กล้องดิจิตอลวงจรปิด

(Closed circuit digital camera)

โดย

นางสาว อนัญญา จรัสงามสว่าง 48011050

นาย เอกพนธ์ กึกก้อง 48011144

นาย เอกภัทร ปัญญาแก้ว 48011146

อาจารย์ที่ปรึกษา

อ. ชินภัทร นันทจิวากรชัย

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

กล้องดิจิตอลวงจรปิด

CLOSED CIRCUIT DIGITAL CAMERA

โดย

นางสาว อนัญญา	จรัสงามสว่าง	48011050
นาย เอกพนธ์	กึกก้อง	48011144
นาย เอกภัทร	ปัญญาแก้ว	48011146

อาจารย์ที่ปรึกษา

อ. ชินภัทร นันทจิวากรชัย

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

ปริญญานิพนธ์ ปีการศึกษา 2551

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง กล้องดิจิตอลวงจรปิด

ผู้จัดทำ นางสาว อนัญญา จรัสงามสว่าง 48011050

นาย เอกพนธ์ กึกก้อง 48011144

นาย เอกภัทร ปัญญาแก้ว 48011146

.....อาจารย์ที่ปรึกษา

(อ.ชินภัทร นันทจิวากรชัย)

วันที่ 20 / มี.ค / 52

คำนำ

เมื่อทรัพย์สินของเราได้เปลี่ยนมือไปเป็นของมิจฉาชีพ หนึ่งในทางออกทางหนึ่งสำหรับเหตุการณ์นี้ คือ การสร้างระบบรักษาความปลอดภัยโดยการติดกล้องวีดิโอวงจรปิด เพื่อจับการเคลื่อนไหวโดยในทางเลือกนี้อาจเป็นทางเลือกที่อาจจะใช้งบประมาณค่อนข้างสูง แต่สำหรับโครงงานนี้จะใช้กล้องถ่ายภาพเป็นหน้าที่หลัก และจะนำภาพที่ได้ไปเก็บลงในหน่วยความจำ ชนิด เอสดี แล้วสามารถนำไปเปิดดูบนเครื่องคอมพิวเตอร์ เพื่อความสะดวกต่อการนำมาข้อมูลมาตรวจสอบและเก็บสำรองภาพ และโครงงานจะสามารถนำไปประยุกต์ใช้กับส่วนอื่นๆ ต่อไปเช่น ระบบเซนเซอร์

อนัญญา จรัสงามสว่าง
เอกพนธ์ กึกก้อง
เอกภัทร ปัญญาแก้ว
ผู้จัดทำ

กล้องถ่ายภาพวงจรปิด

นางสาว อนัญญา จรัสงามสว่าง 48011050
นาย เอกพนธ์ กึกก้อง 48011144
นาย เอกภัทร ปัญญาแก้ว 48011146
อ. ชินภัทร นันทจิวากรชัย อาจารย์ที่ปรึกษา
ปีการศึกษา 2551

บทคัดย่อ

โครงงานนี้เป็นการสร้างกล้องถ่ายภาพวงจรปิด เพื่อใช้ในการรักษาความปลอดภัยต่างๆ โดยจะใช้ไมโครคอนโทรลเลอร์มาควบคุมการถ่ายภาพและกำหนดคุณสมบัติของภาพได้ โดยสามารถเลือกโหมดสำหรับถ่ายภาพและสามารถตั้งเวลาถ่ายภาพได้ สามารถเลือกจำนวนภาพที่จะถ่ายต่อครั้งหรือถ่ายตลอดโดยไม่หยุดได้ และสามารถบันทึกเวลาขณะที่ภาพถูกถ่ายได้ด้วย โดยภาพที่ถ่ายจะถูกนำไปเก็บลงบนหน่วยความจำเอสดีการ์ด ซึ่งสามารถนำไปเปิดดูภาพในคอมพิวเตอร์ได้ และยังมีวงจรไฟสำรองจากแบตเตอรี่เพื่อสำรองไฟขณะเกิดไฟฟ้าดับชั่วคราวหรือไฟตกได้

Closed circuit digital camera

Miss. Anunya Jaratngamsawang ID.48011050

Mr. Ekkapon Kuekong ID.48011144

Mr. Ekkapat Panyakeaw ID.48011146

Chinnapat Nuntajiwakornchai Advisor

Educational Year 2008

Abstract

This report presents about a construction of closed circuit digital camera using in safety and security. Microcontroller is used to be a part of this project to control to take a picture and can set up qualifications and can choose mode for capture such as timing , and numbers of pictures being captured , and it is automatic and then can record the picture in SD card to show on the computer.

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

หน้า

สารบัญรูป

หน้า

บทที่1
บทนำ

1.1 ที่มาของโครงงาน

ในปัจจุบันมีมิจฉาชีพจำนวนมาก และบ่อยครั้งถ้าทรัพย์สินของเราเปลี่ยนมือไปเป็นของเหล่ามิจฉาชีพ ไม่ว่าของชิ้นนั้นจะมีของมาน้อยเพียงใด บางครั้งการที่เจ้าของบ้านจะครอบครองนั้นอาจใช้เวลาและใช้เงินมากมายอยู่พอสมควร ยกตัวอย่างจากทุกวันนี้ เมื่อสมาชิกในบ้านออกไปทำธุระกันหมดทำให้เปิดโอกาสให้พวกมิจฉาชีพเข้ามาขโมยทรัพย์สินที่อยู่ภายในบ้าน หนึ่งในทางออกคือการสร้างระบบรักษาความปลอดภัยโดยการติดกล้องวีดีโอเพื่อจับภาพเคลื่อนไหว แต่ทว่าการกระทำเช่นนี้อาจใช้งบประมานมาก อีกทั้งถ้าช่วงที่มิจฉาชีพกำลังขโมยนั้นตลับเทปอาจหมดม้วนพอดี หากเช่นนั้นคงไม่มีหลักฐานมัดตัวมิจฉาชีพ ดังนั้น เราจึงสร้างโครงงานนี้ ขึ้นมาเพื่อเป็นกล้องถ่ายภาพแล้วเก็บไว้ในหน่วยความจำเอสดีซึ่งจะถ่ายรูปแล้ว ถ้าเกิดข้อมูลเต็มการ์ดแล้ว จะภาพที่ถ่ายใหม่จะวนกลับมาเก็บที่ไฟล์อันแรกใหม่ทับไปเรื่อยๆ ทำให้เมื่อมิจฉาชีพเข้ามาเราจะมีโอกาสในการเก็บรูปมากว่าการใช้กล้องวีดีโอ

1.2 วัตถุประสงค์ของโครงงาน

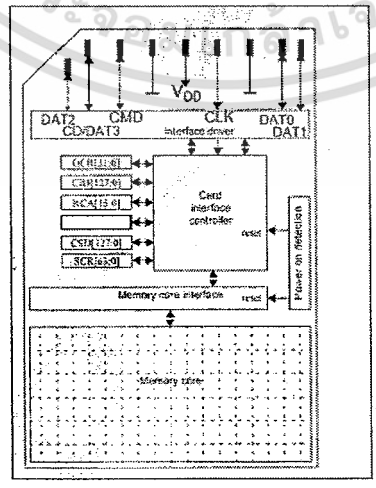
1. สร้างกล้องถ่ายที่บันทึกลงบนหน่วยความจำเอสดีที่มีขนาดเล็กงบประมาณค่อนข้างต่ำ สามารถนำไปใช้เป็นกล้องวงจรปิดได้และนำไปประยุกต์ใช้ในงานอื่นๆ
2. ทำให้ได้ความรู้ในการเขียนโปรแกรมติดต่อกับกล้องและหน่วยความจำเอสดี

บทที่ 2

คุณลักษณะของหน่วยความจำเอสดีการ์ด

หน่วยความจำเอสดีการ์ดเป็นหน่วยความจำที่ได้รับความนิยมเป็นอย่างมากสามรถพกพาได้สะดวกโดยย่อมาจากคำว่า Secure Digital ซึ่งหน่วยความจำเสดีการ์ดนี้จะมีฟังก์ชั่นการทำงานคล้ายเอ็มเอ็มซีการ์ด ซึ่งภายในเอสดีการ์ดนี้ จะประกอบด้วยไมโครคอนโทลเลอร์ แฟลชเมมโมรี่คอนโทรล ซึ่งการโอนย้ายข้อมูลโดยปกติระหว่างเมมโมรี่กับโฮสคอนโทลนี้จะโนย้าย 512 ไบต์ต่อบล็อก ดังนั้นเอสดีการ์ดจึงเหมือนกับฮาร์ดดิสก์ทั่วไปนั่นเอง โดยทั่วไปไฟล์ระบบนั้นจะเป็นแบบแฟต 16/32 เท่านั้นสำหรับการแบ่งส่วน ซึ่งถ้าจะใช้แฟต 32 นั้น ความจุต้องมีค่ามากกว่า 2 จิกกะไบต์

รูปที่ 2.1 ลักษณะของหน่วยความจำเอสดีที่จำหน่ายตามท้องตลาดทั่วไป



รูปที่ 2.2 ลักษณะคอนแทคของหน่วยความจำเอสดี

การติดต่อระหว่างโฮสกับเอสดีการ์ดนั้นมีอยู่ 2 โหมดด้วยกันคือ SD MODE และ SPI MODE ซึ่งในการติดต่อนี้จะติดต่อแบบเอสพีไอโหมด

2.1 SPI MODE

เป็นทางเลือกทางหนึ่งในการติดต่อกับโฮสภายนอก ซึ่งหน่วยความจำจะสามารถติดต่อโดยช่องทาง SPI Port หรือ GPIO Port ที่มีในไมโครคอนโทลเลอร์ทั่วไปเพราฉะนั้น SPI MODE จะเหมาะสมกับการประยุกต์ใช้งานในโครงงานนี้โดยจะทำการเลือกโหมดการทำงานเป็น โหมด 0

2.1.1 Command and response

ใน SPI MODE ทิศทางข้อมูลจะอยู่จะอยู่บน Signal Line ซึ่งการโอนย้ายอนุกรมข้อมูลจะกำหนดเป็นไบต์ ส่วนเฟรมของคอมมานด์เป็นการส่งผ่านจากโฮสไปยังการ์ดซึ่งมีความยาวของขนาดเท่ากับ 6 ไบต์ ต่อ 1 คำสั่งคอมมานด์ เมื่อเฟรมของคอมมานด์เป็นการส่งผ่านไปยังการ์ดแล้วค่า Response (R1 ,R2,R3) ของคอมมานด์นั้นจะถูกส่งกลับมายังโฮส ซึ่งโฮสนั้นจะต้องอ่านค่าตลอดจนกว่าจะส่งมาอย่างถูก ต้องของ Response ที่คอมมานด์นั้นเพราะการส่งผ่านข้อมูลของการ์ดมายังโฮสนั้นจะถูกจะขับจากสัญญาณนาฬิกาที่โฮสกำหนดมานั้นเอง Command response time นั้นจะมีโครงสร้างจาก 0 ถึง 8 ไบต์ โดยสัญญาณ CS นี้จะต้องควบคุมให้เป็นไบต์ 0 ในระหว่างการติดต่อ Command ,Response , Data transfer ยังคงกระทำอยู่ ส่วนไบต์ CRC จะเป็นส่วนประกอบของชุดเฟรมของ Command ซึ่งเป็นทางเลือกใน SPI MODE สำหรับตรวจสอบความถูกต้องของข้อมูลที่ส่งออกไป

2.1.2 SPI Command Set

แต่ละCommand เป็นการแสดงเครื่องหมายย่อ GO_IDLE_STATE หรือ CMD<n>, โดยที่<n> นั้นเป็นตัวเลขของคอมมานด์อินเด็กซ์ และค่าของn เป็นได้ตั้งแต่ 0-63 ซึ่งรูปแบบของตาราง Command จะใช้อธิบาย Command ที่ใช้ทั่วๆ ไปในการอ่านเขียน และการกำหนดค่าเริ่มต้นของการ์ด

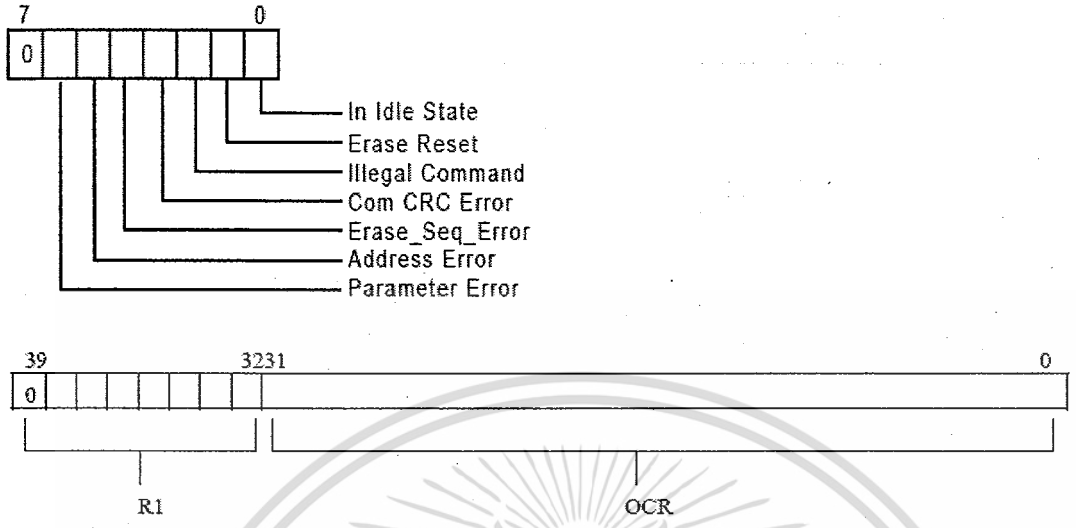
2.2 ขั้นตอนการเซตค่าเริ่มต้นสำหรับ SPI Mode

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
CMD26	No				
CMD27	Yes	None	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28[1]	Yes	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card specific data (WP_GRP_SIZE).
CMD29[4]	Yes	[31:0] data address	R1b	CLR_WRITE_PROT	If the card has write protection features, this command clears the write protection bit of the addressed group.
CMD30	Yes	[31:0] write protect data address	R1	SEND_WRITE_ PROT	If the card has write protection features, this command asks the card to send the status of the write protection bits.[2]
CMD31	Reserved				
CMD32	Yes	[31:0] data address	R1	ERASE_WR_BLK_START _ADDR	Sets the address of the first write block to be erased.
CMD33	Yes	[31:0] data address	R1	ERASE_WR_BLK_END _ADDR	Sets the address of the last write block in a continuous range to be erased.
CMD34 .... CMD37	Reserved				
CMD38	Yes	[31:0] don't care*	R1b	ERASE	Erases all previously selected write blocks.
CMD39	No				
CMD40	No				
CMD41 ... CMD54	Reserved				
CMD55	Yes	[31:0] stuff bits	R1	APP_CMD	Notifies the card that the next command is an application specific command rather than a standard command.
CMD56	Yes	[31:0] stuff bits [0]: RD/WR[3]	R1	GEN_CMD	Used either to transfer a Data Block to the card or to get a Data Block from the card for general purpose/application specific commands. The size of the Data Block is defined with SET_BLOCK_LEN command.
CMD57	Reserved				
CMD58	Yes	None	R3	READ_OCR	Reads the OCR register of a card.
CMD59	Yes	[31:1] don't care* [0:0] CRC option	R1	CRC_ON_OFF	Turns the CRC option on or off. A '1' in the CRC option bit will turn the option on, a '0' will turn it off.
CMD60-63	No				

ตารางที่ 2.1 Commandที่ใช้ทั่วไปในการอ่านเขียนและกำหนดค่าเริ่มต้นของการ์ด

2.1.3 SPI Response

รูปแบบของ Response นั้นจะมีอยู่ 3 รูปแบบ คือ R1 ,R2 , R3 ซึ่งจะขึ้นอยู่แต่ละ command ที่ใช้นั้นๆ โดยไบต์ของ Response R1 จะเป็นค่ากลับมาของ command ส่วนใหญ่



รูปที่ 2.3 ไบต์ของ R1 Response และ R3 Response

ซึ่งแต่ละบิตสำหรับ R1 Response ที่แสดงนั้นถ้าค่าของไบต์ R1 = 0x00 จะหมายความว่าติดต่อกับหน่วยความจำเอสดีการ์ดสำเร็จแล้ว เมื่อไหร่ก็ตามที่เกิดข้อผิดพลาดขึ้น R1 นั้น จะเซตขึ้นดังที่สอดคล้องกับรูป ค่าของ R3 Response นั้น จะสำหรับ CMD58 เท่านั้น มันจะมีไบต์แรกเหมือนกับ R1 และจะต่อท้ายเป็นขบวนที่บรรจุของ OCR

บางคอมมานด์จะใช้เวลานานมากกว่าจำนวนของ NCR ที่กำหนดดังนั้นโฮสต์คอนโทรลเลอร์นั้นจะรอค่า Response กลับมาด้วยโดยการส่งค่า 0xff ตลอดจนกว่าจะรับ Response กลับมาได้แล้ว

2.2 ขั้นตอนการเซตค่าเริ่มต้นสำหรับ SPI Mode

หลังจาก Power on reset หน่วยความจำเอสดีนั้นจะเข้าสู่โหมดทำงาน active mode ซึ่งมีรูปแบบการทำงานดังนี้

2.2.1 Power on

หลังจากจ่ายมาถึงค่าแรงดัน 2.2 โวลต์ เราจะต้องเซตค่าของ DI และ CS ให้เป็น 1 ให้มากว่า 74 พัลส์ ของสัญญาณนาฬิกา ซึ่งจะทำให้การ์ดนั้นสามารถรับคอมมานด์ต่างๆ ได้

2.2.2 Software Reset

ส่ง CMD0 ร่วมกับ CS เป็น 0 เพื่อที่จะไปรีเซตการ์ด การ์ดนั้นจะไปตรวจสอบสัญญาณ CS เมื่อ CMD ตรวจพบ ถ้า CS เป็น 0 นั้นการ์ด จะเข้าสู่ SPI Mode ดังนั้น CMD0 จะต้องส่ง native command

ด้วย และในส่วนของ CRC นั้นจะต้องมีค่า CRC ที่ถูกต้องด้วย ทันทีที่การ์ดเข้าสู่ SPI Mode การตรวจสอบค่า CRC จะไม่พิจารณาอีก ดังนั้นการส่ง CM0 จะกำหนดค่าของ CRC ให้เท่ากับ0x95 ได้เลยเมื่อ CMD นั้นถูกยอมรับเสร็จการ์ดนั้นจะเข้าสู่สถานะว่าง (idle state) และผลของค่า response R1 จะมีค่าเท่ากับ0x01 นั่นคือบิตของ in idle state จะถูกเซตขึ้นซึ่งแสดงว่าการ์ดเอสดีนั้นอยู่ในสถานะว่าง

