คู่มือการใช้งาน

User's manual

ET-USB FLASH DRIVE

ETT

บริษัท อีทีที จำกัด ETT CO., LTD.

1112/96-98 ถนนสุขุมวิท แขวงพระโขนง เขตคลองเตย กรุงเทพฯ 10110 http:/www.etteam.com

1112/96-98 Sukhumvit Rd., Phrakanong Klongtoey Bangkok 10110 http:/www.ett.co.th

www.etteam.com Tel : 02-7121120 Fax : 02-3917216 email : sale@etteam.com

Manual

ET-USB FLASH DRIVE

MN-ET V1.0

สารบัญ

สารบัญ (ต่อ)

เรื่อง **หน้า**

ตาราง

ET-USB FLASH DRIVE ก็คืออุปกรณ์ที่ใช้สำหรับอ่านเขียนไฟล์ข้อมูล,ลบไฟล์,สร้างไฟล์ และอื่นๆ ที่เก็บอยู่ใน FLASH DRIVE โดยใช้วิธีการส่ง Command ต่างๆ ผ่านทาง Serial Port ไปให้กับ ET-USB FLASH DRIVE เพื่อให้ติดต่อไปยังตัวเก็บข้อมูล Flash Drive โดยสามารถใช้งานร่วมกับ PC ผ่านทาง Port RS232 โดยใช้โปรแกรม Hyperterminal หรือ Procomm หรือโปรแกรมอื่นๆที่สามารถสื่อสารผ่านทาง RS232ได้ เป็นตัวกลางในการรับ-ส่งคำสั่งและข้อมูลต่างๆ รวมทั้งแสดงผลการอ่านหรือการเขียนข้อมูลให้กับผู้ใช้ นอกจากนี้ก็ยังสามารถต่อใช้งานร่วมกับไมโครคอนโทรลเลอร์ (MCU) แทน PC ก็ได้ โดยจะสื่อสารผ่านทาง UART Port ของ MCU

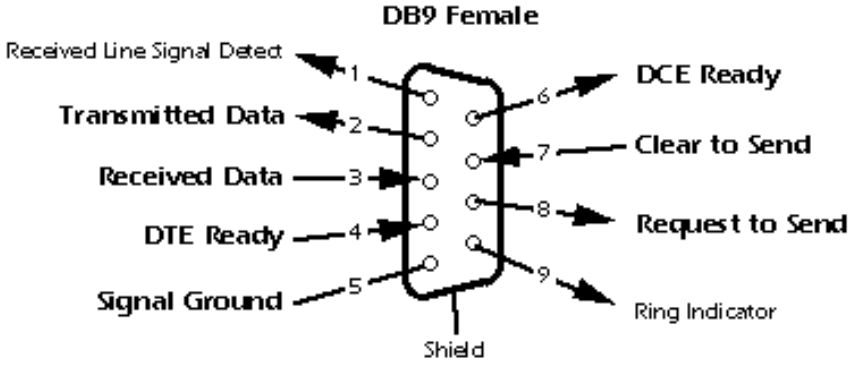
- คุณสมบัติของ ET-USB FLASH DRIVE

1. สามารถติดต่อกับตัวเก็บข้อมูล Flash Drive ที่มีโครงสร้างไฟล์แบบ FAT 12 ,FAT16 หรือ FAT32 ได้
2. รองรับชื่อไฟล์ในรูปแบบ 8.3 คือชื่อไฟล์ไม่เกิน 8 ตัวอักษร นามสกุล 3 ตัวอักษร เช่น A1234567.txt
3. ในระบบ FAT32 จะไม่รองรับชื่อไฟล์แบบยาว ถ้าชื่อไฟล์ยาวเกิน 8.3 จะแสดงชื่อไฟล์ให้เห็นเพียง 8.3
4. ควบคุมการอ่านเขียน Flash Drive โดยใช้การส่ง Command ผ่านทาง RS232
5. สามารถเลือก Baud Rate ในการติดต่อสื่อสารทาง RS232ได้
6. สามารถส่ง Command โดยใช้ PC หรือ MCU ได้
7. สามารถ สร้างและลบ ไฟล์ หรือ Directory ใน Flash Driveได้
8. สามารถกำหนดจำนวน Byte ของข้อมูลที่จะทำการอ่านหรือเขียนจากไฟล์ที่อยู่ใน Flash Driveได้
9. สามารถกำหนดตำแหน่งที่จะอ่านข้อมูลจากไฟล์ หรือเขียนข้อมูลลงไฟล์ ที่อยู่ใน Flash Drive ได้
10 . สามารถอ่านข้อมูลออกมาทีเดียวทั้งไฟล์ จากไฟล์ที่อยู่ใน Flash Driveได้
11. หลังจากปิดไฟล์แล้ว สามารถเปิดไฟล์เดิมขึ้นมาทำการเขียนข้อมูลต่อจากของเดิมได้โดยข้อมูลเก่ายังอยู่
12. สามารถเปลี่ยนชื่อไฟล์หรือชื่อ Directory ใหม่ได้
13. สามารถเข้าไป อ่านเขียน สร้างหรือลบไฟล์ ที่อยู่ใน Directory ย่อยได้
14. สามารถเลือกรูปแบบการส่งคำสั่งได้ 2 แบบ คือ ส่งในรูปแบบอักขระ ASCII (Extended Mode) หรือส่งในรูปแบบ Hex เลขฐาน16 (Short Mode)

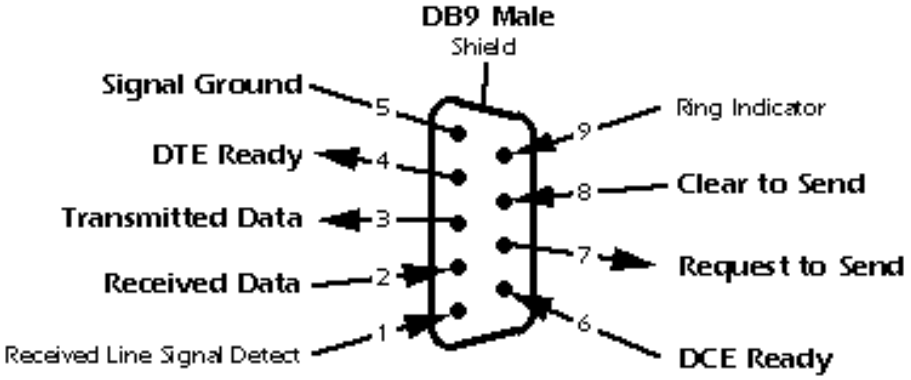
- การต่อใช้งาน ET-USB FLASH DRIVE

การใช้งาน ET-USB FLASH DRIVE จะใช้การ Interface ผ่านทาง Serial Port (RS232 หรือ Uart) โดยจะต้องกำหนดคุณสมบัติในการติดต่อสื่อสารทาง Serial Port ดังนี้

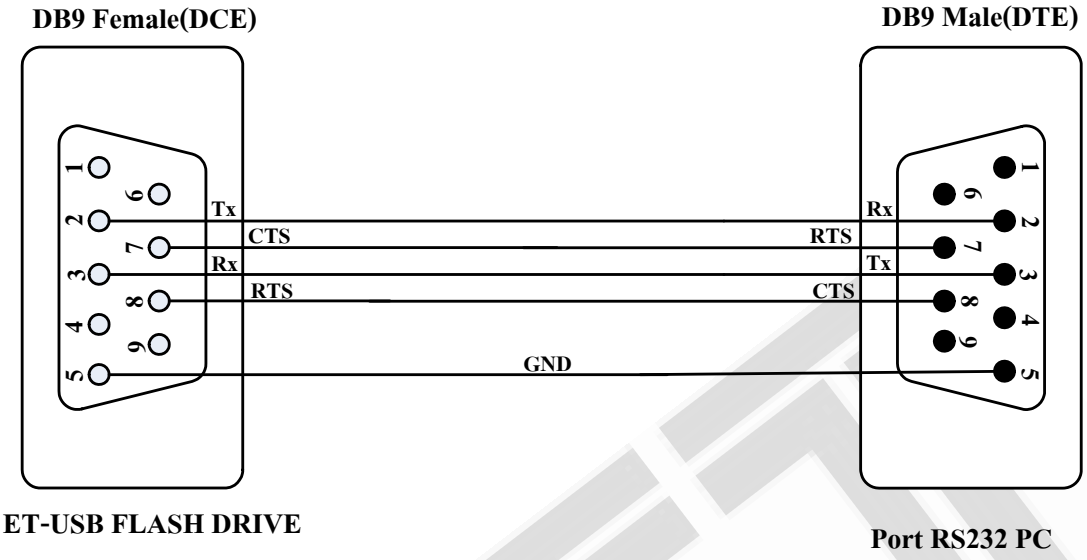
- Baud Rate จะต้องกำหนดเริ่มต้น default ไว้ที่ 9600 bit/s สามารถส่งคำสั่งเปลี่ยนแปลงได้ในภายหลัง
- 8 Data bit , 1 stop bit และ No parity
- Flow Control : ให้กำหนดที่ Hardware ซึ่งก็คือ RTS/CTS จะต้องถูก Enable เพื่อใช้เป็น handshake
- ในกรณีที่ไม่ต้องการใช้ Handshake ใช้เพียงขา Rx(ขา3) และ Tx(ขา2) และ กราวด์(ขา5) ในการติดต่อสื่อสารเท่านั้น ก็จะต้องทำการ Jump ขา RTS(ขา 7) และ CTS (ขา 8)ที่ Port DB9 ของ ET-USB FLASH DRIVE เข้าด้วยกัน จากนั้นก็ต่อขา Rx และ Tx ของ ET-USB FLASH DRIVE ไปยังขา Rx และ Tx ของอุปกรณ์ที่นำมาควบคุม โดยจะต้องต่อแบบไขว้ขากันคือ ต่อขา Rx เข้ากับ ขา Tx และต่อขา Tx เข้ากับ Rx ของอีกฝั่งหนึ่ง ส่วนกราวด์ให้ต่อเข้าด้วยกัน ดังแสดงในรูปที่4



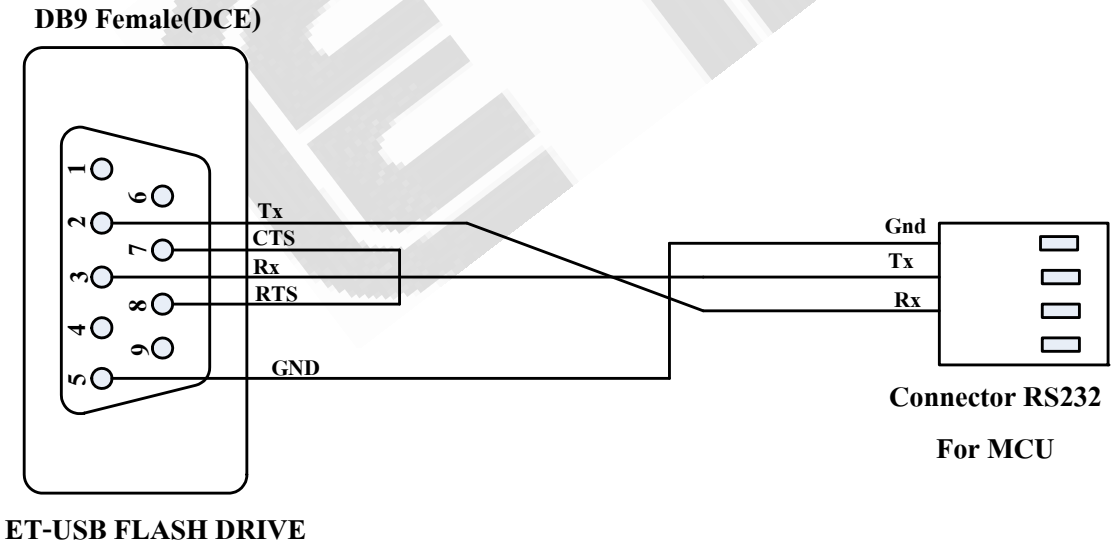
รูปที่1.ตำแหน่งขา DB9 female (DCE) ของ ET-USB FLASH DRIVE



รูปที่2. ตำแหน่งขา DB9 male(DTE) ของ PC



รูปที่3 แสดงการต่อสายสัญญาณ RS232 ระหว่าง ET-USB FLASH DRIVE กับ PC แบบใช้ Handshake (CTS,RTS)



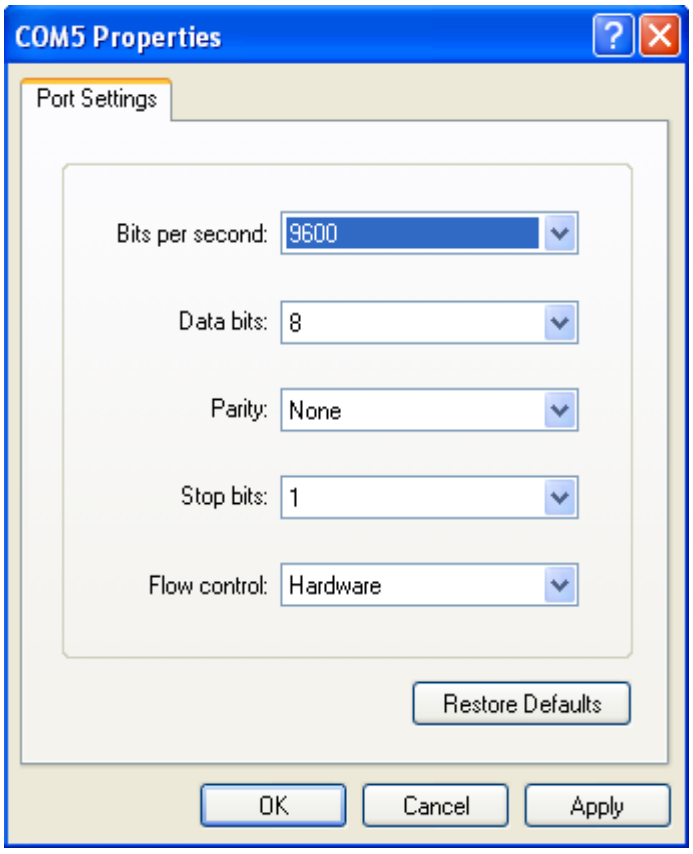
รูปที่4 แสดงการต่อสายสัญญาณ RS232 ระหว่าง ET-USB FLASH DRIVE
กับ MCU แบบไม่ใช้ Handshake (Jump ขา 7 CTS และขา8 RTS เข้าด้วยกัน)

- ขั้นตอนการทดสอบใช้งานกับ PC

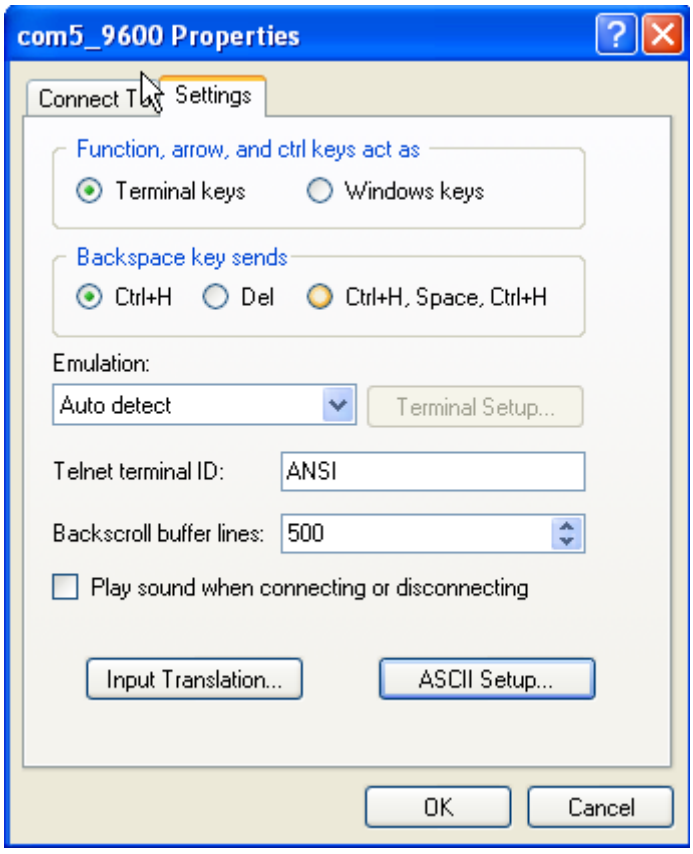
1.) ต่อ Flash Drive เข้าที่ขั้วต่อ USB ของ ชุด ET-USB FLASH DRIVE

2.) ต่อสาย RS232 จาก PC หรือ MCU เข้าที่ขั้วต่อ RS232 ของ ชุด ET-USB FLASH DRIVE

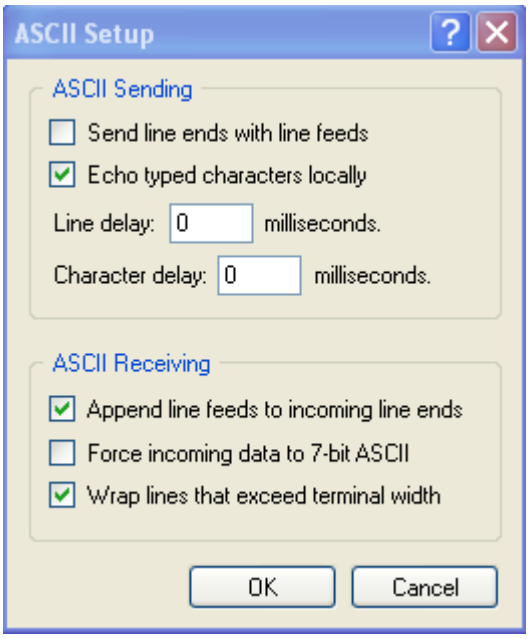
3.) ถ้าส่งคำสั่งผ่านทาง PC ให้เปิดโปรแกรม HyperTeminal หรือโปรแกรมที่ใช้รับส่งข้อมูลผ่านทาง RS232 ขึ้นมารอไว้ โดยให้ Set Up คุณสมบัติดังรูปที่5 จากนั้นให้กด Ok ก็จะได้หน้าจอ Hyperterminal ขึ้นมา จากนั้นให้คลิกที่ Icon Properties ( ) จะได้หน้าต่างออกมาดังในรูปที่ 6 ให้เลือกที่ TAB Setting และคลิกเลือกที่ปุ่ม ASCII Setup… จากนั้นก็จะได้หน้าต่างในรูปที่3ขึ้นมาให้ทำการกำหนดค่าตามในรูปที่ 7 จากนั้นให้กด Ok ในแต่ละหน้าต่างเป็นอันเรียบร้อยในการ Set การใช้งาน Hyperterminal



รูปที่ 5



รูปที่ 6



รูปที่ 7

4.) จ่ายไฟ 7-12 VDC ให้กับชุด ET-USB FLASH DRIVE ให้สังเกตที่ LED Status สีเขียวจะติดค้างนั่นแสดงว่าตัว ET-USB FLASH DRIVE ถูกต่อเข้ากับ FLASH DRIVE และ อุปกรณ์ที่ใช้สื่อสารทางด้าน RS232 เรียบร้อยพร้อมใช้งานแล้ว แต่ถ้า LED Status กระพริบสลับระหว่างสีเขียวและแดง แสดงว่า การเชื่อมต่อยังไม่สมบูรณ์ ในขณะที่มีการอ่านเขียนข้อมูล LED สีเขียวจะกระพริบ

5.) หลังจากจ่ายไฟเรียบร้อยแล้วให้รอจนกว่าจะมีข้อความ :

```
Ver 03.55VDAPF On-Line:
Device Detected P2
No Upgrade
D:\>
```

ปรากฏขึ้นที่หน้าต่าง HyperTerminal และแสดง D:\> พร้อมที่จะรับคำสั่งต่างๆจากผู้ใช้เพื่อติดต่อใช้งาน USB FLASH DRIVE เมื่อพิมพ์คำสั่งเสร็จหรือพิมพ์คำสั่งผิดให้กด Enter เพื่อเริ่มต้นคำสั่งใหม่จะสังเกต D:\> จะขึ้นเสมอแสดงความพร้อมในการรับคำสั่ง โดยค่า default ของ ET-USB Flash Drive จะถูกกำหนดไว้ดังนี้คือ Baud Rate ในการสื่อสาร 9600 bit/s , รับคำสั่งในโหมด Extended Mode และ กำหนดให้มีการรับค่าหรือส่งผ่านค่าที่เป็นตัวเลขในแบบ Binary Mode (IPH) จากนั้นให้ลองทำการทดสอบการเขียนและอ่านไฟล์ ตามตัวอย่างในหัวข้อด้านล่าง

