่อ จำนวน 6 สาย และสายที่ใช้จ่ายไฟอีก 2 สาย ประกอบด้วย

1. CMD: Command is bi-directional signal.(Host and card driver are operating in push pull mode.)
2. DAT0-3: Data lines are bi-directional signals. (Host and card drivers are operating in push pull mode.)
3. CLK: Clock is a host to cards signal. (CLK operates in push pull mode.)
4. *VDD: VDD is the power supply line for all cards*
5. VSS: VSS are two ground lines

ขั้นตอนการตรวจสอบ คำสั่งจะถูกส่งไปที่การ์ดแต่ละตัวโดยจะยอมให้ Application ตรวจสอบการ์ดและระบุ Logical Address เพื่อนำไปสู่ Physical Slot ข้อมูลจะถูกส่งไปที่การ์ดอย่างสม่ำเสมอ อย่างไรก็ดีเพื่อง่ายต่อการควบคุม Card Stack หลังจากเสร็จสิ้นขั้นตอนการตรวจสอบ คำสั่งทั้งหมดจะถูกส่งไปพร้อมๆกันกับทุกการ์ด Address Information จะถูกเตรียมมาใน Command Packet

SD Bus ยอมให้มีการปรับแต่งจำนวนสายส่งข้อมูลหลังจากที่ปรับปรุงเพิ่มประสิทธิภาพ จากปกติ SD Card จะใช้แค่ DAT0 เพื่อการส่งข้อมูลหลังจากขั้นตอนการตรวจสอบ Host สามารถที่จะเปลี่ยนความกว้างของระบบ Bus (จำนวนของสายส่งข้อมูล) ขั้นตอนการติดต่อแบบนี้มีความสมดุลระหว่าง ราคาของอุปกรณ์ และประสิทธิภาพของระบบ

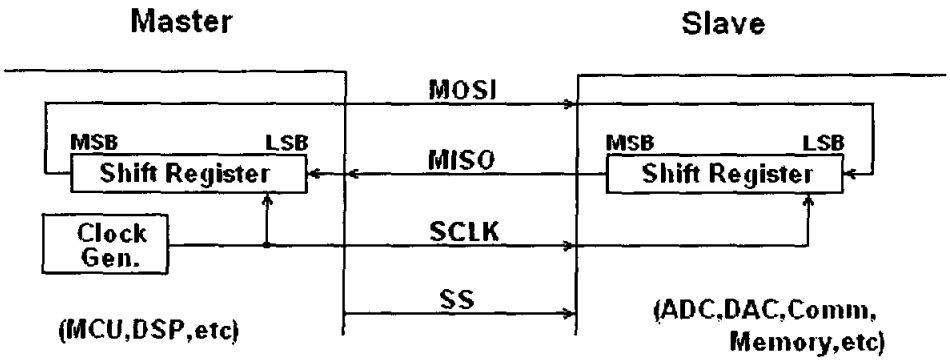
2.1.2 วิธีการส่งข้อมูลแบบ SPI Bus

SD Card แบบ SPI interface มีความเข้ากันได้กับ SPI Hosts ที่มีทั่วไปในตลาด เหมือนอุปกรณ์ที่ติดต่อแบบ SPI อื่นๆ SD card แบบ SPI Channel ประกอบด้วย

1. CS: Host to card Chip Select signal.
2. SCLK: Host to card clock signal.
3. Data In: Host to card data signal.
4. Data Out: Card to host data signal.

โครงสร้างของการเชื่อมต่อแบบ SPI แสดงดังรูปที่ 2.1 ตัว Master และตัว Slave จะติดต่อกันด้วยสายสัญญาณสามเส้น คือ SCLK (Serial Clock), MISO (Master-In Slave-Out) และ MOSI (Master-Out Slave-In) ส่วนสาย SS (Slave Selected) ที่เพิ่มขึ้นมานั้นจะใช้เป็นสายสำหรับเลือกตัว Slave ที่จะมาติดต่อกับตัว Master

ในโหมด SPI นี้ การส่งข้อมูลจะทำการส่ง MSB (Most Significant Bit) ออกไปก่อน



รูปที่ 2.1 โครงสร้างอย่างง่ายของการติดต่อในโหมดSPI

ลักษณะเฉพาะของ SPI Bus ทั้งที่ติดตั้งอยู่ใน SD Card ก็ดี คือการส่งข้อมูลแบบ Byte ข้อมูลทั้งหมดจะถูกแปลงเป็น 8 bit Bytes และจะถูกเรียงเป็นเส้นเดียว คือสัญญาณ CS

DPI Standard จะกำหนด Physical Link เท่ากัน จะไม่เกี่ยวกับการส่ง Data transfer protocol ในโหมด SPI Bus SD card จะถูกใช้เป็นส่วนย่อยของ SD Protocol และเซตคำสั่ง

การยืนยัน SD card และ Addressing Algorithms จะถูกแทนโดยสัญญาณ Hardware Chip Select (CS) การ์ดจะถูกเลือกในทุกๆคำสั่ง โดยการ Active Low ที่ขาสัญญาณ CS

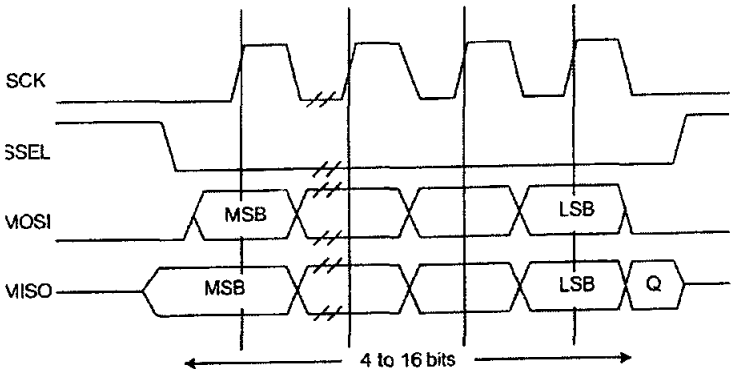
สัญญาณ CS จะถูก Active ต่อเนื่องเป็นช่วงเวลาของ SPI transaction (คำสั่ง, การตอบสนอง และข้อมูล) ข้อยกเว้นข้อเดียวคือ การ์ด Programming Time ที่เวลานี้ Host สามารถที่จะยืนยัน CS Signal โดยที่ไม่ต้องผ่านกระบวนการ Programming

ในระบบ SPI Bus จะไม่มีการกำหนด Address การเลือกอุปกรณ์ตัวที่ต้องการติดต่อ ทำได้ด้วยการส่งสัญญาณไปที่ขา Slave Select ซึ่งเป็นขาแยกต่างหาก ดังนั้นถ้ามีอุปกรณ์ SPI หลายตัวที่ต่อร่วมกันจะต้องต่อขา SCK, MISO, MOSI ขนานกันได้ แต่ขาที่ต่อไปยัง Slave Select จะต้องแยกจากกัน

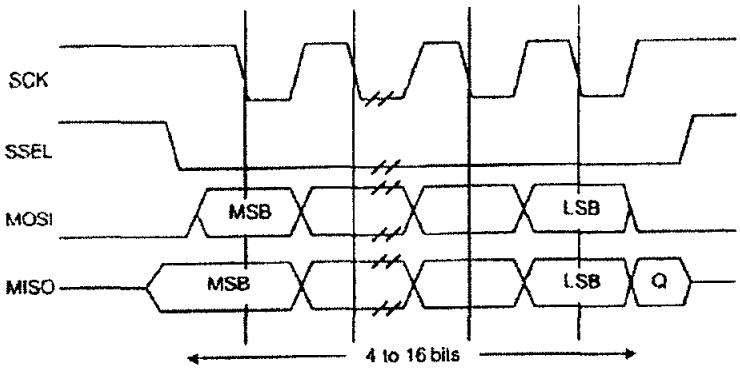
ใน SPI Bus จะแบ่งอุปกรณ์ได้เป็น 2 ประเภท คือ Master และ Slave ถ้าให้ทำงานเป็น Master ทำได้โดยการต่อขา SSEL ให้มีค่า Logic เป็น 1 ถ้าทำงานเป็น Slave ทำได้โดยต่อขา SSEL เข้ากับอุปกรณ์ที่เป็น Master

อุปกรณ์ที่เป็น Master จะเป็นผู้ควบคุม SPI Bus โดยส่งสัญญาณ SCK ไปให้อุปกรณ์ทุกตัว เมื่อต้องการเลือกอุปกรณ์ตัวใด ให้ส่งค่า Logic 0 ไปยังขา SSEL (ในอุปกรณ์ SPI จำนวนมากจะตั้งชื่อขาสัญญาณนี้เป็น CS) ของอุปกรณ์แล้วจึงส่งข้อมูลให้อุปกรณ์ทางขา MOSI ถ้าจะรับข้อมูลจากอุปกรณ์ รับทางขา MISO

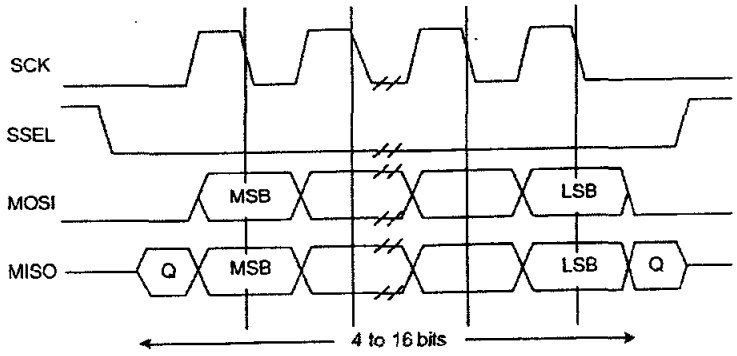
ขั้นตอนต่อมาเป็นการกำหนดการขั้วของสัญญาณนาฬิกา (Clock Polarity: CPOL) ว่าให้ทำงานที่ระดับแรงดัน 1 หรือ 0 และกำหนดเฟสของสัญญาณนาฬิกา ซึ่งกำหนดได้หลายแบบ และจะมีผลต่อการส่งข้อมูล เมื่อกำหนดค่า CPOL และ CPHA ต่างกัน ดังรูป



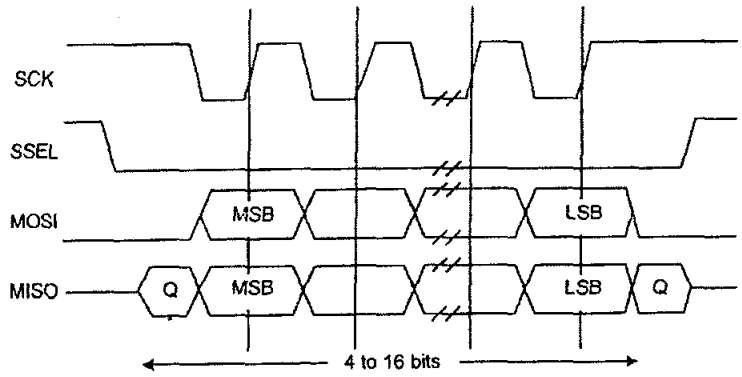
รูปที่ 2.2 รูปแบบเฟรมของ SPI เมื่อ CPOL=0 และ CPHA=0



รูปที่ 2.3 รูปแบบเฟรมของ SPI เมื่อ CPOL=0 และ CPHA=1



รูปที่ 2.4 รูปแบบเฟรมของ SPI เมื่อ CPOL=1 และ CPHA=0



รูปที่ 2.5 รูปแบบเฟรมของ SPI เมื่อ CPOL=1 และ CPHA=1

เมื่อกำหนดค่าตัวแปร CPOL และ CPHA แล้วจะแสดงความสัมพันธ์ ระหว่างสัญญาณ SCK และข้อมูลได้ดังตาราง

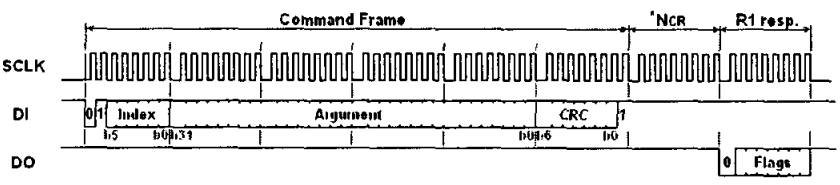
ตารางที่ 2.1 แสดงความสัมพันธ์ ระหว่าง CPHA และข้อมูล

CPOL และ CPHA	ข้อมูลบิตแรกจะมาเมื่อ	ข้อมูลบิตอื่นๆ	สุ่มข้อมูลที่
CPOL = 0, CPHA = 0	ก่อนที่จะพบขอบขาขึ้นของ SCK ตัวแรก	ขอบขาลงของ SCK	ขอบขาขึ้นของ SCK
CPOL = 0, CPHA = 1	ขอบขาขึ้นของ SCK ตัวแรก	ขอบขาขึ้นของ SCK	ขอบขาลงของ SCK
CPOL = 1, CPHA = 0	ก่อนที่จะพบขอบขาลงของ SCK	ขอบขาขึ้นของ SCK	ขอบขาลงของ SCK
CPOL = 1, CPHA = 1	ขอบขาลงของ SCK ตัวแรก	ขอบขาลงของ SCK	ขอบขาขึ้นของ SCK

ในการกำหนดค่าของ CPOL และ CPHA จะต้องกำหนดตามคู่มือของอุปกรณ์ SPI ที่ต้องการติดต่อ

2.1.3 คำสั่งและการตอบกลับ

ในโหมด SPI นั้น โครงสร้างของคำสั่งจะถูกกำหนดไว้ให้มีความยาว 6 Byte ดังแสดงในรูปที่ 2.6 เมื่อชุดคำสั่งถูกส่งไปที่การ์ด จะมีการตอบกลับจากการ์ดไปที่ Host ในรูปแบบ R1, R2 หรือ R3 เนื่องจากการถ่ายโอนข้อมูลนั้นจะถูกขับโดย Clock แบบอนุกรมที่ Host สร้างขึ้น ดังนั้น Host จะต้องสร้าง Clock ต่อไปเรื่อยๆ จนกว่าจะได้รับการตอบกลับจากการ์ด ช่วงระยะเวลาก่อนที่การ์ดจะตอบกลับหลังจากได้รับคำสั่งจาก Host (NCR) คือ 0 ถึง 8 Byte สำหรับ SDC และ 1 ถึง 8 Byte สำหรับ MMC ในช่วงที่มีการโอนถ่ายข้อมูลนี้ ขา SS จะต้องถูกกำหนดให้มีสถานะลอจิกเป็น 0 เสมอ



รูปที่ 2.6 Command Frame ที่ส่งจาก Host ไปหาการ์ด

2.1.4 ชุดคำสั่งที่ใช้ในโหมด SPI

คำสั่งดังแสดงให้เห็นนี้เป็นเพียงคำสั่งโดยทั่วไปที่ใช้สำหรับการอ่าน เขียนและ Initial Card เท่านั้น

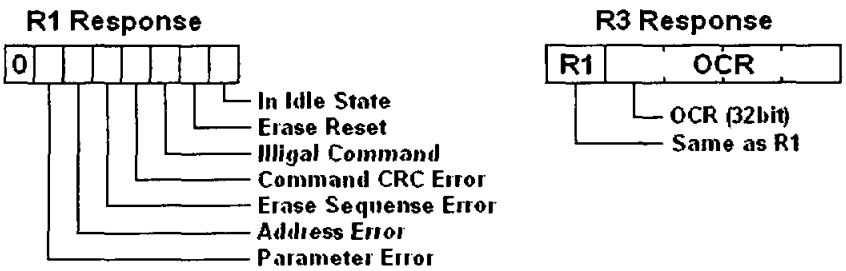
ตารางที่ 2.2 แสดงชุดคำสั่งของโหมด SPI

Command Index	Argument	Response	Data	Abbreviation	Description
CMD0	None(0)	R1	No	GO_IDLE_STATE	Software reset.
CMD1	None(0)	R1	No	SEND_OP_COND	Initiate initialization process.
CMD9	None(0)	R1	Yes	SEND_CSD	Read CSD register.
CMD10	None(0)	R1	Yes	SEND_CID	Read CID register.
CMD12	None(0)	R1b	No	STOP_TRANSMISSION	Stop to read data.
CMD17	Address[31:0]	R1	Yes	READ_SINGLE_BLOCK	Read a block.
CMD18	Address[31:0]	R1	Yes	READ_MULTIPLE_BLOCK	Read multiple blocks.
CMD23	Number of blocks[15:0]	R1	No	SET_BLOCK_COUNT	For only MMC. Define number of blocks to transfer with next multi-block read/write command.
ACMD23 (*1)	Number of blocks[22:0]	R1	No	SET_WR_BLOCK_ERASE_ COUNT	For only SDC. Define

					number of blocks to pre-erase with next multi-block write command.
CMD24	Address[31:0]	R1	Yes	WRITE_BLOCK	Write a block.
CMD25	Address[31:0]	R1	Yes	WRITE_MULTIPLE_BLOCK	Write multiple blocks.
CMD55	None(0)	R1	No	APP_CMD	Application specific command.
CMD58	None(0)	R3	No	READ_OCR	Read OCR.
*1: ACMD <n> means a command sequence of CMD55-CMD <n>.					

2.1.5 Command Response ในโหมดSPI

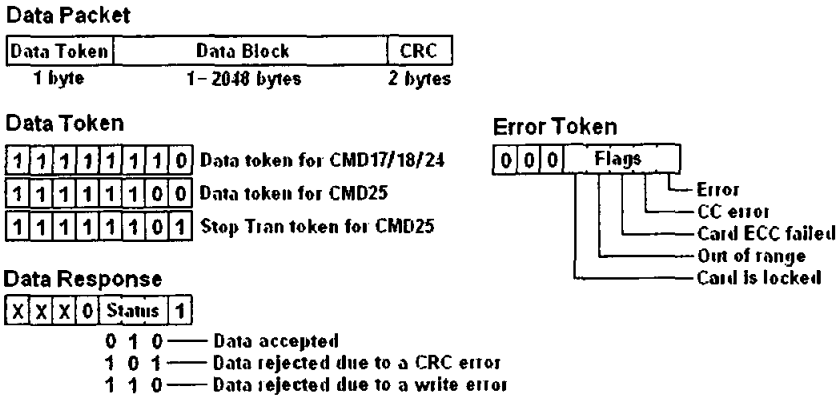
การตอบกลับในโหมด SPI จะมีอยู่สามรูปแบบ คือ R1, R2 และ R3 ดังรูปที่ 2.7 ขึ้นอยู่กับคำสั่งที่ Host ส่งมา แต่คำสั่งส่วนมากจะมีการตอบกลับในรูปแบบ R1 ถ้าหาก R1 มีค่าเป็น 0x00 แสดงว่าไม่มีข้อผิดพลาดเกิดขึ้น ส่วนการตอบสนองแบบ R3 นั้นจะตอบกลับเฉพาะ CMD58 เท่านั้น



รูปที่ 2.7 การตอบกลับแบบต่างๆ

2.2 การถ่ายโอนข้อมูล

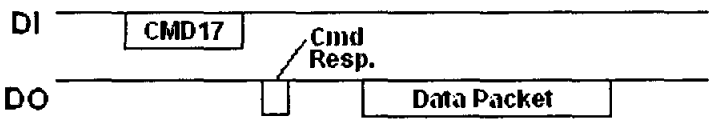
2.2.1 Data Packet และ Data response



รูปที่ 2.8 โครงสร้างของData Packet

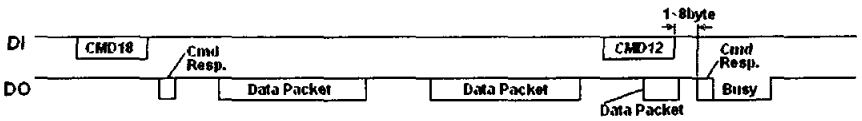
ในการถ่ายโอนข้อมูล จะมีการถ่ายโอนหลังจากมีการส่ง Command Response แล้ว บล็อกข้อมูลจะถูกถ่ายโอนในรูป Data Packet ซึ่งประกอบไปด้วย Data Token, Data Block และ CRC ดังแสดงในรูปที่ 2.8 จะเห็นได้ว่า Data Token จะมีอยู่สามแบบ ซึ่งขึ้นอยู่กับแต่ละคำสั่งที่ส่ง

2.2.2 คำสั่ง Single Block Read



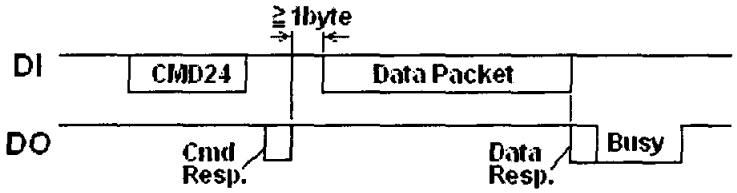
Argument ที่ต่อท้ายคำสั่งนี้จะเป็นตัวกำหนดแอดเดรสเริ่มต้นของข้อมูลที่จะอ่าน เมื่อคำสั่งนี้ได้รับการตอบกลับ กระบวนการอ่านข้อมูลก็จะเริ่มขึ้นโดยชุดข้อมูลที่ต้องการอ่านนั้นก็จะถูกส่งไปยัง Host เมื่อ Host ตรวจพบ Data Token ที่ส่งมา Host ก็จะทำการอ่านบล็อกข้อมูลที่ตามหลัง Token นั้น เมื่อมีความผิดพลาดเกิดขึ้นในการทำงาน Token ที่แสดงความผิดพลาดจะถูกส่งมาแทนที่บล็อกข้อมูล

2.2.3 คำสั่ง Multiple Block Read



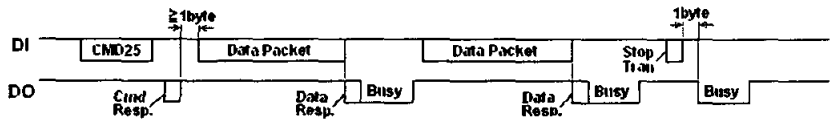
คำสั่งนี้เป็นการอ่านข้อมูลครั้งละหลายๆ บล็อกจากตำแหน่งแอดเดรสที่กำหนด ถ้าหากไม่มีการกำหนดจำนวนบล็อกในการอ่านต่อหนึ่งครั้งไว้ กระบวนการอ่านนี้จะดำเนินต่อเนื่องไปเรื่อยๆ จนกว่าจะมีการส่งคำสั่ง CMD12 ไปที่การ์ด การอ่านข้อมูลนี้จึงจะหยุดลง

2.2.4 คำสั่ง Single Block Write



เมื่อคำสั่งนี้ได้รับการตอบรับ Host ก็จะทำการส่งแพ็คเกจข้อมูลไปที่การ์ดหลังจากนั้นเป็นช่วงประมาณ 1 Byte แพ็คเกจของข้อมูลนี้ก็จะมีลักษณะเหมือนกับแพ็คเกจของข้อมูลในคำสั่ง Read เมื่อข้อมูลถูกส่งไปที่การ์ดแล้ว การ์ดก็จะตอบกลับมาด้วย Data Response ในทันที การ์ดโดยส่วนใหญ่จะสามารถเขียนข้อมูลได้บล็อกละ 512 Byte

2.2.5 คำสั่ง Multiple Block Write



คำสั่งนี้จะเขียนข้อมูลครั้งละหลายๆ บล็อกลงไปในแอดเดรสที่เรากำหนด ถ้าหากไม่มีการกำหนดจำนวนบล็อกในการถ่ายโอน การถ่ายโอนนี้ก็จะดำเนินต่อไปเรื่อยๆ จนกว่าจะถูกหยุดโดย Stop Tran Token ซึ่งก็จะทำให้เกิด Busy Flag ขึ้นเป็นช่วงระยะ 1 Byte ต่อจาก Stop Tran Token

2.2.6 ขั้นตอนการทำงานเมื่อเป็น Master

เมื่อกำหนดให้ Microcontroller เป็น Master จะมีขั้นตอนต่างๆ ในการทำงานดังนี้

1. กำหนดค่า Register SPI Clock Counter ให้สัญญาณนาฬิกามีค่าตามต้องการ
2. กำหนดค่า Register ควบคุมการทำงานของ SPI ให้มีค่าตามที่ต้องการ
3. เขียนข้อมูลที่ต้องการส่งไปยัง Register เก็บข้อมูลของ SPI การเขียนข้อมูลนี้จะสั่งให้เริ่มต้นส่งข้อมูล
4. เมื่อส่งข้อมูลครบทุกบิตแล้ว บิต SPIF ของ SPI Status Register จะมีค่าเป็น 1
5. อ่านค่าสถานะของ SPI จาก SPI Status Register
6. อ่านข้อมูลที่ได้รับจาก SPI Data Register
7. กลับไปยังข้อ 3 ถ้ามีข้อมูลที่ต้องการส่งอีก

หมายเหตุ : ในการล้างค่าบิต SPIF ทำได้โดยการอ่านหรือเขียนค่าไปยัง SPI Data Register

2.2.7 การกำหนดค่าเริ่มต้นสำหรับการติดต่อกับอุปกรณ์ผ่าน SPI0

ภายในไมโครคอนโทรลเลอร์ตระกูล LPC2000 มีวงจรสำหรับติดต่อกับบัส SPI จำนวนสองวงจร คือ SPI0 และ SPI1 โดย SPI1 มีชื่อเรียกเฉพาะว่า Synchronous Serial Port (SSP)

หลังจากที่สั่งที่ Register PINSEL0 ให้ขาที่เกี่ยวข้องมีการทำงานเป็น SPI0 แล้ว ขั้นต่อมาจะต้องกำหนดค่าให้กับ Register ที่เกี่ยวข้องกับ SPI0 ซึ่งมีทั้งหมด 5 ตัวดังตาราง

ตารางที่ 2.3 Register ที่เกี่ยวข้องกับ SPI0 ทั้ง 5 ตัว

ชื่อ	ความหมาย	การติดต่อ	ค่าหลังรีเซท	Address
S0SPCR	SPI Control Register ใช้ควบคุมการทำงานของ SPI	อ่าน/เขียน	0x00	0xE002 0000
S0SPSR	SPI Status Register แสดงสถานการณ์ทำงานของ SPI	อ่าน	0x00	0xE002 0004
S0SPDR	SPI Data Register เมื่อต้องการส่งข้อมูล ให้เขียนค่ามายัง Register นี้ในโหมดการรับข้อมูล ให้อ่านค่าจาก Register นี้	อ่าน/เขียน	0x00	0xE002 0008
S0SPCCR	SPI Clock Counter Register ใช้ควบคุมค่าความถี่ SCK0 ของ Master	อ่าน/เขียน	0x00	0xE002 000C
S0SPINT	SPI Interrupt Flag เก็บ Interrupt Flag ของ SPI	อ่าน/เขียน	0x00	0xE002 001C

2.3 ประเภทของไฟล์ภาพ

2.3.1 BMP (Bitmap)

Bitmap คือคำที่ใช้เรียกข้อมูลภาพที่เก็บอยู่ในหน่วยความจำซึ่งเป็นภาพที่เก็บในลักษณะ Pixel ต่อ Pixel โดยเก็บข้อมูลสีของแต่ละ Pixel เรียงกันไปในหน่วยความจำ ข้อมูลสีของแต่ละ Pixel จะมีขนาดที่ใช้ในการเก็บขึ้นอยู่กับความละเอียดสีที่ใช้ เช่นถ้าเป็น Bitmap แบบ 256 สี 1 Pixel จะมีขนาด 8 Bit (2 กำลัง 8) ดังนั้น 1 Pixel ของ Bitmap 256 สี จะมีขนาด 1 Byte โดยลักษณะของ Bitmap 256 สี ก็เป็น Array ของข้อมูลขนาด 1 Byte ในลักษณะเดียวกัน ถ้าเป็น 32 Bit ต่อ Pixel จะได้ว่า 1 Pixel มีขนาด 4 Byte ต่อ Pixel และจะเก็บได้ 2 ยกกำลัง 32 สี

เนื้อที่ที่จะต้องใช้ในการเก็บ Bitmap ในหน่วยความจำจะเป็นผลคูณของ Byte ต่อ Pixel, Width และ Height ของ Bitmap ตัวอย่างเช่น Bitmap แบบ 256 สี กว้าง 640 สูง 480 จะมีขนาดประมาณ 640 x 480 x 1 Byte

เนื่องจาก Bitmap ที่เก็บอยู่ในหน่วยความจำนั้นจะเป็นข้อมูล Pixel เรียงต่อกันไปเรื่อยๆ จนหมดภาพ ดังนั้นจึงต้องมีการกำหนดความกว้างและความยาวของ Bitmap ด้วย เพื่อให้โปรแกรมส่วนที่นำ Bitmap ไปใช้ทราบว่า Bitmap มีความกว้าง – ยาว เท่าไหร่

2.3.2 JPEG (Joint Photographic Expert Group)

JPEG คือรูปแบบการบีบอัดแฟ้มภาพแบบสูญเสีย โดยยังให้เสียความละเอียดน้อยที่สุด รูปแบบแฟ้มสำหรับวิธีการนี้ได้แก่ .jpeg, .jpg, .jpe, .jfif, .jfi (อาจจะเป็นตัวเล็กหรือตัวใหญ่ก็ได้) รูปแบบแฟ้ม JPEG

นี้ เป็นรูปแบบแฟ้มที่ใช้กันในการจัดเก็บและแลกเปลี่ยนรูปภาพบนอินเตอร์เน็ตมากที่สุด โดยเฉพาะภาพถ่าย เนื่องจากสามารถเก็บความละเอียดสูงได้โดยใช้ขนาดไฟล์ที่เล็ก สามารถเก็บภาพสีได้หลากหลายระดับความแม่นยำของสี (Bit Depth) ความสามารถในการย่อขนาดไฟล์ของแฟ้ม JPEG นั้นเกิดจากการใช้เทคนิคการย่อขนาดภาพแบบการบีบอัดคงข้อมูลหลัก (Lossy Compression) หรือการบีบอัดแบบมีความสูญเสียทำให้ไม่นิยมใช้กับภาพที่เป็น ลายเส้น หรือไอคอนต่างๆ เนื่องจากจะไม่ได้ประสิทธิภาพเท่าการเก็บในรูปแบบอื่น อย่าง PNG หรือ GIF การบีบอัดของ JPEG นั้นจะใช้เทคนิคที่เรียกว่า DCT (Discrete Cosine Transform) ซึ่งเป็นการแปลงค่าความสว่างของภาพให้อยู่ในรูปแบบเชิงความถี่ (Frequency Domain) ทำให้สามารถเลือกแทนค่าของสัมประสิทธิ์หรือในที่นี้คือแอมพลิจูดของค่าความถี่ต่างๆ ได้โดยอาศัยตัวแปรที่มีนัยสำคัญที่ต่างกันได้ การที่สามารถลดนัยสำคัญของค่าตัวเลขลงไปได้ทำให้สามารถลดขนาดของหน่วยความจำหรือขนาดไฟล์ที่ใช้เก็บตามไปได้

2.3.3 GIF (Graphics Interchange Format)

มีนามสกุล (file extension) คือ .gif เป็นวิธีการเก็บไฟล์ภาพแบบบีบอัดคล้ายกับ JPEG โดยทั่วไปแล้วไม่สามารถเก็บภาพที่ถ่ายจากธรรมชาติได้มีขนาดเล็กเท่ากับแบบ JPEG แต่สามารถเก็บภาพ เช่น ภาพการ์ตูน ได้เป็นอย่างดี นอกจากนี้ GIF ยังสามารถเก็บภาพไว้ได้หลายๆภาพ ในไฟล์เดียว จึงถูกนำไปใช้สร้างภาพเคลื่อนไหวง่ายๆ เช่น ในอินเตอร์เน็ต ในการบีบอัดของ GIF จะเป็นประเภทไม่สูญเสีย (lossless compression) โดยรองรับภาพที่มีสีสูงสุด 256 สี ด้วยเหตุนี้มาตรฐาน GIF จึงไม่ความต่อเนื่องของสี คือ นิยมใช้กับภาพถ่ายซึ่งเป็นรูปที่มีจำนวนสีมากกว่าและมีความต่อเนื่องของสี

2.4 ตารางการจัดเรียงไฟล์ (File Allocation Table: FAT)

ระบบปฏิบัติการของคอมพิวเตอร์แต่ละ Platform จะมีการจัดระบบไฟล์ในฮาร์ดดิสก์ที่แตกต่างกัน บางระบบสามารถใช้ระบบไฟล์ได้หลายรูปแบบ โดยระบบไฟล์นั้นเป็นตารางที่ใช้บอกตำแหน่งของข้อมูลต่างๆ ที่อยู่บนฮาร์ดดิสก์ว่าอะไรอยู่ตรงไหน ปกติเมื่อซื้อฮาร์ดดิสก์มาใหม่ต้องทำการจัดข้อมูล (Format) ให้ฮาร์ดดิสก์ก่อนที่จะนำไปบรรจุข้อมูล การจัดข้อมูลในฮาร์ดดิสก์เป็นการแบ่งฮาร์ดดิสก์ออกเป็นส่วนๆ เพื่อให้คอมพิวเตอร์รู้ว่าตำแหน่งของข้อมูลอยู่ตรงไหน

FAT เป็นระบบไฟล์ที่ใช้ในระบบปฏิบัติการของ Microsoft และเป็นระบบไฟล์ที่มีการพัฒนาอย่างต่อเนื่อง โดยจะมีลักษณะคือ เป็นการกำหนดหมายเลขให้กับทุกๆ Cluster ในแต่ละส่วนแบ่งของฮาร์ดดิสก์แล้วทำการสร้างตารางที่มีจำนวนช่อง ตามจำนวนของ Cluster เพื่อเป็นการระบุตำแหน่งหรือ Cluster ที่ทำการเก็บข้อมูลของไฟล์แต่ละไฟล์ และมีตารางอีกตารางที่เรียกว่า Directory สำหรับเก็บข้อมูล รายละเอียดของไฟล์ เช่น Attribute ต่างๆ และหมายเลข Cluster เริ่มต้นที่เก็บตัวข้อมูลจริงๆ

ระบบจัดการไฟล์แบบ FAT เป็นระบบที่ไม่ยุ่งยากซับซ้อน ดังนั้นจึงถูกรองรับ จากระบบปฏิบัติการที่มีอยู่สำหรับคอมพิวเตอร์ส่วนบุคคล และยังสะดวกในการแลกเปลี่ยนข้อมูลระหว่างระบบปฏิบัติการที่แตกต่างกันซึ่งถูกติดตั้งบนคอมพิวเตอร์เครื่องเดียวกัน

ข้อด้อยของระบบจัดการไฟล์แบบ FAT คือ เมื่อไฟล์ถูกลบ และไฟล์ใหม่ถูกเขียนลงไป Fragment ของแต่ละไฟล์จะมีโอกาสกระจัดกระจายออกไปอยู่ทั่วทั้งหน่วยความจำ ส่งผลให้การอ่านและการเขียนไฟล์ทำได้ช้า การจัดเรียงข้อมูลเป็นหนึ่งในวิธีการทำให้การจัดเรียงไฟล์แบบ FAT เป็นระเบียบแต่จะต้องทำการจัดเรียงข้อมูลอยู่บ่อย และเป็นกระบวนการที่ใช้เวลามาก

ระบบไฟล์ FAT มีหลายรุ่นดังต่อไปนี้

2.4.1 FAT12

ระบบ FAT12 เป็นรุ่นที่เก่าแก่ที่สุดของตระกูล FAT ใช้กับระบบปฏิบัติการ Dos ซึ่งใช้ 12 bits ในการอ้างอิงถึงหมายเลข Cluster เพราะฉะนั้นสามารถอ้างถึง Cluster ได้มากสุด 4086 Cluster ระบบ FAT12 จึงเหมาะกับหน่วยความจำที่มีเนื้อที่ไม่มาก หรือมีเนื้อที่ไม่เกิน 16 MB และระบบไฟล์ชนิดนี้สามารถใช้งานกับไฟล์ที่มีชื่อยาวเพียง 8.3 ตัวอักษรเท่านั้น

2.4.2 FAT16

สามารถใช้งานร่วมกับระบบปฏิบัติการที่หลากหลายได้ เช่น Dos Windows95, 98, ME ซึ่งใช้ 16 bits เพื่ออ้างถึงหมายเลข Cluster เพราะฉะนั้นสามารถอ้างถึงได้ทั้งหมด 65526 Cluster FAT16 นั้นถูกออกแบบมาเพื่อใช้งานกับไฟล์ต่างๆ บนดิสก์ขนาดเล็ก และขนาดกลาง ซึ่งมีเนื้อที่ประมาณ 2048 MB

ปัญหาของ FAT16 คือ

1. ระบบ FAT16 ใช้งานพื้นที่ของหน่วยความจำสิ้นเปลืองมาก เพราะจำนวนสูงสุดของ Cluster ต่อ Partition นั้นถูกกำหนดเอาไว้ตายตัว (65526 Cluster) ดังนั้นเมื่อหน่วยความจำมีขนาดที่ใหญ่ขึ้น แต่ว่าจำนวน Cluster ยังเท่าเดิม ก็หมายความว่าขนาดของ Cluster จะใหญ่ขึ้นไปด้วย อย่างในกรณีของหน่วยความจำ ขนาด 2 GB นั้นจะมี Cluster ที่ใหญ่ถึง 32 KB นั้นก็หมายความว่า ต่อให้พิมพ์เอกสารที่มีตัวอักษรเพียง 10 ตัว แต่ขนาดของไฟล์ก็จะมีถึง 32 KB ซึ่งเป็นขนาดเล็กสุดของ Cluster เลยทีเดียว
2. ขีดจำกัดเรื่องขนาดสูงสุดของหน่วยความจำที่รองรับได้เพราะเริ่มต้น FAT16 ถูกออกแบบมาเพื่อใช้งานกับดิสก์ที่มีขนาดเล็ก ดังนั้นในยุคแรกๆ จึงมีปัญหาตามมา เมื่อระบบ FAT16 ใน MS-Dos ยุคแรก สามารถรองรับหน่วยความจำได้เพียง 32 MB เท่านั้น แต่ต่อมาถูกแก้ไขให้รองรับได้เป็น 128 MB และเรื่อยมาเป็น 2 GB ในปัจจุบัน
3. ระบบ FAT16 สามารถใช้งานกับไฟล์ที่มีชื่อยาวเพียง 8.3 ตัวอักษรเท่านั้น เช่นเดียวกับ FAT12 แต่ได้รับการปรับปรุงให้มีความสามารถมากขึ้นในเวลาต่อมาเพื่อให้สามารถใช้งานกับไฟล์ที่มีชื่อยาวได้ไม่เกิน 256 ตัวอักษร เรียก FAT16 รุ่นนี้ว่า Virtual FAT หรือ VFAT

2.4.3 FAT32

ระบบ FAT32 ใช้กับระบบปฏิบัติการ Windows 95, 98, ME, 2000 โดยทำการแก้ไขเพิ่มเติมจาก FAT16 เพื่อที่จะได้มีจำนวน Cluster ต่อ Partition มากขึ้น ดังนั้นจึงมีความสามารถที่จะรองรับหน่วยความจำ

ที่มีขนาดใหญ่สูงสุด ได้ถึง 2 TB ซึ่งจะใช้ 28 bits (อีก 4 bits สำรองเอาไว้) ฉะนั้นสามารถอ้างถึง Cluster ได้ทั้งหมด 286 ล้าน Cluster และระบบ FAT32 ยังสามารถรองรับชื่อไฟล์แบบยาว คือ 255 ตัวอักษรได้อีกด้วย

ตารางที่ 2.4 เปรียบเทียบระหว่าง FAT12, FAT16 และ FAT32

	FAT12	**FAT16**	**FAT32**
Developer	Microsoft		
Full Name	File Allocation Table		
	(12-bit version)	(16-bit version)	(32-bit version)
Introduced	1977 (Microsoft Disk BASIC)	July 1988 (MS-DOS 4.0)	August 1996 (Windows 95 OSR2)
Structures			
Directory contents	Table		
File allocation	Linked List		
Bad blocks	Linked List		
Limits			
Max file size	32 MB	2 GB	4 GB
Max number of files	4,077	65,517	268,435,437
Max file name size	8.3 or 255 when using LFNs		
Max volume size	32 MB	2 GB 4 GB	2 TB
Features			
Dates recorded	Creation, modified, access		
Date range	January 1, 1980 – December 31, 2107		
Forks	Not natively		
Attributes	Read-only, hidden, system, volume label, subdirectory, archive		
Permissions	No		
Transparent compression	Per-volume, Stacker, DoubleSpace, Drive Space		No
Transparent encryption	Per-volume only with DR-DOS		No

2.4.4 โครงสร้างของดิสก์หลัก (Main Disk Structure)

Boot sector	More reserved sectors (optional)	File Allocation Table#1	File Allocation Table#2	Root Directory	Data Region (for files and directories)...(To end of partition or disk)

ระบบไฟล์แบบ FAT ประกอบไปด้วย 4 ส่วนที่แตกต่างกัน ดังนี้

1. เซคเตอร์สำรอง (Reserved sectors) อยู่ในส่วนต้นๆ โดย Reserved sector แรกสุดเป็น Boot Sector ซึ่งรวมพื้นที่ที่เรียกว่า BIOS Parameter Block โดยปกติจะประกอบด้วยโค้ดเริ่มต้นของระบบปฏิบัติการ ซึ่งจำนวนของ Reserved sector ทั้งหมดจะถูกระบุอยู่ในส่วนนี้ ข้อมูลสำคัญต่างๆ จาก Boot Sector สามารถเข้าถึงผ่านโครงสร้างระบบปฏิบัติการ ที่เรียกว่า Drive Parameter Block ใน DOS และ OS/2

2. บริเวณตารางจัดเรียงไฟล์ (FAT Region) ส่วนนี้ประกอบไปด้วยตารางการจัดเรียงไฟล์ 2 ชุด ชุดหนึ่งสำหรับใช้งานจริงและอีกชุดเป็นการสำรองไว้ใช้ในเวลาจำเป็น โดยบริเวณตารางจัดเรียงไฟล์นี้จะสอดคล้องกับ บริเวณข้อมูล (Data Region) ทำหน้าที่ระบุตำแหน่ง Cluster ของไฟล์และ Directory ต่างๆ

3. บริเวณไดเรคทอรี่ (Root Directory Region) นี่คือตาราง Directory ซึ่งเก็บข้อมูลเกี่ยวกับ File และ Directory ต่างๆ ในระบบจัดการไฟล์แบบ FAT12 และ FAT16 ตาราง Directory นี้จะอยู่ในบริเวณ Directory เท่านั้น และมีขนาดสูงสุดที่แน่นอน แต่ในระบบจัดการไฟล์แบบ FAT32 ตาราง Directory จะอยู่รวมกับไฟล์ต่างๆ ในบริเวณข้อมูล ซึ่งสามารถขยายขนาดออกไปได้ไม่จำกัด

4. บริเวณข้อมูล เป็นส่วนที่ไฟล์ข้อมูลอยู่จริง ขนาดของ File และ Directory ย่อยสามารถเพิ่มขึ้นได้เท่าที่มี Cluster เหลืออยู่

2.4.4.1 Boot Sector และโครงสร้าง BPB (Bios Parameter Block)

ตารางที่ 2.5 โครงสร้างพื้นฐาน 36 Byte แรกของ Boot Sector ซึ่งใช้ในทุกเวอร์ชันของ FAT

Byte Offset	ความยาว (Byte)	รายละเอียด
0x00	3	คำสั่งกระโดด (เพื่อจะข้ามส่วนหัวของการบูท)
0x03	8	ชื่อ OEM ค่าพื้นฐานคือ ไอบีเอ็ม 3.3 และ MS-DOS 5.0
0x0b	2	จำนวน Byte ต่อ Sector, บล็อกของพารามิเตอร์ BIOS เริ่มต้นที่นี่
0x0d	1	จำนวน Sector ต่อ Cluster
0x0e	2	Reserved Sector
0x10	1	จำนวนของตารางการจัดเรียงข้อมูล

0x11	2	จำนวนสูงสุดของ Directory
0x13	2	จำนวน Sector ทั้งหมด
0x15	1	ตัวบรรยาย Media 0xF8 ด้านเดียว, 80 แทรกต่อด้าน, 9 Sector ต่อแทรก 0xF9 สองด้าน, 80 แทรกต่อด้าน, 9 Sector ต่อแทรก 0xFA ด้านเดียว, 80 แทรกต่อด้าน, 8 Sector ต่อแทรก 0xFB สองด้าน, 80 แทรกต่อด้าน, 8 Sector ต่อแทรก 0xFC ด้านเดียว, 40 แทรกต่อด้าน, 9 Sector ต่อแทรก 0xFD สองด้าน, 40 แทรกต่อด้าน, 9 Sector ต่อแทรก 0xFE ด้านเดียว, 40 แทรกต่อด้าน, 8 Sector ต่อแทรก 0xFF สองด้าน, 40 แทรกต่อด้าน, 8 Sector ต่อแทรก
0x16	2	จำนวน Sector ต่อตารางการจัดเรียงข้อมูล
0x18	2	จำนวน Sector ต่อ Track
0x1a	2	จำนวนของ Head
0x1c	4	จำนวนของ Sector ที่โดนซ่อน
0x20	4	จำนวนของ Sector ทั้งหมด (ใช้ในกรณีมากกว่า 65535 Sector)

ตารางที่ 2.6 โครงสร้างของ Boot Sector ที่เพิ่มขึ้นมาใน FAT32

Byte Offset	**ความยาว (Byte)**	**รายละเอียด**
0x24	4	จำนวน Sector ต่อตารางการจัดเรียงข้อมูล
0x28	2	เครื่องหมายของตารางการจัดเรียงข้อมูล
0x2a	2	เวอร์ชัน
0x2c	4	หมายเลข Cluster ของ Directory เริ่มต้น
0x30	2	หมายเลข Sector ของ Sector ข้อมูล FS
0x32	2	หมายเลข Sector ของสำเนา Boot Sector
0x34	12	สำรอง
0x40	1	Physical Drive Number
0x41	1	สำรอง
0x42	1	Signature
0x43	4	หมายเลข Serial
0x47	11	Volume Label
0x52	8	ชนิดของระบบไฟล์แบบ FAT32

0x5a	420	Code Boot ระบบปฏิบัติการ
0x1FE	2	Last Sector (0x55 0xAA)

2.4.4.2 ตาราง Directory

ตาราง Directory เป็นไฟล์ชนิดพิเศษที่แสดง Directory (ปัจจุบันรู้จักกันในนามของ Folder) แต่ละไฟล์หรือ Directory ประกอบไปด้วย 32 Byte แต่ละ Byte จะบันทึกชื่อไฟล์, นามสกุลไฟล์, คุณลักษณะของไฟล์ (ชนิดเอกสารสำคัญ, Directory, ถูกซ่อน, อ่านได้อย่างเดียว, ระบบ และความจุ), วัน เวลาที่ไฟล์ถูกสร้าง Address ของ Cluster แรกของไฟล์หรือ Directory นั้น และสุดท้ายคือขนาดของไฟล์หรือ Directory

จำนวน Directory ทั้งหมดในบริเวณ Directory และใน Directory ย่อยมีรูปแบบดังนี้

ตารางที่ 2.7 โครงสร้างของตาราง Directory

Byte Offset	**ความยาว (Byte)**	**รายละเอียด**		
0x00	8	0x00		
0x08	3	เครื่องหมายของตารางจัดเรียงข้อมูล		
0x0b	1	**Bit**	**Mask**	**รายละเอียด**
		0	0x01	อ่านได้อย่างเดียว
		1	0x02	ถูกซ่อน
		2	0x04	ระบบ
		3	0x08	Volume Label
		4	0x10	Directory ย่อย
		5	0x20	เอกสารสำคัญ
		6	0x40	อุปกรณ์
		7	0x80	ไม่ใช้
0x0c	1	สำรอง ถูกใช้โดย NT		
0x0d	1	เวลาสร้างไฟล์, ความละเอียด 10 มิลลิวินาที ค่าอยู่ระหว่าง 0-199		
0x0e	2	**Bit**	**รายละเอียด**	
		15-11	ชั่วโมง (0-23)	
			นาที (0-59)	
		10-5	วินาที/2 (0-29)	
		4-0		
0x10	2	**Bit**	**รายละเอียด**	

		15-9 8-5 4-0	ปี (0=1980, 127=2107) เดือน (1= มกราคม, 12=ธันวาคม) วันที่ (1-31)
0x12	2	วันที่ใช้งานไฟล์ล่าสุด ดู Byte Offset 0x10 ประกอบ	
0x14	2	ดัชนี EA (ถูกใช้โดย OS/2 และ NT) ใน FAT12 และ FAT16 หรือ 2 Byte สูงสุดของ Cluster แรกใน FAT32	
0x16	2	เวลาที่แก้ไขไฟล์ล่าสุด ดู Byte Offset 0x0e ประกอบ	
0x18	2	วันที่แก้ไขไฟล์ล่าสุด ดู Byte Offset 0x10 ประกอบ	
0x1a	2	Cluster แรกใน FAT12 และ FAT 16 หรือ 2 Byte ต่ำของ Cluster แรกใน FAT32	
0x1c	4	ขนาดไฟล์	

2.4.4.3 ตาราง FAT

ขนาดของแต่ละ Cluster ขึ้นอยู่กับชนิดของระบบไฟล์แบบ FAT ที่ใช้และขนาดของ Partition โดยปกติขนาดของ Cluster จะอยู่ระหว่าง 2 KB ถึง 32 KB แต่ละไฟล์อาจจะกินเนื้อที่มากกว่า 1 Cluster ขึ้นอยู่กับขนาดของไฟล์ แต่ไม่จำเป็นว่าจะต้องเป็น Cluster ที่ติดกัน

ตารางการจัดเรียงข้อมูลเป็นสมุดรายชื่อของไฟล์ที่สัมพันธ์กับตำแหน่ง Cluster ของไฟล์ แต่ละ Cluster ใน FAT จะบันทึกข้อมูล 1 ใน 5 ดังนี้

- Address ของ Cluster ถัดไปของไฟล์
- สถานะแสดงจุดสิ้นสุดของไฟล์
- สถานะแสดงว่าเป็น Bad Sector
- สถานะแสดงว่าเป็น Cluster ที่ถูกสำรองไว้
- ศูนย์เพื่อแสดงว่าเป็น Cluster ที่ไม่ถูกใช้

แต่ละเวอร์ชันของระบบไฟล์แบบตารางการจัดเรียงข้อมูล จะมีขนาดของ Cluster ในตารางการจัดเรียงข้อมูลที่ต่างกัน ซึ่งขนาดจะถูกระบุโดยชื่อของแต่ละเวอร์ชัน เช่น ระบบไฟล์ FAT16 Cluster จะมีขนาด 16 Bit ในขณะที่ FAT32 จะมีขนาด 32 Bit ตามขนาดของ Partition ที่ใหญ่ขึ้น ซึ่ง FAT32 จะมีประสิทธิภาพมากกว่าในกรณี FAT16 เนื่องจาก FAT32 สามารถแบ่ง Cluster ให้มีขนาดที่เล็กกว่าซึ่งหมายความว่าจะมีพื้นที่สูญเปล่าน้อยกว่าในกรณีของ FAT16

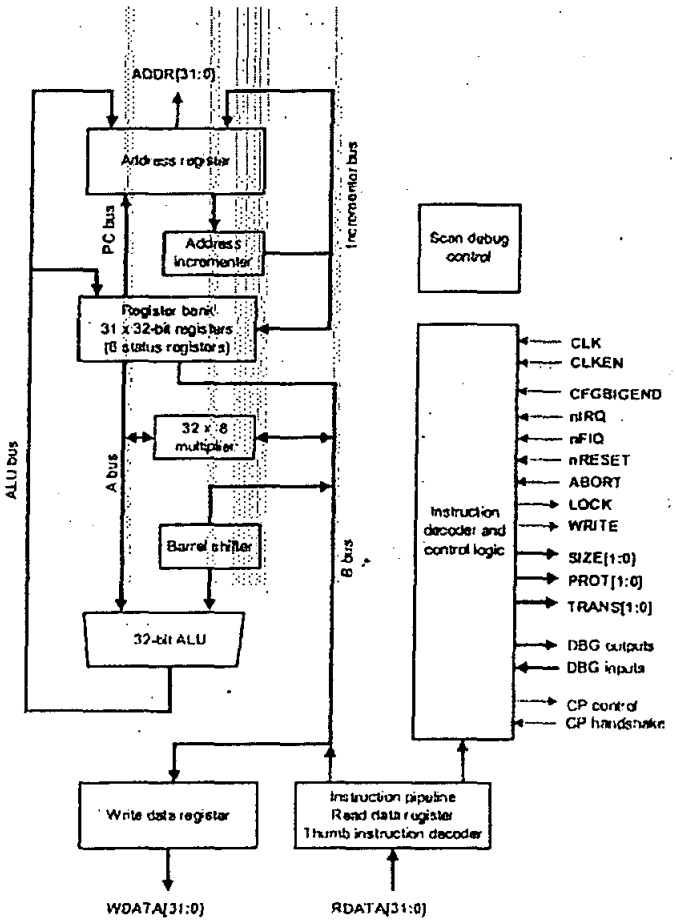
ตารางที่ 2.8 ค่าต่างๆ ในตารางการจัดเรียงข้อมูล

FAT12	FAT16	FAT32	รายละเอียด
0x000	0x0000	0x?0000000	Cluster ว่าง

0x001	0x0001	0x?0000001	Reserved Cluster
0x002 – 0xFEF	0x0002 – 0xFFEF	0x?0000002 – 0x?FFFFFEF	Cluster ที่ถูกนำไปใช้งาน, ค่าของ Cluster ถัดไป
0xFF0 – 0xFF6	0xFFF0 – 0xFFF6	0x?FFFFFF0 – 0x?FFFFFF6	ค่าสำรอง
0xFF7	0xFFF7	0x?FFFFFF7	Cluster เสีย
0xFF8 – 0xFFF	0xFFF8 – 0xFFFF	0x?FFFFFF8 – 0x?FFFFFFF	Cluster สุดท้ายของไฟล์

2.5 สถาปัตยกรรมซีพียู ARM7

สถาปัตยกรรมของ ARM7 เป็นซีพียูแบบ RISC ขนาด 32 บิต ภายในมี Bus ขนาด 32 บิต ตัวเดียวที่ใช้สำหรับรับส่งข้อมูลและคำสั่ง ชุดคำสั่งจะมีขนาด 32 บิตคงที่ ในขณะที่ข้อมูลสามารถเลือกได้ว่าจะมีขนาด 8, 16 หรือ 32 บิต โดยแสดงแกนกลาง (Core) ของ ซีพียู ARM7 ได้ดังรูป



รูปที่ 2.9 แกนกลางของซีพียู ARM7

โครงสร้างของ ARM7 จะเป็นแบบที่เรียบง่าย มีชุดคำสั่งไม่มากนัก ประหยัดพื้นที่สารกึ่งตัวนำที่ใช้สร้าง ประหยัดพลังงาน

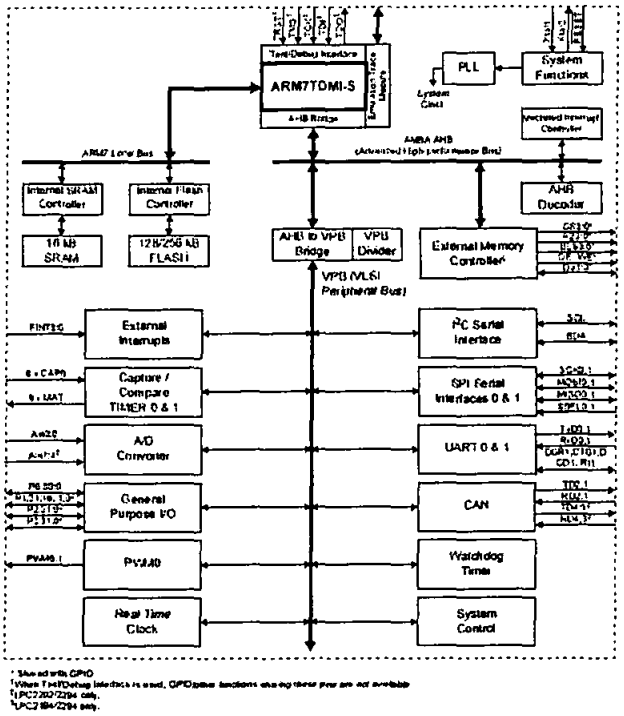
สถาปัตยกรรมของ ARM7 จะเป็นแบบ load-and-store ในการประมวลผลข้อมูลใดๆ ต้องกระทำผ่านทาง Register เริ่มต้นด้วยการโหลดค่าจากหน่วยความจำเก็บใน Register นำค่ามาประมวลผล เสร็จแล้วจะเขียนค่าเก็บในหน่วยความจำดังเดิม

Register ของ ARM7 ที่ใช้งานได้สำหรับผู้ใช้มีทั้งหมด 16 ตัว คือ R0 – R15 โดยทุกตัวมีขนาด 32 บิต โดย R0 – R12 เป็น Register ทั่วไปที่ไม่ได้กำหนดหน้าที่การทำงานพิเศษ ส่วน R13 ทำหน้าที่เป็น Stack Pointer (SP) R14 ทำหน้าที่เป็น Link Register (LR) และ R15 ทำหน้าที่เป็น Program Counter (PC)

2.5.1 ไมโครคอนโทรลเลอร์ ARM 7 Philips LPC2119

- ไมโครคอนโทรลเลอร์ขนาด 16/32 บิต ARM7TDMI-S ในตัวถัง LQFP 64 ขา
- หน่วยความจำ Static RAM ขนาด 16 kB
- หน่วยความจำ Flash Program Memory ขนาด 128 kB อยู่ภายในชิปที่สามารถ ลบ/เขียน ซ้ำได้ถึง 10,000 ครั้ง
- โปรแกรมชิปได้ทันทีผ่าน In System Programming (ISP) และ In-Application Programming (IAP) โดยใช้ซอฟต์แวร์ boot-loader ที่อยู่ภายในชิป
- วงจรแปลงแอนะลอกเป็นดิจิตอลความละเอียด 10 บิต จำนวน 4 ชุด โดยมีเวลาในการแปลงค่าต่ำถึง 2.44 ms
- วงจรไทเมอร์ขนาด 32 บิต 2 ชุด (มี 4 capture และ 4 compare channel)
- PWM (Pulse width modulation) 6 เอาต์พุต
- โมดูลนาฬิกาเวลาจริง (Real Time Clock) และ วอชดอกซ์ (Watchdog)
- วงจรสื่อสารอนุกรม UART (16C550) จำนวน 2 ชุด
- วงจรสื่อสารอนุกรม I^2C ความเร็วสูง (400 kbits/s)
- วงจรสื่อสารอนุกรม SPI 2 ชุด
- มีวงจร Phase lock loop ภายในเพื่อคูณค่าให้สัญญาณนาฬิกาภายในทำงานที่ความถี่สูงสุด 60 MHz
- Vectored Interrupt Controller ที่สามารถกำหนดลำดับความสำคัญ และกำหนดแอดเดรสของเวกเตอร์ได้
- ใช้กับแหล่งจ่ายไฟชุดเดียวขนาด 3.0 V ถึง 3.6 V (3.3 V ± 10%)
- มี I/O pin อเนกประสงค์ที่สามารถใช้กับระดับแรงดัน 5 V ได้สูงสุด 45 ขา โดยสามารถจัดเป็นขาอินเตอร์รัปต์จากภายนอกได้สูงสุด 12 ขา
- มีโหมดประหยัดพลังงาน 2 โหมด ได้แก่ Idle และ Power-down