ในสถานะว่างเอสดีการ์ดนั้นจะยอมรับเฉพาะ CMD0 , CMD1 , CMD58 เท่านั้น คอมมานด์อื่นๆ จะ ถูกปฏิเสธ เมื่อการ์ดนั้นตรวจพบ CMD1 การ์ดนั้นจะเริ่มพร้อมจะทำงาน ซึ่งโฮสนั้นจะต้องส่ง CMD1 ซ้ำเรื่อยๆ และต้องคอยตรวจสอบ response พวกนี้จนกว่าค่าของ response R1 จะตอบกลับมาเป็น 0x00 ดังนั้น CMD1 นั้นคือ เป็นการเคลียร์ค่าของ R1 นั่นเองเพื่อจะทำการส่งค่า Command ต่อไปในขบวนการ initial การ์ดจะใช้เวลาหลายมิลลิวินาทีขึ้นอยู่กับขนาดของการ์ดนั้นๆ ดังนั้นเราจำพิจารณาค่านี้ได้จาก Data Sheet ที่ผู้ผลิตให้มา ซึ่งหลังจากการทำ initial เสร็จก็จะสามารถอ่านและเขียนข้อมูลทั่วไปได้แล้ว ในช่วงเวลานี้จะสามารถอ่านค่ารีจิสเตอร์ OCR และ CID ได้เพื่อที่จะหาย่านของแรงดันที่กระทำและขนาดของการ์ดหรือคุณสมบัติอื่นๆ ที่เราต้องการ

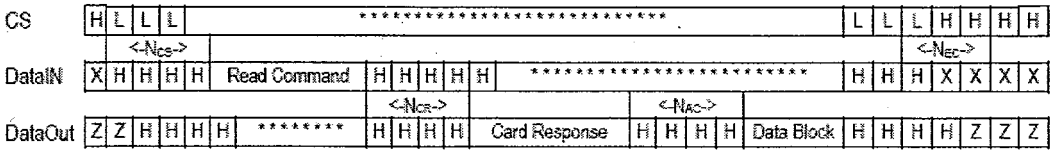
2.3 การโอนย้ายข้อมูล

2.3.1 Data packet and Data Response

ในการติดต่อเกี่ยวกับการโอนย้ายข้อมูล 1 block หรือมากกว่า1 block นั้นจะต้องทำการส่งหรือรับจาก command response ซึ่งการโอนย้ายข้อมูลนั้นจะประกอบไปด้วยกลุ่มของข้อมูลคือ Data Token 1 ไบต์และ CRC อีก 2 ไบต์

2.3.2 การอ่านแบบ single block

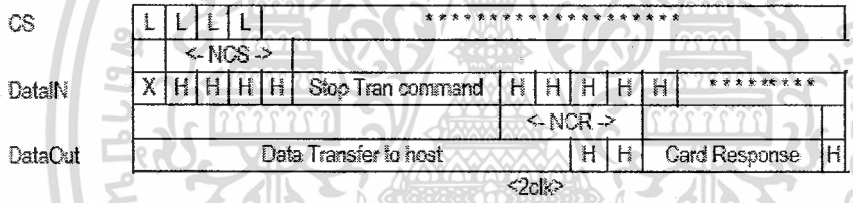
Argument นั้นจะเป็นการระบุตำแหน่งที่จะเริ่มอ่านในหน่วยไบต์เมื่อ CMD17 ถูกยอมรับแล้ว การกระทำของข้อมูล data block จะส่งไป ยังโฮสหลังจาก Data token ถูกตรวจพบสิ่งที่โฮสจะได้รับตามมา คือ data block และ CRC 2 ไบต์ โดยปกติขนาดของบล็อกจะเท่ากับ 512 ไบต์ซึ่งเราสามารถเปลี่ยนแปลงได้โดย CMD16 ถ้าเกิดข้อผิดพลาดในระหว่างการอ่านข้อมูล Error token จะถูกส่งมาแทน



รูปที่ 2.4 แสดงการอ่านแบบ Single block

2.3.3 การอ่านแบบ Multiple Block

การอ่านข้อมูลแบบ multiple block เป็นลำดับจากการระบุที่อยู่ เมื่อจำนวนของการโอนย้าย Block จะไม่เจาะจงก่อนคอมมานด์นี้ การติดต่อจะเริ่มต้นที่การเปิดและจบที่การอ่าน Multi block การอ่านนั้นจะต่อเนื่องจนกระทั่งหยุดโดย CMD12 ไบต์ที่ได้รับ โดยทันทีทันใดนั้นจะตามหลัง CMD12 มันจะถูกตัดทิ้งก่อนได้รับ response ของ CMD12



รูปที่ 2.5 แสดงการอ่านแบบ Multiple Block

2.3.4 การอ่านค่ารีจิสเตอร์ CSD และ CID

การอ่านค่ารีจิสเตอร์ทั้ง 2 ตัวนี้จะเหมือนกับการอ่านแบบ single block ยกเว้นสำหรับขนาดความยาว ของ block CID และ CSD จะถูกส่งไปยังโฮสเท่ากับ 16 ไบต์ของ data block

บทที่3

FAT (File Allocation Table)

3.1 ความหมายของFAT

FAT ย่อมาจากคำว่า File Allocation Table ซึ่งคือ ตารางที่ใช้เก็บตำแหน่งข้อมูลต่างๆ ที่อยู่บนฮาร์ดดิสก์ เพื่อใช้ในการจัดสรรและติดตามการใช้เนื้อที่ในฮาร์ดดิสก์

ประเภทของFAT

FAT เป็นระบบจัดการไฟล์ที่ถูกกำหนดโดยซอฟท์แวร์ระบบปฏิบัติการ (operating system) จึงแตกต่างกันไปได้หลายแบบ FAT ที่เป็นที่รู้จัก ได้แก่

1. FAT(File Allocation Table)
   เป็นระบบไฟล์มาตรฐานสำหรับ DOS และ Windows และด้วยการที่ FAT เป็นที่นิยมใช้อย่างกว้างขวาง จึงสามารถใช้ร่วมกับระบบปฏิบัติการอื่น ได้ด้วย
2. FAT32(32-bit File Allocation Table)
   ระบบนี้จะอยู่ใน window 95 OSR2 ในรุ่นที่มีการติดตั้งจากผู้ผลิต และ window 98 โดย FAT32 กำจัดข้อจำกัดของ FAT หลายประการออกไป แต่ไม่สามรถใช้ใน Window 95 และ Window 98 ได้
3. VFAT(Virtual File Allocation Table)
   เป็นระบบไฟล์ FAT เวอร์ชั่นที่มีลักษณะเป็น Protected Mode ซึ่งถูกใช้โดย Window 9x ระบบไฟล์นี้จะคล้ายๆ กับ FAT ต่างกันตรงที่จะสามรถรับชื่อไฟล์ยาวๆ ได้
4. NTFS(NT File System)
   เป็นระบบไฟล์ที่ถูกออกแบบมาเพื่อใช้กับ Window NT โดยเฉพาะแม้ว่าสามรถติดตั้ง Window NT ในระบบไฟล์ FAT32 ได้ แต่ว่า NTFS จะให้ประสิทธิภาพที่ดีกว่าในด้านระบบรักษาความปลอดภัยในการเข้าถึงไฟล์มากกว่า และเสียพื้นที่น้อยกว่า
5. HPFS(High Performance File System)
   เป็นระบบไฟล์ที่ออกแบบมาเพื่อใช้กับ OS/2 ซึ่ง HPFS เหมือนกับ NTFS ที่จะมีระบบรักษาความปลอดภัยที่ดีมีความเชื่อถือได้ของข้อมูล มีความเร็วสูงกว่า FAT

ในการจัดสรรเนื้อที่บันทึกข้อมูลในฮาร์ดดิสก์ เพื่อให้การอ่าน - เขียนกระทำได้อย่างรวดเร็ว จะต้องใช้วิธีกำหนดเป็นไซลินเดอร์ (Cylinder) หรือ เพื่อให้แทรกตรงกันทุกหน้าของฮาร์ดดิสก์เป็นการ

แบ่งกลุ่มของไซลินเดอร์ที่อยู่ต่อเนื่องกัน ทำให้มีลักษณะเป็นโลจิกคอลไดรฟ์ และในแต่ละพาร์ทิชั่นจะมีการจัดทำตาราง FAT สำหรับพาร์ทิชั่นนั้น

FAT กำหนดเนื้อที่สำหรับไฟล์โดยใช้หน่วย คลัสเตอร์(cluster) คือเป็นกลุ่มของเซ็กเตอร์ซึ่งอาจประกอบด้วย 4 ถึง 64 เซกเตอร์ หรือขนาดตั้งแต่ 2 K ถึง 32 K

ระบบปฏิบัติการดอส และวินโดว์รุ่นแรกๆ ใช้FAT16 อ้างอิงหรือชี้ตำแหน่งคลัสเตอร์ได้สูงสุด 65,536 คลัสเตอร์ ในระบบ FAT16 สามารถมีตำแหน่งของคลัสเตอร์ได้สูงสุดที่ 2 GB

ดังนั้นถ้าผู้ใช้ฮาร์ดดิสก์ขนาด 2 GB หรือฮาร์ดดิสก์ที่มีพาร์ทิชั่นเท่ากับ 2 GB หมายความว่าขนาดของคลัสเตอร์ที่เล็กที่สุดมีขนาดเท่ากับ 32 KB ตัวอย่างเช่น ถ้าต้องการเก็บไฟล์ขนาด 1kb ลงในฮาร์ดดิสก์ที่เป็น FAT16 ฮาร์ดดิสก์จะต้องจองพื้นที่เพื่อเก็บไฟล์นี้เท่ากับ 32 KB ซึ่งหมายความว่าผู้ใช้จะต้องเสียพื้นที่ที่ใช้เก็บไปเท่ากับ 31 KB

ดังนั้นยิ่งมีไฟล์ที่มีขนาดเล็กกว่า 32 KB มากเท่าไหร่ยิ่งสูญเสียเนื้อที่ในฮาร์ดดิสก์โดยไม่ได้ใช้ประโยชน์มากขึ้นเท่านั้น

ในระบบ FAT32 จำนวนคลัสเตอร์จะเท่ากับ 268,436,456 คลัสเตอร์ดังนั้นเมื่อใช้ขนาดของคลัสเตอร์ 4 KB ขนาดของพาร์ทิชั่นที่สูงสุดจะมีขนาดท่ากับ 8 GB และขนาดของคลัสเตอร์สูงสุดที่ 32 KB จะทำให้ฮาร์ดดิสก์มีพาร์ทิชั่นได้สูงสุดเท่ากับ 2TB (1 Teta Byte = 1,024 GB)

FAT เป็นระบบไฟล์ชนิดหนึ่งที่ถูกกำหนดโดยซอฟท์แวร์ระบบปฏิบัติการ (Operating System) ซึ่งระบบไฟล์ที่นิยมใช้ในปัจจุบันมี

- FAT ของระบบดอสและ วินโดว์
- NTFS ของระบบปฏิบัติการวินโดว์เอ็นที
- HPFS ของระบบปฏิบัติการโอเอสทู

FAT ที่นิยมใช้กันอยู่ใน ของระบบดอส และวินโดว์ คือ FAT16 โดย FAT จะทำหน้าที่ จัดการข้อมูลหลายๆ เซ็กเตอร์โดยแต่ละเซ็กเตอร์แบ่งเป็นหลายๆ คลัสเตอร์ซึ่งในระบบ FAT16 นั้นสามารถอ้างอิงหรือชี้ตำแหน่งคลัสเตอร์ได้สูงสุด 65,536 คลัสเตอร์

แต่ในระบบของ FAT16สามรถมีขนาดของคลัสเตอร์ได้ถึง 32 KB ดังนั้นในระบบ FAT16 จึงสามรถอ้างอิงข้อมูลในหนึ่งพาร์ทิชั่นได้สูงสุดถึง 2 GB

ดังนั้นถ้าผู้ใช้ฮาร์ดดิสก์ขนาด 2 GB หรือฮาร์ดดิสก์ที่มีพาร์ทิชั่นเท่ากับ 2 GB หมายความว่าขนาดของคลัสเตอร์ที่เล็กที่สุดมีขนาดเท่ากับ 32 KB ตัวอย่างเช่น ถ้าต้องการเก็บไฟล์ขนาด 1 kb ลงในฮาร์ดดิสก์ที่เป็น FAT16 ฮาร์ดดิสก์จะต้องจองพื้นที่เพื่อเก็บไฟล์นี้เท่ากับ 32 KB ซึ่งหมายความว่าผู้ใช้จะต้องเสียพื้นที่ที่ใช้เก็บไปเท่ากับ 31 KB

ดังนั้นยิ่งมีไฟล์ที่มีขนาดเล็กกว่า 32 KB มากเท่าไหร่ยิ่งสูญเสียเนื้อที่ในฮาร์ดดิสก์โดยไม่ได้ใช้ประโยชน์มากขึ้นเท่านั้น วิธีแก้ไขปัญหาการสูญเสียเนื้อที่นี้อาจทำได้ 2 กรณี

1.ถ้าผู้ใช้ยังต้องการใช้ฮาร์ดดิสก์ที่เป็น FAT16 อยู่ ผู้ใช้ต้องแบ่งพาร์ทิชั่นฮาร์ดดิสก์ให้มีขนาดเล็กลงเพื่อให้ฮาร์ดดิสก์มีขนาดคลัสเตอร์เล็กลงจะทำให้พื้นที่ว่างไม่สามรถใช้งานเหลือน้อยลง ดังตารางข้างล่างนี้ ถ้าผู้ใช้พาร์ทิชั่นฮาร์ดดิสก์ไว้ที่ขนาด 512 MB ต่อ พาร์ทิชั่น ขนาดของคลัสเตอร์จะลดลงเหลือเพียงแค่ 8 KB ซึ่งจะทำให้สูญเสียพื้นที่บนฮาร์ดดิสก์แบบเปล่าประโยชน์ลดลงถึง 3 เท่าเมื่อเปรียบเทียบการแบ่งพาร์ทิชั่นไว้ที่ขนาด 2 GB แต่วิธีการแบ่งฮาร์ดดิสก์ออกเป็นหลายๆ พาร์ทิชั่นอาจทำให้เกิดความยุ่งยากเช่น มีไดร์ฟอาจทำให้สับสนในเวลาในการใช้งานฮาร์ดดิสก์ออกเป็นหลายๆ พาร์ทิชั่นนี้อาจทำให้เกิดความยุ่งยากเช่นเช่น มีไดร์ฟอาจทำให้สับสนเวลาใช้งาน

2. ให้ใช้ฮาร์ดดิสก์ที่เป็น FAT32 ซึ่งเป็นระบบFAT แบบใหม่ 32 บิต ซึ่งมีในระบบปฏิบัติการวินโดว์ 95 OSR2 หรือ วินโดว์ 98 หรือใช้โปรแกรมที่ช่วยให้ FAT16 เป็น FAT32 สำหรับ FAT32 จำนานคลัสเตอร์ที่กำลังอ้างอิงถึงนี้เท่ากับ 2 ยกกำลัง 28 หรือเท่ากับ 268,436,456 คลัสเตอร์ ดังนั้นเมื่อใช้งานคลัสเตอร์ 4 KB ขนาดของพาร์ทิชั่นสูงสุดที่จะมีจะเท่ากับ 8 GB ล้าคลัสเตอร์สูงสุดที่ 32 KB จะทำให้ฮาร์ดดิสก์มีค่าสูงสุด 2 TB

ขนาดพาร์ทิชั่น	ขนาดของคลัสเตอร์	ขนาดของคลัสเตอร์ FAT 16
น้อยกว่า 260 MB	512 Byte	4 KB
260 – 510 MB	4 KB	8 KB
512-1023 MB	4 KB	8 KB
1024-2048 MB	4 KB	16 KB
2-8 GB	4 KB	32 KB
8-16 GB	8 KB	32 KB
16-32 GB	16 KB	32 KB
มากกว่า32 GB	32 KB	32 KB

ตารางที่ 3.1 แสดงการเปรียบเทียบขนาดระหว่างคลัสเตอร์ FAT32 และ FAT16

ในเครื่องคอมพิวเตอร์ที่ใช้ระบบปฏิบัติการวินโดว์ 95 นั้นถ้าผู้ใช้สำรวจดูในโฟลเดอร์ต่างๆ จะพบว่าจะมีไฟล์ขนาดเล็กๆ เป็นจำนวนหลายร้อยไฟล์ ตัวอย่างไฟล์ที่มีนามสกุล .dll ซึ่งไฟล์เหล่านั้นเป็นต้นต่อที่ทำให้ก่อเกิดการสูญเสียของฮาร์ดดิสก์อย่างไร้ประโยชน์โดยไม่ว่า ฮาร์ดดิสก์จะมีคลัสเตอร์เป็นแบบ FAT32 หรือ FAT16 ก็ตามจะแสดงถึงไฟล์ต่างๆ ในโฟล์เดอร์ของวินโดว์ 95

โดยทั่วไปเมื่อมี คลัสเตอร์เป็น 4 KB เนื้อที่ในฮาร์ดดิสก์ 83.5 เปอร์เซนต์เป็นเนื้อที่ที่เสียไปโดยไม่มีประโยชน์ และเมื่อใช้ฮาร์ดดิสก์ที่มีคลัสเตอร์เป็น 32 KB เนื้อที่ในฮาร์ดดิสก์ 97.9 เปอร์เซ็นต์เป็นเนื้อที่ที่เสียไปโดยเปล่าประโยชน์

3.2 ข้อเสียของ FAT32

1. ไม่สนับสนุนการทำงานร่วมกับปฏิบัติการอื่น รวมถึงวินโดว์ 95 เวอร์ชั่นเดิม และวินโดว์เอ็นที 4.0
2. ในการใช้โปรแกรม Utility ที่จัดการกับฮาร์ดดิสก์ ผู้ใช้ต้องตรวจสอบโปรแกรมนั้นว่าสนับสนุนระบบ FAT32 หรือไม่ ถ้าไม่สนับสนุนหรือไม่ได้ตรวจสอบแล้วผู้ใช้โปรแกรมนั้นกับฮาร์ดดิสก์ที่เป็น FAT32 จะทำให้เกิดความเสียหายกับข้อมูลในฮาร์ดดิสก์โดยไม่สามารถกู้ข้อมูลกลับมาได้

บทที่4

กล้อง C328

โมดูลกล้อง C328 เป็นบอร์ดรวมกล้องแบบอนุกรมซึ่งสามารถนำมาเชื่อมต่อกับwireless หรือพีดีเอโฮส โดยทำหน้าที่เปรียบเสมือนกล้องวีดิโอหรือ กล้องบีบอัดไฟล์ JPEG โดยที่จะมีอินเตอร์เฟสอนุกรมRS-232 และ เครื่องบีบไฟล์ JPEG และโมดูลกล้องใช้พลังงานต่ำ