- ตัวอย่างการทดสอบ ET-USB FLASH DRIVE

ทดสอบกับโปรแกรม Hyper Terminal

1) ทดสอบการเขียนข้อมูลให้กับไฟล์ชื่อ test01.txt จำนวน 10 Byte

- ส่งคำสั่ง IPA เพื่อกำหนดรูปแบบการผ่านค่า จำนวน Byte ข้อมูลที่จะเขียน ให้กับ Monitor ใน ASCII Mode
- ส่งคำสั่ง OPW test01.txt เพื่อเปิดไฟล์สำหรับเขียน
- ส่ง คำสั่ง WRF 10 เพื่อเขียนไฟล์โดยระบุจำนวน Byte ที่จะเขียน = 10 Byte แล้ว Enter
- ทำการเขียนไฟล์ 'abcdefghij' เมื่อครบ 10 Byte จะมี Response D:\> ส่งออกมาแสดงว่าเขียนข้อมูลครบแล้ว
- ส่งคำสั่ง CLF test01.txt เพื่อทำการปิดไฟล์ที่เขียน

D:\>	*[Prompt ใน Extended mode]*
IPA ↵	*[กำหนดรูปแบบการส่งจำนวน byte ที่จะเขียน ในแบบ ASCII Mode]*
D:\>	*[Response Prompt]*
OPW␣test01.txt ↵	*[ทำการ Open file ชื่อ test01.txt]*
D:\>	*[Response Prompt]*
WRF␣10 ↵	*[ส่งคำสั่งเขียน file โดยระบุจำนวน byte ที่จะเขียน 10 Byte]*
abcdefghij	*[เขียน data 10 Byte]*
D:\>	*[Response Prompt จะแสดงอัตโนมัติเมื่อเขียนข้อมูลตัวที่ 10 เรียบร้อย]*
CLF␣test01.txt↵	*[ส่งคำสั่งปิด test01.txt ที่ได้เปิดเขียนไว้]*
D:\>	*[Response Prompt สิ้นสุดการเขียน file]*

2) ทดสอบการอ่านข้อมูลจากไฟล์ชื่อ test01.txt ออกมาจำนวน 5 Byte ซึ่งค่าที่จะต้องอ่านได้คือ abcde

- ส่งคำสั่ง IPA เพื่อกำหนดรูปแบบการผ่านค่า จำนวน Byte ข้อมูลที่จะอ่าน ให้กับ Monitor ใน ASCII Mode
- ส่งคำสั่ง OPR test01.txt เพื่อเปิดไฟล์สำหรับอ่าน
- ส่ง คำสั่ง RDF 5 เพื่ออ่านไฟล์โดยระบุจำนวน Byte ที่จะอ่าน = 5 Byte แล้ว Enter
- ข้อมูลจะถูกอ่านออกมา 5 byte คือ abcde โดยข้อมูลที่อ่านได้นี้จะถูกนำด้วยค่า 0x0D ส่งออกมาก่อนแล้วถึงตามด้วยข้อมูล 5 byte และปิดด้วยท้ายข้อมูลด้วย D:\>

D:\>	*[Prompt ใน Extended mode]*
IPA ↵	*[กำหนดรูปแบบการส่งจำนวน byte ที่จะอ่าน ในแบบ ASCII Mode]*
D:\>	*[Response Prompt]*
OPR␣test01.txt ↵	*[ส่งคำสั่ง Open file ชื่อ test01.txt]*
D:\>	*[Response Prompt]*
RDF␣5 ↵	*[ส่งคำสั่งอ่าน file ที่เปิดอยู่ โดยระบุจำนวน byte ที่จะอ่าน 5 Byte]*
↵**abcde**D:\>	*[data ถูกอ่านออกมา 5 Byte โดยข้อมูลจะถูกนำด้วย 0x0D และปิดด้วย <prompt>]*
↵	*[ส่งคำสั่ง enter เพื่อรอรับคำสั่งต่อไป]*
D:\>	*[Response <prompt>]*

<u>คำสั่งสำหรับติดต่อกับ ET-USB FLASH DRIVE (Monitor Command)</u>

การติดต่อและการควบคุม ET-USB FLASH DRIVE จะใช้วิธีการส่ง Command Monitor ผ่านทางการสื่อสาร RS232 ซึ่ง Command Monitor นี้จะเป็นประโยชน์เมื่อใช้ไมโครคอนโทรลเลอร์ในการติดต่ออ่านเขียนข้อมูล สัญลักษณ์ Prompt(> หรือ D:\>) จะถูกส่งออกมาเมื่อ ET-USB FLASH DRIVE พร้อมที่จะรับคำสั่ง ข้อมูล Status และ Error จะถูกส่งแจ้งออกมายัง Monitor หลังจากที่กระทำคำสั่งนั้นๆสมบูรณ์แล้ว

1. ข้อกำหนดต่างๆในการติดต่อกับ ET-USB FLASH DRIVE ด้วย Command Monitor

1.1) Monitor Modes :

สำหรับ Command Monitor นี้ จะรองรับรูปแบบของคำสั่งอยู่ 2 โหมด คือแบบ Monitor-Extended และแบบ Monitor-Short ซึ่งนี่จะเป็นการกำหนดรูปแบบการแสดงของสัญลักษณ์ <prompt> และ รูปแบบของคำสั่งที่จะส่งเข้าไปติดต่อกับ Monitor , ใน Extended Mode เราจะส่งคำสั่งในลักษณะที่เป็นตัวอักษรซึ่งคำสั่งที่พิมพ์นั้นก็จะมีขนาดยาว โดยคำสั่งจะถูกเขียนอยู่ในรูป ASCII Code โหมดนี้จะถูกกำหนดให้เป็นค่า default , ส่วนใน Short Mode คำสั่งนั้นจะถูกกำหนดให้สั้นลง สำหรับใช้เขียนโปรแกรมควบคุมผ่านทาง MCU ดังนั้นคำสั่งก็จะถูกเขียนแทนด้วยค่า Binary(Hex)

<u>ข้อสังเกต</u> หลังจาก Reset ค่า default จะถูกกำหนดไว้ที่ Extended Mode

1.2) Numerical Modes :

ก็คือการกำหนดรูปแบบการรับและส่งค่าพารามิเตอร์ของแต่ละคำสั่งที่เป็นจำนวนตัวเลขให้กับมอนิเตอร์ ซึ่งตัว Monitor Command จะรับรู้ได้ 2 แบบ โดยแต่ละแบบจะเป็นอิสระต่อคำสั่งที่เราเลือกใช้งาน กล่าวคือในการส่งคำสั่งที่ต้องระบุค่าพารามิเตอร์ที่เป็นจำนวนตัวเลขลงไปด้วยนั้น ผู้ใช้สามารถเลือกใส่จำนวนตัวเลขในรูปแบบใดแบบหนึ่งเข้าไปก็ได้ ซึ่งขึ้นอยู่กับผู้ใช้ว่าได้กำหนดรูปแบบของจำนวนตัวเลขที่ใช้แทนลงไปว่าเป็นแบบใด , แบบที่1 ASCII Mode ซึ่งจะแทนจำนวนตัวเลขด้วยตัวอักษร ASCII (กำหนดด้วยคำสั่ง IPA) , แบบที่2 Binary Mode ซึ่งจะแทนจำนวนตัวเลขด้วยค่า Binary Hex (กำหนดด้วยคำสั่ง IPH)

ในการแทนค่าด้วย Binary Hex นั้น จำนวน Byte ที่ใช้แทนลงไปจะถูกกำหนดโดยคำสั่งนั้นๆซึ่งจะไม่เท่ากันให้ดูในรายละเอียดของคำสั่งอีกครั้งหนึ่ง , ส่วนในการแทนค่าจำนวนตัวเลขด้วย ASCII นั้น สามารถแทนค่าตัวเลขได้ทั้งแบบเลขฐาน10 และเลขฐาน16 ในรูปของ ASCII , สมมุติว่าเลขที่จะส่งเป็นฐาน10 ถ้าต้องการส่งด้วยฐาน16 ผู้ใช้จะต้องทำการแปลงเลขฐาน10นั้นให้เป็นเลขฐาน16ก่อน แล้วถึงนำค่าเลขฐาน16ที่แปลงได้ส่งออกไปซึ่งจะต้องนำหน้าด้วยตัวอักษร ‘$’ หรือ ‘0x’ อย่างใดอย่างหนึ่งเสมอเพื่อบอกให้ Monitor Command ทราบว่าเป็นค่าตัวเลขฐาน16 สำหรับค่า decimal ในรูปของ ASCII นั้นจะสามารถพิมพ์ส่งออกไปได้เลย โดยพิมพ์ได้สูงสุด 8 ตัวอักษร หรือ 8 หลักนั่นคือเราสามารถแทนด้วยเลขฐาน10ได้ถึง 99,999,999 หรือเขียนแทนในรูปเลขฐาน16ได้สูงสุดเท่ากับ 0x5F5E0FF

<u>ข้อสังเกต</u> *หลังจาก Reset ค่า default จะถูกกำหนดไว้ที่ Binary Mode ซึ่งใน Binary Mode นี้ค่าจำนวนตัวเลขที่ส่งเข้าไปจะเริ่มจาก MSB เป็น byte แรก ในขณะที่ค่า Output ที่เป็นจำนวนตัวเลข ที่ถูกส่งออกมาจาก Monitorให้กับผู้ใช้จะส่ง byte LSB ออกมาเป็น byte แรกไม่ว่าจะกำหนดอยู่ในรูป Binary Mode หรือ ASCII Mode ก็ตาม*

1.2.1) Number Input: ก็คือค่าพารามิเตอร์ที่เป็นจำนวนตัวเลข ที่ต้องส่งผ่านไปยัง Monitor จะแสดงดังตัวอย่างต่อไปนี้ สมมุติกำหนดการส่งค่าจำนวนตัวเลขในแบบ ASCII Mode(กำหนดด้วยคำสั่ง IPA) โดยส่งคำสั่งแบบ Extended Command (กำหนดด้วยคำสั่ง ECS) จะเป็นดังนี้

เมื่อสัญลักษณ์ ↵ = Enter (0x0D) , ␣ = Space (0x20) 1 ช่องว่าง

D:\> [prompt ใน Extended Mode]

IPA ↵ [คำสั่งกำหนดการส่งจำนวนตัวเลขแบบ ASCII Mode]

D:\>

SBD␣$384100 ↵ [คำสั่ง Set Baud Rate = 9600 ตัวเลขที่ใส่จะเป็นค่า Hex ในรูป ASCII] หรือ

SBD␣0x384100 ↵ [คำสั่ง Set Baud Rate = 9600 ตัวเลขที่ใส่จะเป็นค่า Hex ในรูป ASCII] หรือ

SBD␣3686656 ↵ [คำสั่ง Set Baud Rate = 9600 ตัวเลขที่ใส่จะเป็นค่า Decimal ในรูป ASCII] หรือ

SBD␣03686656 ↵ [คำสั่ง Set Baud Rate = 9600 ตัวเลขที่ใส่จะเป็นค่า Decimal ในรูป ASCII]

D:\>

ในคำสั่ง Set Baud Rate 2 คำสั่งแรกจะส่งตัวเลขเป็นค่า Hex ในรูป ASCII โดยจะถูกนำด้วยสัญลักษณ์ '$' และ '0x' และ 2 คำสั่งสุดท้าย ค่าตัวเลข 0x384100 จะต้อง Convert ให้อยู่ในรูปเลขฐาน10 แล้วก็นำค่าที่ได้ส่งออกไปได้เลย ส่วน 0 ที่ใส่ไปข้างหน้านั้นจะไม่มีผลใดๆ

ในตัวอย่างนี้ถ้าเราจะส่งจำนวนตัวเลขในรูปแบบของ Binary mode(กำหนดด้วยคำสั่งIPH) โดยใช้ Short Command (กำหนดด้วยคำสั่ง SCS) เราก็จะต้องส่งค่า Hex Code ออกไปดังนี้ 14 20 38 41 00 0D โดยจะต้องส่ง Byte MSB ออกไปเป็น Byte แรก

1.2.2) Number Output: ก็คือค่าของจำนวนตัวเลขที่ถูกส่งออกมาจาก Monitor ให้กับผู้ใช้ ซึ่งก็จะส่งออกมาในรูปแบบของเลขฐาน16 ส่วนจะเป็นเลขฐาน16 ในรูปของ ASCII หรือ Binary นั้นก็ขึ้นอยู่กับการกำหนดของผู้ใช้จากคำสั่ง IPA(ASCII) หรือ IPH(Binary) โดย Monitor จะส่งค่าจำนวนตัวเลขที่เป็น Byte LSB ออกมาเป็น Byte แรก เช่นค่าตัวเลข 0x000310aa เมื่ออยู่ใน ASCII Mode Monitor ก็จะส่งค่าออกมาดังนี้

$AA␣$10␣$03␣$00␣↵ [สังเกตว่าหลังจาก byte สุดท้ายถูกส่งออกมา จะตามด้วย space 1ช่อง ก่อนที่จะปิดด้วย Carry return (0x0D)]

ถ้าอยู่ใน Binary Mode ค่าที่ถูกส่งออกมาจะเป็นดังนี้

AA 10 03 00 0D [ค่าจะถูกส่งออกมาติดกันและจบด้วยค่า Carry return(0x0D)]

<u>ข้อสังเกต</u> *ในการส่งค่าจำนวนตัวเลขให้กับ Monitor จะต้องส่ง Byte MSB เป็น Byte แรก ส่วนในการรับค่าจำนวนตัวเลขจาก Monitor ตัวMonitor จะส่ง Byte LSB ออกมาให้เป็น Byte แรก*

1.3) Filename :

ชื่อไฟล์ที่สร้างขึ้นมาใช้กับ ET-USB FLASH DRIVE จะอยู่ในรูปแบบ 8.3 โดยชื่อไฟล์ต้องขึ้นต้นด้วยตัวอักษรหรือตัวเลข หรือตัวอักษรต่อไปนี้ตัวใดตัวตัวหนึ่งคือ $ % ‘ - _ @ ~ ! ( ) { } ^ # & ในระบบ File FAT32 จะไม่รองรับการตั้งชื่อไฟล์แบบยาว นั่นคือชื่อไฟล์จะแสดงให้เห็นแค่ 8 ตัวอักษรและนามสกุลอีก 3 ตัวอักษร

1.4) Monitor Startup :

เมื่อโมดูล ET-USB FLASH DRIVE เริ่มต้นใช้งาน ในส่วนของ Monitor Command ก็จะส่งข้อความออกมาดังนี้

Ver 03.55VDAPF On-Line : ↵

นี่จะเป็นการแสดงเวอร์ชั่นของ firmware (03.55) และชนิดของ firmware(VDAPF)

1.5) โครงสร้างคำสั่ง :

สำหรับโครงสร้างคำสั่งในการติดต่อสื่อสารกับ Monitor ของ ET-USB FLASH DRIVE ผ่านทาง RS232 นั้นจะมีโครงดังนี้

แบบที่1 CMM ↵

แบบที่2 CMM␣ *parameter*1 ↵

แบบที่3 CMM␣ *parameter1*␣ *parameter2* ↵

เมื่อ ↵ หมายถึง Enter (0x0D)

␣ หมายถึง Space (0x20) 1 ช่องว่าง

CMM หมายถึง คำสั่ง

Parameter หมายถึง ค่าที่ต้องส่งไปให้ Monitor พร้อมกับคำสั่ง อาจเป็นชื่อไฟล์ หรือค่าจำนวนตัวเลข

จากโครงสร้างคำสั่งนี้ CMM ก็คือส่วนที่เป็นคำสั่ง ซึ่งในส่วนนี้สามารถกำหนดได้ 2 รูปแบบ คือ

1. Extended Mode ซึ่งจะเลือกด้วยใช้คำสั่ง ECS คำสั่งจะอยู่ในรูปของตัวอักษร ASCII
2. Short Mode ซึ่งจะเลือกโดยใช้คำสั่ง SCS คำสั่งจะอยู่ในรูปของ Hex Code

ถ้าคำสั่งไหนมีการกำหนดค่าพารามิเตอร์จะต้องเว้นช่องว่าง 1 ช่องระหว่างคำสั่งและค่าพารามิเตอร์ ถ้ามีค่าพารามิเตอร์ 2 ชุดก็ต้องเว้นช่องว่างระหว่างค่าพารามิเตอร์ชุดที่ 1 และ 2 ด้วย เหมือนในแบบที่ 3 และจะต้องจบคำสั่งนั้นๆด้วย Enter หรือ 0x0D เสมอคำสั่งถึงจะถูกส่งออกไป

สำหรับค่าพารามิเตอร์นั้นเราจะมีการแทนค่าอยู่ 2 แบบ คือ

1. การแทนค่าพารามิเตอร์ที่เป็นชื่อไฟล์ - ผู้ใช้สามารถแทนค่าพารามิเตอร์ด้วยตัวอักษร ASCII ได้เลย
2. การแทนค่าพารามิเตอร์ที่เป็นจำนวนตัวเลข – ผู้ใช้สามารถแทนค่าได้ 2 แบบ คือ
   - ASCII Mode การแทนค่าในโหมดนี้จะต้อง Set โดยใช้คำสั่ง IPA ก่อน
   - Binary Mode การแทนค่าในโหมดนี้จะต้อง Set โดยใช้คำสั่ง IPH ก่อน(ถูกกำหนดเป็นค่า default)

ผู้ใช้สามารถดูรูปแบบการแทนค่าทั้ง 2 Mode ได้ในหัวข้อที่ 1.2 และในตัวอย่างของแต่ละคำสั่ง ดังนั้นในการแทนค่าที่เป็นจำนวนตัวเลข ผู้ใช้จะต้องคำนึงเสมอว่าได้กำหนดให้ Monitor ทำงานอยู่ใน Mode ใด

1.6) Command Responses :

โดยปกติแล้วหลังจากที่กระทำคำสั่งเสร็จแล้ว Monitor Command ก็จะส่ง <prompt> หรือ Error ตอบกลับมาให้ผู้ใช้เพื่อบอกให้ทราบว่าพร้อมที่จะรับคำสั่งต่อไป ทุกครั้งที่ Monitor ส่ง <prompt> หรือ Error ตอบกลับมาแล้วก็จะส่ง 0x0D ตามมาปิดท้ายเสมอ

1.*6.1) Successful Command Prompt :* รูปแบบของ Command Prompt ที่จะส่งออกมาเมื่อกระทำคำสั่งนั้นๆ สมบูรณ์แล้วมีอยู่ด้วยกัน 2 แบบ โดยจะแสดงดังตารางด้านล่าง

Extended Command Mode	Short Command Mode
D:\> ↵	> ↵ (3E 0D)h

1.6.2) Empty Command Prompt : คือรูปแบบของ Command ที่ส่งออกมาเพื่อบอกสถานะของ Flash Drive (Disk Drive เก็บข้อมูล) ว่าถูกต่ออยู่หรือไม่ โดยจะส่งออกมาหลังจากมีการ Reset หรือเมื่อผู้ใช้ส่งคำสั่งใดๆออกไปยัง Monitor ซึ่งจะแสดงดังตารางด้านล่าง

Status Disk Drive	Extended Command Mode	Short Command Mode
Flash Drive ถูกต่อ	D:\> ↵	> ↵ (3E 0D)h
Flash Drive ไม่ถูกต่อ	No Disk ↵	ND ↵ (4E 44 0D)h

1.6.3) Error Messages : ถ้ามีการส่งคำสั่งที่ผิดรูปแบบตามที่กำหนดไว้เข้ามา ตัว Monitor ก็จะมีการส่ง Bad Command error ออกไปให้ผู้ใช้ได้รับทราบ และพร้อมที่จะรับคำสั่งใหม่อีกครั้งหนึ่งเมื่อแจ้ง Bad Command error เสร็จสิ้น ซึ่งรูปแบบของ Bad Command Error จะแสดงดังตารางด้านล่างตามโหมดคำสั่งที่เลือกใช้