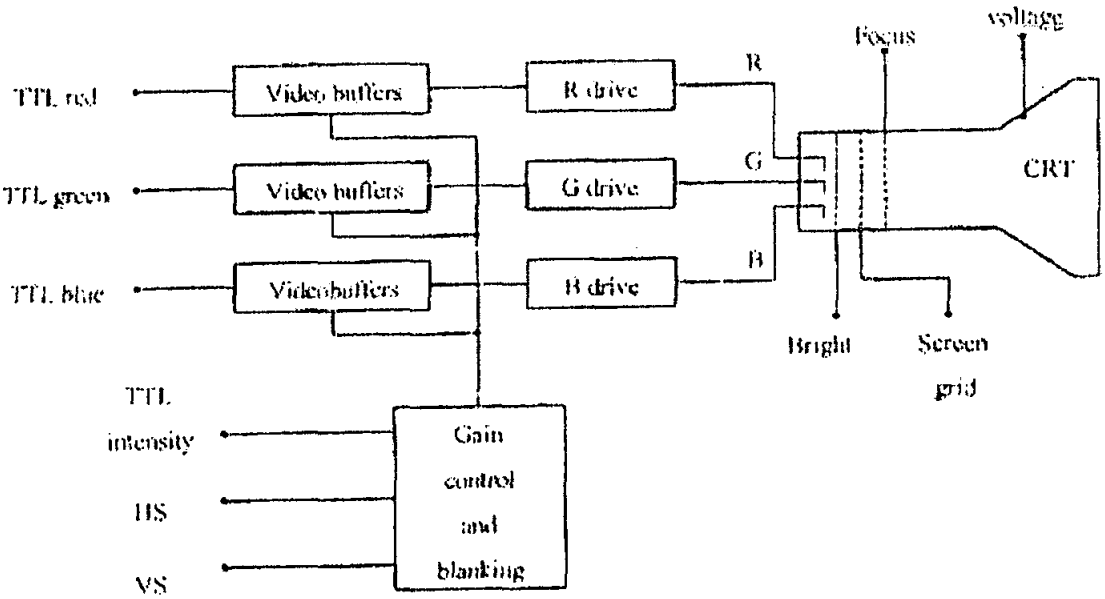
2.5.2 บล็อกไดอะแกรมของ LPC2119



2.6 การสร้างสัญญาณภาพ

2.6.1 การสร้างภาพสีของจอมอนิเตอร์

การสร้างภาพสีของจอมอนิเตอร์เกิดขึ้นได้โดยการใช้หลอดภาพสีที่มีโครงสร้างภายนอกเหมือนกับของหลอดภาพขาวดำ แต่ด้านในของจอภาพจะฉาบเอาไว้ด้วยสารฟอสเฟอร์ 3 ชนิดที่มีคุณสมบัติเปล่งแสงสีแดง สีเขียว และสีน้ำเงินออกมาเมื่อถูกลำอิเล็กตรอนวิ่งมาชน ลักษณะการฉาบสารฟอสเฟอร์จะใช้วิธีการฉาบเป็นจุดหรือเป็นเส้นที่เล็กมากเรียงสลับกันไปจนเต็มตลอดหน้าจอ เนื่องจากสารฟอสเฟอร์มี 3 ชนิด ดังนั้นจึงต้องใช้อิเล็กตรอนกันถึงสามอันเพื่อแยกยิงลำอิเล็กตรอนแต่ละลำให้ไปชนสารฟอสเฟอร์แต่ละสี โดยจะตั้งอิเล็กตรอนกันแต่ละอัน ให้ยิงไปชนแต่ละสีหนึ่งสีใดโดยเฉพาะ เช่นอิเล็กตรอนกันที่ยิงลำอิเล็กตรอนกันไปชนเฉพาะสารฟอสเฟอร์สีแดงจะทำการสร้างแต่สีแดงเท่านั้นให้ปรากฏออกมาหน้าจอภาพทุกครั้งที่อิเล็กตรอนกันอันนี้ทำงานจึงเรียกว่าอิเล็กตรอนกันสีแดง ส่วนอิเล็กตรอนกันที่เหลืออีก 2 อัน ซึ่งตั้งเล็งยิงลำอิเล็กตรอนไปชนสารฟอสเฟอร์สีเขียวและสีน้ำเงิน เพื่อสร้างสีเขียวและสีน้ำเงินให้เกิดที่หน้าจอเราเรียกว่า อิเล็กตรอนกันสีเขียว และอิเล็กตรอนกันสีน้ำเงินหลอดภาพของจอมอนิเตอร์ ก็จะสร้างแสงให้เกิดขึ้นที่หน้าจอโดยมีวงจรไบอัสจ่ายแรงดันให้กับชิ้นส่วนต่างๆ ที่อยู่ในอิเล็กตรอนกันทั้ง 3 เพื่อให้ยิงลำอิเล็กตรอนไปชนจอโดยปรับระดับไบอัสเพื่อทำให้ปริมาณอิเล็กตรอนทั้ง 3 ที่ยิงไปชนจอพร้อมๆ กันนั้นสร้างแสงขาวให้ปรากฏขึ้นที่หน้าจอ โดยมีความสว่าง 50% ขณะที่ยังไม่มีสัญญาณป้อนเข้ามายังอิเล็กตรอนกัน ลำอิเล็กตรอนทั้งสามจะถูกเบี่ยงเบนโดยสนามแม่เหล็กจากขดลวดชุดเบี่ยงเบนทางแนวนอนและแนวตั้งเพื่อให้เกิดการกราดสร้างแสงขึ้นเต็มจอ สัญญาณที่ส่งมาที่อิเล็กตรอนกันทั้งสามต้องมีสามสัญญาณเช่นกัน คือ สัญญาณสีแดงส่งป้อนให้อิเล็กตรอนกันสีแดง เพื่อให้เกิดการสร้างแสงสีแดง และสัญญาณสีเขียวกับสีน้ำเงินส่งป้อนให้กับอิเล็กตรอนกันสีเขียวกับอิเล็กตรอนกันสีน้ำเงิน



รูปที่ 2.10 การป้อนสัญญาณขับจอมอนิเตอร์

เพื่อสร้างแสงสีเขียวและสีน้ำเงิน สัญญาณทั้ง 3 จะป้อนเข้าไปเพื่อทำให้อิเล็กตรอนกันแต่ละอันเปลี่ยนแปลงปริมาณของลำอิเล็กตรอน ที่ยิงไปชนจอ เป็นการสร้างสีต่างๆ ให้เกิดบนจอตามต้องการ เช่น เมื่อส่วนของสัญญาณที่ส่งมาเป็นสีแดงอิเล็กตรอนกันสีแดงจะได้รับสัญญาณเพื่อเพิ่มปริมาณอิเล็กตรอนที่ยิงไปชนสารฟอสเฟอร์สีแดงมากขึ้นจึงเกิดการเปล่งแสงเป็นสีแดง โดยอิเล็กตรอนกัน 2 อันที่เหลือจะถูกทำให้เกิดคัทออฟถ้าส่วนของภาพเป็นสีไซอันก็จะมีสัญญาณสีเขียวและสีน้ำเงินป้อนให้อิเล็กตรอนกันสีเขียวและสีน้ำเงินเพื่อทำให้เพิ่มปริมาณลำอิเล็กตรอนที่ยิงไปชนสารฟอสเฟอร์ที่หน้าจอทำให้เกิดการเปล่งแสงสีเขียวและสีน้ำเงินออกมาพร้อมๆ กันซึ่งทำให้เกิดผลรวมเป็นสีไซอัน โดยอิเล็กตรอนกันสีแดงจะทำให้คัทออฟในจังหวะนั้นสำหรับการสร้างภาพขาวดำนั้นหลอดภาพจะได้รับสัญญาณพร้อมกันทั้ง 3 สัญญาณทำให้อิเล็กตรอนกันทั้งสาม เปลี่ยนแปลงลำอิเล็กตรอนที่ยิงไปชนจออย่างเป็นสัดส่วนต่อกันจึงเกิดการสร้างส่วนของภาพที่เป็น สีขาว เทาอ่อน เทาแก่ตามลำดับ และดำ ที่หน้าจอเราเรียกสัญญาณขาวดำว่า สัญญาณลูมิแนนซ์ (Luminance) หรือเรียกย่อๆ ว่าสัญญาณวาย (Y) คือสัญญาณความสว่าง วิธีการสร้างสัญญาณ Y ก็คือการนำสัญญาณ R, G, B มารวมกันทางไฟฟ้าตามสัดส่วนรู้สึกสว่างเทียบกับแสงสีขาว (15) ดังสมการต่อไปนี้ $Y = 0.3R + 0.59GB + 0.11B$

สำหรับสีขาว มีส่วนผสมแม่สีคือ R =1, B = 1, G = 1 แทนในสมการจะได้

$Y = 0.3 + 0.59 + 0.11$

$Y = 1$ ได้สีขาวสว่าง 100%

สำหรับสีแดง มีส่วนผสมของแม่สีคือ $R = 1, G = 0, B = 0$ แทนในสมการจะได้

$Y = 0.3$ ได้สีแดงความสว่าง 30%

สำหรับสีเขียว มีส่วนผสมแม่สีคือ $R = 0, G = 1, B = 0$ แทนในสมการจะได้

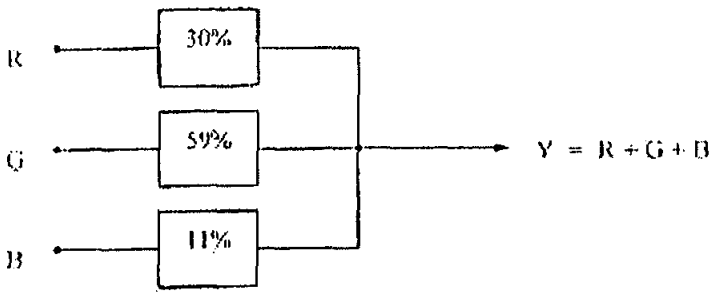
$Y = 0.59$ ได้สีเขียวความสว่าง 59%

สำหรับสีน้ำเงิน มีส่วนผสมแม่สีคือ R = 0, G = 0, B = 1 แทนในสมการจะได้

Y = 0.11 ได้สีน้ำเงินความสว่าง 11%

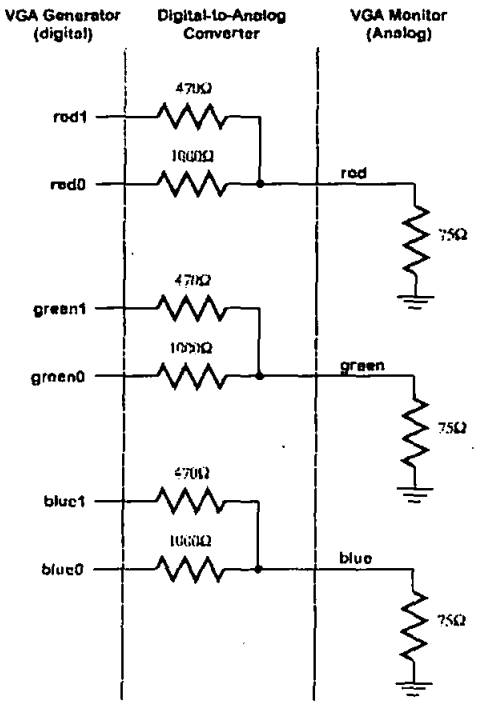
เราสามารถสร้างสัญญาณ Y ได้จากสัญญาณ R, G, B โดยใช้ตัวต้านทานลดระดับแรงดันของสัญญาณทั้ง 3 ให้ได้ขนาดความแรงของสัญญาณ เป็นสัดส่วนตามสมการที่กล่าวมาแล้วคือ

Y = 0.3R + 0.59G + 0.11B แล้วนำสัญญาณที่ได้มารวมกันดังรูป



รูปที่ 2.11 การสร้างสัญญาณลูมิแนนซ์

การสร้างสัญญาณสี 3 สี ประกอบด้วย สีแดง สีเขียว และสีน้ำเงิน โดยจะส่งข้อมูลสีไปยังหน้าจอมอนิเตอร์ โดยสัญญาณแต่ละสัญญาณจะนำไปขับปืนลำแสงอิเล็กตรอนโดยจะยิงออกไป และจะเป็นจุดสีที่หน้าจอมอนิเตอร์ ระดับสัญญาณ Analog มีค่าระหว่าง 0 ถึง 0.7 V ในการควบคุมเส้น ทำให้สีรวมกันเป็น Dot (หรือ Pixel) บนหน้าจอมอนิเตอร์ โดยสีที่เป็นอินพุท Analog สามารถตั้งค่าตั้งแต่ระดับ 1 ถึงระดับ 4 โดยค่าเอาท์พุทที่เป็น Digital 2 ค่า ดังรูปที่ 2.12 ค่าของ 4 ระดับที่อาจเป็นไปได้ในการรวมค่าอินพุทของ Analog โดยมอนิเตอร์สร้าง Pixel = 4x4x4 = 64 สีที่ต่างกัน ดังนั้นค่า Digital 6 ค่าจะสามารถควบคุมเส้นจากสีที่รวมกัน 64 สี

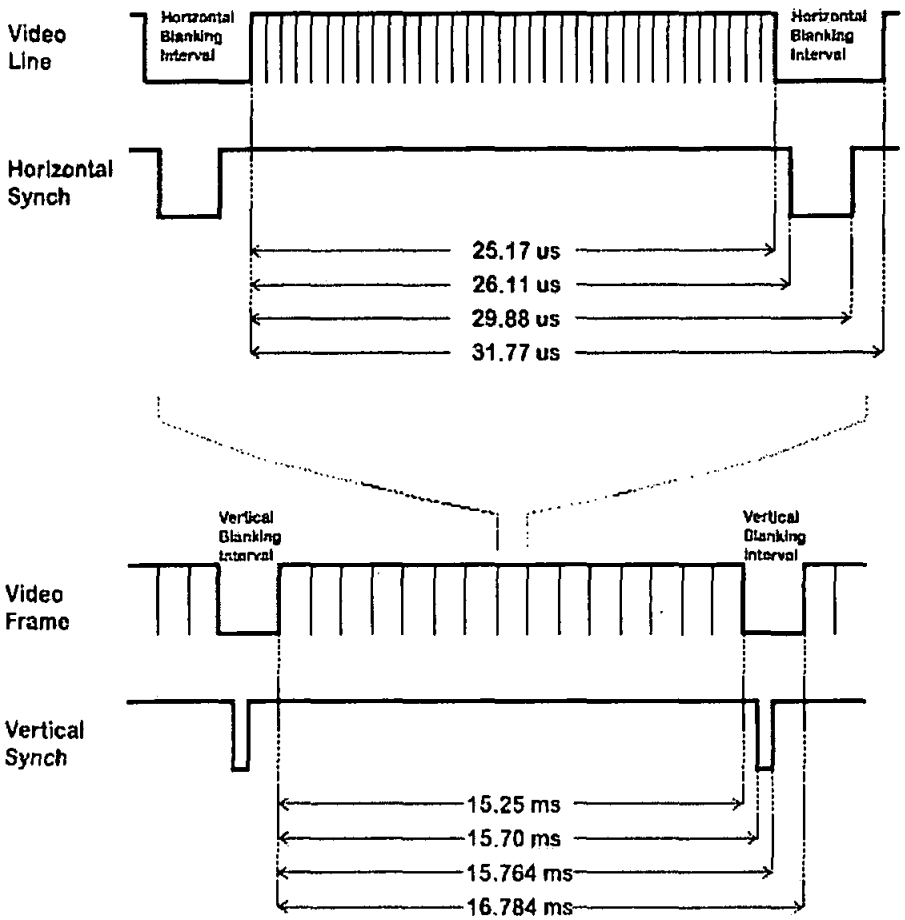


รูปที่ 2.12 แสดงการแปลงค่าเอาท์พุทแบบดิจิตอล ไปเป็นอนาลอก

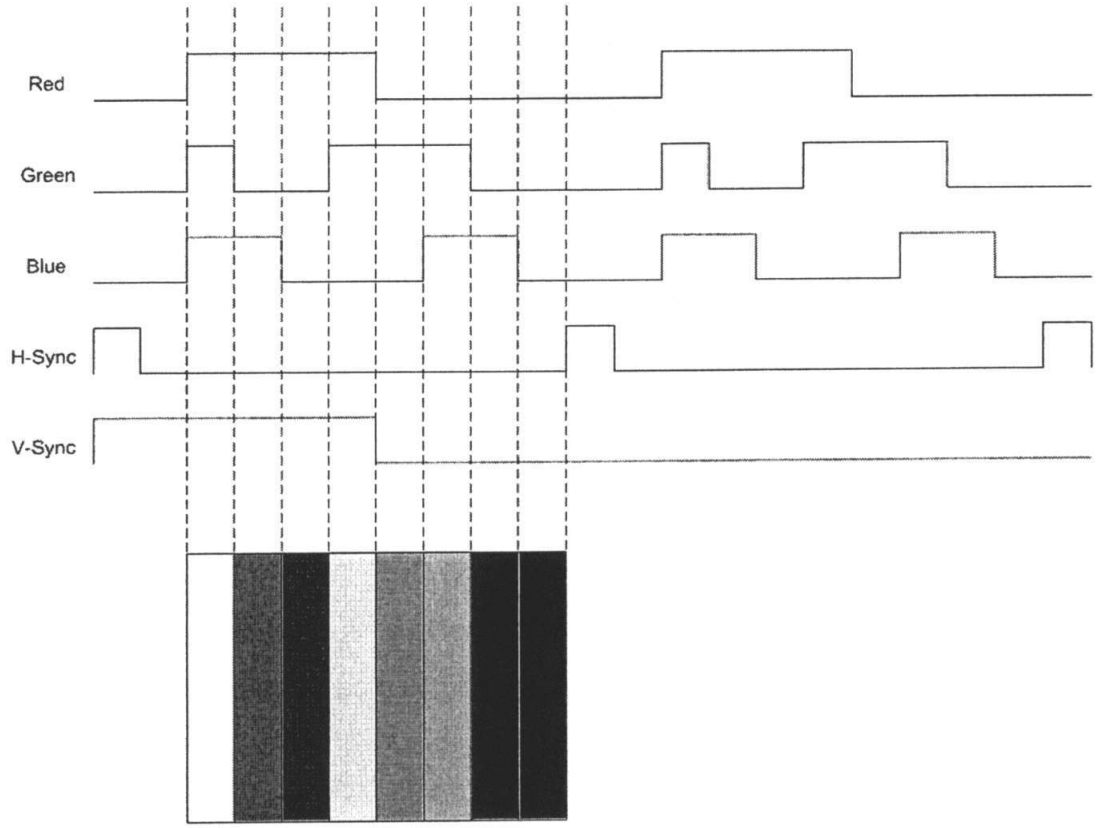
จุดสี (Dot) บนจอมอนิเตอร์ ไม่สามารถให้ข้อมูลมากๆ ได้และเส้นตามแนวนอนของพิกเซลให้ข้อมูลน้อยมากๆ แต่เส้นขอบหลายเส้นสามารถแสดงภาพบนจอมอนิเตอร์ได้ เส้นขอบของสัญญาณ VGA จะมีเส้น 480 เส้น แต่ละเส้นจะมี 640 พิกเซล โดยวงจรในมอนิเตอร์จะมีปืนยิงลำแสงอิเล็กตรอนเพื่อจะส่งอิเล็กตรอนจากไปยังจอมอนิเตอร์ ปืนคู่นี้จะยิงลำแสงจากซ้ายไปขวาและบนลงล่าง ตัดผ่านบนจอภาพจะมีสัญญาณตรงกัน (synchronization) ในคำสั่ง เริ่มและหยุดที่เวลาจริง ดังนั้นเส้นของพิกเซลที่ตัดกันบนจอมอนิเตอร์และเส้นจากบนลงล่างจากรูปภาพ จะมีสัญญาณเวลาตรงกันดังรูปที่ 2.13

สัญญาณแนวนอนที่ Synchronization จะเริ่มต้นที่เส้นและพิกเซลของจอมอนิเตอร์ระหว่างซ้ายและขวา พิกเซลที่ส่งไปยังจอมอนิเตอร์ใช้เวลาไม่เกิน 25.17 us สัญญาณในการ Synchronization ที่ลดลงต่ำสุดประมาณ 0.94 us หลังจากพิกเซลสุดท้าย และยังคงต่ำที่ 3.77 us เส้นพิกเซลใหม่สามารถเริ่มที่ค่าต่ำๆ คือ 1.89 us หลังจากเส้นแนวนอนหยุดนิ่ง ดังนั้นเส้นสัญญาณจะอยู่ระหว่างค่า 25.17 us ถึง 31.77 us
สัญญาณแนวตั้งที่ Synchronization จะเริ่มต้นที่เส้นและพิกเซลของจอมอนิเตอร์ระหว่างบนและล่างของมอนิเตอร์ เส้นจะส่งสัญญาณไปยังจอมอนิเตอร์ด้วยเวลา 15.25 ms

สัญญาณในการ Synchronization ที่ลดลงต่ำสุดประมาณ 0.45 ms หลังจากพิกเซลสุดท้ายและยังคงต่ำที่ 64 us เส้นแรกที่ถัดจากขอบเริ่มต้นด้วยค่าน้อย 1.02 ms หลังจากเส้นแนวตั้ง sync สัญญาณจากขอบมีค่าระหว่าง 15.25 ms ถึง 16.784 ms



รูปที่ 2.13 แสดงสัญญาณเวลาที่ synchronous กันของเส้นแนวนอนและแนวตั้ง

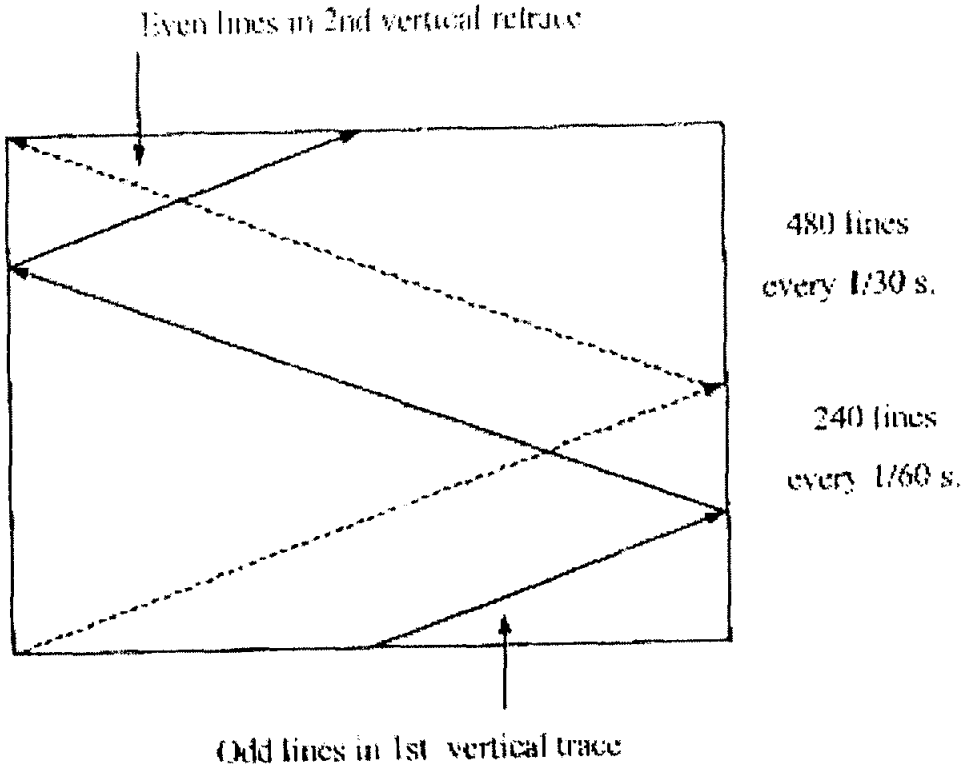


รูปที่ 2.14 ตัวอย่างการสแกนภาพ

2.6.2 การสแกน

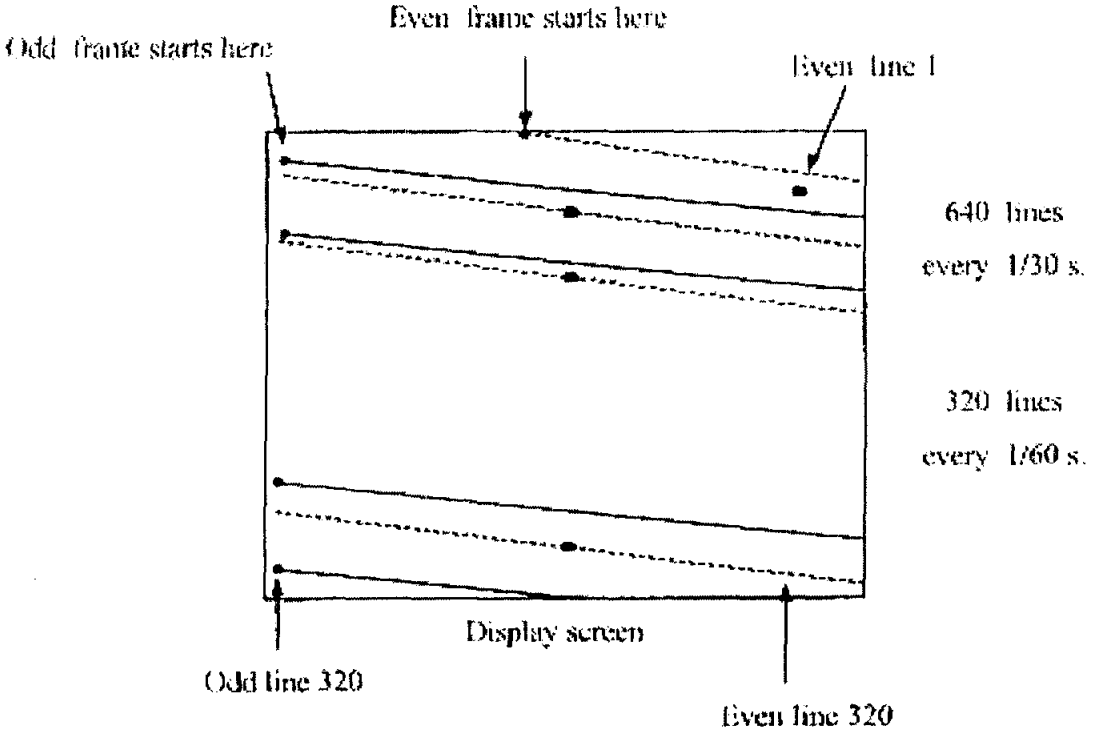
การสแกนคือการนำเอาสัญญาณภาพที่อยู่ในรูปของของสัญญาณไฟฟ้านำมาเรียงกันให้เกิดเป็นภาพ โดยการกวาดเป็นเส้นภาพที่หน้าจอโดยตัวที่มีหน้าที่สำคัญคือหลอดภาพ หลอดภาพมีโครงสร้างคล้ายกับกับหลอดสุญญากาศทั่ว ๆ ไปที่ปล่อยอิเล็กตรอนมาจาก Cathode แล้วมีการดึงลำอิเล็กตรอนให้วิ่งเป็นลำกระทบเข้ากับหน้าจอหรือ Anode การสแกนมี 2 วิธีคือ การสแกนแบบเดินหน้า (Progressive Scanning) กับการสแกนแบบสลับเส้น (Interlaced Scanning)

การที่จะทำให้การสแกนมีความต่อเนื่องขององค์ประกอบภาพต้องคำนึงถึงหลัก 3 ประการ คือ ลำอิเล็กตรอนที่กวาดไปทางแนวนอน (Horizontal Scanning) ในแต่ละครั้งจะต้องครอบคลุมองค์ประกอบภาพทั้งหมดของเส้นนั้น ในแต่ละเส้นของการสแกนลำอิเล็กตรอน ลำแสงต้องกวาดกลับด้วยความเร็วสูงไปยังด้านซ้ายเพื่อเริ่มการสแกนในลำดับต่อไป เวลาของการสะบัดกลับเราเรียกว่า Retrace หรือ Fly back ในกรณีดังกล่าวจะต้องไม่มีข้อมูลภาพใดๆ เพราะหลอดภาพจะเกิดการ Blank out ในขณะนั้น ในขณะที่เส้นสแกนสะบัดกลับมาเพื่อเริ่มต้นทางซ้ายใหม่ตำแหน่งทางแนวตั้งต้องต่ำกว่าตำแหน่งเดิมเพื่อทำให้การสแกนเส้นต่อไปไม่ทับกันทั้งนี้โดยการควบคุมทางแนวตั้ง (Vertical Scanning)



รูปที่ 2.15 การสแกนทางแนวนอน

การสแกนที่ใช้ในจอมอนิเตอร์ต้องใช้การเรียงภาพ 30 ถึง 60 ภาพต่อวินาทีจึงจะเกิดเป็นภาพที่ต่อเนื่องแต่ก็ยังมีการกระพริบ (Flicker) เนื่องจากว่าการสแกนเริ่มจากขอบบนลงมาด้านล่างแสงทางด้านบนเริ่มมืดลงกว่าด้านล่างจึงมองเห็นว่ามันกระพริบ และเวลาที่ลำแสงการสแกนวกกลับไปด้านบนด้านล่างก็เกิดปัญหาเช่นเดียวกันความรู้สึกต่อกรณีนี้ก็คือ เกิดแสงกระพริบหรือวูบวาบขึ้นซึ่งจะเกิดขึ้น ในการสแกนแบบเดินหน้า (Progressive Scanning) ซึ่งเป็นการสแกนพื้นฐาน เพื่อแก้ปัญหาการกระพริบจึงต้องใช้การสแกนสลับเส้นหรือการสแกนแบบสอดแทรก (Interlaced Scanning)โดยครั้งแรกจะสแกนที่ฟิลด์คี่ (Odd line trace) และครั้งต่อไปจะสแกนแบบฟิลด์คู่ (Even line trace) เป็นการสแกนแบบเส้นเว้นเส้น หมายความว่าการที่จะได้ภาพ 1 ภาพหรือ 1 เฟรมต้องใช้การสแกนแนวตั้ง 2 ครั้งหรือ 2 ฟิลด์ (Field) มาตรฐานของจอ VGA จะใช้เส้นสแกน320เส้นทางแนวนอนและ 240 เส้นทางแนวตั้งเริ่มต้นการสแกนสมมุติว่าจากเส้นสแกนคี่ทางแนวนอนโดยเริ่มจากทางซ้ายแล้วกวาดไปทางขวานับเป็นเส้น สแกนที่ 1 แล้วจึงสแกนเส้นที่ 3, 5, 7, 9 และต่อๆ ไปจนได้เส้นสแกนครบ 320 เส้นดังแสดงรูปด้านล่าง (ในส่วนที่เป็นเส้นทึบ)

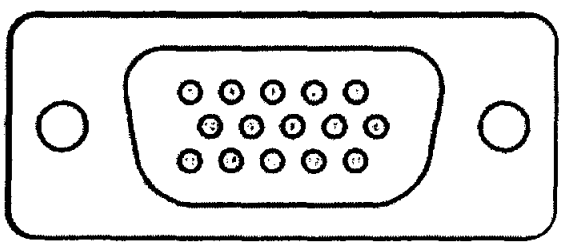


รูปที่ 2.16 การสแกนทางแนวตั้ง

ที่จุดสิ้นสุดการสแกนเส้นคี่ทางแนวนอนนี้จะเป็นจุดเริ่มต้นการสแกนทางแนวตั้งของฟิลด์คี่ซึ่งเราเรียกว่าการสแกนทางแนวตั้งว่า Vertical Retrace หรือ Fly back เพื่อดึงกลับไปยังด้านบนของจอภาพในตำแหน่งนี้เพื่อให้เริ่มต้นการสแกนเส้นคู่ต่อไป ที่กล่าวมาทั้งหมดคือการสแกนในหนึ่งฟิลด์ Retrace Times ทั้งทางแนวนอนและแนวตั้งเป็นเวลาสั้นๆ ถึงอย่างไรก็ตามเราไม่ต้องการให้เส้นสแกนของช่วงที่เป็นการสะบัดกลับเข้ามารบกวนให้เกิดภาพในส่วนนี้จึงต้องทำการลบเส้นสะบัดกลับ Retrace Times จะใช้เวลาประมาณ 10 – 16 เปอร์เซ็นต์ของเวลาทั้งหมดในการสแกน การสแกนของเส้นคู่ก็เป็นในทำนองเดียวกันต่างกันที่จุดเริ่มต้นเท่านั้น โดยจะเริ่มที่จุดสุดท้ายของขั้นตอนก่อนหน้านี้ และก็จะสลับกันเป็นเช่นนี้เรื่อยไป

2.6.3 คอนเนคเตอร์สำหรับ VGA

คอนเนคเตอรสำหรับ VGA เป็นชนิด DB-15 มีขา 3 แถว หรืออาจเรียกว่า D-Sub 15 ใช้กับจอมอนิเตอร์ทั่วไป ซึ่งแสดงแบบขาดังรูป



รูปที่ 2.17 แสดงขาตัวเมียของพอร์ต DB-15

ตารางที่2.9 แสดงตำแหน่งของขา DB-15

ตำแหน่งขา	สัญญาณ	
Pin 1	RED	Red video
Pin 2	GREEN	Green video
Pin 3	BLUE	Blue video
Pin 4	N/C	Not connected
Pin 5	GND	Ground (HSync)
Pin 6	RED_RTN	Red return
Pin 7	GREEN_RTN	Green return
Pin 8	BLUE_RTN	Blue return
Pin 9	+5 V	+5 V (DDC)
Pin 10	GND	Ground (VSync, DDC)
Pin 11	N/C	Not connected
Pin 12	SDA	I^2C data
Pin 13	HSync	Horizontal sync
Pin 14	VSync	Vertical sync
Pin 15	SCL	I^2C clock

บทที่ 3

การออกแบบ

3.1 การติดต่อระหว่างไมโครคอนโทรลเลอร์กับ SD Card

ทางผู้จัดทำได้ใช้วิธีการให้ไมโครคอนโทรลเลอร์ส่งข้อมูลให้กับ SD Card โดยใช้ Port SPI โดยจะให้ไมโครคอนโทรลเลอร์ส่งคำสั่งอ่านข้อมูล 1 บล็อก จาก SD Card ผ่านทาง Port SPI จากนั้นข้อมูลที่ได้จาก SD Card จะถูกส่งกลับมายังไมโครคอนโทรลเลอร์ ซึ่งไมโครคอนโทรลเลอร์จะทำการถอดรหัสไฟล์ภาพ แล้วให้กำเนิดสัญญาณภาพส่งออกทาง Port D-Sub

3.2 วงจรรวมของอุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์

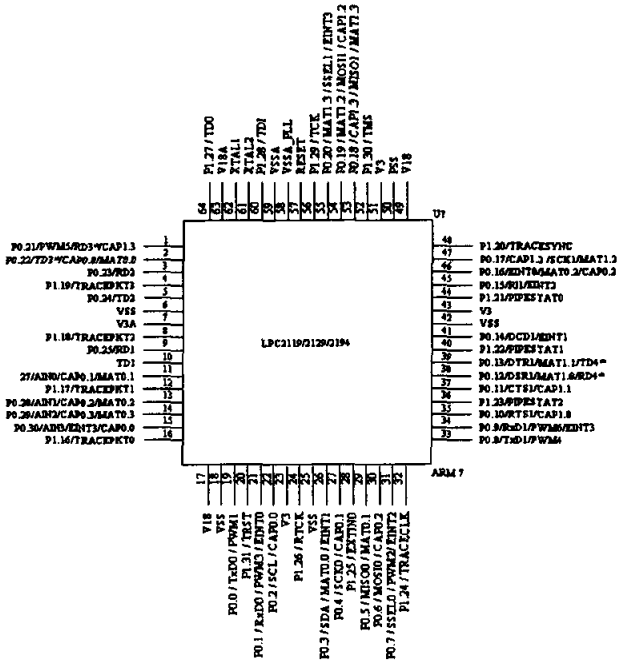
อุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์ แบ่งอุปกรณ์เป็น 2 ส่วน ดังนี้

3.2.1 ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ที่ใช้เป็นไมโครคอนโทรลเลอร์ตระกูล ARM7 เบอร์ LPC2119 เนื่องจากมีความเร็วในการประมวลผลทางสัญญาณเร็ว มีการรองรับการติดต่อในระบบต่างๆ มากมาย ซึ่งได้เลือกใช้ระบบ SPI Bus ซึ่งมีขนาดของหน่วยความจำที่เพียงพอ ไมโครคอนโทรลเลอร์ตัวนี้จะทำงานที่ 16/32 bits และศึกษาถึงโครงสร้างต่างๆ ในการใช้ไมโครคอนโทรลเลอร์ตัวนี้ ซึ่งเป็นที่นิยมใช้งานในการประมวลสัญญาณที่มีความเร็วสูง

ในส่วนการติดต่อกับ SD/MMC มี 4 ขา คือ SSEL, SCK, SDO, SDI

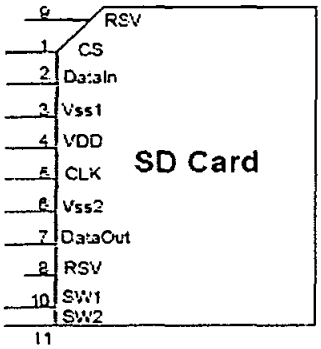
ในส่วนของการสร้างสัญญาณภาพ จะสร้างสัญญาณภาพออกเป็นสัญญาณอนาลอก ออกทาง Port D-Sub



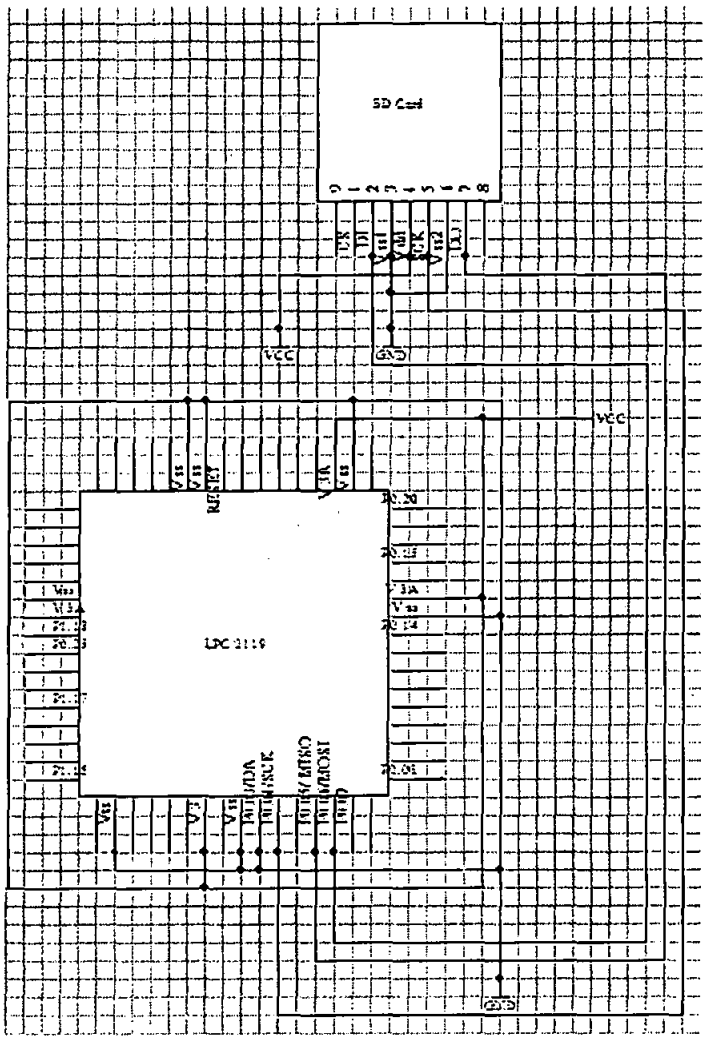
รูปที่ 3.1 ไมโครคอนโทรลเลอร์ตระกูล ARM7 เบอร์ LPC2119

3.2.2 SD Card

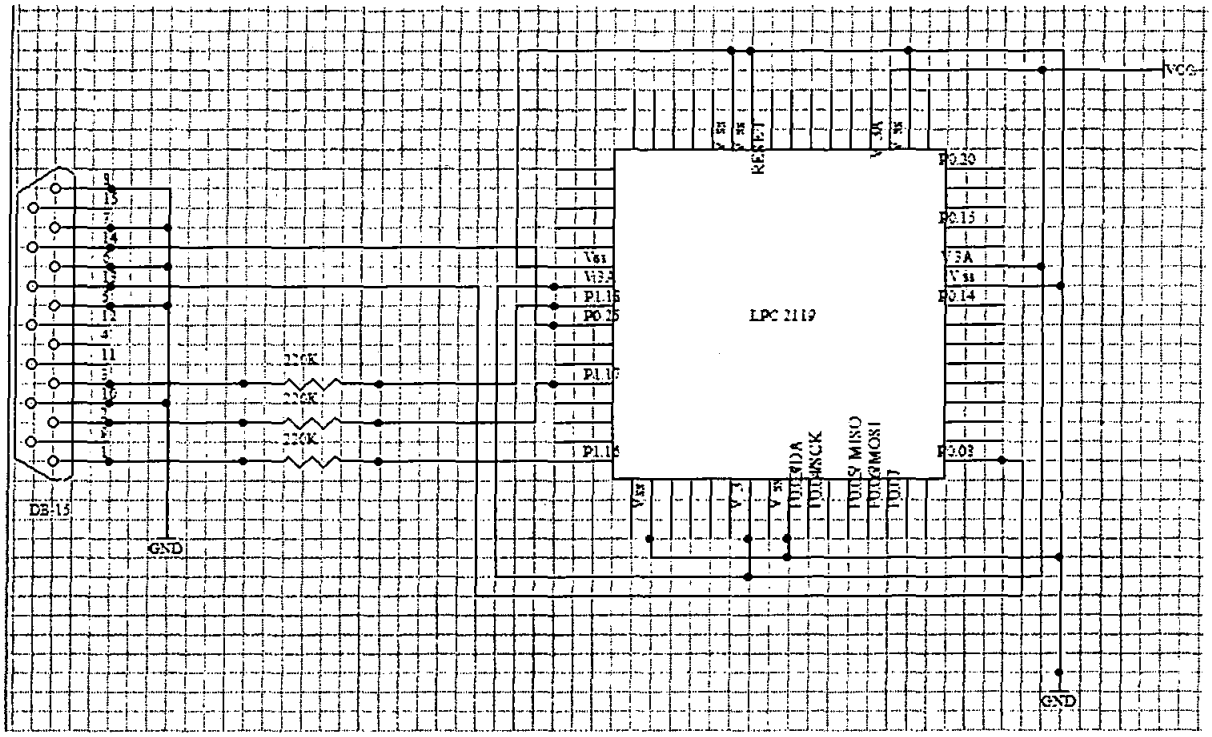
ผู้จัดทำได้เลือกใช้ SD Card ในการเก็บข้อมูลไฟล์ภาพ เนื่องจากเห็นว่ามีความเหมาะสมในเรื่องของราคาที่ไม่ค่อยสูงมาก และเป็น Card ที่หาซื้อได้ง่าย



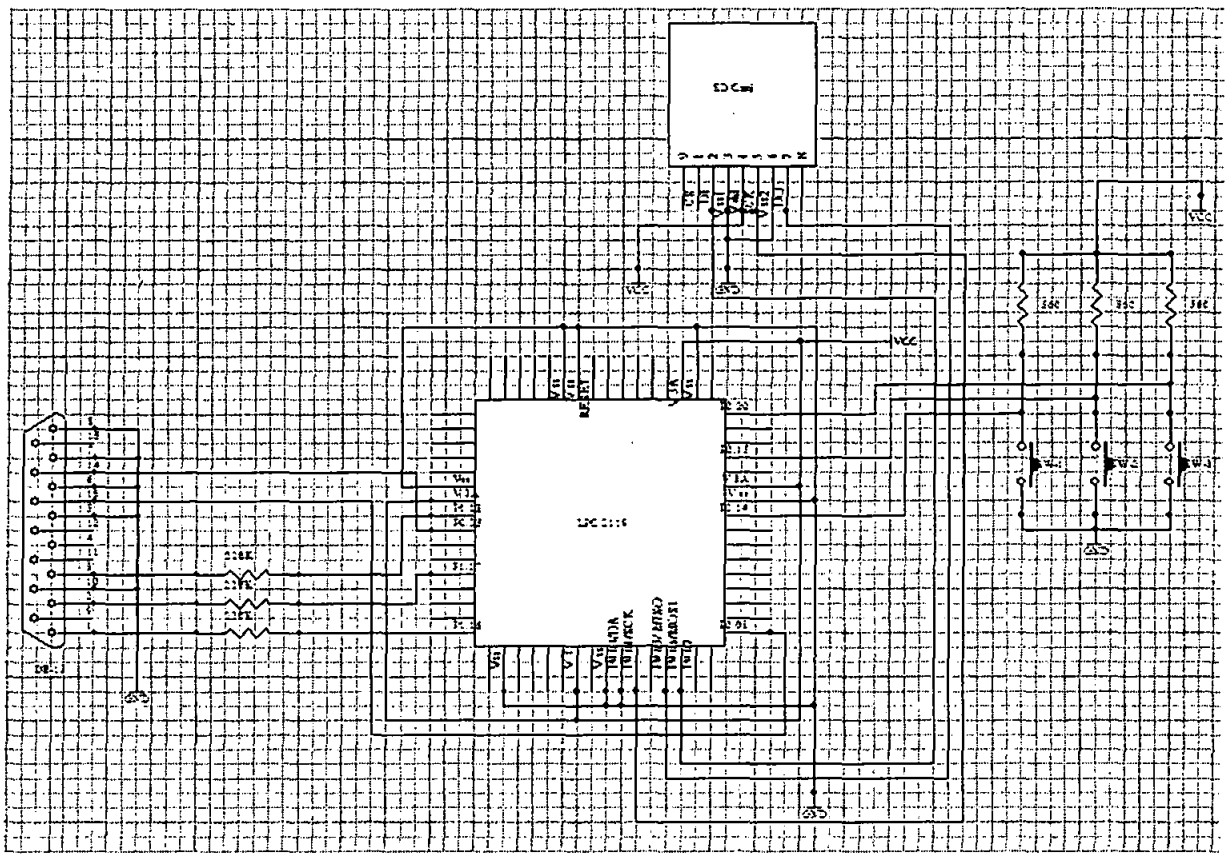
รูปที่ 3.2 SD Card



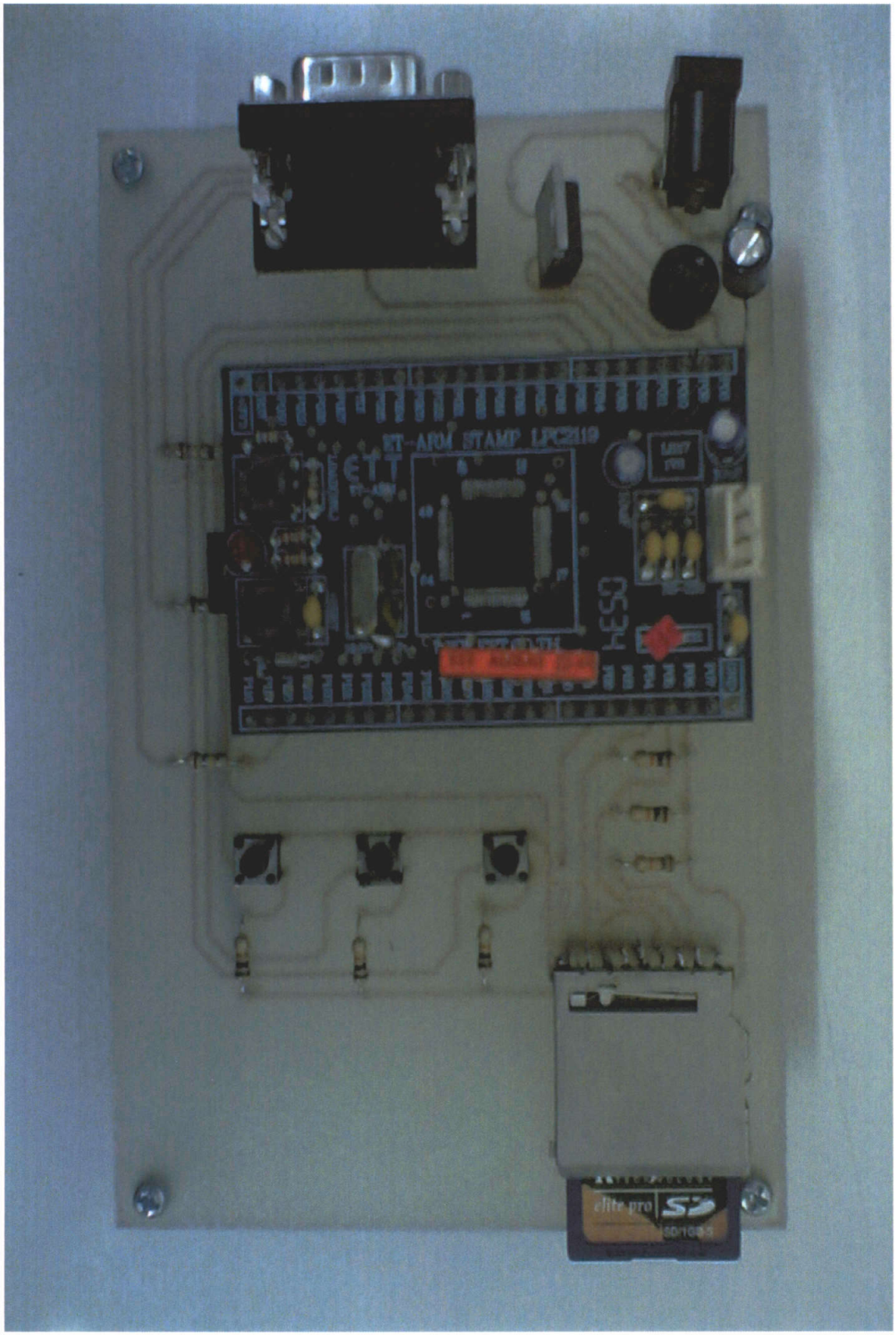
รูปที่ 3.3 วงจรการติดต่อระหว่าง SD Card กับ ARM7 เบอร์ LPC2119



รูปที่ 3.4 วงจรการติดต่อระหว่าง DB-15 กับ ARM7 เบอร์ LPC2119

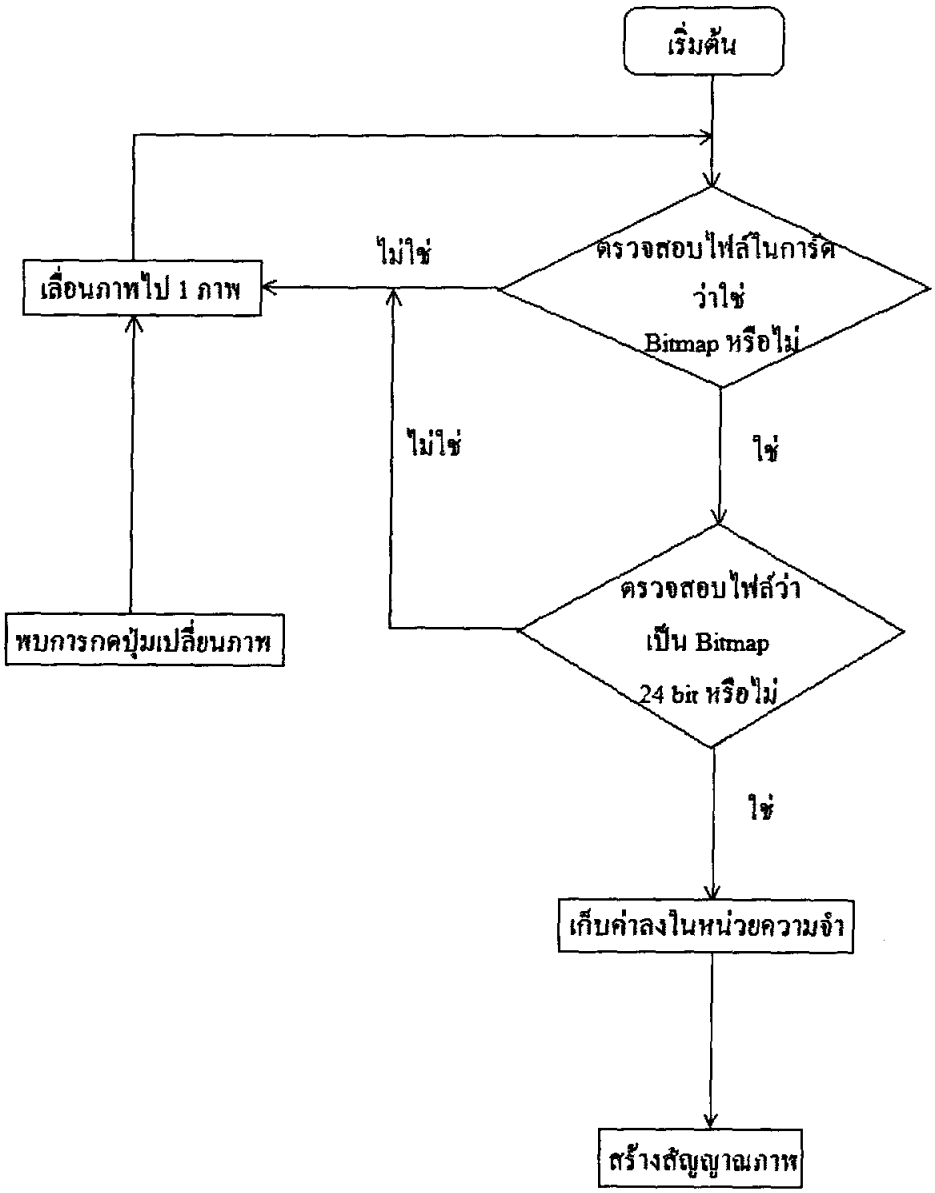


รูปที่ 3.5 วงจรการติดต่อ SD Card, DB-15 และ ARM7 เบอร์ LPC2119



รูปที่ 3.6 อุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์

3.3 ผังการทำงานของอุปกรณ์แสดงภาพทางจอมอนิเตอร์



รูปที่ 3.7 ผังการทำงานของอุปกรณ์แสดงภาพทางจอมอนิเตอร์

3.4 การคำนวณ Resolution

การคำนวณหา Resolution ของหน้าจอแสดงผลขนาดมาตรฐาน 640x480 pixel จะใช้เวลาในการสแกนภาพประมาณ 40 us แต่เวลาที่คอนโทรลเลอร์สามารถทำได้คือ 400 us ซึ่งสามารถแสดงการคำนวณได้ดังนี้

จากความละเอียดมาตรฐานของ VGA คือ 640x480 pixel 60Hz เพราะฉะนั้นจะได้ว่า

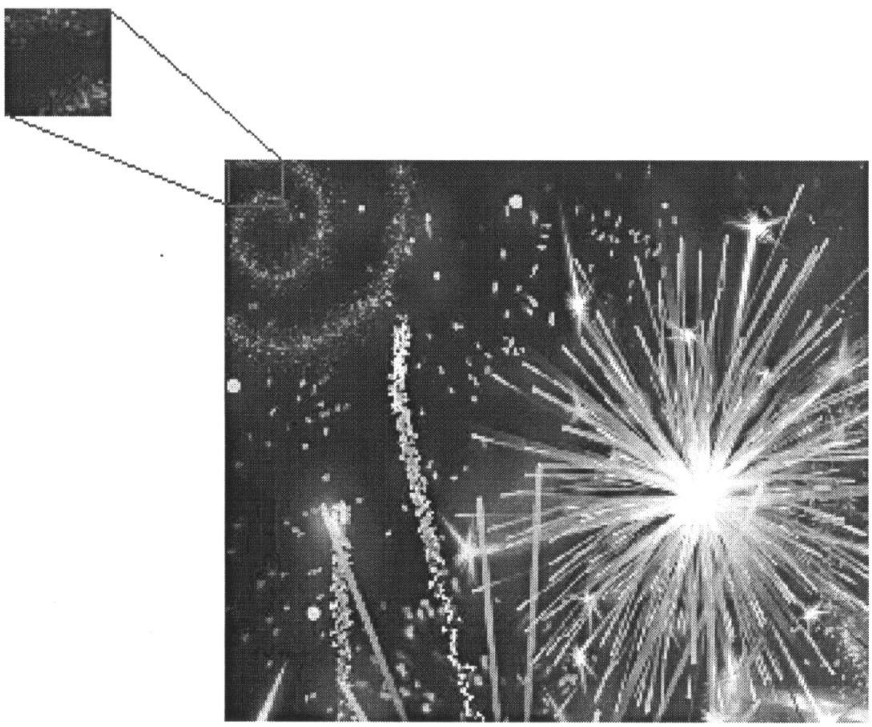
คาบเวลา = $\frac{1}{60}$ = 16.7 ms เพราะฉะนั้น 1 เส้นใช้เวลา = $\frac{16.7\ ms}{480}$ = 34.7 us

ซึ่งจะได้เป็นความถี่ Horizontal Sync = $\frac{1}{34.7\ \text{us}}$ = 29 kHz

เวลาที่สามารถส่งสัญญาณสีออกไปได้เนื่องจากมี front pouch และ back pouch = 34.7 us – 2.5 us = 32.2 us

เพราะฉะนั้น 1 pixel ใช้เวลาในการส่งสัญญาณสี = $\frac{32.2\ us}{640}$ = 50 ns

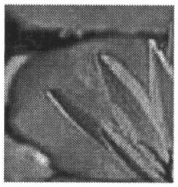
แต่ไมโครสามารถส่งสัญญาณสีได้เร็วสุดแค่ 900 *ns* เพราะฉะนั้นสามารถทำความละเอียดได้สูงสุดได้ = $\frac{32.2\ us}{900\ ns}$ = 35.7 pixel

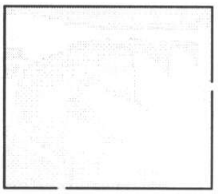


รูปที่ 3.8 แสดงขนาด 40 X40 pixel จากภาพขนาด 640x480 pixels

ไฟล์บิตแมพ 24 Bitจะจัดเรียงแบบ 2 byteโดย2 byte แรกเป็นสีแดง สีเขียว สีน้ำเงินน้ำเงิน

00 00 00 โดย จะแบ่งพิจารณาเป็นสีๆ ไป ถ้าสีไหนมีค่ามากกว่า 0x0F จะตัดให้มีค่าเป็น 1 แต่ถ้าน้อยกว่าจะให้ค่า เป็น 0 แสดงได้ดั้งภาพ