4.1 คุณสมบัติโดยทั่วไป

1. มีขนาดเล็กและโมดูลกล้องใช้พลังงานต่ำ (3.3 โวลต์)
2. สำหรับระบบป้องกันบัสแบบอนุกรมประสิทธิภาพการแก้ปัญหาสูงหรือเป็นอุปกรณ์เสริมพีดีเอ
3. มี EEPROM ในตัวเพื่อสร้างอินเตอร์เฟสแบบคำสั่งพื้นฐานไปยังตัวโฮสภายนอกโดยใช้ RS-232
4. มีเซนเซอร์สีแบบ OmniVision OV7640/8 VGA ในตัว
5. มีตัว JPEG CODEC สำหรับแก้ปัญหาในตัว
6. มีวงจรแสดงตัวอย่าง วงจรควบคุม ภายในโมดูล สำหรับการใช้งานกับอิมเมจ VGA QVGA ที่ความละเอียด160×120 หรือ 80×60
7. มีวงจรปรับเปลี่ยนสีภายในตัวเพื่อใช้สำหรับ 2 บิต สีเทา, 4 บิต สีเทา, 8 บิต สีเทา, 16 บิต RGB หรือแสดงตัวอย่างของไฟล์ JPEG
8. ไม่จำเป็นต้องใช้ DRAM ภายนอก

4.2 คุณสมบัติของระบบ

1. เซนเซอร์กล้อง C328-7640 ใช้ Camera Chipsกล้องถ่ายภาพสีชนิด OmniVision OV7640/8 VGA กับอินเตอร์เฟสแบบ 8 บิต YbCr

2. OV528 บริดจ์แบบอนุกรม เป็นชิพคอนโทรลเลอร์แบบ JPEG CODEC ที่สามารถบีบอัดและส่งข้อมูลอิมเมจ จาก Camera chip ไปยังอุปกรณ์ภายนอกได้ และยังสามารถดึงเอา 8 บิต Ycbr 422 มาจากOV7640/8 ได้ อินเตอร์เฟสของกล้องจะเชื่อมต่อกับข้อมูลขาเข้าของวีดิโอ และแสดงถึงฟังก์ชั่นการแสดงตัวอย่างการควบคุมและหน้าต่าง รวมไปถึงการเปลี่ยนแปลงของสี โดยผ่านทาง

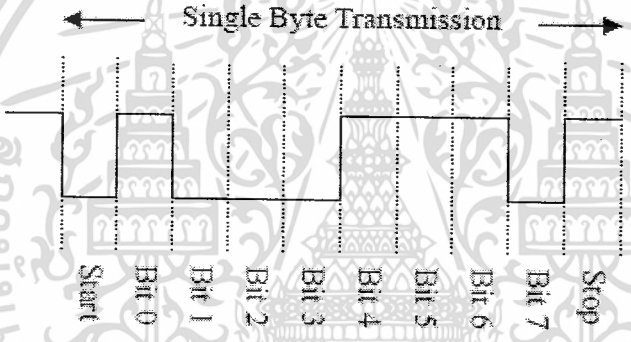
คำสั่งจากโฮสบัสแบบอนุกรม JPEG CODEC ยังสามารถที่จะบีบอัดภาพได้ดีและภาพที่ภาพบีบอัดมีคุณภาพสูง

3. โปรแกรม EEPROM ได้ถูกสร้างขึ้นเพื่อใช้สำหรับ C328 -7640 เพื่อเตรียมชุดคำสั่งไปยังโฮสภายนอก

4.3 อินเตอร์เฟสแบบอนุกรม

4.3.1 Timing diagram ของ 1 ไบต์

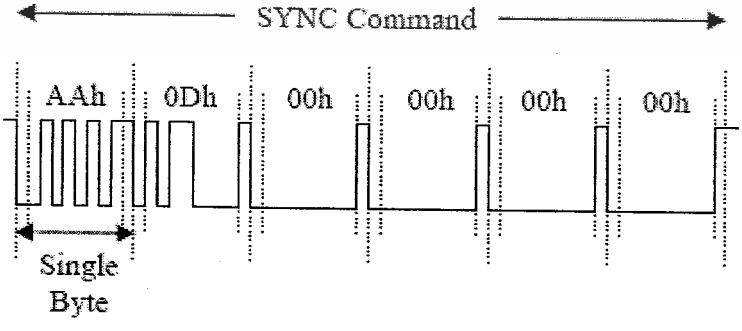
ตัวส่งRS-232 แบบ 1 ไบต์ประกอบไปด้วยบิตเริ่มต้น 8บิต ทั่วไปและบิตหยุดบิตเริ่มต้นจะมีสภาวะเป็น 0 ส่วนบิตหยุดจะมีสภาวะเป็น 1 และบิตต่ำจะเซตค่าที่สภาวะเป็น 1



รูปที่ 4.1 RS-232 Timing diagram ของ 1 ไบต์

4.3.2 Timing diagram ของคำสั่ง

การส่งคำสั่งผ่าน RS -232 1 คำสั่งประกอบด้วย 6 ไบต์ ตัวอย่างเช่น คำสั่ง SYNC



รูปที่ 4.2 Timing diagram ของคำสั่ง SYNC ผ่านRS – 232

4.4 ชุดคำสั่งของโมดูล C328 -7640

โมดูลสามารถรองรับได้ทั้งหมด 11 คำสั่งสำหรับการเชื่อมต่อไปยังโฮสตามตารางดังต่อไปนี้

Command	ID Number	Parameter1	Parameter2	Parameter3	Parameter4
Initial	AA01h	00h	Color Type	RAW Resolution (Still image only)	JPEG Resolution
Get Picture	AA04h	Picture Type	00h	00h	00h
Snapshot	AA05h	Snapshot Type	Skip Frame Low Byte	Skip Frame High Byte	00h
Set Package Size	AA06h	08h	Package Size Low Byte	Package Size High Byte	00h
Set Baudrate	AA07h	1st Divider	2nd Divider	00h	00h
Reset	AA08h	Reset Type	00h	00h	xxh*
Power Off	AA09h	00h	00h	00h	00h
Data	AA0Ah	Data Type	Length Byte 0	Length Byte 1	Length Byte 2
SYNC	AA0Dh	00h	00h	00h	00h
ACK	AA0Eh	Command ID	ACK counter	00h / Package ID Byte 0	00h / Package ID Byte 1
NAK	AA0Fh	00h	NAK counter	Error Number	00h
Light Frequency	AA13h	Frequency Type	00h	00h	00h

ตารางที่4.1 ตารางชุดคำสั่งทั่วไปของโมดูล C328 -7640

4.4.1 ชุดคำสั่งเริ่มต้น (AA01h)

โฮสจะส่งคำสั่งชุดนี้เพื่อกำหนดกำหนดขนาดภาพและสี หลังจากได้รับชุดคำสั่งนี้แล้วโมดูลจะส่ง คำสั่ง ACK ไปยังโฮสถ้าการส่งไม่มีข้อผิดพลาด มิฉะนั้นโมดูลจะส่งคำสั่ง NACK ออกมา

4.4.1.1 ชนิดของสี

โมดูลจะสามารถรองรับได้ถึง 7 สี ดังต่อไปนี้

2-bit Gray Scale 01h

4-bit Gray Scale 02h

8-bit Gray Scale 03h

12-bit Color 05h

16-bit Color 06h

JPEG 07h

4.4.1.2 ความละเอียดของภาพที่แสดง

80x60 01h

160x120 03h

4.4.1.3 ความละเอียดของไฟล์ JPEG

โหมดของตัวแสดงความละเอียดของไฟล์ JPEG สามารถที่จะรองรับขนาดของภาพดังต่อไปนี้ได้

80x64 01h

160x128 03h

320x240 05h

640x480 07h

4.4.2 การรับภาพ (AA04h)

1. ตัวโฮส ได้รับภาพมาจากด้วยการส่งชุดคำสั่งต่อไปนี้

   การถ่ายภาพชั่วขณะ 10h

   ภาพที่แสดง 02h

   ภาพที่แสดงโดยไฟล์ JPEG 05h

2. การถ่ายภาพชั่วขณะ (aa05h)

   จะเก็บข้อมูลภาพของไฟล์ JPEG หลังจากได้รับชุดคำสั่งต่อไปนี้

   ภาพที่บีบอัด 00h

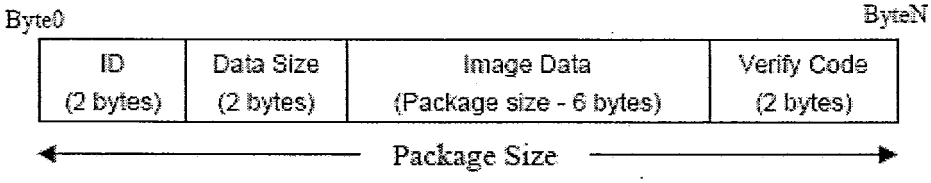
   ภาพที่ไม่บีบอัด 01h

3. การกำหนดแพคเกจ (AA06h)

โฮสจะส่งคำสั่งนี้ออกมาเพื่อทำการเปลี่ยนขาดของภาพ ตัวคำสั่งนี้จะส่งออกมาก่อนที่เราจะทำการส่งคำสั่งถ่ายภาพ หรือคำสั่งรับภาพ

4.4.3 ขนาดแพคเกจ (AA06h)

ขนาดภาพเบื้องต้นมีขนาดเท่ากับ 64 ไบต์ และขนาดมากสุดเท่ากับ 512 ไบต์



รูปที่4.3 โครงสร้างของPackage Size

4.4.4 ชุดคำสั่งบอดเรต (AA07h)

การจะกำหนดบอดเรตโดยใช้คำสั่งนี้ เมื่อโมดูลของเราจะทำการตรวจสอบบอดเรตอัตโนมัติได้ โฮสจะสามารถทำการเชื่อมกับบอดเรตตามสัญญาณดังต่อไปนี้
ได้

Baudrate	1st Divider	2nd Divider	Baudrate	1st Divider	2nd Divider
7200 bps	ffh	01h	28800 bps	3fh	01h
9600 bps	bfh	01h	38400 bps	2fh	01h
14400 bps	7fh	01h	57600 bps	1fh	01h
19200 bps	5fh	01h	115200 bps	0fh	01h

ตารางที่ 4.2 ตารางแสดงค่าบอดเรต

4.4.5 ชุดคำสั่งรีเซต (AA08h)

โฮสจะทำการรีเซตด้วยคำสั่งนี้

00h จะทำการรีเซตทั้งระบบ

01h จะทำการรีเซตแค่กล้อง

4..4.6 เพาเวอร์ออฟ (AA09h)

โมดูลจะทำการประหยัดพลังงานเมื่อได้รับคำสั่งนี้ และจะทำงานอีกครั้งเมื่อได้รับคำสั่ง ACK

4.4.7 การส่งข้อมูล(AA0Ah)

โมดูลจะส่งคำสั่งนี้มาเพื่อจะบอกว่าขนาดของภาพและชนิดภาพที่ส่งและพร้อมที่จะส่งไปยัง โฮสแล้วชนิดของการส่งข้อมูล

ภาพถ่าย 01h

การแสดงภาพตัวอย่าง 02h

การแสดงภาพตัวอย่างแบบ JPEG 05h

โดยไบต์ทั้ง 3 ตัว จะบอกถึงความยาวของข้อมูล

4.4.8 คำสั่งติดต่อหรือ คำสั่ง SYNC (AA0Dh)

โฮสจะทำการส่งคำสั่งนี้เพื่อทำการเชื่อมต่อได้

4.4.9 ACK (AA0Eh)

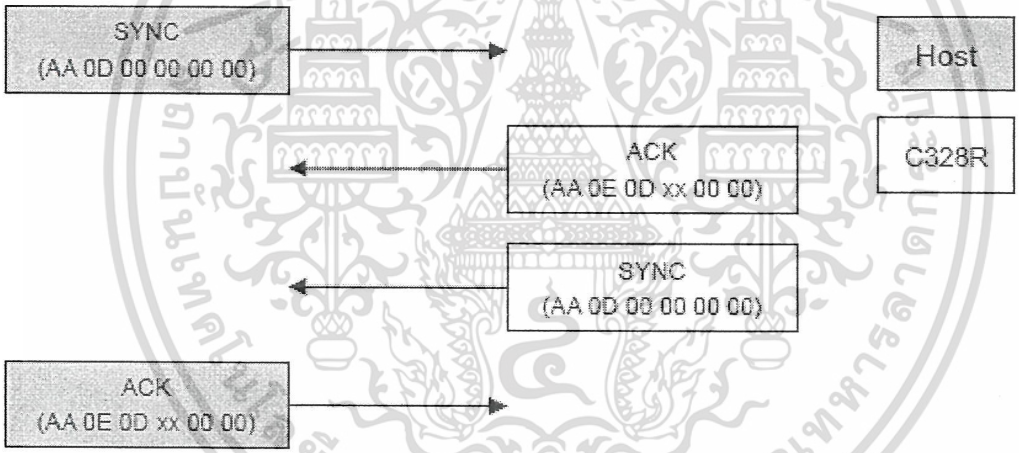
คำสั่งชุดนี้จะเป็นคำสั่งที่โมดูลจะส่งกลับมาหลังจากโฮสส่งคำสั่งที่ถูกต้อง ยกเว้นคำสั่งแสดงข้อมูลของภาพตัวอย่าง

4.4.10 NACK (AA0Fh)

คำสั่งนี้จะเป็นคำสั่งที่กล้องจะส่งมาเมื่อได้รับคำสั่งที่ผิดหรือ ไม่ได้รองรับ

4.5 โปรโตคอล คำสั่ง

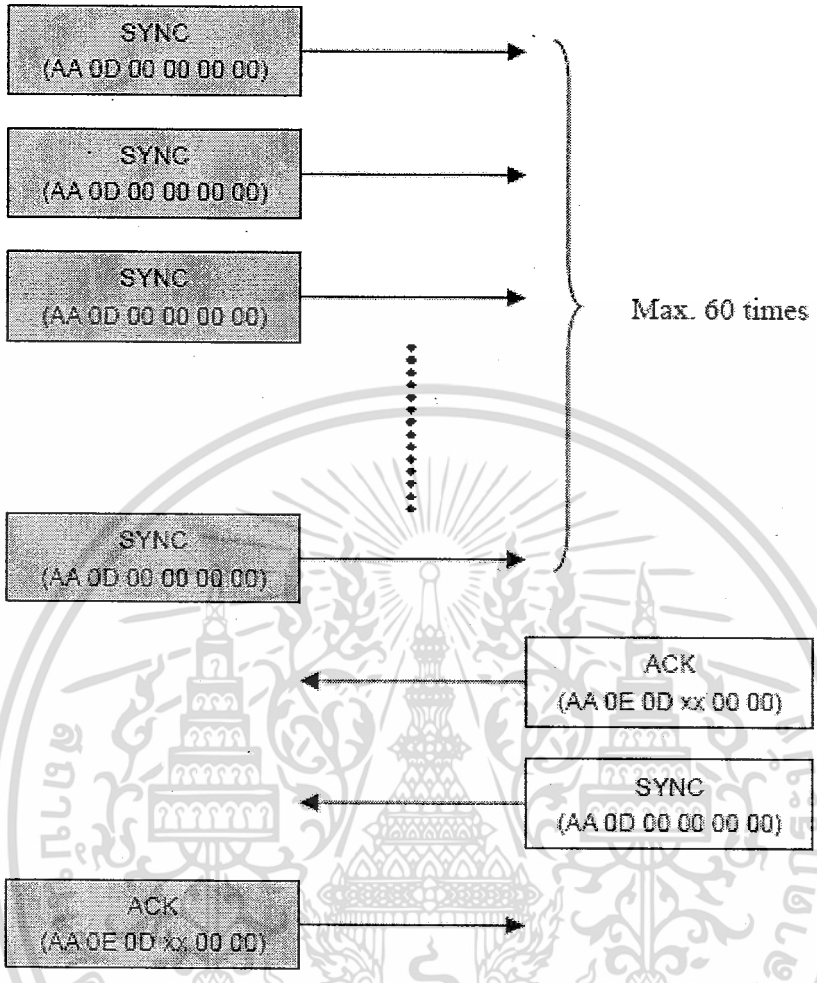
4.5.1 คำสั่ง SYNC



รูปที่ 4.4 คำสั่ง SYNC

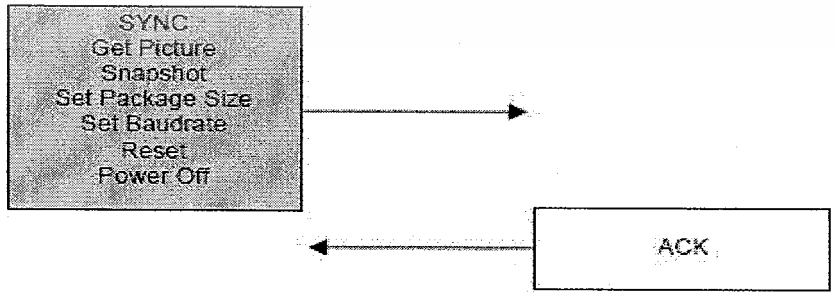
4.5.2 คำสั่งติดต่อกับ C328 -7640

จะส่งคำสั่ง SYNC ที่ 14400bps จนกว่าจะได้รับ คำสั่ง ACK จากโมดูล โดยคำสั่งนี้จะส่งคำสั่งนี้สำเร็จหลังจากจ่ายไฟเข้าโมดูลแล้ว



รูปที่ 4.5 คำสั่งติดต่อกับ C328-7640

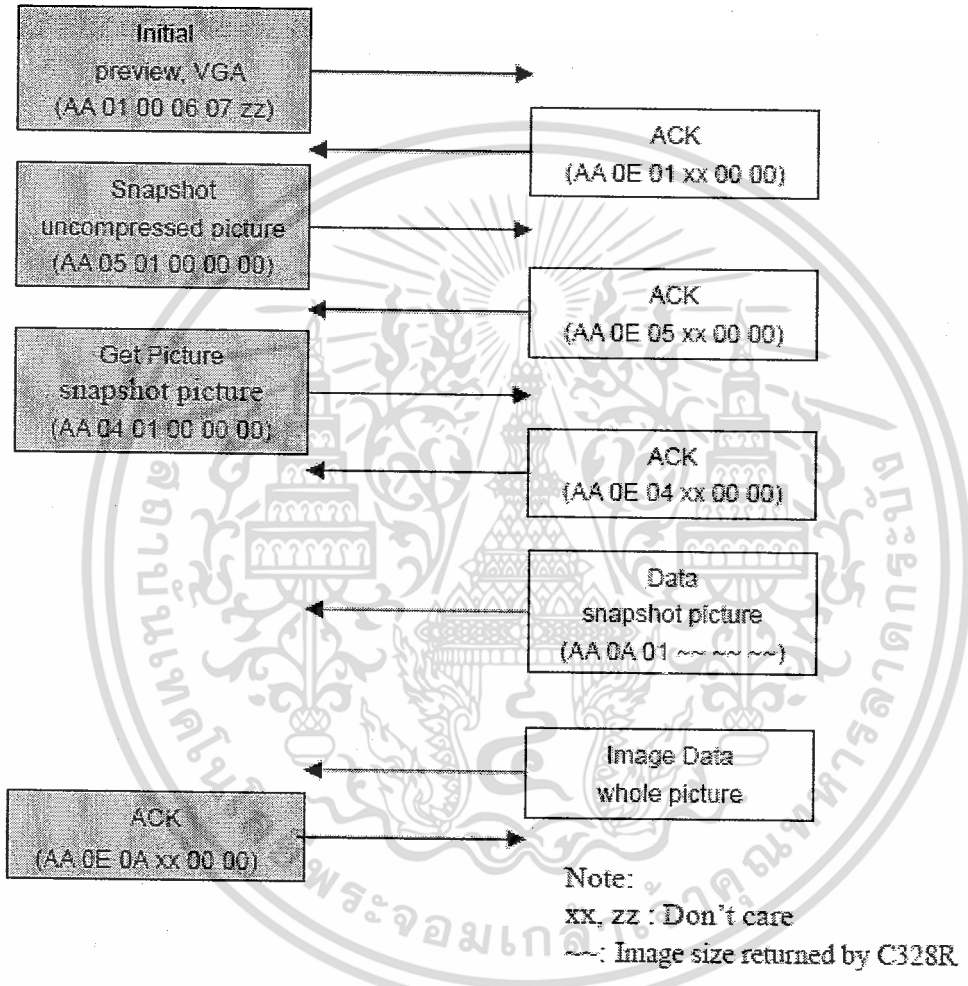
4.5.3 การเริ่มต้นส่งภาพ ถ่ายภาพ และกำหนดขนาดภาพในการส่ง และกำหนดบอดเรตและรีเซตและหยุดจ่ายไฟ