Extended Command Mode	Short Command Mode	ความหมาย
Bad␣Command ↵	BC ↵ (42 43 0D)h	ตัว Monitor ไม่รู้จักคำสั่งนี้
Command␣Failed ↵	CF ↵ (43 46 0D)h	ไม่พบชื่อ File หรือชื่อ directory ที่เลือก
Disk␣Full ↵	DF ↵ (44 46 0D)h	ไม่มีพื้นที่ว่างบนดิสก์
Invalid ↵	FI ↵ (46 49 0D)h	การเปิดหรือการเปลี่ยน File สำหรับอ่าน เขียน เป็นโมฆะ หรือ ไม่สำเร็จ

Read ␣Only ↵	RO↵ (52 4F 0D)h	บอกให้ทราบว่า File เปิดอ่านได้เท่านั้น เพราะมีการ Lock การเขียนไว้
File␣Open ↵	FO↵ (46 4F 0D)h	บอกให้ทราบว่ามีไฟล์ถูกเปิดเพื่อเขียนอยู่ ต้องปิดก่อนที่จะทำคำสั่งต่อไป
Dir␣Not␣Empty ↵	NE↵ (4E 45 0D)h	บอกให้ทราบว่า Directory ที่จะลบนั้นไม่ ว่าง ยังมีไฟล์อื่นอยู่ภายใน

2. **Monitor Command Set**

ในหัวข้อนี้จะกล่าวถึงคำสั่ง MONITOR ที่ใช้ติดต่อกับ ET-USB FLASH DRIVE ทั้งแบบ Short Command Mode และแบบ Extended Command Mode โดยคำสั่งในแบบ Extended Command สามารถแทนคำสั่งด้วยตัวอักษรพิมพ์ใหญ่หรือพิมพ์เล็กก็ได้ ความหมายของสัญลักษณ์ที่ใช้แทนในตารางคำสั่ง มีดังนี้

file หมายถึง ชื่อ Directory หรือ ชื่อ File และ นามสกุล File ในรูปแบบ 8.3 ซึ่งต้องขึ้นต้นด้วย ตัวอักษร หรือตัวเลข รวมถึงอักษรต่อไปนี้ด้วย $ % ‘ - _ @ ~ ! ( ) { } ^ # & ก็ได้

data หมายถึง ค่าตัวเลขขนาด16 bit ที่ใช้แทนในส่วนของ FAT file ในรูปแบบของ วัน

datetime หมายถึง ค่าตัวเลขขนาด 32 bit ที่ใช้แทนในส่วนของ FAT file ในรูปแบบของ วัน และ เวลา

divisor หมายถึง ค่าข้อมูลขนาด 3 Byte ที่ใช้แทนค่าของ Baud Rate ที่ถูกเข้ารหัสมา

qword หมายถึง ค่าข้อมูลขนาด 64 Bit(8 byte)

dword หมายถึง ค่าข้อมูลขนาด 32 bit(4 byte)

word หมายถึง ค่าข้อมูลขนาด 16 bit(2 byte)

byte หมายถึง ค่าข้อมูลขนาด 8 bit(1 byte)

data หมายถึง ข้อมูลที่อ่านได้จากไฟล์ใน Disk Drive โดยไม่มีการปรับปรุงแก้ไขใดๆจาก Monitor ซึ่งข้อมูลนี้จะเป็นข้อมูลจริงๆของไฟล์ที่ทำการอ่านออกมา

<prompt> หมายถึง เครื่องหมาย prompt แสดงในรูปของ ASCII ได้แก่ ‘D:\>’ หรือ ‘>’

↵ หมายถึง Enter (0x0D,\r)

␣ หมายถึง Space (0x20) 1 ช่องว่าง

2.1) คำสั่งในกลุ่มที่เกี่ยวกับ Configuration

สำหรับคำสั่งกลุ่มนี้จะเป็นคำสั่งที่เกี่ยวข้องกับการ Setup Monitor และตรวจสอบ Version ของ Firmware ที่ใช้กับ ET-USB FLASH DRIVE ซึ่งคำสั่งต่างๆจะแสดงใน ตารางที่ 2.1

2.1.1) Short Command Set (SCS) : คำสั่งนี้จะเป็นการเลือกใช้ Command Monitor Mode แบบ Short Command Code ซึ่งการส่งคำสั่งนี้ไปยัง Monitor สามารถส่งได้ทั้งแบบ Short หรือ Extended Command Mode ไม่ว่าขณะนั้นจะทำงานอยู่ใน Mode ใดก็ตาม

Ex. ส่งคำสั่ง SCS ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[prompt เมื่ออยู่ใน Extended Mode]*

SCS ↵

> *[Response Prompt เปลี่ยนเป็น Short Mode]*

Ex. ส่งคำสั่ง SCS ด้วย MCU (แบบ Extended Mode)

```
printf("scs\r") ;          // \r = 0x0D
```

Ex. ส่งคำสั่ง SCS ด้วย MCU (แบบ Short Mode)

```
char  num = 0x10   ;
printf("%c\r",num)   ;   // \r = 0x0D
```

ตารางที่ 2.1 Monitor Configuration Command

Extended Command Set (ASSCII Code)	Short Command Set (Hexadecimal Code)	Function	Response
SCS ↵	10 0D	เลือกใช้คำสั่งในรูปแบบ Short (Hex)	'>'0x0D
ECS ↵	11 0D	เลือกใช้คำสั่งในรูปแบบ Extended (ASCII)	'D:\>'0x0D
IPA ↵	90 0D	ให้ Monitor รับการส่งผ่านค่าตัวเลขแบบ ASCII	<prompt>0x0D
IPH ↵	91 0D	ให้ Monitor รับการส่งผ่านค่าตัวเลขแบบ Binary(Hex)	<prompt>0x0D
SBD␣*divisor* ↵	14 20 *divisor* 0D	Set Baud Rate	<prompt>0x0D
FWV ↵	13 0D	แสดง Version firmware	ดูหัวข้อ 2.1.6
E ↵	45 0D	Echo 'E' สำหรับ Sync การรับส่ง	'E'0x0D
e ↵	65 0D	Echo 'e' สำหรับ Sync การรับส่ง	'e'0x0D
↵ (Enter=0x0D)	0x0D	ตรวจสอบสถานการต่อ Flash Drive และใช้เมื่อจบคำสั่งหรือจบ Response ต่างๆ	Prompt หรือ No Disk , ND

2.1.2) Extended Command Set (ECS) : คำสั่งนี้จะเป็นการเลือกใช้ Command Monitor Mode แบบ Extended Command Code ซึ่งการส่งคำสั่งนี้ไปยัง Monitor สามารถส่งได้ทั้งแบบ Short หรือ Extended Command Mode ไม่ว่าขณะนั้นจะทำงานอยู่ใน Mode ใดก็ตาม

Ex. ส่งคำสั่ง ECS ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

> *[prompt เมื่ออยู่ใน Short Mode]*

ECS ↵

D:\> *[Response Prompt เปลี่ยนเป็นแบบ Extended Mode]*

Ex. ส่งคำสั่ง ECS ด้วย MCU (แบบ Extended Mode)

```
printf("ecs\r") ;          // \r = 0x0D
```

Ex. ส่งคำสั่ง ECS ด้วย MCU (แบบ Short Mode)

```
char  num = 0x11   ;
printf("%c\r",num)   ;   // \r = 0x0D
```

2.1.3) Monitor Mode ASCII (IPA) : คำสั่งนี้จะเป็นการกำหนดให้ Monitor รับค่า Input ที่เป็นจำนวนตัวเลข หรือแสดงค่า Output ที่เป็นจำนวนตัวเลข ออกมาในรูปแบบของตัวอักษร ASCII ซึ่งการส่งคำสั่งนี้จะส่งแบบ Short หรือแบบ Extended Command Mode นั้นขึ้นอยู่กับว่า ผู้ใช้กำหนดให้ Monitor รับคำสั่งในรูปแบบใดไว้

Ex. ส่งคำสั่ง IPA ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

```
D:\>                [ prompt เมื่ออยู่ใน Extended Mode]
IPA  ↵
D:\>                [ prompt แสดงการทำคำสั่งเรียบร้อย]
```

Ex. ส่งคำสั่ง IPA ด้วย MCU (ส่งแบบ Extended Mode)

```
printf("ipa\r") ;          // \r = 0x0D
```

Ex. ส่งคำสั่ง IPA ด้วย MCU (ส่งแบบ Short Mode)

```
char  num = 0x90   ;
printf("%c\r",num)   ;  // \r = 0x0D
```

หลังจากส่งคำสั่งนี้ออกไป คำสั่งที่กระทำต่อจากนี้ที่มีการส่งผ่านค่า จะต้องส่งผ่านค่าในรูปแบบของตัวอักษร ASCII ซึ่งสามารถส่งในรูปของเลขฐาน10 แบบ ASCII หรือในรูปเลขฐาน16แบบ ASCII ก็ได้โดยถ้าส่งแบบเลขฐาน 16 จะต้องขึ้นต้นด้วย ASCII '$' หรือ ASCII '0x' เช่น ต้องการส่งผ่านค่า 25 ให้ Monitor ค่า25ที่กำหนดในคำสั่งจะเป็นดังนี้

ค่า 25 ฐาน10 = กำหนดในรูปแบบของ ASCII ฐาน10 จะได้ ' **25** ' (Code ASCII=0x32,0x35)

ค่า 25 ฐาน10 = กำหนดในรูปแบบของ ASCII ฐาน 16 จะได้ **'$19**' (Code ASCII=0x24,0x31,0x39) หรือ

'**0x19**'(Code ASCII=0x30,0x78,0x31,0x39)

2.1.4) Monitor Mode Binary (IPH) : คำสั่งนี้จะเป็นการกำหนดให้ Monitor รับค่า Input ที่เป็นจำนวนตัวเลข หรือแสดงค่า Output ที่เป็นจำนวนตัวเลขออกมาในรูปแบบของ Binary (Hex Code) ซึ่งการส่งคำสั่งนี้จะส่งแบบ Short หรือแบบ Extended Command Mode นั้นขึ้นอยู่กับว่า ผู้ใช้กำหนดให้ Monitor รับคำสั่งในรูปแบบใดไว้

Ex. ส่งคำสั่ง IPH ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

```
D:\>                [prompt เมื่ออยู่ใน Extended Mode]
IPH  ↵
D:\>                [Prompt แสดงการทำคำสั่งเรียบร้อย ]
```

Ex. ส่งคำสั่ง IPH ด้วย MCU (แบบ Extended Mode)

```
printf("iph\r") ;          // \r = 0x0D
```

Ex. ส่งคำสั่ง IPH ด้วย MCU (แบบ Short Mode)

```
char  num = 0x91   ;
printf("%c\r",num)   ;   // \r = 0x0D
```

หลังจากส่งคำสั่งนี้ออกไป คำสั่งที่กระทำต่อจากนี้ที่มีการส่งผ่านค่า จะต้องส่งผ่านค่าในรูปแบบของ Binary (Hex Code) ซึ่งก็คือค่าที่ส่งผ่านจะต้องอยู่ในรูปของเลขฐาน16 จริงๆ และต้องส่งให้ครบตามจำนวน Byte ที่กำหนดไว้ในคำสั่งนั้นๆด้วย เช่น ต้องการส่งผ่านค่า 25 ให้ Monitor ค่า25ที่กำหนดในคำสั่งจะเป็นดังนี้

ค่า 25 ฐาน10 = กำหนดในรูปแบบของ ฐาน16 จะได้ 0x19

ตัวอย่าง การส่งผ่านค่าเมื่อใช้คำสั่ง IPA และ IPH สมมุติว่าต้องการอ่าน File ชื่อ test.txt ออกมา 18 byte ทำได้ดังนี้

- เมื่อเลือกรูปแบบการส่ง Command Monitor แบบ Extended Mode (กำหนดด้วยคำสั่ง ECS)

Ex. ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal (ส่งผ่านค่าแบบ IPA)

```
D:\>
OPR␣test.txt ↵                    [ส่งคำสั่ง Open File test.txt เพื่ออ่าน]
D:\>                              [Prompt แสดงการทำคำสั่งเรียบร้อย ]
RDF␣18 ↵   หรือ  RDF␣$12  ↵ หรื อ RDF␣0x12 ↵        [ส่งคำสั่งอ่านไฟล์ 18 Byte]
↵1234567890abcdefgh D:\>          [Display Data 18 Byte และ Prompt เมื่ออ่านครบ 18 byte]
```

Ex. ส่งคำสั่งด้วย MCU (ส่งผ่านค่าแบบ IPA)

```
printf("opr  test.txt\r") ;                  // Open file
printf("rdf  18\r")       ; หรือ  printf("rdf  $12\r") ; หรือ  printf("rdf  0x12\r") // Read File 18 Byte
```

Ex. ส่งคำสั่งด้วย MCU (ส่งผ่านค่าแบบ IPH)

```
char  byte1=0x00,byte2=0x00,byte3=0x00,byte4=0x12 ;
printf("opr  test.txt\r")  ;                  // Open File test.txt
printf("rdf  %c%c %c%c\r",byte1,byte2,byte3,byte4)  ;     //ส่งคำสั่ง Read File 18 Byte
```

ในตัวอย่างสุดท้ายนี้ เนื่องจากรูปแบบคำสั่งของ RDF ได้กำหนดการส่งผ่านค่าไว้ 4 Byte ดังนั้นเมื่อเราส่งผ่านค่าใน Mode IPH เราก็จะต้องส่งผ่านค่าให้ครบ 4 Byte ด้วย (ส่ง Byte MSB เป็น Byte แรก) ซึ่งจะต่างกับใน Mode IPA เราสามารถส่งค่าที่เป็น ASCII ออกไปได้เลยโดยไม่ต้องคำนึงถึงจำนวน Byte ที่คำสั่งกำหนดไว้ พูดง่ายๆก็คือ เมื่อใช้ Mode IPH เวลาส่งผ่านค่าให้กับ Monitor จะต้องส่งค่าให้ครบตามจำนวน Byte ที่ได้กำหนดไว้ในคำสั่งนั้นๆเสมอ ถ้าค่าที่ส่งมีจำนวน Byte น้อยกว่า ที่คำสั่งนั้นๆกำหนดไว้ก็ให้ใส่ 0 ไปด้านหน้าให้ครบตามจำนวน Byte ที่กำหนด

- เมื่อเลือกรูปแบบการส่ง Command Monitor แบบ Short Mode(กำหนดด้วยคำสั่ง SCS)

Ex. ส่งคำสั่งด้วย MCU (ส่งผ่านค่าแบบ IPA)

```
char  cm1=0x0E,cm2=0x20,cm3=0x0B,cm4=0x20 ;
printf("%c%ctest.txt\r",cm1,cm2) ;                          // Open File
printf("%c%c18",cm3,cm4) ; หรือ  printf("%c%c0x12",cm3,cm4) ;    //ส่งคำสั่ง Read File
```

Ex. ส่งคำสั่งด้วย MCU (ส่งผ่านค่าแบบ IPH)

```
char  char  cm1=0x0E,cm2=0x20,cm3=0x0B,cm4=0x20 ,byte1=0,byte2=0,byte3=0,byte4=0x12  ;
printf("%c%ctest.txt\r",cm1,cm2)   ;                        // Open file
printf("%c%c%c%c%c%c",cm3,cm4,byte1,byte2,byte3,byte4) ;  //ส่งคำสั่ง Read File
```

2.1.5) Set Baud Rate (SBD) : คำสั่งนี้จะใช้ในการ Set Baud Rate สำหรับการติดต่อทาง Serial Port RS232 โดยค่า default จะถูกกำหนดเริ่มต้นไว้ที่ 9600 ในการใช้คำสั่งนี้จะต้องมีการส่งผ่านค่าพารามิเตอร์ Encoder Baud Rate ขนาด 3 Byte ให้กับ Monitor หลังจากทำการส่งคำสั่งไปแล้ว Prompt จะถูกส่งออกมาก่อนที่ Baud Rate จะถูกเปลี่ยน จากนั้นเมื่อจะส่งคำสั่งต่อไปผู้ใช้จะต้อง Set Baud Rate ของอุปกรณ์ที่ใช้ส่งคำสั่งใหม่เสียก่อนถึงจะส่งคำสั่งต่อไปได้ถูกต้อง สำหรับค่า Encoder Baud Rate จะแสดงในตารางที่2.1.5

ตารางที่2.1.5 Encoder Baud Rate

Baud Rate	Encoder Baud Rate (Decimal Code)	Encoder Baud Rate (Hex Code)		
		Byte ที่1(MSB)	Byte ที่2	Byte ที่3
300	1056768	0x10	0x20	0x00
600	8917760	0x88	0x13	0x00
1200	12847360	0xC4	0x09	0x00
2400	14812160	0xE2	0x04	0x00
4800	7406080	0x71	0x02	0x00
9600	3686656	0x38	0x41	0x00
19200	10256384	0x9C	0x80	0x00
38400	5160960	0x4E	0xC0	0x00
57600	3457024	0x34	0xC0	0x00
115200	1703936	0x1A	0x00	0x00

จากตาราง ถ้าผู้ใช้เลือก Mode การส่งผ่านค่า แบบ ASCII (IPA) ก็สามารถเลือกค่าที่จะส่งได้ทั้งช่อง Decimal Code หรือ Hex Code ก็ได้ โดยถ้าเลือกในช่อง Hex Code ให้นำหน้า Byte ที่1 ด้วย 0x หรือ $ เพียงครั้งเดียวส่วน Byte อื่นเอาตัวเลขมาต่อได้เลย เช่น เลือก Baud Rate =57600 ค่าที่แทนในคำสั่งก็คือ '0x34C000' เป็นต้น ถ้าส่งด้วยค่า Decimal Code ค่าที่จะส่งคือ '3457024' เป็นต้น

ถ้าผู้ใช้เลือก Mode การส่งผ่านค่า แบบ Binary (IPH) ก็จะต้องใช้ค่าในช่อง Hex Code เท่านั้นและส่งออกไปทีละ Byte ตามในตาราง

- เมื่อเลือกรูปแบบการส่ง Command Monitor แบบ Extended Mode (กำหนดด้วยคำสั่ง ECS)

Ex. ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal (ส่งผ่านค่าแบบ IPA)

```
D:\>
SBD␣3457024 ↵        [Set Baud Rate 57600 ส่งแบบ ASCII ฐาน10]   หรือ
SBD␣0x34C000 ↵       [Set Baud Rate 57600 ส่งแบบ ASCII ฐาน16]   หรือ
SBD␣$34C000 ↵
D:\>
```

Ex. ส่งคำสั่งด้วย MCU (ส่งผ่านค่าแบบ IPA)

```
printf("sbd 3457024\r")  ;   // Set Baud Rate 57600      หรือ
printf("sbd 0x34C000\r") ;   // Set Baud Rate 57600      หรือ
printf("sbd $34C000\r")  ;   // Set Baud Rate 57600
```

Ex. ส่งคำสั่งด้วย MCU (ส่งผ่านค่าแบบ IPH)

```
char byte1 = 0x34 , byte2 = 0xC0 , byte3 = 0x00 ;
printf("sbd %c%c %c\r",byte1,byte2,byte3)   ;        // Set Baud Rate 57600
```

- เมื่อเลือกรูปแบบการส่ง Command Monitor แบบ Short Mode (กำหนดด้วยคำสั่ง SCS)

Ex. ส่งคำสั่งด้วย MCU (ส่งผ่านค่าแบบ IPA)

```
char cm1=0x14 , cm2=0x20 ;
printf("%c%3457024\r",cm1,cm2)  ;  // Set Baud Rate 57600  หรือ
printf("%c%0x34C000\r",cm1,cm2) ;  // Set Baud Rate 57600
```

Ex. ส่งคำสั่งด้วย MCU (ส่งผ่านค่าแบบ IPH)

```
char cm1=0x14 , cm2=0x20 , byte1=0x34 , byte2=0xC0 , byte3=0x00 ;
printf("%c%c%c%c%c",cm1,cm2,byte1,byte2,byte3) ; //Set Baud rate 57600
```

2.1.6) Firmware Version (FWV) : คำสั่งนี้จะใช้สำหรับดู Version ของ Firmware เมื่อส่งคำสั่งนี้ไปยัง Monitor จะได้ Output ดังตัวอย่าง