รูปที่ 3.9 เปรียบเทียบระหว่าง 24 bit กับภาพที่แปลงเหลือ 1 bit

บทที่ 4

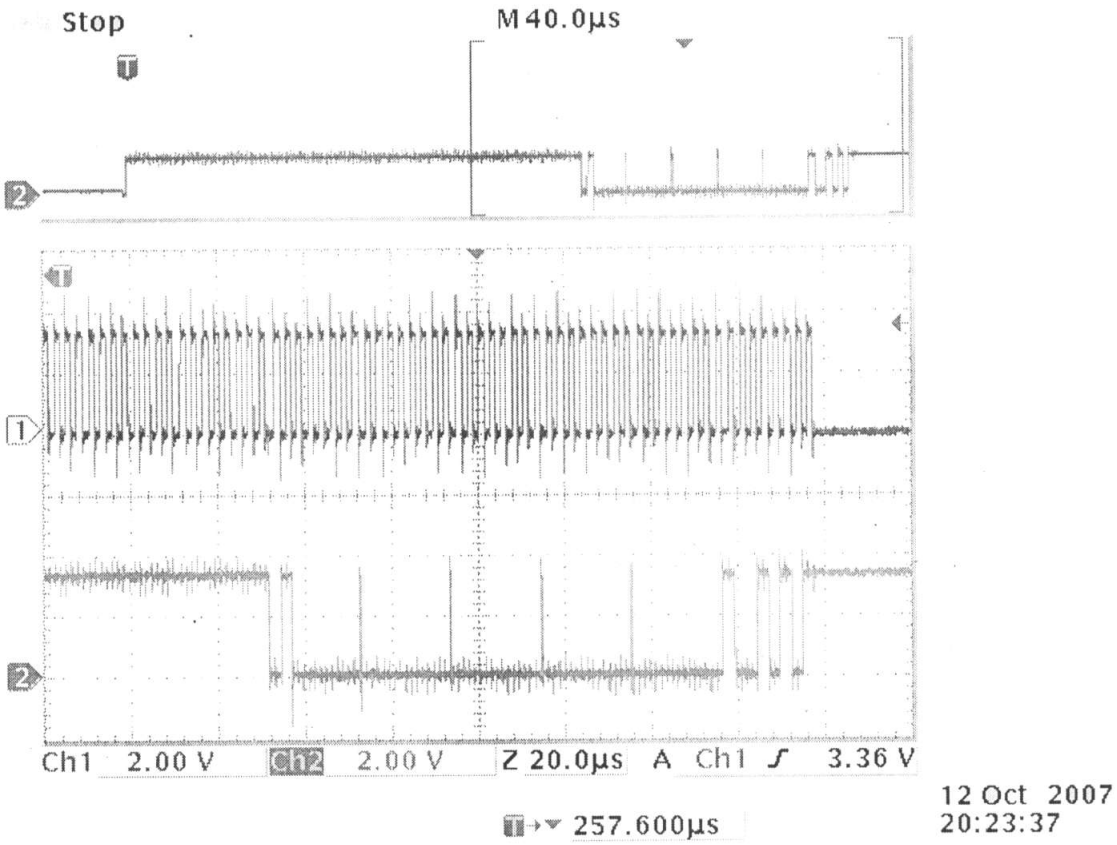
การทดลองและผลการทดลอง

การทดลองแบ่งออกเป็น 2 การทดลอง ดังนี้

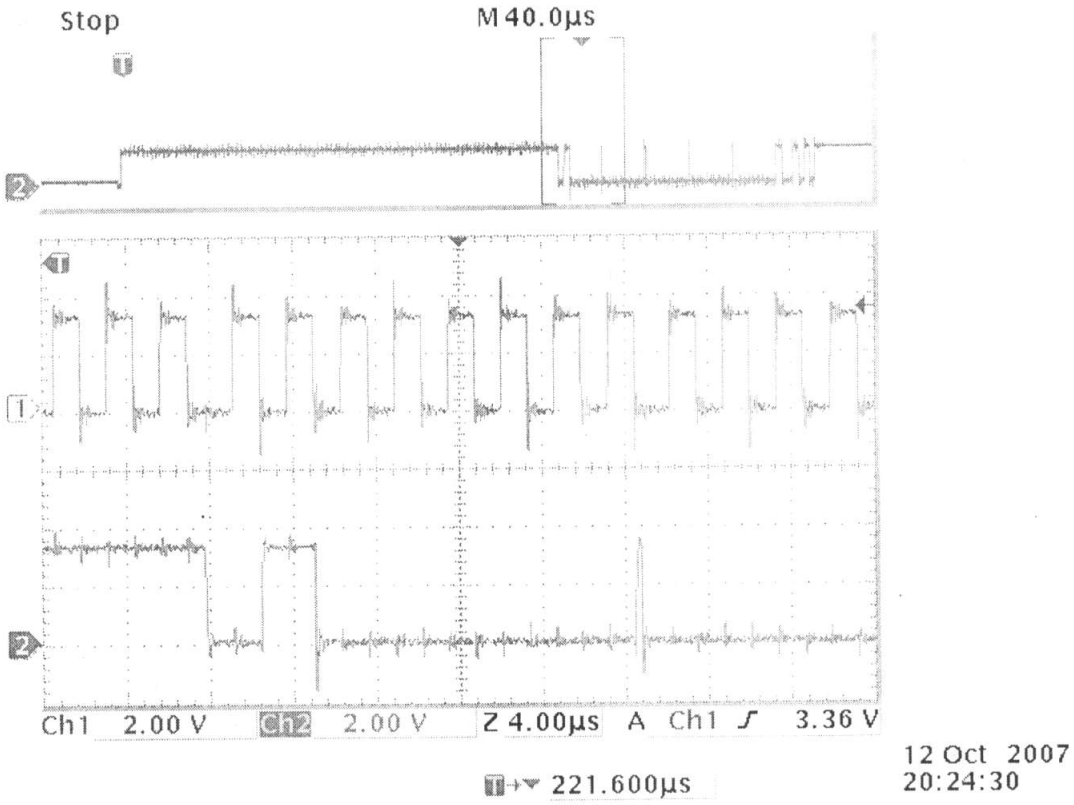
การทดลองที่ 1 วัดสัญญาณของการติดต่อแบบอนุกรมในลักษณะ SPI Bus

การทดลองที่ 2 วัดสัญญาณของหน้าจอมอนิเตอร์

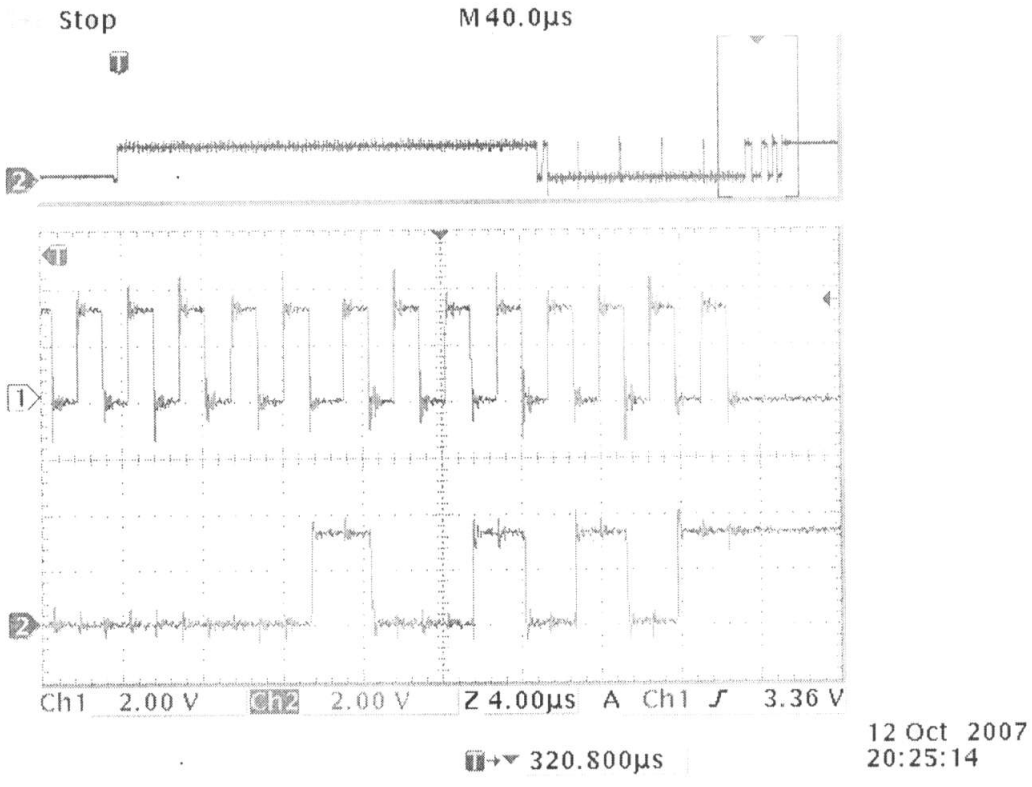
การทดลองที่ 1 วัดสัญญาณของการติดต่อแบบอนุกรมในลักษณะ SPI Bus



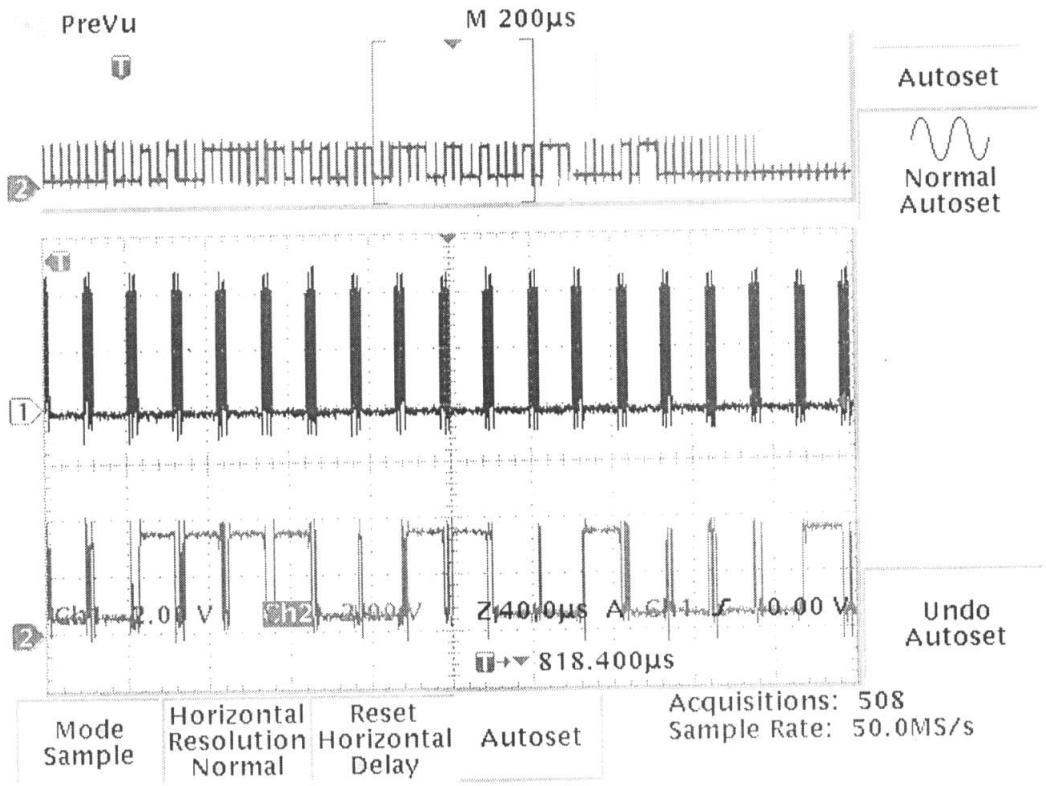
รูปที่ 4.1 รูปสัญญาณของชุดคำสั่งที่ส่งมาจากไมโครคอนโทรลเลอร์



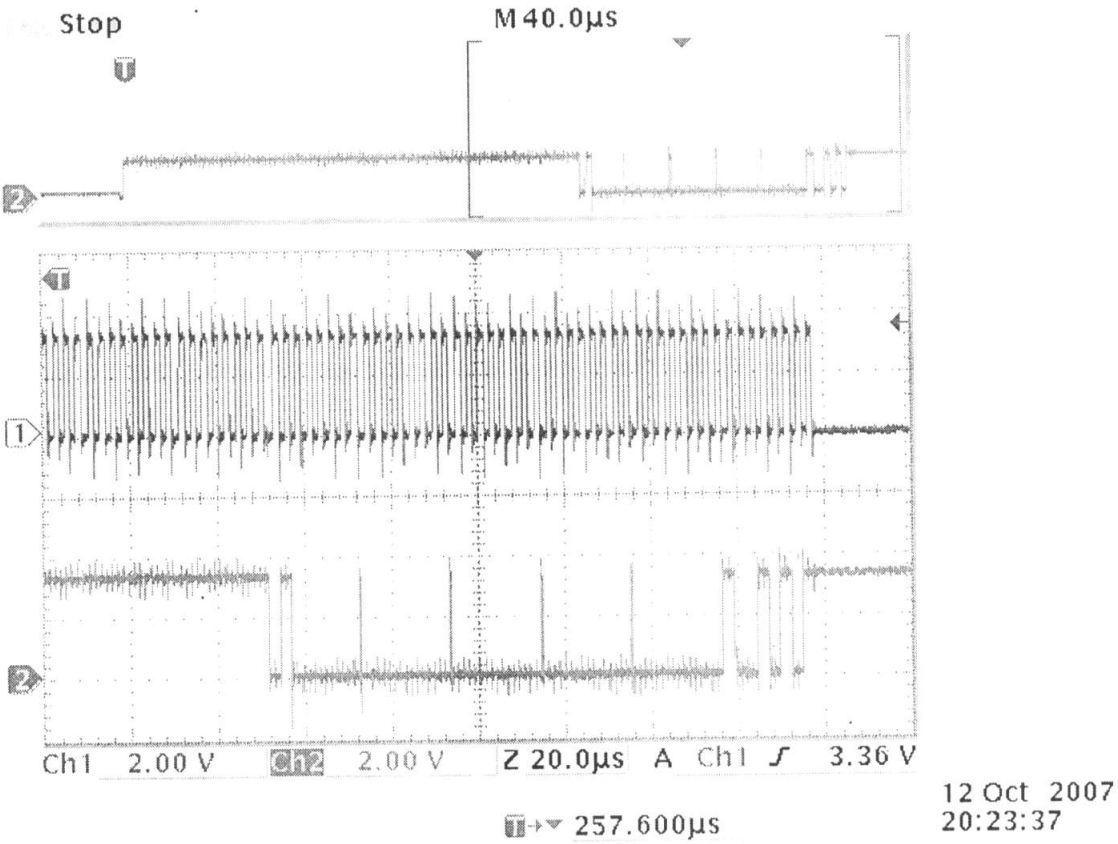
รูปที่ 4.2 รูปสัญญาณคำสั่ง RESET (0x40)



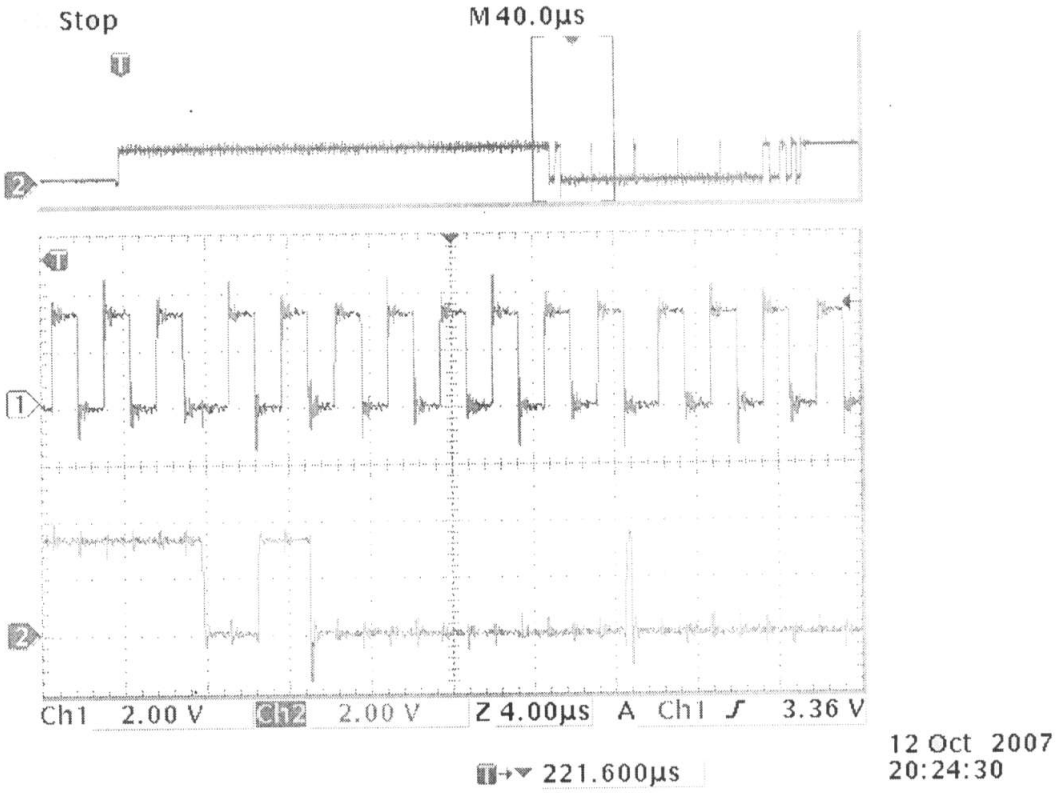
รูปที่ 4.3 รูปสัญญาณคำสั่งRESET (0x95)



รูปที่ 4.4 รูปสัญญาณของชุดข้อมูลที่ได้รับจากการ์ด

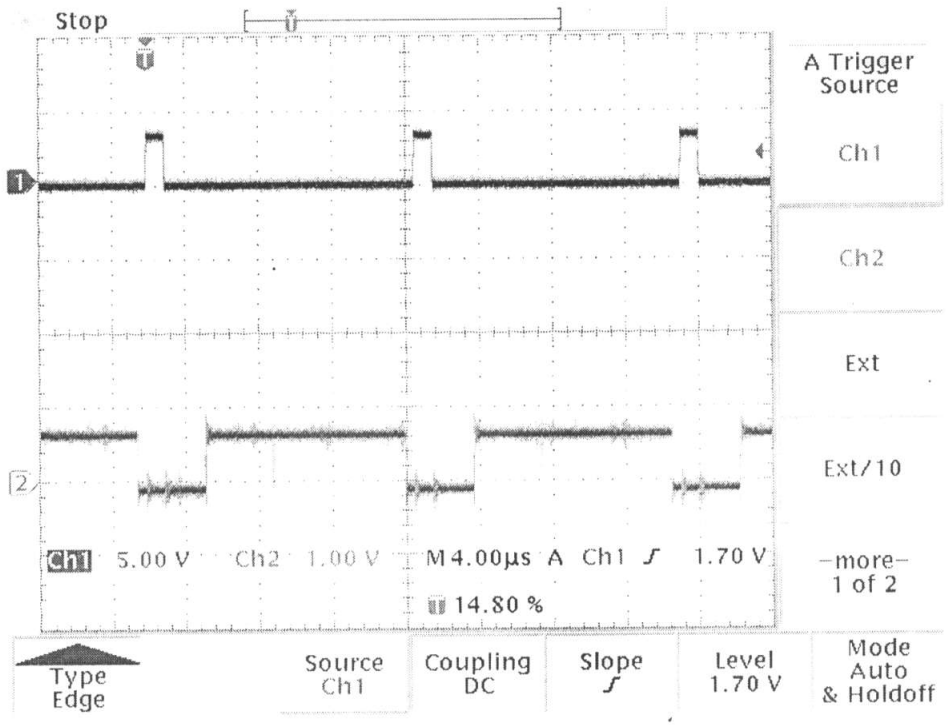


รูปที่ 4.5 รูปสัญญาณ SPI

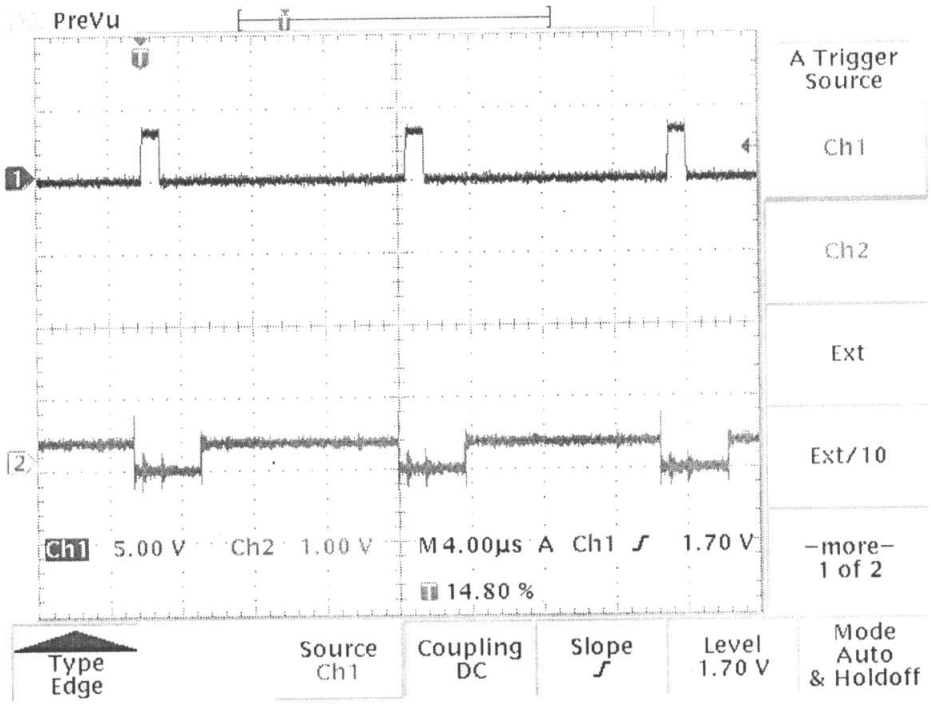


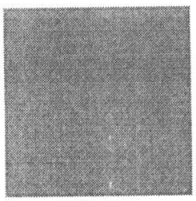
รูปที่ 4.6 รูปสัญญาณ SPI (ส่วนขยาย)

การทดลองที่ 2 วัดสัญญาณของหน้าจอมอนิเตอร์

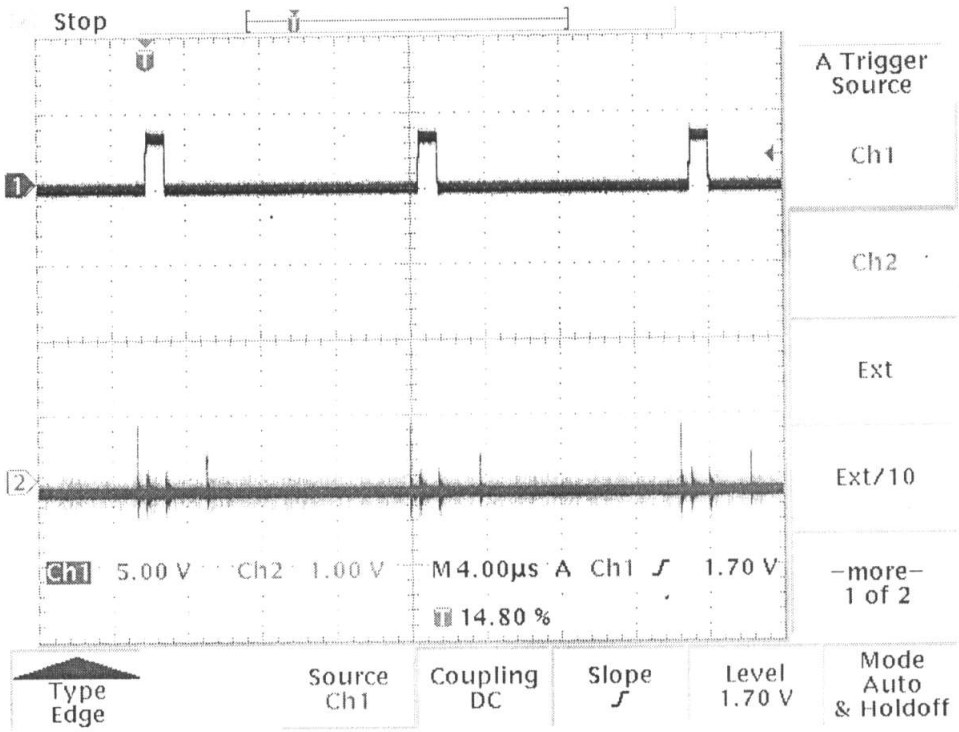


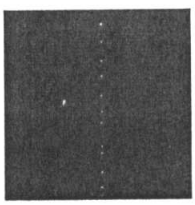
รูปที่ 4.7 รูปสัญญาณสีแดง สีน้ำเงิน สีเขียว (สีขาว)



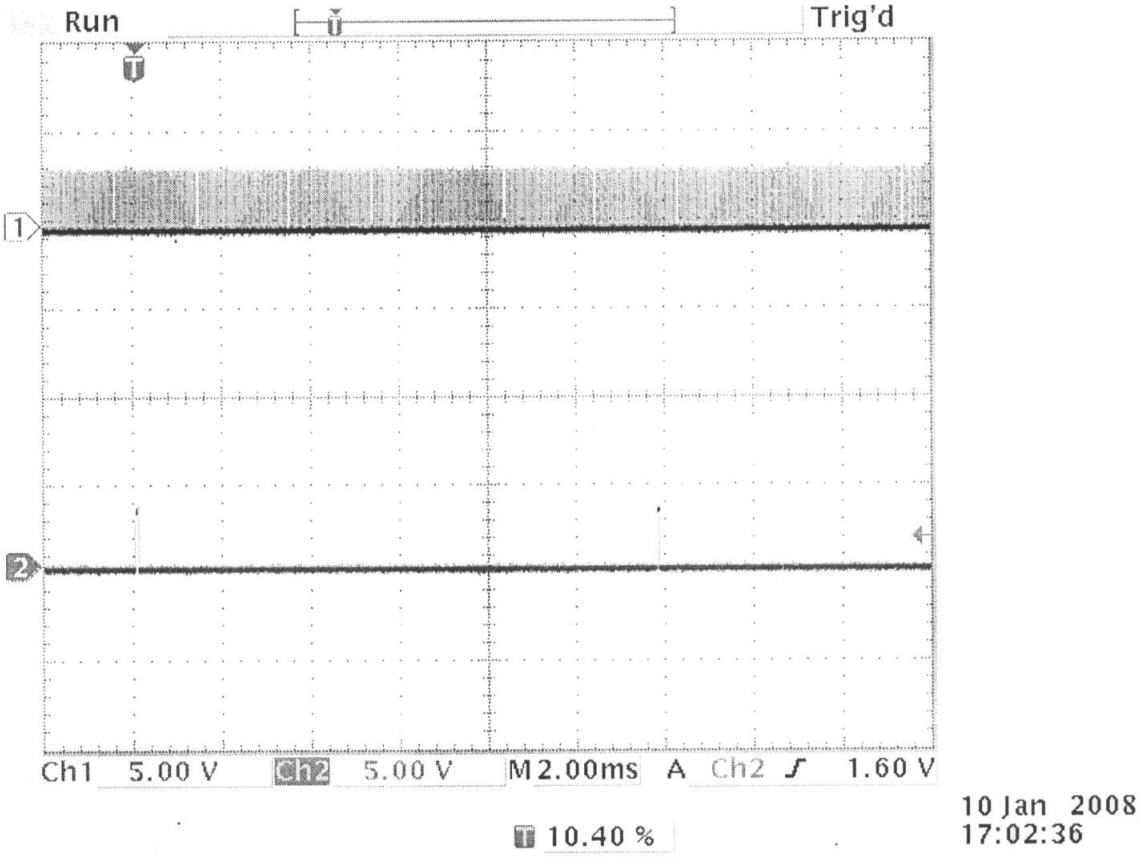


รูปที่ 4.8 รูปสัญญาณของสีผสมที่อ่อนลงมา

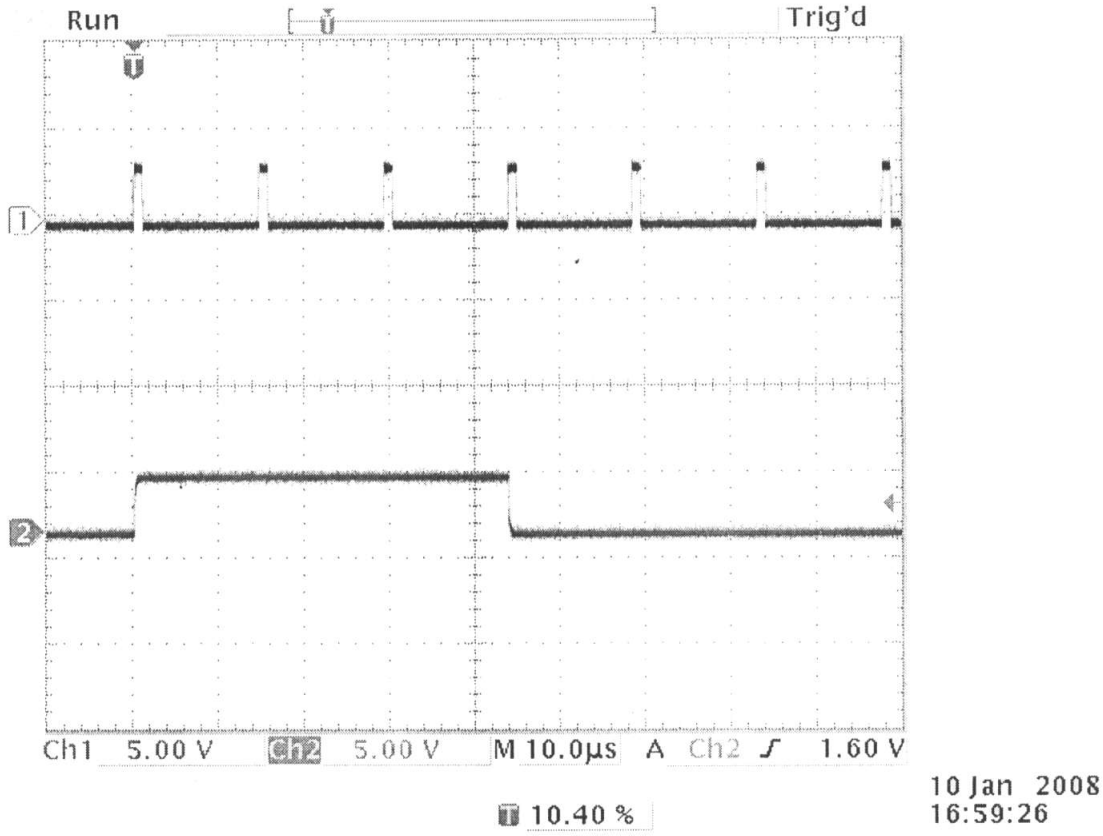




รูปที่ 4.9 รูปสัญญาณของสีดำ

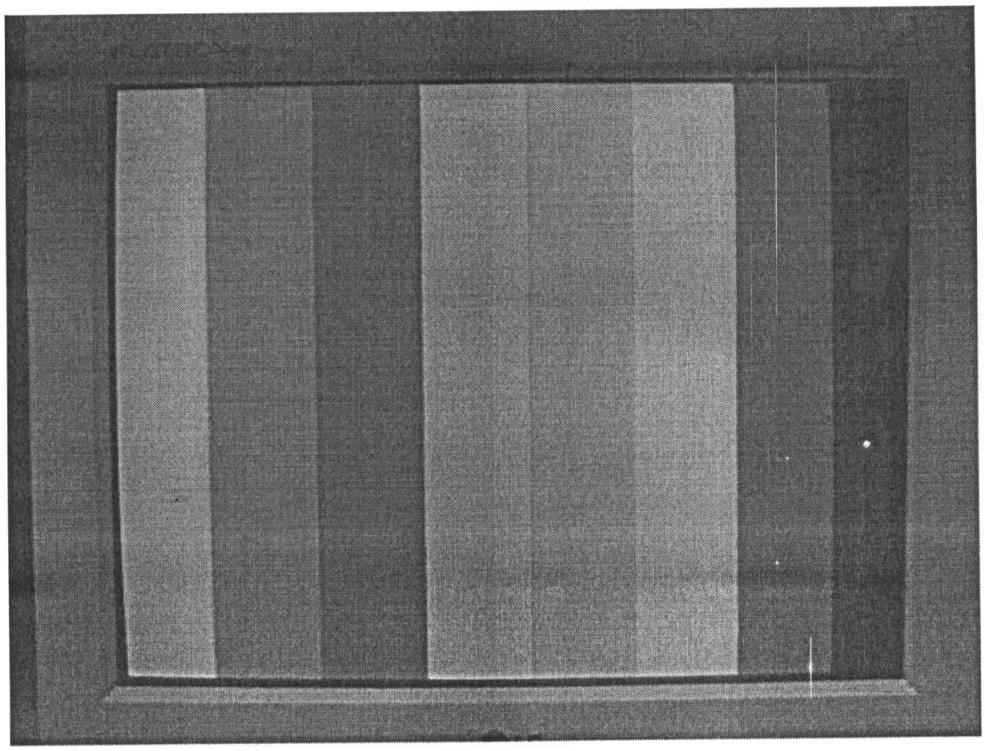


รูปที่ 4.10 รูปสัญญาณ Horizontal sync, Vertical sync

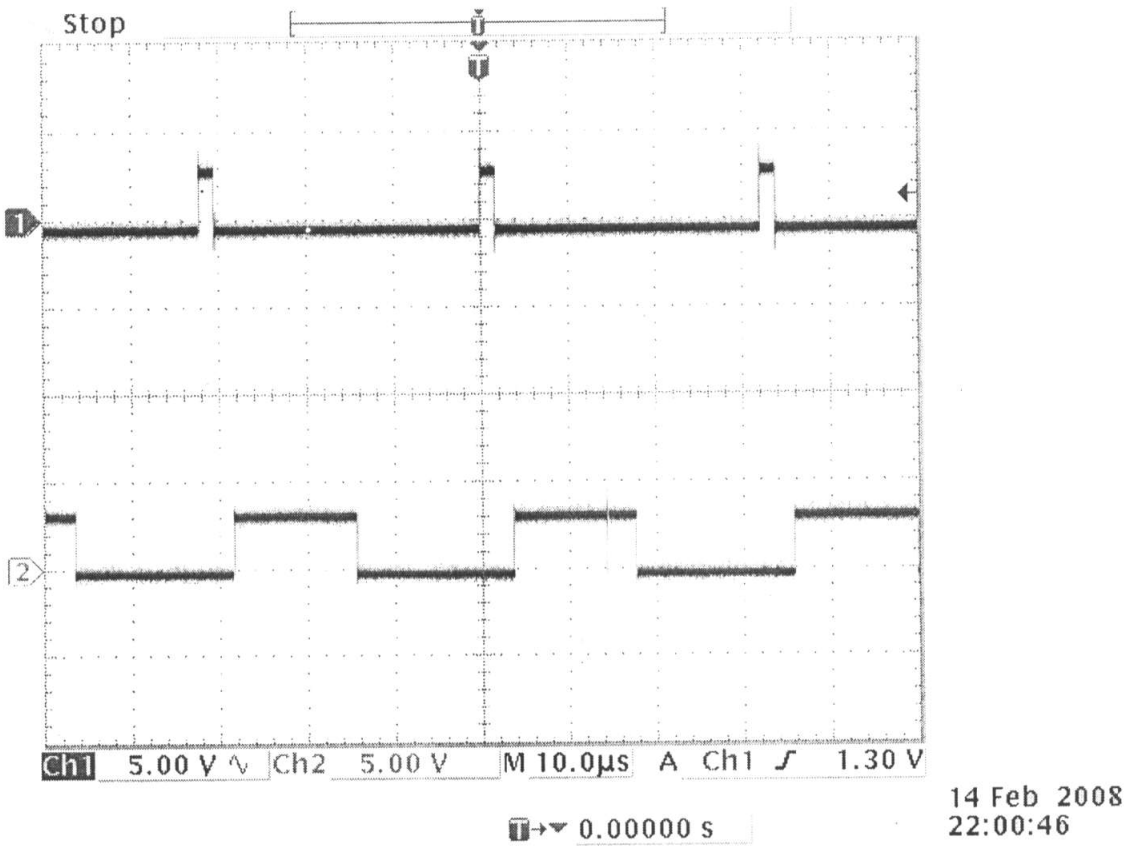


รูปที่ 4.11 รูปสัญญาณ Horizontal sync, Vertical sync (ส่วนขยาย)

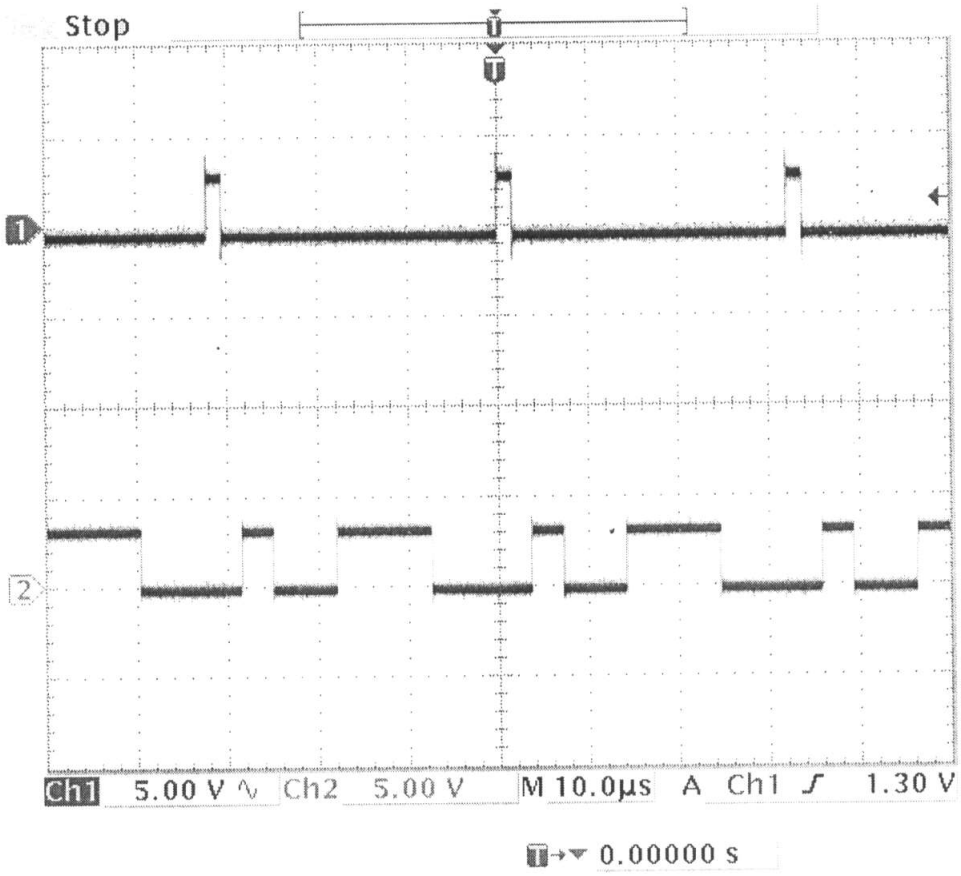
ตัวอย่างภาพ



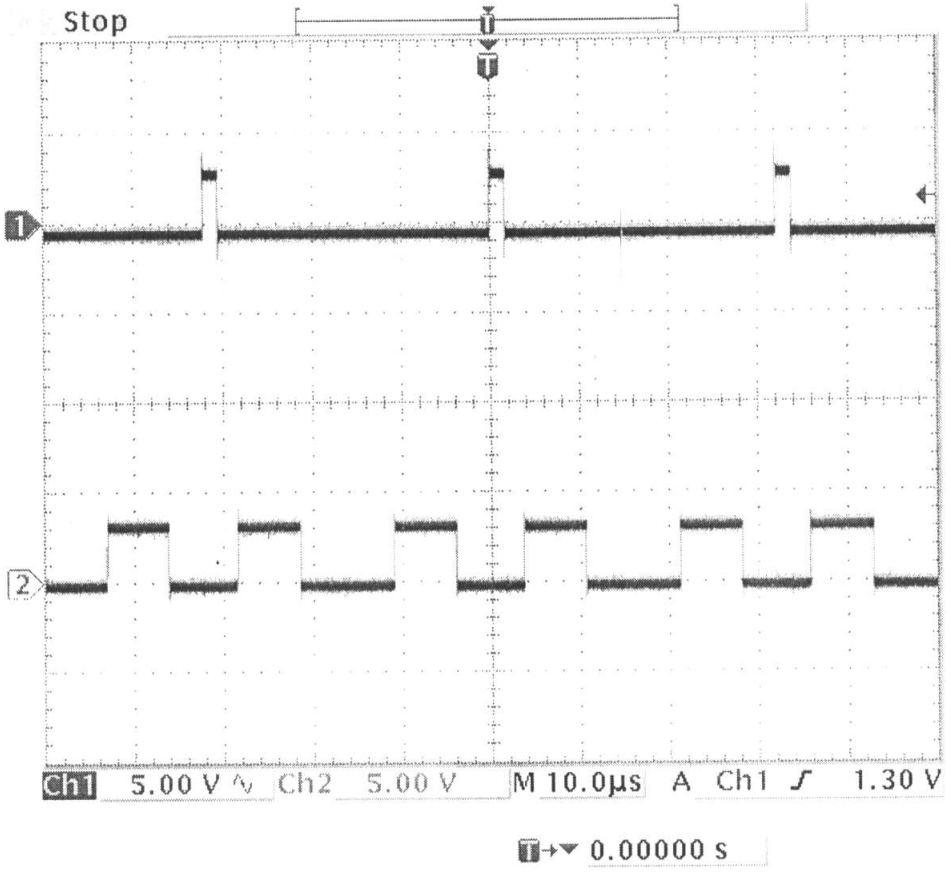
รูปที่4.12รูปแบบแถบสี



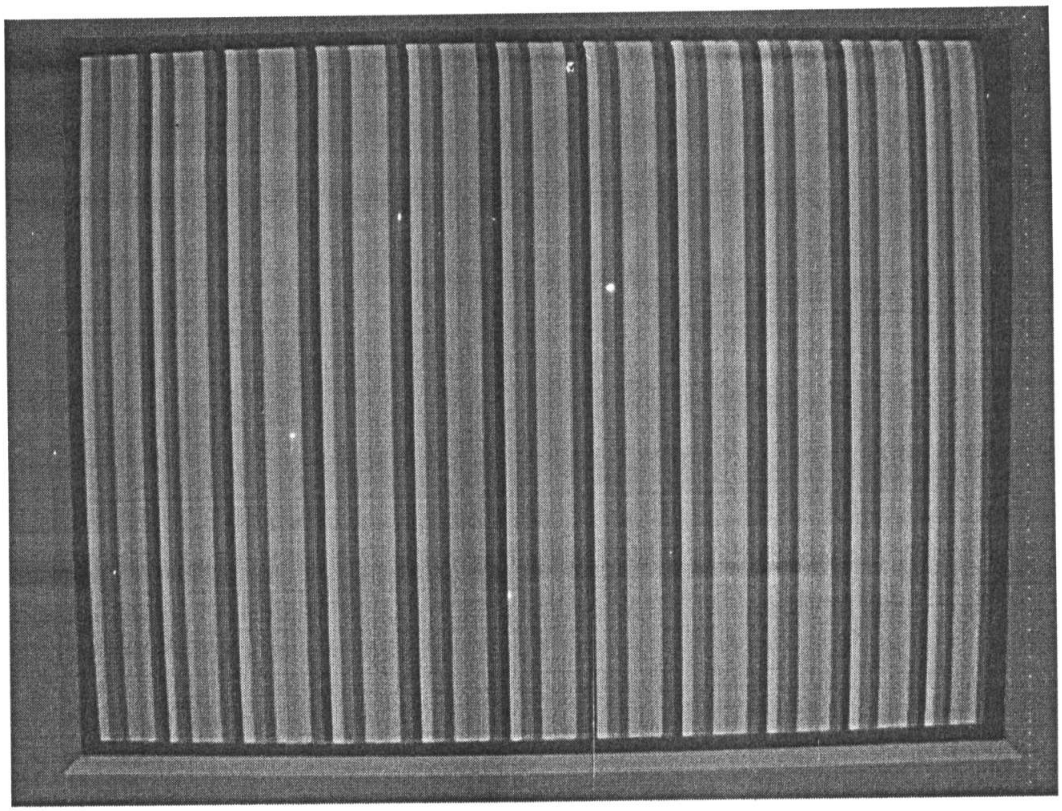
รูปที่ 4.13 รูปสัญญาณ HSync เทียบกับ Red



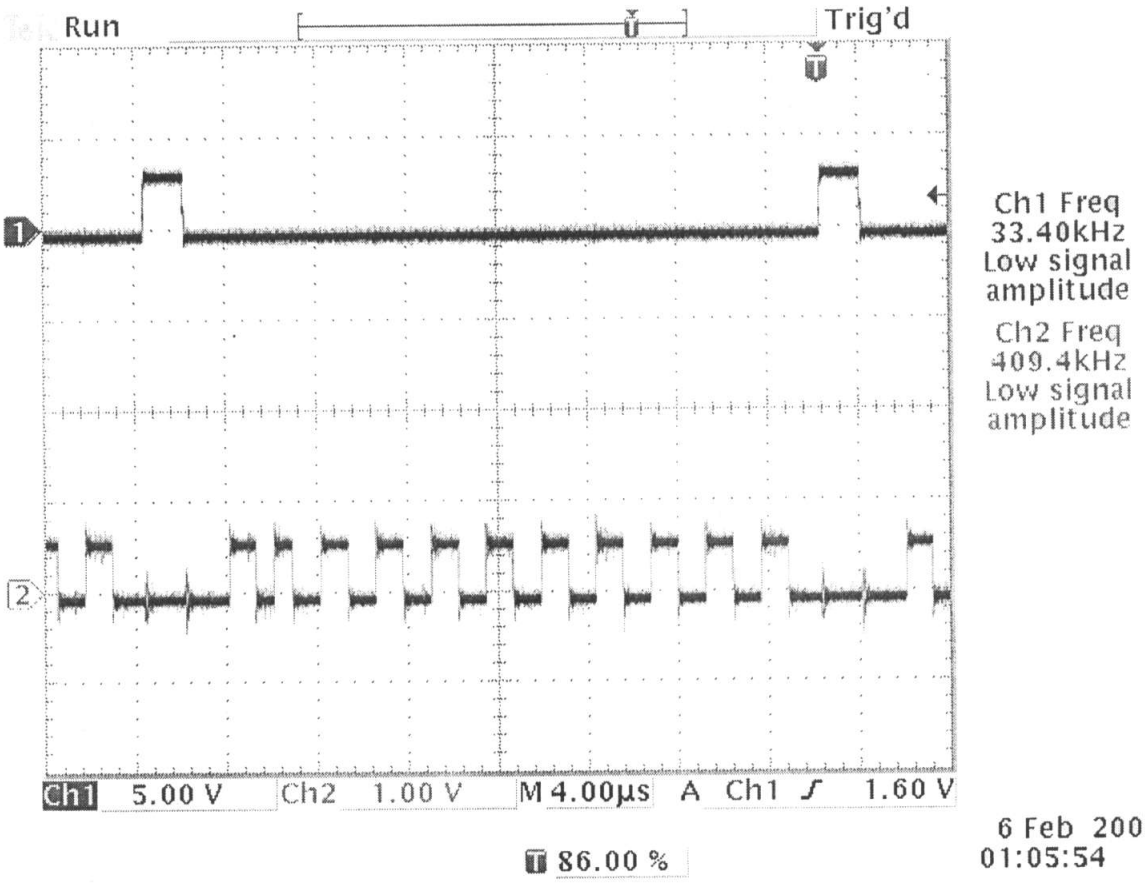
. รูปที่ 4.14 รูปสัญญาณ HSync เทียบกับGreen



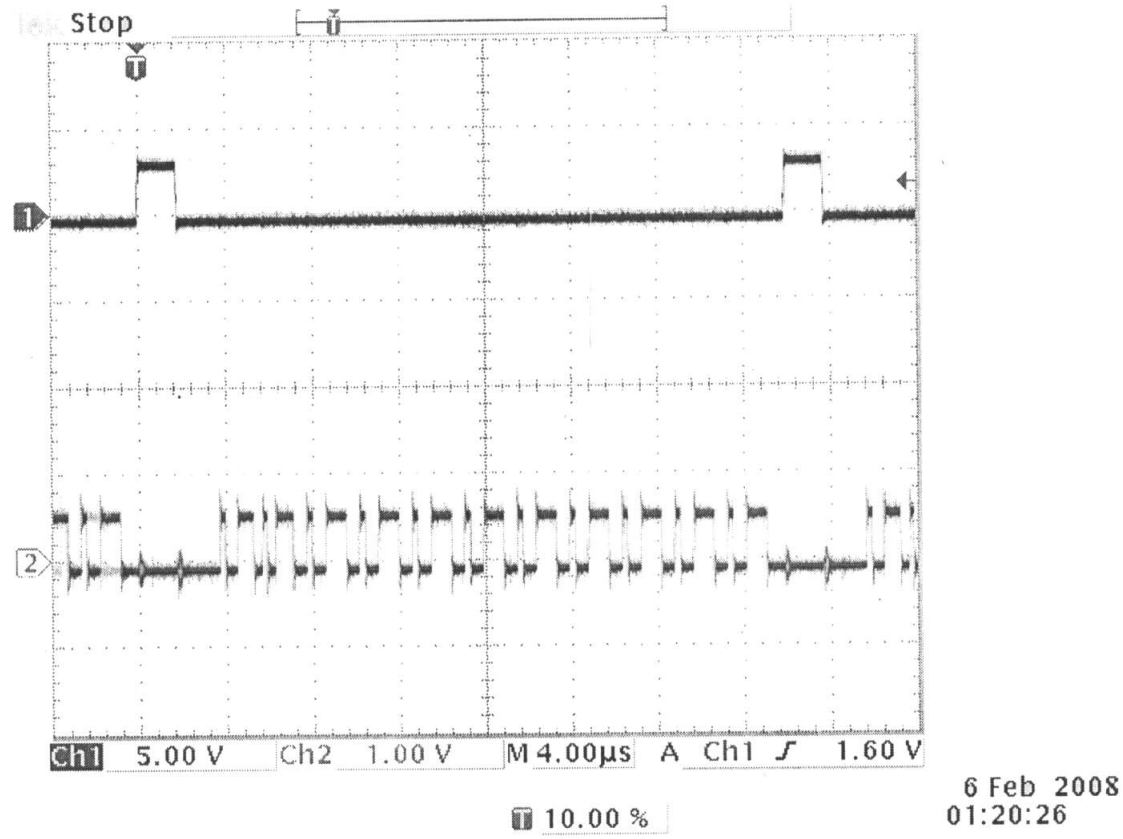
รูปที่ 4.15 รูปสัญญาณ HSync เทียบกับ Blue



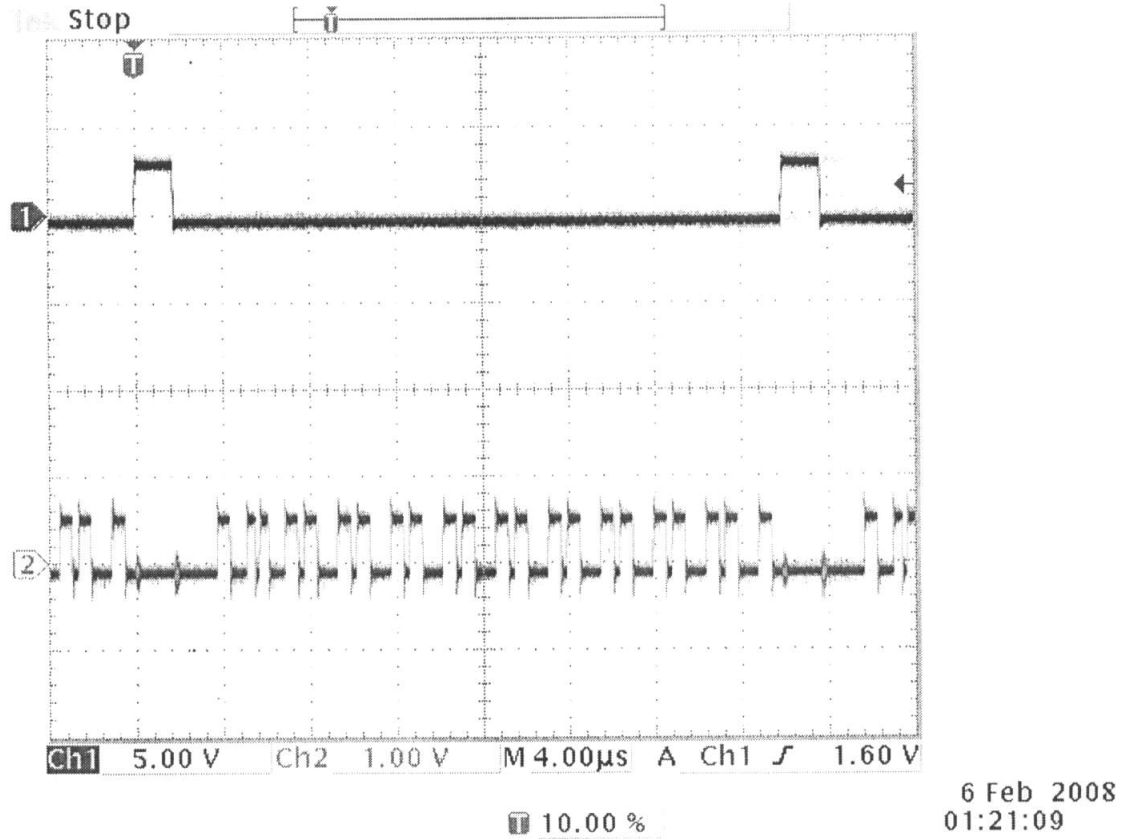
รูปที่ 4.16 รูปแบบแถบสี



รูปที่ 4.17 รูปสัญญาณ HSync เทียบกับ Red



รูปที่ 4.18 รูปสัญญาณ HSync เทียบกับGreen



รูปที่ 4.19 รูปสัญญาณ HSync เทียบกับ Blue

บทที่ 5
สรุป

5.1 สรุป

ในรายงานได้กล่าวถึงความเป็นมาของโครงงาน แนวคิด ทฤษฎี และรายละเอียดการสร้าง อุปกรณ์สร้างสัญญาณภาพสำหรับจอมอนิเตอร์ รวมถึงการทดสอบเบื้องต้น ในการทดสอบการติดต่อระหว่างไมโครคอนโทรเลอร์ (LPC2119) กับตัว SD Card สามารถต่อเข้าโดยตรงผ่าน Port SPI ได้เลย

ผลการทดลองแสดงให้เห็นว่า SD Card สามารถเก็บข้อมูลไฟล์ภาพได้ และแสดงภาพทางหน้าจอมนิเตอร์และสามารถเลือกคุภาพได้

สามารถอ่านไฟล์ภาพได้เฉพาะไฟล์ Bitmap

5.2 ปัญหาและแนวทางแก้ไข

- เนื่องจากอุปกรณ์หลัก (ไมโครคอนโทรลเลอร์ ARM7 และ Socket SD Card) เป็นเทคโนโลยี Surface Mount ทำให้ยากต่อการทดสอบ ผู้จัดทำจึงได้ใช้บอร์ดทดลองของไมโครคอนโทรลเลอร์ตัวนี้ โดยได้ใช้เบอร์ LPC2119

- เนื่องจาก SD Card มีการจัดเรียงข้อมูลในระบบ FAT32 ในการตรวจสอบความถูกต้องจากการอ่านข้อมูลภายใน SD Card จึงทำได้ยาก ดังนั้นผู้จัดทำจึงใช้ซอร์ฟแวร์ Winhex มาช่วยตรวจสอบข้อมูลและตำแหน่ง Address ภายใน SD Card

-เนื่องจาก ARM มีความเร็วในการทำงานช้า ทำให้ไม่สามารถแสดง pixel ของรูปภาพให้มากกว่าการทดลองได้ ดังนั้นจึงต้องเพิ่มสัญญาณนาฬิกาเข้ามาช่วยให้ความเร็วเพิ่มมากขึ้น

5.3 ประโยชน์ที่ได้รับ

- มีความรู้ความเข้าใจในการเข้ารหัสไฟล์ Bitmap
- มีความรู้ความเข้าใจในการแปลงไฟล์ภาพ Bitmap ภายในตัวไมโครคอนโทรลเลอร์
- เนื่องจาก SD Card มีการจัดเรียงไฟล์แบบ FAT32 ทำให้ผู้จัดทำมีความเข้าใจในตัวระบบ FAT32 เพื่อใช้ในการติดต่อระหว่างไมโครคอนโทรลเลอร์กับ SD Card
- มีความรู้ความเข้าใจในการติดต่อสื่อสารข้อมูลในแบบ SPI Bus
- มีความรู้ความเข้าใจในการสร้างสัญญาณภาพแบบอนาลอกผ่านทาง Port D-Sub
- มีความรู้ความเข้าใจในการใช้งานไมโครคอนโทรลเลอร์ตระกูล ARM7