รูปที่ 4.6 การเริ่มต้นส่งภาพ ถ่ายภาพ และกำหนดขนาดภาพ

4.5.4 การส่งคำสั่งถ่ายถ่ายผ่าน RS -232

การส่งคำสั่งถ่ายภาพแบบไม่บีบอัดภาพ



รูปที่ 4.7 การส่งคำสั่งถ่าย ถ่ายผ่านRS-232

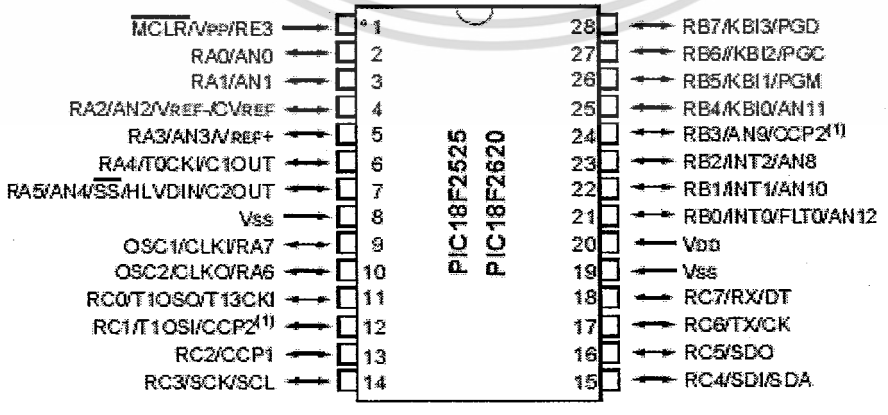
บทที่ 5
ไมโครคอนโทรลเลอร์

โครงงานนี้ใช้ไมโครคอนโทรเลอร์เบอร์ PIC18F2525 ซึ่งทำหน้าที่รับภาพจากกล้องแล้วบันทึกลงในหน่วยความจำเอสดีการ์ด

5.1 รายละเอียดโดยทั่วไป

- สถาปัตยกรรมภายในถูกออกแบบให้ใช้สถาปัตยกรรมแบบ RISE (Reduce Instruction Set Computer)
- สัญญาณนาฬิกาภายใน 31 kHz – 8 MHz
- หน่วยความจำแบบ flash สำหรับบันทึก Program Memory ขนาด 48 kBytes
- หน่วยความจำแบบ EEPROM สำหรับเก็บ DATA Memory ขนาด 1024 Bytes
- หน่วยความจำแบบ RAM 3,968 Bytes
- อินพุต/เอาต์พุต 25 พอร์ต
- ย่านแรงดันใช้งาน 2.0-5.5 โวลต์
- 10 บิต ADC, 10 ช่อง, 100k sample ต่อวินาที
- รองรับ SPI และ I^2C
- 4 โมดูล timer

5.2 ขาการใช้งาน

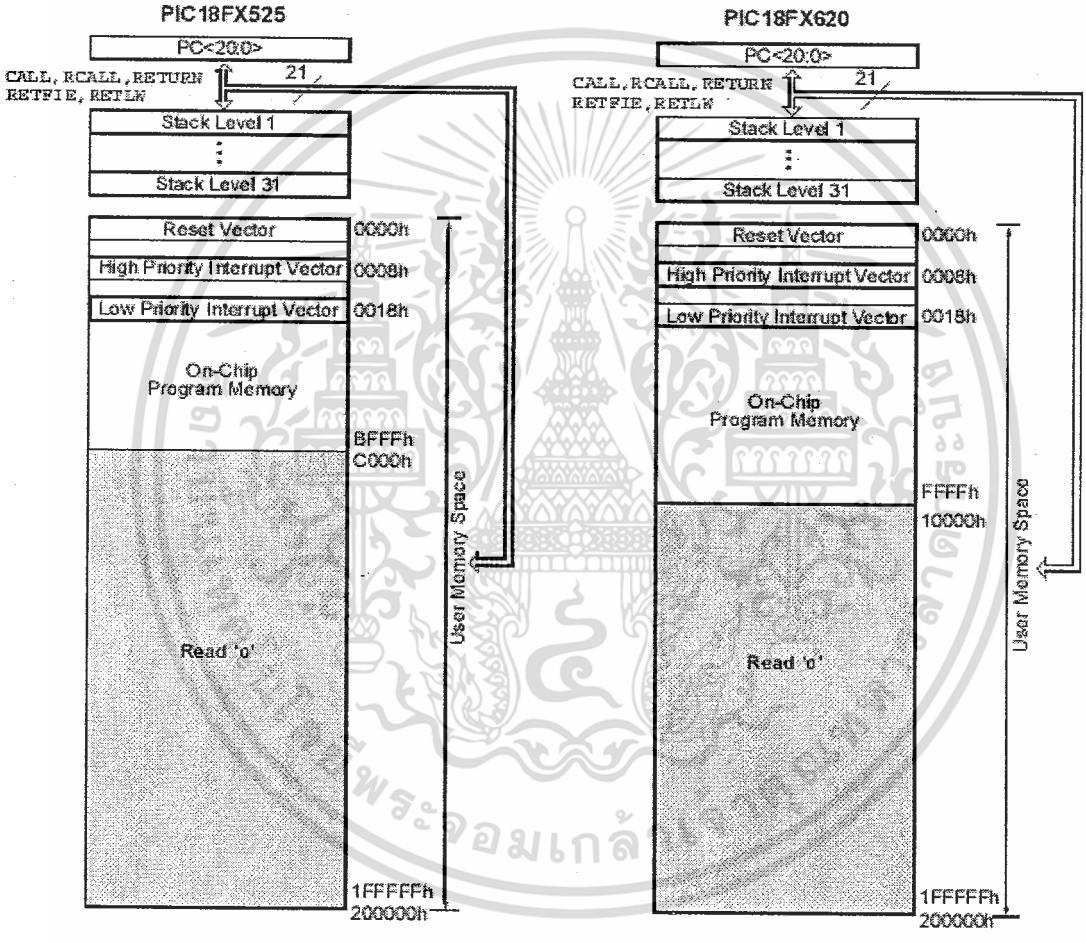


รูปที่ 5.1 แสดงขาใช้งานของ 18F2525

5.3 หน่วยความจำ

PIC18F2525 มีหน่วยความจำ 3 ชนิด คือ

- หน่วยความจำโปรแกรม
- หน่วยความจำ RAM
- หน่วยความจำ EEPROM



รูปที่ 5.2 แสดงหน่วยความจำโปรแกรมของ 18F2525

บทที่ 6

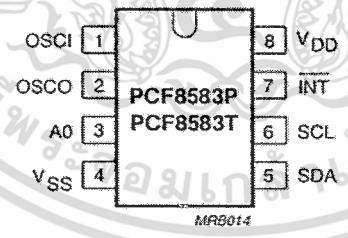
วงจร CLOCK

ในโครงงานนี้เป็นโครงงานเกี่ยวกับกล้องดิจิตอลวงจรปิดซึ่งเป็นการรักษาความปลอดภัยดังนั้น จึงจำเป็นที่จะต้องมีวงจร real time เพื่อจะได้ทราบว่ารูปที่ถ่ายนั้นถ่ายเมื่อใดในโครงงานนี้ใช้เป็นไอซี realtime เบอร์ PCF8583

6.1 คุณลักษณะทั่วไปของ PCF8583

ไอซีนี้เป็นวงจรเวลาและวันที่ ซึ่งการส่งข้อมูลและการกำหนดแอดเดรสจะทำผ่าน I^2C BUS ซึ่งเป็นการส่งข้อมูลโดยใช้สายสัญญาณเพียง 2 เส้น และเวริดแอดเดรสรีจิสเตอร์นั้นเพิ่มโดยอัตโนมัติ หลังจากที่เขียนหรืออ่านข้อมูลและมีวงจรออสซิลเลเตอร์ความถี่ 32 KHz อยู่ภายในและไบต์แรกของ หน่วยความจำภายนอกใช้สำหรับฟังก์ชั่นตัวนับกับเวลาและวันที่และอีก 8 ไบต์ต่อมาสำหรับเสียง สัญญาณเตือน หรือปล่อยให้ว่างไว้กับพื้นที่ที่เหลืออีก 240 ไบต์

6.2 ขาใช้งานของไอซี PCF8583

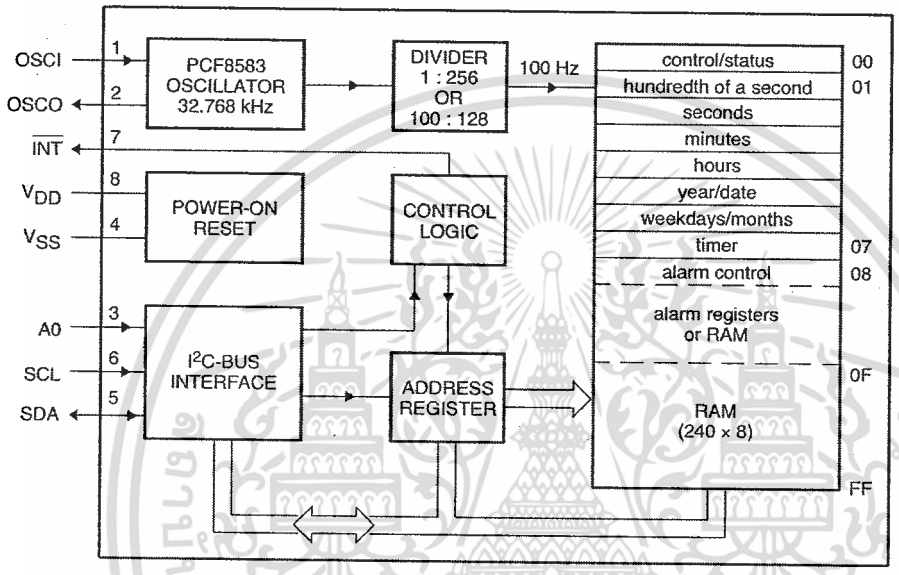


รูปที่ 6.1 รูปขาต่างๆของไอซี PCF8583

ชื่อขา	ขา	คำอธิบาย
OSCI	1	ขาอินพุทออสซิเลเตอร์
OSCO	2	ขาเอาท์พุทออสซิเลเตอร์
A0	3	ขาอินพุทแอดเดรส
V_{ss}	4	ขาแรงดันไฟลบ
SDA	5	ขาสายข้อมูลอนุกรม

SCL	6	ขาสายสัญญาณนาฬิกา
INT	7	ขาอินเตอร์รัปต์
V_{DD}	8	ขาแรงดันไฟบวก

6.3 การทำงานของไอซี PCF8583



รูปที่ 6.2 บล็อกไดอะแกรม

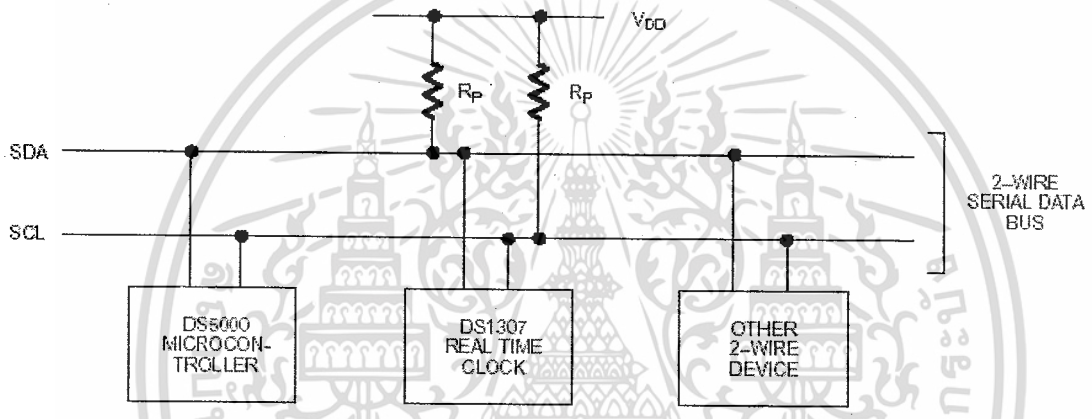
การทำงานไอซีนี้ คือ จะรับสัญญาณมาสร้างฐานเวลาจากขา OSCI และเมื่อผ่านวงจร ออสซิเลเตอร์ แล้วนำมาผ่านวงจรหารความถี่แล้วความถี่ที่ออกมาจะมีค่าเท่ากับ 100 Hz จากนั้นจะส่งต่อไปที่หน่วยความจำและขา A0 เพื่อกำหนดแอดเดรสของตัวฮาร์ดแวร์และขา SCL จะรับสัญญาณนาฬิกามาเพื่อรับส่งข้อมูลที่ขา SDA แล้วส่งมาที่หน่วยความจำโดยผ่าน I^2C อินเตอร์เฟส เพื่อไปเลือก ฟังก์ชั่นใช้งาน

6.4 การจัดการฟังก์ชั่นในหน่วยความจำของไอซี PCF8583

ใน16 ไบต์แรกของหน่วยความจำนั้น (00 – 0F) จะเป็นที่เก็บของฟังก์ชั่นพิเศษ โดยรีจิสเตอร์ แรก (00) ใช้สำหรับการควบคุมและสถานะของไอซี และหน่วยความจำถัดมา (01-07) ใช้เป็นตัวนับ สำหรับฟังก์ชั่นนาฬิกา และสุดท้าย (08-0F) นั้นใช้สำหรับฟังก์ชั่นเสียงเตือนภัย หรือปล่อย ให้ว่างไว้เมื่อไม่ได้ใช้ฟังก์ชั่นนี้

การเชื่อมต่ออุปกรณ์แบบ I^2C ( )

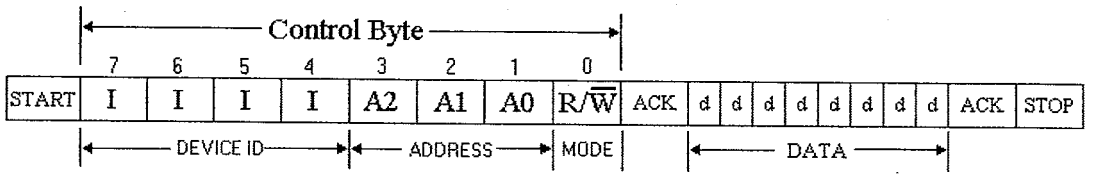
I^2C Bus ย่อมาจาก Inter Integrate Circuit Bus (IIC) นิยมเรียกสั้นๆว่า BUS (ไอ-แสคว-ซี-บัส) เป็นการสื่อสารอนุกรม แบบซิงโครนัส (Synchronous) เพื่อใช้ ติดต่อสื่อสาร ระหว่างไมโครคอนโทรล เลอร์ (MCU) กับอุปกรณ์ภายนอก ซึ่งถูกพัฒนาขึ้นโดยบริษัท Philips Semiconductors โดยใช้สาย สัญญาณเพียง 2 เส้นเท่านั้น คือ serial data (SDA) และสาย serial clock (SCL) ซึ่งสามารถ เชื่อมต่ออุปกรณ์ จำนวนหลายๆ ตัว เข้าด้วยกันได้ ทำให้ MCU ใช้พอร์ตเพียง 2 พอร์ตเท่านั้น



รูปที่ 6.3 การเชื่อมต่ออุปกรณ์แบบ BUS

BUS ใช้สายสัญญาณ 2 เส้น คือ SCL , SDA สำหรับติดกับอุปกร์แบบ 2 ทิศทาง โดยที่ขาสัญญาณทั้ง 2 จะต้องต่อกับตัวต้านทานแบบ pull up 2-10 K เนื่องจากเอาต์พุตมีลักษณะเป็น แบบ Open Darin หรือเป็นแบบ Open Collector เพื่อให้เอาต์พุตเชื่อมต่อกันได้หลายตัว

6.5 การเขียน-อ่านข้อมูลกับอุปกรณ์แบบ I^2C BUS



รูปที่ 6.4 รูปแบบการเขียนและอ่านข้อมูลแบบ BUS

การรับ-ส่งข้อมูลแบบ BUS MCU จะเริ่มต้นการส่งข้อมูลด้วยการ

- ส่งสถานะเริ่มต้น (START Conditions) เพื่อแสดงการขอใช้บัส

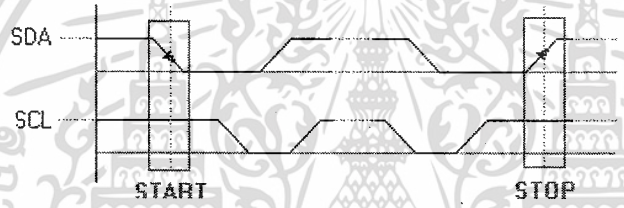
- แล้วตามด้วย รหัสควบคุม (Control Byte) ซึ่งประกอบ ด้วยรหัส ประจำตัวอุปกรณ์ Device ID, Device Address ,และ Mode ในการเขียนหรืออ่านข้อมูล

- เมื่ออุปกรณ์ รับทราบว่า MCU ต้องการ จะติดต่อด้วยก็ต้องส่งสถานะรับรู้ (Acknowledge) หรือแจ้งให้ MCU รับรู้ว่าข้อมูลที่ได้ส่งมามีความถูกต้อง

- และเมื่อสิ้นสุดการส่งข้อมูล MCU จะต้องส่ง สถานะสิ้นสุด (STOP Conditions) เพื่อบอกกับอุปกรณ์ว่า สิ้นสุดการใช้บัส

- สถานะบัสว่าง คือ เมื่อบัสไม่ได้ถูกใช้งาน ทั้ง SCL และ SDA จะเป็น 1 ทั้งคู่

6.5.1 การกำหนดสถานะเริ่มต้นและสถานะสิ้นสุดของ I^2C BUS (START and STOP Conditions)