Ex. ส่งคำสั่ง FWV ด้วย Keyboard ผ่านโปรแกรม HyperTerminal (Extended Mode)

```
D:\>
FWV↵
↵                    [Respond 0x0D]
```

MAIN 03.55VDAPF↵ *[Respond Firm ware และ 0x0D ปิดท้าย]*

RPRG 1.00R ↵ *[Respond Firm ware และ 0x0D ปิดท้าย]*

D:\> *[Response <prompt>]*

2.1.7) Echo (E , e) : คำสั่งนี้จะใช้สำหรับ synchronization ระหว่าง Monitor กับ ผู้ใช้ เพื่อให้ผู้ใช้ตรวจสอบความพร้อมของ Monitor ในการรับคำสั่งต่อไป ซึ่งคำสั่งนี้สามารถส่งได้ทั้งแบบ Short หรือ Extended Command Mode ไม่ว่าขณะนั้น Monitor จะถูกกำหนดให้ทำงานอยู่ใน Mode ใดก็ตาม เมื่อส่งคำสั่งนี้ไปยัง Monitor ด้วยตัว E หรือ e ตัว Monitor ก็จะส่ง Echo 'E' หรือ 'e' กลับมาให้ผู้ใช้ตามที่ส่งไปให้

Ex. ส่งคำสั่ง E ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\>

E ↵ *[ส่งคำสั่ง E]*

E ↵ *[Respond 'E'ตามด้วย 0x0D]*

D:\>

2.1.8) Enter (0x0D) : คำสั่งนี้จะใช้สำหรับเช็คความพร้อมในการรับคำสั่งของ Monitor หรือเช็คสถานะของ Flash Drive ว่าถูกต่อพร้อมใช้งานหรือไม่ ถ้าพร้อมก็จะส่ง Prompt ('D:\>' หรือ '>')ออกมา ถ้าไม่พร้อมใช้งาน ก็จะส่ง 'No Disk' หรือ 'ND' ออกมาให้ และยังใช้เมื่อสิ้นสุดการส่งคำสั่ง หรือจบ Response นั้นๆ

Ex. ส่งคำสั่ง Enter ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

↵ *[ส่งคำสั่ง Enter]*

D:\> *[แสดงสถานะ Flash Drive ถูกต่อ]*

2.2) คำสั่งในกลุ่มที่เกี่ยวกับ Disk

คำสั่งในกลุ่มนี้จะใช้สำหรับจัดการเกี่ยวกับไฟล์ ซึ่งจะรองรับระบบไฟล์ทั้ง FAT12 , FAT16 และ FAT32 สำหรับใน Firmware นี้เมื่อเราจะทำการอ่านเขียนไฟล์อันดับแรกต้องทำการ Open File ก่อนด้วยคำสั่ง OPW หรือ OPR จากนั้นก็สามารถ เขียน-อ่าน ไฟล์โดยใช้คำสั่ง WRF หรือ RDF และเมื่อเลิกใช้งานก็จะต้องปิดไฟล์นั้นด้วยคำสั่ง CLF

ในการเปิดไฟล์นั้นสามารถเปิดขึ้นมาได้ครั้งละ 1 ไฟล์ , ถ้าเปิดไฟล์ขึ้นมาเพื่อเขียน มันสามารถจะเขียนข้อมูลลงไปหรืออ่านข้อมูลออกมาได้ แต่ถ้าเปิดไฟล์สำหรับอ่านมันจะอ่านข้อมูลได้เพียงอย่างเดียวเท่านั้น การอ่านเขียนนั้นจะกระทำกับไฟล์ที่ถูกเปิดอยู่ในขณะนั้นเท่านั้น, คำสั่ง SEK สามารถใช้ชี้ตำแหน่งของข้อมูลภายในไฟล์ได้ อย่างไรก็ตามไฟล์ที่เปิดสำหรับเขียน หลังจากเขียนเสร็จจะต้องทำการปิดไฟล์เสมอ มิฉะนั้นข้อมูลที่เขียนลงไปอาจสูญหาย หรือทำให้ไฟล์นั้นเสียหายทั้งไฟล์ได้

ตารางที่ 2.2 Disk Command

Extended Command Set (ASSCII Code)	Short Command Set (Hexadecimal Code)	Function
DIR ↵	01 0D	ใช้สำหรับดูรายชื่อไฟล์ใน Directory
DIR␣ *file*↵	01 20 *file* 0D	เรียกดูไฟล์ตามชื่อที่ระบุ และแสดงขนาดไฟล์
CD␣ *file* ↵	02 20 *file* 0D	ใช้สำหรับเปลี่ยนไปยัง Directory ตามชื่อที่ระบุ
CD␣.. ↵	02 20 2E 0D	ใช้สำหรับออกจาก Directory ที่เข้าไป 1 ชั้น
RD␣ *file* ↵	04 20 *file* 0D	ใช้สำหรับอ่านชื่อไฟล์ที่กำหนด
DLD␣ *file*↵	05 20 *file* 0D	ใช้สำหรับลบ Directory ย่อย จาก Directory ปัจจุบัน
MKD␣ *file*↵	06 20 *file* 0D	สร้าง Directory ย่อย ใหม่ ใน directory ปัจจุบัน
MKD␣ *file*␣ *datetime* ↵	06 20 *file* 20 *datetime* 0D	สร้าง Directory ย่อย ใหม่ ใน directory ปัจจุบัน พร้อมทั้งระบุ วัน-เวลา ให้กับ directory ที่สร้างได้
DLF␣ *file* ↵	07 20 *file* 0D	ใช้ลบ File ตามชื่อที่กำหนด
WRF␣ *dword* ↵ *data*	08 20 *dword* 0D *data*	ใช้สำหรับเขียนข้อมูลลงในไฟล์ที่เปิดอยู่ โดยต้องระบุจำนวน Byte ที่จะเขียนด้วย
OPW␣ *file* ↵	09 20 *file* 0D	ใช้สำหรับเปิดไฟล์ที่จะเขียน หรือ สร้างไฟล์ใหม่
OPW␣ *file*␣ *datetime*↵	09 20 *file* 20 *datetime* 0D	ใช้สำหรับเปิดไฟล์ที่จะเขียน หรือ สร้างไฟล์ใหม่ และระบุวัน-เวลา ของไฟล์ที่เขียนได้
CLF␣ *file*↵	0A 20 *file* 0D	ใช้สำหรับปิด file
RDF␣ *dword* ↵	0B 20 *dword* 0D	อ่านข้อมูลของไฟล์ที่เปิดอยู่ ตามจำนวน Byte ที่กำหนด
OPR␣ *file*↵	0E 20 *file* 0D	ใช้เปิด file เพื่ออ่านเท่านั้น
OPR␣ *file*␣ *date*↵	0E 20 *file* 20 *date* 0D	ใช้เปิด file เพื่ออ่าน โดยระบุวันที่เข้ามาอ่านไฟล์ได้
SEK␣ *dword* ↵	28 20 *dword* 0D	เข้าไปยังตำแหน่ง Byte ที่กำหนด ของ file ที่เปิดอยู่
REN␣ *file*␣ *file* ↵	0C 20 *file* 20 *file* 0D	เปลี่ยนชื่อ file หรือ directory
FS ↵	12 0D	ใช้เรียกดูพื้นที่ว่างที่ใช้งานได้บน disk แสดงขนาดของพื้นที่ว่างบน disk ได้ไม่เกิน 4 GB.
FSE ↵	93 0D	ใช้เรียกดูพื้นที่ว่างที่ใช้งานได้บน disk และแสดงขนาดของพื้นที่ว่างบน Disk ได้มากกว่า 4 GB
IDD ↵	0F 0D	แสดงข้อมูลเกี่ยวกับ disk ที่ใช้ต่ออยู่ โดยยอมให้ disk ที่นำมาต่อต้องมีความจุน้อยกว่า 4 GB
IDDE ↵	94 0D	แสดงข้อมูลเกี่ยวกับ disk ที่ใช้ต่ออยู่ โดยยอมให้ disk ที่นำมาต่อต้องมีความจุไม่เกิน 2 TB

ตารางที่ 2.2 Disk Command (ต่อ)

Extended Command Set (ASSCII Code)	Short Command Set (Hexadecimal Code)	Function
DSN ↵	2D 0D	แสดง Serial Number ของ disk
DVL ↵	2E 0D	แสดงชื่อ volume ของ disk
DIRT␣ *file*↵	2F 20 *file* 0D	แสดงข้อมูลของ file ที่กำหนด ได้แก่ ชื่อและนามสกุลของ file,วันและเวลาที่สร้าง file,วันและเวลาที่ปรับปรุง file และ วันที่เข้ามาดู file

หมายเหตุ *ในการแทนค่า parameter file จะต้องแทนด้วยชื่อ file ตามด้วยจุดและนามสกุลของ file เสมอ โดยแทนด้วยตัวอักษร ASCII ส่วนในการแทนค่าด้วย parameter ที่ต้องใส่จำนวน byte ก็ให้แทนตามวิธีการของคำสั่ง IPA หรือ IPH ที่ผู้ใช้กำหนดไว้ ,ส่วนการแทนค่าด้วยวันและเวลานั้นให้ดูการจัดเรียง Bit ของวันและเวลาในตารางที่ 2.3 เสียก่อน แล้วแทนค่าลงไปในแต่ละบิต จากนั้นแปลงเป็น Hex เอามาแทนค่าลงในคำสั่งอีกครั้งหนึ่ง*

ตารางที่ 2.3 แสดง Bit field ของ Date และ Time

32 Bit Value	16 Bit Value	ประเภท	ค่าที่กำหนดได้	ความหมาย
bit 31:25	bit 15:9	Year	0-127 หรือ 0x00 - 0x7F	0 = ค.ศ.1980 ; 127 = ค.ศ. 2107
bit 24:21	bit 8:5	Months	1-12 หรือ 0x01 - 0x0C	1 = มกราคม ; 12 = ธันวาคม
bit 20:16	bit 4:0	Days	1-31 หรือ 0x01 - 0x1F	1 = วันแรกของเดือน
bit 15:11	N/A	Hours	0-23 หรือ 0x00 - 0x17	24 ชั่วโมง
bit 10:5	N/A	Minutes	0-59 หรือ 0x00 - 0x3B	60 นาที
bit 4:0	N/A	Seconds/2	0-29 หรือ 0x00 - 0x1D	0 = 0 วินาที ; 29 = 58 วินาที

ในตารางที่ 2.3 นี้จะเป็นการจัดเรียงบิตในส่วนของวันและเวลา เมื่อค่าวันและเวลาที่ส่งหรืออ่านออกมาจาก Monitor จะอยู่ในรูปของ เลขฐาน16 ดังนั้นผู้ใช้จะไม่สามารถอ่านหรือส่งค่าได้โดยตรง จะต้องนำค่าที่อ่านหรือที่จะส่งมาเทียบในแต่ละบิต กับตารางที่ 2.3 ก่อน เพื่อจะได้ตีความค่าที่อ่านหรือส่ง แล้วจึงส่งค่า Hex code ให้กับ Monitor ได้อย่างถูกต้อง

ตัวอย่าง สมมุติต้องการส่งค่าวันและเวลาแบบ 32 บิต โดยกำหนดวันและเวลาคือ 2007-06-07 14:24:51 (ปีค.ศ. 2007- เดือน มิถุนายน-วันที่7-เวลา14 น.-24นาที-51วินาที) เราก็จะต้องแปลงค่านี้ให้อยู่ในรูปของ Hex code 32 bit โดยเทียบกับตารางที่2.3 จะได้คือ

- Bit 31:25 (7bit) = 2007-1980 = 27 = 0x1B = 001 1011 ----> (ปี2007)
- Bit 24:21 (4bit) = 6 = 0x06 = 0110 ----> (เดือน มิถุนายน)
- Bit 20:16 (5bit) = 7 = 0x07 = 0 0111 ----> (วันที่ 7)
- Bit 15:11 (5bit) = 14 = 0x0E = 0 1110 ----> (เวลา 14 น.)
- Bit 10:5 (6 bit) = 24 = 0x18 = 01 1000 ----> (เวลา 24 นาที)
- Bit 4:0 (5bit) = 51/2 = 25.5~25 = 0x19 = 1 1001 ----> (เวลา 51 วินาที)

เมื่อแทนค่าลงไปครบทั้ง 32 bit แล้ว จากนั้นให้นำค่า binary ที่ได้มาเรียงกันโดยเริ่มจากบิต MSB ก่อน จะได้ดังนี้

0011 0110 1100 0111 0111 0011 0001 1001 = 0x36C77319

จากนั้นเราก็จะนำค่าที่แปลงได้คือ 0x36C77319 ส่งไปให้ Monitor โดยให้ส่ง Byte MSB ออกไปเป็น Byte แรก ส่วนการอ่านก็เช่นกัน Monitor ก็จะส่งค่าวันและเวลาออกมาให้ในรูปของ Hex Code โดยจะส่ง Byte LSB ออกมาเป็น Byte แรก ซึ่งผู้ใช้ก็จะต้องทำการเรียงและแยกค่าที่ส่งออกมา เพื่อหาวันเวลาที่แท้จริงจากการเทียบกับตารางที่2.3 อีกครั้งหนึ่ง

2.2.1) Directory (DIR , DIR** file **) :

- คำสั่ง DIR นี้จะใช้สำหรับเรียกดูรายชื่อไฟล์ใน Directory ปัจจุบันโดยไม่ต้องมีการผ่านค่าพารามิเตอร์ ให้กับ Monitor ลักษณะของ filename ที่ส่งออกมาจาก Monitor จะอยู่ในรูปแบบ 8.3 สำหรับ filename ที่ยาวเกิน 8.3 มันก็จะถูกตัดให้เหลือแค่ 8.3 เท่านั้น

Error Code ของคำสั่ง ได้แก่ :

- Command Failed – เกิดจากไม่พบชื่อไฟล์ที่ระบุใน Directory นั้น

Ex. ส่งคำสั่ง DIR ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[Prompt Extended mode]*

DIR ↵ *[ส่งคำสั่ง DIR]*

↵ *[Respond 0x0D]*

FILE1.TXT ↵ *[แสดงชื่อไฟล์และตามด้วย 0x0D]*

ABC DIR ↵ *[แสดงชื่อ directory และตามด้วย 0x0D]*

COPYOF~1.TXT ↵ *[แสดงชื่อไฟล์ที่มีความยาวเกิน 8 .3 และตามด้วย 0x0D]*

NEWFOL~1 DIR ↵ *[แสดงชื่อ Directory ที่มีความยาวเกิน 8 ตัวอักษรและตามด้วย 0x0D]*

D:\> *[Prompt]*

ถ้า Directory ปัจจุบันที่ผู้ใช้อยู่ เป็น Directory ย่อย เมื่อใช้คำสั่ง DIR ใน list file ก็จะแสดง '.' และ '..'อยู่หน้า Directory เพื่อบอกลำดับของ Directory ปัจจุบัน และ Directory ที่สูงกว่า

Ex. ส่งคำสั่ง DIR ด้วย Keyboard ผ่านโปรแกรม HyperTerminal สมมุติอยู่ใน Director ย่อย

D:\> *[Prompt Extended mode]*

DIR ↵ *[ส่งคำสั่ง DIR]*

↵ *[Respond 0x0D]*

. DIR↵ *[แสดงลำดับ Directory ชั้นแรก]*

.. DIR↵ *[แสดงลำดับ Directory ชั้นที่2]*

FILE1.TXT ↵ *[แสดงชื่อไฟล์ใน Director ย่อยและจบด้วย 0x0D]*

D:\> *[Prompt]*

Ex. ส่งคำสั่ง DIR ด้วย MCU (Extended Mode)

```
printf("dir \r")          ;
```

Ex. ส่งคำสั่ง DIR ด้วย MCU (Short Mode)

```
char  cmm = 0x01      ;
printf(" %c\r",cmm)   ;
```

- คำสั่ง DIR *file* คำสั่งนี้จะใช้สำหรับดูรายชื่อ ไฟล์ หรือ directory ตามชื่อที่ผู้ใช้ระบุในพารามิเตอร์ *file* ว่ามีอยู่ใน Directory ปัจจุบันหรือไม่ ถ้ามี Monitor ก็จะแสดงชื่อไฟล์พร้อมกับขนาดของไฟล์ที่ระบุออกมา โดยค่าพารามิเตอร์ที่เป็นชื่อไฟล์จะต้องระบุนามสกุลด้วย แต่ถ้าเป็นชื่อ Directory ก็ให้ใส่เฉพาะชื่ออย่างเดียว ซึ่งขนาดของ file นั้นจะมีหน่วยเป็น byte โดยจะถูกส่งออกมาต่อจากชื่อไฟล์จำนวน 4 byte โดย Monitor จะส่ง byte LSB ออกมาเป็น Byte แรก

Ex. ส่งคำสั่ง DIR *file* ด้วย Keyboard ผ่านโปรแกรม HyperTerminal (เมื่อเลือก ASCII mode[IPA])

```
D:\>                                  [Prompt Extended mode]
DIR␣file1.txt ↵                       [ส่งคำสั่ง DIR และชื่อไฟล์.txt]
↵                                     [Respond 0x0D]
FILE1.TXT␣$48␣$00␣$00␣$00 ↵           [Respond ขนาดไฟล์ = 0x00000048=72 byte จบด้วย 0x0D]
D:\>                                  [Prompt]
```

Ex. ส่งคำสั่ง DIR *file* ด้วย Keyboard ผ่านโปรแกรม HyperTerminal (เมื่อเลือก ASCII mode[IPA])

```
D:\>                                  [Prompt Extended mode]
DIR␣abc ↵                             [ส่งคำสั่ง DIR และชื่อ Directory abc]
↵                                     [Respond 0x0D]
ABC␣$00␣$00␣$00␣$00 ↵                 [Respond ขนาดไฟล์ = 0 byte เมื่อเป็น Directory จบด้วย 0x0D]
D:\>                                  [Prompt]
```

Ex. ส่งคำสั่ง DIR *file* ด้วย MCU (Extended Mode)

```
printf("dir file1.txt \r")   ;
```

Ex. ส่งคำสั่ง DIR *file.xxx* ด้วย MCU (Short Mode)

```
char  cmm1 = 0x01,cmm2 = 0x20        ;
printf(" %c%cfile1.txt\r",cmm1,cmm2)  ;
```

2.2.2) Change Directory (CD** file **,CD ..) :

- CD *file* คำสั่งนี้ จะใช้สำหรับเข้าไปยัง Directory ย่อย ตามชื่อที่กำหนดในพารามิเตอร์ *file* โดยจะต้องเข้าจากชั้นนอกสุดไปทีละชั้น จะระบุชื่อ ของ Directory ชั้นในสุดแล้วกระโดดเข้าไปเลยไม่ได้
- CD **..** คำสั่งนี้จะใช้สำหรับออกจาก Directory ย่อย ครั้งละ 1 ชั้น

การใช้งานคำสั่งนี้ถ้ามีการเปิดไฟล์สำหรับเขียนอยู่จะต้องทำการปิดไฟล์นั้นเสียก่อนถึงจะเริ่มใช้งานคำสั่งนี้ได้

Error Code ของคำสั่ง ได้แก่ :

- Command Failed – เกิดจากไม่พบชื่อไฟล์ที่ระบุใน Directory นั้น
- File Open – มีไฟล์ถูกเปิดสำหรับเขียน ให้ปิดไฟล์นั้นก่อนทำคำสั่งนี้

Ex. ส่งคำสั่ง CD *file* ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

```
D:\>                          [Prompt Extended mode]
CD␣dir1 ↵                     [ส่งคำสั่ง CD เพื่อเข้าไปยัง Directory ชื่อ dir1]
D:\>                          [Prompt ]
```

Ex. ส่งคำสั่ง CD **..** ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

```
D:\>                          [Prompt Extended mode]
CD␣..  ↵                      [ส่งคำสั่ง CD .. เพื่อออกจาก Directory dir1]
D:\>                          [Prompt]
```

หลังจากส่งคำสั่ง CD ไปแล้วผู้ใช้สามารถใช้คำสั่ง DIR ตรวจสอบดูไฟล์ใน Directory ที่เข้าไปหรือออกมาได้ว่ามีการเข้าไปจริงหรือออกมาจาก Director นั้นจริงหรือไม่