ภาคผนวก

ภาคผนวก ก

โค้ดโปรแกรม (Source Code)

```
/*************************************/
/* Project: Single Chip VGA Controller */
/* Program: Main Function                    */
/* Name: MainFunc.c                                */
/* MCU: ET-ARM STAMP LPC2119                 */
/* Version: V1.0 Beta                        */
/*************************************/

#include "lpc21xx.h"
#include "SPICardCmd.h"
#include "SPICardDriver.h"
#include "VGAOutputCmd.h"
#include "VGAOutputDriver.h"
#include "Stdio.h"
#define CR 0x0D

void SPISet()
{
        /* Initial SPI0 Pin Connect on P0.4 to P0.7 */
        PINSEL0 |= 0x00005500; // Set P0.4 is SCK (SPI0)
        /* Initial SPI0 Function Interface */
        /* SPI0 Clock Counter Register */
        /* SCK SPI = (CCLK/VPBDIV)/S0PCCR) */
        S0SPCCR = 0x000000F;

        /* SPI0 Control Register */
        S0SPCR &= 0x000000F7; // CPHA = 0 -> Rising    Clock Shift Data
        S0SPCR &= 0x000000EF; // CPOL = 0 -> Normal Clock
        S0SPCR |= 0x00000020; // MSTR = 1 -> Master SPI
        S0SPCR &= 0x000000BF; // LSBF = 0 -> MSB First
        S0SPCR &= 0x0000007F; // SPIE = 0 -> Disable SPI Interrupt
}


void OutputSet()
{
        PINSEL1 |= 0x00000000;      // Set P0.00 - P0.31 as GPIO
        PINSEL2 |= 0x00000000;      // Set P1.16 - P1.31 as GPIO
        IODIR0 |= 0x02000000;       // Set 25 as Vertical Sync Output
        IODIR1 |= 0x00070000;       // Set P1.16 - P1.18 as Colour Output
        IODIR0 |= 0x000F0000;       // Set P0.16 - P0.19 as Card Status
        IOCLR1 = 0x00070000;        // Clear Pin P1.16 - P1.18
        IOCLR0 = 0x03CF0000;        // Clear Pin P0.22 - P0.25
        IODIR0 |= 0x00000008;       // Set P0.03 as SSEL0
        IOSET0 = 0x00000008;        // Set P0.03 DeSelect Card
}
```

```
void SetDutyCycle()
{
        PWMMR4 = (int)((PWM_MR0INIT*7)/100L);
        PWMLER |= (1<<4); // Enable PWM Match4 Latch
}

void InitialisePWM()
{
        PINSEL0 &= 0xFFFCFFFF; //P0.8 as PWM4 as VSync
        PINSEL0 |= 0x00020000;
        PWMTCR = PWMCR_RESET;       // reset PWM counter
        PWMPR = PWM_PCLK_DIV;       // set PWM Prescaler
        PWMMR0 = PWM_MR0INIT;       // set Match0 (for the Single Edge PWM-
Channels)
        PWMLER |= (1<<0);           // Enable PWM Match0 Latch
        PWMMCR |= (1<<1);           // reset PWM Timer on PWMMR0 match
        PWMPCR &= ~((1<<4));        // Speciefied PWM
        PWMPCR |= (1<<12);                  // enable outputs for specified PWM
        PWMTCR = PWMCR_ENABLE | (1<<3); // Counter Enable & PWM Enable
        EXTMODE |= 0x02;            // Falling edge interrupt
        VICVectAddr0 = (unsigned) isr_eint1;
        VICVectCntl0 = 0x20 | 8;
        VICIntEnable |= 1<<8;
}

void Uart0_Set(unsigned int Baudrate)
{
        unsigned short u0dl;
        u0dl = 29491200/(16*Baudrate);
        PINSEL0 |= 0x00000005;
        U0LCR = 0x83;
        U0DLL = u0dl & 0xFF;
        U0DLM = (u0dl>>8);
        U0LCR &= 0x7F;
}

int getchar (void)              // Read character from Serial Port
{
        while(!(U0LSR &0x01));
        return(U0RBR);
}

int putchar(int ch)             // Write character to Serial Port
{
        if(ch == '\n')
```

```
        {
                while(!(U0LSR & 0x20));
                U0THR = CR;
        }
        while(!(U0LSR & 0x20));
        return (U0THR = ch);
}

 void main()
{
        SPISet();
        OutputSet();
        InitialisePWM();
        SetDutyCycle();
        Uart0_Set(9600);        // Set UART0 = 9600,8,N,1
        //printf("What the HeLL!!!\n\r");
        Card_Init();
        AccessFile();
        PWMMCR |= (1<<0);// interrupt PWM Timer on PWMMR0 match
}

/*************************************************/
/* Project: Single Chip VGA Controller          */
/* Program: SPI and Card command interface module */
/* Name: SPICardCmd.h                                    */
/* MCU: ET-ARM STAMP LPC2119                      */
/* Version: V1.0 Beta                                 */
/*************************************************/

#include "lpc21xx.h"
#include "SPICardDriver.h"
#include "VGAOutputDriver.h"
#include "Stdio.h"

void SPI_Send(unsigned char *Buf, int Length)
{
        unsigned char Dummy;
        if(Length == 0)
                return;
        while(Length != 0)
        {
                S0SPDR = *Buf;
                //printf("SPI Send = %X\n\r", *Buf);
                while((S0SPSR & 0x80) != 0x80);    // wait until the busy bit is cleared
                Dummy = S0SPDR;
                Length--;
```

```
            Buf++;
        }
        return;
}

unsigned char SPIReceiveByte(void)
{
        unsigned char data;
        S0SPDR = 0xFF;      // Write dummy byte out to generate clock, then read data
from MISO
        while((S0SPSR & 0x80) != 0x80);    // wait until the busy bit is cleared
        data = S0SPDR;
        return(data);
}

void SPI_Receive(unsigned char *Buf, int Length)
{
        int i;
        for(i=0; i<Length; i++)
        {
                *Buf = SPIReceiveByte();
                Buf++;
        }
        return;
}

int CardResponse(unsigned char Response)  // Repeatedly reads the CARD until we get
the response we want or Timeout
{
        int count = 0xFF;
        unsigned char result;
        while(count>0)
        {
                result = SPIReceiveByte();
                if(result==Response)
                {
                        break;
                }
                count--;
        }
        if(count==0)
        {
        //printf("Card Response = 0\n\r");
        return 0;       // Failure, loop was exited due to timeout
        }
        else
```

```
        {
        return 1;       // Normal, loop was exited before timeout
        }
}

void Card_Init()
{
        IOSET0 = SPISel; // Set SPI SSEL DeSelect Card
        for(i=0; i<80; i++)
        {
                CardRDData[i] = 0xFF;
        }
        SPI_Send(CardRDData, 80);
        IOCLR0 = SPISel;    // Clear SPI SSEL Select Card
        SPI_Send(CardCmd0, CardCmdSize);        //Send CMD0 (RESET or
GO_IDLE_STATE) command, all argumentsm are 0x00 for the reset command,
precalculaed checksum
        if(CardResponse(0x01)==0)  // if = 0 then there was a timeout waiting for 0x01
from the Card
        {
                IOSET0 = IdleStateTimeout;
                IOSET0 = SPISel;    // Set SPI SSEL DeSelect Card
                printf("CMD0 Fail!!! Check Card.\n");
                while(1);
        }
        IOSET0 = SPISel;    // Set SPI SSEL DeSelect Card
        SPIReceiveByte();   // Send some dummy clocks after go idle state
        IOCLR0 = SPISel;    // Clear SPI SSEL Select Card
        i = MaxTimeout;
        do      // send Card CMD1(SEND_OP_COND) to bring out of idle state
        {
                SPI_Send(CardCmd1, CardCmdSize);
                i--;
        }
        while((CardResponse(0x00)!=1) && (i>0) );
        if(i==0)        // timeout waiting for 0x00 from the Card
        {
                IOSET0 = OPCondTimeout;
                IOSET0 = SPISel;    // Set SPI SSEL DeSelect Card
                printf("CMD1 Fail!!! Check Card.\n");
                while(1);
        }
        IOSET0 = SPISel;    // Set SPI SSEL DeSelect Card
        SPIReceiveByte();   // Send some dummy clocks after Send OP Cond
        //printf("Initial Card Complete\n");
}
```

```
void CardRead()
{
	CardCmd[0] = 0x51;
	CardCmd[1] = Buffer[0];
	CardCmd[2] = Buffer[1];
	CardCmd[3] = Buffer[2];
	CardCmd[4] = Buffer[3];
	CardCmd[5] = 0xFF;
	IOCLR0 = SPISel;	// Clear SPI SSEL Select Card
	SPI_Send(CardCmd, CardCmdSize);
	if(CardResponse(0x00)==0)  // if = 0 then there was a timeout waiting for 0x01
from the Card
	{
		IOSET0 = ReadBlockTimeOut;
		IOSET0 = SPISel;	// Set SPI SSEL DeSelect Card
		printf("Read Block TimeOut!!! Check Card.\n");
		while(1);
	}
	if(CardResponse(0xFE)==0)  // if = 0 then there was a timeout waiting for 0xFE
from the Card
	{
		IOSET0 = ReadBlockDataTokenMissing;
		IOSET0 = SPISel;	// Set SPI SSEL DeSelect Card
		printf("Read Block Data Token Missing!!! Check Card.\n");
		while(1);
	}
	SPI_Receive(CardRDData, 512);
	/*printf("\n");
	l = 0;
	for(m=0; m<512; m++)
	{
		printf("%X ", CardRDData[m]);
		if(l==15)
		{
			printf("\n");
			l=0;
		}
		else
		{
		l++;
		}
	}*/
	//printf("\n");
	IOSET0 = SPISel;	// Set SPI SSEL DeSelect Card
	SPIReceiveByte();	// Send some dummy clocks after Send OP Cond
```

```
        SPIReceiveByte();     // Send some dummy clocks after Send OP Cond
        //printf("Read Block Complete!!!\n");
}

void FindStartAdd()             //Find start address of root directory
{
        rsv_size = CardRDData[15];
        rsv_size = rsv_size<<8;
        rsv_size += CardRDData[14];
        ClusterSize = CardRDData[13];       //Number of secter per cluster (Cluster Size
in Sector)
        fat = CardRDData[16];
        fat_size = CardRDData[37];
        fat_size = fat_size<<8;
        fat_size += CardRDData[36];
        root_start = rsv_size + (fat*fat_size);
        root_start = root_start*0x200;
        root_add[0] = (root_start>>24)&0xFF;
        root_add[1] = (root_start>>16)&0xFF;
        root_add[2] = (root_start>>8)&0xFF;
        root_add[3] = root_start&0xFF;
        for(i=0; i<4; i++)
        {
                Buffer[i] = root_add[i];
        }
        BytePerSector = CardRDData[12];
        BytePerSector = BytePerSector<<8;
        BytePerSector += CardRDData[11];
        CardRead();
}

void ReceivedColour()
{
        for(q=q; q<512; q++)
        {
                if(p==0)
                {
                        if(CardRDData[q]>0x0F)
                        {
                                Colour += 0x1;
                                Colour = Colour<<4;
                        }
                        else
                        {
                                Colour += 0x0;
                                Colour = Colour<<4;
```

```
        }
        p++;
}
else if(p==1)
{
        if(CardRDData[q]>0x0F)
        {
                Colour += 0x1;
                Colour = Colour<<4;
        }
        else
        {
                Colour += 0x0;
                Colour = Colour<<4;
        }
        p++;
}
else if(p==2)
{
        if(CardRDData[q]>0x0F)
        {
                Colour += 0x1;
                Colour = Colour<<4;
        }
        else
        {
                Colour += 0x0;
                Colour = Colour<<4;
        }
        p=0;
        if(Colour == 0x0000)
        {
                dis[r][s] = Black;
        }
        else if(Colour == 0x0010)
        {
                dis[r][s] = Red;
        }
        else if(Colour == 0x0100)
        {
                dis[r][s] = Green;
        }
        else if(Colour == 0x1000)
        {
                dis[r][s] = Blue;
        }
```

```
                        else if(Colour == 0x1010)
                        {
                                dis[r][s] = Fuchsia;
                        }
                        else if(Colour == 0x0110)
                        {
                                dis[r][s] = Yellow;
                        }
                        else if(Colour == 0x1100)
                        {
                                dis[r][s] = Aqua;
                        }
                        else if(Colour == 0x1110)
                        {
                                dis[r][s] = White;
                        }
                        Colour = 0;
                        if(s<39)
                        {
                                s++;
                        }
                        else if(s==39)
                        {
                                s=0;
                                r++;
                        }
                }
        }
}

void GetData()
{
        SecNum = 0;
        while(SecNum<SecTotal+1)
        {
                file_add[0] = (start_add>>24)&0xFF;
                file_add[1] = (start_add>>16)&0xFF;
                file_add[2] = (start_add>>8)&0xFF;
                file_add[3] = start_add&0xFF;
                for(i=0; i<4; i++)
                {
                        Buffer[i] = file_add[i];
                }
                CardRead();
                if(SecNum==0)
                {
```

```
                BeginningOfBitmapData = CardRDData[13];
                BeginningOfBitmapData = BeginningOfBitmapData<<8;
                BeginningOfBitmapData += CardRDData[12];
                BeginningOfBitmapData = BeginningOfBitmapData<<8;
                BeginningOfBitmapData += CardRDData[11];
                BeginningOfBitmapData = BeginningOfBitmapData<<8;
                BeginningOfBitmapData += CardRDData[10];
                q = BeginningOfBitmapData;
                BitmapWidth = CardRDData[21];
                BitmapWidth = BitmapWidth<<8;
                BitmapWidth += CardRDData[20];
                BitmapWidth = BitmapWidth<<8;
                BitmapWidth += CardRDData[19];
                BitmapWidth = BitmapWidth<<8;
                BitmapWidth += CardRDData[18];
                BitmapHeight = CardRDData[25];
                BitmapHeight = BitmapHeight<<8;
                BitmapHeight += CardRDData[24];
                BitmapHeight = BitmapHeight<<8;
                BitmapHeight += CardRDData[23];
                BitmapHeight = BitmapHeight<<8;
                BitmapHeight += CardRDData[22];
                BitDept = CardRDData[28];
                s = 0;
                r = 0;
                ReceivedColour();
        }
        else
        {
                q = 0;
                ReceivedColour();
        }
        start_add = start_add+0x200;
        SecNum++;

    }
    //printf("Get data Complete!!!\n");
}

void CheckFile()
{
    FileName = CardRDData[0];
    if(FileName == 0)
    {
        printf("No File in card, Check Card!!!\n");
        while(1);
```

```
        }
        start_add = CardRDData[21+(CountFile*32)];
        start_add = start_add<<8;
        start_add += CardRDData[20+(CountFile*32)];
        start_add = start_add<<8;
        start_add += CardRDData[27+(CountFile*32)];
        start_add = start_add<<8;
        start_add += CardRDData[26+(CountFile*32)];
        if((start_add & 0xFFFF) == 0)
        {
                //printf("No data in file, Check Card!!!\n");
                CountFile++;
        }
}

void GetFile()
{
        CheckFile();
        start_add = CardRDData[21+(CountFile*32)];
        start_add = start_add<<8;
        start_add += CardRDData[20+(CountFile*32)];
        start_add = start_add<<8;
        start_add += CardRDData[27+(CountFile*32)];
        start_add = start_add<<8;
        start_add += CardRDData[26+(CountFile*32)];
        start_add = (start_add-0x02)*ClusterSize*BytePerSector;
        start_add = start_add+root_start;
        FileSize = CardRDData[31+(CountFile*32)];
        FileSize = FileSize<<8;
        FileSize += CardRDData[30+(CountFile*32)];
        FileSize = FileSize<<8;
        FileSize += CardRDData[29+(CountFile*32)];
        FileSize = FileSize<<8;
        FileSize += CardRDData[28+(CountFile*32)];
        SecTotal = FileSize/BytePerSector;
        if(!((CardRDData[8+(CountFile*32)]==0x42) &&
(CardRDData[9+(CountFile*32)]==0x4D) &&
(CardRDData[10+(CountFile*32)]==0x50)))
        {
                printf("Wrong file type. File type is %C%C%C\n",
CardRDData[8+(CountFile*32)], CardRDData[9+(CountFile*32)],
CardRDData[10+(CountFile*32)]);
                while(1);
        }
        printf("File type is %C%C%C\n", CardRDData[8+(CountFile*32)],
CardRDData[9+(CountFile*32)], CardRDData[10+(CountFile*32)]);
```

```
        GetData();
}

void AccessFile()
{
        CardRead();
        FindStartAdd();
        GetFile();
        for(r=0; r<BitmapHeight; r++)
        {
                for(s=0; s<BitmapWidth; s++)
                {
                        printf("%X",dis[r][s]);
                        if(l==9)
                        {
                                printf("\n");
                                l=0;
                        }
                        else
                        {
                        l++;
                        }
                }
        }
}

/*************************************************/
/* Project: Single Chip VGA Controller          */
/* Program: SPI Mode SD/MMC Card interface driver */
/* Name: SPICardDriver.h                         */
/* MCU: ET-ARM STAMP LPC2119                     */
/* Version: V1.0 Beta                            */
/*************************************************/

#ifndef __SPI_CARD_DRIVER__
#define __SPI_CARD_DRIVER__

/* SPI select pin */
#define SPISel 0x00000008
#define CardDataSize 512
#define CardCmdSize 6
#define MaxTimeout 0xFFF
#define IdleStateTimeout 0x00010000
#define OPCondTimeout 0x00020000
#define ReadBlockTimeOut 0x00040000
#define ReadBlockDataTokenMissing 0x00080000
```

```
int i, l, m, n, p, q, r, s, BitDept, rsv_size, fat, fat_size, root_add[4], Buffer[4], SecNum,
SecTotal, file_add[4], BytePerSector, ClusterSize;
int CountFile, FileName, Colour;
unsigned int root_start, start_add, FileSize, BeginningOfBitmapData, BitmapWidth,
BitmapHeight;
unsigned int dis[40][40];

unsigned char CardRDData[512];
unsigned char CardCmd[6];
unsigned char CardCmd0[6] = {0x40, 0x00, 0x00, 0x00, 0x00, 0x95};
unsigned char CardCmd1[6] = {0x41, 0x00, 0x00, 0x00, 0x00, 0xFF};
unsigned char CardCmd17[6] = {0x51, 0x00, 0x00, 0x00, 0x00, 0xFF};

#endif/* __SPI_CARD_DRIVER__ */

/************************************************/
/* Project: Single Chip VGA Controller          */
/* Program: VGA Output                                          */
/* Name: VGAOutputCmd.h                                 */
/* MCU: ET-ARM STAMP LPC2119                              */
/* Version: V1.0 Beta                                   */
/************************************************/

#include "lpc21xx.h"
#include "VGAOutputDriver.h"
#include "SPICardDriver.h"

void delay_us(long int us)     // Delay 0.4 us
{
        long int i;
        for(i=0; i<us; i++);
}

void Generate()
{
        for(k=0; k<24; k++)
        {
                IOPIN1 = dis[j][k];
        }
        IOCLR1 = White;
}

void isr_eint1(void) __irq
{
        if(j<375)
```

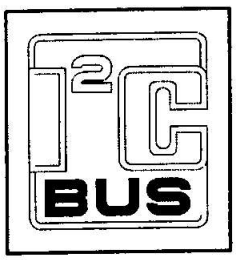
```
        {
                delay_us(36);
                Generate();
                j++;
        }
        else if((j>374)&&(j<380))
        {
                IOSET0 = VSync;
                j++;
        }
        else if(j>379)
        {
                IOCLR0 = VSync;
                j=0;
        }
        PWMIR = (1<<0);
        VICVectAddr = 0;     //Acknowledge interrupt
}
/*************************************************/
/* Project: Single Chip VGA Controller          */
/* Program: VGA Output Driver                                    */
/* Name: VGAOutputDriver.h                                  */
/* MCU: ET-ARM STAMP LPC2119                                */
/* Version: V1.0 Beta                                    */
/*************************************************/

#define Red 0x00010000              // Define Red at P1.16
#define Green 0x00020000    // Define Green at P1.17
#define Blue 0x00040000             // Define Blue at P1.18
#define Fuchsia 0x00050000 // Define Fuchsia at P1.16 and P1.18
#define Yellow 0x00030000  // Define Yellow at P1.16 and P1.17
#define Aqua 0x00060000             // Define Aqua at P1.17 and P1.18
#define White 0x00070000    // Define White is P1.16 - P1.18 are active
#define Black 0x00000000    // Define Black is P1.16 - P1.18 no active
#define VSync 0x02000000   // Define Vertical Sync at P0.25
#define PWMFreq 29000               // Define The desired PWM Frequency in Hertz
#define CPUClk 58982400             // Define Core Clock for CPU
#define PBSD 2                                  // Define Pin Output divider value
#define PCLK (CPUClk/PBSD)
#define PWM_PCLK_DIV 2
#define PWMTICSperSEC (PCLK / (PWM_PCLK_DIV+1))
#define PWM_MR0INIT (unsigned long)((PWMTICSperSEC/PWMFreq) + 0.5)
#define PWMCR_RESET (1<<1)
#define PWMCR_ENABLE (1 << 0)

int j, k;
```

ภาคผนวก ข
Data Sheets

USER MANUAL



LPC2119/2129/2194/2292/2294
USER MANUAL

Preliminary
Supersedes data of 2004 Feb 03

2004 May 03

Philips
Semiconductors

PHILIPS

2. LPC2119/2129/2292/2294 MEMORY ADDRESSING

MEMORY MAPS

The LPC2119/2129/2194/2292/2294 incorporates several distinct memory regions, shown in the following figures. Figure 2 shows the overall map of the entire address space from the user program viewpoint following reset. The interrupt vector area supports address re-mapping, which is described later in this section.

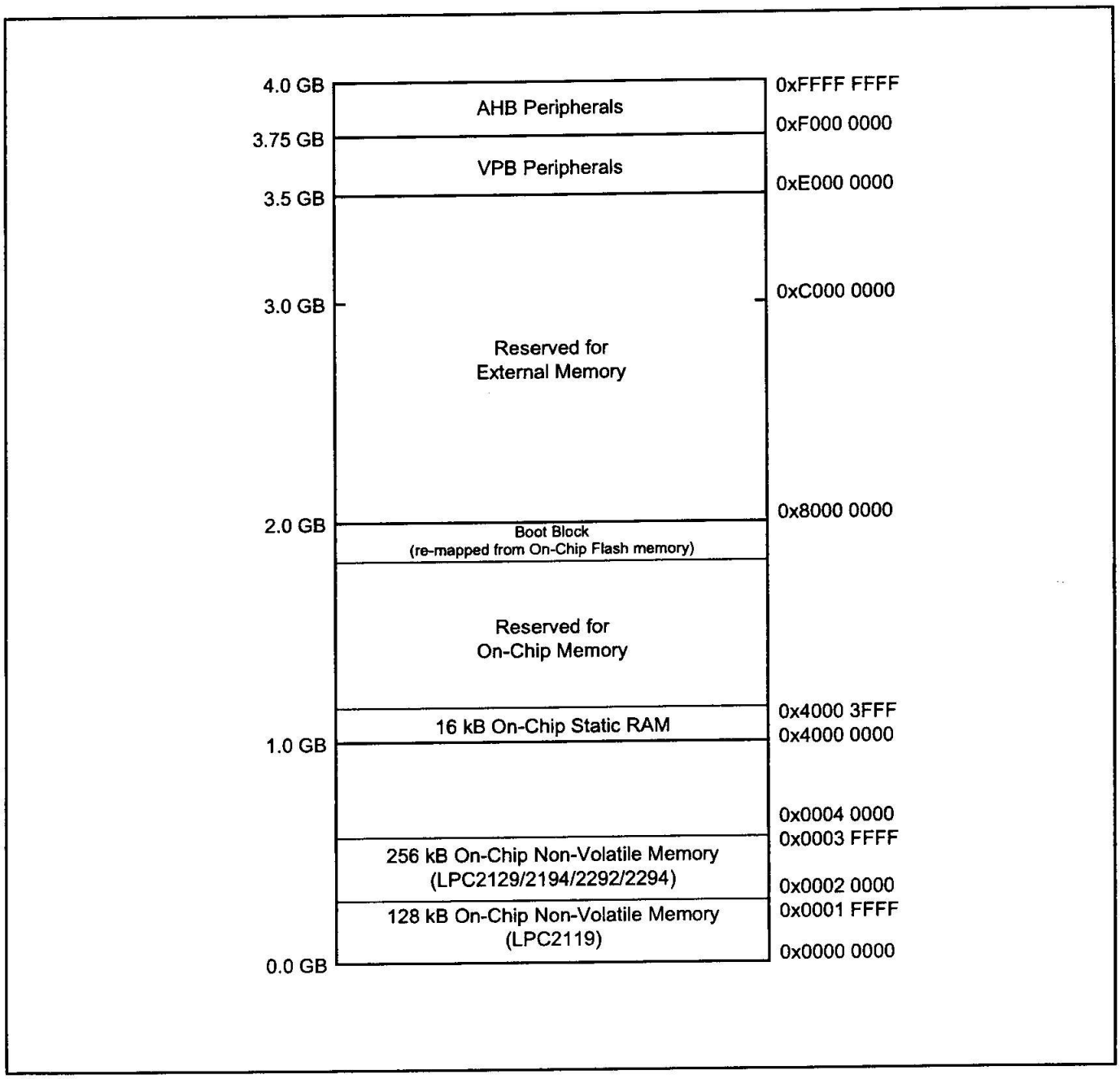


Figure 2: System Memory Map

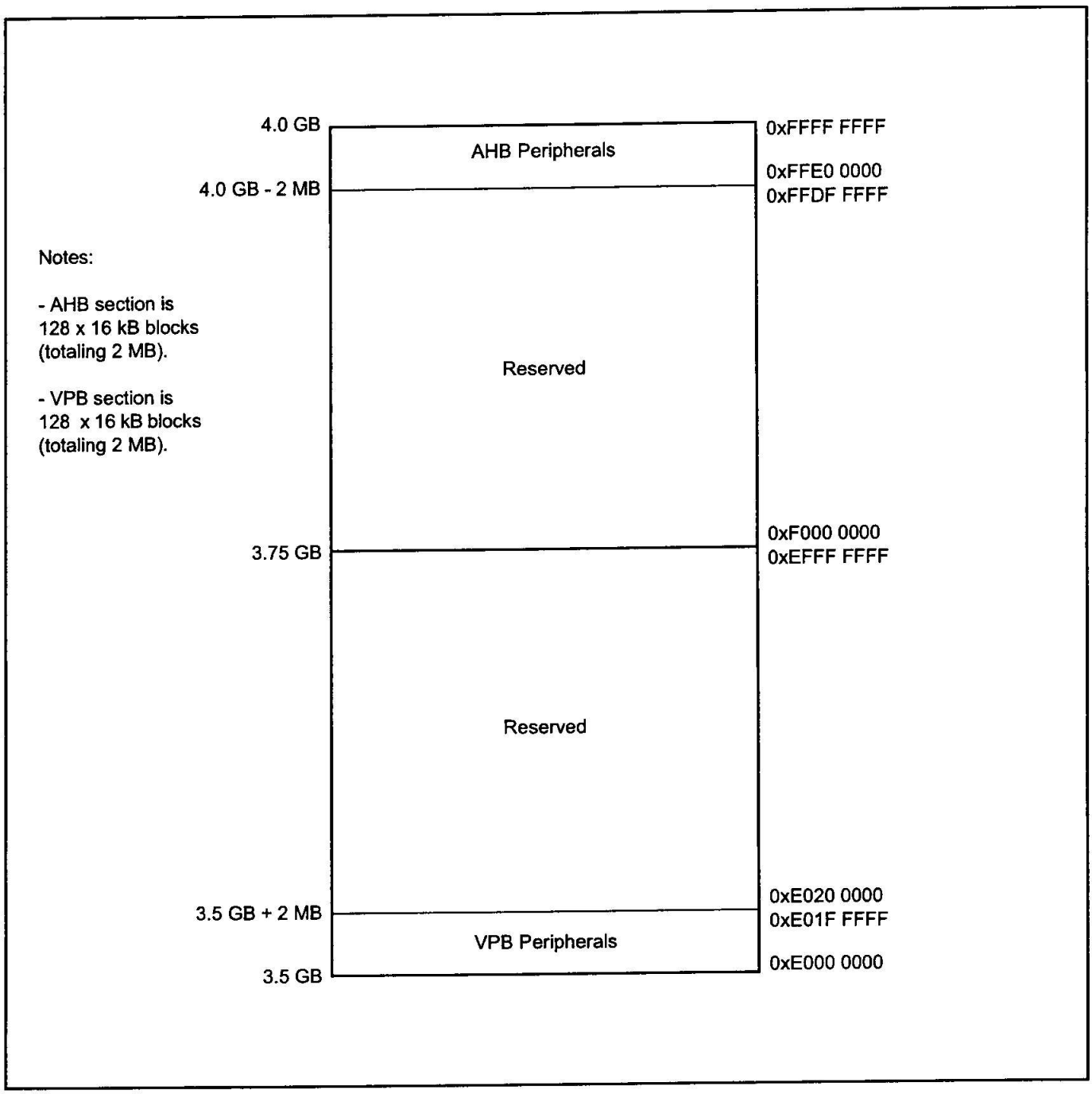


Figure 3: Peripheral Memory Map

Figures 3 through 5 show different views of the peripheral address space. Both the AHB and VPB peripheral areas are 2 megabyte spaces which are divided up into 128 peripherals. Each peripheral space is 16 kilobytes in size. This allows simplifying the address decoding for each peripheral. All peripheral register addresses are word aligned (to 32-bit boundaries) regardless of their size. This eliminates the need for byte lane mapping hardware that would be required to allow byte (8-bit) or half-word (16-bit) accesses to occur at smaller boundaries. An implication of this is that word and half-word registers must be accessed all at once. For example, it is not possible to read or write the upper byte of a word register separately.

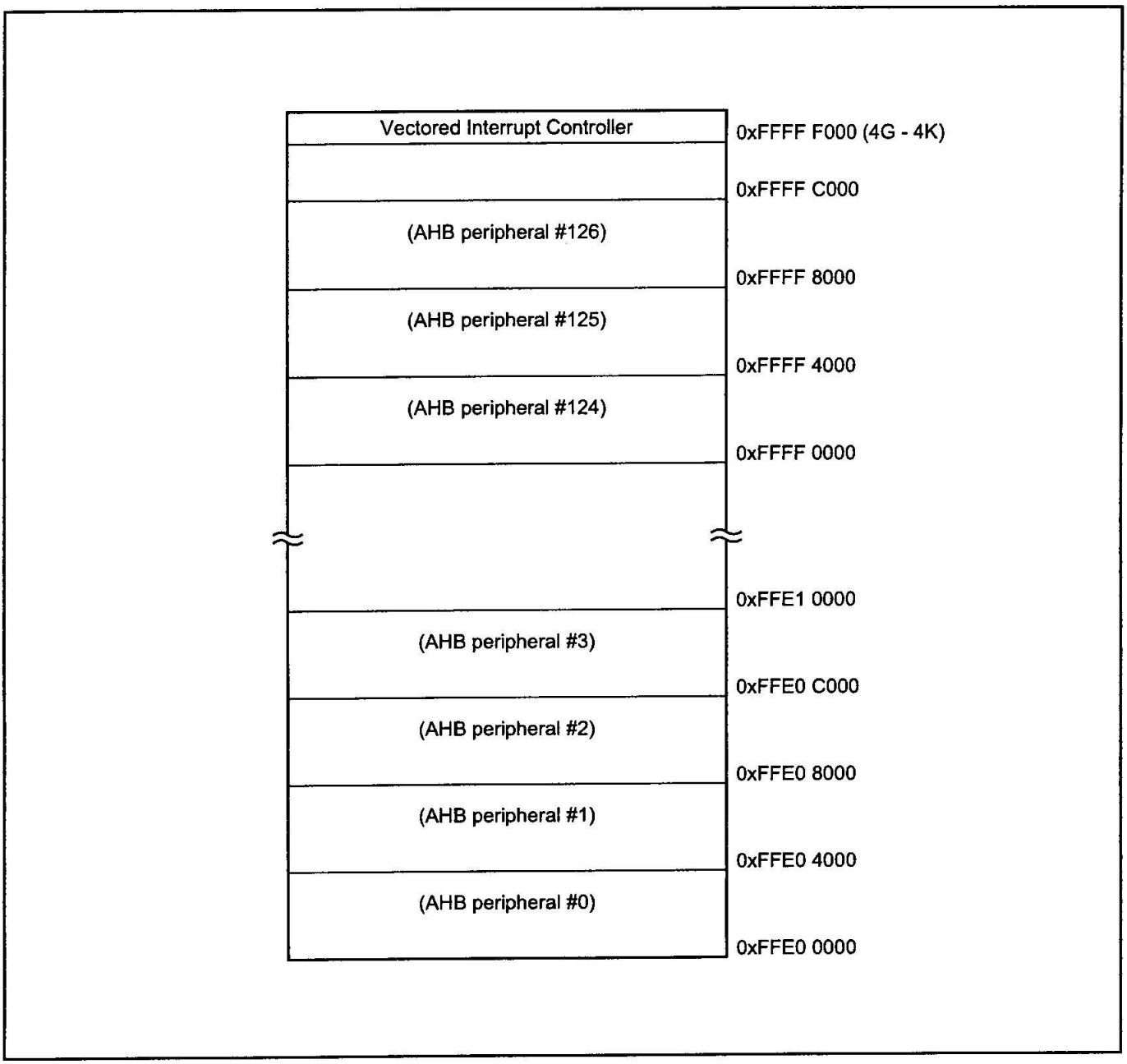


Figure 4: AHB Peripheral Map

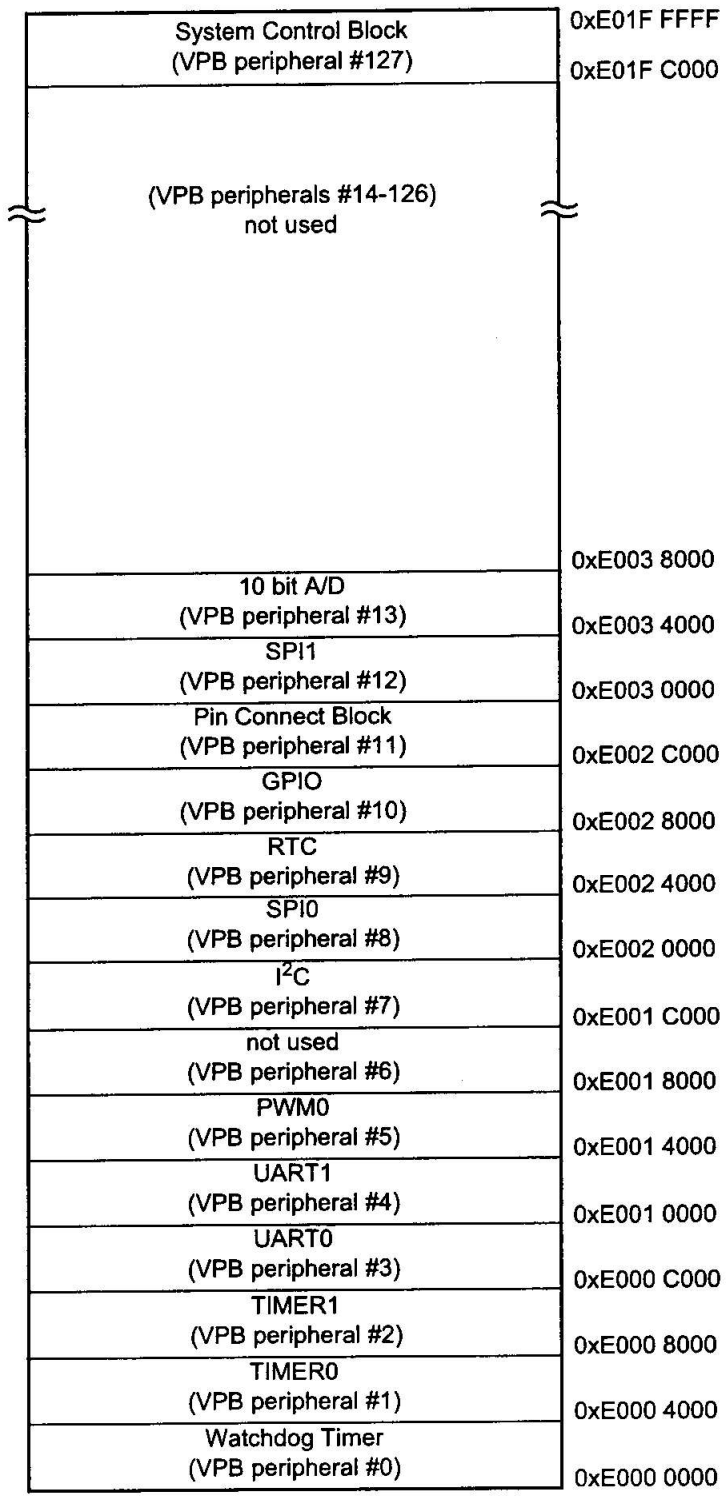


Figure 5: VPB Peripheral Map

LPC2119/2129/2194/2292/2294 MEMORY RE-MAPPING AND BOOT BLOCK

Memory Map Concepts and Operating Modes

The basic concept on the LPC2119/2129/2194/2292/2294 is that each memory area has a "natural" location in the memory map. This is the address range for which code residing in that area is written. The bulk of each memory space remains permanently fixed in the same location, eliminating the need to have portions of the code designed to run in different address ranges.

Because of the location of the interrupt vectors on the ARM7 processor (at addresses 0x0000 0000 through 0x0000 001C, as shown in Table 3 below), a small portion of the Boot Block and SRAM spaces need to be re-mapped in order to allow alternative uses of interrupts in the different operating modes described in Table 4. Re-mapping of the interrupts is accomplished via the Memory Mapping Control feature described in the System Control Block section.

Table 3: ARM Exception Vector Locations

Address	Exception
0x0000 0000	Reset
0x0000 0004	Undefined Instruction
0x0000 0008	Software Interrupt
0x0000 000C	Prefetch Abort (instruction fetch memory fault)
0x0000 0010	Data Abort (data access memory fault)
0x0000 0014	Reserved *
0x0000 0018	IRQ
0x0000 001C	FIQ

*: Identified as reserved in ARM documentation, this location is used by the Boot Loader as the Valid User Program key. This is descibed in detail in Flash Memory System and Programming on page 262.

Table 4: LPC2119/2129/2194/2292/2294 Memory Mapping Modes

Mode	Activation	Usage
Boot Loader mode	Hardware activation by any Reset	The Boot Loader always executes after any reset. The Boot Block interrupt vectors are mapped to the bottom of memory to allow handling exceptions and using interrupts during the Boot Loading process.
User Flash mode	Software activation by Boot code	Activated by Boot Loader when a valid User Program Signature is recognized in memory and Boot Loader operation is not forced. Interrupt vectors are not re-mapped and are found in the bottom of the Flash memory.
User RAM mode	Software activation by User program	Activated by a User Program as desired. Interrupt vectors are re-mapped to the bottom of the Static RAM.
User External mode	Activated by BOOT1:0 pins not 11 at Reset	Activated by the Boot Loader when either or both BOOT pins are low at the end of RESET low. Interrupt vectors are re-mapped from the bottom of the external memory map. **Note:** This mode is available in LPC2292/2294 only!

Memory Re-Mapping

In order to allow for compatibility with future derivatives, the entire Boot Block is mapped to the top of the on-chip memory space. In this manner, the use of larger or smaller flash modules will not require changing the location of the Boot Block (which would require changing the Boot Loader code itself) or changing the mapping of the Boot Block interrupt vectors. Memory spaces other than the interrupt vectors remain in fixed locations. Figure 6 shows the on-chip memory mapping in the modes defined above.

The portion of memory that is re-mapped to allow interrupt processing in different modes includes the interrupt vector area (32 bytes) and an additional 32 bytes, for a total of 64 bytes. The re-mapped code locations overlay addresses 0x0000 0000 through 0x0000 003F. A typical user program in the Flash memory can place the entire FIQ handler at address 0x0000 001C without any need to consider memory boundaries. The vector contained in the SRAM, external memory, and Boot Block must contain branches to the actual interrupt handlers, or to other instructions that accomplish the branch to the interrupt handlers.

There are three reasons this configuration was chosen:

1. To give the FIQ handler in the Flash memory the advantage of not having to take a memory boundary caused by the re-mapping into account.
2. Minimize the need to for the SRAM and Boot Block vectors to deal with arbitrary boundaries in the middle of code space.
3. To provide space to store constants for jumping beyond the range of single word branch instructions.

Re-mapped memory areas, including the Boot Block and interrupt vectors, continue to appear in their original location in addition to the re-mapped address.

Details on re-mapping and examples can be found in System Control Block on page 64.

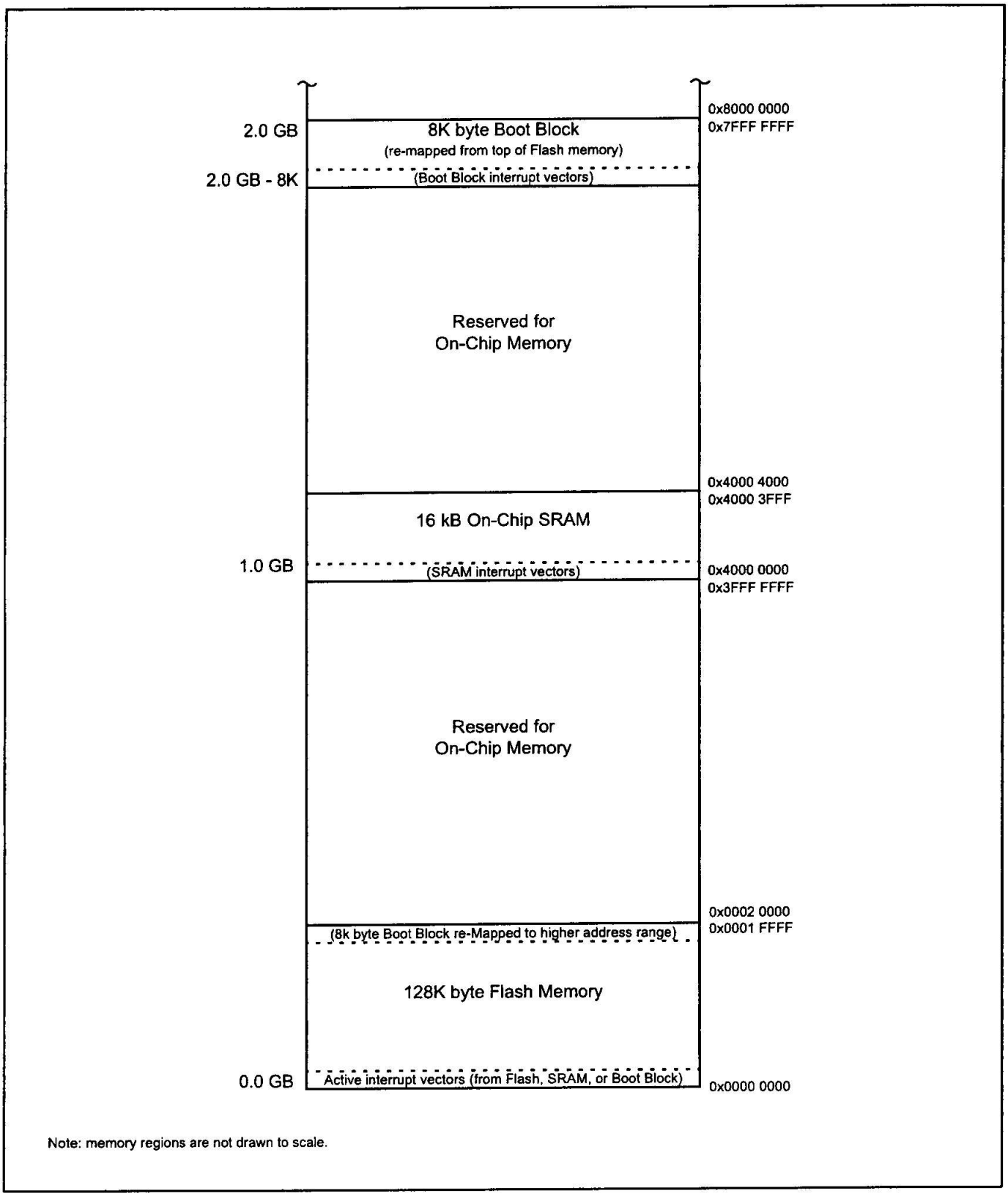


Figure 6: Map of lower memory is showing re-mapped and re-mappable areas (128 kB Flash).

PREFETCH ABORT AND DATA ABORT EXCEPTIONS

The LPC2119/2129/2194/2292/2294 generates the appropriate bus cycle abort exception if an access is attempted for an address that is in a reserved or unassigned address region. The regions are:

- Areas of the memory map that are not implemented for a specific ARM derivative. For the LPC2119/2129/2194/2292/2294, this is:
  - Address space between On-Chip Non-Volatile Memory and On-Chip SRAM, labelled "Reserved for On-Chip Memory" in Figure 2 and Figure 6. For 128 kB Flash device, this is memory address range from 0x0002 0000 to 0x3FFF FFFF, while for 256 kB Flash device this range is from 0x0004 0000 to 0x3FFF FFFF.
  - Address space between On-Chip Static RAM and External Memory. Labelled "Reserved for On-Chip Memory" in Figure 2. This is an address range from 0x4000 3FFF to 0x7FFF DFFF.
  - External Memory other than that provided by the EMC in the 144-pin package.
  - Reserved regions of the AHB and VPB spaces. See Figure 3.
- Unassigned AHB peripheral spaces. See Figure 4.
- Unassigned VPB peripheral spaces. See Figure 5.

For these areas, both attempted data access and instruction fetch generate an exception. In addition, a Prefetch Abort exception is generated for any instruction fetch that maps to an AHB or VPB peripheral address.

Within the address space of an existing VPB peripheral, a data abort exception is not generated in response to an access to an undefined address. Address decoding within each peripheral is limited to that needed to distinguish defined registers within the peripheral itself. For example, an access to address 0xE000D000 (an undefined address within the UART0 space) may result in an access to the register defined at address 0xE000C000. Details of such address aliasing within a peripheral space are not defined in the LPC2119/2129/2194/2292/2294 documentation and are not a supported feature.

Note that the ARM core stores the Prefetch Abort flag along with the associated instruction (which will be meaningless) in the pipeline and processes the abort only if an attempt is made to execute the instruction fetched from the illegal address. This prevents accidental aborts that could be caused by prefetches that occur when code is executed very near a memory boundary.

4. SYSTEM CONTROL BLOCK

SUMMARY OF SYSTEM CONTROL BLOCK FUNCTIONS

The System Control Block includes several system features and control registers for a number of functions that are not related to specific peripheral devices. These include:

- Crystal Oscillator.
- External Interrupt Inputs.
- Memory Mapping Control.
- PLL.
- Power Control.
- Reset.
- VPB Divider.
- Wakeup Timer.

Each type of function has its own register(s) if any are required and unneeded bits are defined as reserved in order to allow future expansion. Unrelated functions never share the same register addresses.

PIN DESCRIPTION

Table 11 shows pins that are associated with System Control block functions.

Table 11: Pin summary

Pin name	Pin direction	Pin Description
X1	Input	**Crystal Oscillator Input-** Input to the oscillator and internal clock generator circuits.
X2	Output	**Crystal Oscillator Output-** Output from the oscillator amplifier.
EINT0	Input	**External Interrupt Input 0-** An active low general purpose interrupt input. This pin may be used to wake up the processor from Idle or Power down modes. Pins P0.1 and P0.16 can be selected to perform EINT0 function. LOW level on this pin immediately after reset is considered as an external hardware request to start the ISP command handler. More details on ISP and Flash memory can be found in "Flash Memory System and Programming" chapter.
EINT1	Input	**External Interrupt Input 1-** See the EINT0 description above. Pins P0.3 and P0.14 can be selected to perform EINT1 function.
EINT2	Input	**External Interrupt Input 2-** See the EINT0 description above. Pins P0.7 and P0.15 can be selected to perform EINT2 function.
EINT3	Input	**External Interrupt Input 3-** See the EINT0 description above. Pins P0.9, P0.20 and P0.30 can be selected to perform EINT3 function.
$\overline{\text{RESET}}$	Input	**External Reset input-** A low on this pin resets the chip, causing I/O ports and peripherals to take on their default states, and the processor to begin execution at address 0.

REGISTER DESCRIPTION

All registers, regardless of size, are on word address boundaries. Details of the registers appear in the description of each function.

Table 12: Summary of System Control Registers

Name	Description	Access	Reset Value*	Address
External Interrupts				
EXTINT	External Interrupt Flag Register.	R/W	0	0xE01FC140
EXTWAKE	External Interrupt Wakeup Register.	R/W	0	0xE01FC144
EXTMODE	External Interrupt Mode Register.	R/W	0	0xE01FC148
EXTPOLAR	External Interrupt Polarity Register.	R/W	0	0xE01FC14C
Memory Mapping Control				
MEMMAP	Memory Mapping Control.	R/W	0	0xE01FC040
Phase Locked Loop				
PLLCON	PLL Control Register.	R/W	0	0xE01FC080
PLLCFG	PLL Configuration Register.	R/W	0	0xE01FC084
PLLSTAT	PLL Status Register.	RO	0	0xE01FC088
PLLFEED	PLL Feed Register.	WO	NA	0xE01FC08C
Power Control				
PCON	Power Control Register.	R/W	0	0xE01FC0C0
PCONP	Power Control for Peripherals.	R/W	0x3BE	0xE01FC0C4
VPB Divider				
VPBDIV	VPB Divider Control.	R/W	0	0xE01FC100

*Reset Value refers to the data stored in used bits only. It does not include reserved bits content.

CRYSTAL OSCILLATOR

While an input signal of 50-50 duty cycle within a frequency range from 1 MHz to 50 MHz can be used by LPC2119/2129/2194/2292/2294 if supplied to its input XTAL1 pin, this microcontroller's onboard oscillator circuit supports external crystals in the range of 1 MHz to 30 MHz only. If on-chip PLL system or boot-loader is used, input clock frequency is limited to exclusive range of 10 MHz to 25 MHz.

The oscillator output frequency is called F_{osc} and the ARM processor clock frequency is referred to as cclk for purposes of rate equations, etc. elsewhere in this document. F_{osc} and cclk are the same value unless the PLL is running and connected. Refer to the PLL description in this chapter for details and frequency limitations.

Onboard oscillator in LPC2119/2129/2194/2292/2294 can operate in one of two modes: slave mode and oscillation mode.

In slave mode the input clock signal should be coupled by means of a capacitor of 100 pF (Cc in Figure 12, drawing a), with an amplitude of at least 200 mVrms. X2 pin in this configuration can be left not connected. If slave mode is selected, F_{osc} signal of 50-50 duty cycle can range from 1 MHz to 50 MHz.

External components and models used in oscillation mode are shown in Figure 12, drawings b and c, and in Table 13. Since the feedback resistance is integrated on chip, only a crystal and the capacitances C_{X1} and C_{X2} need to be connected externally in case of fundamental mode oscillation (the fundamental frequency is represented by L, C_L and R_S). Capacitance Cp in Figure 12, drawing c, represents the parallel package capacitance and should not be larger than 7 pF. Parameters F_C, C_L, R_S and C_P are supplied by the crystal manufacturer.

Choosing an oscillation mode as an on-board oscillator mode of operation limits F_{osc} clock selection to 1 MHz to 30 MHz.

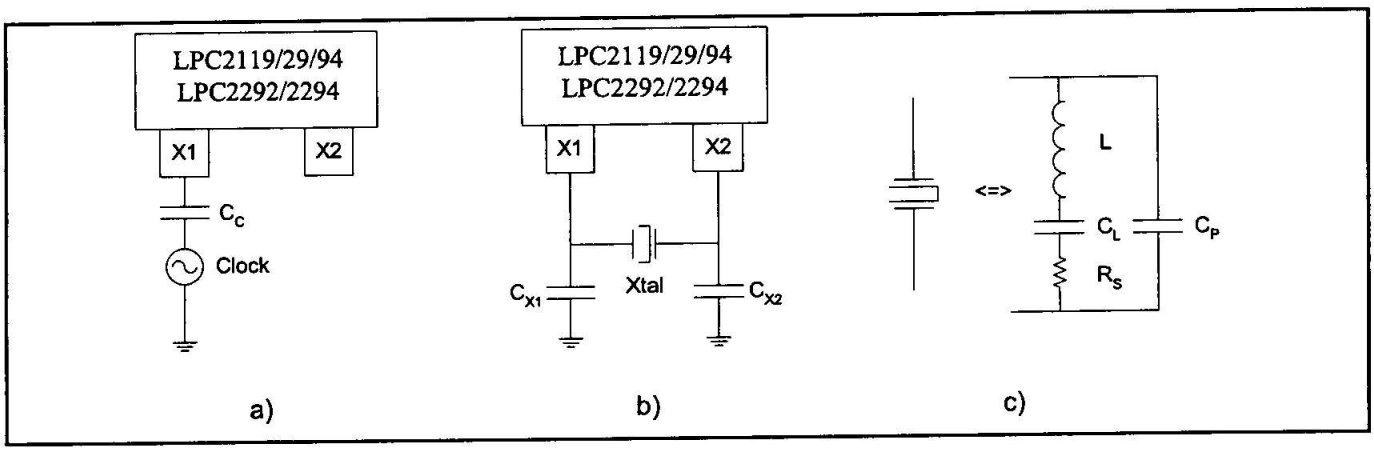


Figure 12: Oscillator modes and models: a) *slave mode* of operation, b) *oscillation mode* of operation, c) external crystal model used for $C_{X1/X2}$ evaluation

Table 13: Recommended values for $C_{X1/X2}$ in oscillation mode (crystal and external components parameters)

Fundamental Oscillation Frequency F_C	Crystal Load Capacitance C_L	Max. Crystal Series Resistence R_S	External Load Capacitors C_{X1}, C_{X2}
1 - 5 MHz	10 pF	n.a.	n.a.
	20 pF	n.a.	n.a.
	30 pF	< 300 Ω	58 pF, 58 pF

Table 13: Recommended values for $C_{X1/X2}$ in oscillation mode (crystal and external components parameters)

Fundamental Oscillation Frequency F_C	Crystal Load Capacitance C_L	Max. Crystal Series Resistence R_S	External Load Capacitors C_{X1}, C_{X2}
5 - 10 MHz	10 pF	< 300 Ω	18 pF, 18 pF
	20 pF	< 300 Ω	38 pF, 38 pF
	30 pF	< 300 Ω	58 pF, 58 pF
10 - 15 MHz	10 pF	< 300 Ω	18 pF, 18 pF
	20 pF	< 220 Ω	38 pF, 38 pF
	30 pF	< 140 Ω	58 pF, 58 pF
15 - 20 MHz	10 pF	< 220 Ω	18 pF, 18 pF
	20 pF	< 140 Ω	38 pF, 38 pF
	30 pF	< 80 Ω	58 pF, 58 pF
20 - 25 MHz	10 pF	< 160 Ω	18 pF, 18 pF
	20 pF	< 90 Ω	38 pF, 38 pF
	30 pF	< 50 Ω	58 pF, 58 pF
25 - 30 MHz	10 pF	<130 Ω	18 pF, 18 pF
	20 pF	<50 Ω	38 pF, 38 pF
	30 pF	n.a.	n.a.

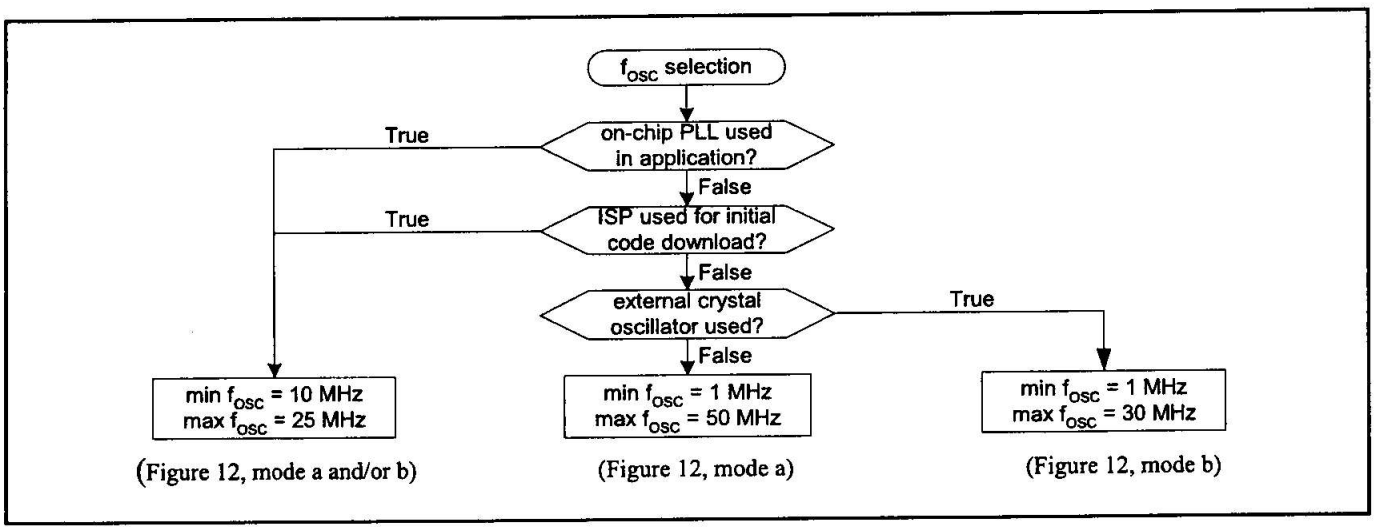


Figure 13: F_{OSC} selection algorithm

EXTERNAL INTERRUPT INPUTS

The LPC2119/2129/2194/2292/2294 includes four External Interrupt Inputs as selectable pin functions. The External Interrupt Inputs can optionally be used to wake up the processor from the Power Down mode.

Register Description

The external interrupt function has four registers associated with it. The EXTINT register contains the interrupt flags, and the EXTWAKEUP register contains bits that enable individual external interrupts to wake up the LPC2119/2129/2292/2294 from Power Down mode. The EXTMODE and EXTPOLAR registers specify the level and edge sensitivity parameters.

Table 14: External Interrupt Registers

Address	Name	Description	Access
0xE01FC140	EXTINT	The External Interrupt Flag Register contains interrupt flags for EINT0, EINT1, and EINT2. See Table 15.	R/W
0xE01FC144	EXTWAKE	The External Interrupt Wakeup Register contains three enable bits that control whether each external interrupt will cause the processor to wake up from Power Down mode. See Table 16.	R/W
0xE01FC148	EXTMODE	The External Interrupt Mode Register controls whether each pin is edge- or level-sensitive.	R/W
0xE01FC14C	EXTPOLAR	The External Interrupt Polarity Register controls which level or edge on each pin will cause an interrupt.	R/W

External Interrupt Flag Register (EXTINT - 0xE01FC140)

When a pin is selected for its external interrupt function, the level or edge on that pin selected by its bits in the EXTPOLAR and EXTMODE registers will set its interrupt flag in this register. This asserts the corresponding interrupt request to the VIC, which will cause an intrerrupt if interrupts from the pin are enabled.

Writing ones to bits EINT0 through EINT3 in EXTINT register clears the corresponding bits. In level-sensitive mode this action is efficacious only when the pin is in its innactive state.

Table 15: External Interrupt Flag Register (EXTINT - 0xE01FC140)

EXTINT	Function	Description	Reset Value
0	EINT0	In level-sensitive mode, this bit is set if the EINT0 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT0 function is selected for its pin, and the selected edge occurs on the pin. Up to two pins can be selected to perform EINT0 function (see P0.1 and P0.16 description in "Pin Configuration" chapter.) This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state.	0
1	EINT1	In level-sensitive mode, this bit is set if the EINT1 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT1 function is selected for its pin, and the selected edge occurs on the pin. Up to two pins can be selected to perform EINT1 function (see P0.3 and P0.14 description in "Pin Configuration" chapter.) This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state.	0
2	EINT2	In level-sensitive mode, this bit is set if the EINT2 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT2 function is selected for its pin, and the selected edge occurs on the pin. Up to two pins can be selected to perform EINT2 function (see P0.7 and P0.15 description in "Pin Configuration" chapter.) This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state.	0
3	EINT3	In level-sensitive mode, this bit is set if the EINT3 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT3 function is selected for its pin, and the selected edge occurs on the pin. Up to three pins can be selected to perform EINT3 function (see P0.9, P0.20 and P0.30 description in "Pin Configuration" chapter.) This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state.	0
7:4	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

External Interrupt Wakeup Register (EXTWAKE - 0xE01FC144)

Enable bits in the EXTWAKE register allow the external interrupts to wake up the processor if it is in Power Down mode. The related EINTn function must be mapped to the pin in order for the wakeup process to take place. It is not necessary for the interrupt to be enabled in the Vectored Interrupt Controller for a wakeup to take place. This arrangement allows additional capabilities, such as having an external interrupt input wake up the processor from Power Down mode without causing an interrupt (simply resuming operation), or allowing an interrupt to be enabled during Power Down without waking the processor up if it is asserted (eliminating the need to disable the interrupt if the wakeup feature is not desirable in the application).

Table 16: External Interrupt Wakeup Register (EXTWAKE - 0xE01FC144)

EXTWAKE	Function	Description	Reset Value
0	EXTWAKE0	When one, assertion of $\overline{\text{EINT0}}$ will wake up the processor from Power Down mode.	0
1	EXTWAKE1	When one, assertion of $\overline{\text{EINT1}}$ will wake up the processor from Power Down mode.	0
2	EXTWAKE2	When one, assertion of $\overline{\text{EINT2}}$ will wake up the processor from Power Down mode.	0
3	EXTWAKE3	When one, assertion of $\overline{\text{EINT3}}$ will wake up the processor from Power Down mode.	0
7:4	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

External Interrupt Mode Register (EXTMODE - 0xE01FC148)

The bits in this register select whether each EINT pin is level- or edge-sensitive. Only pins that are selected for the EINT function (chapter Pin Connect Block on page 126) and enabled via the VICIntEnable register (chapter Vectored Interrupt Controller (VIC) on page 96) can cause interrupts from the External Interrupt function (though of course pins selected for) other functions may cause interrupts from those functions).

Note: Software should only change a bit in this register when its interrupt is disabled in VICIntEnable, and should write the corresponding 1 to EXTINT before re-enabling the interrupt, to clear the EXTINT bit that could be set by changing the mode.

Table 17: External Interrupt Mode Register (EXTMODE - 0xE01FC148)

EXTMODE	Function	Description	Reset Value
0	EXTMODE0	When 0, level-sensitivity is selected for EINT0. When 1, EINT0 is edge-sensitive.	0
1	EXTMODE1	When 0, level-sensitivity is selected for EINT1. When 1, EINT1 is edge-sensitive.	0
2	EXTMODE2	When 0, level-sensitivity is selected for EINT2. When 1, EINT2 is edge-sensitive.	0
3	EXTMODE3	When 0, level-sensitivity is selected for EINT3. When 1, EINT3 is edge-sensitive.	0
7:4	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

External Interrupt Polarity Register (EXTPOLAR - 0xE01FC14C)

In level-sensitive mode, the bits in this register select whether the corresponding pin is high- or low-active. In edge-sensitive mode, they select whether the pin is rising- or falling-edge sensitive. Only pins that are selected for the EINT function (chapter Pin Connect Block on page 126) and enabled in the VICIntEnable register (chapter Vectored Interrupt Controller (VIC) on page 96) can cause interrupts from the External Interrupt function (though of course pins selected for other functions may cause interrupts from those functions).

Note: Software should only change a bit in this register when its interrupt is disabled in VICIntEnable, and should write the corresponding 1 to EXTINT before re-enabling the interrupt, to clear the EXTINT bit that could be set by changing the polarity.

Table 18: External Interrupt Polarity Register (EXTPOLAR - 0xE01FC14C)

EXTPOLAR	Function	Description	Reset Value
0	EXTPOLAR0	When 0, EINT0 is low-active or falling-edge sensitive (depending on EXTMODE0). When 1, EINT0 is high-active or rising-edge sensitive (depending on EXTMODE0).	0
1	EXTPOLAR1	When 0, EINT1 is low-active or falling-edge sensitive (depending on EXTMODE1). When 1, EINT1 is high-active or rising-edge sensitive (depending on EXTMODE1).	0
2	EXTPOLAR2	When 0, EINT2 is low-active or falling-edge sensitive (depending on EXTMODE2). When 1, EINT2 is high-active or rising-edge sensitive (depending on EXTMODE2).	0
3	EXTPOLAR3	When 0, EINT3 is low-active or falling-edge sensitive (depending on EXTMODE3). When 1, EINT3 is high-active or rising-edge sensitive (depending on EXTMODE3).	0
7:4	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Multiple External Interrupt Pins

Software can select multiple pins for each of EINT3:0 in the Pin Select registers, which are described in chapter Pin Connect Block on page 126. The external interrupt logic for each of EINT3:0 receives the state of all of its associated pins from the pins' receivers, along with signals that indicate whether each pin is selected for the EINT function. The external interrupt logic handles the case when more than one pin is so selected, differently according to the state of its Mode and Polarity bits:

- In Low-Active Level Sensitive mode, the states of all pins selected for EINT functionality are digitally combined using a positive logic AND gate.
- In High-Active Level Sensitive mode, the states of all pins selected for EINT functionality are digitally combined using a positive logic OR gate.
- In Edge Sensitive mode, regardless of polarity, the pin with the lowest GPIO port number is used. (Selecting multiple EINT pins in edge-sensitive mode could be considered a programming error.)