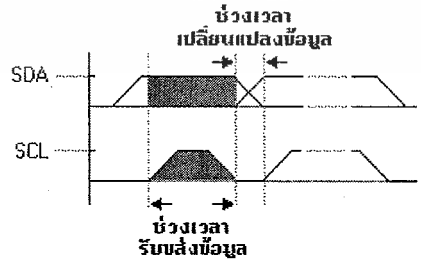


รูปที่ 6.5 ลักษณะการกำหนดสถานะเริ่มต้นและสถานะสิ้นสุดของ BUS

- เมื่อต้องการส่งข้อมูล MCU จะต้องส่งสถานะเริ่มต้น (START Conditions) คือให้ SDA เปลี่ยนจาก 1 มาเป็น 0 ในขณะที่ SCL มีค่าเป็น 1

- เมื่อสิ้นสุดการการใช้บัส MCU จะต้องส่งสถานะสิ้นสุด (STOP Conditions) คือให้ SDA เปลี่ยนจาก 0 มาเป็น 1 ในขณะที่ SCL มีค่าเป็น 1

6.5.2รหัสควบคุมของ I^2C BUS (Control Byte)



รูปที่ 6.6 การรับส่งบิตข้อมูลของ BUS

- สภาวะการรับ-ส่งข้อมูล จะกระทำในขณะที่ขา SCL เป็น 1

- สภาวะการเปลี่ยนแปลงข้อมูล จะกระทำในขณะที่ขา SCL เป็น 0

บทที่ 7

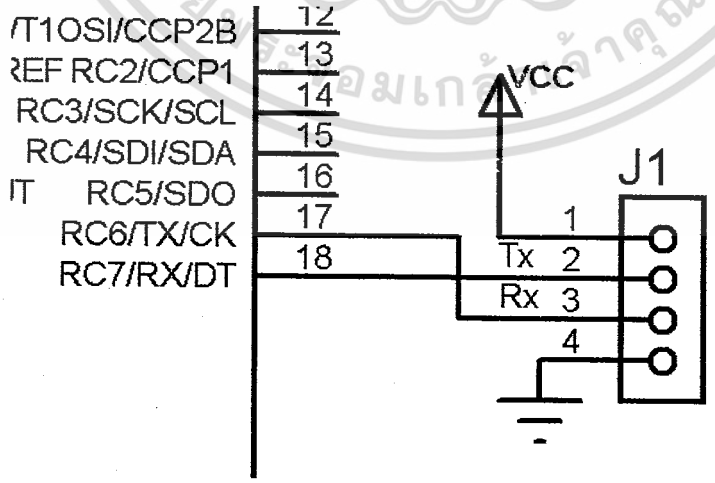
การออกแบบและการสร้าง

การสร้างโครงงานแบ่งออกได้เป็น 4 ส่วนหลักคือ

1. โมดูลกล้อง
2. เอสดีการ์ด
3. ไมโครคอนโทรลเลอร์
4. วงจรRealtime

7.1 โมดูลกล้อง

โครงงานนี้ใช้โมดูลกล้อง C328 ซึ่งเป็นโมดูลกล้องที่สามารถจะส่งข้อมูลภาพออกมาเป็นไฟล์ที่ถูกบีบอัดเป็นไฟล์ชนิดเจเพ็ค (JPEG) หรือเลือกเป็นไฟล์ที่ไม่ถูกบีบอัดได้ ซึ่งการส่งข้อมูลของโมดูลกล้องนั้นเป็นการส่งข้อมูลแบบอนุกรม RS-232 และสามารถเลือกความเร็วในการส่งข้อมูลได้ตั้งแต่ 7200 บิตต่อวินาที ถึง 115200 บิตต่อวินาที โดยขา Tx ของกล้องต่อกับ ขา 18 ของไมโครคอนโทรลเลอร์ซึ่งเป็นขา Rx และขา Rx ของกล้องต่อกับขา 17 ของไมโครคอนโทรลเลอร์ซึ่งเป็นขา Tx โดยไมโครคอนโทรลเลอร์จะส่งคำสั่งต่าง ๆ ออกจากขา 17 และรับข้อมูลจากกล้องผ่านทางขา 18



รูปที่ 7.1 การต่อกล้องกับไมโครคอนโทรลเลอร์

การติดต่อระหว่างกล้องกับไมโครคอนโทรลเลอร์นั้น ไมโครคอนโทรลเลอร์ต้องส่งคำสั่งซิงค์ (sync) เมื่อต้องการเริ่มต่อกับกล้อง และส่งคำสั่งอินนิเชียล (initial) เพื่อกำหนดค่าต่าง ๆ ในการถ่ายภาพ เช่น ขนาดภาพถ่าย ขนาดของภาพตัวอย่าง และสามารถเลือกโหมดในการถ่ายได้ ส่งคำสั่ง snapshot เพื่อถ่ายภาพ สุดท้ายคือคำสั่ง get picture เป็นคำสั่งที่สั่งให้กล้องส่งภาพที่ได้จากการถ่ายออกมาให้ยังไมโครคอนโทรลเลอร์ผ่าน RS-232

7.1.1 โหมดในการถ่ายภาพ

1. สามารถตั้งค่าให้กล้องถ่ายไปเรื่อยๆได้ โดยเมื่อหน่วยความจำเต็มแล้วจะบันทึกไฟล์ใหม่ทับไฟล์แรกที่ถ่าย

2. สามารถตั้งค่าจำนวนภาพที่ต้องการถ่ายรูปได้

3. สามารถตั้งเวลาในการเริ่มต้นถ่ายรูปได้

7.2 เอสดีการ์ด

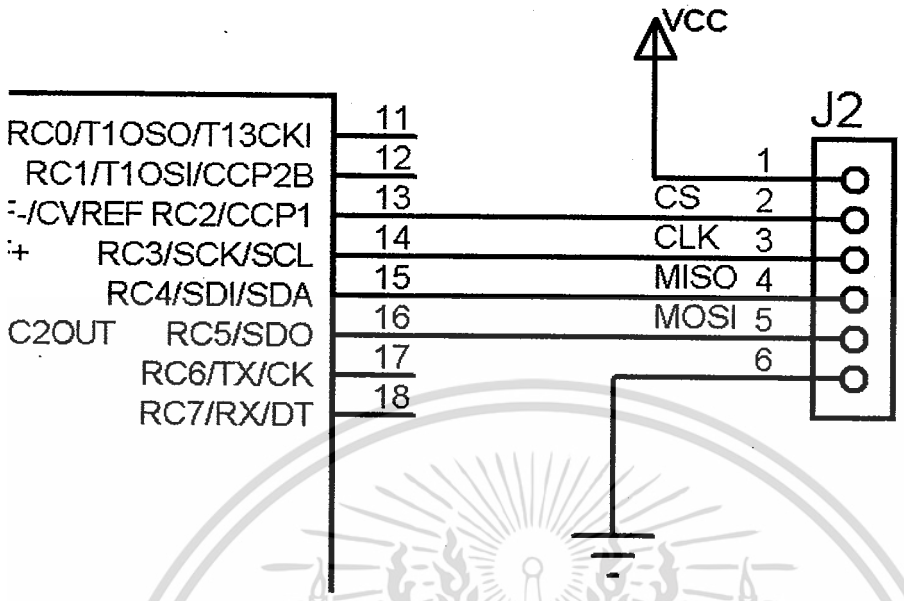
เอสดีการ์ดเป็นการ์ดหน่วยความจำชนิดหนึ่งที่ต้องการไฟเลี้ยง 3.3 โวลต์ ซึ่งในโครงงานได้ใช้การติดต่อกับเอสดีการ์ดผ่านอินเตอร์เฟสแบบ SPI ซึ่งในโหมดนี้เอสดีการ์ดจะมีขาใช้ในการติดต่อกับไมโครคอนโทรลเลอร์ทั้งหมด 4 ขา คือ CS , MISO , MOSI , CLK แต่ละขามีหน้าที่ดังนี้

1. CS (Chip Select) เป็นขาที่ใช้สำหรับเลือกการ์ดให้อยู่ในสถานะพร้อมใช้งาน โดยจะเป็นการเลือกการ์ดนั้น ๆ หากมีสถานะเป็น “ Low ” หากเป็น “ Hi ” การ์ดจะไม่ถูกเลือกและการ์ดจะไม่ทำงาน

2. MISO (Master In Slave Out) เป็นขาที่ไมโครคอนโทรลเลอร์ใช้ให้การรับข้อมูลที่มาจากการ์ด โดยการส่งข้อมูลจะถูกควบคุมด้วย Clock

3. MOSI (Master Out Slave In) เป็นขาที่ไมโครคอนโทรลเลอร์ใช้ในการส่งข้อมูลและเป็นขาที่การ์ดใช้ในการรับข้อมูล โดยการส่งข้อมูลจะถูกควบคุมด้วย Clock

4. CLK (Clock) เป็นขาที่ใช้ควบคุมการรับส่งข้อมูล



รูปที่ 7.2 การต่อเอสดีการ์ดกับไมโครคอนโทรเลอร์

7.3 วงจร Realtime

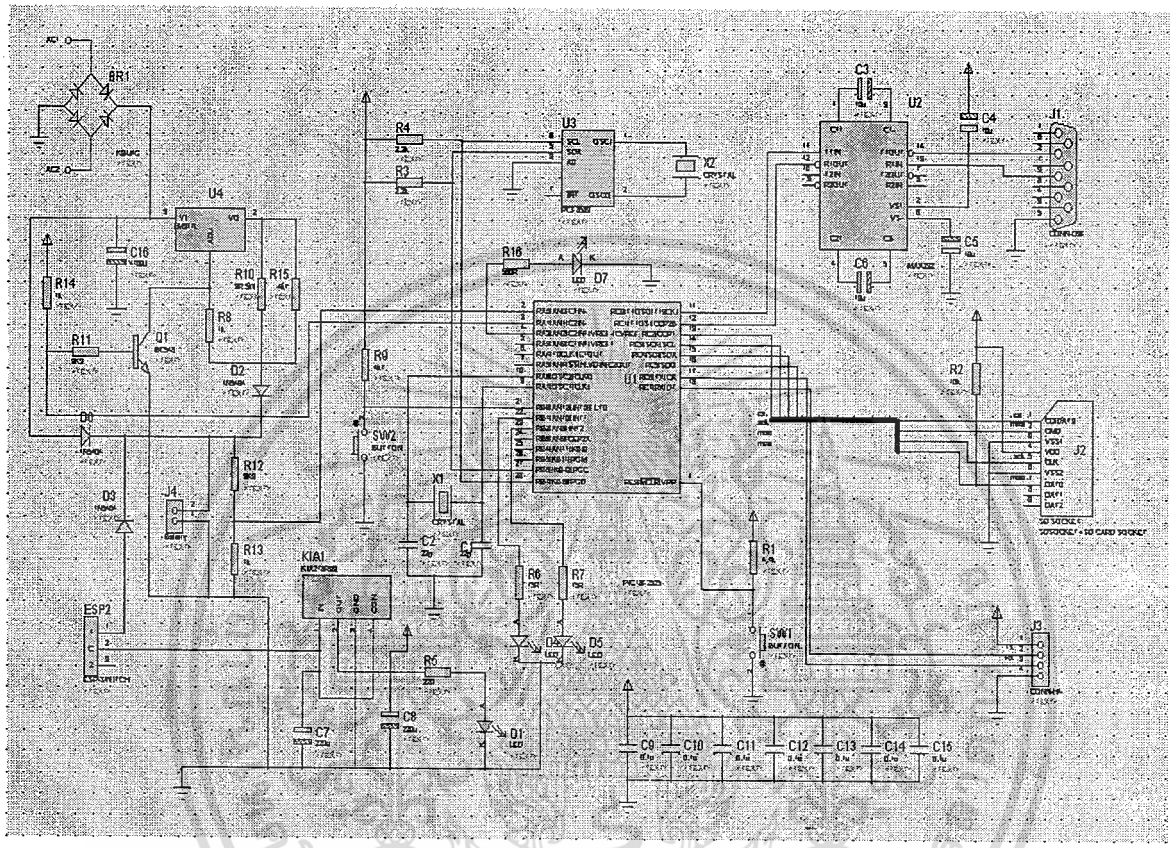
เมื่อเริ่มต้นการทำงานจะจ่ายแรงดันไฟเลี้ยง ไฟเลี้ยง 3.3 โวลต์แล้วจะเริ่มตั้งค่าเพื่อใช้งานไอซี โดยมีขั้นตอนดังนี้

1. เริ่มทำงานด้วยการ initial register ที่แอดเดรส 0x00 เพื่อกำหนดค่าต่างๆ ของ PCF 8385
2. การตั้งค่าเวลา
   - 2.1 เขียน control register เพื่อหยุดการนับเวลา
   - 2.2 ส่งแอดเดรสของ 1/100 วินาที เพื่อเป็นแอดเดรสแรกที่จะเขียน
   - 2.3 เขียนค่าของเวลา เช่น วันที่ เป็นต้น
   - 2.4 เขียน control register (0x00) เพื่อ เริ่มต้นการนับเวลา
3. การอ่านค่า จะต้องส่งค่าแอดเดรสของ second register เพื่อเริ่มต้น register ที่ต้องการอ่านค่า

7.4 ไมโครคอนโทรลเลอร์

การเริ่มต้นการทำงานของไมโครคอนโทรลเลอร์เริ่มที่จ่ายแรงดันไฟเลี้ยง ตั้งแต่ 2-5.5 โวลต์ และ คริสตอลให้กับไมโครคอนโทรลเลอร์ เพื่อเป็นตัวสร้างสัญญาณนาฬิกาให้กับไมโครคอนโทรลเลอร์ ซึ่ง ความถี่ของสัญญาณนาฬิกาที่ไมโครคอนโทรลเลอร์สามารถทำงานได้อยู่ในย่าน 31kHz-32MHz และขา 1 ของไมโครคอนโทรลเลอร์ต้องต่อให้เป็นสถานะ “ Hi ” โดย

การต่อตัวต้านทานพูลอัพเอาไว้ แล้วก็ต่อสวิตช์รีเซตเอาไว้ระหว่างขา 1 กับกราวด์ เมื่อกดสวิตช์รีเซตจะทำให้ขา 1 มีลอจิก “Low” เป็นการรีเซตไมโครคอนโทรลเลอร์



รูปที่ 7.3 รูปวงจรสมบูรณ์

7.5 การเขียนโปรแกรมในไมโครคอนโทรลเลอร์

การเขียนโปรแกรมให้กับไมโครคอนโทรลเลอร์ในโครงงานนี้ได้ใช้ภาษาซีในการเขียนโดยใช้โปรแกรม ccs c ซึ่งเป็นคอมไพล์เลอร์ที่รองรับการเขียนโปรแกรมไมโครคอนโทรลเลอร์ของบริษัทไมโครชิพ (PIC® microchip) ภายในโปรแกรมมี built in function เลือกใช้ได้หลากหลายทำให้การเขียนโปรแกรมทำได้สะดวกขึ้น

7.5.1การทำงานของโปรแกรม

การทำงานของโปรแกรมเริ่มต้นจากการอินนิเชียลเอสดีการ์ด เพื่อเริ่มต้นการทำงานของเอสดีการ์ดให้อยู่ในโหมด SPI และทำการอ่านเอสดีการ์ดว่ามีความจุเท่าใด ตำแหน่งของ FAT (File

Allocation Table) ตำแหน่งของไดเร็กทอรี่ ตำแหน่งของข้อมูล เพื่อที่จะทำการเขียนไฟล์ภาพลงในเอสดีการ์ดได้อย่างถูกต้อง

ลำดับต่อมาคือการอินนิเชียลกล้อง คือ การเตรียมพร้อมกล้องให้พร้อมทำงาน เพื่อให้กล้องรอรับคำสั่งต่อไป เช่น กำหนดบอดเรต (Baud Rate) กำหนดความละเอียดของภาพ (Resolution) จากนั้นไมโครคอนโทรลเลอร์จะค้นหาไฟล์ "CONFIG.TXT" ที่อยู่ภายในเอสดีการ์ด ซึ่งไฟล์ดังกล่าวจะเป็นตัวกำหนดโหมดการทำงานของโปรแกรมที่มีอยู่ 3 โหมดด้วยกัน รวมทั้งเป็นตัวกำหนดความละเอียดของภาพของกล้อง และยังเป็นตัวที่ใช้ในการตั้งเวลาให้กับเรียลไทม์คล็อกอีกด้วย จากนั้นโปรแกรมก็จะแยกทำงานตามโหมดที่ได้ตั้งไว้จากไฟล์ "CONFIG.TXT" ซึ่งมี 3 โหมด ได้แก่

1. ถ่ายตลอดเวลา โปรมแกรมจะยังไม่ถ่ายภาพจนกว่าจะกดปุ่มเริ่ม เมื่อปุ่มเริ่มถูกกดไมโครคอนโทรลเลอร์จะสั่งให้กล้องเริ่มถ่ายภาพ และเก็บภาพที่ได้นั้นบันทึกลงเอสดีการ์ดอย่างต่อเนื่องจนกว่าปุ่มเริ่มจะถูกกดอีกครั้ง เมื่อปุ่มเริ่มถูกกดไมโครคอนโทรลเลอร์จะสั่งให้หยุดการถ่ายภาพ และรอการกดปุ่มอีกครั้งเพื่อเริ่มการถ่ายภาพครั้งใหม่

2. ถ่ายตามจำนวน โปรมแกรมจะยังไม่ถ่ายภาพจนกว่าจะกดปุ่มเริ่ม เมื่อปุ่มเริ่มถูกกดไมโครคอนโทรลเลอร์จะสั่งให้กล้องเริ่มถ่ายภาพ และเก็บภาพที่ได้นั้นบันทึกลงเอสดีการ์ดอย่างต่อเนื่องจนกว่าจะถ่ายภาพจนได้ภาพครบตามจำนวนที่ตั้งไว้ในไฟล์"CONFIG.TXT" เมื่อถ่ายภาพได้ครบตามจำนวนแล้วโปรแกรมจะหยุดถ่ายภาพ และรอรับการกดปุ่มเริ่มครั้งต่อไป

3. ถ่ายภาพตามเวลา โปรแกรมจะอ่านค่าเวลาจากเรียลไทม์คล็อก แล้วนำค่าเวลาปัจจุบันที่อ่านได้ไปเปรียบเทียบว่าอยู่ในช่วงเวลาที่ต้องการให้ถ่ายตามที่ตั้งไว้ในไฟล์ "CONFIG.TXT" หรือไม่ หากเวลาปัจจุบันอยู่ในช่วงเวลาดังกล่าวไมโครคอนโทรลเลอร์จะเริ่มทำการถ่ายรูปจนกว่าเวลาปัจจุบันจะเลยช่วงเวลาที่ตั้งไว้ หากพ้นช่วงเวลานั้นไปแล้วไมโครคอนโทรลเลอร์จะรอจนกว่าถึงเวลานั้นอีกครั้งในวันใหม่