Ex. ส่งคำสั่ง CD *file* ด้วย MCU (Extended Mode)

```
printf("cd dir1\r")   ; //เข้าไปยัง Directory dir1
```

Ex. ส่งคำสั่ง CD **..** ด้วย MCU (Short Mode)

```
char  cmm1 = 0x02,cmm2 = 0x20          ;
printf(" %c%c..\r",cmm1,cmm2)          ;
```

2.2.3) Read file (RD** file**) : คำสั่งนี้จะใช้สำหรับอ่านไฟล์ตามชื่อที่ได้ระบุไว้ในพารามิเตอร์ *file* ออกมาทั้งหมด โดยข้อมูลจะแสดงออกมาเป็น ASCII เหมือนกับไฟล์ต้นฉบับไม่ว่าจะกำหนดรูปแบบข้อมูลด้วยคำสั่ง IPA หรือ IPH ก็ตาม ก่อนจะใช้งานคำสั่งนี้จะต้องทำการปิดไฟล์ใดๆที่เปิดอยู่เสียก่อน ถึงจะใช้งานคำสั่งนี้ได้ การใช้งานคำสั่งนี้ไม่จำเป็นต้องใช้คำสั่งเปิดไฟล์เพื่ออ่านก่อน

Error Code ของคำสั่ง ได้แก่ :

- Command Failed – เกิดจากไม่พบชื่อไฟล์ที่ระบุใน Directory นั้น
- File Open – มีไฟล์ถูกเปิดสำหรับเขียน ให้ปิดไฟล์นั้นก่อนทำคำสั่งนี้

Ex. ส่งคำสั่ง RD *file* ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[Prompt Extended mode]*

RD␣file1.txt ↵ *[ส่งคำสั่ง RD เพื่ออ่านข้อมูลของไฟล์ file1.txt]*

↵ 1234567890abcdefghijklmnopqrD:\> *[แสดงข้อมูลทั้งหมดที่อ่านได้และ Prompt]*

Ex. ส่งคำสั่ง RD *file* ด้วย MCU (Short Mode)

```
char  cmm1 = 0x04 , cmm2 = 0x20          ;
printf(" %c%cfile1.txt\r",cmm1,cmm2)     ;
```

2.2.4) Delete Directory(DLD file) : คำสั่งนี้จะใช้สำหรับลบ Directory ตามชื่อที่ได้ระบุในพารามิเตอร์ *file* ถ้าใน Directory ที่จะลบนั้นมี file หรือ directory ย่อยอยู่ ผู้ใช้จะต้องเข้าไปทำการลบ file หรือ Directory ใน Directory ที่จะลบก่อนแล้วถึงออกมาลบ Directory นั้นได้ พูดง่ายๆก็คือ Directory ที่จะลบได้จะต้องเป็น Directory ที่ว่าง ไม่มีไฟล์ หรือ directory ย่อย ใดๆอยู่ภายในเลย

Error Code ของคำสั่ง ได้แก่ :

- Command Failed – เกิดจากไม่พบชื่อไฟล์ที่ระบุใน Directory นั้น
- Dir Not Empty – ยังมีไฟล์หรือ Directory ย่อย อยู่ใน Directory ที่จะลบ ต้องทำให้ Directory ที่จะลบว่างก่อน

Ex. ส่งคำสั่ง DLD *file* ด้วย Keyboard ผ่านโปรแกรม HyperTerminal สมมุติลบ Directory ชื่อ ABC

D:\> *[Prompt Extended mode]*

DLD␣abc ↵ *[ส่งคำสั่ง DLD เพื่อลบ Directory ชื่อ ABC]*

D:\> ↵ *[Response prompt และ 0x0D]*

Ex. ส่งคำสั่ง DLD *file* ด้วย MCU (ใน Short Mode)

```
char  cmm1 = 0x05 , cmm2 = 0x20          ;
printf(" %c%cABC\r",cmm1,cmm2)           ;
```

2.2.5) Make Directory (MKD file , MKD file datetime) : ทั้ง 2 คำสั่งนี้จะใช้สำหรับสร้าง Directory ย่อย ใน Directory ที่อยู่ปัจจุบันขึ้นมาใหม่ โดย Directory ย่อยนั้นจะถูกสร้างขึ้นตามชื่อที่ระบุใน พารามิเตอร์ *file* ส่วนในคำสั่งที่ 2 นั้นจะมีพารามิเตอร์ *datetime* เพิ่มเข้ามาอีก 1 พารามิเตอร์ ซึ่งผู้ใช้สามารถระบุวันและเวลาให้กับ directory ย่อยที่สร้างขึ้นได้ โดยให้ดูการแทนค่าจากตารางที่ 2.3 ในช่อง “32 bit Value” เมื่อ Directory นั้นถูกสร้างขึ้น ค่าวันและเวลาที่สร้าง,ค่าวันและเวลาที่มีการปรับปรุง และ ค่าวันที่เข้ามายัง Directory นั้น ก็จะถูกกำหนดตามค่า พารามิเตอร์ *datetime* ที่ส่งไปให้ Monitor จากตารางที่ 2.3 ถ้าบิตที่ 23:16 มีค่าเป็น 0 จะทำให้การกำหนดค่าของเดือนและวันที่ผิดพลาด ดังนั้นค่าวันและเวลาของ Directory ที่สร้างก็จะถูกกำหนดให้ใช้ค่า default ซึ่งจะมีค่าคือ 0x31940000 (2004-12-04-12:00:00)

Error Code ที่อาจเกิดขึ้นกับคำสั่งนี้ ได้แก่ :

- Command Failed – เกิดจากชื่อที่ระบุไม่มีอยู่ใน Directory นั้นแล้ว
- Disk Full – เกิดจากพื้นที่ว่างบน disk เหลือไม่พอเขียน

Ex. ส่งคำสั่ง MKD *file* ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[Prompt Extended mode]*

MKD␣dir1 ↵ *[ส่งคำสั่ง MKD เพื่อสร้าง Directory ชื่อ dir1]*

D:\> ↵ *[Response Prompt และ 0x0D]*

Ex. ส่งคำสั่ง MKD *file datetime* ด้วย Keyboard ผ่านโปรแกรม HyperTerminal กำหนดวันดังนี้ 2007-06-07 เวลา 14:24:51 แปลงเป็น hex จะได้ 0x36C77319

D:\> *[Prompt Extended mode]*

IPA ↵ *[ส่งคำสั่ง IPA Set ASCII Mode]*

MKD␣dir2␣0x36C77319 ↵ *[ส่งคำสั่ง MKD เพื่อสร้าง Directory ชื่อ dir2 และกำหนดวันเวลาที่สร้าง]*

D:\> ↵ *[Response Prompt และ 0x0D]*

หลังจากส่งคำสั่ง MKD ไปแล้วสามารถตรวจสอบ directory ที่สร้างโดยใช้คำสั่ง DIR และคำสั่ง DIRT เพื่อดูวันเวลาของ Directory ที่สร้างขึ้น

Ex. ส่งคำสั่ง MKD *file datetime* ด้วย MCU (Extended Mode) ส่ง data แบบ ASCII Mode (IPA)

```
printf("MKD dir2 0x36C77319 \r")          ;
```

Ex. ส่งคำสั่ง MKD *file* ด้วย MCU (Short Mode)

```
char  cmm1 = 0x06 , cmm2 = 0x20           ;
printf(" %c%cdir2\r",cmm1,cmm2)           ;
```

2.2.6) Delete File (DLF file) : คำสั่งนี้จะทำหน้าที่ในการลบไฟล์ตามชื่อที่ได้ระบุในพารามิเตอร์ *file* โดยไฟล์ที่จะลบได้นั้นจะต้องไม่ถูกเปิดเพื่อเขียนอยู่ ถึงจะใช้คำสั่งลบไฟล์นั้นได้

Error Code ที่อาจเกิดขึ้นกับคำสั่งนี้ ได้แก่ :

- Command Fail – เกิดจากใน Directory ไม่มีไฟล์ตามชื่อที่ระบุมา
- Read Only – เกิดจากมีการ Set Attribute ของไฟล์ที่ระบุให้อ่านอย่างเดียว
- File Open – เกิดจากมีการเปิด file สำหรับเขียนอยู่ จะต้องปิดไฟล์นั้นก่อนที่จะใช้คำสั่งนี้

Ex. ส่งคำสั่ง DLF *file* ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[Prompt ใน Extended mode]*

DLF␣ file2.txt ↵ *[ส่งคำสั่ง DLF เพื่อลบไฟล์ชื่อ file2.txt]*

D:\> ↵ *[Response Prompt และ 0x0D]*

หลังจากทำคำสั่งเรียบร้อยสามารถใช้คำสั่ง DIR ดูว่าไฟล์ถูกลบหรือไม่

Ex. ส่งคำสั่ง DLF *file* ด้วย MCU (Extended Mode)

```
printf("DLF file2.txt\r")          ;
```

Ex. ส่งคำสั่ง DLF *file* ด้วย MCU (Short Mode)

```
char  cmm1 = 0x07 , cmm2 = 0x20              ;
printf(" %c%cfile2.txt\r",cmm1,cmm2)    ;
```

2.2.7) Write To File (WRF dword) : คำสั่งนี้จะทำหน้าที่ในการเขียนข้อมูลตามจำนวน Byte ที่ระบุในพารามิเตอร์ *dword*(32bit) ไปยังไฟล์ที่ถูกเปิดอยู่ในปัจจุบัน ถ้าไม่มีการเปิดไฟล์ที่จะเขียนรอไว้ จะมี Error ฟ้องว่า File Open ดังนั้นทุกครั้งที่จะทำการเขียนไฟล์ จะต้องทำการเปิดไฟล์ที่จะเขียนก่อนด้วยคำสั่ง เปิดไฟล์เพื่อเขียน(OPW) แล้วถึงจะใช้งานคำสั่งเขียนไฟล์ได้ ในขณะที่เขียนข้อมูลลงไปในไฟล์ตามจำนวน Byte ที่กำหนด ในช่วงนี้จะไม่มีคำสั่งใดๆมาหยุดการเขียนได้ ข้อมูลที่รับเข้ามาทาง Monitor นั้นจะถูกเขียนเข้าไปยังไฟล์โดยตรง จะไม่มีการ Convert จาก ASCII ไปเป็น Binary ถึงจะทำงานอยู่ในโหมด IPA ก็ตาม

ในส่วนของข้อมูลที่เขียนเข้าไปเก็บนั้น เมื่อเขียนข้อมูลจนครบจำนวน Byte ที่กำหนดในพารามิเตอร์ *dword* แล้ว ผู้ใช้ไม่ต้องส่ง Enter เพื่อสิ้นสุดการเขียนแต่อย่างใด ตัว Monitor จะทราบเอง โดยอัตโนมัติ และจะส่งเครื่องหมาย prompt ออกมาให้ผู้ใช้เมื่อข้อมูลที่เขียนนั้นครบตามจำนวน Byte ที่กำหนดแล้ว

หลังจากเขียนไฟล์และปิดไฟล์เรียบร้อยแล้ว ถ้าผู้ใช้ทำการเปิดไฟล์เดิมขึ้นมาเขียนอีกครั้งหนึ่ง ข้อมูลใหม่ที่เขียนเข้าไปจะถูกเขียนต่อจากตำแหน่งสุดท้ายของข้อมูลเดิมโดยอัตโนมัติ ซึ่งจะไม่ไปทับข้อมูลเก่าให้เสียหาย

ในขณะที่เขียนข้อมูลครบตามจำนวน byte ที่กำหนดแล้ว ถ้ายังไม่มีการปิดไฟล์ ผู้ใช้ก็ยังจะใช้คำสั่ง WRF เขียนข้อมูลตามจำนวน byte ที่กำหนดได้เรื่อยๆ โดยข้อมูลก็จะถูกเขียนต่อจากตำแหน่งสุดท้ายของข้อมูลเก่า

Error Code ของคำสั่ง ได้แก่ :

- Disk Full – พื้นที่ Disk ไม่เพียงพอที่จะทำการเขียนไฟล์ให้สมบูรณ์ได้ จะรายงานหลังจากทำคำสั่งนี้แล้ว
- File Invalid – ไม่มีการเปิดไฟล์สำหรับเขียนไว้ ให้ใช้คำสั่ง OPW เปิดไฟล์ก่อนแล้วถึงจะใช้คำสั่ง WRF

ขั้นตอนการเขียนไฟล์

1. ส่งคำสั่งกำหนดรูปแบบคำสั่งที่จะใช้ แบบ Extended Mode(ECS) หรือ Short Mode(SCS)
2. ส่งคำสั่งกำหนดรูปแบบการผ่านค่าพารามิเตอร์ที่เป็นจำนวนตัวเลข แบบ ASCII (IPA) หรือ Binary(IPH)
3. ส่งคำสั่งทำการเปิดไฟล์ที่ต้องการจะเขียน โดยใช้คำสั่ง OPW *file* หรือ *OPW file datetime*
4. ส่งคำสั่ง WRF เพื่อทำการเขียนไฟล์ โดยระบุจำนวน Byte ที่จะเขียน แล้วจบด้วย 0x0D หรือ enter
5. เริ่มทำการเขียนข้อมูลลงไปตามจำนวน Byte ที่ได้ระบุไว้ในพารามิเตอร์ *dword* โดย 1 ตัวอักษรจะเท่ากับ 1 Byte ในส่วนของข้อมูล Space(0x20) หรือ Enter(0x0D) นั้นจะถูกมองเป็นอักขระ 1 ตัว หรือ 1 Byte เช่นกัน
6. เมื่อข้อมูลถูกเขียนลงไปครบแล้ว Monitor ก็จะส่ง Prompt ออกมาให้ผู้ใช้
7. เมื่อ <prompt> ปรากฏถ้าไม่มีการเขียนต่ออีก จะต้องปิดไฟล์ โดยใช้คำสั่ง CLF *file* เพื่อจบการเขียนไฟล์
8. ถ้าต้องการกลับมาเขียนไฟล์เดิมใหม่ หรือเปลี่ยนไปเขียนในไฟล์อื่นก็ให้กลับไปเริ่มตั้งแต่ขั้นตอนที่ 3 ใหม่

Ex. ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[Prompt ใน Extended mode]*

IPA ↵ *[กำหนดรูปแบบการส่งจำนวน byte ที่จะเขียน ในแบบ ASCII Mode]*

D:\> *[Response Prompt]*

OPW␣f2.txt ↵ *[ทำการ Open file ชื่อ f2.txt]*

D:\> *[Response Prompt]*

WRF␣10 ↵ *[ส่งคำสั่งเขียน file โดยระบุจำนวน byte ที่จะเขียน 10 Byte]*

abcdefghij *[เขียน data 10 Byte]*

D:\> *[Response Prompt จะแสดงอัตโนมัติเมื่อเขียนข้อมูลตัวที่ 10 เรียบร้อย]*

CLF␣f2.txt ↵ *[ส่งคำสั่งปิด file f2.txt ที่ได้เปิดเขียนไว้]*

D:\> *[Response Prompt สิ้นสุดการเขียน file]*

เมื่อปิดไฟล์ที่เขียนเรียบร้อยแล้วลองใช้คำสั่ง RD เพื่ออ่านข้อมูลที่เขียนดูได้

Ex. ส่งคำสั่ง ด้วย MCU (ใน Extended Mode)

```
printf("IPA\r")          ;  //กำหนดรูปแบบการส่งจำนวน byte ที่จะเขียนในแบบ ASCII Mode
echo() ;                  // Check Response 'D:\>'
printf("OPW f2.txt\r") ; // ส่งคำสั่ง OPW เพื่อเปิดไฟล์ชื่อ f2.txt สำหรับเขียน
echo() ;                  //Check Response 'D:\>'
printf("WRF 10\r")       ; //ส่งคำสั่ง WRF เขียนไฟล์โดยระบุจำนวนไบท์ ที่จะเขียน 10 ไบท์
printf(" abcdfghrp")     ; //ส่งข้อมูลที่จะเขียนลง Flash Driveจำนวน 10 byte
echo() ;
printf("CLF f2.txt\r")   ; //หลังจากเขียนข้อมูลครบแล้ว ส่งคำสั่ง CLF เพื่อปิดไฟล์ f2.txt
```

ในส่วนของฟังก์ชั่น echo() จะมีไว้สำหรับตรวจสอบ Response 'D:\>' ก่อนที่จะส่งคำสั่งต่อไป เพื่อให้ การส่งคำสั่งนั้นไม่ผิดพลาด ซึ่งผู้ใช้อาจจะเขียนโปรแกรมในลักษณะวนรอรับข้อมูลที่เข้ามา และนำข้อมูลที่เข้ามาแต่ละครั้งไปตรวจสอบว่าใช่ตัวอักษร '>' หรือไม่ โดยตรวจสอบเพียงตัวเดียวก็ได้ หรือถ้าจะให้แน่ใจจริงๆก็ควรตรวจสอบทุกตัว

Ex. ส่งคำสั่ง ด้วย MCU (ใน Short Mode)

```
char  cmm1 , cmm , n =0x00, n2=0x0A  ;
cmm1 = 0x91                          ;  // Set Command IPH
printf("%c\r",cmm1)          ; //กำหนดรูปแบบการส่งจำนวน byte ที่จะเขียน ในแบบ Binary Mode (IPH)
echo()                               ; // Check Response '>'
cmm1 = 0x09 ; cmm2 = 0x20            ; // Set Command OPW
printf("%c%cf2.txt\r",cmm1,cmm2)     ; // ส่งคำสั่ง OPW เพื่อเปิดไฟล์ชื่อ f2.txt
echo() ;                               //Check Response '>'
```

```
cmm1 = 0x08 ; cmm2 = 0x20              ; // Set Command WRF
printf("%c%c%c%c%c%c\r",cmm1,cmm2,n,n,n,n2)   ; //ส่งคำสั่ง WRF เขียนไฟล์โดยระบุจำนวนไบท์ที่จะ-
                                                //-เขียน 10 ไบท์(0x0000000A)
printf("abcdefghij")                   ; //เริ่มส่งข้อมูลที่จะเขียนลง Flash Driveจำนวน 10 byte
echo() ;
cmm1=0x0A ; cmm2=0x20                  ; // Set Command CLF
printf("%c%cf2.txt\r",cmm1,cmm2)       ; //หลังจากเขียนข้อมูลครบแล้ว ส่งคำสั่ง CLF เพื่อปิดไฟล์ f2.txt
```

หมายเหตุ *เมื่อเลือกรูปแบบการส่งจำนวน Byte แบบ Binary(IPH) ผู้ใช้จะต้องส่งค่าตัวเลขในรูปฐาน16ให้ครบตามจำนวน Byte ที่กำหนดในคำสั่งนั้นๆ จากตัวอย่างข้างต้น ในคำสั่ง WRF จะกำหนดค่าพารามิเตอร์ที่ต้องส่งผ่านคือ 4 Byte(32bit) ดังนั้นเมื่อเราต้องการเขียนข้อมูล 10 byte ตัวเลขที่จะต้องส่งก็คือ 0x0000000A แต่ถ้าส่งในโหมด ASCII(IPA) ค่าที่จะส่งคือ '10' หรือ '0x0A' เท่านั้นไม่ต้องคำนึงเรื่อง Byte ที่คำสั่งนั้นๆระบุไว้*

ในการส่งคำสั่งด้วย printf() และเลือกส่งข้อมูลแบบ Short Command หรือ เลือกส่งผ่านค่าพารามิเตอร์ที่เป็นจำนวนตัวเลขแบบ Binary (IPH) ผู้ใช้จะต้องดูด้วยว่า Code ที่ส่งออกไปยัง Monitor จริงๆนั้นถูกต้องหรือไม่ เพราะ printf() ในภาษา C บางตัว ถ้าส่งค่า 0x00 ด้วย printf() ค่า 0x00 จะไม่ถูกส่งออกไป หรือถ้าส่งค่า 0x0A ค่าที่ส่งออกไปก็จะเป็น 0x0A และ 0x0D ออกไปพร้อมกัน ซึ่งก็จะทำให้ Monitor รับคำสั่งผิดพลาดได้ วิธีแก้คือต้องส่งข้อมูลออกไปยัง Buffer Uart โดยตรง อย่าใช้ฟังก์ชัน printf()