The signal derived by this logic is the EINTi signal in the following logic schematic (Figure 14).

When more than one EINT pin is logically ORed, the interrupt service routine can read the states of the pins from GPIO port using IO0PIN0 and IO1PIN registers, to determine which pin(s) caused the interrupt.

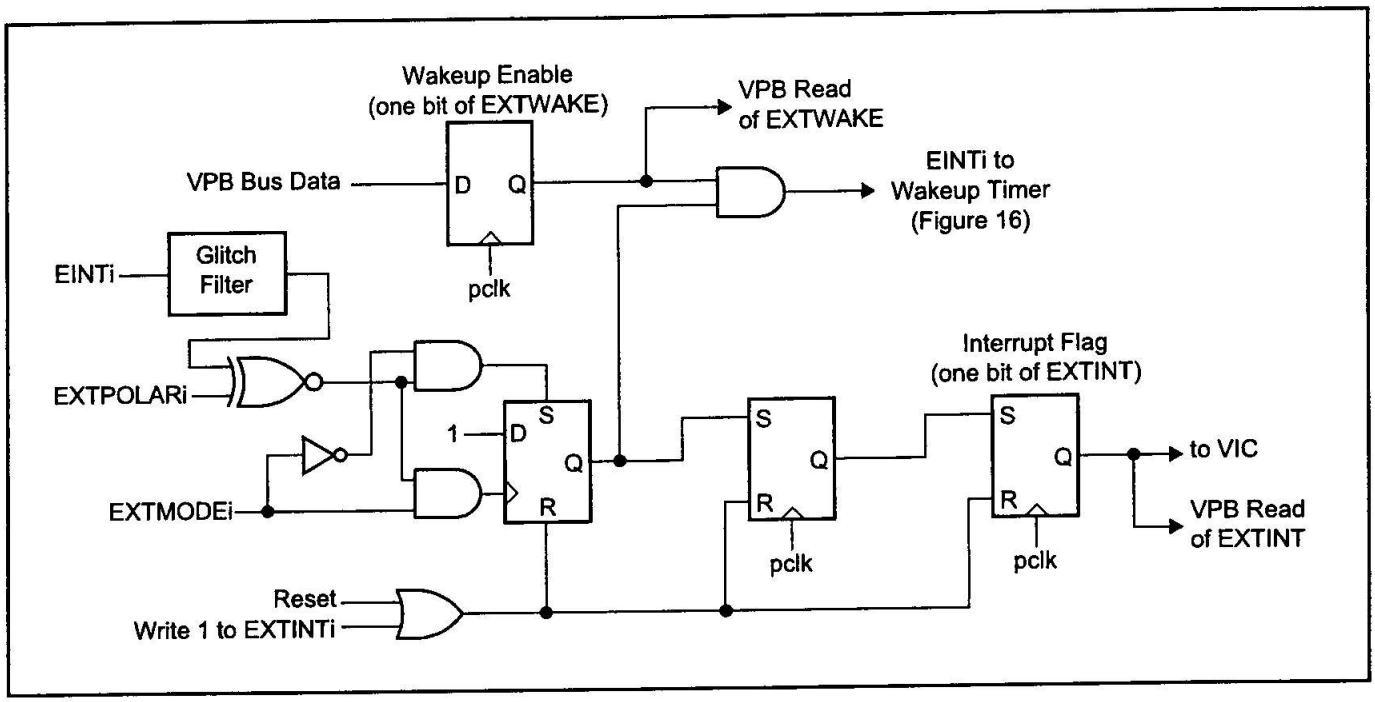


Figure 14: External Interrupt Logic

MEMORY MAPPING CONTROL

The Memory Mapping Control alters the mapping of the interrupt vectors that appear beginning at address 0x00000000. This allows code running in different memory spaces to have control of the interrupts.

Memory Mapping Control Register (MEMMAP - 0xE01FC040)

Table 19: MEMMAP Register

Address	Name	Description	Access
0xE01FC040	MEMMAP	Memory mapping control. Selects whether the ARM interrupt vectors are read from the Flash Boot Block, User Flash or RAM.	R/W

Table 20: Memory Mapping Control Register (MEMMAP - 0xE01FC040)

MEMMAP	Function	Description	Reset Value*
1:0	MAP1:0	00: Boot Loader Mode. Interrupt vectors are re-mapped to Boot Block. 01: User Flash Mode. Interrupt vectors are not re-mapped and reside in Flash. 10: User RAM Mode. Interrupt vectors are re-mapped to Static RAM. 11: User External memory Mode. Interrupt vectors are re-mapped to external memory. This mode is available in L2292/2294 only and must not be specified when LPC2119/2129/2194 are used. **Warning:** Improper setting of this value may result in incorrect operation of the device.	0
7:2	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

*: The hardware reset value of the MAP bits is 00 for LPC2119/2129/2194/2292/2294 parts. The apparent reset value that the user will see will be altered by the Boot Loader code, which always runs initially at reset. User documentation will reflect this difference.

Memory Mapping Control Usage Notes

Memory Mapping Control simply selects one out of three available sources of data (sets of 64 bytes each) necessary for handling ARM exceptions (interrupts).

For example, whenever a Software Interrupt request is generated, ARM core will always fetch 32-bit data "residing" on 0x0000 0008 (see Table 3, "ARM Exception Vector Locations," on page 52). This means that when MEMMAP[1:0]=10 (User RAM Mode), read/fetch from 0x0000 0008 will provide data stored in 0x4000 0008. If MEMMAP[1:0]=01 (User Flash Mode), read/fetch from 0x0000 0008 will provide data stored in on-chip Flash location 0x0000 0008. In case of MEMMAP[1:0]=00 (Boot Loader Mode), read/fetch from 0x0000 0008 will provide data availble also at 0x7FFF E008 (Boot Block remapped from on-chip Flash memory).

PLL (PHASE LOCKED LOOP)

The PLL accepts an input clock frequency in the range of 10 MHz to 25 MHz only. The input frequency is multiplied up into the cclk with the range of 10 MHz to 60 MHz using a Current Controlled Oscillator (CCO). The multiplier can be an integer value from 1 to 32 (in practice, the multiplier value cannot be higher than 6 on the LPC2119/2129/2194/2292/2294 due to the upper frequency limit of the CPU). The CCO operates in the range of 156 MHz to 320 MHz, so there is an additional divider in the loop to keep the CCO within its frequency range while the PLL is providing the desired output frequency. The output divider may be set to divide by 2, 4, 8, or 16 to produce the output clock. Since the minimum output divider value is 2, it is insured that the PLL output has a 50% duty cycle. A block diagram of the PLL is shown in Figure 15.

PLL activation is controlled via the PLLCON register. The PLL multiplier and divider values are controlled by the PLLCFG register. These two registers are protected in order to prevent accidental alteration of PLL parameters or deactivation of the PLL. Since all chip operations, including the Watchdog Timer, are dependent on the PLL when it is providing the chip clock, accidental changes to the PLL setup could result in unexpected behavior of the microcontroller. The protection is accomplished by a feed sequence similar to that of the Watchdog Timer. Details are provided in the description of the PLLFEED register.

The PLL is turned off and bypassed following a chip Reset and when by entering power Down mode. PLL is enabled by software only. The program must configure and activate the PLL, wait for the PLL to Lock, then connect to the PLL as a clock source.

Register Description

The PLL is controlled by the registers shown in Table 21. More detailed descriptions follow.

Warning: Improper setting of PLL values may result in incorrect operation of the device.

Table 21: PLL Registers

Address	Name	Description	Access
0xE01FC080	PLLCON	PLL Control Register. Holding register for updating PLL control bits. Values written to this register do not take effect until a valid PLL feed sequence has taken place.	R/W
0xE01FC084	PLLCFG	PLL Configuration Register. Holding register for updating PLL configuration values. Values written to this register do not take effect until a valid PLL feed sequence has taken place.	R/W
0xE01FC088	PLLSTAT	PLL Status Register. Read-back register for PLL control and configuration information. If PLLCON or PLLCFG have been written to, but a PLL feed sequence has not yet occurred, they will not reflect the current PLL state. Reading this register provides the actual values controlling the PLL, as well as the status of the PLL.	RO
0xE01FC08C	PLLFEED	PLL Feed Register. This register enables loading of the PLL control and configuration information from the PLLCON and PLLCFG registers into the shadow registers that actually affect PLL operation.	WO

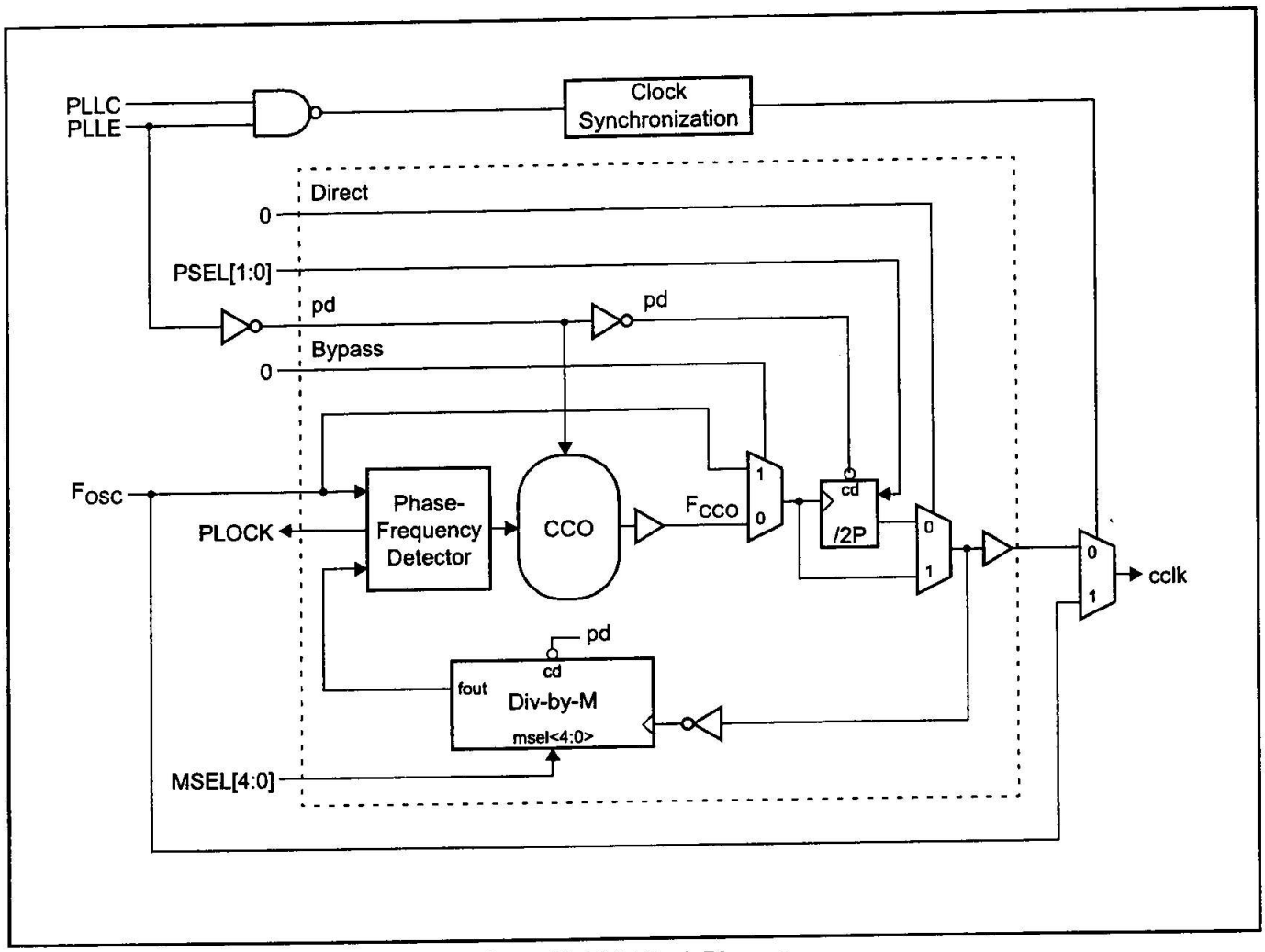


Figure 15: PLL Block Diagram

PLL Control Register (PLLCON - 0xE01FC080)

The PLLCON register contains the bits that enable and connect the PLL. Enabling the PLL allows it to attempt to lock to the current settings of the multiplier and divider values. Connecting the PLL causes the processor and all chip functions to run from the PLL output clock. Changes to the PLLCON register do not take effect until a correct PLL feed sequence has been given (see PLL Feed Register (PLLFEED - 0xE01FC08C) description).

Table 22: PLL Control Register (PLLCON - 0xE01FC080)

PLLCON	Function	Description	Reset Value
0	PLLE	PLL Enable. When one, and after a valid PLL feed, this bit will activate the PLL and allow it to lock to the requested frequency. See PLLSTAT register, Table 24.	0
1	PLLC	PLL Connect. When PLLC and PLLE are both set to one, and after a valid PLL feed, connects the PLL as the clock source for the LPC2119/2129/2194/2292/2294. Otherwise, the oscillator clock is used directly by the LPC2119/2129/2194/2292/2294. See PLLSTAT register, Table 24.	0
7:2	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

The PLL must be set up, enabled, and Lock established before it may be used as a clock source. When switching from the oscillator clock to the PLL output or vice versa, internal circuitry synchronizes the operation in order to ensure that glitches are not generated. Hardware does not insure that the PLL is locked before it is connected or automatically disconnect the PLL if lock is lost during operation. In the event of loss of PLL lock, it is likely that the oscillator clock has become unstable and disconnecting the PLL will not remedy the situation.

PLL Configuration Register (PLLCFG - 0xE01FC084)

The PLLCFG register contains the PLL multiplier and divider values. Changes to the PLLCFG register do not take effect until a correct PLL feed sequence has been given (see PLL Feed Register (PLLFEED - 0xE01FC08C) description). Calculations for the PLL frequency, and multiplier and divider values are found in the PLL Frequency Calculation section.

Table 23: PLL Configuration Register (PLLCFG - 0xE01FC084)

PLLCFG	Function	Description	Reset Value
4:0	MSEL4:0	PLL Multiplier value. Supplies the value "M" in the PLL frequency calculations. **Note:** For details on selecting the right value for MSEL4:0 see section "PLL Frequency Calculation" on page 79.	0
6:5	PSEL1:0	PLL Divider value. Supplies the value "P" in the PLL frequency calculations. **Note:** For details on selecting the right value for PSEL1:0 see section "PLL Frequency Calculation" on page 79.	0
7	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

PLL Status Register (PLLSTAT - 0xE01FC088)

The read-only PLLSTAT register provides the actual PLL parameters that are in effect at the time it is read, as well as the PLL status. PLLSTAT may disagree with values found in PLLCON and PLLCFG because changes to those registers do not take effect until a proper PLL feed has occurred (see PLL Feed Register (PLLFEED - 0xE01FC08C) description).

Table 24: PLL Status Register (PLLSTAT - 0xE01FC088)

PLLSTAT	Function	Description	Reset Value
4:0	MSEL4:0	Read-back for the PLL Multiplier value. This is the value currently used by the PLL.	0
6:5	PSEL1:0	Read-back for the PLL Divider value. This is the value currently used by the PLL.	0
7	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
8	PLLE	Read-back for the PLL Enable bit. When one, the PLL is currently activated. When zero, the PLL is turned off. This bit is automatically cleared when Power Down mode is activated.	0
9	PLLC	Read-back for the PLL Connect bit. When PLLC and PLLE are both one, the PLL is connected as the clock source for the LPC2119/2129/2194/2292/2294. When either PLLC or PLLE is zero, the PLL is bypassed and the oscillator clock is used directly by the LPC2119/2129/2194/2292/2294. This bit is automatically cleared when Power Down mode is activated.	0
10	PLOCK	Reflects the PLL Lock status. When zero, the PLL is not locked. When one, the PLL is locked onto the requested frequency.	0
15:11	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

PLL Interrupt

The PLOCK bit in the PLLSTAT register is connected to the interrupt controller. This allows for software to turn on the PLL and continue with other functions without having to wait for the PLL to achieve lock. When the interrupt occurs (PLOCK = 1), the PLL may be connected, and the interrupt disabled.

PLL Modes

The combinations of PLLE and PLLC are shown in Table 25.

Table 25: PLL Control Bit Combinations

PLLC	PLLE	PLL Function
0	0	PLL is turned off and disconnected. The system runs from the unmodified clock input.
0	1	The PLL is active, but not yet connected. The PLL can be connected after PLOCK is asserted.
1	0	Same as 0 0 combination. This prevents the possibility of the PLL being connected without also being enabled.
1	1	The PLL is active and has been connected as the system clock source.

PLL Feed Register (PLLFEED - 0xE01FC08C)

A correct feed sequence must be written to the PLLFEED register in order for changes to the PLLCON and PLLCFG registers to take effect. The feed sequence is:

1. Write the value 0xAA to PLLFEED
2. Write the value 0x55 to PLLFEED.

The two writes must be in the correct sequence, and must be consecutive VPB bus cycles. The latter requirement implies that interrupts must be disabled for the duration of the PLL feed operation. If either of the feed values is incorrect, or one of the previously mentioned conditions is not met, any changes to the PLLCON or PLLCFG register will not become effective.

Table 26: PLL Feed Register (PLLFEED - 0xE01FC08C)

PLLFEED	Function	Description	Reset Value
7:0	PLLFEED	The PLL feed sequence must be written to this register in order for PLL configuration and control register changes to take effect.	undefined

PLL and Power Down Mode

Power Down mode automatically turns off and disconnects the PLL. Wakeup from Power Down mode does not automatically restore the PLL settings, this must be done in software. Typically, a routine to activate the PLL, wait for lock, and then connect the PLL can be called at the beginning of any interrupt service routine that might be called due to the wakeup. It is important not to attempt to restart the PLL by simply feeding it when execution resumes after a wakeup from Power Down mode. This would enable and connect the PLL at the same time, before PLL lock is established.

PLL Frequency Calculation

The PLL equations use the following parameters:

F_{OSC}	the frequency from the crystal oscillator
F_{CCO}	the frequency of the PLL current controlled oscillator
cclk	the PLL output frequency (also the processor clock frequency)
M	PLL Multiplier value from the MSEL bits in the PLLCFG register
P	PLL Divider value from the PSEL bits in the PLLCFG register

The PLL output frequency (when the PLL is both active and connected) is given by:

$$cclk = M * F_{osc} \quad \text{or} \quad cclk = \frac{F_{cco}}{2 * P}$$

The CCO frequency can be computed as:

$$F_{cco} = cclk * 2 * P \quad \text{or} \quad F_{cco} = F_{osc} * M * 2 * P$$

The PLL inputs and settings must meet the following:

- F_{osc} is in the range of 10 MHz to 25 MHz.
- cclk is in the range of 10 MHz to F_{max} (the maximum allowed frequency for the LPC2119/2129/2194/2292/2294).
- F_{cco} is in the range of 156 MHz to 320 MHz.

Procedure for Determining PLL Settings

If a particular application uses the PLL, its configuration may be determined as follows:

1. Choose the desired processor operating frequency (cclk). This may be based on processor throughput requirements, need to support a specific set of UART baud rates, etc. Bear in mind that peripheral devices may be running from a lower clock than the processor (see the VPB Divider description in this chapter).
2. Choose an oscillator frequency (F_{osc}). cclk must be the whole (non-fractional) multiple of F_{osc}.
3. Calculate the value of M to configure the MSEL bits. M = cclk / F_{osc}. M must be in the range of 1 to 32. The value written to the MSEL bits in PLLCFG is M - 1 (see Table 28).
4. Find a value for P to configure the PSEL bits, such that F_{cco} is within its defined frequency limits. F_{cco} is calculated using the equation given above. P must have one of the values 1, 2, 4, or 8. The value written to the PSEL bits in PLLCFG is 00 for P = 1; 01 for P = 2; 10 for P = 4; 11 for P = 8 (see Table 27).

Table 27: PLL Divider Values

PSEL Bits (PLLCFG bits 6:5)	Value of P
00	1
01	2
10	4
11	8

Table 28: PLL Multiplier Values

MSEL Bits (PLLCFG bits 4:0)	Value of M
00000	1
00001	2
00010	3
00011	4
...	...
11110	31
11111	32

PLL Example

System design asks for F_{osc}= 10 MHz and requires cclk = 60 MHz.

Based on these specifications, M = cclk / F_{osc} = 60 MHz / 10 MHz = 6. Consequenty, M-1 = 5 will be written as PLLCFG 4:0.

Value for P can be derived from P = F_{cco} / (cclk * 2), using condition that F_{cco} must be in range of 156 MHz to 320 MHz. Assuming the lowest allowed frequency for F_{cco} = 156 MHz, P = 156 MHz / (2*60 MHz) = 1.3. The highest F_{cco} frequency criteria produces P = 2.67. The only solution for P that satisfies both of these requirements and is listed in Table 27 is P = 2. Therefore, PLLCFG 6:5 = 1 will be used.

POWER CONTROL

The LPC2119/2129/2194/2292/2294 supports two reduced power modes: Idle mode and Power Down mode. In Idle mode, execution of instructions is suspended until either a Reset or interrupt occurs. Peripheral functions continue operation during Idle mode and may generate interrupts to cause the processor to resume execution. Idle mode eliminates power used by the processor itself, memory systems and related controllers, and internal buses.

In Power Down mode, the oscillator is shut down and the chip receives no internal clocks. The processor state and registers, peripheral registers, and internal SRAM values are preserved throughout Power Down mode and the logic levels of chip pins remain static. The Power Down mode can be terminated and normal operation resumed by either a Reset or certain specific interrupts that are able to function without clocks. Since all dynamic operation of the chip is suspended, Power Down mode reduces chip power consumption to nearly zero.

Entry to Power Down and Idle modes must be coordinated with program execution. Wakeup from Power Down or Idle modes via an interrupt resumes program execution in such a way that no instructions are lost, incomplete, or repeated. Wake up from Power Down mode is discussed further in the description of the Wakeup Timer later in this chapter.

A Power Control for Peripherals feature allows individual peripherals to be turned off if they are not needed in the application, resulting in additional power savings.

Register Description

The Power Control function contains two registers, as shown in Table 29. More detailed descriptions follow.

Table 29: Power Control Registers

Address	Name	Description	Access
0xE01FC0C0	PCON	Power Control Register. This register contains control bits that enable the two reduced power operating modes of the LPC2119/2129/2194/2292/2294. See Table 30.	R/W
0xE01FC0C4	PCONP	Power Control for Peripherals Register. This register contains control bits that enable and disable individual peripheral functions, Allowing elimination of power consumption by peripherals that are not needed.	R/W

Power Control Register (PCON - 0xE01FC0C0)

The PCON register contains two bits. Writing a one to the corresponding bit causes entry to either the Power Down or Idle mode. If both bits are set, Power Down mode is entered.

Table 30: Power Control Register (PCON - 0xE01FC0C0)

PCON	Function	Description	Reset Value
0	IDL	Idle mode - when 1, this bit causes the processor clock to be stopped, while on-chip peripherals remain active. Any enabled interrupt from a peripheral or an external interrupt source will cause the processor to resume execution.	0
1	PD	Power Down mode - when 1, this bit causes the oscillator and all on-chip clocks to be stopped. A wakeup condition from an external interrupt can cause the oscillator to re-start, the PD bit to be cleared, and the processor to resume execution.	0
7:2	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Power Control for Peripherals Register (PCONP - 0xE01FC0C4)

The PCONP register allows turning off selected peripheral functions for the purpose of saving power. A few peripheral functions cannot be turned off (i.e. the Watchdog timer, GPIO, the Pin Connect block, and the System Control block). Each bit in PCONP controls one of the peripherals. The bit numbers correspond to the related peripheral number as shown in the VPB peripheral map in the LPC2119/2129/2292/2294 Memory Addressing section.

Table 31: Power Control for Peripherals Register for LPC2119/2129/2292 (PCONP - 0xE01FC0C4)

PCONP	Function	Description	Reset Value
0	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0
1	PCTIM0	When 1, TIMER0 is enabled. When 0, TIMER0 is disabled to conserve power.	1
2	PCTIM1	When 1, TIMER1 is enabled. When 0, TIMER1 is disabled to conserve power.	1
3	PCURT0	When 1, UART0 is enabled. When 0, UART0 is disabled to conserve power.	1
4	PCURT1	When 1, UART1 is enabled. When 0, UART1 is disabled to conserve power.	1
5	PCPWM0	When 1, PWM0 is enabled. When 0, PWM0 is disabled to conserve power.	1
6	Reserved	User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0
7	PCI2C	When 1, the I^2C interface is enabled. When 0, the I^2C interface is disabled to conserve power.	1
8	PCSPI0	When 1, the SPI0 interface is enabled. When 0, the SPI0 is disabled to conserve power.	1
9	PCRTC	When 1, the RTC is enabled. When 0, the RTC is disabled to conserve power.	1
10	PCSPI1	When 1, the SPI1 interface is enabled. When 0, the SPI1 is disabled to conserve power.	1
11	Reserved	User software should write 0 here to reduce power consumption.	1

Table 31: Power Control for Peripherals Register for LPC2119/2129/2292 (PCONP - 0xE01FC0C4)

PCONP	Function	Description	Reset Value
12	PCAD	When 1, the A/D converter is enabled. When 0, the A/D is disabled to conserve power.	1
13	PCCAN1	When 1, CAN Controller 1 is enabled. When 0, it is disabled to save power. **Note:** the Acceptance Filter is enabled if any of CAN Controllers 1-2 is enabled.	1
14	PCCAN2	When 1, CAN Controller 2 is enabled. When 0, it is disabled to save power.	1
31:15	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 32: Power Control for Peripherals Register for LPC2194/2294 (PCONP - 0xE01FC0C4)

PCONP	Function	Description	Reset Value
0	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0
1	PCTIM0	When 1, TIMER0 is enabled. When 0, TIMER0 is disabled to conserve power.	1
2	PCTIM1	When 1, TIMER1 is enabled. When 0, TIMER1 is disabled to conserve power.	1
3	PCURT0	When 1, UART0 is enabled. When 0, UART0 is disabled to conserve power.	1
4	PCURT1	When 1, UART1 is enabled. When 0, UART1 is disabled to conserve power.	1
5	PCPWM0	When 1, PWM0 is enabled. When 0, PWM0 is disabled to conserve power.	1
6	Reserved	User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0
7	PCI2C	When 1, the I^2C interface is enabled. When 0, the I^2C interface is disabled to conserve power.	1
8	PCSPI0	When 1, the SPI0 interface is enabled. When 0, the SPI0 is disabled to conserve power.	1
9	PCRTC	When 1, the RTC is enabled. When 0, the RTC is disabled to conserve power.	1
10	PCSPI1	When 1, the SPI1 interface is enabled. When 0, the SPI1 is disabled to conserve power.	1
11	PCEMC	When 1, the External Memory Controller is enabled. When 0, the EMC is disabled to conserve power.	1
12	PCAD	When 1, the A/D converter is enabled. When 0, the A/D is disabled to conserve power.	1
13	PCCAN1	When 1, CAN Controller 1 is enabled. When 0, it is disabled to save power.	1
14	PCCAN2	When 1, CAN Controller 2 is enabled. When 0, it is disabled to save power.	1
15	PCCAN3	When 1, CAN Controller 3 is enabled. When 0, it is disabled to save power.	1
16	PCCAN4	When 1, CAN Controller 4 is enabled. When 0, it is disabled to save power.	1
31:17	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

POWER CONTROL USAGE NOTES

After every reset, PCONP register contains the value that enables all interfaces and peripherals controlled by the PCONP to be enabled. Therefore, apart from proper configuring via peripheral dedicated registers, user's application has no need to access the PCONP in order to start using any of the on-board peripherals.

Power saving oriented systems should have 1s in the PCONP register only in positions that match peripherals really used in the application. All other bits, declared to be "Reserved" or dedicated to the peripherals not used in the current application, must be cleared to 0.

RESET

Reset has two sources on the LPC2119/2129/2194/2292/2294: the $\overline{\text{RESET}}$ pin and Watchdog Reset. The $\overline{\text{RESET}}$ pin is a Schmitt trigger input pin with an additional glitch filter. Assertion of chip Reset by any source starts the Wakeup Timer (see Wakeup Timer description later in this chapter), causing reset to remain asserted until the external Reset is de-asserted, the oscillator is running, a fixed number of clocks have passed, and the Flash controller has completed its initialization. The relationship between Reset, the oscillator, and the Wakeup Timer are shown in Figure 16.

The Reset glitch filter allows the processor to ignore external reset pulses that are very short, and also determines the minimum duration of $\overline{\text{RESET}}$ that must be asserted in order to guarantee a chip reset. Once asserted, $\overline{\text{RESET}}$ pin can be deasserted only when crystal oscillator is fully running and an adequate signal is present on the X1 pin of the LPC2119/2129/2194/2292/2294. Assuming that an external crystal is used in the crystal oscillator subsystem, after power on, the $\overline{\text{RESET}}$ pin should be asserted for 10 ms. For all subsequent resets when crystal osillator is already running and stable signal is on the X1 pin, the $\overline{\text{RESET}}$ pin needs to be asserted for 300 ns only.

Speaking in general, there are no sequence requirements for powering up the supplies (V_{18}, V_3, V_{18A} and V_{3A}). However, for proper reset handling It is absolutely necessary to have valid voltage supply on V_{18} pins, since on-chip Reset circuit and oscillator dedicated hardware are powered by them. V_3 pins enable microcontroller's interface to the environment via its digital pins. Consequently, not providing V_3 power supply will not affect the reset sequence itself, but will prevent microcontroller from communicating with external world.

When the internal Reset is removed, the processor begins executing at address 0, which isinitially the Reset vector mapped from the Boot Block. At that point, all of the processor and peripheral registers have been initialized to predetermined values.

External and internal Resets have some small differences. An external Reset causes the value of certain pins to be latched to configure the part. External circuitry cannot determine when an internal Reset occurs in order to allow setting up those special pins, so those latches are not reloaded during an internal Reset. Pins that are examined during an external Reset for various purposes are: P1.20/TRACESYNC, P1.26/RTCK, BOOT1 and BOOT0 (see chapters Pin Configuration on page 110, Pin Connect Block on page 126 and External Memory Controller (EMC) on page 56). Pin P0.14 (see Flash Memory System and Programming on page 262) is exemined by on-chip bootloader when this code is executed after reset.

It is possible for a chip Reset to occur during a Flash programming or erase operation. The Flash memory will interrupt the ongoing operation and hold off the completion of Reset to the CPU until internal Flash high voltages have settled.

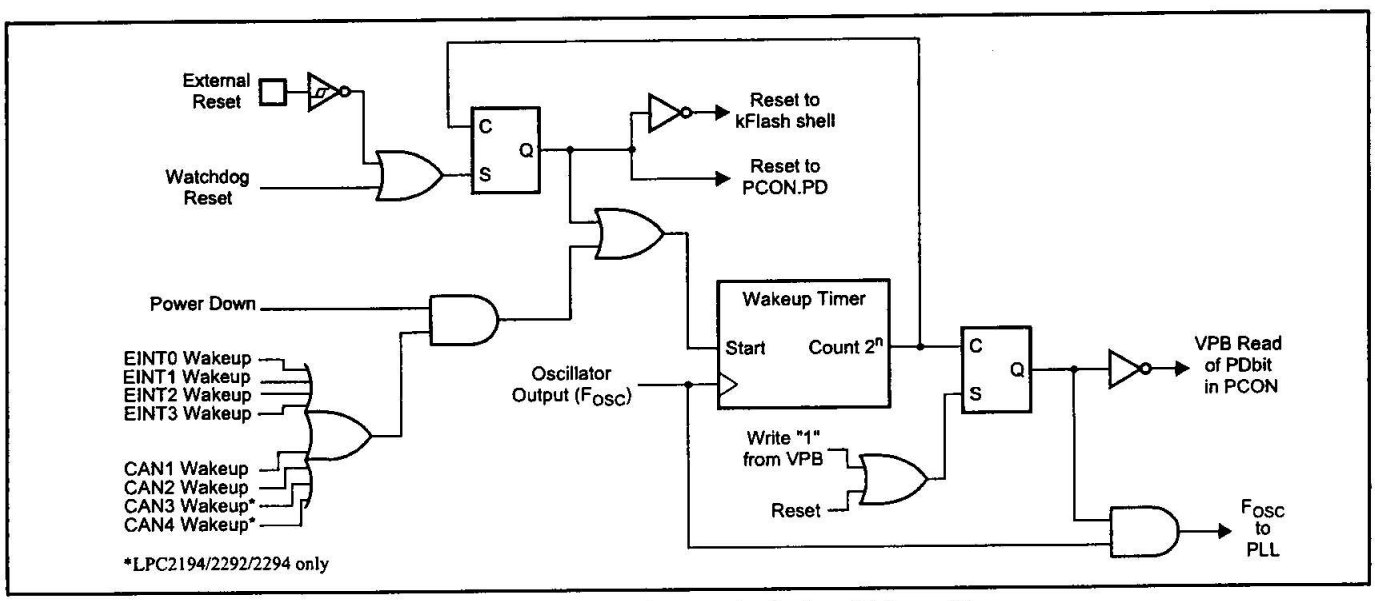


Figure 16: Reset Block Diagram including Wakeup Timer

VPB DIVIDER

The VPB Divider determines the relationship between the processor clock (cclk) and the clock used by peripheral devices (pclk). The VPB Divider serves two purposes. The first is to provides peripherals with desired pclk via VPB bus so that they can operate at the speed chosen for the ARM processor. In order to achieve this, the VPB bus may be slowed down to one half or one fourth of the processor clock rate. Because the VPB bus must work properly at power up (and its timing cannot be altered if it does not work since the VPB divider control registers reside on the VPB bus), the default condition at reset is for the VPB bus to run at one quarter speed. The second purpose of the VPB Divider is to allow power savings when an application does not require any peripherals to run at the full processor rate.

The connection of the VPB Divider relative to the oscillator and the processor clock is shown in Figure 17. Because the VPB Divider is connected to the PLL output, the PLL remains active (if it was running) during Idle mode.

VPBDIV Register (VPBDIV - 0xE01FC100)

The VPB Divider register contains two bits, allowing three divider values, as shown in Table 34.

Table 33: VPBDIV Register Map

Address	Name	Description	Access
0xE01FC100	VPBDIV	Controls the rate of the VPB clock in relation to the processor clock.	R/W

Table 34: VPB Divider Register (VPBDIV - 0xE01FC100)

VPBDIV	Function	Description	Reset Value
1:0	VPBDIV	The rate of the VPB clock is as follows: 0 0: VPB bus clock is one fourth of the processor clock. 0 1: VPB bus clock is the same as the processor clock. 1 0: VPB bus clock is one half of the processor clock. 1 1: Reserved. If this value is written to the VPBDIV register, it has no effect (the previous setting is retained).	0
3:2	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0
5:4	XCLKDIV	In the LPC2292/2294 (parts in 144 packages) only, these bits control the clock that can be driven onto the A23/XCLK pin. They have the same encoding as the VPBDIV bits above. A bit in the PINSEL2 register (Pin Connect Block on page 126) controls whether the pin carries A23 or the clock selected by this field. **Note: If this field and VPBDIV have the same value, the same clock is used on the VPB and XCLK.** (This might be useful for external logic dealing with the VPB peripherals).	0
7:6	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

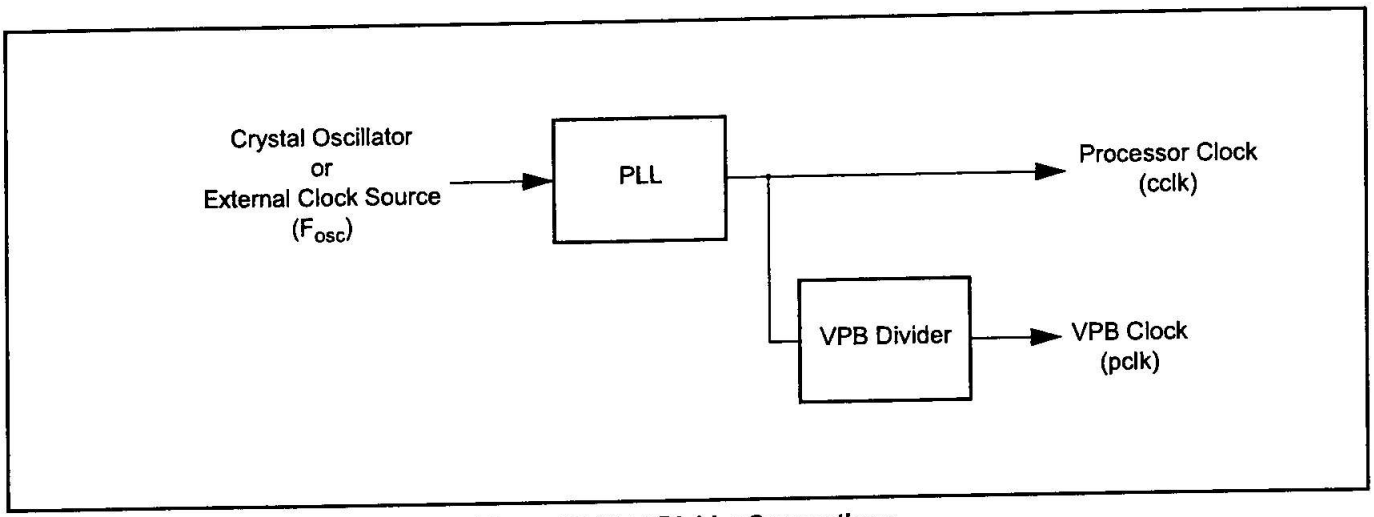


Figure 17: VPB Divider Connections

WAKEUP TIMER

The purpose of the wakeup timer is to ensure that the oscillator and other analog functions required for chip operation are fully functional before the processor is allowed to execute instructions. This is important at power on, all types of Reset, and whenever any of the aforementioned functions are turned off for any reason. Since the oscillator and other functions are turned off during Power Down mode, any wakeup of the processor from Power Down mode makes use of the Wakeup Timer.

The Wakeup Timer monitors the crystal oscillator as the means of checking whether it is safe to begin code execution. When power is applied to the chip, or some event caused the chip to exit Power down mode, some time is required for the oscillator to produce a signal of sufficient amplitude to drive the clock logic. The amount of time depends on many factors, including the rate of Vdd ramp (in the case of power on), the type of crystal and its electrical characteristics (if a quartz crystal is used), as well as any other external circuitry (e.g. capacitors), and the characteristics of the oscillator itself under the existing ambient conditions.

Once a clock is detected, the Wakeup Timer counts 4096 clocks, then enables the Flash memory to initialize. When the Flash memory initialization is complete, the processor is released to execute instructions if the external Reset has been de-asserted. In the case where an external clock source is used in the system (as opposed to a crystal connected to the oscillator pins), the possibility that there could be little or no delay for oscillator start-up must be considered. The Wakeup Timer design then ensures that any other required chip functions will be operational prior to the beginning of program execution.

The LPC2119/2129/2194/2292/2294 does not contain any analog function such as comparators that operate without clocks or any independent clock source such as a dedicated Watchdog oscillator. The only remaining functions that can operate in the absence of a clock source are the external interrupts (EINT0, EINT1, EINT2 and EINT3) and the CAN controllers. When an external interrupt is enabled for wakrup, and its selected event occurs, an oscillator wakeup cycle is started. Similarly,if a CAN block is enabled for wakeup and activity occurs on its CAN bus, an oscillator wakeup cycle is statrted. The actual interrupt (if any) occurs after the wakeup time expires, and is handled by the Vectored Interrupt Controller (VIC).

However, the pin multiplexing on the LPC2119/2129/2194/2292/229 (see Pin Configuration on page 110 and Pin Connect Block on page 126) was designed to allow other peripherals to, in effect, bring the device out of power down mode. The following pin-function pairings allow interrupts from events relating to UART0 or 1, SPI 0 or 1, or the I^2C: RxD0 / EINT0, SDA / EINT1, SSEL0 / EINT2, RxD1 / EINT3, DCD1 / EINT1, RI1 / EINT2, SSEL1 / EINT3.

To put the device in power down mode and allow activity on one or more of these buses or lines to power it back up, software should reprogram the pin function to External Interrupt, select the appropriate mode and polarity for the Interrupt, and then select power down mode. Upon wakeup software should restore the pin mulitplexing to the peripheral function.

All of the bus- or line-activity indications in the list above happen to be low-active. If software wants the device to come out of power -down mode in response to actity on more than one pin that share the same EINTi channel, it should program low-level sensitivity for that channel, because only in level mode will the channel logically OR the signals to wake the device.

The only flaw in this scheme is that the time to restart the oscillator prevents the LPC2119/2129/2194/2292/229 from capturing the bus or line activity that wakes it up. Idle mode is more appropriate than power-down mode for devices that must capture and respond to external activity in a timely manner.

To summarize: on the LPC2119/2129/2194/2292/2294, the Wakeup Timer enforces a minimum reset duration based on the crystal oscillator, and is activated whenever there is a wakeup from Power Down mode or any type of Reset.

5. MEMORY ACCELERATOR MODULE (MAM)

INTRODUCTION

Simply put, the Memory Accelerator Module (MAM) attempts to have the next ARM instruction that will be needed in its latches in time to prevent CPU fetch stalls. The method used is to split the Flash memory into two banks, each capable of independent accesses. Each of the two Flash banks has its own Prefetch Buffer and Branch Trail Buffer. The Branch Trail Buffers for the two banks capture two 128-bit lines of Flash data when an Instruction Fetch is not satisfied by either the Prefetch buffer nor Branch Trail buffer for its bank, and for which a prefetch has not been initiated. Each prefetch buffer captures one 128-bit line of instructions from its Flash bank, at the conclusion of a prefetch cycle initiated speculatively by the MAM.

Each 128 bit value includes four 32-bit ARM instructions or eight 16-bit Thumb instructions. During sequential code execution, typically one Flash bank contains or is fetching the current instruction and the entire Flash line that contains it. The other bank contains or is prefetching the next sequential code line. After a code line delivers its last instruction, the bank that contained it begins to fetch the next line in that bank.

Timing of Flash read operations is programmable and is described later in this section as well as in the System Control Block section.

Branches and other program flow changes cause a break in the sequential flow of instruction fetches described above. When a backward branch occurs, there is a distinct possibility that a loop is being executed. In this case the Branch Trail Buffers may already contain the target instruction. If so, execution continues without the need for a Flash read cycle. For a forward branch, there is also a chance that the new address is already contained in one of the Prefetch Buffers. If it is, the branch is again taken with no delay.

When a branch outside the contents of the Branch Trail and Prefetch buffers is taken, one Flash Access cycle is needed to load the Branch Trail buffers. Subsequently, there will typically be no further fetch delays until another such "Instruction Miss" occurs.

The Flash memory controller detects data accesses to the Flash memory and uses a separate buffer to store the results in a manner similar to that used during code fetches. This allows faster access to data if it is accessed sequentially. A single line buffer is provided for data accesses, as opposed to the two buffers per Flash bank that are provided for code accesses. There is no prefetch function for data accesses.

Memory Accelerator Module Blocks

The Memory Accelerator Module is divided into several functional blocks:

- A Flash Address Latch for each bank. An Incrementer function is associated with the Bank 0 Flash Address latch.
- Two Flash Memory Banks.
- Instruction Latches, Data Latches, Address Comparison latches.
- Wait logic

Figure 18 shows a simplified block diagram of the Memory Accelerator Module data paths.

In the following descriptions, the term "fetch" applies to an explicit Flash read request from the ARM. "prefetch" is used to denote a Flash read of instructions beyond the current processor fetch address.

Flash Memory Banks

There are two banks of Flash memory in order to allow two parallel accesses and eliminate delays for sequential accesses.

Flash programming operations are not controlled by the Memory Accelerator Module, but are handled as a separate function. A "boot block" sector contains Flash programming algorithms that may be called as part of the application program, and a loader that may be run to allow serial programming of the Flash memory.

The Flash memories are wired so that each sector exists in both banks, such that a sector erase operation acts on part of both banks simultaneously. In effect, the existence of two banks is transparent to the programming functions.

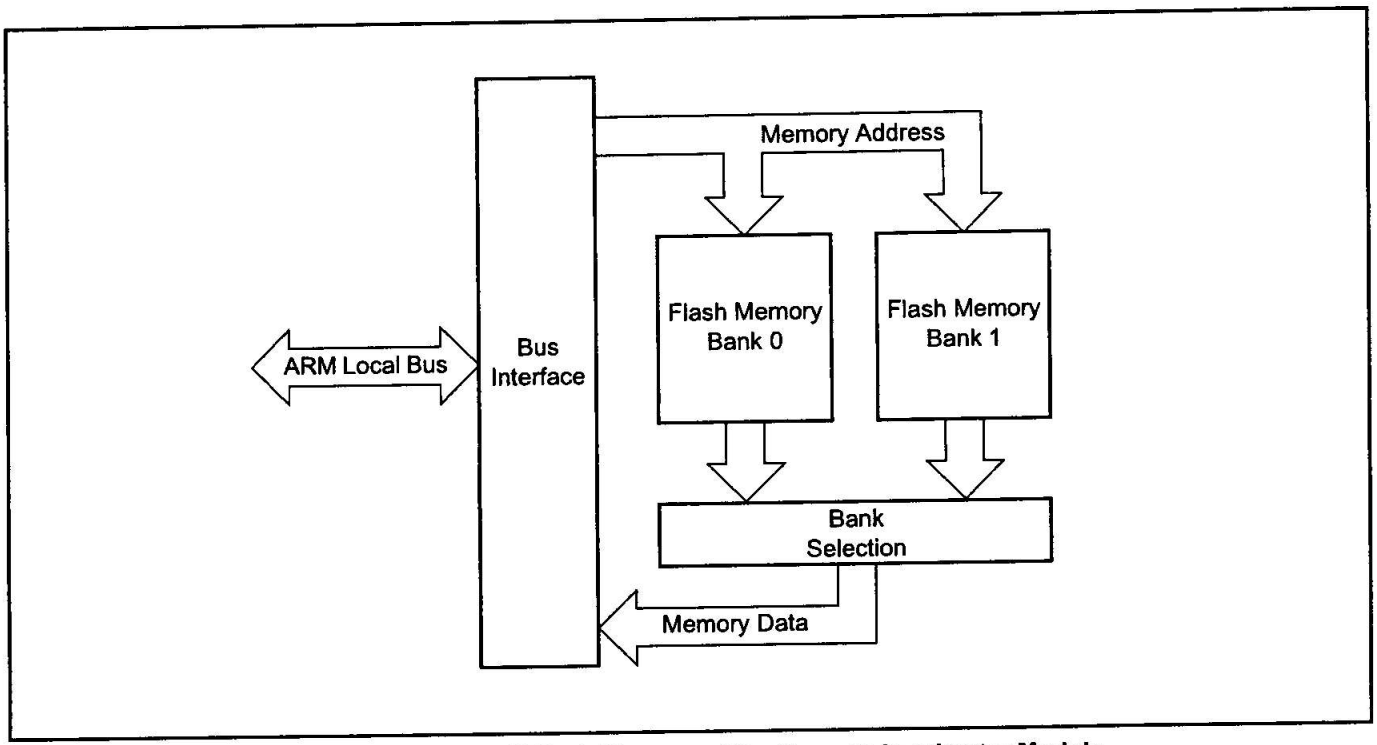


Figure 18: Simplified Block Diagram of the Memory Accelerator Module

Instruction Latches and Data Latches

Code and Data accesses are treated separately by the Memory Accelerator Module. There are two sets of 128-bit Instruction Latches and 12-bit Comparison Address Latches associated with each Flash Bank. One of the two sets, called the Branch Trail Buffer, holds the data and comparison address for that bank from the last Instruction miss. The other set, called the Prefetch Buffer, holds the data and comparison address from prefetches undertaken speculatively by the MAM. Each Instruction Latch holds 4 words of code (4 ARM instructions, or 8 Thumb instructions).

Similarly there is a 128-bit Data Latch and 13-bit Data Address latch, that are used during Data cycles. This single set of latches is shared by both Flash banks. Each Data access that is not in the Data latch causes a Flash fetch of 4 words of data, which are captured in the Data latch. This speeds up sequential Data operations, but has little or no effect on random accesses.

Flash Programming Issues

Since the Flash memory does not allow accesses during programming and erase operations, it is necessary for the MAM to force the CPU to wait if a memory access to a Flash address is requested while the Flash module is busy. (This is accomplished by asserting the ARM7TDMI-S local bus signal CLKEN.) Under some conditions, this delay could result in a Watchdog time-out. The user will need to be aware of this possibility and take steps to insure that an unwanted Watchdog reset does not cause a system failure while programming or erasing the Flash memory.

In order to preclude the possibility of stale data being read from the Flash memory, the MAM holding latches are automatically invalidated at the beginning of any Flash programming or erase operation. Any subsequent read from a Flash address will cause a new fetch to be initiated after the Flash operation has completed.

MEMORY ACCELERATOR MODULE OPERATING MODES

Three modes of operation are defined for the MAM, trading off performance for ease of predictability:

0) MAM off. All memory requests result in a Flash read operation (see note 2 below). There are no instruction prefetches.

1) MAM partially enabled. Sequential instruction accesses are fulfilled from the holding latches if the data is present. Instruction prefetch is enabled. Non-sequential instruction accesses initiate Flash read operations (see note 2 below). This means that all branches cause memory fetches. All data operations cause a Flash read because buffered data access timing is hard to predict and is very situation dependent.

2) MAM fully enabled. Any memory request (code or data) for a value that is contained in one of the corresponding holding latches is fulfilled from the latch. Instruction prefetch is enabled. Flash read operations are initiated for instruction prefetch and code or data values not available in the corresponding holding latches.

Table 35: MAM Responses to Program Accesses of Various Types

Program Memory Request Type	MAM Mode		
	0	1	2
Sequential access, data in MAM latches	Initiate Fetch [2]	Use Latched Data [1]	Use Latched Data [1]
Sequential access, data not in MAM latches	Initiate Fetch	Initiate Fetch [1]	Initiate Fetch [1]
Non-Sequential access, data in MAM latches	Initiate Fetch [2]	Initiate Fetch [1, 2]	Use Latched Data [1]
Non-Sequential access, data not in MAM latches	Initiate Fetch	Initiate Fetch [1]	Initiate Fetch [1]

Table 36: MAM Responses to Data and DMA Accesses of Various Types

Data Memory Request Type	MAM Mode		
	0	1	2
Sequential access, data in MAM latches	Initiate Fetch [2]	Initiate Fetch [2]	Use Latched Data
Sequential access, data not in MAM latches	Initiate Fetch	Initiate Fetch	Initiate Fetch
Non-Sequential access, data in MAM latches	Initiate Fetch [2]	Initiate Fetch [2]	Use Latched Data
Non-Sequential access, data not in MAM latches	Initiate Fetch	Initiate Fetch	Initiate Fetch

1. Instruction prefetch is enabled in modes 1 and 2.
2. The MAM actually uses latched data if it is available, but mimics the timing of a Flash read operation. This saves power while resulting in the same execution timing. The MAM can truly be turned off by setting the fetch timing value in MAMTIM to one clock.

MAM CONFIGURATION

After reset the MAM defaults to the disabled state. Software can turn memory access acceleration on or off at any time. This allows most of an application to be run at the highest possible performance, while certain functions can be run at a somewhat slower but more predictable rate if more precise timing is required.

REGISTER DESCRIPTION

All registers, regardless of size, are on word address boundaries. Details of the registers appear in the description of each function.

Table 37: Summary of System Control Registers

Name	Description	Access	Reset Value*	Address
MAM				
MAMCR	Memory Accelerator Module Control Register. Determines the MAM functional mode, that is, to what extent the MAM performance enhancements are enabled. See Table 38.	R/W	0	0xE01FC000
MAMTIM	Memory Accelerator Module Timing control. Determines the number of clocks used for Flash memory fetches (1 to 7 processor clocks).	R/W	0x07	0xE01FC004

*Reset Value refers to the data stored in used bits only. It does not include reserved bits content.

MAM Control Register (MAMCR - 0xE01FC000)

Two configuration bits select the three MAM operating modes, as shown in Table 38. Following Reset, MAM functions are disabled. Changing the MAM operating mode causes the MAM to invalidate all of the holding latches, resulting in new reads of Flash information as required.

Table 38: MAM Control Register (MAMCR - 0xE01FC000)

MAMCR	Function	Description	Reset Value
1:0	MAM mode control	These bits determine the operating mode of the MAM as follows: 0 0 - MAM functions disabled. 0 1 - MAM functions partially enabled. 1 0 - MAM functions fully enabled. 1 1 - reserved	0
7:2	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

MAM Timing Register (MAMTIM - 0xE01FC004)

The MAM Timing register determines how many cclk cycles are used to access the Flash memory. This allows tuning MAM timing to match the processor operating frequency. Flash access times from 1 clock to 7 clocks are possible. Single clock Flash accesses would essentially remove the MAM from timing calculations. In this case the MAM mode may be selected to optimize power usage.

Table 39: MAM Timing Register (MAMTIM - 0xE01FC004)

MAMTIM	Function	Description	Reset Value
2:0	MAM Fetch Cycle timing	These bits set the duration of MAM Flash fetch operations as follows: 0 0 0 = 0 - Reserved. 0 0 1 = 1 - MAM fetch cycles are 1 processor clock (cclk) in duration. 0 1 0 = 2 - MAM fetch cycles are 2 processor clocks (cclks) in duration. 0 1 1 = 3 - MAM fetch cycles are 3 processor clocks (cclks) in duration. 1 0 0 = 4 - MAM fetch cycles are 4 processor clocks (cclks) in duration. 1 0 1 = 5 - MAM fetch cycles are 5 processor clocks (cclks) in duration. 1 1 0 = 6 - MAM fetch cycles are 6 processor clocks (cclks) in duration. 1 1 1 = 7 - MAM fetch cycles are 7 processor clocks (cclks) in duration. **Warning:** Improper setting of this value may result in incorrect operation of the device.	0x07
7:3	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

MAM USAGE NOTES

When changing MAM timing, the MAM must first be turned off by writing a zero to MAMCR. A new value may then be written to MAMTIM. Finally, the MAM may be turned on again by writing a value (1 or 2) corresponding to the desired operating mode to MAMCR.

For system clock slower than 20 MHz, MAMTIM can be 001. For system clock between 20 MHz and 40 MHz, Flash access time is suggested to be 2 CCLKs, while in systems with system clock faster than 40 MHz, 3 CCLKs are proposed.

6. VECTORED INTERRUPT CONTROLLER (VIC)

FEATURES

- ARM PrimeCell™ Vectored Interrupt Controller
- 32 interrupt request inputs
- 16 vectored IRQ interrupts
- 16 priority levels dynamically assigned to interrupt requests
- Software interrupt generation

DESCRIPTION

The Vectored Interrupt Controller (VIC) takes 32 interrupt request inputs and programmably assigns them into 3 categories, FIQ, vectored IRQ, and non-vectored IRQ. The programmable assignment scheme means that priorities of interrupts from the various peripherals can be dynamically assigned and adjusted.

Fast Interrupt reQuest (FIQ) requests have the highest priority. If more than one request is assigned to FIQ, the VIC ORs the requests to produce the FIQ signal to the ARM processor. The fastest possible FIQ latency is achieved when only one request is classified as FIQ, because then the FIQ service routine can simply start dealing with that device. But if more than one request is assigned to the FIQ class, the FIQ service routine can read a word from the VIC that identifies which FIQ source(s) is (are) requesting an interrupt.

Vectored IRQs have the middle priority, but ony 16 of the 32 requests can be assigned to this category. Any of the 32 requests can be assigned to any of the 16 vectored IRQ slots, among which slot 0 has the highest priority and slot 15 has the lowest.

Non-vectored IRQs have the lowest priority.

The VIC ORs the requests from all the vectored and non-vectored IRQs to produce the IRQ signal to the ARM processor. The IRQ service routine can start by reading a register from the VIC and jumping there. If any of the vectored IRQs are requesting, the VIC provides the address of the highest-priority requesting IRQs service routine, otherwise it provides the address of a default routine that is shared by all the non-vectored IRQs. The default routine can read another VIC register to see what IRQs are active.

All registers in the VIC are word registers. Byte and halfword reads and write are not supported.

Additional information on the Vectored Interrupt Controller is available in the ARM PrimeCell™ Vectored Interrupt Controller (PL190) documentation.

REGISTER DESCRIPTION

The VIC implements the registers shown in Table 40. More detailed descriptions follow.

Table 40: VIC Register Map

Name	Description	Access	Reset Value*	Address
VICIRQStatus	IRQ Status Register. This register reads out the state of those interrupt requests that are enabled and classified as IRQ.	RO	0	0xFFFF F000
VICFIQStatus	FIQ Status Requests. This register reads out the state of those interrupt requests that are enabled and classified as FIQ.	RO	0	0xFFFF F004
VICRawIntr	Raw Interrupt Status Register. This register reads out the state of the 32 interrupt requests / software interrupts, regardless of enabling or classification.	RO	0	0xFFFF F008
VICIntSelect	Interrupt Select Register. This register classifies each of the 32 interrupt requests as contributing to FIQ or IRQ.	R/W	0	0xFFFF F00C
VICIntEnable	Interrupt Enable Register. This register controls which of the 32 interrupt requests and software interrupts are enabled to contribute to FIQ or IRQ.	R/W	0	0xFFFF F010
VICIntEnClr	Interrupt Enable Clear Register. This register allows software to clear one or more bits in the Interrupt Enable register.	W	0	0xFFFF F014
VICSoftInt	Software Interrupt Register. The contents of this register are ORed with the 32 interrupt requests from various peripheral functions.	R/W	0	0xFFFF F018
VICSoftIntClear	Software Interrupt Clear Register. This register allows software to clear one or more bits in the Software Interrupt register.	W	0	0xFFFF F01C
VICProtection	Protection enable register. This register allows limiting access to the VIC registers by software running in privileged mode.	R/W	0	0xFFFF F020
VICVectAddr	Vector Address Register. When an IRQ interrupt occurs, the IRQ service routine can read this register and jump to the value read.	R/W	0	0xFFFF F030
VICDefVectAddr	Default Vector Address Register. This register holds the address of the Interrupt Service routine (ISR) for non-vectored IRQs.	R/W	0	0xFFFF F034
VICVectAddr0	Vector address 0 register. Vector Address Registers 0-15 hold the addresses of the Interrupt Service routines (ISRs) for the 16 vectored IRQ slots.	R/W	0	0xFFFF F100
VICVectAddr1	Vector address 1 register	R/W	0	0xFFFF F104
VICVectAddr2	Vector address 2 register	R/W	0	0xFFFF F108
VICVectAddr3	Vector address 3 register	R/W	0	0xFFFF F10C
VICVectAddr4	Vector address 4 register	R/W	0	0xFFFF F110
VICVectAddr5	Vector address 5 register	R/W	0	0xFFFF F114
VICVectAddr6	Vector address 6 register	R/W	0	0xFFFF F118
VICVectAddr7	Vector address 7 register	R/W	0	0xFFFF F11C
VICVectAddr8	Vector address 8 register	R/W	0	0xFFFF F120
VICVectAddr9	Vector address 9 register	R/W	0	0xFFFF F124

Table 40: VIC Register Map

Name	Description	Access	Reset Value*	Address
VICVectAddr10	Vector address 10 register	R/W	0	0xFFFF F128
VICVectAddr11	Vector address 11 register	R/W	0	0xFFFF F12C
VICVectAddr12	Vector address 12 register	R/W	0	0xFFFF F130
VICVectAddr13	Vector address 13 register	R/W	0	0xFFFF F134
VICVectAddr14	Vector address 14 register	R/W	0	0xFFFF F138
VICVectAddr15	Vector address 15 register	R/W	0	0xFFFF F13C
VICVectCntl0	Vector control 0 register. Vector Control Registers 0-15 each control one of the 16 vectored IRQ slots. Slot 0 has the highest priority and slot 15 the lowest.	R/W	0	0xFFFF F200
VICVectCntl1	Vector control 1 register	R/W	0	0xFFFF F204
VICVectCntl2	Vector control 2 register	R/W	0	0xFFFF F208
VICVectCntl3	Vector control 3 register	R/W	0	0xFFFF F20C
VICVectCntl4	Vector control 4 register	R/W	0	0xFFFF F210
VICVectCntl5	Vector control 5 register	R/W	0	0xFFFF F214
VICVectCntl6	Vector control 6 register	R/W	0	0xFFFF F218
VICVectCntl7	Vector control 7 register	R/W	0	0xFFFF F21C
VICVectCntl8	Vector control 8 register	R/W	0	0xFFFF F220
VICVectCntl9	Vector control 9 register	R/W	0	0xFFFF F224
VICVectCntl10	Vector control 10 register	R/W	0	0xFFFF F228
VICVectCntl11	Vector control 11 register	R/W	0	0xFFFF F22C
VICVectCntl12	Vector control 12 register	R/W	0	0xFFFF F230
VICVectCntl13	Vector control 13 register	R/W	0	0xFFFF F234
VICVectCntl14	Vector control 14 register	R/W	0	0xFFFF F238
VICVectCntl15	Vector control 15 register	R/W	0	0xFFFF F23C

*Reset Value refers to the data stored in used bits only. It does not include reserved bits content.