โครงงานนี้ยังได้ถูกออกแบบให้สามารถชาร์ตแบตเตอร์รี่ได้ในตัวอีกด้วย ในกรณีที่แบตเตอร์รี่ถูกใช้พลังงานไปจนหมด ซึ่งการทำงานของโปรแกรมชาร์จแบตเตอร์รี่นั้นได้ใช้โมดูลไทม์เมอร์ที่มีอยู่ภายในตัวไมโครคอนโทรลเลอร์ โดยไมโครคอนโทรลเลอร์จะวัดแรงดันจากแบตเตอร์รี่ว่าจำเป็นต้องชาร์จหรือไม่ ถ้าจำเป็นต้องชาร์จไมโครคอนโทรลเลอร์จะสั่งให้วงจรชาร์จแบตเตอร์รี่ทำการชาร์จแบตเตอร์รี่ และทำการตรวจสอบว่าแบตเตอร์รี่ถูกชาร์จจนเต็มแล้วหรือไม่ทุก ๆ 30 นาที

รูปแบบการเขียนไฟล์ “CONFIG.TXT”

#resolution_1 ตัวเลขที่ตามหลังเป็นการกำหนดความละเอียดของภาพ

1 คือ ความละเอียด 80x64

2 คือ ความละเอียด 160x128

3 คือ ความละเอียด 320x240

4 คือ ความละเอียด 640x480

#mode_3 ตัวเลขที่ตามหลังเป็นการกำหนดโหมดการทำงานของโครงงานนี้

1 คือโหมดถ่ายต่อเนื่อง ตัวอย่างเช่น #mode_1 ไม่ต้องมีอาร์กิวเมนต์ตามหลัง

2 คือโหมดถ่ายตามจำนวน ตัวอย่างเช่น #mode_2 (123) อาร์กิวเมนต์คือจำนวนภาพ

3 คือโหมดตั้งเวลาถ่าย ตัวอย่างเช่น #mode_3 (15.30-18.55) อาร์กิวเมนต์คือช่วงเวลาที่ต้องการถ่ายภาพ

#set_date_time (07/01/09,10:30) เป็นการตั้งวันที่และเวลาให้กับเรียลไทม์คล็อก ซึ่งรูปแบบการเขียนคือ #set_date_time (DD/MM/YY,hh:mm)

DD : วัน

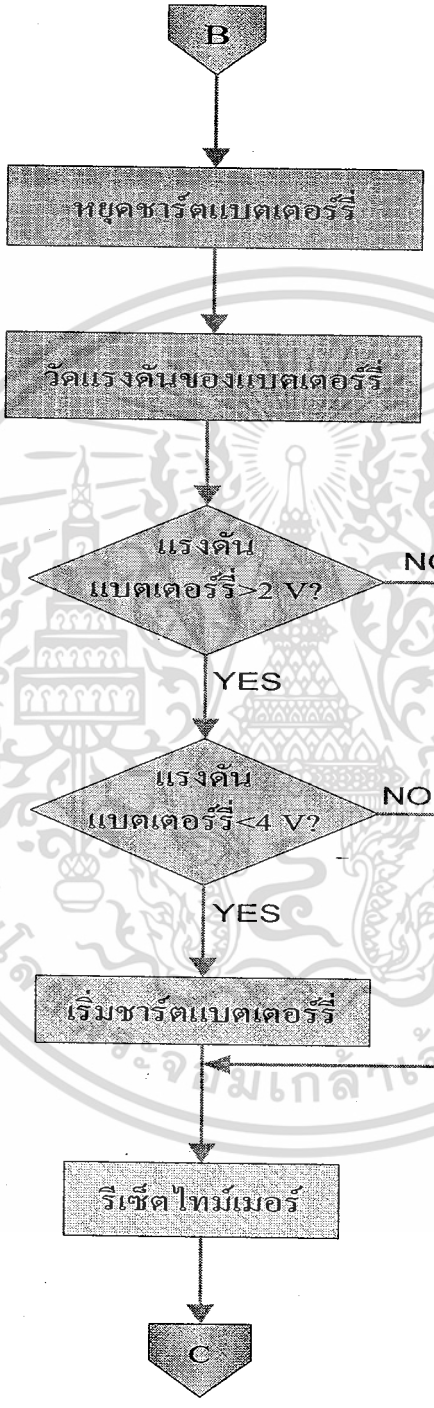
MM : เดือน

YY : ปี

hh : ชั่วโมง(แบบ 24 ชั่วโมง เช่น 23 นาฬิกา)

mm : นาที

แผนผังการทำงานของโปรแกรม



รูปที่ 7.4 แผนผังการทำงานของโปรแกรม

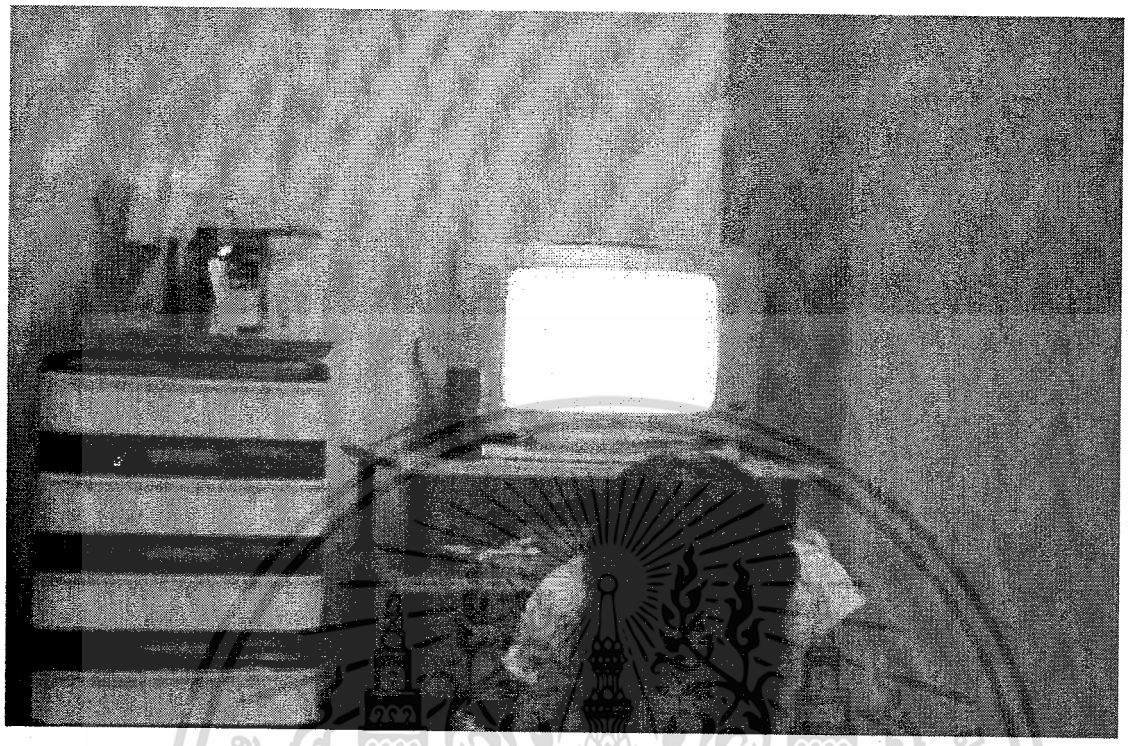
บทที่ 8
บทสรุป

8.1 สรุป

เป้าหมายของการโครงงานนี้ก็คือสามารถถ่ายภาพโดยใช้โมดูลกล้องถ่ายภาพต่อเนื่องทุก ๆ 1 วินาที แล้วเก็บภาพที่ได้จากโมดูลกล้องบันทึกลงในหน่วยความจำเอสดีการ์ด ซึ่งลักษณะของการบันทึกลงในเอสดีการ์ดเป็นการบันทึกในรูปแบบของ FAT16 ซึ่งจะทำให้สามารถอ่านภาพถ่ายจาก SD card ด้วยคอมพิวเตอร์ได้โดยตรง แต่ผลลัพธ์สุดท้ายผู้พัฒนาโครงงานไม่สามารถที่จะทำได้ตามเป้าหมาย คือไม่สามารถที่จะถ่ายภาพด้วยความเร็ว 1 ภาพต่อวินาทีได้เนื่องจากโมดูลกล้องที่ใช้มีการส่งข้อมูลด้วยอินเทอร์เฟส RS232 ความเร็วสูงสุดที่ 115200 บิตต่อวินาที ทำให้การส่งภาพจากโมดูลกล้องมายังไมโครคอนโทรลเลอร์เป็นทำได้ช้า ดังนั้นความเร็วในการบันทึกภาพจึงไม่เป็นไปตามเป้าหมาย

ส่วนของการบันทึกภาพลงในหน่วยความจำเอสดีการ์ดสามารถบันทึกภาพได้ตามเป้าหมาย โดยสามารถนำภาพที่บันทึกลงในหน่วยความจำเอสดีการ์ดไปอ่านจากคอมได้โดยตรง

รูปที่ 8.1 ไฟล์ภาพที่อยู่ในหน่วยความจำเอสดีการ์ด



รูปที่ 8.2 ภาพที่ได้จากกล้อง C328

8.2 อุปสรรคในการทำงาน

ปัญหาและอุปสรรคที่พบใจการทำโครงงานนี้คือ

1. การเขียนโปรแกรมลงบนไมโครคอนโทรลเลอร์ต้องมีการตรวจสอบความถูกต้องของโปรแกรม(Debug) ผ่านพอร์ตอนุกรมของคอมพิวเตอร์ ซึ่งในคอมพิวเตอร์รุ่นใหม่ ๆ บางเครื่องไม่มีพอร์ตอนุกรมจึงเป็นข้อจำกัดในการใช้คอมพิวเตอร์ในการเขียนโปรแกรม

2. การทดลองติดต่อกับโมดูลกล้องต้องใช้เวลานานในการเขียนโปรแกรมเพราะ รายละเอียด โปรโตคอลของโมดูลกล้องที่มาจากผู้ผลิตนั้นมีข้อมูลที่ไม่ละเอียดพอ ผู้พัฒนาโครงงานจึงต้องทดลองแบบลองผิดลองถูก

แนวทางในการพัฒนาและประยุกต์ใช้ร่วมกับงานอื่น ๆ ในอนาคต

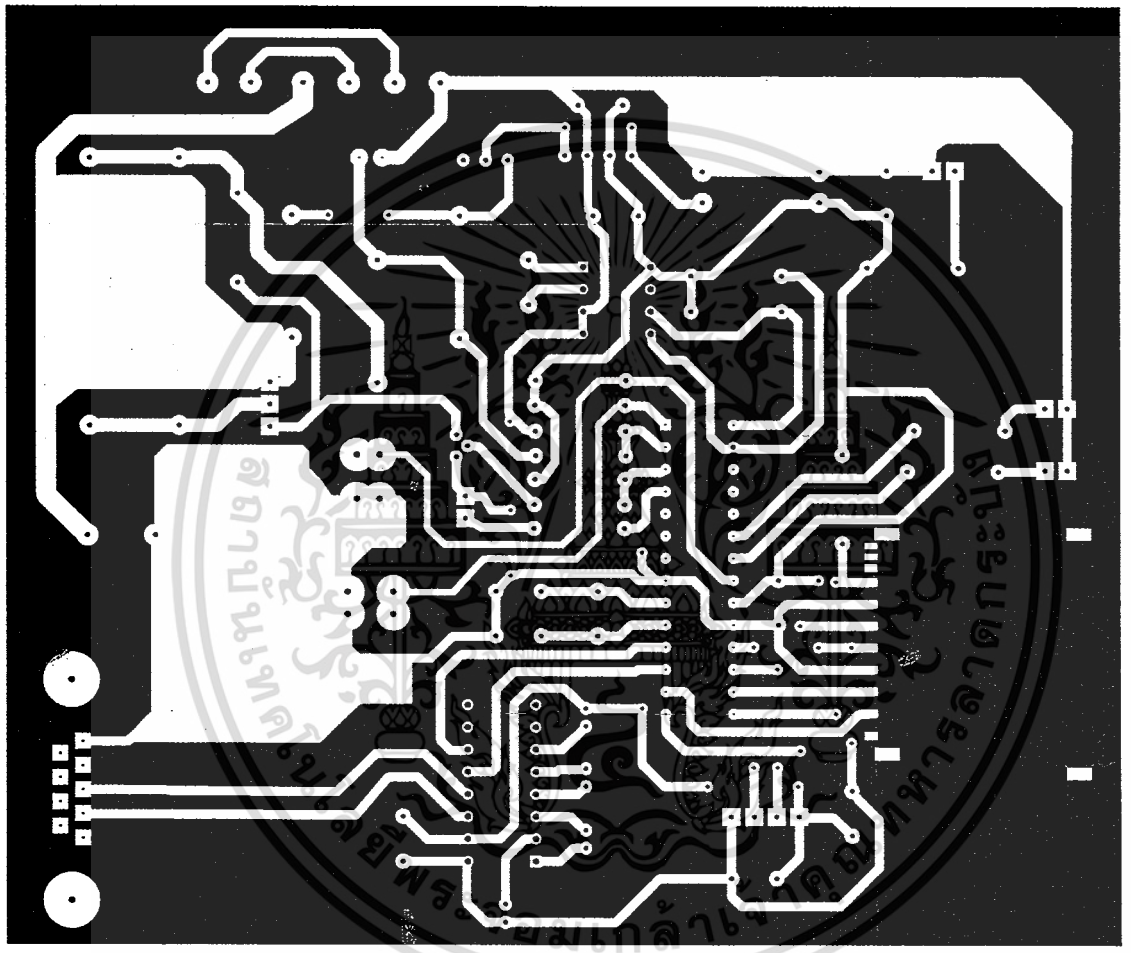
1. ปรับปรุงความความละเอียดของภาพ และความเร็วการถ่ายภาพ โดยใช้โมดูลกล้องที่มีความละเอียดสูงและสามารถส่งภาพได้ความเร็วมากกว่าเดิม