2.2.8) Open File for Write (OPW** file **, OPW** file datetime**) : คำสั่งนี้จะทำหน้าที่เปิดไฟล์ขึ้นมาเพื่อจะทำการเขียนข้อมูล หรือ สร้างไฟล์ขึ้นมาใหม่ถ้าชื่อไฟล์ที่ระบุไม่มีอยู่ใน Directory ที่ทำงานอยู่

ตำแหน่งเริ่มต้นของไฟล์ที่ถูกเปิดขึ้นมานั้นจะถูกกำหนดให้เริ่มต้นที่ตำแหน่งที่ต่อจากตำแหน่งตอนปิดไฟล์ ดังนั้นเวลาใช้คำสั่งเขียน (WRF) ข้อมูลก็จะถูกเขียนต่อจากตำแหน่งล่าสุดตอนปิดไฟล์ทำให้ข้อมูลเดิมไม่ถูกทับ และเราสามารถใช้คำสั่ง (SEK) ส่งต่อจากคำสั่งนี้ได้เพื่อเลื่อนตำแหน่งที่จะเขียน

ในส่วนของค่าพารามิเตอร์ *datetime* ที่ใช้ในคำสั่งนี้จะใช้เมื่อผู้ใช้ต้องการระบุวันและเวลาในการเขียนไฟล์ ซึ่งมีขนาด 32 bit(ให้ดูในตารางที่ 3.2 ช่อง 32 Bit Value) -*ในกรณีที่ผู้ใช้เปิดไฟล์ที่มีอยู่แล้วใน directory ขึ้นมา* ค่าวันและเวลาที่ผู้ใช้กำหนดไว้ในพารามิเตอร์ *datetime* ก็จะถูก Update ให้กับไฟล์ที่เปิดขึ้นมา ถ้าค่าที่กำหนดให้กับพารามิเตอร์ *datetime* ในบิตที่ 23:16 เป็น 0 จะทำให้ ค่าวันและเดือน ผิดพลาด และจะไม่มีการ update ค่าวันละเวลาให้กับไฟล์ที่เปิดขึ้นมา -*ในกรณีที่ ไฟล์นั้นถูกสร้างขึ้นมาใหม่* ค่าวันและเวลาที่ผู้ใช้กำหนดไว้ในพารามิเตอร์ *datetime* ก็จะถูก Update ให้กับไฟล์ที่สร้างขึ้นมา ถ้าค่าที่กำหนดให้กับพารามิเตอร์ *datetime* ในบิตที่ 23:16 เป็น 0 จะทำให้ ค่าวันและเดือน ผิดพลาด ดังนั้นค่าวันและเวลาของไฟล์ที่สร้างขึ้นมาใหม่ก็จะถูกกำหนดด้วยค่า default คือ 0x31940000 (2004-12-04 12:00:00)

หมายเหตุ *ไฟล์ที่ถูกเปิดขึ้นมาสำหรับเขียนนั้น จะต้องถูกปิด เพื่อที่จะ Update ขนาดของไฟล์และบันทึกใน Directory table ถ้าไฟล์ไม่ถูกปิดก่อนที่จะ reset หรือ ก่อนถอดอุปกรณ์เก็บข้อมูลออก ขนาดไฟล์ก็จะไม่ถูกเก็บ ข้อมูลก็จะสูญหาย Error Code ของคำสั่งได้แก่ :*

- Read Only – เกิดจากมีการ Set Attribute ของไฟล์ที่ระบุให้อ่านอย่างเดียว
- Disk Full – Disk เต็ม ไม่มีพื้นที่สำหรับสร้างไฟล์

- File Open – เกิดจาก มีการเปิด file อื่นค้างอยู่ จะต้องปิดไฟล์นั้นก่อนที่จะใช้คำสั่งนี้

Ex. การใช้คำสั่ง OPW *file* ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[Prompt ใน Extended mode]*

OPW␣f2.txt ↵ *[ส่งคำสั่ง OPW เพื่อเปิดไฟล์หรือสร้างไฟล์ใหม่ชื่อ f2.txt]*

D:\> ↵ *[Response Prompt และ 0x0D]*

Ex. การใช้คำสั่ง OPW *file datetime* ส่งคำสั่งด้วย MCU (ใน Extended Mode)

```
printf("IPA\r")                          ; // กำหนดการส่งค่า วันเวลา(พารามิเตอร์ datetime)แบบ ASCII Mode
printf("OPW f2.txt 0x36C77319\r")  ; // ส่งคำสั่ง open ไฟล์ f2.txt กำหนดค่าเวลาให้กับไฟล์คือ
                                          //2007-06-07(ปี-เดือน-วันที่) เวลา 14:24:51 (hh:mm:ss)
```

Ex. ส่งคำสั่ง OPW *file* ด้วย MCU (ใน Short Mode)

```
char  cmm1 = 0x09 , cmm2 = 0x20          ;
printf(" %c%cf2.txt\r",cmm1,cmm2)    ;
```

2.2.9) Close File (CLF file) : คำสั่งนี้จะใช้สำหรับปิดไฟล์ที่เปิดอยู่ ซึ่งชื่อไฟล์ที่ระบุในพารามิเตอร์ *file* นั้นจะต้องตรงกับชื่อไฟล์ที่ถูกเปิดอยู่ด้วย โดยชื่อไฟล์ที่กำหนดจะเป็นไปตามมาตรฐาน 8.3 (ชื่อไฟล์8ตัวอักษร.xxx) ส่วนไฟล์ที่ถูกเปิดสำหรับอ่านไม่จำเป็นต้องมีการปิดไฟล์

Error Code ของคำสั่ง ได้แก่ :

- Command Failed – ไม่พบชื่อไฟล์ที่ระบุ ใน Directory

Ex. ส่งคำสั่ง CLF *file* ด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[Prompt ใน Extended mode]*

CLF␣ f2.txt ↵ *[ส่งคำสั่ง CLF เพื่อปิดไฟล์ชื่อ f2.txt]*

D:\> ↵ *[Response Prompt และ 0x0D]*

Ex. ส่งคำสั่ง CLF *file* ด้วย MCU (ใน Extended Mode)

```
printf("CLF f2.txt\r")              ; //ส่งคำสั่ง Close ไฟล์ f2.txt
```

Ex. ส่งคำสั่ง CLF *file* ด้วย MCU (ใน Short Mode)

```
char  cmm1 = 0x0A , cmm2 = 0x20          ;
printf(" %c%cf2.txt\r",cmm1,cmm2)    ;
```

***2.2.10) Read From File (RDF** dword) :* คำสั่งนี้จะทำหน้าที่ในการอ่านข้อมูลจากไฟล์ใน Disk Drive ที่ถูกเปิดอยู่ออกมาตามจำนวน Byte ที่ระบุในพารามิเตอร์ *dword* (32bit) ทุกครั้งที่จะทำการอ่านไฟล์ด้วยคำสั่งนี้ จะต้องทำการเปิดไฟล์ที่จะอ่านก่อนด้วยคำสั่งเปิดไฟล์เพื่ออ่าน(OPR) แล้วถึงจะใช้งานคำสั่งนี้อ่านไฟล์ได้ ในขณะที่ทำการอ่านข้อมูลจาก Disk Drive ข้อมูลก็จะถูกส่งออกมาจาก Monitor port (RS232) โดยตรง(ข้อมูลจะเหมือนกับต้นฉบับ) ในช่วงนี้จะไม่มีคำสั่งใดๆมาหยุดการอ่านได้ จนกว่าจะอ่านข้อมูลออกมาครบตามจำนวน Byte ที่กำหนด

ในการอ่านข้อมูลแต่ละครั้งนั้น เมื่อข้อมูลถูกอ่านจนครบจำนวน Byte ที่กำหนดแล้ว Monitor ก็จะส่ง <prompt> ออกมาปิดท้ายเสมอ หลังจากเปิดไฟล์ขึ้นมาแล้ว ผู้ใช้สามารถใช้คำสั่ง RDF ทำการอ่านข้อมูลออกมาได้เรื่อยๆจนกว่าจะสั่งปิดไฟล์ โดยข้อมูลที่อ่านแต่ละครั้งนั้น จะถูกอ่านต่อจากตำแหน่งสุดท้ายของการอ่านครั้งก่อนหน้าไปเรื่อยๆ จะไม่ไปเริ่มต้นอ่านที่ตำแหน่งแรกใหม่ นอกเสียจากผู้ใช้ปิดไฟล์ที่อ่านอยู่ แล้วเปิดไฟล์เดิมขึ้นมาอ่านต่อข้อมูลถึงจะเริ่มถูกอ่านที่ตำแหน่งแรกใหม่ ดังนั้นถ้ายังไม่มีการปิดไฟล์ที่อ่านอยู่ การอ่านข้อมูลแต่ละครั้งจะถูกอ่านต่อจากตำแหน่งล่าสุดของการอ่านครั้งก่อนหน้าต่อไปเรื่อยๆ

ถ้าจำนวนของ Byte ข้อมูลที่จะอ่านถูกกำหนดให้มากกว่า จำนวน Byte ของข้อมูลในไฟล์ ที่เปิดอ่านแล้ว การอ่านก็จะยังคงดำเนินการต่อไปจนครบจำนวน Byte ที่กำหนด จะไม่มี error ใดๆแสดงเตือน ข้อมูลในส่วนที่อ่านเกินก็จะอ่านออกมาได้ค่าเป็น 0 หรืออาจเกิดการแฮงค์ได้ ดังนั้นผู้ใช้ก็ควรจะตรวจสอบขนาดของ Byte ข้อมูลในไฟล์ที่จะอ่านก่อนด้วยคำสั่ง DIR เพื่อจะได้ไม่กำหนดจำนวน Byte ที่จะอ่านเกินขนาดของไฟล์ที่เปิดอ่าน

ความแตกต่างระหว่างคำสั่ง RD และ RDF ซึ่ง2คำสั่งนี้จะเป็นคำสั่งอ่านไฟล์เช่นเดียวกัน โดยคำสั่ง RD นั้นจะเป็นการอ่านข้อมูลของไฟล์ออกมาทั้งหมด เวลาใช้งานไม่ต้องทำการเปิดไฟล์ก่อน สามารถใช้คำสั่งอ่านไฟล์ตามชื่อที่กำหนดในพารามิเตอร์ได้เลย แต่จะมีข้อเสียคือการอ่านไฟล์ครั้งเดียวมากๆ ก็อาจจะทำให้ข้อมูลนั้นส่งออกมาผิดพลาดได้เช่นกัน ส่วนคำสั่ง RDF นั้นจะสามารถกำหนดจำนวนของข้อมูลที่จะอ่านออกมาได้ตามจำนวน Byte ที่ต้องการ แต่ในการใช้คำสั่งนี้จะต้องทำการเปิดไฟล์ที่จะอ่านขึ้นมาก่อนด้วยคำสั่ง OPR แล้วถึงตามด้วยคำสั่งนี้ ข้อดีของคำสั่งนี้จะทำให้การส่งข้อมูลนั้นไม่ผิดพลาด เนื่องจากสามารถกำหนดจำนวน Byte ข้อมูลในการอ่านแต่ละครั้งไม่ให้มากจนเกินไปได้

<u>ขั้นตอนการอ่านไฟล์ด้วยคำสั่ง RDF</u>

1. ส่งคำสั่งกำหนดรูปแบบคำสั่งที่จะใช้ แบบ Extended Mode(ECS) หรือ Short Mode(SCS)
2. ส่งคำสั่งกำหนดรูปแบบการผ่านค่าพารามิเตอร์ที่เป็นจำนวนตัวเลข แบบ ASCII (IPA) หรือ Binary(IPH)
3. ส่งคำสั่งทำการเปิดไฟล์ที่ต้องการจะอ่าน โดยใช้คำสั่ง OPR *file* หรือ *OPR file date*
4. ส่งคำสั่ง RDF เพื่อทำการอ่านไฟล์ โดยระบุจำนวน Byte ที่จะอ่านในพารามิเตอร์ *dword* (32 bit = 4 Byte) แล้วจบด้วย 0x0D หรือ enter
5. จากนั้นข้อมูลก็จะถูกอ่านออกมาทาง RS232 จนครบจำนวน Byte ที่ได้ระบุไว้และปิดด้วย <prompt>ต่อท้าย
6. เมื่อ <prompt> ปรากฏ และจะอ่านข้อมูลในไฟล์เดิมต่ออีกก็ให้กลับไปทำซ้ำตั้งแต่ขั้นตอนที่ 4 อีกครั้ง แต่ถ้าต้องการกลับไปอ่านข้อมูลในไฟล์เดิม ในตำแน่งเริ่มต้นไฟล์ จะต้องทำการปิดไฟล์เดิมก่อนด้วยคำสั่ง CLF แล้วถึงทำการเปิดไฟล์เดิมขึ้นมาอีกครั้ง ข้อมูลก็จะถูกอ่านที่ตำแหน่งเริ่มต้นใหม่อีกครั้ง
7. ถ้าจะอ่านข้อมูลในไฟล์ใหม่ก็ให้กลับไปทำในขั้นตอนที่ 3 ใหม่ โดยไม่จำเป็นต้องปิดไฟล์เก่าที่เปิดอ่านอยู่ก็ได้

Error Code ของคำสั่งได้แก่ :

- Command Failed – เกิดจากตำแหน่งที่อ่านไฟล์อยู่ในตำแหน่ง End of file ไม่มีข้อมูลให้อ่าน

Ex. การใช้คำสั่ง RDF *dword* ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[Prompt ใน Extended mode]*

IPA ↵ *[กำหนดรูปแบบการส่งจำนวน byte ที่จะอ่าน ในแบบ ASCII Mode]*

D:\> *[Response Prompt]*

OPR ␣ f2.txt ↵ *[ส่งคำสั่ง Open file ชื่อ f2.txt]*

D:\> *[Response Prompt]*

RDF ␣ 12 ↵ *[ส่งคำสั่งอ่าน file ที่เปิดอยู่ โดยระบุจำนวน byte ที่จะอ่าน 12 Byte]*

↵ **ab1234567890**D:\> *[data ถูกอ่านออกมา 12 Byte โดยข้อมูลจะถูกนำด้วย 0x0D และปิดด้วย <prompt>]*

↵ *[ส่งคำสั่ง enter เพื่อรอรับคำสั่งต่อไป]*

D:\> *[Response <prompt>]*

Ex. การใช้คำสั่ง RDF *dword* ส่งคำสั่งด้วย MCU (Extended Mode)

```
char m , dat[13] ;
printf("IPA\r")          ;  // กำหนดรูปแบบการส่งจำนวน byte ที่จะอ่านในแบบ ASCII Mode
echo() ;                    // Check Response 'D:\>'
printf("OPR f2.txt\r")   ;  // ส่งคำสั่ง OPR เพื่อเปิดไฟล์ชื่อ f2.txt
echo() ;                    // Check Response 'D:\>'
printf("RDF 12\r")       ;  //ส่งคำสั่ง RDF เพื่ออ่านไฟล์โดยระบุจำนวนไบท์ ที่จะอ่าน 12 ไบท์
for(m=0;m<13;m++)           // Loop รับข้อมูลเข้ามาเก็บไว้ใน Buffer dat ข้อมูลใน dat[0] = 0x0D
  scanf("%c",&dat[m]) ;     // ส่วนข้อมูลที่อ่านจากไฟล์จะเริ่มถูกเก็บที่ dat[1]- dat[12] = 12 Byte
echo()                      ; // Check Response 'D:\>' ว่าพร้อมรับคำสั่งต่อไปหรือยัง
```

Ex. การใช้คำสั่ง RDF *dword* ส่งคำสั่งด้วย MCU (Short Mode)

```
char  cmm1 , cmm 2 , m , n = 0x00 , n2 = 0x0C , dat[13] ;
cmm1 = 0x91                         ;  // Set Command  IPH
printf("%c\r",cmm1)         ; //กำหนดรูปแบบการส่งจำนวน byte ที่จะเขียน ในแบบ Binary Mode (IPH)
echo()                              ;  // Check Response '>'
cmm1 = 0x0E ; cmm2 = 0x20           ; // Set Command OPR
printf("%c%cf2.txt\r",cmm1,cmm2)    ; // ส่งคำสั่ง OPR เพื่อเปิดไฟล์ชื่อ f2.txt เพื่อจะอ่าน
echo() ;                              //Check Response '>'
```

```
cmm1 = 0x0B ; cmm2 = 0x20                          ; // Set Command RDF
printf("%c%c%c%c%c%c\r",cmm1,cmm2,n,n,n,n2)   ; //ส่งคำสั่ง RDF อ่านไฟล์โดยระบุจำนวนไบท์ที่จะ-
                                                    //-อ่าน 12 byte (0x0000000C)
for(m=0;m<13;m++)        // Loop รับข้อมูลเข้ามาเก็บไว้ใน Buffer dat ข้อมูลใน dat[0] = 0x0D
   scanf("%c",&dat[m]) ; // ส่วนข้อมูลที่อ่านจากไฟล์จะเริ่มถูกเก็บที่ dat[1]- dat[12] = 12 Byte
echo()                   ; // Check Response '>' แสดงว่าอ่านข้อมูลครบตามที่กำหนดแล้ว
```

ในส่วนของฟังก์ชั่น echo() จะมีไว้สำหรับตรวจสอบ Response <prompt> ('D:\>' หรือ '>') ก่อนที่จะส่งคำสั่งต่อไป เพื่อให้ การส่งคำสั่งนั้นไม่ผิดพลาด ซึ่งผู้ใช้อาจจะเขียนโปรแกรมในลักษณะวนรอรับข้อมูลที่เข้ามา และนำข้อมูลที่เข้ามาแต่ละครั้งไปตรวจสอบว่าใช่ตัวอักษร '>' หรือไม่ โดยตรวจสอบเพียงตัวเดียวก็ได้ หรือถ้าจะให้แน่ใจจริงๆก็ควรตรวจสอบทุกตัว

<u>หมายเหตุ</u> *เมื่อเลือกรูปแบบการส่งจำนวน Byte แบบ Binary(IPH) ผู้ใช้จะต้องส่งค่าตัวเลขในรูปฐาน16ให้ครบตามจำนวน Byte ที่กำหนดในคำสั่งนั้นๆ จากตัวอย่างข้างต้น ในคำสั่ง RDF จะกำหนดค่าพารามิเตอร์ที่ต้องส่งผ่านคือ 4 Byte(32bit) ดังนั้นเมื่อเราต้องการเขียนข้อมูล 12 byte ตัวเลขที่จะต้องส่งก็คือ 0x0000000C แต่ถ้าส่งในโหมด ASCII(IPA) ค่าที่จะส่งคือ '12' หรือ '0x0C' เท่านั้นไม่ต้องคำนึงเรื่องจำนวน Byte ที่คำสั่งนั้นๆระบุไว้*

ในการส่งคำสั่งด้วย printf() และเลือกส่งข้อมูลแบบ Short Command หรือเลือกส่งผ่านค่าพารามิเตอร์ที่เป็นจำนวนตัวเลขแบบ Binary (IPH) ผู้ใช้จะต้องดูด้วยว่า Code ที่ส่งออกไปยัง Monitor จริงๆนั้นถูกต้องหรือไม่ เพราะ printf() ในภาษา C บางตัว ถ้าส่งค่า 0x00 ด้วย printf() ค่า 0x00 จะไม่ถูกส่งออกไป หรือถ้าส่งค่า 0x0A ค่าที่ส่งออกไปก็จะเป็น 0x0A และ 0x0D ออกไปพร้อมกัน ซึ่งก็จะทำให้ Monitor รับคำสั่งผิดพลาดได้ วิธีแก้คือต้องส่งข้อมูลออกไปยัง Buffer Uart โดยตรง อย่าใช้ฟังก์ชัน printf()

2.2.11) Open File for Read (OPR** file **หรือ OPR** file date**) : คำสั่งนี้จะทำหน้าที่เปิดไฟล์ที่ระบุในพารามิเตอร์ *file* สำหรับอ่าน ข้อมูลจะถูกอ่านจากไฟล์ที่เปิดนี้ด้วยคำสั่ง RDF ถ้าไฟล์ที่เปิดไม่มีอยู่ใน Directory จะไม่มีการสร้างไฟล์ใหม่ขึ้นมาเหมือนกับคำสั่ง OPW ส่วนคำสั่ง SEK สามารถส่งต่อจากคำสั่งนี้ได้เพื่อกำหนดตำแหน่งเริ่มต้นในการอ่านข้อมูล