VIC REGISTERS

This section describes the VIC registers in the order in which they are used in the VIC logic, from those closest to the interrupt request inputs to those most abstracted for use by software. For most people, this is also the best order to read about the registers when learning the VIC.

Software Interrupt Register (VICSoftInt - 0xFFFFF018, Read/Write)

The contents of this register are ORed with the 32 interrupt requests from the various peripherals, before any other logic is applied.

Table 41: Software Interrupt Register (VICSoftInt - 0xFFFFF018, Read/Write)

VICSoftInt	Function	Reset Value
31:0	1:force the interrupt request with this bit number. 0: do not force the interrupt request with this bit number. Writing zeroes to bits in VICSoftInt has no effect, see VICSoftIntClear.	0

Software Interrupt Clear Register (VICSoftIntClear - 0xFFFFF01C, Write Only)

This register allows software to clear one or more bits in the Software Interrupt register, without having to first read it.

Table 42: Software Interrupt Clear Register (VICSoftIntClear - 0xFFFFF01C, Write Only)

VICSoftIntClear	Function	Reset Value
31:0	1: writing a 1 clears the corresponding bit in the Software Interrupt register, thus releasing the forcing of this request. 0: writing a 0 leaves the corresponding bit in VICSoftInt unchanged.	0

Raw Interrupt Status Register (VICRawIntr - 0xFFFFF008, Read Only)

This register reads out the state of the 32 interrupt requests and software interrupts, regardless of enabling or classification.

Table 43: Raw Interrupt Status Register (VICRawIntr - 0xFFFFF008, Read-Only)

VICRawIntr	Function	Reset Value
31:0	1: the interrupt request or software interrupt with this bit number is asserted. 0: the interrupt request or software interrupt with this bit number is negated.	0

Interrupt Enable Register (VICIntEnable - 0xFFFFF010, Read/Write)

This register controls which of the 32 interrupt requests and software interrupts contribute to FIQ or IRQ.

Table 44: Interrupt Enable Register (VICINtEnable - 0xFFFFF010, Read/Write)

VICIntEnable	Function	Reset Value
31:0	When this register is read, 1s indicate interrupt requests or software interrupts that are enabled to contribute to FIQ or IRQ. When this register is written, ones enable interrupt requests or software interrupts to contribute to FIQ or IRQ, zeroes have no effect. See the VICIntEnClear register (Table 45 below), for how to disable interrupts.	0

Interrupt Enable Clear Register (VICIntEnClear - 0xFFFFF014, Write Only)

This register allows software to clear one or more bits in the Interrupt Enable register, without having to first read it.

Table 45: Software Interrupt Clear Register (VICIntEnClear - 0xFFFFF014, Write Only)

VICIntEnClear	Function	Reset Value
31:0	1: writing a 1 clears the corresponding bit in the Interrupt Enable register, thus disabling interrupts for this request. 0: writing a 0 leaves the corresponding bit in VICIntEnable unchanged.	0

Interrupt Select Register (VICIntSelect - 0xFFFFF00C, Read/Write)

This register classifies each of the 32 interrupt requests as contributing to FIQ or IRQ.

Table 46: Interrupt Select Register (VICIntSelect - 0xFFFFF00C, Read/Write)

VICIntSelect	Function	Reset Value
31:0	1: the interrupt request with this bit number is assigned to the FIQ category. 0: the interrupt request with this bit number is assigned to the IRQ category.	0

IRQ Status Register (VICIRQStatus - 0xFFFFF000, Read Only)

This register reads out the state of those interrupt requests that are enabled and classified as IRQ. It does not differentiate between vectored and non-vectored IRQs.

Table 47: IRQ Status Register (VICIRQStatus - 0xFFFFF000, Read-Only)

VICIRQStatus	Function	Reset Value
31:0	1: the interrupt request with this bit number is enabled, classified as IRQ, and asserted.	0

FIQ Status Register (VICFIQStatus - 0xFFFFF004, Read Only)

This register reads out the state of those interrupt requests that are enabled and classified as FIQ. If more than one request is classified as FIQ, the FIQ service routine can read this register to see which request(s) is (are) active.

Table 48: IRQ Status Register (VICFIQStatus - 0xFFFFF004, Read-Only)

VICFIQStatus	Function	Reset Value
31:0	1: the interrupt request with this bit number is enabled, classified as FIQ, and asserted.	0

Vector Control Registers 0-15 (VICVectCntl0-15 - 0xFFFFF200-23C, Read/Write)

Each of these registers controls one of the 16 vectored IRQ slots. Slot 0 has the highest priority and slot 15 the lowest. Note that disabling a vectored IRQ slot in one of the VICVectCntl registers does not disable the interrupt itself, the interrupt is simply changed to the non-vectored form.

Table 49: Vector Control Registers (VICVectCntl0-15 - 0xFFFFF200-23C, Read/Write)

VICVectCntl0-15	Function	Reset Value
5	1: this vectored IRQ slot is enabled, and can produce a unique ISR address when its assigned interrupt request or software interrupt is enabled, classified as IRQ, and asserted.	0
4:0	The number of the interrupt request or software interrupt assigned to this vectored IRQ slot. As a matter of good programming practice, software should not assign the same interrupt number to more than one enabled vectored IRQ slot. But if this does occur, the lower-numbered slot will be used when the interrupt request or software interrupt is enabled, classified as IRQ, and asserted.	0

Vector Address Registers 0-15 (VICVectAddr0-15 - 0xFFFFF100-13C, Read/Write)

These registers hold the addresses of the Interrupt Service routines (ISRs) for the 16 vectored IRQ slots.

Table 50: Vector Address Registers (VICVectAddr0-15 - 0xFFFFF100-13C, Read/Write)

VICVectAddr0-15	Function	Reset Value
31:0	When one or more interrupt request or software interrupt is (are) enabled, classified as IRQ, asserted, and assigned to an enabled vectored IRQ slot, the value from this register for the highest-priority such slot will be provided when the IRQ service routine reads the Vector Address register (VICVectAddr).	0

Default Vector Address Register (VICDefVectAddr - 0xFFFFF034, Read/Write)

This register holds the address of the Interrupt Service routine (ISR) for non-vectored IRQs.

Table 51: Default Vector Address Register (VICDefVectAddr - 0xFFFFF034, Read/Write)

VICDefVectAddr	Function	Reset Value
31:0	When an IRQ service routine reads the Vector Address register (VICVectAddr), and no IRQ slot responds as described above, this address is returned.	0

Vector Address Register (VICVectAddr - 0xFFFFF030, Read/Write)

When an IRQ interrupt occurs, the IRQ service routine can read this register and jump to the value read.

Table 52: Vector Address Register (VICVectAddr - 0xFFFFF030, Read/Write)

VICVectAddr	Function	Reset Value
31:0	If any of the interrupt requests or software interrupts that are assigned to a vectored IRQ slot is (are) enabled, classified as IRQ, and asserted, reading from this register returns the address in the Vector Address Register for the highest-priority such slot (lowest-numbered) such slot. Otherwise it returns the address in the Default Vector Address Register. Writing to this register does not set the value for future reads from it. Rather, this register should be written near the end of an ISR, to update the priority hardware.	0

Protection Enable Register (VICProtection - 0xFFFFF020, Read/Write)

This one-bit register controls access to the VIC registers by software running in User mode.

Table 53: Protection Enable Register (VICProtection - 0xFFFFF020, Read/Write)

VICProtection	Function	Reset Value
0	1: the VIC registers can only be accessed in privileged mode. 0: VIC registers can be accessed in User or privileged mode.	0

INTERRUPT SOURCES

Table 54 lists the interrupt sources for each peripheral function. Each peripheral device has one interrupt line connected to the Vectored Interrupt Controller, but may have several internal interrupt flags. Individual interrupt flags may also represent more than one interrupt source.

Table 54: Connection of Interrupt Sources to the Vectored Interrupt Controller

Block	Flag(s)	VIC Channel #
WDT	Watchdog Interrupt (WDINT)	0
-	Reserved for software interrupts only	1
ARM Core	Embedded ICE, DbgCommRx	2
ARM Core	Embedded ICE, DbgCommTx	3
TIMER0	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	4
TIMER1	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	5
UART0	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI)	6
UART1	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Status Interrupt (MSI)	7
PWM0	Match 0 - 6 (MR0, MR1, MR2, MR3, MR4, MR5, MR6)	8
I^2C	SI (state change)	9
SPI0	SPI Interrupt Flag (SPIF) Mode Fault (MODF)	10
SPI1	SPI Interrupt Flag (SPIF) Mode Fault (MODF)	11
PLL	PLL Lock (PLOCK)	12
RTC	Counter Increment (RTCCIF) Alarm (RTCALF)	13
System Control	External Interrupt 0 (EINT0)	14
System Control	External Interrupt 1 (EINT1)	15
System Control	External Interrupt 2 (EINT2)	16
System Control	External Interrupt 3 (EINT3)	17
A/D	A/D Converter	18

Table 54: Connection of Interrupt Sources to the Vectored Interrupt Controller

Block	Flag(s)	VIC Channel #
CAN	CAN and Acceptance Filter (1 ORed CAN, LUTerr int)	19
	CAN1 Tx	20
	CAN2 Tx	21
	CAN3 Tx (LPC2194/2292/2294 only, otherwise Reserved)	22
	CAN4 Tx (LPC2194/2292/2294 only, otherwise Reserved)	23
	Reserved	24-25
	CAN1 Rx	26
	CAN2 Rx	27
	CAN3 Rx (LPC2194/2292/2294 only, otherwise Reserved)	28
	CAN4 Rx (LPC2194/2292/2294 only, otherwise Reserved)	29
	Reserved	30-31

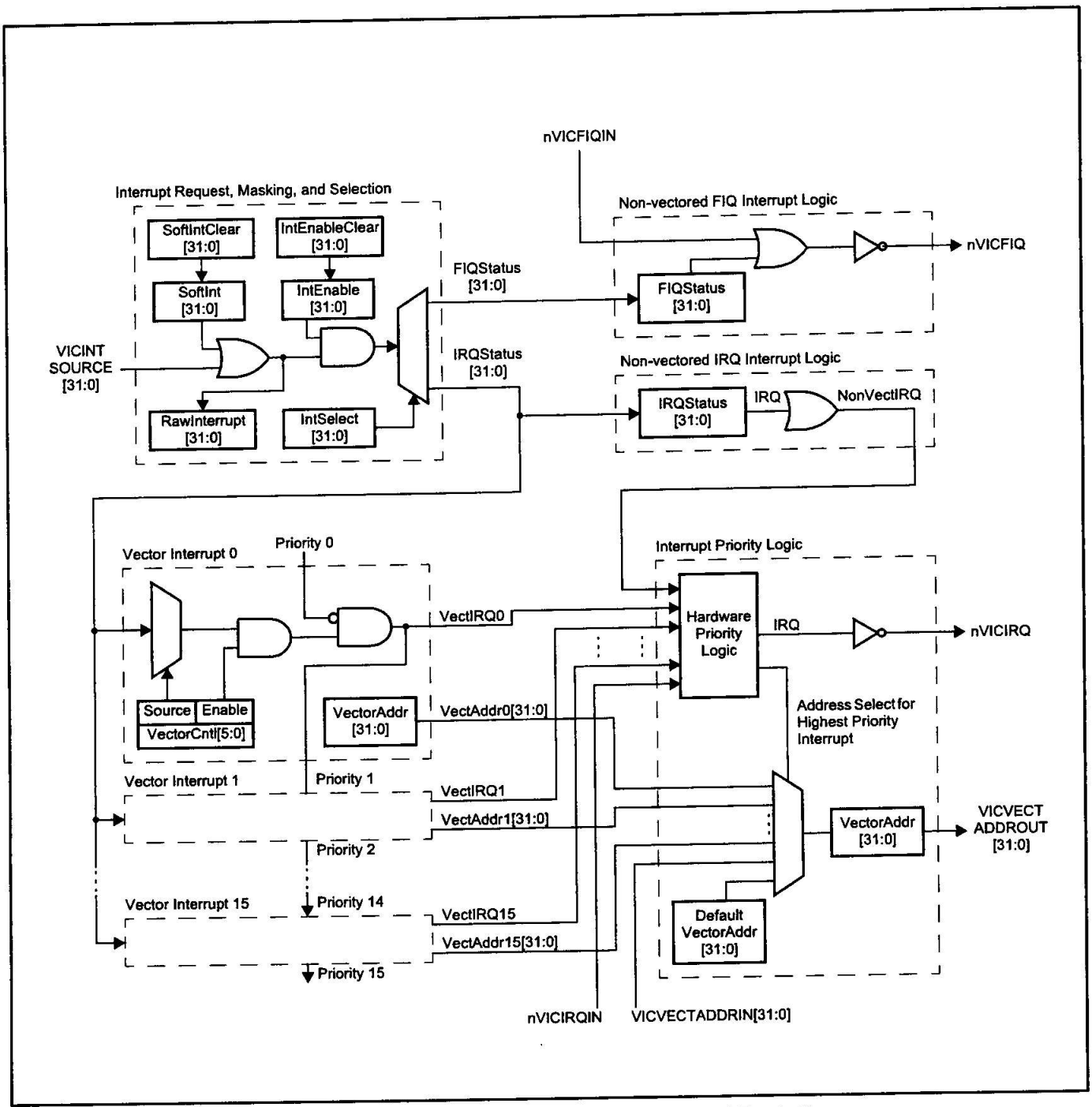


Figure 19: Block Diagram of the Vectored Interrupt Controller

SPURIOUS INTERRUPTS

Spurious interrupts are possible to occur in the ARM7TDMI based microcontroller such as the LPC2119/2129/2194/2292/2294 due to the asynchronous interrupt handling. The asynchronous character of the interrupt processing has its roots in the interaction of the core and the VIC. If the VIC state is changed between the moments when the core detects an interrupt and the core actually processes an interrupt, problems may be generated.

Real-life application may experience following scenario:

1) VIC decides there is an IRQ interrupt and sends the IRQ signal to the core.

2) Core latches the IRQ state.

3) Processing continues for a few cycles due to pipelining.

4) Core loads IRQ address from VIC.

Furthermore, It is possible that the VIC state has changed during the step 3. For example, VIC was modified so that the interrupt that triggered the sequence starting with step 1) is no longer pending -interrupt got disabled in the executed code. In this case, the VIC will not be able to clearly identify the interrupt that generated the interrupt request, and as a result the VIC will return the default interrupt VicDefVectAddr (0xFFFF F034).

This potentialy disastrous chain of events can be prevented in two ways:

1. Application code should be set up in a way to prevent the spurious interrupts to ever happen. Simple guarding of changes to the VIC may not not be enough, since for example glitches on level sensitive interrupts can also cause spurious interrupts.

2. VIC default handler should be set up and tested properly.

Details and Case Studies on Spurious Interrupts

This chapter contains details that can be obtained from the official ARM website (http://www.arm.com), FAQ section under the "Technical Support" link: http://www.arm.com/support/faqip/3677.html.

What happens if an interrupt occurs as it is being disabled?

Applies to: ARM7TDMI

If an interrupt is received by the core during execution of an instruction that disables interrupts, the ARM7 family will still take the interrupt. This occurs for both IRQ and FIQ interrupts.

For example, consider the follow instruction sequence:

```
MRS     r0, cpsr
ORR     r0, r0, #I_Bit:OR:F_Bit    ;disable IRQ and FIQ interrupts
MSR     cpsr_c, r0
```

If an IRQ interrupt is received during execution of the MSR instruction, then the behavior will be as follows:

- The IRQ interrupt is latched
- The MSR cpsr, r0 executes to completion setting both the I bit and the F bit in the CPSR
- The IRQ interrupt is taken because the core was committed to taking the interrupt exception before the I bit was set in the CPSR.
- The CPSR (with the I bit and F bit set) is moved to the SPSR_irq

This means that, on entry to the IRQ interrupt service routine, one can see the unusual effect that an IRQ interrupt has just been taken while the I bit in the SPSR is set. In the example above, the F bit will also be set in both the CPSR and SPSR. This means that FIQs are disabled upon entry to the IRQ service routine, and will remain so until explicitly re-enabled. FIQs will not be re-enabled automatically by the IRQ return sequence.

Although the example shows both IRQ and FIQ interrupts being disabled, similar behavior occurs when only one of the two interrupt types is being disabled. The fact that the core processes the IRQ after completion of the MSR instruction which disables IRQs does not normally cause a problem, since an interrupt arriving just one cycle earlier would be expected to be taken. When the interrupt routine returns with an instruction like:

```
SUBS    pc, lr, #4
```

the SPSR_IRQ is restored to the CPSR. The CPSR will now have the I bit and F bit set, and therefore execution will continue with all interrupts disabled.

However, this can cause problems in the following cases:

Problem 1: A particular routine maybe called as an IRQ handler, or as a regular subroutine. In the latter case, the system guarantees that IRQs would have been disabled prior to the routine being called. The routine exploits this restriction to determine how it was called (by examining the I bit of the SPSR), and returns using the appropriate instruction. If the routine is entered due to an IRQ being received during execution of the MSR instruction which disables IRQs, then the I bit in the SPSR will be set. The routine would therefore assume that it could not have been entered via an IRQ.

Problem 2: FIQs and IRQs are both disabled by the same write to the CPSR. In this case, if an IRQ is received during the CPSR write, FIQs will be disabled for the execution time of the IRQ handler. This may not be acceptable in a system where FIQs must not be disabled for more than a few cycles.

Workaround:

There are 3 suggested workarounds. Which of these is most applicable will depend upon the requirements of the particular system.

Solution 1: Add code similar to the following at the start of the interrupt routine.

```
SUB     lr, lr, #4          ; Adjust LR to point to return
STMFD   sp!, {..., lr}      ; Get some free regs
MRS     lr, SPSR            ; See if we got an interrupt while
TST     lr, #I_Bit          ; interrupts were disabled.
LDMNEFD sp!, {..., pc}^     ; If so, just return immediately.
                            ; The interrupt will remain  pending since we haven't
                            ; acknowledged it and will be reissued when interrupts are
                            ; next enabled.
                            ; Rest of interrupt routine
```

This code will test for the situation where the IRQ was received during a write to disable IRQs. If this is the case, the code returns immediately - resulting in the IRQ not being acknowledged (cleared), and further IRQs being disabled.

Similar code may also be applied to the FIQ handler, in order to resolve the first issue.

This is the recommended workaround, as it overcomes both problems mentioned above. However, in the case of problem two, it does add several cycles to the maximum length of time FIQs will be disabled.

Solution 2: Disable IRQs and FIQs using separate writes to the CPSR, eg:

```
MRS     r0, cpsr
ORR     r0, r0, #I_Bit      ;disable IRQs
MSR     cpsr_c, r0
ORR     r0, r0, #F_Bit      ;disable FIQs
MSR     cpsr_c, r0
```

This is the best workaround where the maximum time for which FIQs are disabled is critical (it does not increase this time at all). However, it does not solve problem one, and requires extra instructions at every point where IRQs and FIQs are disabled together.

Solution 3: Re-enable FIQs at the beginning of the IRQ handler. As the required state of all bits in the c field of the CPSR are known, this can be most efficiently be achieved by writing an immediate value to CPSR_c, for example:

```
MSR     cpsr_c, #I_Bit:OR:irq_MODE      ;IRQ should be disabled
                                        ;FIQ enabled
                                        ;ARM state, IRQ mode
```

This requires only the IRQ handler to be modified, and FIQs may be re-enabled more quickly than by using workaround 1. However, this should only be used if the system can guarantee that FIQs are never disabled while IRQs are enabled. It does not address problem one.

VIC USAGE NOTES

If user's code is runing from the on-chip RAM and an aplication uses interrupts, interrupt vectors must be re-mapped to flash address 0x0. This is necessary because all the exception vectors are located at addresses 0x0 and above. This is easily achieved by configuring MEMMAP register (located in System Control Block) to User RAM mode. Application code should be linked such that at 0x4000 0000 the Interrupt Vector Tabe (IVT) will reside.

Although multiple sources can be selected (VICIntSelect) to generate FIQ request, only one interrupt service routine should be dedicated to service all available/present FIQ request(s). Therefore, if more than one interrupt sources are classified as FIQ the FIQ interrupt service routine must read VICFIQStatus to decide based on this content what to do and how to process the interrupt request. However, it is recommended that only one interrupt source should be classified as FIQ. Classifying more than one interrupt sources as FIQ will increase the interrupt latency.

Following the completion of the desired interrupt service routine, clearing of the interrupt flag on the peripheral level will propagate to corresponding bits in VIC registers (VICRawIntr, VICFIQStatus and VICIRQStatus). Also, before the next interrupt can be serviced, it is necessary that write is performed into the VICVectAddr register before the return from interrupt is executed. This write will clear the respective interrupt flag in the internal interrupt priority hardware.

In order to disable the interrupt at the VIC you need to clear corresponding bit in the VICIntEnClr register, which in turn clears the related bit in the VICIntEnable register. This also applies to the VICSoftInt and VICSoftIntClear in which VICSoftIntClear will clear the respective bits in VICSoftInt. For example, if VICSoftInt=0x0000 0005 and bit 0 has to be cleared, VICSoftIntClear=0x0000 0001 will acomplish this. Before the new clear operation on the same bit in VICSoftInt using writing into VICSoftIntClear is performed in the future, VICSoftIntClear=0x0000 0000 must be assigned. Therefore writing 1 to any bit in Clear register will have one-time-effect in the destination register.

If the watchdog is enabled for interrupt on underflow or invalid feed sequence only then there is no way of clearing the interrupt. The only way you could perform return from interrupt is by disabling the interrupt at the VIC(using VICIntEnClr).

Example:

Assuming that UART0 and SPI0 are generating interrupt requests that are classified as vectored IRQs (UART0 being on the higher level than SPI0), while UART1 and I^2C are generating non-vectored IRQs, the following could be one possibility for VIC setup:

```
VICIntSelect = 0x0000 0000(SPI0, I2C, UART1 and UART0 are IRQ => bit10, bit9, bit7  and bit6=0)
VICIntEnable = 0x0000 06C0(SPI0, I2C, UART1 and UART0 are enabled interrupts => bit10, bit9, bit 7 and bit6=1)
VICDefVectAddr = 0x...       (holds address at what routine for servicing non-vectored IRQs (i.e. UART1 and I2C) starts)
VICVectAddr0 = 0x...         (holds address where UART0 IRQ service routine starts)
VICVectAddr1 = 0x...         (holds address where SPI0 IRQ service routine starts)
VICVectCntl0 = 0x0000 0026(interrupt source with index 6 (UART0) is enabled as the one with priority 0 (the highest))
VICVectCntl1 = 0x0000 002A(interrupt source with index 10 (SPI0) is enabled as the one with priority 1)
```

After any of IRQ requests (SPI0, I2C, UART0 or UART1) is made, microcontroller will redirect code execution to the address specified at location 0x00000018. For vectored and non-vectored IRQ's the following instruction could be placed at 0x18:

```
LDR pc,[pc,#-0xFF0]
```

This instruction loads PC with the address that is present in VICVectAddr register.

In case UART0 request has been made, VICVectAddr will be identical to VICVectAddr0, while in case SPI0 request has been made value from VICVectAddr1 will be found here. If neither UART0 nor SPI0 have generated IRQ request but UART1 and/or I^2C were the reason, content of VICVectAddr will be identical to VICDefVectAddr.

7. PIN CONFIGURATION

LPC2119/2129/2194 PINOUT

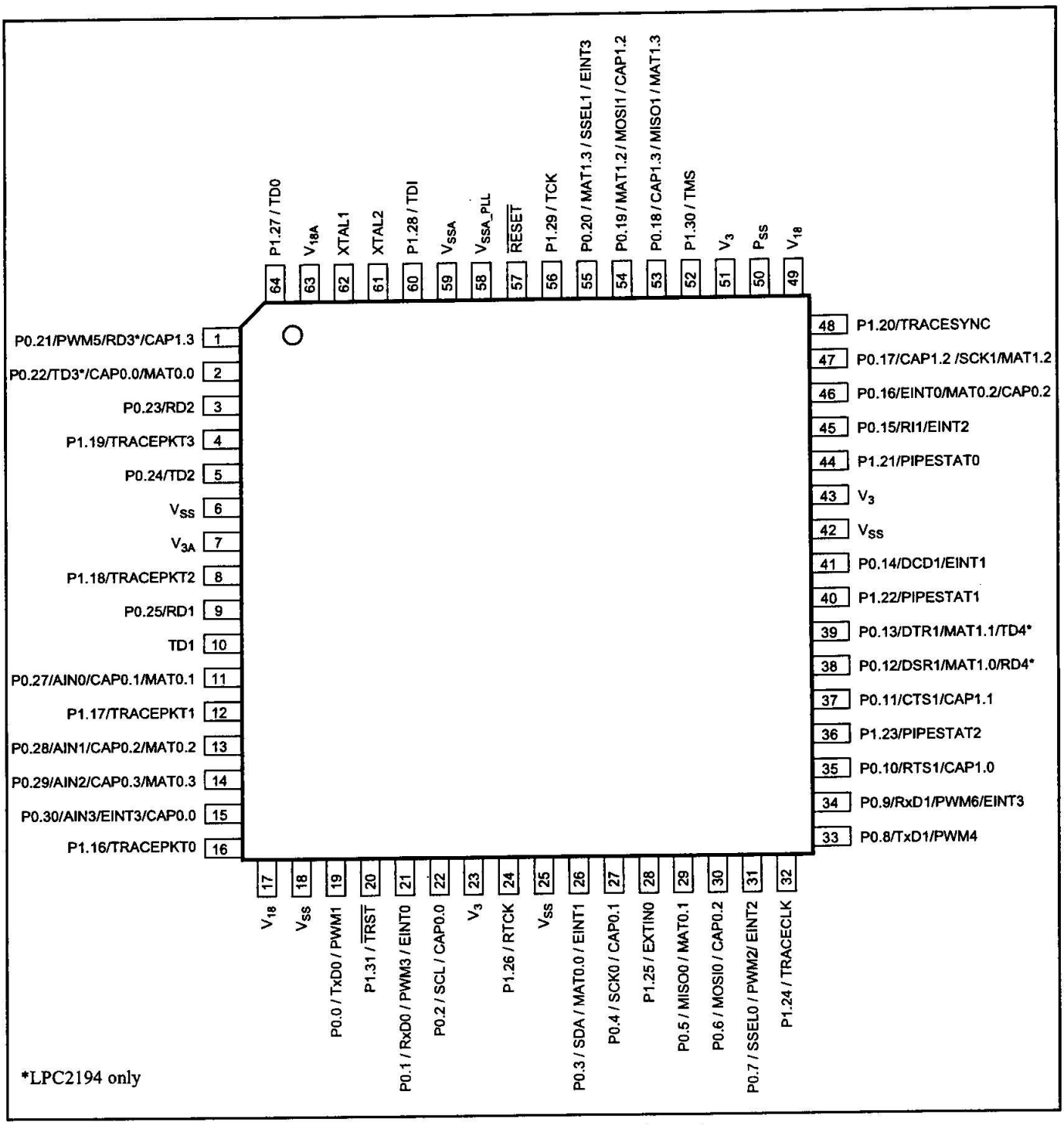


Figure 20: LPC2119/2129/2194 64-pin package

PIN DESCRIPTION FOR LPC2119/2129/2194

Pin description for LPC2119/2129/2194 and a brief of corresponding functions are shown in the following table.

Table 55: Pin description for LPC2119/2129/2194

Pin Name	LQFP64 Pin #	Type	Description		
P0.0 to P0.31		I/O	**Port 0:** Port 0 is a 32-bit bi-directional I/O port with individual direction controls for each bit. The operation of port 0 pins depends upon the pin function selected via the Pin Connect Block. Pins 26 and 31 of port 0 are not available. **Note:** All Port 0 pins excluding those that can be used as A/D inputs (P0.27, P0.28, P0.29 and P0.30) are functionally 5V tolerant. If the A/D converter is not used at all, pins associated with A/D inputs can be used as 5V tolerant digital IO pins. See "A/D Converter" chapter for A/D input pin voltage considerations.		
	19	O O	**P0.0**	**TxD0** **PWM1**	Transmitter output for UART0. Pulse Width Modulator output 1.
	21	I O I	**P0.1**	**RxD0** **PWM3** **EINT0**	Receiver input for UART0. Pulse Width Modulator output 3. External interrupt 0 input.
	22	I/O I	**P0.2**	**SCL** **CAP0.0**	I^2C clock input/output. Open drain output (for I^2C compliance). Capture input for TIMER0, channel 0.
	26	I/O O	**P0.3**	**SDA** **MAT0.0** **EINT1**	I^2C data input/output. Open drain output (for I^2C compliance). Match output for TIMER0, channel 0. External interrupt 1 input.
	27	I/O I	**P0.4**	**SCK0** **CAP0.1**	Serial Clock for SPI0. SPI clock output from master or input to slave. Capture input for TIMER0, channel 1.
	29	I/O O	**P0.5**	**MISO0** **MAT0.1**	Master In Slave Out for SPI0. Data input to SPI master or data output from SPI slave. Match output for TIMER0, channel 1.
	30	I/O I	**P0.6**	**MOSI0** **CAP0.2**	Master Out Slave In for SPI0. Data output from SPI master or data input to SPI slave. Capture input for TIMER0, channel 2.
	31	I O I	**P0.7**	**SSEL0** **PWM2** **EINT2**	Slave Select for SPI0. Selects the SPI interface as a slave. Pulse Width Modulator output 2. External interrupt 2 input.
	33	O O	**P0.8**	**TxD1** **PWM4**	Transmitter output for UART1. Pulse Width Modulator output 4.
	34	I O I	**P0.9**	**RxD1** **PWM6** **EINT3**	Receiver input for UART1. Pulse Width Modulator output 6. External interrupt 3 input.

Table 55: Pin description for LPC2119/2129/2194

Pin Name	LQFP64 Pin #	Type	Description		
	35	O	**P0.10**	**RTS1**	Request to Send output for UART1.
		I		**CAP1.0**	Capture input for TIMER1, channel 0.
	37	I	**P0.11**	**CTS1**	Clear to Send input for UART1.
		I		**CAP1.1**	Capture input for TIMER1, channel 1.
	38	I	**P0.12**	**DSR1**	Data Set Ready input for UART1.
		O		**MAT1.0**	Match output for TIMER1, channel 0.
		I		**RD4**	CAN4 receiver input (available in LPC2194 only).
	39	O	**P0.13**	**DTR1**	Data Terminal Ready output for UART1.
		O		**MAT1.1**	Match output for TIMER1, channel 1.
		O		**TD4**	CAN4 transmitter output (available in LPC2194 only).
	41	I	**P0.14**	**DCD1**	Data Carrier Detect input for UART1.
		I		**EINT1**	External interrupt 1 input. LOW on this pine while $\overline{\text{RESET}}$ is LOW forces on-chip boot-loader to take over control of the part after reset.
				Important:	LOW on pin P0.14 while $\overline{\text{RESET}}$ is LOW forces on-chip boot-loader to take over control of the part after reset.
	45	I	**P0.15**	**RI1**	Ring Indicator input for UART1.
		I		**EINT2**	External interrupt 2 input.
	46	I	**P0.16**	**EINT0**	External interrupt 0 input.
		O		**MAT0.2**	Match output for TIMER0, channel 2.
		I		**CAP0.2**	Capture input for TIMER0, channel 2.
	47	I	**P0.17**	**CAP1.2**	Capture input for TIMER1, channel 2.
		I/O		**SCK1**	Serial Clock for SPI1. SPI clock output from master or input to slave.
		O		**MAT1.2**	Match output for TIMER1, channel 2.
	53	I	**P0.18**	**CAP1.3**	Capture input for TIMER1, channel 3.
		I/O		**MISO1**	Master In Slave Out for SPI1. Data input to SPI master or data output from SPI slave.
		O		**MAT1.3**	Match output for TIMER1, channel 3.
	54	O	**P0.19**	**MAT1.2**	Match output for TIMER1, channel 2.
		I/O		**MOSI1**	Master Out Slave In for SPI1. Data output from SPI master or data input to SPI slave.
		O		**CAP1.2**	Capture input for TIMER1, channel 2.
	55	O	**P0.20**	**MAT1.3**	Match output for TIMER1, channel 3.
		I		**SSEL1**	Slave Select for SPI1. Selects the SPI interface as a slave.
		I		**EINT3**	External interrupt 3 input.
	1	O	**P0.21**	**PWM5**	Pulse Width Modulator output 5.
		I		**RD3**	CAN3 receiver input (available in LPC2194 only).
		I		**CAP1.3**	Capture input for TIMER1, channel 3.

Table 55: Pin description for LPC2119/2129/2194

Pin Name	LQFP64 Pin #	Type	Description		
		O	**P0.22**	**TD3**	CAN3 transmitter output (available in LPC2194 only)
	2	I		**CAP0.0**	Capture input for TIMER0, channel 0.
		O		**MAT0.0**	Match output for TIMER0, channel 0.
	3	I	**P0.23**	**RD2**	CAN2 receiver input.
	5	O	**P0.24**	**TD2**	CAN2 transmitter output.
	9	I	**P0.25**	**RD1**	CAN1 receiver input.
		I	**P0.27**	**AIN0**	A/D converter, input 0. This analog input is always connected to its pin.
	11	I		**CAP0.1**	Capture input for TIMER0, channel 1.
		O		**MAT0.1**	Match output for TIMER0, channel 1.
		I	**P0.28**	**AIN1**	A/D converter, input 1. This analog input is always connected to its pin.
	13	I		**CAP0.2**	Capture input for TIMER0, channel 2.
		O		**MAT0.2**	Match output for TIMER0, channel 2.
		I	**P0.29**	**AIN2**	A/D converter, input 2. This analog input is always connected to its pin.
	14	I		**CAP0.3**	Capture input for TIMER0, channel 3.
		O		**MAT0.3**	Match output for TIMER0, channel 3.
		I	**P0.30**	**AIN3**	A/D converter, input 3. This analog input is always connected to its pin.
	15	I		**EINT3**	External interrupt 3 input.
		I		**CAP0.0**	Capture input for TIMER0, channel 0.
P1.16 to P1.31		I/O	**Port 1:** Port 1 is a 32-bit bi-directional I/O port with individual direction controls for each bit. The operation of port 1 pins depends upon the pin function selected via the Pin Connect Block. Only pins 16 through 31 of port 1 are available. **Note:** All Port 1 pins are 5V tolerant with built-in pull-up resistor that sets input level to high when corresponding pin is used as input.		
	16	O	**P1.16**	**TRACEPKT0**	Trace Packet, bit 0. Standard I/O port with internal pull-up.
	12	O	**P1.17**	**TRACEPKT1**	Trace Packet, bit 1. Standard I/O port with internal pull-up.
	8	O	**P1.18**	**TRACEPKT2**	Trace Packet, bit 2. Standard I/O port with internal pull-up.
	4	O	**P1.19**	**TRACEPKT3**	Trace Packet, bit 3. Standard I/O port with internal pull-up.
		O	**P1.20**	**TRACESYNC**	Trace Synchronization. Standard I/O port with internal pull-up. LOW on this pin while $\overline{\text{RESET}}$ is LOW enables pins P1.25:16 to operate as a Trace port after reset.
	48			**Important:**	LOW on pin P1.20 while $\overline{\text{RESET}}$ is LOW enables pins P1.25:16 to operate as a Trace port after reset.

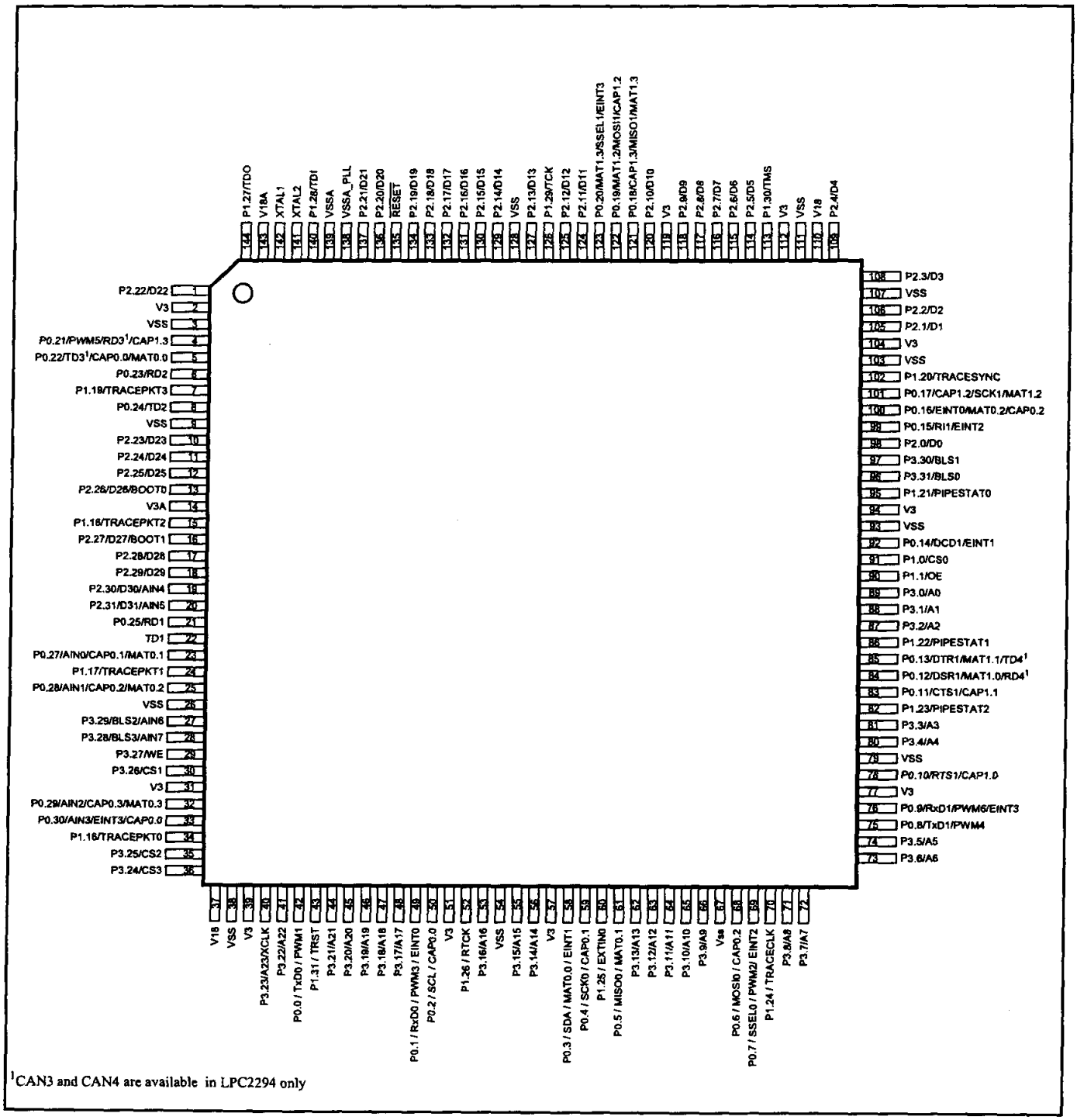
Table 55: Pin description for LPC2119/2129/2194

Pin Name	LQFP64 Pin #	Type	Description
	44	O	**P1.21 PIPESTAT0** Pipeline Status, bit 0. Standard I/O port with internal pull-up.
	40	O	**P1.22 PIPESTAT1** Pipeline Status, bit 1. Standard I/O port with internal pull-up.
	36	O	**P1.23 PIPESTAT2** Pipeline Status, bit 2. Standard I/O port with internal pull-up.
	32	O	**P1.24 TRACECLK** Trace Clock. Standard I/O port with internal pull-up.
	28	I	**P1.25 EXTIN0** External Trigger Input. Standard I/O with internal pull-up.
	24	I/O	**P1.26 RTCK** Returned Test Clock output. Extra signal added to the JTAG port. Assists debugger synchronization when processor frequency varies. Bi-directional pin with internal pullup. LOW on this pin while $\overline{\text{RESET}}$ is LOW enables pins P1.31:26 to operate as a Debug port after reset. **Important:** LOW on pin P1.26 while $\overline{\text{RESET}}$ is LOW enables pins P1.31:26 to operate as a Debug port after reset.
	64	O	**P1.27 TDO** Test Data out for JTAG interface.
	60	I	**P1.28 TDI** Test Data in for JTAG interface.
	56	I	**P1.29 TCK** Test Clock for JTAG interface.
	52	I	**P1.30 TMS** Test Mode Select for JTAG interface.
	20	I	**P1.31 $\overline{\text{TRST}}$** Test Reset for JTAG interface.
TD1	10	O	**TD1**:CAN1 transmitter output. Pin is 5 V tolerant with built-in pull-up.
$\overline{\text{RESET}}$	57	I	**External Reset Input**: A LOW on this pin resets the device, causing I/O ports and peripherals to take on their default states, and processor execution to begin at address 0. TTL with hysteresis, 5V tolerant.
XTAL1	62	I	Input to the oscillator circuit and internal clock generator circuits.
XTAL2	61	O	Output from the oscillator amplifier.
V_{SS}	6, 18, 25, 42, 50	I	**Ground:** 0V reference.
V_{SSA}	59	I	**Analog Ground:** 0V reference. This should nominally be the same voltage as V_{SS}, but should be isolated to minimize noise and error.
V_{SSA_PLL}	58	I	**PLL Analog Ground:** 0V reference. This should nominally be the same voltage as V_{SS}, but should be isolated to minimize noise and error.

Table 55: Pin description for LPC2119/2129/2194

Pin Name	LQFP64 Pin #	Type	Description
V_{18}	17, 49	I	**1.8V Core Power Supply:** This is the power supply voltage for internal circuitry.
V_{18A}	63	I	**Analog 1.8V Core Power Supply:** This is the power supply voltage for internal circuitry. This should be nominally the same voltage as V18 but should be isolated to minimize noise and error.
V_3	23, 43, 51	I	**3.3V Pad Power Supply:** This is the power supply voltage for the I/O ports.
V_{3A}	7	I	**Analog 3.3V Pad Power Supply:** This should be nominally the same voltage as V3 but should be isolated to minimize noise and error. Level on this pin is used as a reference for AD convertor.

LPC2292/2294 PINOUT



[1]CAN3 and CAN4 are available in LPC2294 only

Figure 21: LPC2292/2294 144-pin package

PIN DESCRIPTION FOR LPC2292/2294

Pin description for LPC2292/2294 and a brief of corresponding functions are shown in the following table.Pin Description

Table 56: Pin description for LPC2292/2294

Pin Name	LQFP144 Pin #	Type	Description
P0.0 to P0.31	42,49,50,58,59,61,68,69,75,76,78,83-85,92,99,100,101,121-123,4-6,8,21,23,25,32,33	I/O	**Port 0:** Port 0 is a 32-bit bi-directional I/O port with individual direction controls for each bit. The operation of port 0 pins depends upon the pin function selected via the Pin Connect Block. Pins 26 and 31 of Port 0 are not available. **Note:** All Port 0 pins excluding those that can be used as A/D inputs (P0.27, P0.28, P0.29 and P0.30) are functionally 5V tolerant. If the A/D converter is not used at all, pins associated with A/D inputs can be used as 5V tolerant digital IO pins. See "A/D Converter" chapter for A/D input pin voltage considerations.
	42	O O	**P0.0** **TxD0** Transmitter output for UART0. **PWM1** Pulse Width Modulator output 1.
	49	I O I	**P0.1** **RxD0** Receiver input for UART0. **PWM3** Pulse Width Modulator output 3. **EINT0** External interrupt 0 input.
	50	I/O I	**P0.2** **SCL** I^2C clock input/output. Open drain output (for I^2C compliance). **CAP0.0** Capture input for TIMER0, channel 0.
	58	I/O O I	**P0.3** **SDA** I^2C data input/output. Open drain output (for I^2C compliance). **MAT0.0** Match output for TIMER0, channel 0. **EINT1** External interrupt 1 input.
	59	I/O I	**P0.4** **SCK0** Serial Clock for SPI0. SPI clock output from master or input to slave. **CAP0.1** Capture input for TIMER0, channel 1.
	61	I/O O	**P0.5** **MISO0** Master In Slave Out for SPI0. Data input to SPI master or data output from SPI slave. **MAT0.1** Match output for TIMER0, channel 1.
	68	I/O I	**P0.6** **MOSI0** Master Out Slave In for SPI0. Data output from SPI master or data input to SPI slave. **CAP0.2** Capture input for TIMER0, channel 2.
	69	I O I	**P0.7** **SSEL0** Slave Select for SPI0. Selects the SPI interface as a slave. **PWM2** Pulse Width Modulator output 2. **EINT2** External interrupt 2 input.
	75	O O	**P0.8** **TxD1** Transmitter output for UART1. **PWM4** Pulse Width Modulator output 4.
	76	I O I	**P0.9** **RxD1** Receiver input for UART1. **PWM6** Pulse Width Modulator output 6. **EINT3** External interrupt 3 input.

Table 56: Pin description for LPC2292/2294

Pin Name	LQFP144 Pin #	Type	Description		
	78	O	**P0.10**	**RTS1**	Request to Send output for UART1.
		I		**CAP1.0**	Capture input for TIMER1, channel 0.
	83	I	**P0.11**	**CTS1**	Clear to Send input for UART1.
		I		**CAP1.1**	Capture input for TIMER1, channel 1.
	84	I	**P0.12**	**DSR1**	Data Set Ready input for UART1.
		O		**MAT1.0**	Match output for TIMER1, channel 0.
		I		**RD4**	CAN4 receiver input (available in LPC2294 only).
	85	O	**P0.13**	**DTR1**	Data Terminal Ready output for UART1.
		O		**MAT1.1**	Match output for TIMER1, channel 1.
		O		**TD4**	CAN4 transmitter output (available in LPC2294 only).
	92	I	**P0.14**	**DCD1**	Data Carrier Detect input for UART1.
		I		**EINT1**	External interrupt 1 input. LOW on this pin while $\overline{\text{RESET}}$ is LOW forces on-chip boot-loader to take over control of the part after reset.
				Important:	LOW on pin P0.14 while $\overline{\text{RESET}}$ is LOW forces on-chip boot-loader to take over control of the part after reset.
	99	I	**P0.15**	**RI1**	Ring Indicator input for UART1.
		I		**EINT2**	External interrupt 2 input.
	100	I	**P0.16**	**EINT0**	External interrupt 0 input.
		O		**MAT0.2**	Match output for TIMER0, channel 2.
		I		**CAP0.2**	Capture input for TIMER0, channel 2.
	101	I	**P0.17**	**CAP1.2**	Capture input for TIMER1, channel 2.
		I/O		**SCK1**	Serial Clock for SPI1. SPI clock output from master or input to slave.
		O		**MAT1.2**	Match output for TIMER1, channel 2.
	121	I	**P0.18**	**CAP1.3**	Capture input for TIMER1, channel 3.
		I/O		**MISO1**	Master In Slave Out for SPI1. Data input to SPI master or data output from SPI slave.
		O		**MAT1.3**	Match output for TIMER1, channel 3.
	122	O	**P0.19**	**MAT1.2**	Match output for TIMER1, channel 2.
		I/O		**MOSI1**	Master Out Slave In for SPI1. Data output from SPI master or data input to SPI slave.
		O		**CAP1.2**	Capture input for TIMER1, channel 2.
	123	O	**P0.20**	**MAT1.3**	Match output for TIMER1, channel 3.
		I		**SSEL1**	Slave Select for SPI1. Selects the SPI interface as a slave.
		I		**EINT3**	External interrupt 3 input.
	4	O	**P0.21**	**PWM5**	Pulse Width Modulator output 5.
		I		**RD3**	CAN3 receiver input (available in LPC2294 only).
		I		**CAP1.3**	Capture input for TIMER1, channel 3.

Table 56: Pin description for LPC2292/2294

Pin Name	LQFP144 Pin #	Type	Description		
	5	O	**P0.22**	**TD3**	CAN3 transmitter output (available in LPC2294 only).
		I		**CAP0.0**	Capture input for TIMER0, channel 0.
		O		**MAT0.0**	Match output for TIMER0, channel 0.
	6	I	**P0.23**	**RD2**	CAN2 receiver input.
	8	O	**P0.24**	**TD2**	CAN2 transmitter output.
	21	I	**P0.25**	**RD1**	CAN1 receiver input.
	23	I	**P0.27**	**AIN0**	A/D converter, input 0. This analog input is always connected to its pin.
		I		**CAP0.1**	Capture input for TIMER0, channel 1.
		O		**MAT0.1**	Match output for TIMER0, channel 1.
	25	I	**P0.28**	**AIN1**	A/D converter, input 1. This analog input is always connected to its pin.
		I		**CAP0.2**	Capture input for TIMER0, channel 2.
		O		**MAT0.2**	Match output for TIMER0, channel 2.
	32	I	**P0.29**	**AIN2**	A/D converter, input 2. This analog input is always connected to its pin.
		I		**CAP0.3**	Capture input for TIMER0, channel 3.
		O		**MAT0.3**	Match output for TIMER0, channel 3.
	33	I	**P0.30**	**AIN3**	A/D converter, input 3. This analog input is always connected to its pin.
		I		**EINT3**	External interrupt 3 input.
		I		**CAP0.0**	Capture input for TIMER0, channel 0.
P1.0 to P1.31	91,90,34,24,15,7,102,95,86,82,70,60,52,144,140,126,113,43	I/O	**Port 1:** Port 1 is a 32-bit bi-directional I/O port with individual direction controls for each bit. The operation of port 1 pins depends upon the pin function selected via the Pin Connect Block. Pins 2 through 15 of port 1 are not available. **Note:** All Port 1 pins are 5V tolerant with built-in pull-up resistor that sets input level to high when corresponding pin is used as input.		
	91	O	**P1.0**	**CS0**	Low-active Chip Select 0 signal. (Bank 0 addresses range 8000 0000 - 80FF FFFF)
	90	O	**P1.1**	**OE**	Low -active Output Enable signal.
	34	O	**P1.16**	**TRACEPKT0**	Trace Packet, bit 0. Standard I/O port with internal pull-up.
	24	O	**P1.17**	**TRACEPKT1**	Trace Packet, bit 1. Standard I/O port with internal pull-up.
	15	O	**P1.18**	**TRACEPKT2**	Trace Packet, bit 2. Standard I/O port with internal pull-up.

Table 56: Pin description for LPC2292/2294

Pin Name	LQFP144 Pin #	Type	Description
	7	O	**P1.19** **TRACEPKT3** Trace Packet, bit 3. Standard I/O port with internal pull-up.
	102	O	**P1.20** **TRACESYNC** Trace Synchronization. Standard I/O port with internal pull-up. LOW on this pin while $\overline{\text{RESET}}$ is LOW enables pins P1.25:16 to operate as a Trace port after reset. **Important:** LOW on pin P1.20 while $\overline{\text{RESET}}$ is LOW enables pins P1.25:16 to operate as a Trace port after reset.
	95	O	**P1.21** **PIPESTAT0** Pipeline Status, bit 0. Standard I/O port with internal pull-up.
	86	O	**P1.22** **PIPESTAT1** Pipeline Status, bit 1. Standard I/O port with internal pull-up.
	82	O	**P1.23** **PIPESTAT2** Pipeline Status, bit 2. Standard I/O port with internal pull-up.
	70	O	**P1.24** **TRACECLK** Trace Clock. Standard I/O port with internal pull-up.
	60	I	**P1.25** **EXTIN0** External Trigger Input. Standard I/O with internal pull-up.
	52	I/O	**P1.26** **RTCK** Returned Test Clock output. Extra signal added to the JTAG port. Assists debugger synchronization when processor frequency varies. Bi-directional pin with internal pullup. LOW on this pin while $\overline{\text{RESET}}$ is LOW enables pins P1.31:26 to operate as a Debug port after reset. **Important:** LOW on pin P1.26 while $\overline{\text{RESET}}$ is LOW enables pins P1.31:26 to operate as a Debug port after reset.
	144	O	**P1.27** **TDO** Test Data out for JTAG interface.
	140	I	**P1.28** **TDI** Test Data in for JTAG interface.
	126	I	**P1.29** **TCK** Test Clock for JTAG interface.
	113	I	**P1.30** **TMS** Test Mode Select for JTAG interface.
	43	I	**P1.31** **$\overline{\text{TRST}}$** Test Reset for JTAG interface.
P2.0 to P2.31	98,105,106,108,109,114-118,120,124,125,127,129-134,136,137,1,10-13,16-20	I/O	**Port 2:** Port 2 is a 32-bit bi-directional I/O port with individual direction controls for each bit. The operation of port 2 pins depends upon the pin function selected via the Pin Connect Block. **Note:** All Port 2 pins excluding those that can be used as A/D inputs (P2.30 and P2.31) are functionally 5V tolerant. Port 2 pin configured to perform an input function will use built-in pull-up resistor to set the default input level to high. If the A/D converter is not used at all, pins associated with A/D inputs can be used as 5V tolerant digital IO pins. See "A/D Converter" chapter for A/D input pin voltage considerations.

Table 56: Pin description for LPC2292/2294

Pin Name	LQFP144 Pin #	Type	Description		
	98	I/O	**P2.0**	**D0**	External memory data line 0.
	105	I/O	**P2.1**	**D1**	External memory data line 1.
	106	I/O	**P2.2**	**D2**	External memory data line 2.
	108	I/O	**P2.3**	**D3**	External memory data line 3.
	109	I/O	**P2.4**	**D4**	External memory data line 4.
	114	I/O	**P2.5**	**D5**	External memory data line 5.
	115	I/O	**P2.6**	**D6**	External memory data line 6.
	116	I/O	**P2.7**	**D7**	External memory data line 7.
	117	I/O	**P2.8**	**D8**	External memory data line 8.
	118	I/O	**P2.9**	**D9**	External memory data line 9.
	120	I/O	**P2.10**	**D10**	External memory data line 10.
	124	I/O	**P2.11**	**D11**	External memory data line 11.
	125	I/O	**P2.12**	**D12**	External memory data line 12.
	127	I/O	**P2.13**	**D13**	External memory data line 13.
	129	I/O	**P2.14**	**D14**	External memory data line 14.
	130	I/O	**P2.15**	**D15**	External memory data line 15.
	131	I/O	**P2.16**	**D16**	External memory data line 16.
	132	I/O	**P2.17**	**D17**	External memory data line 17.
	133	I/O	**P2.18**	**D18**	External memory data line 18.
	134	I/O	**P2.19**	**D19**	External memory data line 19.
	136	I/O	**P2.20**	**D20**	External memory data line 20.
	137	I/O	**P2.21**	**D21**	External memory data line 21.

Table 56: Pin description for LPC2292/2294

Pin Name	LQFP144 Pin #	Type	Description		
	1	I/O	**P2.22**	**D22**	External memory data line 22.
	10	I/O	**P2.23**	**D23**	External memory data line 23.
	11	I/O	**P2.24**	**D24**	External memory data line 24.
	12	I/O	**P2.25**	**D25**	External memory data line 25.
	13	I/O I	**P2.26**	**D26** **BOOT0**	External memory data line 26. While RESET is low, together with BOOT1 controls booting and internal operation. Internal pullup ensures high state if pin is left unconnected.
	16	I/O I	**P2.27**	**D27** **BOOT1**	External memory data line 27. While RESET is low, together with BOOT0 controls booting and internal operation. Internal pullup ensures high state if pin is left unconnected. BOOT1:0=00 selects 8-bit memory on CS0 for boot. BOOT1:0=01 selects 16-bit memory on CS0 for boot. BOOT1:0=10 selects 32-bit memory on CS0 for boot. BOOT1:0=11 selects Internal Flash memory.
	17	I/O	**P2.28**	**D28**	External memory data line 28.
	18	I/O	**P2.29**	**D29**	External memory data line 29.
	19	I/O I	**P2.30**	**D30** **AIN4**	External memory data line 30. A/D converter, input 4. This analog input is always connected to its pin.
	20	I/O I	**P2.31**	**D31** **AIN5**	External memory data line 31. A/D converter, input 5. This analog input is always connected to its pin.
P3.0 to P3.31	89-87,81,80,74-71,66-62,56,55,53,48-44,41,40,36,35,30-27,97,96	I/O	**Port 3:** Port 3 is a 32-bit bi-directional I/O port with individual direction controls for each bit. The operation of port 3 pins depends upon the pin function selected via the Pin Connect Block. **Note:** All Port 3 pins excluding those that can be used as A/D inputs (P3.28 and P3.29) are functionally 5V tolerant. Port 3 pin configured to perform an input function will use built-in pull-up resistor to set the default input level to high. If the A/D converter is not used at all, pins associated with A/D inputs can be used as 5V tolerant digital IO pins. See "A/D Converter" chapter for A/D input pin voltage considerations.		
	89	O	**P3.0**	**A0**	External memory address line 0.

Table 56: Pin description for LPC2292/2294

Pin Name	LQFP144 Pin #	Type	Description		
	88	O	**P3.1**	**A1**	External memory address line 1.
	87	O	**P3.2**	**A2**	External memory address line 2.
	81	O	**P3.3**	**A3**	External memory address line 3.
	80	O	**P3.4**	**A4**	External memory address line 4.
	74	O	**P3.5**	**A5**	External memory address line 5.
	73	O	**P3.6**	**A6**	External memory address line 6.
	72	O	**P3.7**	**A7**	External memory address line 7.
	71	O	**P3.8**	**A8**	External memory address line 8.
	66	O	**P3.9**	**A9**	External memory address line 9.
	65	O	**P3.10**	**A10**	External memory address line 10.
	64	O	**P3.11**	**A11**	External memory address line 11.
	63	O	**P3.12**	**A12**	External memory address line 12.
	62	O	**P3.13**	**A13**	External memory address line 13.
	56	O	**P3.14**	**A14**	External memory address line 14.
	55	O	**P3.15**	**A15**	External memory address line 15.
	53	O	**P3.16**	**A16**	External memory address line 16.
	48	O	**P3.17**	**A17**	External memory address line 17.
	47	O	**P3.18**	**A18**	External memory address line 18.
	46	O	**P3.19**	**A19**	External memory address line 19.
	45	O	**P3.20**	**A20**	External memory address line 20.
	44	O	**P3.21**	**A21**	External memory address line 21.
	41	O	**P3.22**	**A22**	External memory address line 22.