2. นำไปประยุกต์ใช้กับงานด้านความผลปลอดภัย

3. ประยุกต์ใช้เป็นกล้องวงจรปิดที่เลือกช่วงเวลาการทำงานได้

ภาคผนวก

ลายวงจรกล้องดิจิตอลวงจรปิดที่ออกแบบ



โค้ดโปรแกรม

```
#include "D:\project\main.h"
#include "C328.C"
#include "mmc.c"
#include "PCF8583.C"
#include <ctype.h>

int32 total_sector,total_cluster,dir_addr;
int16 byte_sector,sector_fat,reserve,first_sector,
      fat1_addr,fat2_addr,first_data,number_shot,shot_count,
      cnt_time=0,adc_value_old,adc_value_new;
int8  sector_cluster,config;
int1  start=0,charge=0
////////////////////////////////////write_block////////////////////////////////////
#separate
int write_block(int32 address,char *buff)
{
     long i;
     int8 r1;
     r1 = mmc_cmd(0x58,address,16);
     if (r1==0x00)        goto send_token ;                    // we can send data payload
     if (r1==0x40) goto invalid;
     return(false);
invalid:   return(false);

send_token: SPI_READ(0xFE);
     for (i=0;i < 512;i++){
          SPI_READ(buff[i]);       // send payload
     }
```

```
        SPI_READ(0xFF);             // send dummy chcksum
        SPI_READ(0xFF);
        r1=SPI_READ(0xFF);
        for( i=0;i<0x0fff;i++)
        {
            r1=SPI_READ(0xFF);// digest prior operation
            if(r1!=0x00) break;
        }
}

////////////////////////////////clr directory////////////////////////////////
#separate
int clr_dir(int32 fst_cluster,char *data)
{
    int8 i;
    int16 j;
    for(i=0;i<32;i++)               //loop of dir sector, have 32 sector
    {
        read_block((dir_addr+i)*512,data);
        fprintf(com,"\n\r\ %lu \n\r",dir_addr+i);
        for(j=0;j<16;j++)           //loop of 16 byte of dir in 1 sector
        {
            if(make32(0,0,data[j*32+0x1B],data[j*32+0x1A])==fst_cluster)
            {
                if(data[j*32]!=0xE5)
                {
                    data[j*32]=0xE5;
                    write_block((dir_addr+i)*512,data);
                    return(1);
                }
```

```
            }
        }
    }
    return(0);
}

//////////////////////////////// CLR_FAT ////////////////////////////////////
#separate
int clr_fat(int32 Size_Cluster,int32 nxt,char *buff,char *data)          //recieve size of cluzter
{
    int32 i;
    int8 j=0;
    int1 chk=1;                          //chk indecate the last is FFFF if chk=1
    if((nxt+Size_Cluster)>total_cluster)  //nxt is pointer that point to start cluster
        nxt=3;
    read_block((nxt/256+fat1_addr)*512,buff);  //read FAT at sector of first
    for(i=(nxt*2);i<(nxt+Size_Cluster)*2;i++)  // in range of cluster of data
    {
        if(i%512==0)                    // i%512=0 indicate it is end of sector
        {
            j+=1;
            write_block((nxt/256+(j-1)+fat1_addr)*512,buff);     //save data changed
            write_block((nxt/256+(j-1)+fat2_addr)*512,buff);
            read_block((nxt/256+j+fat1_addr)*512,buff);     //open next sector
        }
        if((buff[i%512]==0xff)&&(buff[(i+1)%512]==0xff))  //if found end of cluster
        {
            if(i==6)                    // here is first cluster and have one byte
            {
                buff[i%512]=make8(i/2+1,0);
```

```
            buff[(i+1)%512]=make8(i/2+1,1);
            clr_dir(2,data);                    // clr directory
            chk=1;
            i+=1;
        }
        else if(chk==1)
        {
            buff[i%512]=make8(i/2+1,0);
            buff[(i+1)%512]=make8(i/2+1,1);
            clr_dir(i/2,data);
            i+=1;
        }
        else                        //(chk==0)          //this is end of file
        {
            chk=1;
            buff[i%512]=make8(i/2+1,0);
            buff[(i+1)%512]=make8(i/2+1,1);
            i+=1;
        }
    }
    else    // buff!=ff
    {
        if(chk==1)
        {
            if((buff[i%512]==0)&&(buff[(i+1)%512]==0))
            {
                chk=1;
            }
            else
            {
```

```
                clr_dir(i/2,data);
                chk=0;
            }
            buff[i%512]=make8(i/2+1,0);
            buff[(i+1)%512]=make8(i/2+1,1);
            i+=1;
        }
        else
        {
            buff[i%512]=make8(i/2+1,0);
            buff[(i+1)%512]=make8(i/2+1,1);
            i+=1;
        }
    }
}
buff[(i-2)%512]=0xFF;
buff[(i-2)%512+1]=0xFF;
write_block((nxt/256+j+fat1_addr)*512,buff);
write_block((nxt/256+j+fat2_addr)*512,buff);
read_block((nxt/256+j+fat1_addr)*512,buff);          //plus 18/8/08
while(chk==0)
{
    if(i%512==0)                    // i%512=0 indicate it is end of sector
    {
        j+=1;
        write_block((nxt/256+(j-1)+fat1_addr)*512,buff);    //save data changed
        write_block((nxt/256+(j-1)+fat2_addr)*512,buff);
        read_block((nxt/256+j+fat1_addr)*512,buff);         //open next sector
    }
```

```
        if((buff[i%512]==0xff)&&(buff[i%512+1]==0xff))
            {
                chk=1;
                buff[i%512]=0;
                buff[i%512+1]=0;
                write_block((nxt/256+j+fat1_addr)*512,buff);
                write_block((nxt/256+j+fat2_addr)*512,buff);
                break;
            }
            else if((buff[i%512]==0)&&(buff[i%512+1]==0))
            {
                write_block((nxt/256+j+fat1_addr)*512,buff);
                write_block((nxt/256+j+fat2_addr)*512,buff);
                break;
            }
            else
            {
                buff[i%512]=0;
                buff[i%512+1]=0;
            }
            i+=2;
        }
        return 1;
}
//////////////////////////////// copy blog  ////////////////////////////////
#separate
int cpy_block (char *buff)
{
        int16 i;
```

```
    for(i=0;i<512;i++)
    {
        buff[i]=buff[i+512];
    }
    return 1;
}
////////////////////////////////  GET MBR INFO  ////////////////////////////////
#separate
int get_mbr (void)
{
    int8 data[512];
    int32 i;
    i=0;

    while(1)
    {
        read_block(i*512,data);
        if((data[0]==0xeb)&&(data[2]==0x90)){
            first_sector=i;
            break;
        }
        i+=1;
        if(i>5000) return(0);
    }
    if((data[0x13]+data[0x14])==0)
    total_sector=(make32(data[0x23],data[0x22],data[0x21],data[0x20]));
    if((data[0x13]+data[0x14])!=0)
    total_sector=(int32)make16(data[0x14],data[0x13]);
    byte_sector=make16(data[0x0c],data[0x0b]);
```

```
        reserve=make16(data[0x0f],data[0x0e]);
        sector_cluster=data[0x0d];
        sector_fat=make16(data[0x17],data[0x16]);
        first_data=first_sector+sector_fat*2+reserve+32;
        total_cluster=(total_sector-(first_data-first_sector))/sector_cluster;
        fat1_addr=first_sector+reserve;
        fat2_addr=fat1_addr+sector_fat;
        dir_addr=fat2_addr+sector_fat;
        return(1);
}
////////////////////////////////////write_dir//////////////////////////////////
#inline
int write_dir(int8 *count_dir,int32 *data_length,int16 data_location,int8 *name2,int8 *name1,int8
*name0)
{
      int8 data[512];
      int16 i=0,create_date;
      int32 create_time;

      create_date=get_dir_date();
      create_time=get_dir_time();
      read_block((*count_dir+dir_addr)*512,data);   //(dir_addr+*count)
      while((data[32*i]!=0)&&(data[32*i]!=0xE5))
      {
            i++;
            if(i==16)
            {
                  i=0;
                  *count_dir+=1;
```

```
            if(*count_dir==32)
            {
                *count_dir=0;
            }
            read_block((*count_dir+dir_addr)*512,data);   //(dir_addr+*count)
        }
    }
    data[32*i]  ='K';
    data[32*i+1]='M';
    data[32*i+2]='I';
    data[32*i+3]='T';
    data[32*i+4]='L';
    data[32*i+5]=*name2;
    data[32*i+6]=*name1;
    data[32*i+7]=*name0;
    data[32*i+8]='J';
    data[32*i+9]='P';
    data[32*i+0x0A]='G';
    data[32*i+0x0B]=0x20;          //entry attributes
    data[32*i+0x0C]=0x18;
    data[32*i+0x0D]=make8(create_time,0);         // create time
    data[32*i+0x0E]=make8(create_time,1);         // create time
    data[32*i+0x0F]=make8(create_time,2);         // create time
    data[32*i+0x10]=make8(create_date,0);          //create date
    data[32*i+0x11]=make8(create_date,1);           //create date
    data[32*i+0x1A]=make8(data_location,0);          //data location
    data[32*i+0x1B]=make8(data_location,1);          //data location
    data[32*i+0x1C]=make8(*data_length,0);
    data[32*i+0x1D]=make8(*data_length,1);
    data[32*i+0x1E]=make8(*data_length,2);
```

```
        data[32*i+0x1F]=make8(*data_length,3);
        write_block((*count_dir+dir_addr)*512,data);
        *name0+=1;
        if(*name0==0x3A)
        {
            *name0=0x30;
            *name1+=1;
            if(*name1==0x3A)
            {
                *name1=0x30;
                *name2+=1;
                if(*name2==0x3A)
                {
                    *name2=0x30;
                    *name1=0x30;
                    *name0=0x31;
                }
            }
        }
        return 1;
}
////////////////////////////////////////////////external interrupt//////////////////////////////////////////////////////
#INT_EXT
void ext_isr()
{
        if((config&0x0F)==2)                    //mode 2 trigger
        {
            if(shot_count<=number_shot)
            {
                start=0;
```

```
            }
        }
        else if((config&0x0F)==1)                    //continue mode
        {
            start=!start;
        }
}
////////////////////////////////////////////////////seek config file/////////////////////////////////////////////////
#separate
int8 seek_config (int32 *time_start,int32 *time_end,int16 *number_shot)
{
    int8 j,m,data[512],config=0;     //j=number of sector(0-31)
    int16 k,i;
    int32 n=0,location;
    date_time_t dt;

    for(j=0;j<32;j++)
    {
        read_block((j+dir_addr)*512,data);          //read directory
        for(i=0;i<16;i++)
        {
            if((data[i*32]=='C')&&((data[i*32+4]=='I')&&(data[i*32+5]=='G')))
             {
                if(((data[i*32+1]=='O')&&(data[i*32+2]=='N'))&&(data[i*32+3]=='F'))
                {
                    location=make16(data[i*32+0x1B],data[i*32+0x1A]);
                    read_block(((location-2)*sector_cluster+first_data)*512,data);
                    //read data in config
                    for(k=0;k<512;k++)
                    {
```

```
        if(((data[k]=='#')&&(data[k+1]=='s'))&&
        ((data[k+5]=='d')&&(data[k+10]=='t')))
        {
        dt.day=(data[k+16]&0x0F)*10+(data[k+17]&0x0F);
        dt.month=(data[k+19]&0x0F)*10+(data[k+20]&0x0F);
        dt.year=(data[k+22]&0x0F)*10+(data[k+23]&0x0F);
        dt.hours=(data[k+25]&0x0F)*10+(data[k+26]&0x0F);
        dt.minutes=(data[k+28]&0x0F)*10+(data[k+29]&0x0F);
        PCF8583_set_datetime(&dt);                    //set time of PCF8583
    }
    if(((data[k]=='#')&&(data[k+1]=='r'))&&
    ((data[k+2]=='e')&&(data[k+3]=='s')))
    {
        config=config|(swap(data[k+12]&0x0F));
    }
    if(((data[k]=='#')&&(data[k+1]=='m'))&&
    ((data[k+2]=='o')&&(data[k+3]=='d')))
    {
        config=config|(data[k+6]&0x0F);
        if(data[k+6]=='2')
        {
            *number_shot=(data[k+9]&0x0F)*100+
            (data[k+10]&0x0F)*10+(data[k+11]&0x0F);
        }
        else if(data[k+6]=='3')
        {
            *time_start=(data[k+9]&0x0F)*10+(data[k+10]&0x0F);
            *time_start=(*time_start)<<6;
            *time_start=(*time_start)|((data[k+12]&0x0F)*10+
            (data[k+13]&0x0F));
```

```
                        *time_start=(*time_start)<<13;
                        *time_end=(data[k+15]&0x0F)*10+(data[k+16]&0x0F);
                        *time_end=(*time_end)<<6;
                        *time_end=(*time_end)|((data[k+18]&0x0F)*10+
                        (data[k+19]&0x0F));
                        *time_end=(*time_end)<<13;
                    }
                }
            }
            write_block((n+first_data)*512,data);
            return(config);
        }
    }
}
return(0);
}

////////////////////////////////////////////INTERRUPT TIMER1////////////////////////////////////////////
#int_TIMER1
void  TIMER1_isr(void)                  //timer1 overflow every 131 mS
{
    cnt_time++;
}
////////////////////////////////////////////check battery////////////////////////////////////////////
int1 check_bat(void)
{
    output_bit(PIN_B2,charge);
    if((charge==0)&&(cnt_time>=230))          // time 30 s
```

```
{
    adc_value_new=read_adc();
    if((adc_value_new>256)&&(adc_value_new<512))
    //true if battery are connected && less than 4V
    {
        charge=1;                   //start charge
        output_low(PIN_A1);         //drive current to battery
        adc_value_old=adc_value_new;
        output_high(PIN_B2);
    }
    cnt_time=0;
}
else if((charge==1)&&(cnt_time>=4580))      //10 min
{
    output_high(PIN_A1);            //stop drive current to battery
    if(cnt_time>=4790)              //wait 30 s   wait for bat steady
    {
    adc_value_new=read_adc();
    if(adc_value_new<adc_value_old)      // full
    {
        if((adc_value_old-adc_value_new)>=11)
        {
            charge=0;               //stop charge
            output_high(PIN_A1);    //stop drive current to battery
        }
        else
            output_low(PIN_A1);     //drive current to battery
    }
    else
    {
```

```
                adc_value_old=adc_value_new;
                output_low(PIN_A1);             //drive current to battery
            }
            cnt_time=0;
        }
        else
            return(1);
    }
    else
        return(1);
}
////////////////////////////////////////////////MAIN////////////////////////////////////////////////////////////////////
void main()
{
    int8 count_dir=0,name2=0x30,name1=0x30,name0=0x31;
    int32 i,j,nxt=3,data_length,time_start,time_end,time_buff;
    int16 position=0,max_loop,current_block=0,count=0,data_size;
    char buff[1024],temp[6],data[512];
    #use fast_io(B)
    setup_adc_ports(AN0|VSS_VDD);
    setup_adc(ADC_CLOCK_DIV_64|ADC_TAD_MUL_0);
    setup_spi(SPI_MASTER|SPI_H_TO_L|SPI_XMIT_L_TO_H|SPI_CLK_DIV_16);
    setup_wdt(WDT_OFF);
    setup_timer_0(RTCC_INTERNAL);
    setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);
    setup_timer_2(T2_DISABLED,0,1);
    setup_comparator(NC_NC_NC_NC);
    setup_vref(FALSE);
    setup_oscillator(False);
```

```
    set_tris_B(0xF1);
    ext_int_edge( H_TO_L );  // Sets up EXT
    enable_interrupts(INT_EXT);
    enable_interrupts(INT_TIMER1);
    enable_interrupts(GLOBAL);
    output_high(PIN_A1);              //stop drive current to battery
    output_low(PIN_B2);               //turn off green LED
    set_timer1(0);
    set_adc_channel(0);
    check_bat();
    j=Init(16);                       //intial SD card
    PCF8583_init();
    get_mbr();
    config=seek_config(&time_start,&time_end,&number_shot);
    delay_ms(1000);
    while(!sync());
    start=1;
loop: output_high(PIN_B1);
    delay_ms(1000);
    while(start)
    {
        check_bat();
        if((config&0x0F)==1)          //continue mode
        {
            start=1;
        }
        else if((config&0x0F)==2)
        {
```

```
            start=1;
        }
        else if((config&0x0F)==3)
        {
            time_buff=get_dir_time();
            if((time_buff&0x00F80000)==(time_start&0x00F80000))          //check hour
            {
                if((time_buff&0x0007E000)>=(time_start&0x0007E000))      //check minute
                {
                    start=0;
                }
            }
            if((time_buff&0x00F80000)==(time_end&0x00F80000))            //check hour
            {
                if((time_buff&0x0007E000)>=(time_end&0x0007E000))        //check minute
                {
                    start=1;
                }
            }
        }
}
 while(!sync());
output_low(PIN_B1);
#use rs232(baud=57600,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=cam)
switch(config&0xF0)
{
    case 0x10:fprintf(com,"\n\r%x",initial(0x07,0x01,0x01));
    break;
    case 0x20:fprintf(com,"\n\r%x",initial(0x07,0x01,0x03));
    break;
```

```
        case 0x30:fprintf(com,"\n\r%x",initial(0x07,0x01,0x05));
        break;
        case 0x40:fprintf(com,"\n\r%x",initial(0x07,0x01,0x07));
        break;
        default:fprintf(com,"\n\r%x",initial(0x07,0x01,0x07));
        break;
}
fprintf(com,"\n\r%x",package_size(0x00,0x02));   // pack size is 512
while(1)
{
        snapshot();             //Take photo
        get_pic(01);            // pic type snapshot
        temp[0]=fgetc(cam);                     //read dummies
        temp[1]=fgetc(cam);                     //read dummies
        temp[2]=fgetc(cam);                     //read dummies
        temp[3]=fgetc(cam);
        temp[4]=fgetc(cam);
        temp[5]=fgetc(cam);
        data_size=make32(0,temp[5],temp[4],temp[3]);  //get size of data
        if(data_size%506!=0)
                max_loop=(data_size/506);
        else
                max_loop=data_size/506-1;
        while(1)
        {
                if(position>=512)
                {
                        write_block(((nxt-2)*sector_cluster+first_data+current_block)*512,buff);
                        // save current block
                        current_block+=1;
```

```
        cpy_block(buff);
        position-=512;
    }
    else
    {
        if(count==max_loop)
        {
            cam_cmd(0xAA,0x0E,0,0,make8(count,0),make8(count,1));
            temp[0]=fgetc(cam);       //read dummies
            temp[1]=fgetc(cam);       //read dummies
            temp[2]=fgetc(cam);       //package size
            temp[3]=fgetc(cam);       //package size
            for(i=position;i<(position+make16(temp[3],temp[2]));i++)
            {
                buff[i]=fgetc(cam);
            }
            temp[0]=fgetc(cam);       //read dummies
            temp[0]=fgetc(cam);       //read dummies
            if(i>511)
            {
                write_block(((nxt-2)*sector_cluster+
                first_data+current_block)*512,buff);
                current_block+=1;
                cpy_block(buff);
                write_block(((nxt-2)*sector_cluster+first_data+
                current_block)*512,buff);
            }
            else
            {
```

```
                write_block(((nxt-2)*sector_cluster+first_data+
                current_block)*512,buff);
            }
            goto end;
        }
        else
        {
            cam_cmd(0xAA,0x0E,0,0,make8(count,0),make8(count,1));
            temp[0]=fgetc(cam);          //read dummies
            temp[0]=fgetc(cam);          //read dummies
            temp[0]=fgetc(cam);          //read dummies
            temp[0]=fgetc(cam);          //read dummies
            for(i=position;i<(position+506);i++)
            {
              buff[i]=fgetc(cam);
            }
            temp[0]=fgetc(cam);          //read dummies
            temp[0]=fgetc(cam);          //read dummies
            position=i;
            count+=1;
        }
    }
}
end:cam_cmd(0xAA,0x0E,0x00,0x00,0xF0,0xF0);
 if(data_size%(sector_cluster*512)!=0)
    clr_fat(data_size/(sector_cluster*512)+1,nxt,buff,data);
else
    clr_fat(data_size/(sector_cluster*512),nxt,buff,data);
data_length=data_size;
write_dir(&count_dir,&data_length,nxt,&name2,&name1,&name0);
```

```
        if(data_size%(sector_cluster*512)!=0)
            nxt=nxt+data_size/(sector_cluster*512)+1;
        else
            nxt=nxt+data_size/(sector_cluster*512);
            count=0;
            position=0;
            current_block=0;
//*****************************************************************************
        if((config&0x0F)==2)
        {
            shot_count++;
            if(shot_count>=number_shot)
            {
                start=1;
                shot_count=0;
            }
        }
        else if((config&0x0F)==3)
        {
            time_buff=get_dir_time();
            if((time_buff&0x00F80000)>=(time_end&0x00F80000))        //check hour