ในส่วนของค่าพารามิเตอร์ *date* ที่ใช้ในคำสั่งนี้จะใช้เมื่อผู้ใช้ต้องการระบุวันที่เข้ามาอ่านไฟล์ ซึ่งมีขนาด 16 bit (ให้ดูในตารางที่ 3.2 ช่อง 16 Bit Value) เมื่อผู้ใช้เปิดไฟล์ขึ้นมา ค่าในพารามิเตอร์ *date* ก็จะถูก Update ให้กับไฟล์ที่เปิดขึ้นมา ถ้าค่าที่กำหนดให้กับพารามิเตอร์ *date* ในบิตที่ 7:0 เป็น 0 จะทำให้ค่าวันและเดือนผิดพลาด ซึ่งก็จะไม่มีการเปลี่ยนแปลง ค่าวันละเวลาให้กับไฟล์ที่เปิดขึ้นมา

การเปิดไฟล์สำหรับอ่านนี้ ไม่จำเป็นต้องปิดไฟล์ หลังจากอ่านเสร็จ สามารถส่งคำสั่งอื่นๆได้เลย แต่ถ้าเป็นการเปิดไฟล์สำหรับเขียน เมื่อเขียนข้อมูลเสร็จ จะต้องทำการปิดไฟเสมอ ไม่เช่นนั้นจะทำให้การส่งคำสั่งต่อไป Error ได้
Error Code ของคำสั่งได้แก่ :

- Command Failed – เกิดจากชื่อไฟล์ที่ระบุไม่มีอยู่ใน Directory ปัจจุบัน
- File Invalid – เกิดจากมีการใช้งานไฟล์ที่ระบุอยู่ใน Directoryปัจจุบัน หรือ เป็นชื่อของ directory ย่อย
- File Open – เกิดจากมีการเปิดไฟล์อื่นสำหรับเขียนอยู่ จะต้องทำการปิดไฟล์นั้นก่อนจะใช้คำสั่งนี้

Ex. การใช้คำสั่ง OPR *file* ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[Prompt ใน Extended mode]*

OPR ␣ f2.txt ↵ *[ส่งคำสั่ง OPR เพื่อเปิดไฟล์ชื่อ f2.txt สำหรับอ่าน]*

D:\> ↵ *[Response Prompt และ 0x0D]*

Ex. การใช้คำสั่ง OPR *file date* ส่งคำสั่งด้วย MCU (ใน Extended Mode)

```
printf("IPA\r")                    ; // กำหนดการส่งค่า วัน (พารามิเตอร์ date)แบบ ASCII Mode
printf("OPR  f2.txt  0x36C7\r")    ; // ส่งคำสั่ง open ไฟล์ f2.txt กำหนดค่าเวลาให้กับไฟล์คือ
                                     //2007-06-07(ปี-เดือน-วันที่)
```

Ex. ส่งคำสั่ง OPR *file* ด้วย MCU (ใน Short Mode)

```
char  cmm1  , cmm2 = 0x20          ;
cmm1 = 0x10                        ;   // Set คำสั่ง SCS
printf("%c%c",cmm1,cmm3)           ;   //ส่งคำสั่ง SCS เพื่อกำหนดใช้คำสั่งแบบ Short Mode
cmm1 = 0x0E                        ;  // Set คำสั่ง OPR
printf(" %c%cf2.txt\r",cmm1,cmm2)  ;  //ส่งคำสั่ง OPR เพื่อเปิดไฟล์ชื่อ f2.txt สำหรับอ่าน
```

2.2.12) Seek (SEK dword) : คำสั่งนี้จะใช้สำหรับเลื่อนตำแหน่งของ pointer ให้ชี้ไปยังตำแหน่งของข้อมูลที่อยู่ในไฟล์ที่ผู้ใช้ได้ทำการเปิดเพื่อเขียนหรือเพื่ออ่านไว้ ซึ่งผู้ใช้สามารถกำหนดตำแหน่งของ pointer ให้ชี้ไปยังข้อมูลในไฟล์ได้โดยผ่านพารามิเตอร์ *dword* (32 bit = 4 byte) ค่าเริ่มต้นของตำแหน่ง pointer จะเริ่มต้นจากค่า 0x00000000 ซึ่งจะเป็นการชี้ไปยังตำแหน่งข้อมูล Byte แรกของ file ที่เปิดอยู่

ถ้าจะใช้คำสั่ง SEK นี้จะต้องใช้ต่อจากคำสั่ง OPR หรือ OPW นั่นคือ จะต้องมีการเปิดไฟล์เพื่ออ่านหรือเพื่อเขียนขึ้นมาก่อน จากนั้นจึงส่งคำสั่ง SEK เพื่อระบุตำแหน่งในการอ่านหรือเขียนข้อมูลให้กับไฟล์ที่เปิดอยู่ แล้วจึงตามด้วยคำสั่งอ่านหรือเขียน ข้อมูลที่อ่านขึ้นมาหรือที่เขียนลงไปก็จะถูกเริ่มต้นที่ตำแหน่ง ที่ได้ระบุในพารามิเตอร์ *dword* ของคำสั่ง SEK ถ้าไม่ได้ใช้คำสั่ง SEK โดยปกติข้อมูลก็จะถูกอ่านจากตำแน่งเริ่มต้น(0x00000000) ของ file ถ้าเป็นการเขียน ข้อมูลก็จะถูกเขียนต่อจากตำแหน่งสุดท้ายที่เขียนค้างไว้

ในการอ่านข้อมูล หลังจากเปิดไฟล์เพื่ออ่านแล้ว (OPR) ถ้าไม่มีการปิดไฟล์ ผู้ใช้สามารถใช้คำสั่ง SEK กำหนดตำแหน่งการอ่านข้อมูล สลับกับคำสั่ง RDF เพื่ออ่านข้อมูลในตำแหน่งที่ต้องการออกมาได้ตลอด

ในการเขียนข้อมูล หลังจากเปิดไฟล์เพื่อเขียนแล้ว(OPW) ผู้ใช้สามารถใช้คำสั่ง SEK กำหนดตำแหน่งที่จะเริ่มเขียนข้อมูลได้ และเมื่อมีการส่งคำสั่งเขียน(WRF) ออกไป ข้อมูลก็จะถูกเริ่มเขียนในตำแหน่งที่กำหนด โดยข้อมูลเดิมที่อยู่หลังตำแหน่งที่กำหนด จะถูกลบทิ้งหมด ส่วนข้อมูลที่อยู่ก่อนหน้าตำแหน่งที่กำหนดจะยังคงอยู่ เมื่อเขียนข้อมูลครบตามจำนวน Byte ที่กำหนดแล้ว ยังสามารถใช้คำสั่ง WRF เขียนข้อมูลต่อได้ ถ้ายังไม่มีการปิดไฟล์โดยข้อมูลที่ถูกเขียนเข้าไปครั้งที่สองนี้ ก็จะถูกเขียนต่อจากข้อมูลในครั้งแรก จะไม่มีการไปทับข้อมูลเดิม แต่คำสั่ง SEK นั้นจะไม่สามารถใช้ได้อีกถ้ายังไม่มีการปิดไฟล์และเปิดไฟล์ขึ้นมาใหม่ก่อน

Error Code ของคำสั่งได้แก่ :

- Command Failed – เกิดจากค่า Pointer ที่กำหนดในพารามิเตอร์ *dword* ระบุเกินตำแหน่งค่า End of file

Ex. การใช้คำสั่ง SEK *dword* ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal สมมุตว่าในไฟล์ f2.txt มีข้อมูลอยู่คือ abcdefghijklmnop

D:\>	*[Prompt ใน Extended mode]*
IPA ↵	*[กำหนดการส่งค่าพารามิเตอร์แบบ ASCII Mode]*
D:/> ↵	*[Response Prompt และ 0x0D]*
OPR ␣ f2.txt ↵	*[ส่งคำสั่ง OPR เพื่อเปิดไฟล์ชื่อ f2.txt สำหรับอ่าน]*
D:\> ↵	*[Response Prompt และ 0x0D]*
SEK ␣5 ↵	*[ส่งคำสั่ง SEK กำหนดตำแหน่งเริ่มต้นการอ่านข้อมูลที่ตำแหน่ง 5 (นับจาก0)*]
D:\> ↵	*[Response Prompt และ 0x0D]*
RDF ␣4 ↵	*[ส่งคำสั่ง RDF อ่านไฟล์ ออกมา 4 Byte]*
↵fghiD:\>	*[ข้อมูลที่อ่านได้จะเริ่มจากตำแหน่งที่ 5 ก็คือ ตัว 'f ']*
↵	*[ส่ง Enter เช็คความพร้อมในการรับคำสั่งต่อไป]*
D:\> ↵	*[Response Prompt และ 0x0D]*
OPW ␣f2.txt ↵	*[ส่งคำสั่ง OPW เพื่อเปิดไฟล์ f2.txt สำหรับเขียน]*
D:\> ↵	*[Response Prompt และ 0x0D]*
SEK ␣8 ↵	*[ส่งคำสั่ง SEK กำหนดตำแหน่งเริ่มต้นการเขียนข้อมูลที่ตำแหน่ง 8*]
D:\> ↵	*[Response Prompt และ 0x0D]*
WRF␣3 ↵	*[ส่งคำสั่งเขียนไฟล์จำนวน 3 Byte]*
123	*[ส่งข้อมูล จำนวน 3 byte]*
D:\> ↵	*[Response Prompt ข้อมูลถูกเขียนเรียบร้อย]*
RD ␣f2.txt↵	*[ส่งคำสั่ง RD อ่านข้อมูลทั้งหมดจากไฟล์ f2.txt]*
↵abcdefgh123D:\>	*[ข้อมูลที่อ่านได้หลังจากเขียนข้อมูลเข้าไปใหม่]*

จะเห็นว่า ข้อมูลที่เขียนเข้าไปใหม่นั้นจะเริ่มถูกเขียนในตำแหน่งที่กำหนดจากคำสั่ง SEK(ตำแหน่ง8)และข้อมูลเก่าทั้งหมดที่อยู่หลังตำแหน่งที่8 จะถูกลบไปหมดเหลือเฉพาะข้อมูลใหม่ 3 Byte ที่ถูกเขียนเข้าไปแทนที่ ส่วนข้อมูลเก่าที่อยู่ก่อนหน้าตำแหน่งที่ 8 ยังคงอยู่เหมือนเดิม

Ex. การใช้คำสั่ง SEK *dword* ส่งคำสั่งด้วย MCU (ใน Extended Mode)

```
char n , dat[20]                ;
  printf("IPA\r")               ; // กำหนดการส่งค่า พารามิเตอร์ dword แบบ ASCII Mode
  echo()                        ; // Check Response
  printf("OPR f2.txt\r")        ; // ส่งคำสั่ง เปิดไฟล์ f2.txt เพื่ออ่าน
  echo()                        ; // Check Response
  printf("SEK 5\r")             ; // ส่งคำสั่งกำหนดตำแหน่งเริ่มต้นสำหรับอ่านในตำแหน่ง Byteที่ 5
  echo()                        ;
  printf("RDF 4\r")             ; // ส่งคำสั่งอ่านข้อมูล 4 Byte
  for(n = 0 ; n < 5 ; n++)      ; // Loop รับข้อมูลไว้ใช้งาน
    scanf("%c",dat[n])          ; // เก็บข้อมูลไว้ใน Buffer dat
```

ในตัวอย่างนี้ข้อมูลที่อ่านออกมา Byte แรกคือ 0x0D ซึ่งจะเป็น Byte นำมาก่อน จากนั้นส่วนที่เป็นข้อมูลจริงก็จะถูกเริ่มอ่านจากตำแหน่ง byte ที่5ของไฟล์ f2.txt ส่งตามออกมา 4 Byte

2.2.13) Rename File (REN** file file**) : คำสั่งนี้จะใช้สำหรับเปลี่ยนชื่อไฟล์หรือชื่อ directory ย่อย โดยพารามิเตอร์ *file* ตัวแรกจะเป็นชื่อไฟล์เก่า ส่วนพารามิเตอร์ *file* ตัวที่2 จะเป็นชื่อไฟล์ใหม่ที่จะเปลี่ยน ซึ่งในการเปลี่ยนชื่อไฟล์นั้นจะต้องใส่ทั้งชื่อและนามสกุลของไฟล์เก่าและไฟล์ใหม่ด้วย โดยรูปแบบของชื่อไฟล์จะต้องเป็นแบบ 8.3 ส่วนการเปลี่ยนชื่อให้กับ Directory นั้นก็ให้ใส่เฉพาะชื่อ directory ไม่เกิน 11 ตัวอักษร ไม่ต้องใส่นามสกุล และระหว่างชื่อไฟล์เก่าและใหม่จะต้องมีการเว้น 1 ช่องว่างด้วย

Error Code ของคำสั่งได้แก่ :

- Command Failed – เกิดจากชื่อไฟล์เก่าที่จะทำการเปลี่ยน ไม่มีอยู่ใน Directory
- File Open – มีไฟล์อื่นถูกเปิดสำหรับเขียนอยู่ ต้องทำการปิดก่อนถึงจะใช้คำสั่งนี้ได้

Ex. การใช้คำสั่ง REN *file file* ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal

```
D:\>                          [Prompt ใน Extended mode]
REN␣ f2.txt ␣test.txt ↵       [ส่งคำสั่ง REN เพื่อเปลี่ยนชื่อ f2.txt ไปเป็น test.txt]
D:\> ↵                        [Response Prompt และ 0x0D]
```

Ex. การใช้คำสั่ง REN *file file* ส่งคำสั่งด้วย MCU (ใน Extended Mode)

```
printf("REN etdir testdir\r")   ; // ส่งคำสั่ง REN เปลี่ยนDirectoryชื่อ etdir ไปเป็นชื่อ testdir
```

Ex. การใช้คำสั่ง REN *file file* ด้วย MCU (ใน Short Mode)

```
char  cmm1  , cmm2 = 0x20       ;
cmm1 = 0x10                     ; // Set คำสั่ง SCS
```

```
printf("%c%c",cmm1,cmm3)          ;  //ส่งคำสั่ง SCS เพื่อกำหนดใช้คำสั่งแบบ Short Mode
cmm1 = 0x0C                       ;  // Set คำสั่ง REN
printf(" %c%cf2.txt%ctest.txt\r",cmm1,cmm2,cmm2) ;  //ส่งคำสั่ง REN เพื่อเปลี่ยนชื่อ f2.txt ไปเป็น test.txt
```

2.2.14) Free Space (FS และ FSE) : คำสั่งนี้จะใช้สำหรับตรวจสอบพื้นที่ว่างบน Disk Drive ว่าเหลือเท่าไหร่ โดย Monitor จะ Return ค่าของพื้นที่ว่างที่เหลืออยู่ออกมาให้กับผู้ใช้ทราบ โดยจะส่งค่าออกมาในรูปของ ASCII Mode หรือ Binary Mode ก็ขึ้นอยู่กับการกำหนดของผู้ใช้เอง

ถ้าพื้นที่ว่างมีขนาดน้อยกว่า 4 Gb ก็สามารถใช้คำสั่ง FS อ่านค่าออกมาได้ แต่ถ้าพื้นที่ว่างมีขนาดมากกว่า 4 Gb เมื่อใช้คำสั่ง FS ตัว Monitor จะรายงานค่าพื้นที่ว่างออกมาเท่ากับ 0xFFFFFFFF ดังนั้น จะต้องใช้คำสั่ง FSE แทนเพื่ออ่านค่าพื้นที่ว่างที่มีมากกว่า 4 Gb

คำสั่ง FS จะ Return ค่าพื้นที่ว่างออกมาในรูปของ Hex 4 byte ส่วนคำสั่ง FSE จะ Return ค่าพื้นที่ว่างออกมาในรูปของ Hex 6 Byte โดยรูปแบบของ Hex ที่ Return ออกมานั้นจะเป็น Hex ใน ASCII Mode หรือ Hex ใน Binary Mode ก็ขึ้นอยู่กับการกำหนดจากคำสั่ง IPA และ IPH ของผู้ใช้ว่าจะเลือกรูปแบบใด โดยค่า Hex ที่ถูกส่งออกมานั้นจะส่ง Byte Low ออกมาเป็น Byte แรก

Ex. การใช้คำสั่ง FS ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal

```
D:\>                          [Prompt ใน Extended mode]
IPA ↵                         [กำหนดให้ Monitor มีการส่งค่าในแบบ ASCII Mode]
FS  ↵                         [ส่งคำสั่ง FS เพื่อเช็คพื้นที่ว่างของ Disk Drive]
↵ $00_$C0_$6B_$3B↵            [ค่าพื้นที่ว่างบนดิสก์ = 0x3B6BC000 = 996,917,248 byte ]
D:\> ↵                        [Response Prompt และ 0x0D]
```

Ex. การใช้คำสั่ง FSE ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal

```
D:\>                          [Prompt ใน Extended mode]
IPA ↵                         [กำหนดให้ Monitor มีการส่งค่าในแบบ ASCII Mode]
FS  ↵                         [ส่งคำสั่ง FS เพื่ออ่านค่าพื้นที่ว่างของ Disk Drive]
↵ $00_$C0_$6B_$3B_$00_$00 ↵   [ค่าพื้นที่ว่างบนดิสก์ = 0x00003B6BC000 = 996,917,248 byte ]
D:\> ↵                        [Response Prompt และ 0x0D]
```

Ex. การใช้คำสั่ง FS ส่งคำสั่งด้วย MCU (ใน Extended Mode)

```
char n ;
printf("IPA\r")               ; // ส่งคำสั่ง IPA กำหนดให้ Monitor ส่งค่าแบบ ASCII Mode]
printf("FS\r")                ; // ส่งคำสั่งอ่านค่าพื้นที่ว่างของ Disk Drive
for(n = 0 ; n < 5 ; n++)      ;  // Loop รับข้อมูลไว้ใช้งาน
   scanf("%c",dat[n])         ; // เก็บข้อมูลไว้ใน Buffer dat
```

ในตัวอย่างนี้ byte แรกที่รับมาจะเป็น byte นำมีค่า 0x0D(เก็บที่dat[0]) ส่วน byte ที่2 ถึงจะเริ่มเป็นข้อมูลจริงซึ่งจะเป็น Byte LSB ถูกส่งมาก่อน และถูกเก็บใน dat[1]

2.2.15) Identify Disk Drive (IDD และ IDDE) : ทั้ง 2 คำสั่งนี้จะใช้สำหรับแสดงข้อมูลเกี่ยวกับ Disk Drive ที่นำมาต่อ โดยคำสั่ง IDDE นั้นจะสามารถสนับสนุน Disk Drive ที่มีความจุถึง 2 Terra byte ด้วย

Ex. การใช้คำสั่ง IDD ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal

```
D:\>                    [Prompt ใน Extended mode]
IDD ↵                   [ส่งคำสั่ง IDD ดูข้อมูลเกี่ยวกับ Disk Drive ที่นำมาต่อ]
↵                       [Response 0x0D]
USB VID = $0951         [Response Data Information Disk Drive]
USB PID = $1603
Vendor Id = Kingston
Product Id = DataTraveler 2.0
Revision Level = 1.00
I/F = SCSI
FAT32
Bytes/Sector = $0200
Bytes/Cluster = $001000
Capacity = $3B760000 Bytes
Free Space = $3B6BC000 Bytes
↵
D:\>
```

จะสังเกตว่าก่อน จะส่งข้อมูลออกมานั้น Monitor จะส่ง 0x0D ออกมาก่อน และเมื่อสิ้นสุดข้อมูลก็จะส่ง 0x0D ออกมาปิดท้ายแล้วจบด้วย Prompt

2.2.16) Disk Volume Label (DVL) : คำสั่งนี้จะใช้สำหรับเรียกดูชื่อ Volume ของ disk โดยชื่อ Volume นี้จะมีอยู่ไม่เกิน 11 ตัวอักษร ถูกเก็บอยู่ในส่วนของ Master Boot Record (MBR) ของ disk