Table 56: Pin description for LPC2292/2294

Pin Name	LQFP144 Pin #	Type	Description		
	40	I/O O	**P3.23**	**A23** **XCLK**	External memory address line 23. Clock output.
	36	O	**P3.24**	**CS3**	Low-active Chip Select 3 signal. (Bank 3 addresses range 8300 0000 - 83FF FFFF)
	35	O	**P3.25**	**CS2**	Low-active Chip Select 2 signal. (Bank 2 addresses range 8200 0000 - 82FF FFFF)
	30	O	**P3.26**	**CS1**	Low-active Chip Select 1 signal. (Bank 1 addresses range 8100 0000 - 81FF FFFF)
	29	O	**P3.27**	**WE**	Low-active Write enable signal.
	28	O I	**P3.28**	**BLS3** **AIN7**	Low-active Byte Lane Select signal (Bank 3). A/D converter, input 7. This analog input is always connected to its pin.
	27	O I	**P3.29**	**BLS2** **AIN6**	Low-active Byte Lane Select signal (Bank 2). A/D converter, input 6. This analog input is always connected to its pin.
	97	O	**P3.30**	**BLS1**	Low-active Byte Lane Select signal (Bank 1).
	96	O	**P3.31**	**BLS0**	Low-active Byte Lane Select signal (Bank 0).
TD1	22	O	**TD1**:CAN1 transmitter output.Pin is 5 V tolerant with built-in pull-up.		
$\overline{\text{RESET}}$	135	I	**External Reset input**: A LOW on this pin resets the device, causing I/O ports and peripherals to take on their default states, and processor execution to begin at address 0. TTL with hysteresis, 5V tolerant.		
XTAL1	142	I	Input to the oscillator circuit and internal clock generator circuits.		
XTAL2	141	O	Output from the oscillator amplifier.		
V_{SS}	3, 9, 26, 38, 54, 67, 79, 93, 103, 107, 111, 128	I	**Ground:** 0V reference.		
V_{SSA}	139	I	**Analog Ground:** 0V reference. This should nominally be the same voltage as V_{SS}, but should be isolated to minimize noise and error.		
V_{SSA_PLL}	138	I	**PLL Analog Ground:** 0V reference. This should nominally be the same voltage as V_{SS}, but should be isolated to minimize noise and error.		

Table 56: Pin description for LPC2292/2294

Pin Name	LQFP144 Pin #	Type	Description
V18	37, 110	I	**1.8V Core Power Supply:** This is the power supply voltage for internal circuitry.
V18A	143	I	**Analog 1.8V Core Power Supply:** This is the power supply voltage for internal circuitry. This should be nominally the same voltage as V18 but should be isolated to minimize noise and error.
V3	2, 31, 39, 51, 57, 77, 94, 104, 112, 119	I	**3.3V Pad Power Supply:** This is the power supply voltage for the I/O ports.
V3A	14	I	**Analog 3.3V Pad Power Supply:** This should be nominally the same voltage as V3 but should be isolated to minimize noise and error.

8. PIN CONNECT BLOCK

FEATURES

- Allows individual pin configuration

APPLICATIONS

The purpose of the Pin Connect Block is to configure the microcontroller pins to the desired functions.

DESCRIPTION

The pin connect block allows selected pins of the microcontroller to have more than one function. Configuration registers control the multiplexers to allow connection between the pin and the on chip peripherals.

Peripherals should be connected to the appropriate pins prior to being activated, and prior to any related interrupt(s) being enabled. Activity of any enabled peripheral function that is not mapped to a related pin should be considered undefined.

Selection of a single function on a port pin completely excludes all other functions otherwise available on the same pin.

The only partial exception from the above rule of exclusion is the case of inputs to the A/D converter. Regardless of the function that is selected for the port pin that also hosts the A/D input, this A/D input can be read at any time and variations of the voltage level on this pin will be reflected in the A/D readings. However, valid analog reading(s) can be obtained if and only if the function of an analog input is selected. Only in this case proper interface circuit is active in between the physical pin and the A/D module. In all other cases, a part of digital logic necessary for the digital function to be performed will be active, and will disrupt proper behavior of the A/D.

REGISTER DESCRIPTION

The Pin Control Module contains 2 registers as shown in Table 57. below.

Table 57: Pin Connect Block Register Map

Name	Description	Access	Reset Value	Address
PINSEL0	Pin function select register 0	Read/Write	0x0000 0000	0xE002C000
PINSEL1	Pin function select register 1	Read/Write	0x1540 0000	0xE002C004
PINSEL2	Pin function select register 2	Read/Write	See Table 63 and Table 64	0xE002C014

Pin Function Select Register 0 (PINSEL0 - 0xE002C000)

The PINSEL0 register controls the functions of the pins as per the settings listed in Table 65. The direction control bit in the IO0DIR register is effective only when the GPIO function is selected for a pin. For other functions, direction is controlled automatically.

Table 58: Pin Function Select Register 0 for LPC2119/2129/2292 (PINSEL0 - 0xE002C000)

PINSEL0	Pin Name	Function when 00	Function when 01	Function when 10	Function when 11	Reset Value
1:0	P0.0	GPIO Port 0.0	TxD (UART0)	PWM1	Reserved	00
3:2	P0.1	GPIO Port 0.1	RxD (UART0)	PWM3	EINT0	00
5:4	P0.2	GPIO Port 0.2	SCL (I^2C)	Capture 0.0 (TIMER0)	Reserved	00
7:6	P0.3	GPIO Port 0.3	SDA (I^2C)	Match 0.0 (TIMER0)	EINT1	00
9:8	P0.4	GPIO Port 0.4	SCK (SPI0)	Capture 0.1 (TIMER0)	Reserved	00
11:10	P0.5	GPIO Port 0.5	MISO (SPI0)	Match 0.1 (TIMER0)	Reserved	00
13:12	P0.6	GPIO Port 0.6	MOSI (SPI0)	Capture 0.2 (TIMER0)	Reserved	00
15:14	P0.7	GPIO Port 0.7	SSEL (SPI0)	PWM2	EINT2	00
17:16	P0.8	GPIO Port 0.8	TxD UART1	PWM4	Reserved	00
19:18	P0.9	GPIO Port 0.9	RxD (UART1)	PWM6	EINT3	00
21:20	P0.10	GPIO Port 0.10	RTS (UART1)	Capture 1.0 (TIMER1)	Reserved	00
23:22	P0.11	GPIO Port 0.11	CTS (UART1)	Capture 1.1 (TIMER1)	Reserved	00
25:24	P0.12	GPIO Port 0.12	DSR (UART1)	Match 1.0 (TIMER1)	Reserved	00
27:26	P0.13	GPIO Port 0.13	DTR (UART1)	Match 1.1 (TIMER1)	Reserved	00
29:28	P0.14	GPIO Port 0.14	CD (UART1)	EINT1	Reserved	00
31:30	P0.15	GPIO Port 0.15	RI (UART1)	EINT2	Reserved	00

Table 59: Pin Function Select Register 0 for LPC2194/2294 (PINSEL0 - 0xE002C000)

PINSEL0	Pin Name	Function when 00	Function when 01	Function when 10	Function when 11	Reset Value
1:0	P0.0	GPIO Port 0.0	TxD (UART0)	PWM1	Reserved	00
3:2	P0.1	GPIO Port 0.1	RxD (UART0)	PWM3	EINT0	00
5:4	P0.2	GPIO Port 0.2	SCL (I^2C)	Capture 0.0 (TIMER0)	Reserved	00
7:6	P0.3	GPIO Port 0.3	SDA (I^2C)	Match 0.0 (TIMER0)	EINT1	00
9:8	P0.4	GPIO Port 0.4	SCK (SPI0)	Capture 0.1 (TIMER0)	Reserved	00
11:10	P0.5	GPIO Port 0.5	MISO (SPI0)	Match 0.1 (TIMER0)	Reserved	00
13:12	P0.6	GPIO Port 0.6	MOSI (SPI0)	Capture 0.2 (TIMER0)	Reserved	00
15:14	P0.7	GPIO Port 0.7	SSEL (SPI0)	PWM2	EINT2	00
17:16	P0.8	GPIO Port 0.8	TxD UART1	PWM4	Reserved	00
19:18	P0.9	GPIO Port 0.9	RxD (UART1)	PWM6	EINT3	00

Table 59: Pin Function Select Register 0 for LPC2194/2294 (PINSEL0 - 0xE002C000)

PINSEL0	Pin Name	Function when 00	Function when 01	Function when 10	Function when 11	Reset Value
21:20	P0.10	GPIO Port 0.10	RTS (UART1)	Capture 1.0 (TIMER1)	Reserved	00
23:22	P0.11	GPIO Port 0.11	CTS (UART1)	Capture 1.1 (TIMER1)	Reserved	00
25:24	P0.12	GPIO Port 0.12	DSR (UART1)	Match 1.0 (TIMER1)	RD4[1](CAN Controller 4)	00
27:26	P0.13	GPIO Port 0.13	DTR (UART1)	Match 1.1 (TIMER1)	TD4[1](CAN Controller 4)	00
29:28	P0.14	GPIO Port 0.14	CD (UART1)	EINT1	Reserved	00
31:30	P0.15	GPIO Port 0.15	RI (UART1)	EINT2	Reserved	00

[1]CAN Controller 4 is available in LPC2294 only. Fields in the table related to CAN4 have Reserved value for all other parts.

Pin Function Select Register 1 (PINSEL1 - 0xE002C004)

The PINSEL1 register controls the functions of the pins as per the settings listed in following tables. The direction control bit in the IO0DIR register is effective only when the GPIO function is selected for a pin. For other functions direction is controlled automatically.

Table 61: Pin Function Select Register 1 for LPC2119/2129/2292 (PINSEL1 - 0xE002C004)

PINSEL1	Pin Name	Function when 00	Function when 01	Function when 10	Function when 11	Reset Value
1:0	P0.16	GPIO Port 0.16	EINT0	Match 0.2 (TIMER0)	Capture 0.2 (TIMER0)	00
3:2	P0.17	GPIO Port 0.17	Capture 1.2 (TIMER1)	SCK (SPI1)	Match 1.2 (TIMER1)	00
5:4	P0.18	GPIO Port 0.18	Capture 1.3 (TIMER1)	MISO (SPI1)	Match 1.3 (TIMER1)	00
7:6	P0.19	GPIO Port 0.19	Match 1.2 (TIMER1)	MOSI (SPI1)	Match 1.3 (TIMER1)	00
9:8	P0.20	GPIO Port 0.20	Match 1.3 (TIMER1)	SSEL (SPI1)	EINT3	00
11:10	P0.21	GPIO Port 0.21	PWM5	Reserved	Capture 1.3 (TIMER1)	00
13:12	P0.22	GPIO Port 0.22	Reserved	Capture 0.0 (TIMER0)	Match 0.0 (TIMER0)	00
15:14	P0.23	GPIO Port 0.23	RD2 (CAN Controller 2)	Reserved	Reserved	00
17:16	P0.24	GPIO Port 0.24	TD2 (CAN Controller 2)	Reserved	Reserved	00
19:18	P0.25	GPIO Port 0.25	RD1 (CAN Controller 1)	Reserved	Reserved	00
21:20	P0.26	Reserved				00
23:22	P0.27	GPIO Port 0.27	AIN0 (A/D Converter)	Capture 0.1 (TIMER0)	Match 0.1 (TIMER0)	01
25:24	P0.28	GPIO Port 0.28	AIN1 (A/D Converter)	Capture 0.2 (TIMER0)	Match 0.2 (TIMER0)	01
27:26	P0.29	GPIO Port 0.29	AIN2 (A/D Converter)	Capture 0.3 (TIMER0)	Match 0.3 (TIMER0)	01
29:28	P0.30	GPIO Port 0.30	AIN3 (A/D Converter)	EINT3	Capture 0.0 (TIMER0)	01
31:30	P0.31	Reserved				00

Table 62: Pin Function Select Register 1 for LPC2194/2294 (PINSEL1 - 0xE002C004)

PINSEL1	Pin Name	Function when 00	Function when 01	Function when 10	Function when 11	Reset Value
1:0	P0.16	GPIO Port 0.16	EINT0	Match 0.2 (TIMER0)	Capture 0.2 (TIMER0)	00
3:2	P0.17	GPIO Port 0.17	Capture 1.2 (TIMER1)	SCK (SPI1)	Match 1.2 (TIMER1)	00
5:4	P0.18	GPIO Port 0.18	Capture 1.3 (TIMER1)	MISO (SPI1)	Match 1.3 (TIMER1)	00
7:6	P0.19	GPIO Port 0.19	Match 1.2 (TIMER1)	MOSI (SPI1)	Match 1.3 (TIMER1)	00
9:8	P0.20	GPIO Port 0.20	Match 1.3 (TIMER1)	SSEL (SPI1)	EINT3	00
11:10	P0.21	GPIO Port 0.21	PWM5	RD3[1](CAN Controller 3)	Capture 1.3 (TIMER1)	00
13:12	P0.22	GPIO Port 0.22	TD3[1](CAN Controller 3)	Capture 0.0 (TIMER0)	Match 0.0 (TIMER0)	00
15:14	P0.23	GPIO Port 0.23	RD2 (CAN Controller 2)	Reserved	Reserved	00
17:16	P0.24	GPIO Port 0.24	TD2 (CAN Controller 2)	Reserved	Reserved	00
19:18	P0.25	GPIO Port 0.25	RD1 (CAN Controller 1)	Reserved	Reserved	00
21:20	P0.26	Reserved				00
23:22	P0.27	GPIO Port 0.27	AIN0 (A/D Converter)	Capture 0.1 (TIMER0)	Match 0.1 (TIMER0)	01
25:24	P0.28	GPIO Port 0.28	AIN1 (A/D Converter)	Capture 0.2 (TIMER0)	Match 0.2 (TIMER0)	01
27:26	P0.29	GPIO Port 0.29	AIN2 (A/D Converter)	Capture 0.3 (TIMER0)	Match 0.3 (TIMER0)	01
29:28	P0.30	GPIO Port 0.30	AIN3 (A/D Converter)	EINT3	Capture 0.0 (TIMER0)	01
31:30	P0.31	Reserved				00

[1]CAN Controller 3 is available in LPC2294 only. Fields in the table related to CAN3 have Reserved value for all other parts.

Pin Function Select Register 2 (PINSEL2 - 0xE002C014)

The PINSEL2 register controls the functions of the pins as per the settings listed in Table 63. The direction control bit in the IO1DIR register is effective only when the GPIO function is selected for a pin. For other functions direction is controlled automatically.

Warning: use read-modify-write operation when accessing PINSEL2 register. Accidental write of 0 to bit 2 and/or bit 3 results in loss of debug and/or trace functionality! Changing of either bit 4 or bit 5 from 1 to 0 may cause an incorrect code execution!

Table 63: Pin Function Select Register 2 for LPC2119/2129/2194 (PINSEL2 - 0xE002C014)

PINSEL2	Description	Reset Value
1:0	Reserved.	00
2	When 0, pins P1.36:26 are used as GPIO pins. When 1, P1.31:26 are used as a Debug port.	$\overline{\text{P1.26/RTCK}}$
3	When 0, pins P1.25:16 are used as GPIO pins. When 1, P1.25:16 are used as a Trace port.	$\overline{\text{P1.20/ TRACESYNC}}$

Table 63: Pin Function Select Register 2 for LPC2119/2129/2194 (PINSEL2 - 0xE002C014)

PINSEL2	Description	Reset Value
4:5	Reserved. **Note:** These bits must not be altered at any time. Changing them may result in an incorrect code execution.	11
6:31	Reserved.	NA

Table 64: Pin Function Select Register 2 for LPC2292/2294 (PINSEL2 - 0xE002C014)

PINSEL2	Description			Reset Value
1:0	Reserved.			00
2	When 0, pins P1.36:26 are used as GPIO pins. When 1, P1.31:26 are used as a Debug port.			$\overline{\text{P1.26/RTCK}}$
3	When 0, pins P1.25:16 are used as GPIO pins. When 1, P1.25:16 are used as a Trace port.			$\overline{\text{P1.20/ TRACESYNC}}$
5:4	Controls the use of the data bus and strobe pins:			BOOT1:0
	Pins P2.7:0	11 = P2.7:0	0x or 10 = D7:0	
	Pin P1.0	11 = P1.0	0x or 10 = CS0	
	Pin P1.1	11 = P1.1	0x or 10 = OE	
	Pin P3.31	11 = P3.31	0x or 10 = BLS0	
	Pins P2.15:8	00 or 11 = P2.15:8	01 or 10 = D15:8	
	Pin P3.30	00 or 11 = P3.30	01 or 10 = BLS1	
	Pins P2.27:16	0x or 11 = P2.27:16	10 = D27:16	
	Pins P2.29:28	0x or 11 = P2.29:28 or Reserved	10 = D29:28	
	Pins P2.31:30	0x or 11 = P2.31:30 or AIN5:4	10 = D31:30	
	Pins P3.29:28	0x or 11 = P3.29:28 or AIN6:7	10 = BLS2:3	
6	If bits 5:4 are not 10, controls the use of pin P3.29: 0 enables P3.29, 1 enables AIN6.			1
7	If bits 5:4 are not 10, controls the use of pin P3.28: 0 enables P3.28, 1 enables AIN7.			1
8	Controls the use of pin P3.27: 0 enables P3.27, 1 enables WE.			0
10:9	Reserved.			-
11	Controls the use of pin P3.26: 0 enables P3.26, 1 enables CS1.			0
12	Reserved.			-
13	If bits 25:23 are not 111, controls the use of pin P3.23/A23/XCLK: 0 enables P3.23, 1 enables XCLK.			0
15:14	Controls the use of pin P3.25: 00 enables P3.25, 01 enables CS2, 10 and 11 are reserved values.			00
17:16	Controls the use of pin P3.24: 00 enables P3.24, 01 enables CS3, 10 and 11 are reserved values.			00
19:18	Reserved.			-
20	If bits 5:4 are not 10, controls the use of pin P2.29:28: 0 enables P2.29:28, 1 is reserved.			0
21	If bits 5:4 are not 10, controls the use of pin P2.30: 0 enables P2.30, 1 enables AIN4.			1
22	If bits 5:4 are not 10, controls the use of pin P2.31: 0 enables P2.31, 1 enables AIN5.			1
23	Controls whether P3.0/A0 is a port pin (0) or an address line (1).			1 if BOOT1:0=00 at $\overline{\text{RESET}}$=0, 0 otherwise
24	Controls whether P3.1/A1 is a port pin (0) or an address line (1).			$\overline{\text{BOOT1}}$ during Reset

Table 64: Pin Function Select Register 2 for LPC2292/2294 (PINSEL2 - 0xE002C014)

PINSEL2	Description		Reset Value
27:25	Controls the number of pins among P3.23/A23/XCLK and P3.22:2/A2.22:2 that are address lines: 000 = None 001 = A3:2 are address lines. 010 = A5:2 are address lines. 011 = A7:2 are address lines.	100 = A11:2 are address lines. 101 = A15:2 are address lines. 110 = A19:2 are address lines. 111 = A23:2 are address lines.	000 if BOOT1:0=11 at Reset, 111 otherwise
31:28	Reserved.		-

Pin Function Select Register Values

The PINSEL registers control the functions of device pins as shown below. Pairs of bits in these registers correspond to specific device pins.

Table 65: Pin Function Select Register Bits

Pinsel0 and Pinsel1 Values		Function	Value after Reset
0	0	Primary (default) function, typically GPIO Port	00
0	1	First alternate function	
1	0	Second alternate function	
1	1	Reserved	

The direction control bit in the IO0DIR/IO1DIR register is effective only when the GPIO function is selected for a pin. For other functions, direction is controlled automatically. Each derivative typically has a different pinout and therefore a different set of functions possible for each pin. Details for a specific derivative may be found in the appropriate data sheet.

BOOT CONTROL ON 144-PIN PACKAGE

In the 144-pin package only, the state of the BOOT1:0 pins, while $\overline{\text{RESET}}$ is low, controls booting and initial operation. Internal pullups in the receivers ensure high state if a pin is left unconnected. Board designers can connect weak pulldown resistors (~10 kΩ) or transistors that drive low while $\overline{\text{RESET}}$ is low, to these pins to select among the following options:

Table 66: Boot Control on BOOT1:0

BOOT1 (latched from P2.27/D27 on Reset pin rising edge only)	BOOT0 (latched from P2.26/D26 on Reset pin rising edge only)	Boot from
0	0	8-bit memory on CS0
0	1	16-bit memory on CS0
1	0	32-bit memory on CS0
1	1	Internal Flash Memory

Note that if an application enables the Watchdog Timer to Reset the part if it's not serviced, transistors driven by $\overline{\text{RESET}}$ should not be used.

9. GPIO

FEATURES

- Direction control of individual bits
- Separate control of output set and clear
- All I/O default to inputs after reset

APPLICATIONS

- General purpose I/O
- Driving LEDs, or other indicators
- Controlling off-chip devices
- Sensing digital inputs

PIN DESCRIPTION

Table 67: GPIO Pin Description

Pin Name	Type	Description
P0.0 - P0.31 P1.16 - P1.31	Input/ Output	General purpose input/output. The number of GPIOs actually available depends on the use of alternate functions.
P2.0 - P2.31 P3.0 - P3.31	Input/ Output	External bus data/address lines shared with GPIO, digital and analog functions. The number of GPIOs/digital and analog functions actually available depends on the selected bus structure. PORT2 and PORT3 are available in LPC2292/2294 only.

REGISTER DESCRIPTION

LPC2119/2129/2194 has two 32-bit General Purpose I/O ports. Total of 30 out of 32 pins are available on PORT0. PORT1 has up to 16 pins available for GPIO functions. PORT0 and PORT1 are controlled via two groups of 4 registers as shown in Table 68. LPC2292/2294 has two 32-bit additional ports, PORT2 and PORT3, and they are configured to be used either as external memory data address and data bus, or as GPIOs sharing pins with a handful of digital and analog functions. Details on PORT2 and PORT3 usage can be found in Pin Configuration and Pin Connect Block chapters.

Table 68: GPIO Register Map

Generic Name	Description	Access	Reset Value	PORT0 Address & Name	PORT1 Address & Name	PORT2 Address & Name	PORT3 Address & Name
IOPIN	GPIO Port Pin value register. The current state of the GPIO configured port pins can always be read from this register, regardless of pin direction and mode. Activity on non-GPIO configured pins will not be reflected in this register.	Read Only	NA	0xE0028000 IO0PIN	0xE0028010 IO1PIN	0xE0028020 IO2PIN	0xE0028030 IO3PIN
IOSET	GPIO Port Output set register. This register controls the state of output pins in conjunction with the IOCLR register. Writing ones produces highs at the corresponding port pins. Writing zeroes has no effect.	Read/ Write	0x0000 0000	0xE0028004 IO0SET	0xE0028014 IO1SET	0xE0028024 IO2SET	0xE0028034 IO3SET
IODIR	GPIO Port Direction control register. This register individually controls the direction of each port pin.	Read/ Write	0x0000 0000	0xE0028008 IO0DIR	0xE0028018 IO1DIR	0xE0028028 IO2DIR	0xE0028038 IO3DIR
IOCLR	GPIO Port Output clear register. This register controls the state of output pins. Writing ones produces lows at the corresponding port pins and clears the corresponding bits in the IOSET register. Writing zeroes has no effect.	Write Only	0x0000 0000	0xE002800C IO0CLR	0xE002801C IO1CLR	0xE002802C IO2CLR	0xE002803C IO3CLR

GPIO Pin Value Register (IO0PIN - 0xE0028000, IO1PIN - 0xE0028010, IO2PIN - 0xE0028020, IO3PIN - 0xE0028030)

This register provides the value of the GPIO pins. Register's value reflects any outside world influence on the GPIO configured pins only. Monitoring of non-GPIO configured port pins using IOPIN register will not be valid, since activities on non-GPIO configured pins are not indicated in the IOPIN register.

Selection of a single function on a port pin completely excludes all other functions otherwise available on the same pin.

The only partial exception from the above rule of exclusion is in the case of inputs to the A/D converter. Regardless of the function that is selected for the port pin that also hosts the A/D input, this A/D input can be read at any time and variations of the voltage level on this pin will be reflected in the A/D readings. However, valid analog reading(s) can be obtained if and only if the function of an analog input is selected. Only in this case proper interface circuit is active in between the physical pin and the A/D module. In all other cases, a part of digital logic necessary for the digital function to be performed will be active, and will disrupt proper behavior of the A/D.

Table 69: GPIO Pin Value Register (IO0PIN - 0xE0028000, IO1PIN - 0xE0028010, IO2PIN - 0xE0028020, IO3PIN - 0xE0028030)

IOPIN	Description	Value after Reset
31:0	GPIO pin value bits. Bit 0 in IO0PIN corresponds to P0.0 ... Bit 31 in IO0PIN corresponds to P0.31	Undefined

GPIO Output Set Register (IO0SET - 0xE0028004, IO1SET - 0xE0028014, IO2SET - 0xE0028024, IO3SET - 0xE0028034)

This register is used to produce a HIGH level output at the port pins if they are configured as GPIO in an OUTPUT mode. Writing 1 produces a HIGH level at the corresponding port pins. Writing 0 has no effect. If any pin is configured as an input or a secondary function, writing to IOSET has no effect.

Reading the IOSET register returns the value of this register, as determined by previous writes to IOSET and IOCLR (or IOPIN as noted above). This value does not reflect the effect of any outside world influence on the I/O pins.

Table 70: GPIO Output Set Register (IO0SET - 0xE0028004, IO1SET - 0xE0028014, IO2SET - 0xE0028024, IO3SET - 0xE0028034)

IOSET	Description	Value after Reset
31:0	Output value SET bits. Bit 0 in IO0SET corresponds to P0.0 ... Bit 31 in IO0SET corresponds to P0.31	0

GPIO Output Clear Register

(IO0CLR - 0xE002800C, IO1CLR - 0xE002801C, IO2CLR - 0xE002802C, IO3CLR - 0xE002803C)

This register is used to produce a LOW level at port pins if they are configured as GPIO in an OUTPUT mode. Writing 1 produces a LOW level at the corresponding port pins and clears the corresponding bits in the IOSET register. Writing 0 has no effect. If any pin is configured as an input or a secondary function, writing to IOCLR has no effect.

Table 71: GPIO Output Clear Register (IO0CLR - 0xE002800C, IO1CLR - 0xE002801C, IO2CLR - 0xE002802C, IO3CLR - 0xE002803C)

IOCLR	Description	Value after Reset
31:0	Output value CLEAR bits. Bit 0 in IO0CLR corresponds to P0.0 ... Bit 31 in IO0CLR corresponds to P0.31	0

GPIO Direction Register (IO0DIR - 0xE0028008, IO1DIR - 0xE0028018, IO2DIR - 0xE0028028, IO3DIR - 0xE0028038)

This register is used to control the direction of the pins when they are configured as GPIO port pins. Direction bit for any pin must be set according to the pin functionality.

Table 72: GPIO Direction Register (IO0DIR - 0xE0028008, IO1DIR - 0xE0028018, IO2DIR - 0xE0028028, IO3DIR - 0xE0028038)

IODIR	Description	Value after Reset
31:0	Direction control bits (0 = INPUT, 1 = OUTPUT). Bit 0 in IO0DIR controls P0.0 ... Bit 31 in IO0DIR controls P0.31	0

GPIO USAGE NOTES

If for the specified output pin corresponding bit is set both in GPIO Output Set Register (IOnSET) and in GPIO Output Clear Register (IOnCLR), observed pin will output level determined by the later write access of IOnSET nad IOnCLR. This means that in case of sequence:

IO0SET = 0x0000 0080

IO0CLR = 0x0000 0080

pin P0.7 will have low output, since access to Clear register came after access to Set register.

Applications that require instanatneous appearance of zeros and ones on the respected parallel port can use direct access to port's corresponding GPIO Pin Value Register (IOPIN).

Assuming that pins P0.8 to P0.15 are configured as output, write to IO0PIN:

IO0PIN = 0x0000 C700

will produce the same output as following sequence of writes:

IO0SET = 0x0000 C700

IO0CLR = 0x0000 3800

Solution utilizing access to IO0SET and IO0CLR will take more steps compared to a single IO0PIN write access.

10. UART0

FEATURES

- 16 byte Receive and Transmit FIFOs.
- Register locations conform to '550 industry standard.
- Receiver FIFO trigger points at 1, 4, 8, and 14 bytes.
- Built-in baud rate generator.

PIN DESCRIPTION

Table 73: UART0 Pin Description

Pin Name	Type	Description
RxD0	Input	**Serial Input**. Serial receive data.
TxD0	Output	**Serial Output**. Serial transmit data.

REGISTER DESCRIPTION

Table 74: UART0 Register Map

Name	Description	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	Access	Reset Value*	Address
U0RBR	Receiver Buffer Register	MSB			READ DATA				LSB	RO	un-defined	0xE000C000 DLAB = 0
U0THR	Transmit Holding Register	MSB			WRITE DATA				LSB	WO	NA	0xE000C000 DLAB = 0
U0IER	Interrupt Enable Register	0	0	0	0	0	Enable Rx Line Status Interrupt	Enable THRE Interrupt	Enable Rx Data Available Interrupt	R/W	0	0xE000C004 DLAB = 0
U0IIR	Interrupt ID Register	FIFOs Enabled		0	0	IIR3	IIR2	IIR1	IIR0	RO	0x01	0xE000C008
U0FCR	FIFO Control Register	Rx Trigger		Reserved		-	Tx FIFO Reset	Rx FIFO Reset	FIFO Enable	WO	0	0xE000C008
U0LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	Number of Stop Bits	Word Length Select		R/W	0	0xE000C00C
U0LSR	Line Status Register	Rx FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE000C014
U0SCR	Scratch Pad Register	MSB							LSB	R/W	0	0xE000C01C
U0DLL	Divisor Latch LSB	MSB							LSB	R/W	0x01	0xE000C000 DLAB = 1
U0DLM	Divisor Latch MSB	MSB							LSB	R/W	0	0xE000C004 DLAB = 1

*Reset Value refers to the data stored in used bits only. It does not include reserved bits content.

UART0 contains ten 8-bit registers as shown in Table 74. The Divisor Latch Access Bit (DLAB) is contained in U0LCR7 and enables access to the Divisor Latches.

UART0 Receiver Buffer Register (U0RBR - 0xE000C000 when DLAB = 0, Read Only)

The U0RBR is the top byte of the UART0 Rx FIFO. The top byte of the Rx FIFO contains the oldest character received and can be read via the bus interface. The LSB (bit 0) represents the "oldest" received data bit. If the character received is less than 8 bits, the unused MSBs are padded with zeroes.

The Divisor Latch Access Bit (DLAB) in U0LCR must be zero in order to access the U0RBR. The U0RBR is always Read Only.

Table 75: UART0 Receiver Buffer Register (U0RBR - 0xE000C000 when DLAB = 0, Read Only)

U0RBR	Function	Description	Reset Value
7:0	Receiver Buffer Register	The UART0 Receiver Buffer Register contains the oldest received byte in the UART0 Rx FIFO.	un-defined

UART0 Transmitter Holding Register (U0THR - 0xE000C000 when DLAB = 0, Write Only)

The U0THR is the top byte of the UART0 Tx FIFO. The top byte is the newest character in the Tx FIFO and can be written via the bus interface. The LSB represents the first bit to transmit.

The Divisor Latch Access Bit (DLAB) in U0LCR must be zero in order to access the U0THR. The U0THR is always Write Only.

Table 76: UART0 Transmit Holding Register (U0THR - 0xE000C000 when DLAB = 0, Write Only)

U0THR	Function	Description	Reset Value
7:0	Transmit Holding Register	Writing to the UART0 Transmit Holding Register causes the data to be stored in the UART0 transmit FIFO. The byte will be sent when it reaches the bottom of the FIFO and the transmitter is available.	N/A

UART0 Divisor Latch LSB Register (U0DLL - 0xE000C000 when DLAB = 1)

UART0 Divisor Latch MSB Register (U0DLM - 0xE000C004 when DLAB = 1)

The UART0 Divisor Latch is part of the UART0 Baud Rate Generator and holds the value used to divide the VPB clock (pclk) in order to produce the baud rate clock, which must be 16x the desired baud rate. The U0DLL and U0DLM registers together form a 16 bit divisor where U0DLL contains the lower 8 bits of the divisor and U0DLM contains the higher 8 bits of the divisor. A 'h0000 value is treated like a 'h0001 value as division by zero is not allowed.The Divisor Latch Access Bit (DLAB) in U0LCR must be one in order to access the UART0 Divisor Latches.

Table 77: UART0 Divisor Latch LSB Register (U0DLL - 0xE000C000 when DLAB = 1)

U0DLL	Function	Description	Reset Value
7:0	Divisor Latch LSB Register	The UART0 Divisor Latch LSB Register, along with the U0DLM register, determines the baud rate of the UART0.	0x01

Table 78: UART0 Divisor Latch MSB Register (U0DLM - 0xE000C004 when DLAB = 1)

U0DLM	Function	Description	Reset Value
7:0	Divisor Latch MSB Register	The UART0 Divisor Latch MSB Register, along with the U0DLL register, determines the baud rate of the UART0.	0

UART0 Interrupt Enable Register (U0IER - 0xE000C004 when DLAB = 0)

The U0IER is used to enable the four UART0 interrupt sources.

Table 79: UART0 Interrupt Enable Register Bit Descriptions (U0IER - 0xE000C004 when DLAB = 0)

U0IER	Function	Description	Reset Value
0	RBR Interrupt Enable	0: Disable the RDA interrupt. 1: Enable the RDA interrupt. U0IER0 enables the Receive Data Available interrupt for UART0. It also controls the Character Receive Time-out interrupt.	0
1	THRE Interrupt Enable	0: Disable the THRE interrupt. 1: Enable the THRE interrupt. U0IER1 enables the THRE interrupt for UART0. The status of this interrupt can be read from U0LSR5.	0
2	Rx Line Status Interrupt Enable	0: Disable the Rx line status interrupts. 1: Enable the Rx line status interrupts. U0IER2 enables the UART0 Rx line status interrupts. The status of this interrupt can be read from U0LSR[4:1].	0
7:3	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

UART0 Interrupt Identification Register (U0IIR - 0xE000C008, Read Only)

The U0IIR provides a status code that denotes the priority and source of a pending interrupt. The interrupts are frozen during an U0IIR access. If an interrupt occurs during an U0IIR access, the interrupt is recorded for the next U0IIR access.

Table 80: UART0 Interrupt Identification Register Bit Descriptions (U0IIR - 0xE000C008, Read Only)

U0IIR	Function	Description	Reset Value
0	Interrupt Pending	0: At least one interrupt is pending. 1: No pending interrupts. Note that U0IIR0 is active low. The pending interrupt can be determined by evaluating U0IER3:1.	1
3:1	Interrupt Identification	011: 1. Receive Line Status (RLS) 010: 2a.Receive Data Available (RDA) 110: 2b.Character Time-out Indicator (CTI) 001: 3. THRE Interrupt. U0IER3 identifies an interrupt corresponding to the UART0 Rx FIFO. All other combinations of U0IER3:1 not listed above are reserved (000,100,101,111).	0
5:4	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	FIFO Enable	These bits are equivalent to U0FCR0.	0

Interrupts are handled as described in Table 81. Given the status of U0IIR[3:0], an interrupt handler routine can determine the cause of the interrupt and how to clear the active interrupt. Interrupts are handled as described in Table 81. The U0IIR must be read in order to clear the interrupt prior to exitting the Interrupt Service Routine.

The UART0 RLS interrupt (U0IIR3:1=011) is the highest priority interrupt and is set whenever any one of four error conditions occur on the UART0 Rx input: overrun error (OE), parity error (PE), framing error (FE) and break interrupt (BI). The UART0 Rx error condition that set the interrupt can be observed via U0LSR4:1. The interrupt is cleared upon an U0LSR read.

The UART0 RDA interrupt (U0IIR3:1=010) shares the second level priority with the CTI interrupt (U0IIR3:1=110). The RDA is activated when the UART0 Rx FIFO reaches the trigger level defined in U0FCR7:6 and is reset when the UART0 Rx FIFO depth falls below the trigger level. When the RDA interrupt goes active, the CPU can read a block of data defined by the trigger level.

The CTI interrupt (U0IIR3:1=110) is a second level interrupt and is set when the UART0 Rx FIFO contains at least one character and no UART0 Rx FIFO activity has occurred in 3.5 to 4.5 character times. Any UART0 Rx FIFO activity (read or write of UART0 RSR) will clear the interrupt. This interrupt is intended to flush the UART0 RBR after a message has been received that is not a multiple of the trigger level size. For example, if a peripheral wished to send a 105 character message and the trigger level was 10 characters, the CPU would receive 10 RDA interrupts resulting in the transfer of 100 characters and 1 to 5 CTI interrupts (depending on the service routine) resulting in the transfer of the remaining 5 characters.

Table 81: UART0 Interrupt Handling

U0IIR[3:0]	Priority	Interrupt Type	Interrupt Source	Interrupt Reset
0001	-	none	none	-
0110	Highest	Rx Line Status / Error	OE or PE or FE or BI	U0LSR Read
0100	Second	Rx Data Available	Rx data available or trigger level reached in FIFO (U0FCR0=1)	U0RBR Read or UART0 FIFO drops below trigger level
1100	Second	Character Time-out Indication	Minimum of one character in the Rx FIFO and no character input or removed during a time period depending on how many characters are in FIFO and what the trigger level is set at (3.5 to 4.5 character times). The exact time will be: [(word length) X 7 - 2] X 8 + {(trigger level - number of characters) X 8 + 1] RCLKs	U0 RBR Read
0010	Third	THRE	THRE	U0IIR Read (if source of interrupt) or THR write
note: values "0000", "0011", "0101", "0111", "1000", "1001", "1010", "1011","1101","1110","1111" are reserved.				

The UART0 THRE interrupt (U0IIR3:1=001) is a third level interrupt and is activated when the UART0 THR FIFO is empty provided certain initialization conditions have been met. These initialization conditions are intended to give the UART0 THR FIFO a chance to fill up with data to eliminate many THRE interrupts from occurring at system start-up. The initialization conditions implement a one character delay minus the stop bit whenever THRE=1 and there have not been at least two characters in the U0THR at one time since the last THRE=1 event. This delay is provided to give the CPU time to write data to U0THR without a THRE interrupt to decode and service. A THRE interrupt is set immediately if the UART0 THR FIFO has held two or more characters at one time and currently, the U0THR is empty. The THRE interrupt is reset when a U0THR write occurs or a read of the U0IIR occurs and the THRE is the highest interrupt (U0IIR3:1=001).

UART0 FIFO Control Register (U0FCR - 0xE000C008)

The U0FCR controls the operation of the UART0 Rx and Tx FIFOs.

Table 82: UART0 FIFO Control Register Bit Descriptions (U0FCR - 0xE000C008)

U0FCR	Function	Description	Reset Value
0	FIFO Enable	Active high enable for both UART0 Rx and Tx FIFOs and U0FCR7:1 access. This bit must be set for proper UART0 opearation. Any transition on this bit will automatically clear the UART0 FIFOs.	0
1	Rx FIFO Reset	Writing a logic 1 to U0FCR1 will clear all bytes in UART0 Rx FIFO and reset the pointer logic. This bit is self-clearing.	0
2	Tx FIFO Reset	Writing a logic 1 to U0FCR2 will clear all bytes in UART0 Tx FIFO and reset the pointer logic. This bit is self-clearing.	0
5:3	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	Rx Trigger Level Select	00: trigger level 0 (default=1 character or 0x01h) 01: trigger level 1 (default=4 characters or 0x04h) 10: trigger level 2 (default=8 characters or 0x08h) 11: trigger level 3 (default=14 characters or 0x0eh) These two bits determine how many receiver UART0 FIFO characters must be written before an interrupt is activated. The four trigger levels are defined by the user at compilation allowing the user to tune the trigger levels to the FIFO depths chosen.	0

UART0 Line Control Register (U0LCR - 0xE000C00C)

The U0LCR determines the format of the data character that is to be transmitted or received.

Table 83: UART0 Line Control Register Bit Descriptions (U0LCR - 0xE000C00C)

U0LCR	Function	Description	Reset Value
1:0	Word Length Select	00: 5 bit character length 01: 6 bit character length 10: 7 bit character length 11: 8 bit character length	0
2	Stop Bit Select	0: 1 stop bit 1: 2 stop bits (1.5 if U0LCR[1:0]=00)	0
3	Parity Enable	0: Disable parity generation and checking 1: Enable parity generation and checking	0
5:4	Parity Select	00: Odd parity 01: Even parity 10: Forced "1" stick parity 11: Forced "0" stick parity	0
6	Break Control	0: Disable break transmission 1: Enable break transmission. Output pin UART0 TxD is forced to logic 0 when U0LCR6 is active high.	0
7	Divisor Latch Access Bit	0: Disable access to Divisor Latches 1: Enable access to Divisor Latches	0

UART0 Line Status Register (U0LSR - 0xE000C014, Read Only)

The U0LSR is a read-only register that provides status information on the UART0 Tx and Rx blocks.

Table 84: UART0 Line Status Register Bit Descriptions (U0LSR - 0xE000C014, Read Only)

U0LSR	Function	Description	Reset Value
0	Receiver Data Ready (RDR)	0: U0RBR is empty 1: U0RBR contains valid data U0LSR0 is set when the U0RBR holds an unread character and is cleared when the UART0 RBR FIFO is empty.	0
1	Overrun Error (OE)	0: Overrun error status is inactive. 1: Overrun error status is active. The overrun error condition is set as soon as it occurs. An U0LSR read clears U0LSR1. U0LSR1 is set when UART0 RSR has a new character assembled and the UART0 RBR FIFO is full. In this case, the UART0 RBR FIFO will not be overwritten and the character in the UART0 RSR will be lost.	0
2	Parity Error (PE)	0: Parity error status is inactive. 1: Parity error status is active. When the parity bit of a received character is in the wrong state, a parity error occurs. An U0LSR read clears U0LSR2. Time of parity error detection is dependent on U0FCR0. A parity error is associated with the character being read from the UART0 RBR FIFO.	0
3	Framing Error (FE)	0: Framing error status is inactive. 1: Framing error status is active. When the stop bit of a received character is a logic 0, a framing error occurs. An U0LSR read clears U0LSR3. The time of the framing error detection is dependent on U0FCR0. A framing error is associated with the character being read from the UART0 RBR FIFO. Upon detection of a framing error, the Rx will attempt to resynchronize to the data and assume that the bad stop bit is actually an early start bit. However, it cannot be assumed that the next received byte will be correct even if there is no Framing Error.	0
4	Break Interrupt (BI)	0: Break interrupt status is inactive. 1: Break interrupt status is active. When RxD0 is held in the spacing state (all 0's) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once the break condition has been detected, the receiver goes idle until RxD0 goes to marking state (all 1's). An U0LSR read clears this status bit. The time of break detection is dependent on U0FCR0. The break interrupt is associated with the character being read from the UART0 RBR FIFO.	0
5	Transmitter Holding Register Empty (THRE)	0: U0THR contains valid data. 1: U0THR is empty. THRE is set immediately upon detection of an empty UART0 THR and is cleared on a U0THR write.	1
6	Transmitter Empty (TEMT)	0: U0THR and/or the U0TSR contains valid data. 1: U0THR and the U0TSR are empty. TEMT is set when both U0THR and U0TSR are empty; TEMT is cleared when either the U0TSR or the U0THR contain valid data.	1
7	Error in Rx FIFO (RXFE)	0: U0RBR contains no UART0 Rx errors or U0FCR0=0. 1: UART0 RBR contains at least one UART0 Rx error. U0LSR7 is set when a character with a Rx error such as framing error, parity error or break interrupt, is loaded into the U0RBR. This bit is cleared when the U0LSR register is read and there are no subsequent errors in the UART0 FIFO.	0

UART0 Scratch Pad Register (U0SCR - 0xE000C01C)

The U0SCR has no effect on the UART0 operation. This register can be written and/or read at user's discretion. There is no provision in the interrupt interface that would indicate to the host that a read or write of the U0SCR has occurred.

Table 85: UART0 Scratchpad Register (U0SCR - 0xE000C01C)

U0SCR	Function	Description	Reset Value
7:0	-	A readable, writable byte.	0

ARCHITECTURE

The architecture of the UART0 is shown below in the block diagram.

The VPB interface provides a communications link between the CPU or host and the UART0.

The UART0 receiver block, U0Rx, monitors the serial input line, RxD0, for valid input. The UART0 Rx Shift Register (U0RSR) accepts valid characters via RxD0. After a valid character is assembled in the U0RSR, it is passed to the UART0 Rx Buffer Register FIFO to await access by the CPU or host via the generic host interface.

The UART0 transmitter block, U0Tx, accepts data written by the CPU or host and buffers the data in the UART0 Tx Holding Register FIFO (U0THR). The UART0 Tx Shift Register (U0TSR) reads the data stored in the U0THR and assembles the data to transmit via the serial output pin, TxD0.

The UART0 Baud Rate Generator block, U0BRG, generates the timing enables used by the UART0 Tx block. The U0BRG clock input source is the VPB clock (pclk). The main clock is divided down per the divisor specified in the U0DLL and U0DLM registers. This divided down clock is a 16x oversample clock, NBAUDOUT.

The interrupt interface contains registers U0IER and U0IIR. The interrupt interface receives several one clock wide enables from the U0Tx and U0Rx blocks.

Status information from the U0Tx and U0Rx is stored in the U0LSR. Control information for the U0Tx and U0Rx is stored in the U0LCR.

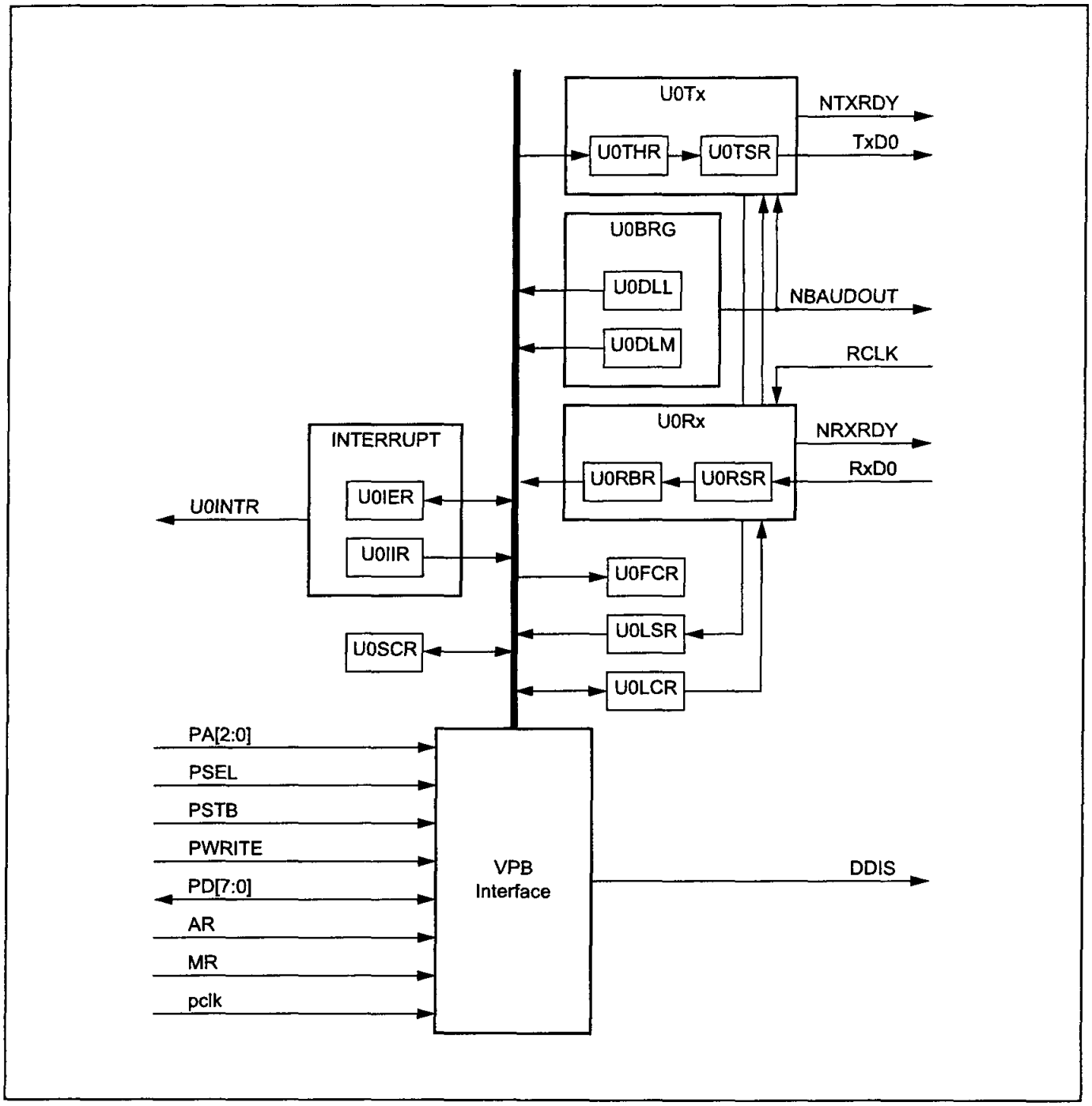


Figure 22: UART0 Block Diagram

13. SPI INTERFACE

FEATURES

- Two complete and independent SPI cintrollers
- Compliant with Serial Peripheral Interface (SPI) specification.
- Synchronous, Serial, Full Duplex Communication.
- Combined SPI master and slave.
- Maximum data bit rate of one eighth of the input clock rate.

DESCRIPTION

SPI Overview

SPI0 and SPI1 are full duplex serial interfaces. They can handle multiple masters and slaves being connected to a given bus. Only a single master and a single slave can communicate on the interface during a given data transfer. During a data transfer the master always sends a byte of data to the slave, and the slave always sends a byte of data to the master.

SPI Data Transfers

Figure 33 is a timing diagram that illustrates the four different data transfer formats that are available with the SPI. This timing diagram illustrates a single 8 bit data transfer. The first thing one should notice in this timing diagram is that it is divided into three horizontal parts. The first part describes the SCK and SSEL signals. The second part describes the MOSI and MISO signals when the CPHA variable is 0. The third part describes the MOSI and MISO signals when the CPHA variable is 1.

In the first part of the timing diagram, note two points. First, the SPI is illustrated wit CPOL set to both 0 and 1. The second point to note is the activation and de-activation of the SSEL signal. When CPHA = 1, the SSEL signal will always go inactive between data transfers. This is not guaranteed when CPHA = 0 (the signal can remain active).

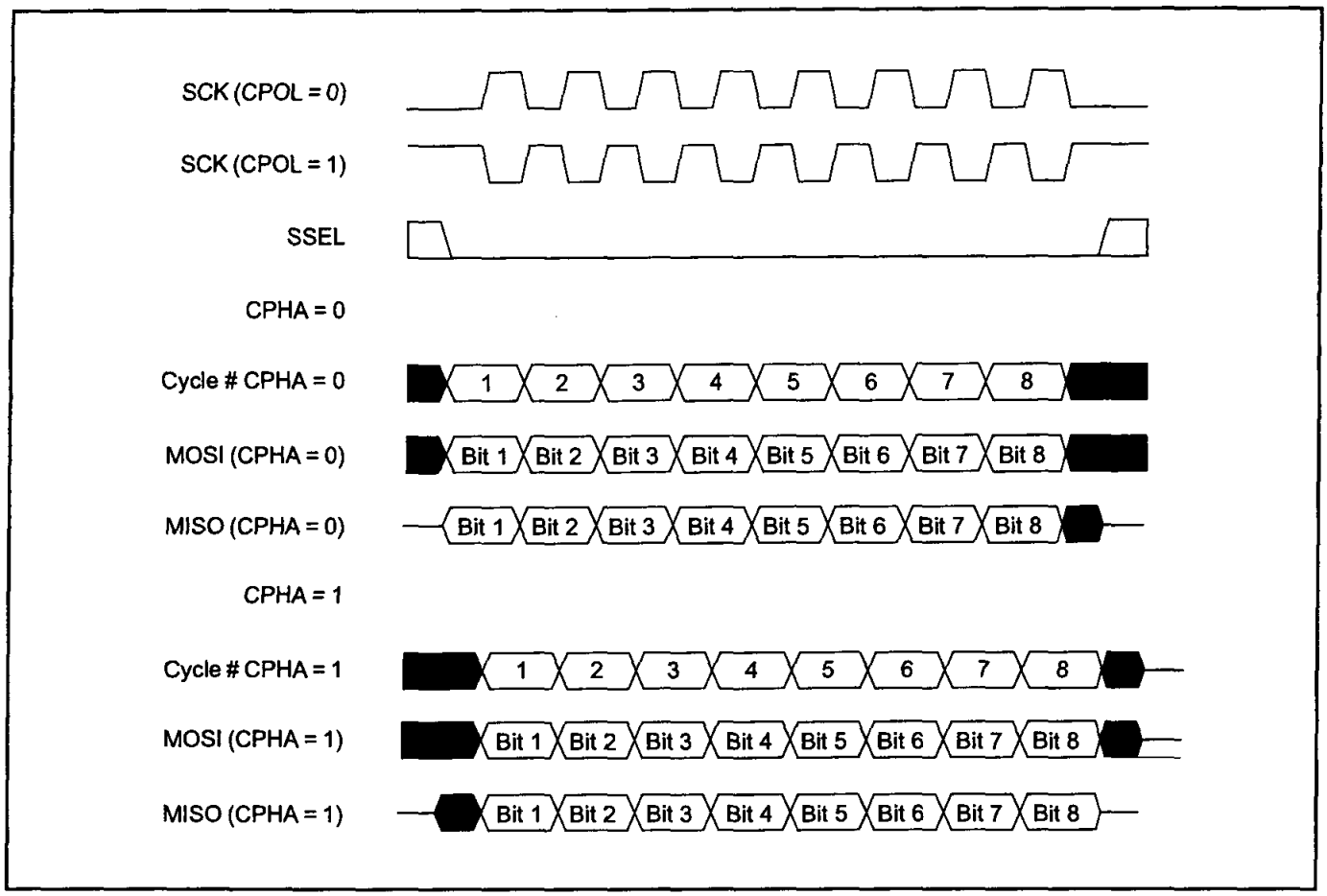


Figure 33: SPI Data Transfer Format (CPHA = 0 and CPHA = 1)

The data and clock phase relationships are summarized in Table 113. This table summarizes the following for each setting of CPOL and CPHA.

- When the first data bit is driven.
- When all other data bits are driven.
- When data is sampled.

Table 113: SPI Data To Clock Phase Relationship

CPOL And CPHA Settings	First Data Driven	Other Data Driven	Data Sampled
CPOL = 0, CPHA = 0	Prior to first SCK rising edge	SCK falling edge	SCK rising edge
CPOL = 0, CPHA = 1	First SCK rising edge	SCK rising edge	SCK falling edge
CPOL = 1, CPHA = 0	Prior to first SCK falling edge	SCK rising edge	SCK falling edge
CPOL = 1, CPHA = 1	First SCK falling edge	SCK falling edge	SCK rising edge

The definition of when an 8 bit transfer starts and stops is dependent on whether a device is a master or a slave, and the setting of the CPHA variable.

When a device is a master, the start of a transfer is indicated by the master having a byte of data that is ready to be transmitted. At this point, the master can activate the clock, and begin the transfer. The transfer ends when the last clock cycle of the transfer is complete.

When a device is a slave, and CPHA is set to 0, the transfer starts when the SSEL signal goes active, and ends when SSEL goes inactive. When a device is a slave, and CPHA is set to 1, the transfer starts on the first clock edge when the slave is selected, and ends on the last clock edge where data is sampled.

SPI Peripheral Details

General Information

There are four registers that control the SPI peripheral. They are described in detail in "Register Description" section.

The SPI control register contains a number of programmable bits used to control the function of the SPI block. The settings for this register must be set up prior to a given data transfer taking place.

The SPI status register contains read only bits that are used to monitor the status of the SPI interface, including normal functions, and exception conditions. The primary purpose of this register is to detect completion of a data transfer. This is indicated by the SPIF bit. The remaining bits in the register are exception condition indicators. These exceptions will be described later in this section.

The SPI data register is used to provide the transmit and receive data bytes. An internal shift register in the SPI block logic is used for the actual transmission and reception of the serial data. Data is written to the SPI data register for the transmit case. There is no buffer between the data register and the internal shift register. A write to the data register goes directly into the internal shift register. Therefore, data should only be written to this register when a transmit is not currently in progress. Read data is buffered. When a transfer is complete, the receive data is transferred to a single byte data buffer, where it is later read. A read of the SPI data register returns the value of the read data buffer.

The SPI clock counter register controls the clock rate when the SPI block is in master mode. This needs to be set prior to a transfer taking place, when the SPI block is a master. This register has no function when the SPI block is a slave.

The I/Os for this implementation of SPI are standard CMOS I/Os. The open drain SPI option is not implemented in this design. When a device is set up to be a slave, its I/Os are only active when it is selected by the SSEL signal being active.

Master Operation

The following sequence describes how one should process a data transfer with the SPI block when it is set up to be the master. This process assumes that any prior data transfer has already completed.

1. Set the SPI clock counter register to the desired clock rate.
2. Set the SPI control register to the desired settings.
3. Write the data to transmitted to the SPI data register. This write starts the SPI data transfer.
4. Wait for the SPIF bit in the SPI status register to be set to 1. The SPIF bit will be set after the last cycle of the SPI data transfer.
5. Read the SPI status register.
6. Read the received data from the SPI data register (optional).
7. Go to step 3 if more data is required to transmit.

Note that a read or write of the SPI data register is required in order to clear the SPIF status bit. Therefore, if the optional read of the SPI data register does not take place, a write to this register is required in order to clear the SPIF status bit.

Slave Operation

The following sequence describes how one should process a data transfer with the SPI block when it is set up to be a slave. This process assumes that any prior data transfer has already completed. It is required that the system clock driving the SPI logic be at least 8X faster than the SPI.

1. Set the SPI control register to the desired settings.
2. Write the data to transmitted to the SPI data register (optional). Note that this can only be done when a slave SPI transfer is not in progress.
3. Wait for the SPIF bit in the SPI status register to be set to 1. The SPIF bit will be set after the last sampling clock edge of the SPI data transfer.
4. Read the SPI status register.
5. Read the received data from the SPI data register (optional).
6. Go to step 2 if more data is required to transmit.

Note that a read or write of the SPI data register is required in order to clear the SPIF status bit. Therefore, at least one of the optional reads or writes of the SPI data register must take place, in order to clear the SPIF status bit.

Exception Conditions

Read Overrun - A read overrun occurs when the SPI block internal read buffer contains data that has not been read by the processor, and a new transfer has completed. The read buffer containing valid data is indicated by the SPIF bit in the status register being active. When a transfer completes, the SPI block needs to move the received data to the read buffer. If the SPIF bit is active (the read buffer is full), the new receive data will be lost, and the read overrun (ROVR) bit in the status register will be activated.

Write Collision - As stated previously, there is no write buffer between the SPI block bus interface, and the internal shift register. As a result, data must not be written to the SPI data register when a SPI data transfer is currently in progress. The time frame where data cannot be written to the SPI data register is from when the transfer starts, until after the status register has been read when the SPIF status is active. If the SPI data register is written in this time frame, the write data will be lost, and the write collision (WCOL) bit in the status register will be activated.

Mode Fault - The SSEL signal must always be inactive when the SPI block is a master. If the SSEL signal goes active, when the SPI block is a master, this indicates another master has selected the device to be a slave. This condition is known as a mode fault. When a mode fault is detected, the mode fault (MODF) bit in the status register will be activated, the SPI signal drivers will be de-activated, and the SPI mode will be changed to be a slave.

Slave Abort - A slave transfer is considered to be aborted, if the SSEL signal goes inactive before the transfer is complete. In the event of a slave abort, the transmit and receive data for the transfer that was in progress are lost, and the slave abort (ABRT) bit in the status register will be activated.