            {
                if((time_buff&0x0007E000)>=(time_end&0x0007E000))      //check minute
                {
                    start=1;
                }
            }
        }
```

```
        if(start==1)
        {
            output_high(PIN_B1);          //start = 1 is stop,0 start
            delay_ms(500);
            output_low(PIN_B2);
            reset_cam(0x01);
            pow_off();
            goto loop;
        }
        check_bat();
    }
}
```

```
#include <18F2525.h>
#device adc=10                  //change

#FUSES NOWDT                    //No Watch Dog Timer
#FUSES WDT128                   //Watch Dog Timer uses 1:128 Postscale
#FUSES HS                       //High speed Osc (> 4mhz)
#FUSES NOPROTECT                //Code not protected from reading
#FUSES IESO                     //Internal External Switch Over mode enabled
#FUSES NOBROWNOUT               //No brownout reset
#FUSES BORV20                   //Brownout reset at 2.0V
#FUSES NOPUT                    //No Power Up Timer
#FUSES NOCPD                    //No EE protection
#FUSES STVREN                   //Stack full/underflow will cause reset
#FUSES NODEBUG                  //No Debug mode for ICD
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOWRT                    //Program memory not write protected
#FUSES NOWRTD                   //Data EEPROM not write protected
#FUSES NOEBTR                   //Memory not protected from table reads
#FUSES NOCPB                    //No Boot Block code protection
#FUSES NOEBTRB                  //Boot block not protected from table reads
#FUSES NOWRTC                   //configuration not registers write protected
#FUSES NOWRTB                   //Boot block not write protected
#FUSES FCMEN                    //Fail-safe clock monitor enabled
#FUSES NOXINST                  //Extended set extension and Indexed Addressing mode disabled
#FUSES PBADEN                   //PORTB pins are configured as analog input channels on RESET
#FUSES LPT1OSC                  //Timer1 configured for low-power operation
#FUSES MCLR                     //Master Clear pin enabled

#use delay(clock=16000000)
#use rs232(baud=38400,xmit=PIN_C0,rcv=PIN_C1,bits=8,stream=com)
```

```
#use rs232(baud=14400,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=cam)
#use i2c(Master,sda=PIN_B6,scl=PIN_B7)
```

```
#define cs PIN_C2
//////////////////////////// mmc_cmd/////////////////////////////////////////
#separate
int mmc_cmd(int8 cmd,int32 address,int8 tries)
{
    int i,r1;
    for( i=0;i<16;i++) SPI_READ(0xFF);// digest prior operation
    SPI_READ(cmd);
    SPI_READ(MAKE8(address,3));
    SPI_READ(MAKE8(address,2));
    SPI_READ(MAKE8(address,1));
    SPI_READ(MAKE8(address,0));
    SPI_READ(0x95);
    for(i=0;i< tries;i++){
        r1=SPI_READ(0xFF);
        if ((r1&0x80)==0)  break;
    }
    return(r1);
}
////////////////////////////////////Initial sd card/////////////////////////////////
#separate
int8 Init (int8 max_tries)
{
    int16 i;
    int8 c,tries=0;
    spi_read(0xff);
    output_high(cs);                    //Releasing CS line to "H"
    delay_ms(20);
    for(i=1;i<=20;i++) spi_write(0xff);     //Send min 80 clock
    output_low(cs);                     //Pull CS line to "L"
```

```
    delay_ms(20);
cmd0:   c=mmc_cmd(0x40,0x00000000,128);          //send cmd0
    if(c==0x01){
        goto excmd55;
    }
    if(tries++<max_tries)        goto cmd0;
    else  return(0x01);
    excmd55: tries=0;
cmd55:  c=mmc_cmd(0x77,0x00000000,128);  //send cmd55
acmd41: c=mmc_cmd(0x41,0x00000000,128);          //send ACMD41
    if(c==0x00)
    {
        c=mmc_cmd(0x7A,0x00000000,128);       //send cmd58 to read OCR
        if(c==0x00)
        {
            spi_read(0xff);
            spi_read(0xff);
            spi_read(0xff);
            spi_read(0xff);
        }
        else  return(0x58);                  // if CMD58 is fail return 0x58
        return (0x11);
    }
    else if(c==0x04)
    {                                        // If R1 of ACMD41 is 0x04 indicate that card is mmc
        tries=0;                             // start initial mmc not sdc
cmd1:   c=mmc_cmd(0x41,0x00000000,128);       //send cmd1
        if(c==0){
            return(0x11);
        }
```

```
        if(tries++<max_tries) goto cmd1;
        else return(0x01);
    }
    else
    {                                           // R1 of ACMD is 01
        if(tries++<max_tries) goto cmd55;       // less than max_tries goto cmd55
        else return (0x01);
    }
}
////////////////////////// read_BLOCK ////////////////////////////////////
#separate
int read_BLOCK( int32 address, int8 *buff)
{
    int r1;
    long i,iw;                      // allows large gt 255 buff size addressing
    r1=mmc_cmd(0x51,address,16);
    if (r1==0x00) goto get_token ;      // we can read data payload
    if (r1==0x40) goto invalid;
invalid:   return(0);
get_token:  for(iw=0;iw<1024;iw++){
        r1=SPI_READ(0xFF);
        if (r1==0xFE)  goto  read_data; // read token $FE
    }
    return(0);
read_data:  for (i=0;i<512;i++){
        buff[i]=SPI_READ(0xFF);
    }
    SPI_READ(0xFF); // read crc
    SPI_READ(0xFF);
```

```
    return(1);
}
```

```
/******************************** PCF8583.C*****************************/

#ifndef PCF8583_WRITE_ADDRESS
#define PCF8583_WRITE_ADDRESS 0xA0
#define PCF8583_READ_ADDRESS  0xA1
#endif

// Register addresses
#define PCF8583_CTRL_STATUS_REG   0x00
#define PCF8583_100S_REG          0x01
#define PCF8583_SECONDS_REG       0x02
#define PCF8583_MINUTES_REG       0x03
#define PCF8583_HOURS_REG         0x04
#define PCF8583_DATE_REG          0x05
#define PCF8583_MONTHS_REG        0x06
#define PCF8583_TIMER_REG         0x07

#define PCF8583_ALARM_CONTROL_REG 0x08
#define PCF8583_ALARM_100S_REG    0x09
#define PCF8583_ALARM_SECS_REG    0x0A
#define PCF8583_ALARM_MINS_REG    0x0B
#define PCF8583_ALARM_HOURS_REG   0x0C
#define PCF8583_ALARM_DATE_REG    0x0D
#define PCF8583_ALARM_MONTHS_REG  0x0E
#define PCF8583_ALARM_TIMER_REG   0x0F

// Use the first NVRAM address for the year byte.
#define PCF8583_YEAR_REG          0x10
```

```
// Commands for the Control/Status register.
#define PCF8583_START_COUNTING    0x00
#define PCF8583_STOP_COUNTING     0x80


/*char const weekday_names[7][10] =
{
{"Sunday"},
{"Monday"},
{"Tuesday"},
{"Wednesday"},
{"Thursday"},
{"Friday"},
{"Saturday"}
};
*/
// This structure defines the user's date and time data.
// The values are stored as unsigned integers.  The user
// should declare a structure of this type in the application
// program. Then the address of the structure should be
// passed to the PCF8583 read/write functions in this
// driver, whenever you want to talk to the chip.
typedef struct
{
int8 seconds;   // 0 to 59
int8 minutes;   // 0 to 59
int8 hours;     // 0 to 23  (24-hour time)
int8 day;       // 1 to 31
int8 month;     // 1 to 12
```

```
int8 year;      // 00 to 99
int8 weekday;   // 0 = Sunday, 1 = Monday, etc.
}date_time_t;


//-----------------------------------------
void PCF8583_write_byte(int8 address, int8 data)
{
//disable_interrupts(GLOBAL);
i2c_start();
i2c_write(PCF8583_WRITE_ADDRESS);
i2c_write(address);
i2c_write(data);
i2c_stop();
//enable_interrupts(GLOBAL);
}

//-----------------------------------------
int8 PCF8583_read_byte(int8 address)
{
int8 retval;

disable_interrupts(GLOBAL);
i2c_start();
i2c_write(PCF8583_WRITE_ADDRESS);
i2c_write(address);
i2c_start();
i2c_write(PCF8583_READ_ADDRESS);
retval = i2c_read(0);
i2c_stop();
```

```
enable_interrupts(GLOBAL);
return(retval);
}


void PCF8583_init(void)
{
PCF8583_write_byte(PCF8583_CTRL_STATUS_REG,PCF8583_START_COUNTING);

}

//-----------------------------------------
// This function converts an 8 bit binary value
// to an 8 bit BCD value.
// The input range must be from 0 to 99.

int8 bin2bcd(int8 value)
{
char retval;

retval = 0;

while(1)
 {
  // Get the tens digit by doing multiple subtraction
  // of 10 from the binary value.
  if(value >= 10)
    {
     value -= 10;
     retval += 0x10;
```

```
    }
   else // Get the ones digit by adding the remainder.
    {
     retval += value;
     break;
    }
   }

return(retval);
}

//-----------------------------------------
// This function converts an 8 bit BCD value to
// an 8 bit binary value.
// The input range must be from 00 to 99.

char bcd2bin(char bcd_value)
{
char temp;
temp = bcd_value;
// Shifting the upper digit right by 1 is
// the same as multiplying it by 8.
temp >>= 1;

// Isolate the bits for the upper digit.
temp &= 0x78;

// Now return: (Tens * 8) + (Tens * 2) + Ones
return(temp + (temp >> 2) + (bcd_value & 0x0f));
```

```
}

//-----------------------------------------
void PCF8583_set_datetime(date_time_t *dt)
{
int8 bcd_sec;
int8 bcd_min;
int8 bcd_hrs;
int8 bcd_day;
int8 bcd_mon;

// Convert the input date/time into BCD values
// that are formatted for the PCF8583 registers.
bcd_sec = bin2bcd(dt->seconds);
bcd_min = bin2bcd(dt->minutes);
bcd_hrs = bin2bcd(dt->hours);
bcd_day = bin2bcd(dt->day) | (dt->year << 6);
bcd_mon = bin2bcd(dt->month) | (dt->weekday << 5);

// Stop the RTC from counting, before we write to
// the date and time registers.
PCF8583_write_byte(PCF8583_CTRL_STATUS_REG,PCF8583_STOP_COUNTING);

// Write to the date and time registers.  Disable interrupts
// so they can't disrupt the i2c operations.
disable_interrupts(GLOBAL);
i2c_start();
i2c_write(PCF8583_WRITE_ADDRESS);
i2c_write(PCF8583_100S_REG);   // Start at 100's reg.
i2c_write(0x00);               // Set 100's reg = 0
```

```
i2c_write(bcd_sec);
i2c_write(bcd_min);
i2c_write(bcd_hrs);
i2c_write(bcd_day);
i2c_write(bcd_mon);
i2c_stop();
enable_interrupts(GLOBAL);

// Write the year byte to the first NVRAM location.
// Leave it in binary format.
PCF8583_write_byte(PCF8583_YEAR_REG, dt->year);

// Now allow the PCF8583 to start counting again.
PCF8583_write_byte(PCF8583_CTRL_STATUS_REG,
                   PCF8583_START_COUNTING);
}

//-----------------------------------------
// Read the Date and Time from the hardware registers
// in the PCF8583.  We don't have to disable counting
// during read operations, because according to the data
// sheet, if any of the lower registers (1 to 7) is read,
// all of them are loaded into "capture" registers.
// All further reading within that cycle is done from
// those registers.

void PCF8583_read_datetime(date_time_t *dt)
{
int8 year_bits;
int8 year;
```

```
int8 bcd_sec;
int8 bcd_min;
int8 bcd_hrs;
int8 bcd_day;
int8 bcd_mon;

// Disable interrupts so the i2c process is not disrupted.
disable_interrupts(GLOBAL);

// Read the date/time registers inside the PCF8583.
i2c_start();
i2c_write(PCF8583_WRITE_ADDRESS);
i2c_write(PCF8583_SECONDS_REG);   // Start at seconds reg.
i2c_start();
i2c_write(PCF8583_READ_ADDRESS);

bcd_sec = i2c_read();
bcd_min = i2c_read();
bcd_hrs = i2c_read();
bcd_day = i2c_read();
bcd_mon = i2c_read(0);
i2c_stop();

enable_interrupts(GLOBAL);

// Convert the date/time values from BCD to
// unsigned 8-bit integers.  Unpack the bits
// in the PCF8583 registers where required.
dt->seconds = bcd2bin(bcd_sec);
```

```
dt->minutes = bcd2bin(bcd_min);
dt->hours   = bcd2bin(bcd_hrs & 0x3F);
dt->day     = bcd2bin(bcd_day & 0x3F);
dt->month   = bcd2bin(bcd_mon & 0x1F);
dt->weekday = bcd_mon >> 5;
year_bits   = bcd_day >> 6;

// Read the year byte from NVRAM.
// This is an added feature of this driver.
year = PCF8583_read_byte(PCF8583_YEAR_REG);

// Check if the two "year bits" were incremented by
// the PCF8583.  If so, increment the 8-bit year
// byte (read from NVRAM) by the same amount.
while(year_bits != (year & 3))
    year++;

dt->year = year;

// Now update the year byte in the NVRAM
// inside the PCF8583.
PCF8583_write_byte(PCF8583_YEAR_REG, year);

}
int16 get_dir_date (void)
{
  int8 year=0,mt=0,day=0;
  int16 date=0;
  date_time_t dt;
```

```
    PCF8583_read_datetime(&dt);

    year=dt.year+20;
    mt=dt.month;
    day=dt.day;

    date=date|year;
    date=date<<4;
    date=date|mt;
    date=date<<5;
    date=date|day;
    return date;
}

int32 get_dir_time (void)
{
      int32 time=0;
      date_time_t dt;

       PCF8583_read_datetime(&dt);
      time=time|dt.hours;
      time=time<<6;
       time=time|dt.minutes;
       time=time<<5;
      time=time|(dt.seconds/2);
      time=time<<8;
      time=time|((dt.seconds%2)*100);
       return time;
}
```

```
/************************C328.C*****************************/
//////////////////////////////////////////////////////Send command to camera/////////////////////////////////
#separate
void cam_cmd (int8 cmdH,int8 cmdL,int8 par1,int8 par2,int8 par3,int8 par4)
{
      fputc(cmdH,cam);
      fputc(cmdL,cam);
      fputc(par1,cam);
      fputc(par2,cam);
      fputc(par3,cam);
      fputc(par4,cam);
}
//////////////////////////////////////////////////SYNC CAMERA COMMAND/////////////////////////////////////////
#separate
int sync (void)
{
      int8 i,buff[12];
      long j;
      for(i=0;i<100;i++)
      {
            cam_cmd(0xAA,0x0D,0,0,0,0);                  //send sync command
            for(j=0;j<20000;j++)
            {
                  if(kbhit(cam))                         //wait response from cam
                  {
                        buff[0]=fgetc(cam);
                        buff[1]=fgetc(cam);
                        buff[2]=fgetc(cam);
                        buff[3]=fgetc(cam);
                        buff[4]=fgetc(cam);
```

```
                buff[5]=fgetc(cam);
                buff[6]=fgetc(cam);
                buff[7]=fgetc(cam);
                buff[8]=fgetc(cam);
                buff[9]=fgetc(cam);
                buff[10]=fgetc(cam);
                buff[11]=fgetc(cam);

                if(((buff[0]==0xAA)&&(buff[1]==0x0E))&&(buff[2]==0x0D))
                {
                    if((buff[6]==0xAA)&&(buff[7]==0x0D))
                    {
                        goto leb;
                    }
                    else  return(0);
                }
                else  return(0);
            }
        }
        delay_ms(100);
    }
    return(0);
leb:
    cam_cmd(0xAA,0x0E,0x0D,0,0,0);        //send ack command AA0E0D 000000
     return(1);
}
```

```
//////////////////////////////////////////////////////Initial camera command//////////////////////////////////////////////////////
#separate
int initial (int8 color,int8 prv_size,int8 jpeg_size)
{
    int8 buff[6];
    long j;
    cam_cmd(0xAA,0x01,0x00,color,prv_size,jpeg_size);
    for(j=0;j<20000;j++)
    {
        if(kbhit(cam))//&&(fgetc(cam)==0xAA))
        {
            buff[0]=fgetc(cam);
            buff[1]=fgetc(cam);
            buff[2]=fgetc(cam);
            buff[3]=fgetc(cam);
            buff[4]=fgetc(cam);
            buff[5]=fgetc(cam);
            if(((buff[0]==0xAA)&&(buff[1]==0x0E))&&(buff[2]==0x01))
                return(1);
            else return(0);
        }
    }
    return(0);
}
//////////////////////////////////////////////////////Set package size//////////////////////////////////////////////////////
#separate
int package_size(int8 Lbyte,int8 Hbyte)
{
    long j;
    int8 buff[6];
```

```
        cam_cmd(0xAA,0x06,0x08,Lbyte,Hbyte,0);   //sen AA0608 H L 00
        for(j=0;j<20000;j++)
        {
            if(kbhit(cam))
            {
                buff[0]=fgetc(cam);
                buff[1]=fgetc(cam);
                buff[2]=fgetc(cam);
                buff[3]=fgetc(cam);
                buff[4]=fgetc(cam);
                buff[5]=fgetc(cam);
                if(((buff[0]==0xAA)&&(buff[1]==0x0E))&&(buff[2]==0x06))
                    return(1);
                else  return(0);
            }
        }
        return(0);
}
////////////////////////////////////////////////////////////Take picture////////////////////////////////////////////////////
#separate
int snapshot(void)
{
        int8 buff[6];
        long j;
        cam_cmd(0xAA,0x05,0,0,0,0);
        for(j=0;j<20000;j++)
        {
            if(kbhit(cam))//&&(fgetc(cam)==0xAA))
            {
                buff[0]=fgetc(cam);
```

```
                buff[1]=fgetc(cam);
                buff[2]=fgetc(cam);
                buff[3]=fgetc(cam);
                buff[4]=fgetc(cam);
                buff[5]=fgetc(cam);
                if(((buff[0]==0xAA)&&(buff[1]==0x0E))&&(buff[2]==0x05))
                    return(1);
                else return(0);
            }
        }
        return(0);
}
////////////////////////////////////////////////get picture from camera////////////////////////////////////////////////
#separate
int get_pic (int8 pic_type)                    // pic type
{                                              // 01 snapshot picture
    int8 buff[6];                              // 02 preview picture
    long j;                                    // 03 JPEG preview picture
    cam_cmd(0xAA,0x04,pic_type,0,0,0);
    for(j=0;j<20000;j++)
    {
        if(kbhit(cam))
        {
            buff[0]=fgetc(cam);
            buff[1]=fgetc(cam);
            buff[2]=fgetc(cam);
            buff[3]=fgetc(cam);
            buff[4]=fgetc(cam);
            buff[5]=fgetc(cam);
```

```
            if(((buff[0]==0xAA)&&(buff[1]==0x0E))&&(buff[2]==0x04))
                return(1);
            else return(0);
        }
    }
    return(0);
}
////////////////////////////////////////////////////////Reset camera/////////////////////////////////////////////////
#inline
int reset_cam (int8 reset_type)
{
    int8 buff[6];
    long j;
    cam_cmd(0xAA,0x08,reset_type,0,0,0);
    for(j=0;j<20000;j++)
    {
        if(kbhit(cam))
        {
            buff[0]=fgetc(cam);
            buff[1]=fgetc(cam);
            buff[2]=fgetc(cam);
            buff[3]=fgetc(cam);
            buff[4]=fgetc(cam);
            buff[5]=fgetc(cam);
            if(((buff[0]==0xAA)&&(buff[1]==0x0E))&&(buff[2]==0x08))
                return(1);
            else return(0);
        }
    }
    return(0);      }
```

```
//////////////////////////////////////////////////////Turn off camera//////////////////////////////////////////////////////
#inline
int pow_off (void)
{
     int8 buff[6];
    long j;
    cam_cmd(0xAA,0x09,0,0,0,0);
    for(j=0;j<20000;j++)
    {
        if(kbhit(cam))
        {
            buff[0]=fgetc(cam);
            buff[1]=fgetc(cam);
            buff[2]=fgetc(cam);
            buff[3]=fgetc(cam);
            buff[4]=fgetc(cam);
            buff[5]=fgetc(cam);
            if(((buff[0]==0xAA)&&(buff[1]==0x0E))&&(buff[2]==0x09))
                return(1);
            else  return(0);
        }
    }
     return(0);
}
//////////////////////////////////////////////////////Set baud rate//////////////////////////////////////////////////////
#inline
int set_baud (int8 first_dvide,int8 second_dvide)
{
    int8 buff[6];
    long j;
```

```
cam_cmd(0xAA,0x01,first_dvide,second_dvide,0,0);       //FF01  7200
for(j=0;j<20000;j++)                                  //BF01  9600
{                                                     //7F01  14400
    if(kbhit(cam))                                    //5F01  19200
    {                                                 //3F01  28800
        buff[0]=fgetc(cam);                           //2F01  38400
        buff[1]=fgetc(cam);                           //1F01  57600
        buff[2]=fgetc(cam);                           //0F01  115200
        buff[3]=fgetc(cam);
        buff[4]=fgetc(cam);
        buff[5]=fgetc(cam);
        if(((buff[0]==0xAA)&&(buff[1]==0x0E))&&(buff[2]==0x09))
        return(1);
        else return(0);
    }
}
return(0);
}
```

กิตติกรรมประกาศ

กิตติกรรมประกาศนี้จัดทำขึ้นเพื่อแสดงการขอบคุณต่อ อ. ชินภัทร นันทจิวากรชัย ที่ช่วยริเริ่มแนวคิดให้ผู้ทำโครงงาน และช่วยจัดหาโปรแกรมและอุปกรณ์ต่างๆในการทำโครงงาน ทั้งยังช่วยแนะนำการใช้โปรแกรมต่างๆ เพื่อให้สามารถใช้โปรแกรมได้อย่างถูกต้องและรวดเร็ว อีกทั้งยังช่วยอธิบายกระบวนการเขียนโปรแกรมในบางโปรแกรมอีกด้วย

บรรณานุกรม

1. sandisk secure digital (SD) card product manual , Rev. 1.9
2. ITM – C328 manual