Ex. การใช้คำสั่ง DVL ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal

```
D:\>                    [Prompt ใน Extended mode]
DVL ↵                   [ส่งคำสั่ง DVL เพื่อเรียกดูชื่อ Volume ของ Disk]
↵ NO_NAME ↵             [Response Volume Disk ]
D:\>
```

2.2.17) Disk Serial Number (DSN) : คำสั่งนี้จะใช้สำหรับเรียกดู Serial Number ของ Disk โดยค่า Serial Number ที่ส่งออกมาจะมีขนาด 32 บิต(4 byte) โดยจะแสดงในรูปของ Hex แบบ ASCII หรือแบบ Binary นั้นก็ขึ้นอยู่กับการกำหนดของผู้ใช้จากคำสั่ง IPA หรือ IPH

Ex. การใช้คำสั่ง DSN ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[Prompt ใน Extended mode]*

IPA ↵ *[กำหนดให้ Monitor มีการส่งค่าในแบบ ASCII Mode]*

DSN ↵ *[ส่งคำสั่ง DSN เพื่อเรียกดู Serial Number ของ Disk]*

↵ $10␣$F8␣$0C␣$B0 ↵ *[Response Serial Number]*

D:\>

2.2.18) Directory File Time Command (DIRT file) : คำสั่งนี้จะใช้สำหรับดูค่าวันและเวลาของไฟล์ที่ระบุในพารามิเตอร์ *file* ซึ่งจะส่งค่าวันและเวลาออกมาจำนวน 10 Byte ในรูปของ Hex ส่วนจะเป็น Hex แบบ ASCCI Mode หรือ Binary Mode จะขึ้นอยู่กับการกำหนดของผู้ใช้ จากคำสั่ง IPA หรือ IPH

ข้อมูลที่ถูกส่งออกมา 10 Byte นี้ตัว Monitor จะส่ง Byte LSB ออกมาเป็น Byte แรก โดยประกอบด้วย Byte ต่างๆ คือ 4 Byte แรกนั้นจะเป็นค่าของ วัน และ เวลา ที่ไฟล์ถูกสร้างขึ้น ส่วน 2 Byte ต่อมาจะเป็นค่าของ วัน ที่เข้ามายังไฟล์ครั้งล่าสุด และ 4 Byte สุดท้ายจะเป็นค่าของ วัน และ เวลา ที่ไฟล์นั้นถูกแก้ไขปรับปรุง ค่าวันและเวลาที่อ่านออกมาได้สามารถนำมาถอดเป็นค่า วันและเวลาจริงๆ โดยให้อ้างอิงแต่ละ bit field กับตารางที่ 2.3

Error Code ของคำสั่งได้แก่ :

- Command Failed – เกิดจากชื่อไฟล์ที่ระบุ ไม่มีอยู่ใน Directory

Ex. การใช้คำสั่ง DIRT *file* ส่งคำสั่งด้วย Keyboard ผ่านโปรแกรม HyperTerminal

D:\> *[Prompt ใน Extended mode]*

IPA ↵ *[กำหนดให้ Monitor มีการส่งค่าในแบบ ASCII Mode]*

DIRT␣f2.txt ↵ *[ส่งคำสั่ง DIRT เพื่อเรียกดู วันและเวลา ของไฟล์ชื่อ f2.txt]*

↵ *[Response 0x0D]*

F2.TXT␣$B2␣$6B␣$F3␣$36␣$F9␣$36␣$C4␣$71␣$F9␣$36 ↵ *[Response Data 10 byte]*

D:\>

จากตัวอย่างนี้ Response ที่ส่งออกมานั้น จะนำด้วยชื่อไฟล์ จากนั้นก็ต่อด้วยค่าวันและเวลา อีก 10 Byte ซึ่งเมื่อนำมาเรียงเป็นส่วนๆจะได้ดังนี้ - ค่าวันและเวลาที่สร้างไฟล์ 4 Byte แรก = 0x36F36BB2 (32 Bit)

- ค่าวันที่เข้ามายังไฟล์ครั้งล่าสุด 2 Byte = 0x36F9 (16 Bit)
- ค่าวันและเวลาที่ปรับปรุงไฟล์ 4 Byte สุดท้าย = 0x36F971C4 (32 Bit)

เมื่อจัดเรียงข้อมูลได้แล้วก็ให้ดูที่ตาราง 2.3 เพื่อถอดค่าวันและเวลาออกมาอีกครั้งหนึ่งถึงจะได้ค่าวันและเวลาในแต่ละส่วนที่แท้จริงออกมาอย่างถูกต้อง

3. ตัวอย่างการ อ่าน-เขียน ไฟล์ด้วยไมโครคอนโทรลเลอร์

สำหรับตัวอย่างที่จะกล่าวถึงต่อไปนี้จะเป็นการใช้ MCU Z8 Encore! #Z8F6422 ในการส่งคำสั่งติดต่อการอ่าน-เขียนไฟล์ กับชุด ET-USB FLASH DRIVE โดยใช้ภาษา C ของ ZDS II ในการเขียนโปรแกรม ก่อนการใช้งานอันดับแรกต่อสายสัญญาณการสื่อสารดังในรูปที่ 4 และต่อ Flash Drive เข้ากับ ET-USB Flash Drive ให้เรียบร้อย

ตัวอย่างที่1 เขียนข้อมูลจำนวน 520 Byte เข้าไปเก็บไว้ยังไฟล์ ett.txt โดยตัวอย่างข้อความที่เขียนคือ 'ETT Welcome' มีขนาด 10 Byte ซึ่งเราจะให้ข้อความนี้เขียนในลักษณะเรียงกันในแนวตั้ง ดังนั้นจะต้องส่งค่า 0x0D และ 0x0A ต่อจากข้อความอีก 2 Byte ดังนั้นใน 1 ข้อความก็จะคิดเป็น 13 Byte เมื่อเราจะส่งข้อความจำนวน 520 Byte เราก็จะต้องเขียนข้อความนี้ออกไปทั้งสิ้น 40 ครั้งโดยให้เรียงกันในแนวตั้ง เมื่อกำหนดให้ Baud Rate = 9600 และให้ส่งคำสั่งในแบบ Extended Mode(ECS) ส่งผ่านค่าจำนวน Byte ที่จะเขียนแบบ ASCII Mode(IPA)

```
#include <stdio.h>
 #include <ez8.h>
//---------- Check Echo Command ----------
void echo()
 {
  unsigned char k;
   do{
        scanf("%c",&k) ;          // รับค่า Response มาเก็บไว้ที่ตัวแปร k เพื่อนำไปตรวจสอบ
     }while(k != '>')    ;        // ในExtended prompt = 'D:\>' แต่เราเลือกเช็คเฉพาะ '>' เพื่อความรวดเร็ว
  }
//----------------------- Main ---------------------------
 main()
  {
    unsigned char m ,n  ;
   char dat[20] = {"ETT Welcome"} ;  // Data 11 Byte
    //---------------- Initial UART0 for Z8Encore -----------------
    U0BRH   = 0x00   ;
    U0BRL   = 120    ;  //Set Baud Rate =120=9600 Kb.
    PAAF    = 0x30   ;  //Set Alternate Function PA4-PA5 for Uart 0
    U0CTL0  = 0xc0   ;  //Control Register Uart0  Enable

 //--------- Write data to Flash Drive 520 byte (Extended Mode[ECS] and ASCII Mode[IPA]) -------

    printf("\r")          ;  //Set prompt ตรวจสอบความพร้อมการรับคำสั่ง
    echo()                ;  //Check Response '>'
    printf("ECS\r")       ;  // Sent Command 'ECS' - Set Command in Extended Mode
    echo()                ;  // Check Response ' > '
```

```
printf("IPA\r")            ; // Sent Command 'IPA' - Set Number in  ASCII Mode
echo()                     ; // Check Response '>'
printf("OPW ett.txt\r")    ; // Sent Command 'OPW' - Open File 'ett.txt' เพื่อเขียน
echo()                     ; // Check Response '>'
printf("SEK 0\r")          ; // Determine Position Start=0 in file 'ett.txt'  for write data
echo()                     ;
printf("WRF 520\r")        ; // Sent Command 'WRF' - To determine write data  number 520 Byte
for(n=0;n<40;n++)          // Loop Sent Data 520 Byte
 {
   for(m=0;m<11;m++)
    printf("%c",dat[m])    ; // Start write data to file ett.txt  11 byte
    printf("\n")           ; // ***** Line feed   \n = 0x0A,0x0D = 2 byte ******
 }
echo()                     ; // Check Response '>' – Data written  fully 520 Byte
printf("CLF ett.txt\r")    ; // Sent Command 'CLF' - To Close file 'ett.txt' (END writing)
```

ในตัวอย่างนี้ จะเห็นว่าเมื่อส่งคำสั่งออกไปแต่ละครั้งนั้นเราจะตรวจสอบ Response <prompt> จาก Monitor เสมอ ด้วยฟังก์ชั่น echo() เพื่อให้การรับคำสั่งนั้นถูกต้อง ในส่วนของคำสั่ง SEK ที่ส่งต่อจากคำสั่ง OPW ก็เพื่อกำหนดตำแหน่งเริ่มต้นในการเขียนไฟล์ ในที่นี้จะกำหนดให้เริ่มต้นที่ตำแหน่ง 0 ของไฟล์ ดังนั้นทุกครั้งที่ Reset โปรแกรม ข้อมูลก็จะถูกเริ่มเขียนที่ตำแหน่งนี้เสมอ ซึ่งจะทำให้ข้อมูลเก่านั้นถูกเขียนทับสูญหายไปด้วย ดังนั้นผู้ใช้จะใช้หรือไม่ใช้คำสั่งนี้ก็ได้ ขึ้นอยู่กับวัตถุประสงค์ของผู้ใช้ ซึ่งถ้าไม่ใช้คำสั่งนี้ ข้อมูลที่เขียนเข้าไปในครั้งแรกในกรณีที่เป็นไฟล์สร้างใหม่ข้อมูลก็จะถูกเริ่มต้นเขียนที่ตำแหน่ง 0 ของไฟล์เช่นกัน แต่ถ้าเป็นไฟล์ที่มีข้อมูลอยู่แล้วข้อมูลที่เขียนเข้าไปใหม่แต่ละครั้งจะถูกนำไปเขียนต่อจากข้อมูลเดิมเสมอซึ่งก็จะทำให้ข้อมูลเก่าไม่สูญหาย เมื่อเขียนข้อมูลครบตามจำนวน Byte ที่กำหนดแล้ว Monitor ก็จะส่ง Response <prompt> ออกมาให้อัตโนมัติ ถ้าผู้ใช้ต้องการเขียนข้อมูลอีกก็ให้ส่งคำสั่ง WRF และข้อมูลที่จะเขียนส่งต่อไปได้เรื่อยๆ เมื่อจะเลิกเขียนข้อมูล และต้องการใช้งานคำสั่งอื่นๆต่อ ก็จะต้องทำการปิดไฟล์เสียก่อนด้วยคำสั่ง CLF มิฉะนั้นข้อมูลอาจสูญหายได้ถ้าไม่ทำการปิดไฟล์ก่อน และถ้าจะกลับมาเขียนข้อมูลต่อก็สามารถทำการเปิดไฟล์ขึ้นมาเขียนใหม่ได้

หลังจากเขียนไฟล์เรียบร้อยแล้วก็สามารถใช้คำสั่งอ่านไฟล์ RD หรือ RDF เพื่ออ่านไฟล์ที่เขียนออกมาดูว่าถูกต้องหรือไม่ ซึ่งดูได้ในตัวอย่างที่ 2 หรือตรวจสอบง่ายๆโดยใช้ PC เปิดดูไฟล์ที่เขียนก็ได้

ตัวอย่างที่2 อ่านข้อมูลจำนวน 130 Byte จากไฟล์ ett.txt ที่อยู่ใน Flash Drive มาเก็บไว้ใน Buffer ของ MCU เพื่อนำข้อมูลที่อ่านได้ไปใช้งานตามความต้องการ เมื่อกำหนดให้ Baud Rate = 9600 และให้ส่งคำสั่งในแบบ Extended Mode(ECS) ส่งผ่านค่าจำนวน Byte ที่จะเขียนแบบ ASCII Mode(IPA)

```
#include <stdio.h>
#include <ez8.h>
//--------- Echo Command --------
void echo()
{
 unsigned char k;
 do{
  scanf("%c",&k)       ;  // รับค่า Response มาเก็บไว้ที่ตัวแปร k เพื่อนำไปตรวจสอบ
  }while(k != '>')     ;  // ใน Extended <prompt> = 'D:\>' แต่เราเลือกเช็คเฉพาะ '>' เพื่อความรวดเร็ว
}
//-------------------------------------- Main ------------------------------------
 main()
 {
   unsigned char  m                ;
   char buf_dat[130]               ; // Buffer keep data 130 Byte
 //---------------------- Initial UART0 --------------------
  U0BRH  = 0x00                    ;
   U0BRL  = 120                    ; // Set Baud Rate =120=9600 Kb.
   PAAF   = 0x30                   ; // Set Alternate Function PA4-PA5 for Uart 0
   U0CTL0 = 0xc0                   ; // Control Register Uart0 Enable
   PBDD   = 0x00                   ; // Port B = output

//---------- Read data from File in Flash Drive 130 byte (Command-Extended Mode[ECS] ,Number-ASCII Mode[IPA])--------

   printf("\r")                    ; // Set prompt
   echo()                          ; // Check Response ' > '
   printf("ECS\r")                 ; // Sent Command 'ECS' - Set Extended Mode
   echo()                          ; // Check Response  ' >'
   printf("IPA\r")                 ;  // Sent Command 'IPA' - Set Number ASCII Mode
   echo()                          ; // Check Response ' > '
   printf("OPR ett.txt\r")         ; // Sent Command 'OPR' - Open File ett.txt เพื่ออ่าน
   echo()                          ; // Check Response ' >'
   printf("SEK 0\r")               ; // Determine Position Start =0 in file 'ett.txt'  for Read data
   echo()                          ;
   printf("RDF 130\r")             ; //Sent Command 'RDF' -To determine Read data number 130 Byte
  for(m=0;m<=130;m++)              // Loop Read data 130 Byte
     scanf("%c",&buf_dat[m])   ; //Start Receive data from  file ett.txt
  echo()                          ;
 PBOUT = ~buf_dat[1]              ; //Test Sent data in buf_dat[1]  To Port B
```

ในตัวอย่างที่ 2 นี้จะเป็นการอ่านข้อมูลที่อยู่ในไฟล์ ett.txt จำนวน 130 byte ออกมาเก็บไว้ยัง Buffer buf_dat โดยจะใช้คำสั่ง RDF ในการอ่านข้อมูลเนื่องจากสามารถกำหนดจำนวน Byte ที่จะอ่านได้ ถ้าใช้คำสั่ง RD นั้นข้อมูลจะถูกอ่านออกมาทั้งไฟล์ เมื่อส่งคำสั่งสำหรับเปิดไฟล์เพื่ออ่านแล้วเราจะส่งคำสั่ง SEK เพื่อกำหนดจุดเริ่มต้นในการอ่านข้อมูลในไฟล์ซึ่งในตัวอย่างนี้จะกำหนดจุดเริ่มต้นในการอ่านไว้ที่ตำแหน่ง 0 ของไฟล์เสมอ ถ้าผู้ใช้ไม่ใช้คำสั่งนี้ก็ได้ แต่เมื่อใช้คำสั่งนี้อ่านข้อมูลในครั้งต่อไปข้อมูลก็จะถูกอ่านต่อจากตำแหน่งล่าสุดที่ได้อ่านค้างไว้ จะไม่ไปเริ่มอ่านจากตำแหน่ง 0 ถ้ายังไม่มีการปิดไฟล์ก่อน ซึ่งเมื่ออ่านข้อมูลครบจำนวน 130 Byte แล้ว Monitor ก็จะส่ง <Prompt> ออกมาให้ หลังจากอ่านเสร็จแล้วผู้ใช้จะทำการปิดไฟล์หรือไม่ปิดก็ได้ เมื่ออ่านไฟล์มาเก็บจนครบ 130 Byte แล้ว ในตัวอย่างจะลองเอาข้อมูลในตำแหน่ง buf_dat[1] ออกมาแสดงผลที่ PortB ซึ่งค่าที่แสดงออกมาก็จะเป็น 0x45 นั่นก็คือตัวอักษร ‘E’ นั่นเองตรงกับข้อมูลที่ได้เขียนลงไปในตัวอย่างที่ 1 มีข้อสังเกตว่าในการรับข้อมูลนั้น Byte แรกที่ถูกส่งนำออกมาในการอ่านข้อมูลแต่ละครั้งก็คือค่า 0x0D จากนั้นถึงจะตามด้วยข้อมูลจริงๆ ซึ่ง Byte ที่ถูกส่งนำมานี้จะไม่ได้ถูกนำไปคิดรวมกับจำนวน Byte ที่ผู้ใช้ต้องการอ่าน จะเป็นเพียง byte นำออกมาเฉยๆ ดังนั้นในตัวอย่างจะเห็นว่า buf_dat[0] นั้นจะมีค่า 0x0D อยู่ ข้อมูลจริงจะเริ่มถูกเก็บที่ buf_dat[1] เป็นต้นไป จากในตัวอย่างที่1 นั้น ข้อมูล 1 บรรทัดเราได้เขียนไว้ 13 Byte ซึ่งจะรวม 0x0A และ 0x0D ด้วย ดังนั้นเมื่ออ่านข้อมูลมาถึง byte ที่13 ข้อมูลใน byte ที่ 14 ที่จะถูกอ่านออกมาก็คือข้อมูลตัวแรกในบรรทัดที่ 2 ซึ่งก็คือตัวอักษร ‘E’

สรุปก็คือในการเขียนข้อมูลนั้นหลังจากส่งคำสั่ง WRF ออกไปแล้ว ข้อมูลที่เขียนต่อจากคำสั่งนี้ไม่ว่าจะเป็นค่าอะไรก็ตามจะถูกนับเป็นจำนวน byte ที่ผู้ใช้ได้กำหนดไว้ในการเขียนด้วย ส่วนในการอ่านนั้น หลังจากส่งคำสั่ง RDF ออกไปข้อมูลที่อ่านได้ใน Byte แรกก็คือ 0x0D ซึ่งจะไม่คิดเป็นจำนวน Byte ในการอ่านที่ผู้ใช้ได้กำหนดไว้ แต่จะเริ่มนับจำนวน Byte ข้อมูลจริงๆใน byte ที่ 2 ซึ่งจะเป็นข้อมูลจริงๆที่ถูกส่งออกมา

<u>หมายเหตุ</u> จากทั้ง 2 ตัวอย่างนี้จะเห็นว่าในการส่งคำสั่งแบบ Extended Mode นั้น ระหว่างคำสั่ง และ พารามิเตอร์จะต้องมีการเว้นช่องว่าง 1 ช่องว่าง(0x20) และจะปิดคำสั่งด้วย \r (0x0D) สำหรับในภาษา C บางตัวนั้นในการใช้ฟังก์ชั่น printf() ส่งค่า 0x00 หรือ 0x0A นั้นจะพบปัญหาคือ ค่า 0x00 นั้นจะไม่ถูกส่งออกไป หรือ เมื่อส่งค่า 0x0A (\n) ออกไป ค่าที่ถูกส่งออกไปจาก MCU จริงๆก็จะเป็นค่า 0x0A และ 0x0D ตามกันออกไป 2 Byte ซึ่งจะทำให้ผู้ใช้กำหนด Loop นับข้อมูลในการ อ่าน-เขียน ผิดพลาดได้ หรือ ถ้าเป็นการส่งคำสั่งก็จะทำให้ Monitor รับคำสั่งผิดพลาดได้ ดังนั้นจากตัวอย่างที่ 1 จะเห็นว่าในบรรทัดที่มีเครื่องหมาย * กำกับอยู่ [printf(“\n”)] โดยปกติจะต้องส่ง printf(“\n\r”) ถึงจะเป็นการเว้นบรรทัด แต่เนื่องจาก C ที่ใช้เขียนนี้เมื่อส่ง \n ออกไปมันก็จะส่งค่า 0x0D และ 0x0A ออกไปทีเดียว 2 byte เลย ซึ่งผู้ใช้ต้องระวังด้วย ทางแก้คือใช้วิธีส่งค่าให้กับ data buffer ของ Uart โดยตรงจะดีกว่า