PIN DESCRIPTION

Table 114: SPI Pin Description

Pin Name	Type	Pin Description
SCK1, SCK0	Input/ Output	**Serial Clock.** The SPI is a clock signal used to synchronize the transfer of data across the SPI interface. The SPI is always driven by the master and received by the slave. The clock is programmable to be active high or active low. The SPI is only active during a data transfer. Any other time, it is either in its inactive state, or tri-stated.
SSEL1, SSEL0	Input	**Slave Select.** The SPI slave select signal is an active low signal that indicates which slave is currently selected to participate in a data transfer. Each slave has its own unique slave select signal input. The SSEL must be low before data transactions begin and normally stays low for the duration of the transaction. If the SSEL signal goes high any time during a data transfer, the transfer is considered to be aborted. In this event, the slave returns to idle, and any data that was received is thrown away. There are no other indications of this exception. This signal is not directly driven by the master. It could be driven by a simple general purpose I/O under software control. **Note: LPC2119/2129/2194/2292/2294 configured to operate as SPI master MUST select SSEL functionality on an apropriate pin and have HIGH level on this pin in order to act as a master.**
MISO1, MISO0	Input/ Output	**Master In Slave Out.** The MISO signal is a unidirectional signal used to transfer serial data from the slave to the master. When a device is a slave, serial data is output on this signal. When a device is a master, serial data is input on this signal. When a slave device is not selected, the slave drives the signal high impedance.
MOSI1, MOSI0	Input/ Output	**Master Out Slave In.** The MOSI signal is a unidirectional signal used to transfer serial data from the master to the slave. When a device is a master, serial data is output on this signal. When a device is a slave, serial data is input on this signal.

REGISTER DESCRIPTION

The SPI contains 5 registers as shown in Table 115. All registers are byte, half word and word accessible.

Table 115: SPI Register Map

Generic Name	Description	Access	Reset Value*	SPI0 Address & Name	SPI1 Address & Name
SPCR	SPI Control Register. This register controls the operation of the SPI.	Read/ Write	0	0xE0020000 S0SPCR	0xE0030000 S1SPCR
SPSR	SPI Status Register. This register shows the status of the SPI.	Read Only	0	0xE0020004 S0SPSR	0xE0030004 S1SPSR
SPDR	SPI Data Register. This bi-directional register provides the transmit and receive data for the SPI. Transmit data is provided to the SPI by writing to this register. Data received by the SPI can be read from this register.	Read/ Write	0	0xE0020008 S0SPDR	0xE0030008 S1SPDR
SPCCR	SPI Clock Counter Register. This register controls the frequency of a master's SCK.	Read/ Write	0	0xE002000C S0SPCCR	0xE003000C S1SPCCR
SPINT	SPI Interrupt Flag. This register contains the interrupt flag for the SPI interface.	Read/ Write	0	0xE002001C S0SPINT	0xE003001C S1SPINT

*Reset Value refers to the data stored in used bits only. It does not include reserved bits content.

SPI Control Register (S0SPCR - 0xE0020000, S1SPCR - 0xE0030000)

The SPCR register controls the operation of the SPI as per the configuration bits setting.

Table 116: SPI Control Register (S0SPCR - 0xE0020000, S1SPCR - 0xE0030000)

SPCR	Function	*Description*	Reset Value
2:0	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
3	CPHA	Clock phase control determines the relationship between the data and the clock on SPI transfers, and controls when a slave transfer is defined as starting and ending. When 1, data is sampled on the second clock edge of the SCK. A transfer starts with the first clock edge, and ends with the last sampling edge when the SSEL signal is active. When 0, data is sampled on the first clock edge of SCK. A transfer starts and ends with activation and deactivation of the SSEL signal.	0
4	CPOL	Clock polarity control. When 1, SCK is active low. When 0, SCK is active high.	0
5	MSTR	Master mode select. When 1, the SPI operates in Master mode. When 0, the SPI operates in Slave mode.	0
6	LSBF	LSB First controls which direction each byte is shifted when transferred. When 1, SPI data is transferred LSB (bit 0) first. When 0, SPI data is transferred MSB (bit 7) first.	0
7	SPIE	Serial peripheral interrupt enable. When 1, a hardware interrupt is generated each time the SPIF or MODF bits are activated. When 0, SPI interrupts are inhibited.	0

SPI Status Register (S0SPSR - 0xE0020004, S1SPSR - 0xE0030004)

The SPSR register controls the operation of the SPI as per the configuration bits setting.

Table 117: SPI Status Register (S0SPSR - 0xE0020004, S1SPSR - 0xE0030004)

SPSR	Function	Description	Reset Value
2:0	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
3	ABRT	Slave abort. When 1, this bit indicates that a slave abort has occurred. This bit is cleared by reading this register.	0
4	MODF	Mode fault. when 1, this bit indicates that a Mode fault error has occurred. This bit is cleared by reading this register, then writing the SPI control register.	0
5	ROVR	Read overrun. When 1, this bit indicates that a read overrun has occurred. This bit is cleared by reading this register.	0
6	WCOL	Write collision. When 1, this bit indicates that a write collision has occurred. This bit is cleared by reading this register, then accessing the SPI data register.	0
7	SPIF	SPI transfer complete flag. When 1, this bit indicates when a SPI data transfer is complete. When a master, this bit is set at the end of the last cycle of the transfer. When a slave, this bit is set on the last data sampling edge of the SCK. This bit is cleared by first reading this register, then accessing the SPI data register. **Note:** this is not the SPI interrupt flag. This flag is found in the SPINT registrer.	0

SPI Data Register (S0SPDR - 0xE0020008, S1SPDR - 0xE0030008)

This bi-directional data register provides the transmit and receive data for the SPI. Transmit data is provided to the SPI by writing to this register. Data received by the SPI can be read from this register. When a master, a write to this register will start a SPI data transfer. Writes to this register will be blocked from when a data transfer starts to when the SPIF status bit is set, and the status register has not been read.

Table 118: SPI Data Register (S0SPDR - 0xE0020008, S1SPDR - 0xE0030008)

SPDR	Function	Description	Reset Value
7:0	Data	SPI Bi-directional data port	0

SPI Clock Counter Register (S0SPCCR - 0xE002000C, S1SPCCR - 0xE003000C)

This register controls the frequency of a master's SCK. The register indicates the number of pclk cycles that make up an SPI clock. The value of this register must always be an even number. As a result, bit 0 must always be 0. The value of the register must also always be greater than or equal to 8. Violations of this can result in unpredictable behavior.

Table 119: SPI Clock Counter Register (S0SPCCR - 0xE002000C, S1SPCCR - 0xE003000C)

SPCCR	Function	Description	Reset Value
7:0	Counter	SPI Clock counter setting	0

The SPI rate may be calculated as: PCLK rate / SPCCR value. The pclk rate is CCLK / VPB divider rate as determined by the VPBDIV register contents.

SPI Interrupt Register (S0SPINT - 0xE002001C, S1SPINT - 0xE003001C)

This register contains the interrupt flag for the SPI interface.

Table 120: SPI Interrupt Register (S0SPINT - 0xE002001C, S1SPINT - 0xE003001C)

SPINT	Function	Description	Reset Value
0	SPI Interrupt	SPI interrupt flag. Set by the SPI interface to generate an interrupt. Cleared by writing a 1 to this bit. Note: this bit will be set once when SPIE=1 and at least one of SPIF and WCOL bits is 1. However, only when SPI Interrupt bit is set and SPI Interrupt is enabled in the VIC, SPI based interrupt can be processed by interrupt handling software.	0
7:1	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

ARCHITECTURE

The block diagram of the SPI solution implemented in SPI0 and SPI1 interfaces is shown in the Figure 34.

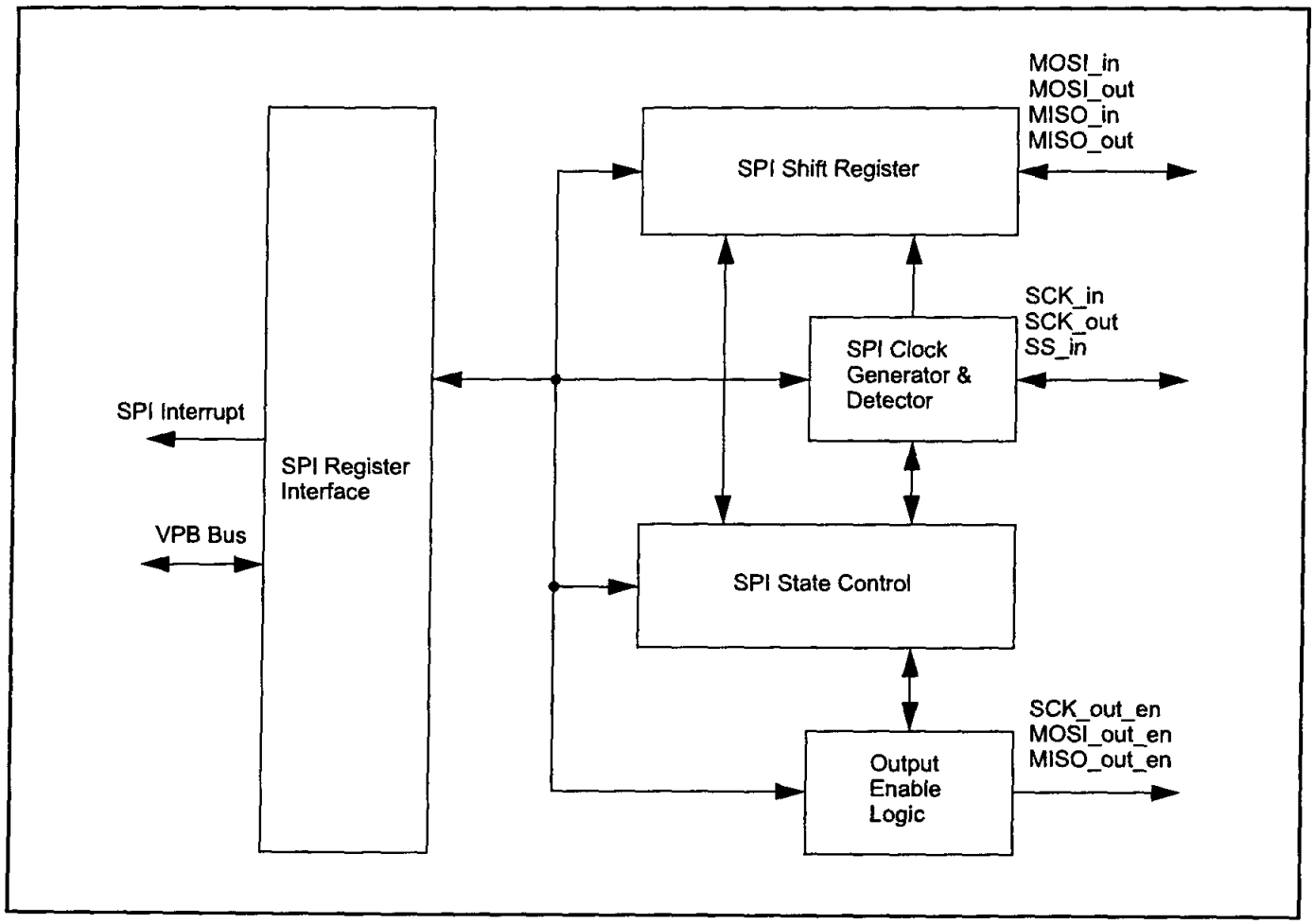


Figure 34: SPI Block Diagram

15. TIMER0 AND TIMER1

Timer0 and Timer1 are functionally identical except for the peripheral base address.

FEATURES

- A 32-bit Timer/Counter with a programmable 32-bit Prescaler.
- Up to four 32-bit capture channels per timer, that can take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.
- Four 32-bit match registers that allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- Up to four external outputs corresponding to match registers, with the following capabilities:
  - Set low on match.
  - Set high on match.
  - Toggle on match.
  - Do nothing on match.

APPLICATIONS

- Interval Timer for counting internal events.
- Pulse Width Demodulator via Capture inputs.
- Free running timer.

DESCRIPTION

The Timer is designed to count cycles of the peripheral clock (pclk) and optionally generate interrupts or perform other actions at specified timer values, based on four match registers. It also includes four capture inputs to trap the timer value when an input signal transitions, optionally generating an interrupt.

PIN DESCRIPTION

Table 156 gives a brief summary of each of the Timer related pins.

Table 156: Pin summary

Pin name	Pin direction	Pin Description
CAP0.3..0 CAP1.3..0	Input	**Capture Signals**- A transition on a capture pin can be configured to load one of the Capture Registers with the value in the Timer Counter and optionally generate an interrupt. Capture functionality can be selected from a number of pins. Also, if for example 2 pins are selected to provide CAP0.2 function in parallel, their inputs will be logically ored and this value will be processed as a single input. CAP0.0 can be selected from/on up to 3 pins at the same time. CAP0.1 can be selected from/on up to 2 pins at the same time. CAP0.2 can be selected from/on up to 3 pins at the same time. CAP0.3 can be selected from/on 1 pin. CAP1.0 can be selected from/on 1 pin. CAP1.1 can be selected from/on 1 pin. CAP1.2 can be selected from/on up to 2 pins at the same time. CAP1.3 can be selected from/on up to 2 pins at the same time.
MAT0.3...0 MAT1.0...0	Output	**External Match Output 0/1**- When a match register 0/1 (MR3:0) equals the timer counter (TC) this output can either toggle, go low, go high, or do nothing. The External Match Register (EMR) controls the functionality of this output. Match Output functionality can be selected on a number of pins in parallel. It is also possible for example, to have 2 pins selected at the same time so that they provide MAT1.3 function in parallel. MAT0.0 can be selected on up to 2 pins at the same time. MAT0.1 can be selected on up to 2 pins at the same time. MAT0.2 can be selected on up to 2 pins at the same time. MAT0.3 can be selected on 1 pin. MAT1.0 can be selected on 1 pin. MAT1.1 can be selected on 1 pin. MAT1.2 can be selected on up to 2 pins at the same time. MAT1.3 can be selected on up to 2 pins at the same time.

REGISTER DESCRIPTION

Each Timer contains the registers shown in Table 157. More detailed descriptions follow.

Table 157: TIMER0 and TIMER1 Register Map

Generic Name	Description	Access	Reset Value*	TIMER0 Address & Name	TIMER1 Address & Name
IR	Interrupt Register. The IR can be written to clear interrupts. The IR can be read to identify which of eight possible interrupt sources are pending.	R/W	0	0xE0004000 T0IR	0xE0008000 T1IR
TCR	Timer Control Register. The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	R/W	0	0xE0004004 T0TCR	0xE0008004 T1TCR
TC	Timer Counter. The 32-bit TC is incremented every PR+1 cycles of pclk. The TC is controlled through the TCR.	RW	0	0xE0004008 T0TC	0xE0008008 T1TC
PR	Prescale Register. The TC is incremented every PR+1 cycles of pclk.	R/W	0	0xE000400C T0PR	0xE000800C T1PR
PC	Prescale Counter. The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented.	R/W	0	0xE0004010 T0PC	0xE0008010 T1PC
MCR	Match Control Register. The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	R/W	0	0xE0004014 T0MCR	0xE0008014 T1MCR
MR0	Match Register 0. MR0 can be enabled through the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt every time MR0 matches the TC.	R/W	0	0xE0004018 T0MR0	0xE0008018 T1MR0
MR1	Match Register 1. See MR0 description.	R/W	0	0xE000401C T0MR1	0xE000801C T1MR1
MR2	Match Register 2. See MR0 description.	R/W	0	0xE0004020 T0MR2	0xE0008020 T1MR2
MR3	Match Register 3. See MR0 description.	R/W	0	0xE0004024 T0MR3	0xE0008024 T1MR3
CCR	Capture Control Register. The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether or not an interrupt is generated when a capture takes place.	R/W	0	0xE0004028 T0CCR	0xE0008028 T1CCR
CR0	Capture Register 0. CR0 is loaded with the value of TC when there is an event on the CAP0.0(CAP1.0) input.	RO	0	0xE000402C T0CR0	0xE000802C T1CR0
CR1	Capture Register 1. See CR0 description.	RO	0	0xE0004030 T0CR1	0xE0008030 T1CR1
CR2	Capture Register 2. See CR0 description.	RO	0	0xE0004034 T0CR2	0xE0008034 T1CR2
CR3	Capture Register 3. See CR0 description.	RO	0	0xE0004038 T0CR3	0xE0008038 T1CR3
EMR	External Match Register. The EMR controls the external match pins MAT0.0-3 (MAT1.0-3).	R/W	0	0xE000403C T0EMR	0xE000803C T1EMR

*Reset Value refers to the data stored in used bits only. It does not include reserved bits content.

Interrupt Register (IR: TIMER0 - T0IR: 0xE0004000; TIMER1 - T1IR: 0xE0008000)

The Interrupt Register consists of four bits for the match interrupts and four bits for the capture interrupts. If an interrupt is generated then the corresponding bit in the IR will be high. Otherwise, the bit will be low. Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

Table 158: Interrupt Register (IR: TIMER0 - T0IR: 0xE0004000; TIMER1 - T1IR: 0xE0008000)

IR	Function	Description	Reset Value
0	MR0 Interrupt	Interrupt flag for match channel 0.	0
1	MR1 Interrupt	Interrupt flag for match channel 1.	0
2	MR2 Interrupt	Interrupt flag for match channel 2.	0
3	MR3 Interrupt	Interrupt flag for match channel 3.	0
4	CR0 Interrupt	Interrupt flag for capture channel 0 event.	0
5	CR1 Interrupt	Interrupt flag for capture channel 1 event.	0
6	CR2 Interrupt	Interrupt flag for capture channel 2 event.	0
7	CR3 Interrupt	Interrupt flag for capture channel 3 event.	0

Timer Control Register (TCR: TIMER0 - T0TCR: 0xE0004004; TIMER1 - T1TCR: 0xE0008004)

The Timer Control Register (TCR) is used to control the operation of the Timer Counter.

Table 159: Timer Control Register (TCR: TIMER0 - T0TCR: 0xE0004004; TIMER1 - T1TCR: 0xE0008004)

TCR	Function	Description	Reset Value
0	Counter Enable	When one, the Timer Counter and Prescale Counter are enabled for counting. When zero, the counters are disabled.	0
1	Counter Reset	When one, the Timer Counter and the Prescale Counter are synchronously reset on the next positive edge of pclk. The counters remain reset until TCR[1] is returned to zero.	0

Timer Counter (TC: TIMER0 - T0TC: 0xE0004008; TIMER1 - T1TC: 0xE0008008)

The 32-bit Timer Counter is incremented when the Prescale Counter reaches its terminal count. Unless it is reset before reaching its upper limit, the TC will count up through the value 0xFFFFFFFF and then wrap back to the value 0x00000000. This event does not cause an interrupt, but a Match register can be used to detect an overflow if needed.

Prescale Register (PR: TIMER0 - T0PR: 0xE000400C; TIMER1 - T1PR: 0xE000800C)

The 32-bit Prescale Register specifies the maximum value for the Prescale Counter.

Prescale Counter Register (PC: TIMER0 - T0PC: 0xE0004010; TIMER1 - T1PC: 0xE0008010)

The 32-bit Prescale Counter controls division of pclk by some constant value before it is applied to the Timer Counter. This allows control of the relationship of the resolution of the timer versus the maximum time before the timer overflows. The Prescale Counter is incremented on every pclk. When it reaches the value stored in the Prescale Register, the Timer Counter is incremented and the Prescale Counter is reset on the next pclk. This causes the TC to increment on every pclk when PR = 0, every 2 pclks when PR = 1, etc.

Match Registers (MR0 - MR3)

The Match register values are continuously compared to the Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the Timer Counter, or stop the timer. Actions are controlled by the settings in the MCR register.

Match Control Register (MCR: TIMER0 - T0MCR: 0xE0004014; TIMER1 - T1MCR: 0xE0008014)

The Match Control Register is used to control what operations are performed when one of the Match Registers matches the Timer Counter. The function of each of the bits is shown in Table 160.

Table 160: Match Control Register (MCR: TIMER0 - T0MCR: 0xE0004014; TIMER1 - T1MCR: 0xE0008014)

MCR	Function	Description	Reset Value
0	Interrupt on MR0	When one, an interrupt is generated when MR0 matches the value in the TC. When zero this interrupt is disabled.	0
1	Reset on MR0	When one, the TC will be reset if MR0 matches it. When zero this feature is disabled.	0
2	Stop on MR0	When one, the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC. When zero this feature is disabled.	0
3	Interrupt on MR1	When one, an interrupt is generated when MR1 matches the value in the TC. When zero this interrupt is disabled.	0
4	Reset on MR1	When one, the TC will be reset if MR1 matches it. When zero this feature is disabled.	0
5	Stop on MR1	When one, the TC and PC will be stopped and TCR[0] will be set to 0 if MR1 matches the TC. When zero this feature is disabled.	0
6	Interrupt on MR2	When one, an interrupt is generated when MR2 matches the value in the TC. When zero this interrupt is disabled.	0
7	Reset on MR2	When one, the TC will be reset if MR2 matches it. When zero this feature is disabled.	0
8	Stop on MR2	When one, the TC and PC will be stopped and TCR[0] will be set to 0 if MR2 matches the TC. When zero this feature is disabled.	0
9	Interrupt on MR3	When one, an interrupt is generated when MR3 matches the value in the TC. When zero this interrupt is disabled.	0
10	Reset on MR3	When one, the TC will be reset if MR3 matches it. When zero this feature is disabled.	0
11	Stop on MR3	When one, the TC and PC will be stopped and TCR[0] will be set to 0 if MR3 matches the TC. When zero this feature is disabled.	0

Capture Registers (CR0 - CR3)

Each Capture register is associated with a device pin and may be loaded with the Timer Counter value when a specified event occurs on that pin. The settings in the Capture Control Register register determine whether the capture function is enabled, and whether a capture event happens on the rising edge of the associated pin, the falling edge, or on both edges.

Capture Control Register (CCR: TIMER0 - T0CCR: 0xE0004028; TIMER1 - T1CCR: 0xE0008028)

The Capture Control Register is used to control whether one of the four Capture Registers is loaded with the value in the Timer Counter when the capture event occurs, and whether an interrupt is generated by the capture event. Setting both the rising and falling bits at the same time is a valid configuration, resulting in a capture event for both edges. In the description below, "n" represents the Timer number, 0 or 1.

Table 161: Capture Control Register (CCR: TIMER0 - T0CCR: 0xE0004028; TIMER1 - T1CCR: 0xE0008028)

CCR	Function	Description	Reset Value
0	Capture on CAPn.0 rising edge	When one, a sequence of 0 then 1 on CAPn.0 will cause CR0 to be loaded with the contents of the TC. When zero, this feature is disabled.	0
1	Capture on CAPn.0 falling edge	When one, a sequence of 1 then 0 on CAPn.0 will cause CR0 to be loaded with the contents of TC. When zero, this feature is disabled.	0
2	Interrupt on CAPn.0 event	When one, a CR0 load due to a CAPn.0 event will generate an interrupt. When zero, this feature is disabled.	0
3	Capture on CAPn.1 rising edge	When one, a sequence of 0 then 1 on CAPn.1 will cause CR1 to be loaded with the contents of the TC. When zero, this feature is disabled.	0
4	Capture on CAPn.1 falling edge	When one, a sequence of 1 then 0 on CAPn.1 will cause CR1 to be loaded with the contents of TC. When zero, this feature is disabled.	0
5	Interrupt on CAPn.1 event	When one, a CR1 load due to a CAPn.1 event will generate an interrupt. When zero, this feature is disabled.	0
6	Capture on CAPn.2 rising edge	When one, a sequence of 0 then 1 on CAPn.2 will cause CR2 to be loaded with the contents of the TC. When zero, this feature is disabled.	0
7	Capture on CAPn.2 falling edge	When one, a sequence of 1 then 0 on CAPn.2 will cause CR2 to be loaded with the contents of TC. When zero, this feature is disabled.	0
8	Interrupt on CAPn.2 event	When one, a CR2 load due to a CAPn.2 event will generate an interrupt. When zero, this feature is disabled.	0
9	Capture on CAPn.3 rising edge	When one, a sequence of 0 then 1 on CAPn.3 will cause CR3 to be loaded with the contents of TC. When zero, this feature is disabled.	0
10	Capture on CAPn.3 falling edge	When one, a sequence of 1 then 0 on CAPn.3 will cause CR3 to be loaded with the contents of TC. When zero, this feature is disabled.	0
11	Interrupt on CAPn.3 event	When one, a CR3 load due to a CAPn.3 event will generate an interrupt. When zero, this feature is disabled.	0

External Match Register (EMR: TIMER0 - T0EMR: 0xE000403C; TIMER1 - T1EMR: 0xE000803C)

The External Match Register provides both control and status of the external match pins M(0-3).

Table 162: External Match Register (EMR: TIMER0 - T0EMR: 0xE000403C; TIMER1 - T1EMR: 0xE000803C)

EMR	Function	Description	Reset Value
0	External Match 0	This bit reflects the state of output MAT0.0/MAT1.0, whether or not this output is connected to its pin. When a match occurs for MR0, this output of the timer can either toggle, go low, go high, or do nothing. Bits EMR[4:5] control the functionality of this output.	0
1	External Match 1	This bit reflects the state of output MAT0.1/MAT1.1, whether or not this output is connected to its pin. When a match occurs for MR1, this output of the timer can either toggle, go low, go high, or do nothing. Bits EMR[6:7] control the functionality of this output.	0
2	External Match 2	This bit reflects the state of output MAT0.2/MAT1.2, whether or not this output is connected to its pin. When a match occurs for MR2, this output of the timer can either toggle, go low, go high, or do nothing. Bits EMR[8:9] control the functionality of this output.	0
3	External Match 3	This bit reflects the state of output MAT0.3/MAT1.3, whether or not this output is connected to its pin. When a match occurs for MR3, this output of the timer can either toggle, go low, go high, or do nothing. Bits EMR[10:11] control the functionality of this output.	0
5:4	External Match Control 0	Determines the functionality of External Match 0. Table 163 shows the encoding of these bits.	0
7:6	External Match Control 1	Determines the functionality of External Match 1. Table 163 shows the encoding of these bits.	0
9:8	External Match Control 2	Determines the functionality of External Match 2. Table 163 shows the encoding of these bits.	0
11:10	External Match Control 3	Determines the functionality of External Match 3. Table 163 shows the encoding of these bits.	0

Table 163: External Match Control

EMR[11:10], EMR[9:8], EMR[7:6], or EMR[5:4]	Function
0 0	Do Nothing
0 1	Clear corresponding External Match output to 0 (LOW if pinned out)
1 0	Set corresponding External Match output to 1(HIGH if pinned out)
1 1	Toggle corresponding External Match output

EXAMPLE TIMER OPERATION

Figure 39 shows a timer configured to reset the count and generate an interrupt on match. The prescaler is set to 2 and the match register set to 6. At the end of the timer cycle where the match occurs, the timer count is reset. This gives a full length cycle to the match value. The interrupt indicating that a match occurred is generated in the next clock after the timer reached the match value.

Figure 40 shows a timer configured to stop and generate an interrupt on match. The prescaler is again set to 2 and the match register set to 6. In the next clock after the timer reaches the match value, the timer enable bit in TCR is cleared, and the interrupt indicating that a match occurred is generated.

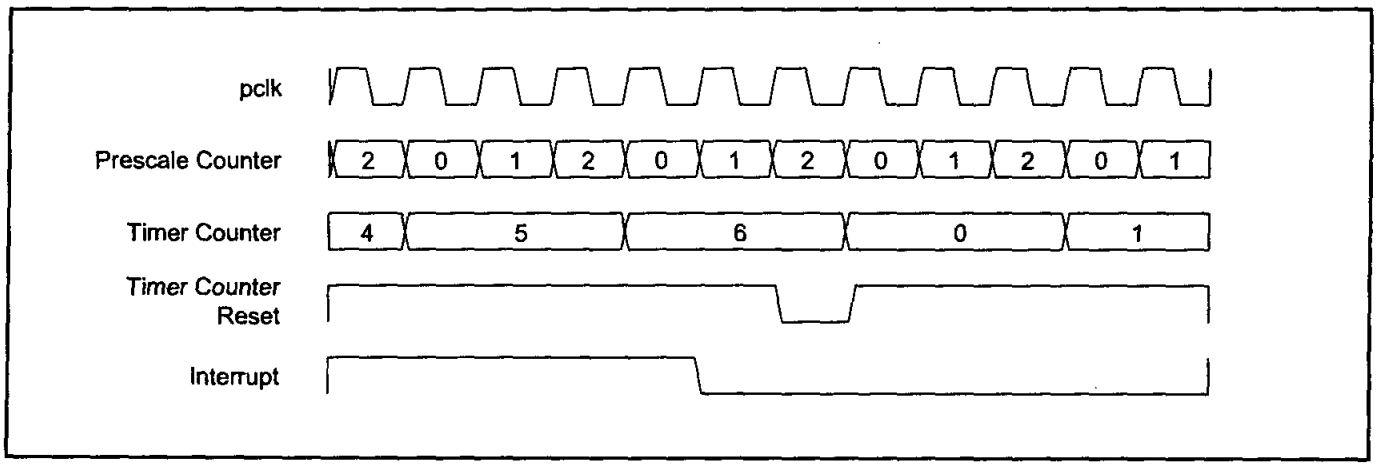


Figure 39: A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled.

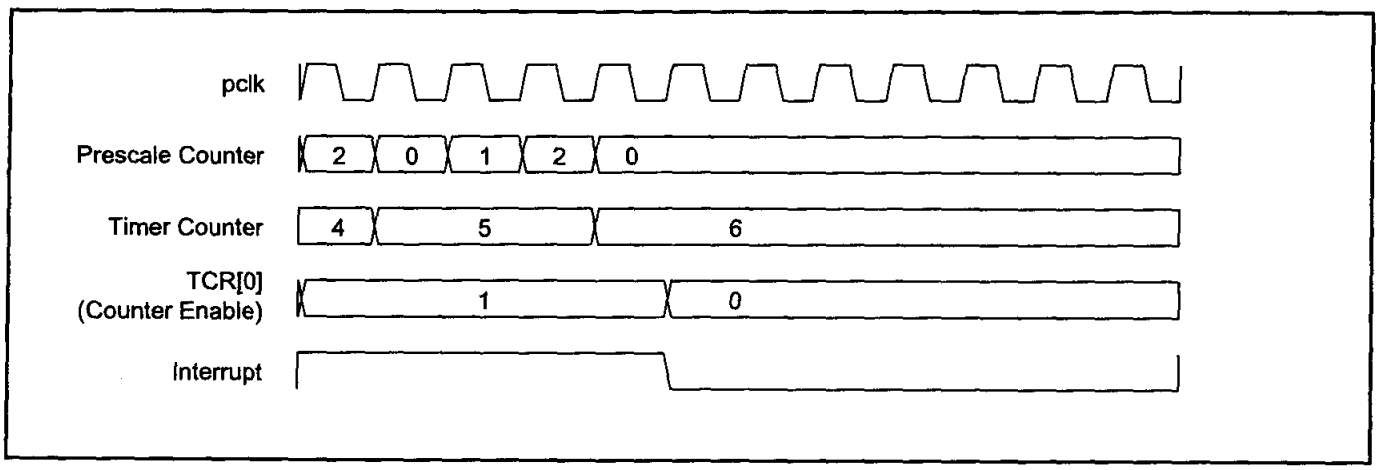


Figure 40: A timer cycle in which PR=2, MRx=6, and both interrupt and stop on match are enabled.

ARCHITECTURE

The block diagram for TIMER0 and TIMER1 is shown in Figure 41.

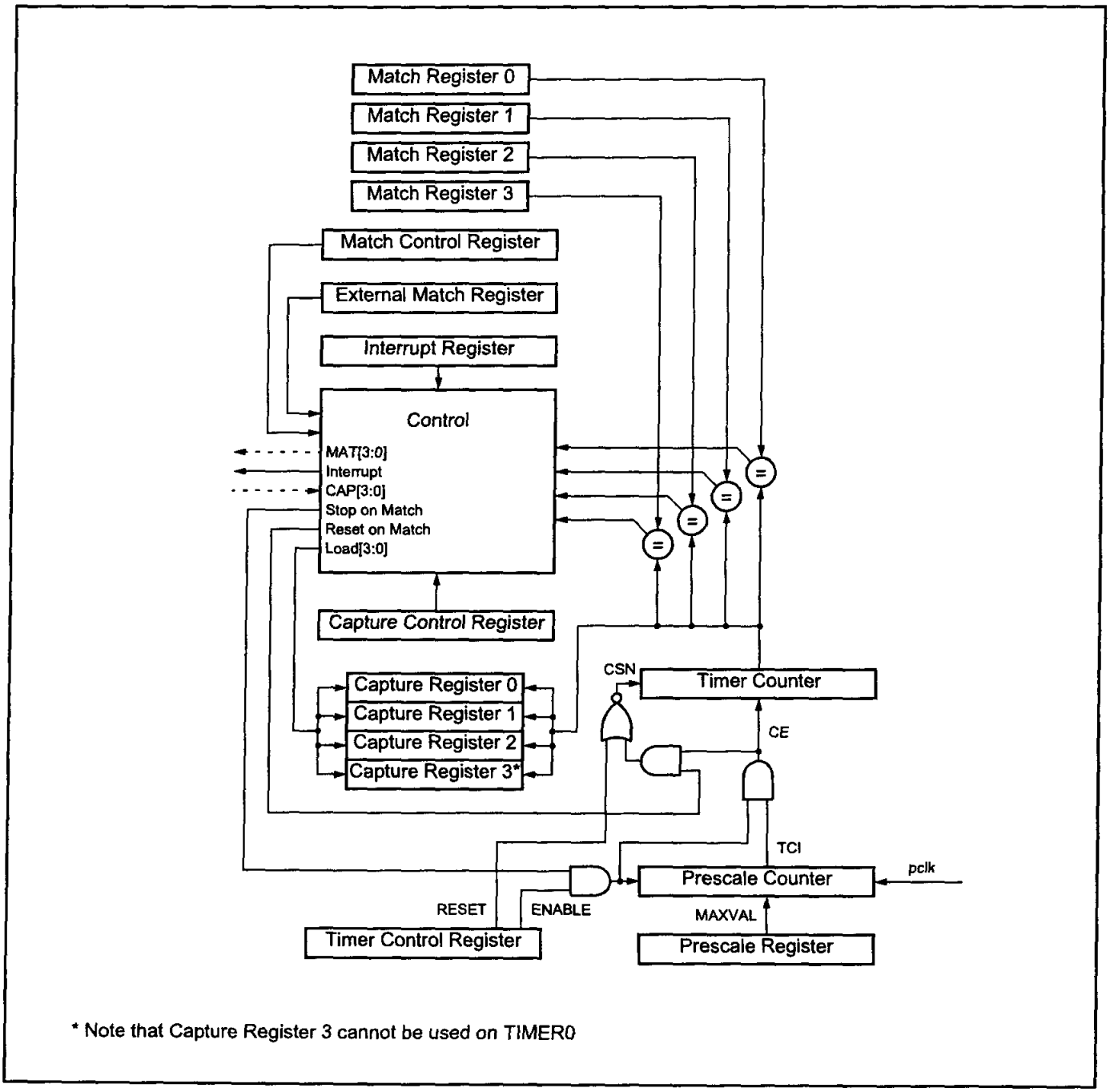


Figure 41: Timer block diagram

16. PULSE WIDTH MODULATOR (PWM)

LPC2119/2129/2194/2292/2294 Pulse Width Modulator is based on standard Timer 0/1 described in previous chapter. Application can choose among PWM and match functions available .

FEATURES

- Seven match registers allow up to 6 single edge controlled or 3 double edge controlled PWM outputs, or a mix of both types. The match registers also allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- An external output for each match register with the following capabilities:
  - Set low on match.
  - Set high on match.
  - Toggle on match.
  - Do nothing on match.
- Supports single edge controlled and/or double edge controlled PWM outputs. Single edge controlled PWM outputs all go high at the beginning of each cycle unless the output is a constant low. Double edge controlled PWM outputs can have either edge occur at any position within a cycle. This allows for both positive going and negative going pulses.
- Pulse period and width can be any number of timer counts. This allows complete flexibility in the trade-off between resolution and repetition rate. All PWM outputs will occur at the same repetition rate.
- Double edge controlled PWM outputs can be programmed to be either positive going or negative going pulses.
- Match register updates are synchronized with pulse outputs to prevent generation of erroneous pulses. Software must "release" new match values before they can become effective.
- May be used as a standard timer if the PWM mode is not enabled.
- A 32-bit Timer/Counter with a programmable 32-bit Prescaler.
- Four 32-bit capture channels take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.

DESCRIPTION

The PWM is based on the standard Timer block and inherits all of its features, although only the PWM function is pinned out on the LPC2119/2129/2194/2292/2294. The Timer is designed to count cycles of the peripheral clock (pclk) and optionally generate interrupts or perform other actions when specified timer values occur, based on seven match registers. It also includes four capture inputs to save the timer value when an input signal transitions, and optionally generate an interrupt when those events occur. The PWM function is in addition to these features, and is based on match register events.

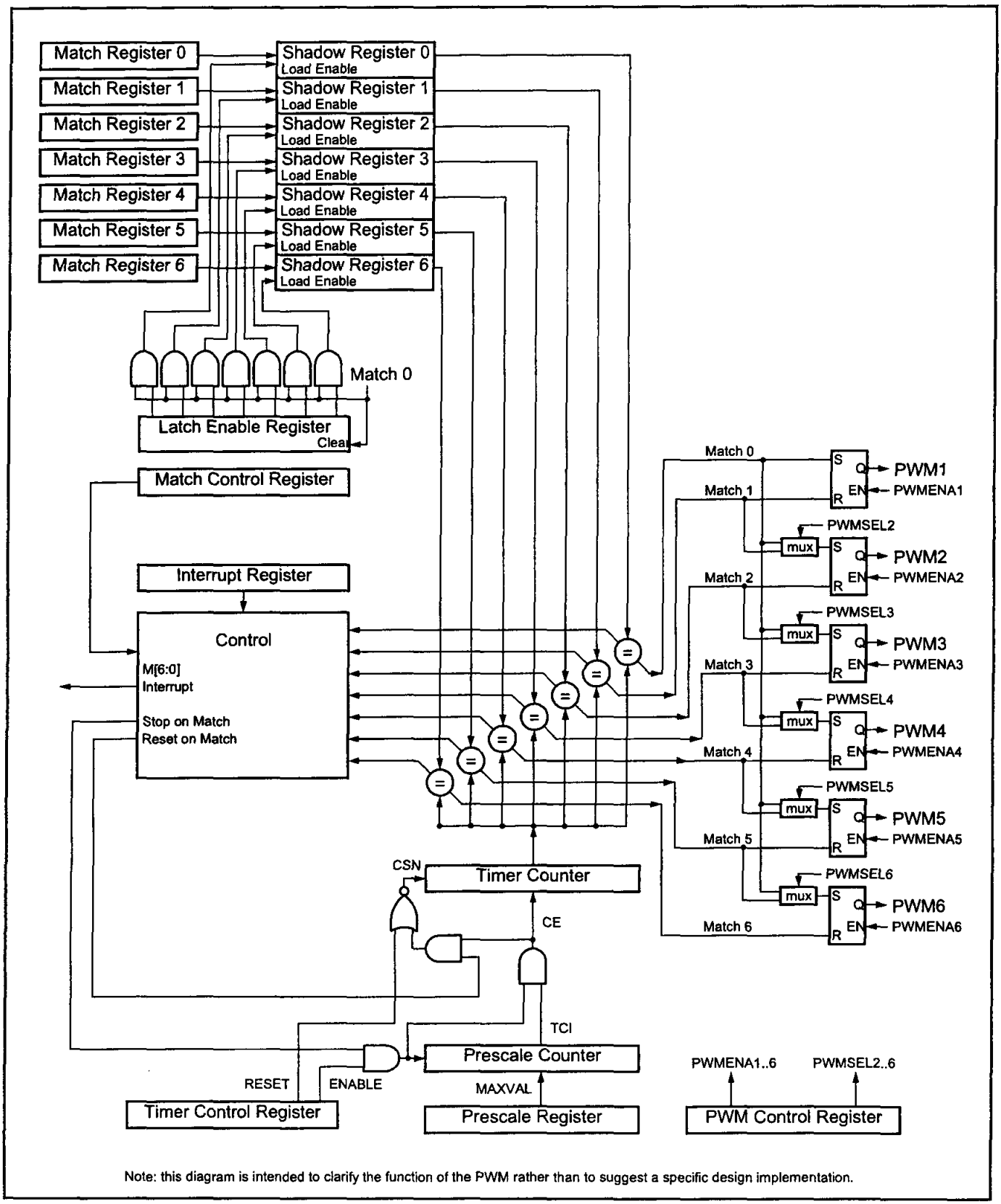
The ability to separately control rising and falling edge locations allows the PWM to be used for more applications. For instance, multi-phase motor control typically requires three non-overlapping PWM outputs with individual control of all three pulse widths and positions.

Two match registers can be used to provide a single edge controlled PWM output. One match register (PWMMR0) controls the PWM cycle rate, by resetting the count upon match. The other match register controls the PWM edge position. Additional single edge controlled PWM outputs require only one match register each, since the repetition rate is the same for all PWM outputs. Multiple single edge controlled PWM outputs will all have a rising edge at the beginning of each PWM cycle, when an PWMMR0 match occurs.

Three match registers can be used to provide a PWM output with both edges controlled. Again, the PWMMR0 match register controls the PWM cycle rate. The other match registers control the two PWM edge positions. Additional double edge controlled PWM outputs require only two match registers each, since the repetition rate is the same for all PWM outputs.

With double edge controlled PWM outputs, specific match registers control the rising and falling edge of the output. This allows both positive going PWM pulses (when the rising edge occurs prior to the falling edge), and negative going PWM pulses (when the falling edge occurs prior to the rising edge).

Figure 42 shows the block diagram of the PWM. The portions that have been added to the standard timer block are on the right hand side and at the top of the diagram.



Note: this diagram is intended to clarify the function of the PWM rather than to suggest a specific design implementation.

Figure 42: PWM block diagram

A sample of how PWM values relate to waveform outputs is shown in Figure 43. PWM output logic is shown in Figure 42 that allows selection of either single or double edge controlled PWM outputs via the muxes controlled by the PWMSELn bits. The match register selections for various PWM outputs is shown in Table 164. This implementation supports up to N-1 single edge PWM outputs or (N-1)/2 double edge PWM outputs, where N is the number of match registers that are implemented. PWM types can be mixed if desired.

The waveforms below show a single PWM cycle and demonstrate PWM outputs under the following conditions:

- The timer is configured for PWM mode.
- Match 0 is configured to reset the timer/counter when a match event occurs.
- Control bits PWMSEL2 and PWMSEL4 are set.
- The Match register values are as follows:
  MR0= 100(PWM rate)
  MR1= 41, MR2 = 78(PWM2 output)
  MR3= 53, MR4 = 27(PWM4 output)
  MR5= 65(PWM5 output)

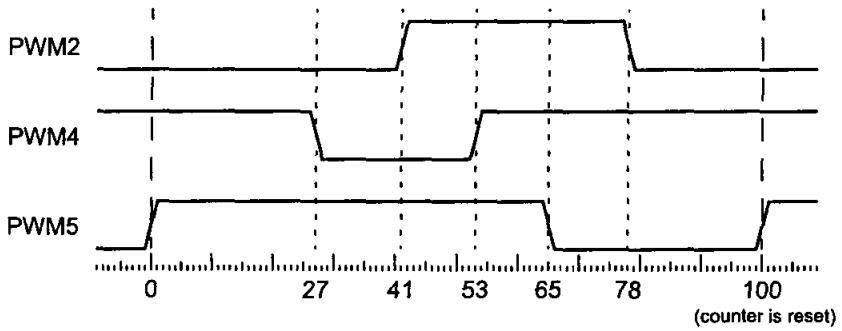


Figure 43: Sample PWM waveforms

Table 164: Set and Reset inputs for PWM Flip-Flops

PWM Channel	Single Edge PWM (PWMSELn = 0)		Double Edge PWM (PWMSELn = 1)	
	Set by	Reset by	Set by	Reset by
1	Match 0	Match 1	Match 0 [1]	Match 1 [1]
2	Match 0	Match 2	Match 1	Match 2
3	Match 0	Match 3	Match 2 [2]	Match 3 [2]
4	Match 0	Match 4	Match 3	Match 4
5	Match 0	Match 5	Match 4 [2]	Match 5 [2]
6	Match 0	Match 6	Match 5	Match 6

Notes:
1. Identical to single edge mode in this case since Match 0 is the neighboring match register. Essentially, PWM1 cannot be a double edged output.
2. It is generally not advantageous to use PWM channels 3 and 5 for double edge PWM outputs because it would reduce the number of double edge PWM outputs that are possible. Using PWM 2, PWM4, and PWM6 for double edge PWM outputs provides the most pairings.

Rules for Single Edge Controlled PWM Outputs

1. All single edge controlled PWM outputs go high at the beginning of a PWM cycle unless their match value is equal to 0.
2. Each PWM output will go low when its match value is reached. If no match occurs (i.e. the match value is greater than the PWM rate), the PWM output remains continuously high.

Rules for Double Edge Controlled PWM Outputs

Five rules are used to determine the next value of a PWM output when a new cycle is about to begin:

1. The match values for the next PWM cycle are used at the end of a PWM cycle (a time point which is coincident with the beginning of the next PWM cycle), except as noted in rule 3.
2. A match value equal to 0 or the current PWM rate (the same as the Match channel 0 value) have the same effect, except as noted in rule 3. For example, a request for a falling edge at the beginning of the PWM cycle has the same effect as a request for a falling edge at the end of a PWM cycle.
3. When match values are changing, if one of the "old" match values is equal to the PWM rate, it is used again once if the neither of the new match values are equal to 0 or the PWM rate, and there was no old match value equal to 0.
4. If both a set and a clear of a PWM output are requested at the same time, clear takes precedence. This can occur when the set and clear match values are the same as in, or when the set or clear value equals 0 and the other value equals the PWM rate.
5. If a match value is out of range (i.e. greater than the PWM rate value), no match event occurs and that match channel has no effect on the output. This means that the PWM output will remain always in one state, allowing always low, always high, or "no change" outputs.

PIN DESCRIPTION

Table 165 gives a brief summary of each of PWM related pins.

Table 165: Pin summary

Pin name	Pin direction	Pin Description
PWM1	Output	Output from PWM channel 1.
PWM2	Output	Output from PWM channel 2.
PWM3	Output	Output from PWM channel 3.
PWM4	Output	Output from PWM channel 4.
PWM5	Output	Output from PWM channel 5.
PWM6	Output	Output from PWM channel 6.

REGISTER DESCRIPTION

The PWM function adds new registers and registers bits as shown in Table 166 below.

Table 166: Pulse Width Modulator Register Map

Name	Description	Access	Reset Value*	Address
PWMIR	PWM Interrupt Register. The IR can be written to clear interrupts. The IR can be read to identify which of the possible interrupt sources are pending.	R/W	0	0xE0014000
PWMTCR	PWM Timer Control Register. The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	R/W	0	0xE0014004
PWMTC	PWM Timer Counter. The 32-bit TC is incremented every PR+1 cycles of pclk. The TC is controlled through the TCR.	RW	0	0xE0014008
PWMPR	PWM Prescale Register. The TC is incremented every PR+1 cycles of pclk.	R/W	0	0xE001400C
PWMPC	PWM Prescale Counter. The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented.	R/W	0	0xE0014010
PWMMCR	PWM Match Control Register. The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	R/W	0	0xE0014014
PWMMR0	PWM Match Register 0. MR0 can be enabled through MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt when it matches the TC. In addition, a match between MR0 and the TC sets all PWM outputs that are in single-edge mode, and sets PWM1 if it is in double-edge mode.	R/W	0	0xE0014018
PWMMR1	PWM Match Register 1. MR1 can be enabled through MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt when it matches the TC. In addition, a match between MR1 and the TC clears PWM1 in either single-edge mode or double-edge mode, and sets PWM2 if it is in double-edge mode.	R/W	0	0xE001401C
PWMMR2	PWM Match Register 2. MR2 can be enabled through MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt when it matches the TC. In addition, a match between MR2 and the TC clears PWM2 in either single-edge mode or double-edge mode, and sets PWM3 if it is in double-edge mode.	R/W	0	0xE0014020
PWMMR3	PWM Match Register 3. MR3 can be enabled through MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt when it matches the TC. In addition, a match between MR3 and the TC clears PWM3 in either single-edge mode or double-edge mode, and sets PWM4 if it is in double-edge mode.	R/W	0	0xE0014024
PWMMR4	PWM Match Register 4. MR4 can be enabled through MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt when it matches the TC. In addition, a match between MR4 and the TC clears PWM4 in either single-edge mode or double-edge mode, and sets PWM5 if it is in double-edge mode.	R/W	0	0xE0014040

Table 166: Pulse Width Modulator Register Map

Name	Description	Access	Reset Value*	Address
PWMMR5	PWM Match Register 5. MR5 can be enabled through MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt when it matches the TC. In addition, a match between MR5 and the TC clears PWM5 in either single-edge mode or double-edge mode, and sets PWM6 if it is in double-edge mode.	R/W	0	0xE0014044
PWMMR6	PWM Match Register 6. MR6 can be enabled through MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt when it matches the TC. In addition, a match between MR6 and the TC clears PWM6 in either single-edge mode or double-edge mode.	R/W	0	0xE0014048
PWMPCR	PWM Control Register. Enables PWM outputs and selects PWM channel types as either single edge or double edge controlled.	R/W	0	0xE001404C
PWMLER	PWM Latch Enable Register. Enables use of new PWM match values.	R/W	0	0xE0014050

*Reset Value refers to the data stored in used bits only. It does not include reserved bits content.

PWM Interrupt Register (PWMIR - 0xE0014000)

The PWM Interrupt Register consists of eleven bits (Table 167), seven for the match interrupts and four reserved for the future use. If an interrupt is generated then the corresponding bit in the PWMIR will be high. Otherwise, the bit will be low. Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

Table 167: PWM Interrupt Register (PWMIR - 0xE0014000)

PWMIR	Function	Description	Reset Value
0	PWMMR0 Interrupt	Interrupt flag for PWM match channel 0.	0
1	PWMMR1 Interrupt	Interrupt flag for PWM match channel 1.	0
2	PWMMR2 Interrupt	Interrupt flag for PWM match channel 2.	0
3	MR3 Interrupt	Interrupt flag for PWM match channel 3.	0
4	Reserved.	Application must not write 1 to this bit.	0
5	Reserved.	Application must not write 1 to this bit.	0
6	Reserved.	Application must not write 1 to this bit.	0
7	Reserved.	Application must not write 1 to this bit.	0
8	PWMMR4 Interrupt	Interrupt flag for PWM match channel 4.	0
9	PWMMR5 Interrupt	Interrupt flag for PWM match channel 5.	0
10	PWMMR6 Interrupt	Interrupt flag for PWM match channel 6.	0

PWM Timer Control Register (PWMTCR - 0xE0014004)

The PWM Timer Control Register (PWMTCR) is used to control the operation of the PWM Timer Counter. The function of each of the bits is shown in Table 168.

Table 168: PWM Timer Control Register (PWMTCR - 0xE0014004)

PWMTCR	Function	Description	Reset Value
0	Counter Enable	When one, the PWM Timer Counter and PWM Prescale Counter are enabled for counting. When zero, the counters are disabled.	0
1	Counter Reset	When one, the PWM Timer Counter and the PWM Prescale Counter are synchronously reset on the next positive edge of pclk. The counters remain reset until TCR[1] is returned to zero.	0
2	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
3	PWM Enable	When one, PWM mode is enabled. PWM mode causes shadow registers to operate in connection with the Match registers. A program write to a Match register will not have an effect on the Match result until the corresponding bit in PWMLER has been set, followed by the occurrence of a PWM Match 0 event. Note that the PWM Match register that determines the PWM rate (PWM Match 0) must be set up prior to the PWM being enabled. Otherwise a Match event will not occur to cause shadow register contents to become effective.	0

PWM Timer Counter (PWMTC - 0xE0014008)

The 32-bit PWM Timer Counter is incremented when the Prescale Counter reaches its terminal count. Unless it is reset before reaching its upper limit, the PWMTC will count up through the value 0xFFFFFFFF and then wrap back to the value 0x00000000. This event does not cause an interrupt, but a Match register can be used to detect an overflow if needed.

PWM Prescale Register (PWMPR - 0xE001400C)

The 32-bit PWM Prescale Register specifies the maximum value for the PWM Prescale Counter.

PWM Prescale Counter Register (PWMPC - 0xE0014010)

The 32-bit PWM Prescale Counter controls division of pclk by some constant value before it is applied to the PWM Timer Counter. This allows control of the relationship of the resolution of the timer versus the maximum time before the timer overflows. The PWM Prescale Counter is incremented on every pclk. When it reaches the value stored in the PWM Prescale Register, the PWM Timer Counter is incremented and the PWM Prescale Counter is reset on the next pclk. This causes the PWM TC to increment on every pclk when PWMPR = 0, every 2 pclks when PWMPR = 1, etc.

PWM Match Registers (PWMMR0 - PWMMR6)

ThePWM Match register values are continuously compared to the PWM Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the PWM Timer Counter, or stop the timer. Actions are controlled by the settings in the PWMMCR register.

PWM Match Control Register (PWMMCR - 0xE0014014)

The PWM Match Control Register is used to control what operations are performed when one of the PWM Match Registers matches the PWM Timer Counter. The function of each of the bits is shown in Table 169.

Table 169: PWM Match Control Register (PWMMCR - 0xE0014014)

PWMMCR	Function	Description	Reset Value
0	Interrupt on PWMMR0	When one, an interrupt is generated when PWMMR0 matches the value in the PWMTC. When zero this interrupt is disabled.	0
1	Reset on PWMMR0	When one, the PWMTC will be reset if PWMMR0 matches it. When zero this feature is disabled.	0
2	Stop on PWMMR0	When one, the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR0 matches the PWMTC. When zero this feature is disabled.	0
3	Interrupt on PWMMR1	When one, an interrupt is generated when PWMMR1 matches the value in the PWMTC. When zero this interrupt is disabled.	0
4	Reset on PWMMR1	When one, the PWMTC will be reset if PWMMR1 matches it. When zero this feature is disabled.	0
5	Stop on PWMMR1	When one, the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR1 matches the PWMTC. When zero this feature is disabled	0
6	Interrupt on PWMMR2	When one, an interrupt is generated when PWMMR2 matches the value in the PWMTC. When zero this interrupt is disabled.	0
7	Reset on PWMMR2	When one, the PWMTC will be reset if PWMMR2 matches it. When zero this feature is disabled.	0
8	Stop on PWMMR2	When one, the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR2 matches the PWMTC. When zero this feature is disabled	0
9	Interrupt on PWMMR3	When one, an interrupt is generated when PWMMR3 matches the value in the PWMTC. When zero this interrupt is disabled	0
10	Reset on PWMMR3	When one, the PWMTC will be reset if PWMMR3 matches it. When zero this feature is disabled.	0
11	Stop on PWMMR3	When one, the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR3 matches the PWMTC. When zero this feature is disabled	0
12	Interrupt on PWMMR4	When one, an interrupt is generated when PWMMR4 matches the value in the PWMTC. When zero this interrupt is disabled.	0
13	Reset on PWMMR4	When one, the PWMTC will be reset if PWMMR4 matches it. When zero this feature is disabled.	0
14	Stop on PWMMR4	When one, the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR4 matches the PWMTC. When zero this feature is disabled	0
15	Interrupt on PWMMR5	When one, an interrupt is generated when PWMMR5 matches the value in the PWMTC. When zero this interrupt is disabled.	0
16	Reset on PWMMR5	When one, the PWMTC will be reset if PWMMR5 matches it. When zero this feature is disabled.	0

Table 169: PWM Match Control Register (PWMMCR - 0xE0014014)

PWMMCR	Function	Description	Reset Value
17	Stop on PWMMR5	When one, the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR5 matches the PWMTC. When zero this feature is disabled	0
18	Interrupt on PWMMR6	When one, an interrupt is generated when PWMMR6 matches the value in the PWMTC. When zero this interrupt is disabled.	0
19	Reset on PWMMR6	When one, the PWMTC will be reset if PWMMR6 matches it. When zero this feature is disabled.	0
20	Stop on PWMMR6	When one, the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR6 matches the PWMTC. When zero this feature is disabled	0

PWM Control Register (PWMPCR - 0xE001404C)

The PWM Control Register is used to enable and select the type of each PWM channel. The function of each of the bits are shown in Table 170.

Table 170: PWM Control Register (PWMPCR - 0xE001404C)

PWMPCR	Function	Description	Reset Value
1:0	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2	PWMSEL2	When zero, selects single edge controlled mode for PWM2. When one, selects double edge controlled mode for the PWM2 output.	0
3	PWMSEL3	When zero, selects single edge controlled mode for PWM3. When one, selects double edge controlled mode for the PWM3 output.	0
4	PWMSEL4	When zero, selects single edge controlled mode for PWM4. When one, selects double edge controlled mode for the PWM4 output.	0
5	PWMSEL5	When zero, selects single edge controlled mode for PWM5. When one, selects double edge controlled mode for the PWM5 output.	0
6	PWMSEL6	When zero, selects single edge controlled mode for PWM6. When one, selects double edge controlled mode for the PWM6 output.	0
8:7	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
9	PWMENA1	When one, enables the PWM1 output. When zero, disables the PWM1 output.	0
10	PWMENA2	When one, enables the PWM2 output. When zero, disables the PWM2 output.	0
11	PWMENA3	When one, enables the PWM3 output. When zero, disables the PWM3 output.	0
12	PWMENA4	When one, enables the PWM4 output. When zero, disables the PWM4 output.	0
13	PWMENA5	When one, enables the PWM5 output. When zero, disables the PWM5 output.	0
14	PWMENA6	When one, enables the PWM6 output. When zero, disables the PWM6 output.	0
15	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

PWM Latch Enable Register (PWMLER - 0xE0014050)

ThePWM Latch Enable Register is used to control the update of the PWM Match registers when they are used for PWM generation. When software writes to the location of a PWM Match register while the Timer is in PWM mode, the value is held in a shadow register. When a PWM Match 0 event occurs (normally also resetting the timer in PWM mode), the contents of shadow registers will be transferred to the actual Match registers if the corresponding bit in the Latch Enable Register has been set. At that point, the new values will take effect and determine the course of the next PWM cycle. Once the transfer of new values has taken place, all bits of the LER are automatically cleared. Until the corresponding bit in the PWMLER is set and a PWM Match 0 event occurs, any value written to the PWM Match registers has no effect on PWM operation.

For example, if PWM2 is configured for double edge operation and is currently running, a typical sequence of events for changing the timing would be:

- Write a new value to the PWM Match1 register.
- Write a new value to the PWM Match2 register.
- Write to the PWMLER, setting bits 1 and 2 at the same time.
- The altered values will become effective at the next reset of the timer (when a PWM Match 0 event occurs).

The order of writing the two PWM Match registers is not important, since neither value will be used until after the write to PWMLER. This insures that both values go into effect at the same time, if that is required. A single value may be altered in the same way if needed.

The function of each of the bits in the PWMLER is shown in Table 171.

Table 171: PWM Latch Enable Register (PWMLER - 0xE0014050)

PWMLER	Function	Description	Reset Value
0	Enable PWM Match 0 Latch	Writing a one to this bit allows the last value written to the PWM Match 0 register to be become effective when the timer is next reset by a PWM Match event. See the description of the PWM Match Control Register (PWMMCR).	0
1	Enable PWM Match 1 Latch	Writing a one to this bit allows the last value written to the PWM Match 1 register to be become effective when the timer is next reset by a PWM Match event. See the description of the PWM Match Control Register (PWMMCR).	0
2	Enable PWM Match 2 Latch	Writing a one to this bit allows the last value written to the PWM Match 2 register to be become effective when the timer is next reset by a PWM Match event. See the description of the PWM Match Control Register (PWMMCR).	0
3	Enable PWM Match 3 Latch	Writing a one to this bit allows the last value written to the PWM Match 3 register to be become effective when the timer is next reset by a PWM Match event. See the description of the PWM Match Control Register (PWMMCR).	0
4	Enable PWM Match 4 Latch	Writing a one to this bit allows the last value written to the PWM Match 4 register to be become effective when the timer is next reset by a PWM Match event. See the description of the PWM Match Control Register (PWMMCR).	0
5	Enable PWM Match 5 Latch	Writing a one to this bit allows the last value written to the PWM Match 5 register to be become effective when the timer is next reset by a PWM Match event. See the description of the PWM Match Control Register (PWMMCR).	0
6	Enable PWM Match 6 Latch	Writing a one to this bit allows the last value written to the PWM Match 6 register to be become effective when the timer is next reset by a PWM Match event. See the description of the PWM Match Control Register (PWMMCR).	0
7	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA